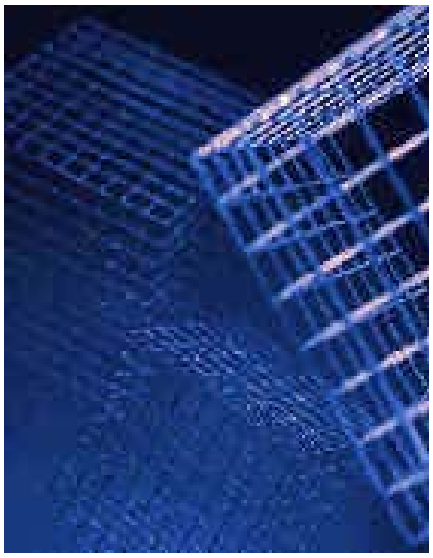
Reference Manual: SAP DB



Versions 7.2 and 7.3

SAP™

Copyright

Icons

Icon	Meaning
	Caution
	Example
	Note
	Recommendation
	Syntax

Typographic Conventions

Type Style	Description
Example text	Words or characters that appear on the screen. These include field names, screen titles, pushbuttons as well as menu names, paths and options. Cross-references to other documentation
Example text	Emphasized words or phrases in body text, titles of graphics and tables
EXAMPLE TEXT	Names of elements in the system. These include report names, program names, transaction codes, table names, and individual key words of a programming language, when surrounded by body text, for example, SELECT and INCLUDE.
`Example text`	Screen output. This includes file and directory names and their paths, messages, names of variables and parameters, source code as well as names of installation, upgrade and database tools.
Example text	Exact user entry. These are words or characters that you enter in the system exactly as they appear in the documentation.
<Example text>	Variable user entry. Pointed brackets indicate that you replace these words and characters with appropriate entries.
`EXAMPLE TEXT`	Keys on the keyboard, for example, function keys (such as `F2`) or the `ENTER` key

Reference Manual: SAP DB 7.2 and 7.3

This document outlines the syntax and semantics of the SQL statements, as used by the SAP DB database system (version 7.2 and 7.3). The syntax notation [Page 246] used in this document is BNF.

An SQL statement performs an operation on the database instance. The parameters used are host variables of a programming language in which the SQL statements are embedded.

You will find general information on the SAP DB database system in the User Manual: SAP DB [Extern].

You will find information on the Optimizer functions for SQL statements in Optimizer: SAP DB 7.3 [Extern].

Concepts

The following terms are explained here:

Data type [Page 21]

Code attribute [Page 24]

Code tables [Page 25]

SERIAL [Page 28]

Parameter [Page 28]

Table [Page 29]

Column [Page 29]

Domain [Page 30]

Index [Page 30]

Synonym [Page 30]

Users and user groups [Page 30]

Privilege [Page 31]

Role [Page 31]

Database catalog/user data [Page 31]

Transaction [Page 32]

Subtransaction [Page 32]

Database session [Page 32]

Data integrity [Page 33]

DB procedure [Page 33]

Trigger [Page 34]

SQL mode [Page 34]

See also:

User Manual: SAP DB → Definition of Terms [Extern]

Data Type

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

- NULL value [Page 22]
- Special NULL value [Page 22]
- Non-NULL value

  Character string [Page 22], LONG column [Page 23], number [Page 23], date value [Page 23], time value [Page 23], timestamp value [Page 24], BOOLEAN [Page 24]

Use

As well as specifying the column name when you define columns [Page 114], you can also specify data types.

If required, a code attribute [Page 24] can also be entered for LONG columns and some kinds of character strings.

See also:

Using data types in SQL statements: data_type [Page 114]

NULL value

The data type [Page 21] NULL value (that is, an unspecified value) is a special value. Its relationship to any other value is always unknown.

Special NULL value

A special NULL value is a special data type [Page 21] and is the result of arithmetic operations that lead to an overflow or a division by 0.

The special NULL value is only permitted for output columns and for columns in the ORDER clause [Page 190]. If an overflow occurs in an arithmetic operation or a division by 0 at another point, the SQL statement is abnormally terminated.

The comparison of a special NULL value with any value is always undefined.

As far as sorting is concerned, the special NULL value is greater than all non-NULL values, but less than the NULL value [Page 22].

Character string

A character string is a data type [Page 21] that consists of a series of alphanumeric characters.

Examples: CHAR[ACTER] [Page 115], VARCHAR [Page 116], LONG[VARCHAR] [Page 116], DATE [Page 118], TIME [Page 118], TIMESTAMP [Page 118]

In a column definition [Page 114], a code attribute [Page 24] can be entered for the data types CHAR[ACTER], VARCHAR and LONG[VARCHAR].

The following comparison options exist for data types **CHAR[ACTER]** and **VARCHAR**:

Character strings with the same code attribute	These character strings can be compared to each other.
Character strings with the code attributes ASCII [Page 25], EBCDIC [Page 26] and UNICODE [Page 24]	These character strings are comparable with the character strings of code attributes EBCDIC, ASCII, and UNICODE, and with date [Page 23], time [Page 23] and timestamp values [Page 24].

LONG column

A LONG column is a data type [Page 21] that contains a sequence of characters of any length to which no functions can be applied.

LONG columns **cannot** be compared to one another. The contents of LONG columns **cannot** be compared to character strings [Page 22] or other data types.

See also:

LONG[VARCHAR] [Page 116]

In a column definition [Page 114], a code attribute [Page 24] can be entered for the data type LONG[VARCHAR].

Number

A number is a special data type [Page 21]. There are fixed point and floating point numbers:

- Fixed point number

  A fixed point number is described by the number of significant digits and the scale. The maximum number of significant digits is 38.

  Examples: FIXED [Page 117], INT[EGER] [Page 117], SMALLINT [Page 117]

- Floating point number

  A floating point number consists of a mantissa and an exponent. The mantissa may have up to 38 significant digits. The valid range of values for floating point numbers consists of the intervals from -9. 99999999999999999999999999999999999999E +62 to -1E-64 and from +1E-64 to +9. 99999999999999999999999999999999999999E +62 and the value 0.0.

  Example: FLOAT [Page 117]

All numbers can be compared to one another.

Date value

The date value date type [Page 21] is a special character string [Page 22].

A date value can be compared to other date values and to character strings with the code attributes [Page 24] ASCII, EBCDIC, and UNICODE.

See also:

DATE [Page 118]

Date and time format [Page 53]

Time value

The time value date type [Page 21] is a special character string [Page 22].

A time value can be compared to other time values and to character strings with the code attributes [Page 24] ASCII, EBCDIC, and UNICODE.

See also:

TIME [Page 118]

Date and time format [Page 53]

Timestamp value

The timestamp value date type [Page 21] is a special character string [Page 22]. A timestamp consists of a date [Page 23] and time value [Page 23] and a microsecond specification.

A timestamp value can be compared to other timestamp values and to character strings with the code attributes [Page 24] ASCII, EBCDIC, and UNICODE.

See also:

TIMESTAMP [Page 118]

Date and time format [Page 53]

BOOLEAN

BOOLEAN is a data type [Page 21] that can only assume one of the states TRUE or FALSE and the NULL value [Page 22].

A boolean value can only be compared to other boolean values.

See also:

BOOLEAN [Page 116]

Code Attribute

For the following character strings [Page 22], a code attribute can be entered as part of a column definition [Page 114], if required: CHAR[ACTER] [Page 115], VARCHAR [Page 116], LONG[VARCHAR] [Page 116]

A code attribute defines the sort sequence to be used for comparing values.

Code attributes

Code Attribute	Column Values
No code attribute	Code attribute defined by the database parameter DEFAULT_CODE [Extern] during database system installation
ASCII	In ASCII-Code [Page 25]
BYTE	Code neutral, i.e. the column values are not converted by the database system
EBCDIC	In EBCDIC-Code [Page 26]
UNICODE	In UNICODE [Page 24]

If you do not specify a code attribute, the code defined in the database system is used.

UNICODE

SAP DB uses the UNICODE code in line with ISO 10646, Page 1. You can find general information on UNICODE in the following documentation:

User Manual: SAP DB → *Definition of Terms* → UNICODE [Extern]

User Manual: SAP DB → SAP DB as UNICODE Database [Extern]

Metadata in UNICODE

The names of the database objects (such as table or column names) can be stored internally in UNICODE and can therefore be displayed in the database tools in the required presentation code.

User data in UNICODE

SAP DB supports the code attribute [Page 24] UNICODE for the data types CHAR[ACTER] [Page 115], VARCHAR [Page 116] and LONG[VARCHAR] [Page 116] and is able to map various presentation codes to the UNICODE format.

Code tables

- ASCII code [Page 25] pursuant to ISO 8859/1
- EBCDIC code [Page 26] CCSID 500, codepage 500

ASCII code

The ASCII code (in accordance with ISO 8859/1.2) is as follows:

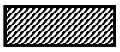
DEC	HEX	CHAR	DEC	HEX	CHAR	DEC	HEX	CHAR	DEC	HEX	CHAR
0	00	NUL	32	20	SP	64	40	@	96	60	`
1	01	SOH	33	21	!	65	41	A	97	61	a
2	02	STX	34	22	"	66	42	B	98	62	b
3	03	ETX	35	23	#	67	43	C	99	63	c
4	04	EOT	36	24	$	68	44	D	100	64	d
5	05	ENQ	37	25	%	69	45	E	101	65	e
6	06	ACK	38	26	&	70	46	F	102	66	f
7	07	BEL	39	27	'	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
128	80	
129	81	
130	82	
131	83	
132	84	
133	85	
134	86	
135	87	
136	88	
137	89	
138	8A	
139	8B	
140	8C	
141	8D	
142	8E	
143	8F	
144	90	
145	91	
146	92	
147	93	
148	94	
149	95	
150	96	
151	97	
152	98	
153	99	
154	9A	
155	9B	
156	9C	
157	9D	
158	9E	
159	9F	

DEC	HEX	CHAR
160	A0	NBSP
161	A1	¡
162	A2	¢
163	A3	£
164	A4	¤
165	A5	¥
166	A6	¦
167	A7	§
168	A8	¨
169	A9	©
170	AA	ª
171	AB	«
172	AC	
173	AD	-
174	AE	®
175	AF	¯
176	B0	°
177	B1	±
178	B2	²
179	B3	³
180	B4	´
181	B5	µ
182	B6	
183	B7	·
184	B8	¸
185	B9	¹
186	BA	º
187	BB	»
188	BC	¼
189	BD	½
190	BE	¾
191	BF	¿

DEC	HEX	CHAR
192	C0	À
193	C1	Á
194	C2	Â
195	C3	Ã
196	C4	Ä
197	C5	Å
198	C6	Æ
199	C7	Ç
200	C8	È
201	C9	É
202	CA	Ê
203	CB	Ë
204	CC	Ì
205	CD	Í
206	CE	Î
207	CF	Ï
208	D0	Ð
209	D1	Ñ
210	D2	Ò
211	D3	Ó
212	D4	Ô
213	D5	Õ
214	D6	Ö
215	D7	×
216	D8	Ø
217	D9	Ù
218	DA	Ú
219	DB	Û
220	DC	Ü
221	DD	Ý
222	DE	Þ
223	DF	ß

DEC	HEX	CHAR
224	E0	à
225	E1	á
226	E2	â
227	E3	ã
228	E4	ä
229	E5	å
230	E6	æ
231	E7	ç
232	E8	è
233	E9	é
234	EA	ê
235	EB	ë
236	EC	ì
237	ED	í
238	EE	î
239	EF	ï
240	F0	ð
241	F1	ñ
242	F2	ò
243	F3	ó
244	F4	ô
245	F5	õ
246	F6	ö
247	F7	÷
248	F8	ø
249	F9	ù
250	FA	ú
251	FB	û
252	FC	ü
253	FD	ý
254	FE	þ
255	FF	ÿ

 possibly used by the operating system

EBCDIC code

EBCDIC code CCSID 500, codepage 500:

DEC	HEX	CHAR
0	00	NUL
1	01	SOH
2	02	STX
3	03	ETX
4	04	PF
5	05	HT
6	06	LC
7	07	DEL
8	08	GE
9	09	RLF
10	0A	SMM
11	0B	VT
12	0C	FF
13	0D	CR
14	0E	SO
15	0F	SI
16	10	DLE
17	11	DC1
18	12	DC2
19	13	TM
20	14	RES
21	15	NL
22	16	BS
23	17	IL
24	18	CAN
25	19	EM
26	1A	CC
27	1B	CU1
28	1C	IFS
29	1D	IGS
30	1E	IRS
31	1F	IUS

DEC	HEX	CHAR
32	20	DS
33	21	SOS
34	22	FS
35	23	
36	24	BYP
37	25	LF
38	26	ETB
39	27	ESC
40	28	
41	29	
42	2A	SM
43	2B	CU2
44	2C	
45	2D	ENQ
46	2E	ACK
47	2F	BEL
48	30	
49	31	
50	32	SYN
51	33	
52	34	PN
53	35	RS
54	36	UC
55	37	EOT
56	38	
57	39	
58	3A	
59	3B	CU3
60	3C	DC4
61	3D	NAK
62	3E	
63	3F	SUB

DEC	HEX	CHAR
64	40	SP
65	41	RSP
66	42	â
67	43	ä
68	44	à
69	45	á
70	46	ã
71	47	å
72	48	ç
73	49	ñ
74	4A	[
75	4B	.
76	4C	<
77	4D	(
78	4E	+
79	4F	!
80	50	&
81	51	é
82	52	ê
83	53	ë
84	54	è
85	55	í
86	56	î
87	57	ï
88	58	ì
89	59	ß
90	5A	]
91	5B	$
92	5C	*
93	5D	)
94	5E	;
95	5F	°

DEC	HEX	CHAR
96	60	-
97	61	/
98	62	Â
99	63	Ä
100	64	À
101	65	Á
102	66	Ã
103	67	Å
104	68	Ç
105	69	Ñ
106	6A	¦
107	6B	,
108	6C	%
109	6D	_
110	6E	>
111	6F	?
112	70	ø
113	71	É
114	72	Ê
115	73	Ë
116	74	È
117	75	Í
118	76	Î
119	77	Ï
120	78	Ì
121	79	`
122	7A	:
123	7B	#
124	7C	@
125	7D	'
126	7E	=
127	7F	"

DEC	HEX	CHAR
128	80	Ø
129	81	a
130	82	b
131	83	c
132	84	d
133	85	e
134	86	f
135	87	g
136	88	h
137	89	i
138	8A	«
139	8B	»
140	8C	ð
141	8D	ý
142	8E	þ
143	8F	±
144	90	°
145	91	j
146	92	k
147	93	l
148	94	m
149	95	n
150	96	o
151	97	p
152	98	q
153	99	r
154	9A	ª
155	9B	º
156	9C	æ
157	9D	¸
158	9E	Æ
159	9F	¤

DEC	HEX	CHAR
160	A0	µ
161	A1	~
162	A2	s
163	A3	t
164	A4	u
165	A5	v
166	A6	w
167	A7	x
168	A8	y
169	A9	z
170	AA	¡
171	AB	¿
172	AC	Ð
173	AD	Ý
174	AE	Þ
175	AF	®
176	B0	¢
177	B1	£
178	B2	¥
179	B3	·
180	B4	©
181	B5	§
182	B6	
183	B7	¼
184	B8	½
185	B9	¾
186	BA	
187	BB	|
188	BC	¯
189	BD	¨
190	BE	´
191	BF	×

DEC	HEX	CHAR
192	C0	{
193	C1	A
194	C2	B
195	C3	C
196	C4	D
197	C5	E
198	C6	F
199	C7	G
200	C8	H
201	C9	I
202	CA	-(SHY)
203	CB	ô
204	CC	ö
205	CD	ò
206	CE	ó
207	CF	õ
208	D0	}
209	D1	J
210	D2	K
211	D3	L
212	D4	M
213	D5	N
214	D6	O
215	D7	P
216	D8	Q
217	D9	R
218	DA	¹
219	DB	û
220	DC	ü
221	DD	ù
222	DE	ú
223	DF	ÿ

DEC	HEX	CHAR
224	E0	\
225	E1	÷
226	E2	S
227	E3	T
228	E4	U
229	E5	V
230	E6	W
231	E7	X
232	E8	Y
233	E9	Z
234	EA	²
235	EB	Ô
236	EC	Ö
237	ED	Ò
238	EE	Ó
239	EF	Õ
240	F0	0
241	F1	1
242	F2	2
243	F3	3
244	F4	4
245	F5	5
246	F6	6
247	F7	7
248	F8	8
249	F9	9
250	FA	³
251	FB	Û
252	FC	Ü
253	FD	Ù
254	FE	Ú
255	FF	EO

SERIAL

SERIAL is a number generator that generates positive integers starting with 1 or a specified value.

SERIAL can be used as a DEFAULT specification [Page 120] for columns (DEFAULT SERIAL) that can only contain fixed point numbers. The maximum value generated is (10**n)-1 if DEFAULT SERIAL is defined for a column of the data type FIXED [Page 117] (n).

SERIAL columns can only be assigned a value when a row is inserted. The values of a SERIAL column cannot be changed with an UPDATE statement. A SERIAL column, therefore, can be used to determine the insertion sequence and identify a row in a table uniquely.

Parameter

SQL statements for the database system can be embedded in programming languages such as C and C++. This enables the database system to be accessed from various programs. The values to be retrieved from stored in the database system can be transferred with the SQL statements using parameters. The parameters are declared variables (so-called host variables) within the embedding program.

The data type of the host variables is defined when they are declared in the programming language. If possible, the values of the host variables are implicitly converted from the programming language data type to the data type of the database system, and vice versa.

Each parameter can be combined with an indicator variable that indicates irregularities which may have occurred when the values were assigned, for example, different value and parameter lengths, NULL value [Page 22], special NULL value [Page 22], etc. Indicator variables are essential for transferring NULL values and special NULL values. The indicator variables are declared as variables in the embedding program.

See also:

Parameter name [Page 46]

Indicator name [Page 45]

Table

- A **table** is a set of rows.
  A **row** is an ordered list of values.
  The row is the smallest unit of data that can be inserted in or deleted from a table.
  Each row in a table has the same number of columns [Page 29] and contains a value for each column.
- A **base table** is a table that usually has a permanent memory representation and description.
  It is also possible to create a base table that has only a temporary memory representation and description. This table and its description are implicitly dropped when a user stops working with the database system (end of session).
- A **result table** is a temporary table that is generated from one or more base table(s) by means of a SELECT statement.
- A **view table** is a table derived from base tables. A view table has a permanent description in the form of a SELECT statement.

Each table has a name that is unique within the overall database system. The names of existing tables can be used to name result tables. The original tables, however, cannot be accessed as long as the result tables exist.

If a table name was defined without an owner [Page 44], the catalog sections (part catalogs) are searched in the following order to locate the specified table name:

1. Catalog part of the current owner
2. Set of PUBLIC synonyms
3. Catalog part of the DBA who created the current user
4. Catalog part of the SYSDBA
5. Catalog part of the owner of the system tables

A table of another user can only be used if the relevant privileges have been granted.

See also:

Table name [Page 50]

Result table name [Page 45]

CREATE TABLE statement [Page 111]

CREATE VIEW statement [Page 137]

Column

All values in a table [Page 29] column have the same data type [Page 21]. A value in a column within a row is the smallest unit of data that can be modified or selected from a table or to which functions can be applied.

- An **alphanumeric column** is a character string [Page 22] column.
  All character strings in an alphanumeric column have the same length.
- A **numeric column** is either a floating point or a fixed point column.
  All numbers in a **floating point column** (floating point number [Page 23]) have the same mantissa length.
  All numbers in a **fixed point column** (fixed point number [Page 23]) have the same format; that is, the same number of digits before and after the decimal point.

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

See also:

Column name [Page 49]

Using column definitions in SQL statements: column_definition [Page 114]

Domain

Domain definitions enable ranges of values to be defined and designated for table columns [Page 29].

Each value range definition has a name that is unique within the overall database system.

If a domain was defined without an owner [Page 44], the catalog sections (part catalogs) are searched in the following order to locate the specified value range:

1. Catalog part of the current owner
2. Catalog part of the DBA who created the current user
3. Catalog part of the SYSDBA

See also:

Domain name [Page 44]

CREATE DOMAIN statement [Page 134]

Index

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

The assigned index name [Page 45] and table name [Page 50] must be unique.

See also:

CREATE INDEX statement [Page 141]

Synonym

A synonym is another name for a table [Page 29].

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

See also:

Synonym name [Page 49]

CREATE SYNONYM statement [Page 136]

Users and Usergroups

The following user names and passwords are defined when the database system is installed.

- DBM user

- Database system administrator (SYSDBA)
- DOMAIN user

There are four database user classes in WARM database mode:

- Database system administrator (SYSDBA)
- Database administrators (DBA users)
- RESOURCE users
- STANDARD users

Usergroups can also be defined. All of the members of a usergroup have the same rights with regard to data assigned to the group.

See also:

User Manual: SAP DB → User Concept [Extern]

User Manual: SAP DB → Definition of Terms [Extern]

User name [Page 43]

Usergroup name [Page 43]

CREATE USER statement [Page 151]

CREATE USERGROUP statement [Page 153]

Privilege

A privilege is used to impose restrictions on operations carried out on certain objects.

Users can only execute operations on objects if they have been granted the privileges to do so. The owner of an object receives all of the relevant privileges when the object is created. Privileges can be explicitly granted to other users. Privileges are not granted to other users implicitly.

Users who are not the owner of an object can only grant privileges to other users if they have already been granted these privileges and are allowed to pass them on, i.e. with the relevant option.

See also:

Privilege type [Page 47]

GRANT statement [Page 161]

Role

A role is a collection of privileges [Page 31].

Like a privilege, a role can be assigned to a different role or to a user.

While privileges are always valid, roles are always inactive, i.e. the privileges they contain are not valid. Roles can be activated for individual sessions. Each user who is assigned a role can also define which roles are to be active in each of his or her sessions without having to define this in each individual session.

All roles are inactive for the current session while data definition commands are being executed.

See also:

Role name [Page 48]

CREATE ROLE statement [Page 160]

Database Catalog/User Data

Logical data storage takes place in the following areas of a SAP DB database:

- The **database catalog** comprises metadata containing the definitions of database objects such as base tables [Page 29], view tables [Page 29], synonyms [Page 30], value ranges [Page 30], indexes [Page 30], users and usergroups [Page 30].
- The **user data** is all of the rows in all of the base tables.

Transaction

You will find an explanation of the term in *User Manual: SAP DB* → *Definition of Terms* → Transaction [Extern].

Locks

You will find an explanation of the term in *User Manual: SAP DB* → *Definition of Terms* → Lock [Extern].

See also:

Transactions [Page 202]

Subtransaction [Page 32]

COMMIT statement [Page 207]

ROLLBACK statement [Page 208]

CONNECT statement [Page 204]

LOCK option [Page 191]

Subtransaction

The purpose of closed, nested transactions (subtransactions) is to let a series of database operations within a transaction [Page 32] appear as a unit with regard to modifications to the database.

Subtransactions are preceded by SUBTRANS BEGIN and closed by SUBTRANS END or SUBTRANS ROLLBACK.

- If a subtransaction is concluded with SUBTRANS END, any modifications made are kept.
- If a subtransaction is closed with SUBTRANS ROLLBACK, all modifications made to the database system are reversed. Modifications made by subtransactions contained in this subtransaction are also reversed, even if they were concluded with SUBTRANS END.

SUBTRANS END and SUBTRANS ROLLBACK do not affect locks. These are only released by COMMIT or ROLLBACK. COMMIT or ROLLBACK implicitly close all subtransactions.

See also:

Transactions [Page 202]

SUBTRANS statement [Page 208]

COMMIT statement [Page 207]

ROLLBACK statement [Page 208]

Database Session

You will find an explanation of the term in *User Manual: SAP DB* → *Definition of Terms* → Database session [Extern].

See also:

Users and user groups [Page 30]

Password [Page 46]

CONNECT statement [Page 204]

SET statement [Page 206]

Data integrity

- **Integrity rules**: the database system provides a wide range of declarative integrity rules, thus reducing the programming requirements for applications. Integrity rules that refer to a table can also be specified (see constraint name [Page 43]).
- **Key**: a key comprising one or more columns can be defined for each table. The database system ensures that each key is unique. A key can also consist of columns of different data types (see key definition [Page 126]).
- **UNIQUE definition**: the uniqueness of the values in other columns and column combinations can also be ensured by using other mechanisms (see UNIQUE definition [Page 126] for "alternate keys").
- **NOT NULL**: by specifying NOT NULL, you can ensure that the NULL value is not accepted in individual columns.
- **DEFAULT definition**: you can define default values for each column (see DEFAULT specification [Page 120]).
- **Referential integrity conditions**: by specifying referential integrity conditions, you can declare deletion and existence dependencies between the rows in two tables (see name of a referential constraint [Page 48]).
- **Database procedures and triggers**: complex integrity rules that require access to further tables can be formulated with database procedures [Page 33] or triggers [Page 34].

Database procedure

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

In client/server configurations, the client and server interact when an SQL statement is executed in the access layer. The number of these interactions can be reduced considerably by transferring the SQL access layer from the client to the server.

SAP DB provides a language (special SQL syntax) for this purpose that allows an SQL access layer to be formulated on the server side. This special SQL syntax can be used to define database procedures and triggers [Page 34].

This has three main advantages:

- The number of interactions between client and server is reduced considerably (several factors). Client/server communication is only required for each operation on the application object, and not for each SQL statement. This enhances the performance of client-server configurations considerably.
- The SQL access layer contains the procedurally formulated integrity and business rules. By concentrating these rules on the server side and eliminating them from the database applications, modifications can be made centrally and thus become valid immediately in all database applications. In this way, the integrity and decision rules also become a part of the catalog in the database system.
- An SQL access layer in the form of database procedures transferred to the server side is an essential customizing tool, as it allows customer-specific database functionality to be included.

To be able to execute a database procedure, users must have the call privilege. This call privilege is independent of the user privileges for the tables and columns used in the database procedure. As a

result, users may be able to use a database procedure to execute SQL statements that they otherwise would not have access to.

Database procedures are called explicitly from the programming language of the application. They can contain parameters, except for LONG column [Page 23]s. The extent to which LONG columns can be used within database procedures depends on the length of the LONG columns and the amount of storage space available.

As with any SQL statement, precautions must be taken to ensure that calling a database procedure has the desired effect, and that errors do not have any lasting effects on the database system. SAP DB provides nested transactions for this purpose. Each database procedure call can run in a subtransaction [Page 32] that can be reset without interfering with transaction control in the database application.

See also:

Name of a database procedure [Page 44]

Trigger

While database procedures [Page 33] are called explicitly from the programming language of the application, triggers are special procedures that run implicitly on a base table (or a view table built on this base table) after a data manipulation statement has been executed.

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

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

A trigger can call further triggers implicitly.

Triggers can be used to check complicated integrity rules, to initiate derived database modifications for the row in question, or to implement complex access protection rules.

SAP DB provides a language (special SQL syntax) that can be used to define database procedures and triggers.

See also:

Trigger name [Page 51]

SQL mode

You will find an explanation of the term in *User Manual: SAP DB → Definition of Terms →* SQL mode [Extern].

This document describes the functionality of the database system provided by the INTERNAL SQL mode.

Only those SQL statements are described for which the same SQL mode is used to generate the object and execute the statement for the object. If database objects are created in one SQL mode and addressed in another, the object may have properties that are not known in the current SQL mode and, therefore, cannot be described.

SQL statement for specifying the SQL mode

CONNECT statement [Page 204]

Basic elements

Character [Page 35]

Character

A `character` is an element of a character string [Page 22] or keyword [Page 39].

Syntax

```
<character> ::= <digit>
              | <letter>
              | <extended_letter>
              | <hex_digit>
              | <language_specific_character>
              | <special_character>
```

digit [Page 35], letter [Page 35], extended_letter [Page 35], hex_digit [Page 36], language_specific_character [Page 36], special_character [Page 36]

Digit

A `digit` is a character [Page 35].

Syntax

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

Letter

A `letter` is a character [Page 35].

Syntax

```
<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

An extended letter is a character [Page 35].

Syntax

```
<extended_letter> ::= # | @ | $
```

hex_digit

A `hex_digit` is a character [Page 35].

Syntax

```
<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

A language-specific character [Page 35] is any letter that occurs in a northern, southern, or central European language and is not contained in the list of letters [Page 35].

German umlauts: ä, ö, ü

French letters with a “grave” accent.

If you have installed a UNICODE [Page 24]-enabled database, a language-specific character is a character that is not included in the ASCII-Code [Page 25] list from 0 to 127.

Special character

A `special character` is any character [Page 35] that is not contained in the following list:

- digit [Page 35]
- letter [Page 35]
- extended_letter [Page 35]
- hex_digit [Page 36]
- language_specific_character [Page 36]
- Characters that indicate the end of a line in a file

String literal

A string literal [Page 36] is a sequence of characters in double quotes. String literals can also be represented in hexadecimal notation by preceding them with x or X.

Syntax

```
<string_literal> ::= '' | '<character>...' | <hex_literal>
```

character [Page 35], hex_literal [Page 37]

```
'69190 Walldorf'
'Anthony Smith'
X'12ab'
```

hex_literal

String literal [Page 36] that contains a value in hexadecimal notation.

Syntax

```
<hex_literal> ::= x'' | X'' | x'<hex_digit_seq>' | X'<hex_digit_seq>'
```

hex_digit_seq [Page 37]

```
x'123F'
X'12ab'
```

hex_digit_seq

Sequence of hexadecimal digits (`hex_digit_seq`).

Syntax

```
<hex_digit_seq> ::=
<hex_digit><hex_digit> | <hex_digit_seq><hex_digit><hex_digit>
```

hex_digit [Page 36]

Numeric literal

A numeric literal [Page 36] is a number [Page 23] represented as a fixed or floating point number.

Syntax

```
<numeric_literal> ::= <fixed_point_literal> | <floating_point_literal>
```

fixed_point_literal [Page 37], floating_point_literal [Page 38]

Fixed point literal

Numeric literal [Page 37] that specifies a number [Page 23] as a fixed point number.

Syntax

```
<fixed_point_literal> ::=   [<sign>]<digit_sequence>[.<digit_sequence>]
| [sign]<digit_sequence>. | [sign].<digit_sequence>
```

sign [Page 37], digit_sequence [Page 38]

```
+123.12
1234.
```

Sign

Sign

Syntax

```
<sign> ::= + | -
```

Digit sequence

Sequence of digits

Syntax

```
<digit_sequence> ::= <digit>...
```

digit [Page 35]

Floating point literal

Numeric literal [Page 37] that specifies a number [Page 23] as a floating point number.

Syntax

```
<floating_point_literal> ::=  <mantissa>E<exponent> | <mantissa>e<exponent>
```

mantissa [Page 38], exponent [Page 38]

```
1e160
-765E-04
```

Mantissa

Mantissa

Syntax

```
<mantissa> ::= <fixed_point_literal>
```

fixed_point_literal [Page 37]

Exponent

Exponent

Syntax

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

sign [Page 37], digit [Page 35]

Unsigned integer

An `unsigned_integer` is a special numeric literal.

Syntax

```
<unsigned_integer> ::= <numeric_literal>
```

numeric_literal [Page 37]

Explanation

An unsigned integer can be represented in any way but must be a positive integer.

Integer

An integer is a special numeric_literal [Page 37]. This integer can be displayed in any number of ways.

Syntax

```
<integer> ::= [sign]<unsigned_integer>
```

sign [Page 37], unsigned_integer [Page 38]

Token

A character set or token comprises a series of characters that are combined to form a lexical unit. A distinction is made between regular and delimiter tokens.

Syntax

```
<token> ::= <regular_token> | <delimiter_token>
```

regular_token [Page 39], delimiter_token [Page 41]

```
SELECT * FROM reservation
ALTER TABLE reservation DROP FOREIGN KEY customer_reservation
```

Explanation

Each token can be followed by any number of blanks. Each regular token must be followed by a delimiter token or a blank.

Double_quotes [Page 41] within a special_identifier [Page 41] are represented by two consecutive quotes.

Regular token

Normal character set (token [Page 39]) (`regular_token`)

Syntax

```
<regular_token> ::= <literal> | <keyword> | <identifier> | <parameter_name>
```

Literal [Page 36], key_word [Page 39], identifier [Page 40], parameter_name [Page 46]

```
SELECT
'Tours10'
```

Keyword

Keyword. A distinction is made between „normal“ and reserved keywords.

Syntax

```
<keyword> ::= <not_reserved_key_word> | <reserved_keyword>
```

Not_reserved_keyword [Page 40], reserved_keyword [Page 40]

Explanation

Keywords can be entered in uppercase/lowercase characters.

Reserved keywords must not be used in simple identifiers [Page 40]. Reserved keywords, however, can be specified in the form of special identifiers [Page 41].

Not reserved keyword

Keywords [Page 39] (`not reserved keywords`). If possible, these key words should not be used to designate objects.

You can find a list of all keywords in the syntax directory: not_reserved_key_word [Page 269]

Reserved keyword

Reserved keywords [Page 39] (`reserved key_word`). These key words must not be used to designate objects.

You can find a list of all keywords in the syntax directory: reserved_key_word [Page 277]

Identifier

A distinction is made between simple identifiers and special identifiers.

Syntax

```
<identifier> ::= <simple_identifier> |
<double_quotes><special_identifier><double_quotes>
```

simple_identifier [Page 40], double_quotes [Page 41], special_identifier [Page 41]

Explanation

Identifiers can be entered in uppercase/lowercase characters.

Reserved keywords [Page 40] must not be used in simple identifiers. Reserved keywords, however, can be specified in the form of special identifiers.

Double quotes within a special identifier are represented by two consecutive quotes.

Simple identifier: `reservation`

Special identifier: `"ADD"`

Simple identifier

Simple identifier [Page 40]. The first character in a simple identifier must not be a digit or underscore [Page 41].

Syntax

```
<simple_identifier> ::= <first_character>[<identifier_tail_character>...]
```

first_character [Page 41], identifier_tail_character [Page 41]

Explanation

Simple identifiers are always converted into uppercase characters in the database. For this reason, simple identifiers are not case sensitive.

If the name of a database object is to contain lowercase letters, special characters, or blanks, the identifier must be specified as a special identifier [Page 41] (in double quotes).

First character

First character in a simple identifier [Page 40].

Syntax

```
<first_character> ::= <letter> | <extended_letter> |
<language_specific_character>
```

letter [Page 35], extended_letter [Page 35], language_specific_character [Page 36]

Identifier tail character

Character allowed after the first character in a simple identifier [Page 40] or password [Page 42].

Syntax

```
<identifier_tail_character> ::=
<letter> | <extended_letter> | <language_specific_character>
| <digit> | <underscore>
```

letter [Page 35], extended_letter [Page 35], language_specific_character [Page 36], digit [Page 35], underscore [Page 41]

Underscore

Underscore

Syntax

```
<underscore> ::= _
```

Double quotes

Quotation marks (`double_quotes`)

Syntax

```
<double_quotes> ::= "
```

Special identifier

Special identifier [Page 40]. A special identifier must be entered in double quotes if it is to be used as an identifier.

Syntax

```
<special_identifier> ::= <special_identifier_character>...
```

`<special_identifier_character> ::=` any characters [Page 35], that can be linked in any sequence

Delimiter token

Delimiter token [Page 39]

Syntax

```
<delimiter_token> ::=
   ( | ) | , | . | + | - | * | / | < | > | <> | != | = | <= | >=
| ¬= | ¬< | ¬> (for machines with EBCDIC code [Page 26])
| ~= | ~< | ~> (for machines with ASCII code [Page 25])
```

Names

Names identify objects. The following list contains names that are frequently used in the syntax description of the SQL statements.

Name	
Alias name [Page 43]	alias_name
Column name [Page 49]	column_name
Constraint name [Page 43]	constraint_name
Name of a database procedure [Page 44]	dbproc_name
Domain name [Page 44]	domain_name
Index name [Page 45]	index_name
Indicator name [Page 45]	indicator_name
MapChar set-name [Page 46]	mapchar_set_name
Owner [Page 44]	owner
Parameter name [Page 46]	parameter_name
Password [Page 46]	password
Reference name [Page 48]	reference_name
Name of a referential constraint [Page 48]	referential_constraint_name
Result table name [Page 45]	result_table_name
Role name [Page 48]	role_name
Sequence name [Page 49]	sequence_name
Synonym name [Page 49]	synonym_name
Table name [Page 50]	table_name
Terminal character set name [Page 50]	termchar_set_name
Trigger name [Page 51]	trigger_name
Usergroup name [Page 43]	usergroup_name
User name [Page 43]	user_name

Explanation

For all names consisting of several identifiers that are separated by a ".", any number of blanks can be specified before and after the dot.

Alias name

An alias name is a new column name [Page 49] that specifies the name of a column in the following types of tables:

- View tables
- Tables defined with a recursive DECLARE CURSOR statement

Syntax

```
<alias_name> ::= <identifier>
```

identifier [Page 40]

Explanation

An error message is output if the name has more than 32 characters.

SQL statement for defining an alias name

CREATE VIEW statement [Page 137]

Recursive DECLARE CURSOR statement [Page 176]

Usergroup name

A usergroup name identifies a usergroup [Page 30].

Syntax

```
<usergroup_name> ::= <identifier>
```

identifier [Page 40]

Explanation

An error message is output if the name has more than 32 characters.

SQL statement for defining a usergroup name

CREATE USERGROUP statement [Page 155]

User name

A user name defines a user [Page 30].

Syntax

```
<user_name> ::= <identifier>
```

identifier [Page 40]

Explanation

An error message is output if the name has more than 32 characters.

SQL statement for defining a user name

CREATE USER statement [Page 151]

Constraint name

A constraint name defines a condition (data integrity [Page 33]) that must be satisfied by the rows in a table.

Syntax

```
<constraint_name> ::= <identifier>
```

identifier [Page 40]

Explanation

An error message is output if the name has more than 32 characters.

SQL statements for defining a constraint name

CONSTRAINT definition [Page 122]

CREATE TABLE statement [Page 111]

ALTER TABLE statement [Page 127]

Name of a database procedure (dbproc_name)

The name of a database procedure (`dbproc_name`) designates a database procedure [Page 33].

Syntax

```
<dbproc_name> ::= [<owner>.]<procedure_name>
<procedure_name> ::= <identifier>
```

owner [Page 44], identifier [Page 40]

Explanation

You cannot specify TEMP as the owner in a database procedure.

SQL statements for creating, calling, and dropping a database procedure

CREATE DBPROC statement [Page 145]

CALL statement [Page 173]

DROP DBPROC statement [Page 148]

Domain name

A domain name identifies a domain [Page 30] in a table column [Page 29].

Syntax

```
<domain_name> ::= [<owner>.]<identifier>
```

owner [Page 44], identifier [Page 40]

Explanation

An error message is output if the name has more than 32 characters.

SQL statement for defining a domain name

CREATE DOMAIN statement [Page 134]

You cannot specify TEMP as the owner in a domain name.

Owner

The owner of an object is defined by specifying the name of owner name.

Syntax

```
<owner> ::= <user_name> | <usergroup_name> | TEMP
```

user_name [Page 43], usergroup_name [Page 43]

Explanation

If the owner is not a member of a usergroup, the owner name and usergroup name are identical.

If TEMP is specified as the owner in a table name [Page 50], the table is a temporary table. The owner of this temporary table is the current owner.

An error message is output if the name of the owner has more than 32 characters.

Result table name

A result table name identifies a result table (see table [Page 29]).

Syntax

```
<result_table_name> ::= <identifier>
```

identifier [Page 40]

Explanation

An error message is output if the name has more than 32 characters.

SQL statement for defining a result table name

QUERY statement [Page 174]

Index name

An index name identifies an index [Page 30].

Syntax

```
<index_name> ::= <identifier>
```

identifier [Page 40]

Explanation

An error message is output if the name has more than 32 characters.

SQL statement for creating an index

CREATE INDEX statement [Page 141]

Indicator name

An indicator name designates an indicator variable in an application that can be specified together with a parameter name.

Syntax

```
<indicator_name> ::= <parameter_name>
```

parameter_name [Page 46]

Explanation

The indicator variable of a parameter [Page 28] provides information on any irregularities (e.g. NULL value, difference value and parameter lengths).

MapChar Set Name (mapchar_set_name)

A MapChar set name (`mapchar_set_name`) identifies a MapChar set.

Syntax

```
<mapchar_set_name> ::= <identifier>
```

identifier [Page 40]

Function in which MapChar sets are used

MAPCHAR [Page 66]

See also:

User Manual: SAP DB → Definition of Terms → Language Support (MapChar Sets [Extern]

Password

Users require a password to connect to the database instance (start a database session [Page 32]).

Syntax

```
<password> ::= <identifier> |
<first_password_character>[<identifier_tail_character>...]
```

identifier [Page 40], identifier_tail_character [Page 41]

```
<first_password_character> ::= <letter> | <extended_letter> |
<language_specific_letter> | <digit>
```

letter [Page 35], extended_letter [Page 35], language_specific_letter [Page 36], digit [Page 35]

Explanation

Passwords are truncated after 18 characters.

SQL statement for defining a user password

CREATE USER statement [Page 151]

SQL statement for changing a user password

ALTER PASSWORD statement [Page 159]

Parameter name

A parameter name identifies a parameter [Page 28] (host variable) in an application containing SQL statements from the database system.

Syntax

```
<parameter_name> ::= :<identifier> | :<identifier>(<identifier>) |
:<identifier>(.<identifier>.)
```

identifier [Page 40]

Explanation

The conventions of the programming language in which the SQL statements of the database system are embedded determine the number of significant characters in the parameter name.

Identifiers for parameter names may contain the characters "." and "_", but not as the first character.

Privilege type (privilege)

A privilege type (`privilege`) identifies a certain privilege [Page 31].

Syntax

```
<privilege> ::= INSERT | UPDATE [(<column_name>,...)]
| SELECT [(<column_name>,...)] | SELUPD [(<column_name>,...)]
| DELETE | INDEX | ALTER | REFERENCES [(<column_name>,...)]
```

column_name [Page 49]

Explanation

Privilege	Explanation
INSERT	Allows the identified user to insert rows in the specified table. The current user must be authorized to grant the INSERT privilege.
UPDATE	Allows the identified user to update rows in the specified table. If column names are specified, the rows may only be updated in the columns identified by these names. The current user must be authorized to grant the UPDATE privilege.
SELECT	Allows the identified user to select rows in the specified table. If column names are specified, the rows may only be selected in the columns identified by these names. The current user must be authorized to grant the SELECT privilege.
SELUPD	The SELECT and UPDATE privileges are granted. If column names are specified, the rows may only be selected or updated in the columns identified by these names. The current user must be authorized to grant both the SELECT and the UPDATE privileges.
DELETE	Allows the identified user to delete rows from the specified table. The current user must be authorized to grant the DELETE privilege.
INDEX	Allows the identified user to execute the CREATE INDEX and DROP INDEX statements for the specified tables. The INDEX privilege can only be granted for base tables. The current user must be authorized to grant the INDEX privilege.
ALTER	Allows the identified user to execute the ALTER TABLE statement for the specified tables. The ALTER privilege can only be granted for base tables. The current user must be authorized to grant the ALTER privilege.
REFERENCES	Allows the identified user to specify the table as a referenced table in a column_definition [Page 114] or referential_constraint_definition [Page 123].

SQL statement for granting or revoking privileges

GRANT statement [Page 161]

REVOKE statement [Page 162]

Name of a referential constraint (referential_constraint_name)

The name of a referential constraint (`referential_constraint_name`) identifies a referential constraint (referential integrity condition, data integrity [Page 33]). This integrity condition specifies deletion and existence dependencies between two tables.

Syntax

```
<referential_constraint_name> ::= <identifier>
```

identifier [Page 40]

Explanation

An error message is output if the name has more than 32 characters.

SQL statement for defining a referential constraint

Referential CONSTRAINT definition [Page 123]

CREATE TABLE statement [Page 111]

ALTER TABLE statement [Page 127]

Reference name

The reference name is an identifier that is declared for a certain validity range and associated with exactly one table [Page 29]. The scope of this declaration is the entire SQL statement.

Syntax

```
<reference_name> ::= <identifier>
```

identifier [Page 40]

Explanation

The same reference name specified in various scopes can be associated with different tables or with the same table.

An error message is output if the name has more than 32 characters.

Role name

A role name identifies a role [Page 31].

Syntax

```
<role_name> ::= <identifier>
```

identifier [Page 40]

Explanation

An error message is output if the name has more than 32 characters.

SQL statement for defining a role

CREATE ROLE statement [Page 160]

SQL statement for granting a role

GRANT statement [Page 161]

SQL statements for activating a role

ALTER USER statement [Page 156]

ALTER USERGROUP statement [Page 157]

SET statement [Page 206]

SQL statement for dropping a role

DROP ROLE statement [Page 160]

Sequence name

A sequence name identifies a sequence of values.

Syntax

```
<sequence_name> ::= <identifier>
```

identifier [Page 40]

Explanation

A sequence is a series of values that are generated in accordance with certain rules. The step width between these values can be defined, among other things.

Sequences can be used to generate unique values. These are not uninterrupted, because values generated within a transaction that was rolled back cannot be used again.

An error message is output if the name has more than 32 characters.

SQL statement for defining a sequence name

CREATE SEQUENCE statement [Page 135]

Column name

A column name identifies a column [Page 29].

Syntax

```
<column_name> ::= <identifier>
```

identifier [Page 40]

Explanation

An error message is output if the name has more than 32 characters.

SQL statements for defining a column name

CREATE TABLE statement [Page 111]

CREATE VIEW statement [Page 137]

ALTER TABLE statement [Page 127]

QUERY statement [Page 174]

Synonym name

A synonym name identifies a synonym [Page 30] for a table name.

Syntax

```
<synonym_name> ::= <identifier>
```

identifier [Page 40]

Explanation

If the synonym is not a PUBLIC synonym, it is only known by one user or usergroup [Page 30]. A PUBLIC synonym is known by every user and user group.

An error message is output if the name has more than 32 characters.

SQL statement for defining a synonym name

CREATE SYNONYM statement [Page 136]

Table name

A table name identifies a table [Page 29].

Syntax

```
<table_name> ::= [<owner>.]<identifier>
```

owner [Page 44], identifier [Page 40]

Explanation

The database system uses certain table names for internal purposes. The identifiers [Page 40] of these tables begin with SYS. To avoid naming conflicts, therefore, you should not use table names that start with SYS.

If the name of the owner [Page 44] was not specified in the table name, the catalog parts are searched for the table name in the following order:

1. Catalog part of the current owner
2. Set of PUBLIC synonyms
3. Catalog part of the DBA who created the current user
4. Catalog part of the SYSDBA
5. Catalog part of the owner of the system tables

An error message is output if the name has more than 32 characters.

SQL statements for defining a table name

CREATE TABLE statement [Page 111]

CREATE VIEW statement [Page 137]

CREATE SYNONYM statement [Page 136]

Terminal character set name (termchar set name)

A terminal character set name (`termchar_set_name`) identifies a terminal character set. This can be made available for the database instance.

Syntax

```
<termchar_set_name> ::= <identifier>
```

identifier [Page 40]

Explanation

An error message is output if the name has more than 32 characters.

SQL statement for specifying a terminal character set name

CONNECT statement [Page 204]

See also:

User Manual: SAP DB → Definition of Terms → Language Support (Termchar Sets [Extern]

Trigger name

A trigger name identifies a trigger [Page 34] that is defined for a table.

Syntax

```
<trigger_name> ::= <identifier>
```

identifier [Page 40]

Explanation

SQL statements for creating and dropping a trigger

CREATE TRIGGER statement [Page 149]

DROP TRIGGER statement [Page 150]

Column specification (column_spec)

A column specification (`column_spec`) specifies a column in a table.

Syntax

```
<column_spec> ::= <column_name> | <table_name>.<column_name>
| <reference_name>.<column_name> | <result_table name>.<column_name>
```

column_name [Page 49], table_name [Page 50], reference_name [Page 48], result_table_name [Page 45]

Explanation

For all names consisting of several identifiers [Page 40] separated by a ".", any number of blanks can be specified before and after the dot.

Parameter specification (parameter spec)

A parameter specification (`parameter spec`) identifies the parameter name and, if necessary, the indicator name that are necessary to specify a parameter.

Syntax

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

parameter_name [Page 46], indicator name [Page 45]

Explanation

The indicator must be declared as a variable in the embedding programming language. It must be possible to assign at least four-digit integers to this variable.

A distinction is made between output parameters and input parameters:

- **Output parameters**: parameters that are to receive values retrieved from the database system.
  - An indicator parameter with the value 0 indicates that the transferred value is not a NULL value and that the parameter value is the transferred value.
  - An indicator with the value –1 indicates that the parameter value is the NULL value.

  - Alphanumeric output parameters: an indicator with a value greater than 0 indicates that the assigned character string was too long and was truncated as a result. The indicator indicates the untruncated length of the original output column.
  - Numeric output parameters: an indicator with a value greater than 0 indicates that the assigned value has too many significant digits and decimal places have been truncated. The indicator indicates the number of digits in the original value.
  - With numeric output parameters, an indicator with the value -2 indicates that the parameter value is the special NULL value [Page 22].

- **Input parameters**: parameters containing values that are to be transferred to the database system.
  - An indicator with a value greater than or equal to 0 indicates that the specified parameter value is the value that is to be transferred to the database system.
  - An indicator with the value less than 0 indicates that the specified parameter value is the NULL value.

Specifying values (extended value spec)

Values can be specified (`extended value spec`) by specifying values (`value spec`) or with one of the keywords DEFAULT or STAMP.

Syntax

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

value_spec [Page 52]

Explanation

- DEFAULT keyword
  DEFAULT identifies the default value for the column in a CREATE TABLE statement [Page 111] or ALTER TABLE statement [Page 127]. DEFAULT cannot be used to specify values if one of these values is not defined.
  The DEFAULT keyword can be used in the following **SQL statements**:
  INSERT statement [Page 164]
  UPDATE statement [Page 169]
  The DEFAULT keyword can be used in a DEFAULT predicate [Page 97].

- STAMP key word
  The database system is able to generate unique values. This is a serial number that starts with X'000000000001'. The values are assigned in ascending order. It cannot be ensured that a sequence of values is uninterrupted. The STAMP key word supplies the next value generated by the database system.
  The STAMP keyword can be used in the following **SQL statements** (only on columns of the data type CHAR(n) BYTE with n>=8, see DEFAULT specification [Page 120]):
  INSERT statement [Page 164]
  UPDATE statement [Page 169]
  If the user wants to find out the generated value before it is applied to the column, the following **SQL statement** must be used:
  NEXT STAMP statement [Page 173]

Specifying values (value spec)

Values can be specified (`value_spec`) by specifying literals, parameter specifications, or a series of keywords.

Syntax

```
<value_spec> ::= <literal> | <parameter_spec>
| NULL | USER | USERGROUP | LOCALSYSDBA
| SYSDBA [(<user_name>)] | SYSDBA [(<usergroup_name>)]
| [<owner>.]<sequence_name>.NEXTVAL | [<owner>.]<sequence_name>.CURRVAL
| DATE | TIME | TIMESTAMP | TIMEZONE | TRUE | FALSE | TRANSACTION | UTCDIFF
```

literal	Literal [Page 36]
parameter spec	Parameter spec [Page 51]
NULL	NULL value [Page 22]
USER	Current user name [Page 43]
USERGROUP	Name of the usergroup [Page 43] to which the user calling the SQL statement belongs. If the user does not belong to a user group, the user name is displayed.
LOCALSYSDBA	SYSDBA of the database instance
SYSDBA [(<user name>)] | SYSDBA [(<usergroup name>)]	SYSDBA of the database instance
[<owner>.]<sequence name>.NEXTVAL	Next value generated for the specified sequence name [Page 49] (of the owner [Page 44] in question).
[<owner>.]<sequence name>.CURRVAL	Value generated last for the specified sequence name using [<owner>.]<sequence name>.NEXTVAL.
DATE	Current date [Page 23]
TIME	Current time [Page 23]
TIMESTAMP	Current timestamp [Page 24]
TIMEZONE	Time zone. This value is assigned the value 0 and cannot be modified.
TRUE | FALSE	Corresponding value of a column of the data type BOOLEAN [Page 24]
TRANSACTION	Identification of the current transaction [Page 32]. This is a value of data type CHAR(10) BYTE.
UTCDIFF	Time difference in hours (in data type FIXED(4,2)) between your local time value and the UTC time value (Greenwich Mean Time)

Date and time format (datetimeformat)

The date and time format (datetimeformat) specifies the way in which date values [Page 23], time values [Page 23], and timestamp values [Page 24] are represented.

Syntax

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

Date value

'YYYY'	Four-digit year format
'MM'	Two-digit month format (01-12)
'DD'	Two-digit day format (01-31)

Format	General Format	Example
EUR	'DD.MM.YYYY'	'23.01.1999'
INTERNAL	'YYYYMMDD'	'19990123'
ISO/JIS	'YYYY-MM-DD'	'1999-01-23'
USA	'MM/DD/YYYY'	'01/23/1999'

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

Time value

'HHHH'	Four-digit hour format
'HH'	Two-digit hour format
'MM'	Two-digit minute format (00-59)
'SS'	Two-digit second format (00-59)

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

In all time formats, the identifier of the hour must consist of at least one digit. In the USA time format, the minute identifier can be omitted completely. In all the other formats, with the exception of INTERNAL, the minute and second identifiers must comprise at least one digit.

Timestamp value

'YYYY'	Four-digit year format
'MM'	Two-digit month format (01-12)
'DD'	Two-digit day format (01-31)
'HH'	Two-digit hour format (0-24)
'MM'	Two-digit minute format (00-59)
'SS'	Two-digit second format (00-59)
'MMMMMM'	Six-digit microsecond format

Format	General Format	Example
EUR/JIS/USA	'YYYY-MM-DD- HH.MM.SS.MMMMMM'	'1999-01-23- 14.30.08.456234'
ISO	'YYYY-MM-DD HH:MM:SS.MMMMMM'	'1999-01-23 14:30:08.456234'

INTERNAL	'YYYYMMDDHHMMSSMMMMMM'	'19990123143008456234'

The microsecond identifier can be omitted in all timestamp formats. In all formats, with the exception of INTERNAL, the month and day identifiers must consist of at least one digit.

Explanation

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

The date and time format is determined when the database system is installed.

Users can change the date and time format for the current session by setting the relevant parameters in the database tools or by specifying the corresponding parameters when using programs.

The ISO date and time format is used by ODBC and JDBC applications and cannot be replaced with a different date and time format.

Specifying a string (string spec)

Only expressions [Page 90] that have an alphanumeric value as a result are allowed as a string specification (`string_spec`).

Specifying a key (key_spec)

A key specification (`key_spec`) allows rows in a table to be located whose key column values match the values in the key specification. A row with the specified key values does not have to exist.

Syntax

```
<key_spec> ::= <column_name> = <value_spec>
```

column_name [Page 49], value_spec [Page 52]

Explanation

The value specification (`value_spec`) must **not** be NULL.

The column name must identify a key column in the table.

The key specification must contain all the key columns in a table. The individual key specifications (`key_spec`) must be separated by commas.

For tables defined without key columns, there is the implicitly generated column SYSKEY CHAR(8) BYTE which contains a key generated by the database system. This column can only be used in a key specification.

Function (function_spec)

There is a series of functions that can be applied to a value (row) as an argument (`function_spec`) and supply a result.

Syntax

```
<function_spec> ::= <arithmetic_function> | <trigonometric_function>
| <string_function> | <date_function> | <time_function>
| <extraction_function> | <special_function>
| <conversion_function>
```

arithmetic_function [Page 56], trigonometric_function [Page 62], string_function [Page 62], date_function [Page 73], time_function [Page 76], extraction_function [Page 77], special_function [Page 80], conversion_function [Page 81]

Explanation

The arguments and results of the functions are numeric, alphanumeric or Boolean values that are subject to certain restrictions. LONG columns [Page 23] are not allowed as arguments.

Arithmetic function

An arithmetic function is a function [Page 55] that supplies a numeric value as a result.

Syntax

```
<arithmetic_function> ::=
  TRUNC    ( <expression>[, <expression>] )
| ROUND    ( <expression>[, <expression>] )
| NOROUND  ( <expression> )
| FIXED    ( <expression>[, <unsigned_integer> [, <unsigned_integer] ] )
| FLOAT    ( <expression>[, <unsigned_integer> ] )
| CEIL     ( <expression> )
| FLOOR    ( <expression> )
| SIGN     ( <expression> )
| ABS      ( <expression> )
| POWER    ( <expression>, <expression> )
| EXP      ( <expression> )
| SQRT     ( <expression> )
| LN       ( <expression> )
| LOG      ( <expression>, <expression> )
| PI
| LENGTH   ( <expression> )
| INDEX    ( <string_spec>, <string_spec> [,<expression>[, <expression>] ] )
```

expression [Page 90], unsigned_integer [Page 38], string_spec [Page 55]

TRUNC(a,n) [Page 61], ROUND(a,n) [Page 60], NOROUND(a) [Page 60], FIXED(a,p,s) [Page 57], FLOAT(a,s) [Page 57], CEIL(a) [Page 56], FLOOR(a) [Page 57], SIGN(a) [Page 61], ABS(a) [Page 56], POWER(a,n) [Page 60], EXP(a) [Page 57], SQRT(a) [Page 61], LN(a) [Page 59], LOG(a,b) [Page 60], PI [Page 60], LENGTH(a) [Page 59], INDEX(a,b,p,s) [Page 58]

ABS(a)

ABS(a) is an arithmetic_function [Page 56] that determines the unsigned value (absolute value) of the number a.

	Result of the ABS(a) function
a is NULL value	NULL value [Page 22]
a is special NULL value	Special NULL value [Page 22]

CEIL(a)

CEIL(a) is an arithmetic function [Page 56] that calculates the smallest integer value that is greater than or equal to the number a.

The result is a fixed point number with 0 decimal places.

An error message is output if it is not possible to represent the result of CEIL(a) as a fixed point number.

	Result of CEIL(a) function

a is the NULL value	NULL value [Page 22]
a is special NULL value	Special NULL value [Page 22]

EXP(a)

EXP(a) is an arithmetic function [Page 56] that calculates the power from base e (2.71828183) and the exponent a ("e to the power of a").

	Result of EXP(a) function
a is NULL value	NULL value [Page 22]
a is special NULL value	Special NULL value [Page 22]

FIXED(a,p,s)

FIXED(a,p,s) is an arithmetic function [Page 56] that is used to output the number a in a format of the data type FIXED [Page 117](p,s).

Digits after the decimal point are rounded to s decimal places, if necessary.

	Result of the FIXED(a,p,s) function
s not specified	Result as for s=0
p not specified	Result as for p=38
ABS(a)>10exp(p-s)	Special NULL value [Page 22]
a is the NULL value	NULL value [Page 22]
a is special NULL value	Special NULL value

FLOAT(a,s)

FLOAT(a,s) is an arithmetic function [Page 56] that outputs the figure a in a format of the data type FLOAT [Page 117](s). It is rounded to s places if necessary.

	Result of the FLOAT(a,s) function
a is NULL value	NULL value [Page 22]
a is special NULL value	Special NULL value [Page 22]

FLOOR(a)

FLOOR(a) is an arithmetic function [Page 56] that calculates the largest integer value that is less than or equal to the number a.

The result is a fixed point number with 0 decimal places.

An error message is output if it is not possible to represent the result of FLOOR(a) as a fixed point number.

	Result of the FLOOR(a) function
a is the NULL value	NULL value [Page 22]

a is special NULL value	Special NULL value [Page 22]

INDEX(a,b,p,s)

INDEX(a,b,p,s) is an arithmetic function [Page 56] that determines the position of the substring specified in b within the character string a.

The parameter p is optional. If p is specified (p>=1), it defines the start position for the search for the substring b. If p is not specified, the search is started at position 1.

The parameter s is optional. If s is specified, it determines the number of searches for the substring b. If s is not specified, the search is carried out for the first occurrence.

	Result of the INDEX(a,b,p,s) function
a, b character strings and b not less than s times substring of a	0
a character string and b empty character string	p
a,b,p or s is NULL value	NULL value [Page 22]
p or s is special NULL value [Page 22]	Error message

Model table: customer [Page 83]

The position of the character string ‘ar’ is to be determined in all customer surnames.

```
SELECT surname, INDEX(surname, 'ar') position_ar FROM customer
```

NAME	POSITION_AR
Porter	2
DATASOFT	0
Porter	3
Peters	0
Brown	0
Porter	5
Howe	0
Randolph	0
Peters	0
Brown	0
Jackson	0
Adams	0
Griffith	0
TOOLware	0
Brown	0

LENGTH(a)

LENGTH(a) is an arithmetic function [Page 56] that specifies the number of bytes that are required to represent the value a internally. The function can be applied to any data type.

	Result of the LENGTH(a) function
a is character string with length n	n The length is calculated without taking the following characters (code attribute ASCII, EBCDIC) or binary zeros (code attribute BYTE) into account.
a is NULL value	NULL value [Page 22]
a is special NULL value	Special NULL value [Page 22]

Model table: customer [Page 83]

The `customer` table is sorted according to the length of the surnames, with names with the same length sorted in alphabetical order.

```
SELECT surname, LENGTH(surname) length
FROM customer ORDER BY length, surname
```

NAME	LENGTH
DATASOFT	4
Porter	5
Brown	5
Brown	5
Howe	5
Griffith	5
Brown	5
Adams	5
Porter	6
Peters	6
Randolph	6
TOOLware	7
Peters	7
Jackson	7
Porter	7

LN(a)

LN(a) is an arithmetic function [Page 56] that calculates the natural logarithm of the number a.

	Result of the LN(a) function
a is NULL value	NULL value [Page 22]
a is special NULL value	Special NULL value [Page 22]

LOG(a,b)

LOG(a,b) is an arithmetic function [Page 56] that calculates the logarithm of the number b to base a.

	Result of the LOG(a,b) function
a or b is the NULL value	NULL value [Page 22]
b is special NULL value	Special NULL value [Page 22]

NOROUND(a)

NOROUND(a) is an arithmetic function [Page 56] that prevents the result of an UPDATE or INSERT statement from being rounded so that it matches the data type of the target column.

If the non-rounded number does not correspond to the data type of the target column, an error message is output.

	Result of the NOROUND(a) function
a is the NULL value	NULL value [Page 22]
a is special NULL value	Special NULL value [Page 22]

PI

PI is an arithmetic function [Page 56] that outputs the value π.

POWER(a,n)

POWER(a,n) is an arithmetic function [Page 56] that calculates the nth power of the number a.

An error message is output if n is not an integer.

	Result of the POWER(a,n) function
a or n is the NULL value	NULL value [Page 22]
a is special NULL value	Special NULL value [Page 22]

ROUND(a,n)

ROUND(a,n) is an arithmetic function [Page 56] with the following result (for the values a and n):

ROUND(a) – round decimal places up and down
ROUND(a,n) – round up and down to the nth place on the right of the decimal point
ROUND(a,-n) – round up and down to the nth place on the left of the decimal point

	Result of the ROUND(a,n) function
a>=0	TRUNC [Page 61](a+0.5*10E-n,n)
a<0	TRUNC(a-0.5*10E-n,n)
n not specified	Result as for n=0
n is not an integer	The integer component of n is used and the result is as with a>=0 or a<0
a is floating point number	Floating point number
a is fixed point number	Fixed point number

a is the NULL value	NULL value [Page 22]
a is special NULL value	Special NULL value [Page 22]

SIGN(a)

SIGN(a) is an arithmetic function [Page 56] that indicates the sign of the number a.

	Result of the SIGN(a) function
a<0	-1
a=0	0
a>0	1
a is the NULL value	NULL value [Page 22]
a is special NULL value	Special NULL value [Page 22]

SQRT(a)

SQRT(a) is an arithmetic function [Page 56] that calculates the square root of the number a.

	Result of the SQRT(a) function
a>0	Square root of a
a=0	0
a<0 or a is NULL value	NULL value [Page 22]
a is special NULL value	Special NULL value [Page 22]

TRUNC(a,n)

TRUNC(a,n) is an arithmetic function [Page 56] with the following result (for the values a and n):

TRUNC(a) – truncate the decimal places of a
TRUNC(a,n) – truncate the number a after n decimal places
TRUNC(a,-n) – set n places in the number a before the decimal point to 0

	Result of the TRUNC(a,n) function
n>0	Number a that is truncated n places after the decimal point
n=0	Integer component of a
n<0	Number a that is truncated s places in front of the decimal point
n not specified	As with n=0
n is not an integer	The integer component of n is used and the result is as with n>0, n=0, or n<0
a is floating point number	Floating point number [Page 23]
a is fixed point number	Fixed point number [Page 23]
a is the NULL value	NULL value [Page 22]
a is special NULL value	Special NULL value [Page 22]

Trigonometric function

A trigonometric function is a function [Page 55] that supplies a numeric value as a result.

Syntax

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

expression [Page 90]

All of the values (`expression`) in every trigonometric function identify an angle in radians. The only exception to this is the RADIANS function.

	Result of the trigonometric function
`<expression>` is NULL value	NULL value [Page 22]
`<expression>` is special NULL value	Special NULL value [Page 22]
`COS(a)`	Cosine of number a
`SIN(a)`	Sine of number a
`TAN(a)`	Tangent of number a
`COT(a)`	Cotangent of number a
`COSH(a)`	Hyperbolic cosine of number a
`SINH(a)`	Hyperbolic sine of number a
`TANH(a)`	Hyperbolic tangent of number a
`ACOS(a)`	Arc cosine of number a
`ASIN(a)`	Arc sine of number a
`ATAN(a)`	Arc tangent of number a
`ATAN2(a,b)`	Arc tangent of the value a/b, where -π<=a<=+π and -π<=b<=+π
`ASIN(a)`	Arc sine of number a
`RADIANS(a)`	Angle in radians of the number a
`DEGREES(a)`	Value in degrees of the number a

String function

A string function is a function [Page 55] that supplies an alphanumeric value as a result.

Syntax

```
<string_function> ::=
  <string_spec> || <string_spec>
| <string_spec> & <string_spec>
| SUBSTR    ( <string_spec>, <expression>[, <expression>] )
| LFILL     ( <string_spec>, <string_literal> [,<unsigned_integer> ] )
| RFILL     ( <string_spec>, <string_literal> [,<unsigned_integer> ] )
| LPAD      ( <string_spec>, <expression>, <string_literal>
[,<unsigned_integer> ] )
| RPAD      ( <string_spec>, <expression>, <string_literal>
[,<unsigned_integer> ] )
| TRIM      ( <string_spec>[, <string_spec> ] )
| LTRIM     ( <string_spec>[, <string_spec> ] )
| RTRIM     ( <string_spec>[, <string_spec> ] )
| EXPAND    ( <string_spec>, <unsigned_integer> )
| UPPER     ( <string_spec> )
| LOWER     ( <string_spec> )
| INITCAP   ( <string_spec> )
| REPLACE   ( <string_spec>, <string_spec>[, <string_spec> ] )
| TRANSLATE ( <string_spec>, <string_spec>, <string_spec> )
| MAPCHAR   ( <string_spec>[, <unsigned_integer> ] [, <mapchar_set_name> ]
)
| ALPHA     ( <string_spec>[, <unsigned_integer> ] )
| ASCII     ( <string_spec> )
| EBCDIC    ( <string_spec> )
| SOUNDEX   ( <string_spec> )
```

string_spec [Page 55], expression [Page 90], string_literal [Page 36], unsigned_integer [Page 38], mapchar_set_name [Page 46]

Concatenation (x||y and x&y) [Page 72], SUBSTR(x,a,b) [Page 69], LFILL(x,a,n) [Page 65], RFILL(x,a,n) [Page 68], LPAD(x,a,y,n) [Page 65], RPAD(x,a,y,n) [Page 68], TRIM(x,y) [Page 71], LTRIM(x,y) [Page 66], RTRIM(x,y) [Page 68], EXPAND(x,n) [Page 64], UPPER(x)/LOWER(x) [Page 72], INITCAP(x) [Page 64], REPLACE(x,y,z) [Page 67], TRANSLATE(x,y,z) [Page 70], MAPCHAR(x,n,i) [Page 66], ALPHA(x,n) [Page 63], ASCII(x)/EBCDIC(x) [Page 64], SOUNDEX(x) [Page 69]

ALPHA(x,n)

ALPHA(x,n) is a string function [Page 62] that enables a character x in ASCII or EBCDIC code (code tables [Page 25]) to be converted to a different one or two-character representation in the DEFAULTMAP (mapchar_set_name [Page 46]). ALPHA(x,n) is used to define the sort sequence.

The function ALPHA(x,n) uses the MAPCHAR(x,n,i) [Page 66] function internally (where i is the DEFAULTMAP) and also performs a conversion to uppercase letters (UPPER(x) [Page 72]).

The parameter n is optional and specifies the maximum length of the result.

	Result of the ALPHA(x,n) function
ALPHA(x,n)	UPPER(MAPCHAR(x,n,DEFAULTMAP))

The function ALPHA enables an appropriate sort, e.g. if "ü" is to be treated for as "UE" sorting purposes. The MAPCHAR SET with the name DEFAULTMAP is used.

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

ASCII/EBCDIC(x)

ASCII(x) or EBCDIC(x) is a string function [Page 62] that converts a character string [Page 22] x in ASCII code [Page 25] to EBCDIC code [Page 26] and vice versa.

	Result of the ASCII(x) or EBCDIC(x) function
Code attribute for x is ASCII or EBCDIC	ASCII(x) is a character string in ASCII notation
Code attribute for x is ASCII or EBCDIC	EBCDIC(x) is a character string in EBCDIC notation
x is NULL value	NULL value [Page 22]

The ASCII and EBCDIC functions are useful when a specific code is to be used for sorting or comparison purposes.

Output of a sorted result table in EBCDIC order:

```
SELECT EBCDIC(name) FROM...WHERE...ORDER BY 1
```

EXPAND(x,n)

EXPAND(x,n) is a string function [Page 62] that inserts as many blanks (code attribute ASCII, EBCDIC (code tables [Page 25])) or binary zeros (code attribute BYTE) to the end of a character string [Page 22] x as are needed to give the string the length specified by n.

	Result of the EXPAND(x,n) function
x is NULL value	NULL value [Page 22]

See also:

LFILL(x,a,n) [Page 65]

RFILL(x,a,n) [Page 68]

INITCAP(x)

INITCAP(x) is a string function [Page 62] that changes a character string [Page 22] in such a way that the first character of a word is an uppercase letter and the rest of the word consists of lowercase characters. Words are separated by one or more characters which are neither letters [Page 35] nor digits [Page 35].

	Result of the INITCAP(x) function
x is NULL value	NULL value [Page 22]

Model table: customer [Page 83]

Standardizing the notation for names

```
SELECT name, INITCAP (name) new_name
FROM customer WHERE firstname IS NULL
```

NAME	NAME_NEW

DATASOFT	Datasoft
TOOLware	Toolware

LFILL(x,a,n)

LFILL(x,a,n) is a string function [Page 62] that inserts the character string a at the start of the character string [Page 22] x as often as required until the character string has reached the length n. If the character string a cannot be completely inserted without exceeding the specified total length, only the part that is needed to reach the total length is inserted.

	Result of the LFILL(x,a,n) function
LFILL(x,a) x must identify a CHAR or VARCHAR column.	The column x is filled with the characters of a until its maximum length is reached.
x is NULL value	NULL value [Page 22]
a or n is the NULL value	Error message

Model table: customer [Page 83]

```
SELECT LFILL (firstname,' ',8) firstname, name, city
FROM customer
WHERE firstname IS NOT NULL AND city = 'Los Angeles'
```

FIRSTNAME	NAME	CITY
Martin	Porter	Los Angeles
Sally	Peters	Los Angeles
Joseph	Peters	Los Angeles
Susan	Brown	Los Angeles
Anthony	Jackson	Los Angeles
Thomas	Adams	Los Angeles

RFILL(x,a,n) [Page 68]

EXPAND(x,n) [Page 64]

LPAD(x,a,y,n)

LPAD(x,a,y,n) is a string function [Page 62] that inserts the character string y at the start of the character string [Page 22] x as often as specified by the parameter a. Leading and subsequent blanks in the character string x are truncated. The optional parameter n defines the maximum total length of the character string created.

The result of the parameter a must be a positive integer.

The optional parameter n must be greater than or equal to the total LENGTH [Page 59](x)+a*LENGTH(y).

	Result of the LPAD(x,a,y,n) function

LPAD(x,a,y) x must identify a CHAR or VARCHAR column.	The maximum length of the character string is the length of the character string x.
x or a is the NULL value	NULL value [Page 22]
a is the special NULL value [Page 22]	Error message

Model table: customer [Page 83]

Creating bar charts: LPAD inserts asterisks in front of the first parameter (in this case, a blank). This is done according to the number equal to account divided by 100.

```
SELECT name, account, LPAD(' ',TRUNC(account/100),'*',50) graph
FROM customer WHERE account > 0 ORDER BY account DESC
```

NAME	ACCOUNT	GRAPH
DATASOFT	4813.50	**********
TOOLware	3770.50	**********
Peters	650.00	******
Brown	440.00	****
Porter	100.00	*

See also:

RPAD(x,a,y,n) [Page 68]

LTRIM(x,y)

LTRIM(x,y) is a string function [Page 62] that removes all of the characters specified in the character string y from the start of the character string [Page 22] x. The result of LTRIM(x,y), therefore, starts with the first character that was not specified in y.

	Result of the LTRIM(x,y) function
LTRIM(x)	Only blanks (code attribute ASCII, EBCDIC (code tables [Page 25])) or binary zeros (code attribute BYTE) are removed from x.
x is NULL value	NULL value [Page 22]

See also:

TRIM(x,y) [Page 71]

RTRIM(x,y) [Page 68]

MAPCHAR(x,n,i)

MAPCHAR(x,n,i) is a string function [Page 62] that converts country-specific letters to a different format (e.g. German umlauts, French letters with a grave accent). These letters are located in the ASCII code [Page 25] and EBCDIC code [Page 26] at positions that can seldom be used for sorting purposes.

MAPCHAR(x,n,i) uses the MapChar set with the name i (mapchar set name [Page 46]) to convert the character string x. If you do not specify a MapChar set name, the MapChar set with the name DEFAULTMAP is used.

The parameter n is optional and specifies the maximum length of the result.

	Result of the MAPCHAR(x,n,i) function
MAPCHAR(x,i)	MAPCHAR(x,n,i), whereby n is the length of the character string x
MAPCHAR(x,i) x is CHAR or VARCHAR column	MAPCHAR(x,n,i), whereby n is the length of the column x
MAPCHAR(x)	MAPCHAR(x,DEFAULTMAP)
x is NULL value	NULL value [Page 22]

The function MAPCHAR enables data to be sorted correctly, e.g. if "ü" is to be treated for as "UE" sorting purposes. The MAPCHAR SET with the name DEFAULTMAP is used.

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

REPLACE(x,y,z)

REPLACE(x,y,z) is a string function [Page 62] that replaces the character string y in the character string [Page 22] x with the character string z.

	Result of the REPLACE(x,y,z) function
REPLACE(x,y)	The character string y in x is deleted
x is NULL value	NULL value [Page 22]
y is NULL value	x remains unchanged
z is NULL value	The character string y in x is deleted

Model table: hotel [Page 84]

The abbreviated street notation is to be written out in full for the sake of uniformity.

```
SELECT hno, zip, city, REPLACE (address, 'tr.', 'treet') address
FROM hotel WHERE address = '%tr'
```

HNO	ZIP	CITY	ADDRESS
40	21109	Los Angeles	155 Beechwood Street
50	20097	Los Angeles	1499 Grove Street
80	20251	Los Angeles	12 Barnard Street

See also:

TRANSLATE(x,y,z) [Page 70]

RFILL(x,a,n)

RFILL(x,a,n) is a string function [Page 62] that inserts the character string a at the end of the character string [Page 22] x as often as required until the character string has reached the length n. If the character string a cannot be completely inserted without exceeding the specified total length, only the part that is needed to reach the total length is inserted.

	Result of the RFILL(x,a,n) function
RFILL(x,a) x must identify a CHAR or VARCHAR column.	The column x is filled with the characters of a until its maximum length is reached.
x is NULL value	NULL value [Page 22]
a or n is the NULL value	Error message

See also:

LFILL(x,a,n) [Page 65]

EXPAND(x,n) [Page 64]

RPAD(x,a,y,n)

RPAD(x,a,y,n) is a string function [Page 62] that inserts the character string y at the end of the character string [Page 22] x as often as specified by the parameter a. Leading and subsequent blanks in the character string x are truncated. The optional parameter n defines the maximum total length of the character string created.

The result of the parameter a must be a positive integer.

The optional parameter n must be greater than or equal to the total LENGTH [Page 59](x)+a*LENGTH(y).

	Result of the RPAD(x,a,y,n) function
RPAD(x,a,y) x must identify a CHAR or VARCHAR column.	The maximum length of the character string is the length of the character string x.
x or a is the NULL value	NULL value [Page 22]
a is the special NULL value [Page 22]	Error message

See also:

LPAD(x,a,y,n) [Page 65]

RTRIM(x,y)

RTRIM(x,y) is a string function that removes the blanks (code attribute ASCII, EBCDIC (code tables [Page 25])) or binary zeros (code attribute BYTE) from the end of the character string [Page 22] x and then all of the characters specified in the character string y. The result of RTRIM(x,y), therefore, ends with the last character that was not specified in y.

	Result of the RTRIM(x,y) function

RTRIM(x)	Only blanks (code attribute ASCII, EBCDIC or binary zeros (code attribute BYTE) are removed from x.
x is NULL value	NULL value [Page 22]

See also:

TRIM(x,y) [Page 71]

LTRIM(x,y) [Page 66]

SOUNDEX(x)

SOUNDEX(x) is a string function [Page 62] that converts a character string [Page 22] x to a format that is generated by the SOUNDEX algorithm.

This representation can be used if the user does not know the exact spelling of the search term.

	Result of the SOUNDEX(x) function
SOUNDEX(x)	The SOUNDEX algorithm is used. The result is a value with the data type CHAR(4).
x is NULL value	NULL value [Page 22]

The SOUNDS predicate [Page 106] is often applied to a column x.

Since inversions cannot be used here, it is advisable for performance reasons to define an additional table column x1 with the data type CHAR(4), in which the result of SOUNDEX(x) is then inserted.

The requests should then refer to x1:
Instead of `x SOUNDS LIKE <string literal>`,
use `x1= SOUNDEX(<string literal>)`

SUBSTR(x,a,b)

SUBSTR(x,a,b) is a string function [Page 62] that outputs part of x (character string [Page 22] with length n).

	Result of the SUBSTR(x,a,b) function
SUBSTR(x,a,b)	Part of the character string x that starts at the a^{th} character and is b characters long.
SUBSTR(x,a)	SUBSTR(x,a,n-a+1) supplies all of the characters in the character string x from the a^{th} character to the last (n^{th}) character.
b is an unsigned integer [Page 38]	SUBSTR(x,a,b) b can also have a value that is greater than (n-a+1).
b is not an unsigned integer	SUBSTR(x,a,b) b must not be greater than (n-a+1).

b>(n-a+1)	SUBSTR(x,a) As many blanks (code attribute ASCII, 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.
x, a or b is the NULL value	NULL value [Page 22]

Model table: customer [Page 83]

The SUBSTR function is used to reduce the firstname to one letter, add a period and a blank, and then concatenate it with the name.

```
SELECT SUBSTR (firstname,1,1)&'. '&name name, city
FROM customer WHERE firstname IS NOT NULL
```

NAME	CITY
J. Porter	New York
?. DATASOFT	Los Angeles
P. Brown	Los Angeles
M. Jackson	Hollywood
G. Howe	New York
F. Miller	New York
R. Brown	Los Angeles
S. Murphy	Los Angeles
B. Smith	Los Angeles
A. Jones	Los Angeles
F. O'Brien	Los Angeles
P. Johnson	New York
E. Thomas	Los Angeles

TRANSLATE(x,y,z)

TRANSLATE(x,y,z) is a string function [Page 62] that replaces the i^{th} character of the character string [Page 22] y with the i^{th} character of the character string z in the character string x. The character strings y and z must have the same length.

	Result of the TRANSLATE(x,y,z) function
x is NULL value	NULL value [Page 22]
y is NULL value	x remains unchanged

Model table: customer [Page 83]

Each occurrence of the i^{th} letter in the first character string is replaced by the i^{th} letter in the second one.

```
SELECT name, TRANSLATE (name, 'ae', 'oi') name_new
FROM customer WHERE firstname IS NOT NULL AND city = 'Los Angeles'
```

NAME	NAME_NEW
Porter	Randolph
Peters	Smith
Peters	Pitirs
Brown	Bokir
Jackson	Jinkins
Adams	Adams

See also:

REPLACE(x,y,z) [Page 67]

TRIM(x,y)

TRIM(x,y) is a string function [Page 62] that removes all of the characters specified in the character string y from the start of the character string [Page 22] x. The result of TRIM(x,y), therefore, starts with the first character that was not specified in y.

TRIM(x,y) also removes the blanks (code attribute ASCII, EBCDIC (code tables [Page 25])) or binary zeros (code attribute BYTE) from the end of the character string and then all of the characters specified in the character string y. The result of TRIM(x,y), therefore, ends with the last character that was not specified in y.

	Result of the TRIM(x,y) function
TRIM(x)	Only blanks (code attribute ASCII, EBCDIC or binary zeros (code attribute BYTE) are removed from x.
x is NULL value	NULL value [Page 22]

Model table: customer [Page 83]

```
SELECT name, TRIM (CHR (account)) & ' DM' account
FROM customer WHERE account > 0.00
```

NAME	ACCOUNT
Porter	100.00 DM
DATASOFT	4,813.50 DM
Peters	650.00 DM
TOOLware	3,770.50 DM
Brown	440.00 DM

See also:

LTRIM(x,y) [Page 66]

RTRIM(x,y) [Page 68]

UPPER/LOWER(x)

UPPER(x) and LOWER(x) are string functions [Page 62] that convert a character string [Page 22] to uppercase/lowercase letters.

	Result of the UPPER(x) or LOWER(x) function
x is NULL value	NULL value [Page 22]

Model table: hotel [Page 84]

Refining searches by specifying uppercase/lowercase letters

```
SELECT * FROM hotel WHERE UPPER (name) = 'FLORA'
```

HNO	NAME	ZIP	CITY	ADDRESS
30	Regency	48153	Portland	477 17th Avenue

Concatenation

A concatenation x||y or x&y is a string function [Page 62] that supplies the following results for x (character string [Page 22] with the length n) and y (character string with the length m):

	Result of the concatenation
x||y or x&y	x and y are concatenated to a character string with the length n+m. If a character string originates from a column, its length is determined without any consideration of trailing blanks (code attribute [Page 24]. ASCII, EBCDIC, UNICODE) or binary zeros (code attribute BYTE).
x or y is the NULL value	NULL value [Page 22]

Columns with the same code attribute can be concatenated.

Columns with different code attributes (ASCII, EBCDIC and UNICODE) can be concatenated together as well as with date value [Page 23]s, time value [Page 23]s, and timestamp value [Page 24]s.

Model table: customer [Page 83]

```
SELECT name, city&', '&state&' '&chr(zip) address FROM customer
```

NAME	ADDRESS
Porter	New York, NY 10580
DATASOFT	Dallas, TX 75243
Porter	Los Angeles, CA 90018
Peters	Los Angeles, CA 90011
Brown	Hollywood, CA 90029
Porter	Washington, DC 20037
Howe	New York, NY 10019
Randolph	Chicago, IL 60601

Peters	Los Angeles, CA 90013
Brown	Los Angeles, CA 90008
Jackson	Los Angeles, CA 90005
Adams	Los Angeles, CA 90014
Griffith	New York, NY 10575
TOOLware	Los Angeles, CA 90002
Brown	Hollywood, CA 90029

Date function

A date function is a function [Page 55] that is applied to date or timestamp values or supplies a date or timestamp value as a result.

Syntax

```
<date_function> ::=
  ADDDATE    ( <date_or_timestamp_expression>, <expression> )
| SUBDATE    ( <date_or_timestamp_expression>, <expression> )
| DATEDIFF   ( <date_or_timestamp_expression>,
<date_or_timestamp_expression> )
| DAYOFWEEK  ( <date_or_timestamp_expression> )
| WEEKOFYEAR ( <date_or_timestamp_expression> )
| DAYOFMONTH ( <date_or_timestamp_expression> )
| DAYOFYEAR  ( <date_or_timestamp_expression> )
| MAKEDATE   ( <expression>, <expression> )
| DAYNAME    ( <date_or_timestamp_expression> )
| MONTHNAME  ( <date_or_timestamp_expression> )
```

date_or_timestamp_expression [Page 75], expression [Page 90]

ADDDATE(t,a)/SUBDATE(t,a) [Page 73], DATEDIFF(t,s) [Page 74], DAYOFWEEK(t), WEEKOFYEAR(t), DAYOFMONTH(t), DAYOFYEAR(t) [Page 75], MAKEDATE(a,b) [Page 75], DAYNAME(t), MONTHNAME(t) [Page 74]

Explanation

Although the Gregorian calendar was not introduced until 1582, it can also be applied to date functions that use dates prior to that year. This means that every year is assumed to have either 365 or 366 days.

A variety of date and time formats [Page 53] (ISO, USA, EUR, JIS, INTERNAL) are available for processing date and time values.

ADDDATE/SUBDATE(t,a)

ADDDATE(t,a) and SUBDATE(t,a) are date functions [Page 73] that calculate a date in the future or past.

t: date_or_timestamp_expression [Page 75]

a: numeric value that represents the number of days. Any decimal places in a are truncated if necessary.

	Result of the ADDDATE(t,a)/SUBDATE(t,a) function

Addition of a to t/ subtraction of a from t	Date value [Page 23] or time stamp value [Page 24]
t or a is NULL value	NULL value [Page 22]
a is special NULL value [Page 22]	Error message

Model table: reservation [Page 86]

Increasing a reservation date by two days

```
SELECT arrival, ADDDATE(arrival,2) arrival2, rno
FROM reservation WHERE rno = 130
```

ARRIVAL	ARRIVAL2	RNO
01.02.1999	03.02.1999	130

DATEDIFF(t,s)

DATEDIFF(t,s) is a date function [Page 73] that calculates the number of days between a start and end date.

t and s: date_or_timestamp_expression [Page 75]

	Result of DATEDIFF(t,s) function
Positive difference between t and s	Numeric value (number of days)
t or s are timestamp values	Only the dates [Page 23] in the timestamp value [Page 24] are used to calculate DATEDIFF(t,s).
t or s is NULL value	NULL value [Page 22]

Model table: reservation [Page 86]

```
SELECT arrival, departure, DATEDIFF(departure,arrival) difference,
rno
FROM reservation WHERE rno = 130
```

ARRIVAL	DEPARTURE	DIFFERENCE	RNO
01.02.1999	03.02.1999	2	130

DAYNAME/MONTHNAME(t)

DAYNAME(t) and MONTHNAME(t) are date functions [Page 73] that supply the weekday or month of the specified day.

t: date_or_timestamp_expression [Page 75]

	Result of the function
DAYNAME(t) or MONTHNAME(t)	Character string [Page 22] that supplies the name of a weekday (Sunday to Saturday) or month (January to December).

t is NULL value	NULL value [Page 22]

DAYOFWEEK/WEEKOFYEAR/DAYOFMONTH/DAYOFYEAR(t)

DAYOFWEEK(t)/WEEKOFYEAR(t)/DAYOFMONTH(t)/DAYOFYEAR(t) are date functions [Page 73] that calculate the following:

t: date_or_timestamp_expression [Page 75]

	Result of the function
DAYOFWEEK(t)	Numeric value between 1 and 7 (weekday) The first day is Monday, the second Tuesday, etc.
WEEKOFYEAR(t)	Numeric value between 1 and 53 (calendar week of the specified day)
DAYOFMONTH(t)	Numeric value between 1 and 31 (day of the month of the specified day)
DAYOFYEAR(t)	Numeric value between 1 and 366 (day of the year of the specified day)
t is NULL value	NULL value [Page 22]

MAKEDATE(a,b)

MAKEDATE(a,b) is a date function [Page 73] that supplies a date value [Page 23] from a year and day value.

The values a and b (expression [Page 90]) in MAKEDATE must supply numeric values.
The following restrictions also apply: a>=0 (a represents a year value)
b>0 (b represents a day value)
Decimal places in a and b are truncated if necessary.

	Result of the MAKEDATE(a,b) function
a or b is the NULL value	NULL value [Page 22]
a or b is the special NULL value [Page 22]	Error message

`MAKEDATE(1999,49)` in the INTERNAL date format [Page 53] outputs '19990218'

date_or_timestamp_expression

When used in a function, the `date_or_timestamp_expression` argument must supply a date value [Page 23], a timestamp value [Page 24], or an alphanumeric value that matches the current date or timestamp format [Page 53].

Time function

A time function is a function [Page 55] that is applied to time or timestamp values or supplies a time or timestamp value as a result.

Syntax

```
<time_function> ::=
  ADDTIME     ( <time_or_timestamp_expression>, <time_expression> )
| SUBTIME     ( <time_or_timestamp_expression>, <time_expression> )
| TIMEDIFF    ( <time_or_timestamp_expression>,
<time_or_timestamp_expression> )
| MAKETIME    ( <hours>, <minutes>, <seconds> )
```

time_or_timestamp_expression [Page 77], time_expression [Page 77], hours, minutes, seconds [Page 77]

ADDTIME/SUBTIME(t,a) [Page 76], TIMEDIFF(t,s) [Page 76], MAKETIME(h,m,s) [Page 76]

A variety of date and time formats [Page 53] (ISO, USA, EUR, JIS, INTERNAL) are available for processing date and time values.

ADDTIME/SUBTIME(t,a)

ADDTIME(t,a) and SUBTIME(t,a) are time functions [Page 76] that calculate a time/timestamp value in the past or future.

t: time_or_timestamp_expression [Page 77]
a: time_expression [Page 77]

	Result of the ADDTIME(t,a)/SUBTIME(t,a) function
Addition of a to t/ subtraction of a from t	Time value [Page 23] or time stamp value [Page 24]
SUBTIME(t,a) and t and a are time values	a must be less than t, otherwise an error is output
t or a is NULL value	NULL value [Page 22]

MAKETIME(h,m,s)

MAKETIME(h,m,s) is a time function [Page 76] that calculates a time value [Page 23] from the total number of hours, minutes, and seconds [Page 77].

	Result of the MAKETIME(h,m,s) function
h,m,s is NULL value	NULL value [Page 22]
h,m,s is special NULL value [Page 22]	Error message

TIMEDIFF(t,s)

TIMEDIFF(t,s) is a time function [Page 76] that calculates the time value between a start and end time.

t and s: time_or_timestamp_expression [Page 77]
Both arguments must have the same data type, i.e. they must be either a time value [Page 23] or a timestamp value [Page 24].

	Result of the TIMEDIFF(t,s) function
Positive difference between t and s	Time value
t or s are timestamp values for alphanumeric values that match the current timestamp format.	The dates [Page 23] contained in the timestamp value are used to calculate TIMEDIFF(t,s).
Difference of more than 9999 hours	Number of hours modulo 10000
t or s is NULL value	NULL value [Page 22]

hours/minutes/seconds

When used in a function, each of these arguments (`hours`, `minutes` or `seconds`) must be an integer greater than or equal to 0.

Decimal places are truncated.

Time expression

When used in a function, the argument `time_expression` must supply a time value [Page 23] or an alphanumeric value that is in the current time format [Page 53].

Time or timestamp expression

When used in a function, the argument `time_or_timestamp_expression` must supply a time value [Page 23], timestamp value [Page 24], or an alphanumeric value that matches the current time or timestamp format [Page 53].

Extraction function

An extraction function is a function [Page 55] that extracts parts of a date value [Page 23], time value [Page 23], or timestamp value [Page 24] or that calculates a date, time, or timestamp value.

Syntax

```
<extraction_function> ::=
  YEAR        ( <date_or_timestamp_expression> )
| MONTH       ( <date_or_timestamp_expression> )
| DAY         ( <date_or_timestamp_expression> )
| HOUR        ( <time_or_timestamp_expression> )
| MINUTE      ( <time_or_timestamp_expression> )
| SECOND      ( <time_or_timestamp_expression> )
| MICROSECOND ( <expression> )
| TIMESTAMP   ( <expression>[, <expression> ] )
| DATE        ( <expression> )
| TIME        ( <expression> )
```

date_or_timestamp_expression [Page 75], time_or_timestamp_expression [Page 77], expression [Page 90]

YEAR(t), MONTH(t), DAY(t) [Page 79], HOUR(t), MINUTE(t), SECOND(t) [Page 78], MICROSECOND(a) [Page 78], TIMESTAMP(a,b) [Page 79], DATE(a) [Page 78], TIME(a) [Page 79]

A variety of date and time formats [Page 53] (ISO, USA, EUR, JIS, INTERNAL) are available for processing date and time values.

DATE(a)

DATE(a) is a function (extraction [Page 77]) that calculates a date value [Page 23].

	Result of DATE(a) function
a is a date value or an alphanumeric value that matches the current date format	This date value
a is an alphanumeric value that does not match the current date format	Error message
a is a timestamp value [Page 24] or an alphanumeric value that matches the current timestamp format	The date value that is part of the timestamp
a is a fixed point or floating point number [Page 23]	Date value that is equal to the x^{th} day after December 31, 0000 (x=TRUNC(a) [Page 61])
a is NULL value	NULL value [Page 22]
a is special NULL value [Page 22]	Error message

HOUR/MINUTE/SECOND(t)

HOUR(t), MINUTE(t), and SECOND(t) are functions (extraction [Page 77]) that extract the hours, minutes, or seconds from the specified time or timestamp value.

t: time_or_timestamp_expression [Page 77]

	Result of the function
HOUR(t), MINUTE(t), or SECOND(t)	Numeric value (hours, minutes, and seconds)
t is NULL value	NULL value [Page 22]

MICROSECOND(a)

MICROSECOND(a) is a function (extraction [Page 77]) that extracts the microseconds from a.

The value a must be a timestamp value [Page 24] or supply an alphanumeric value that matches the current timestamp format.

	Result of MICROSECOND(a) function
MICROSECOND(a)	Numeric value (microseconds)

a is NULL value	NULL value [Page 22]

TIME(a)

TIME(a) is a function (extraction [Page 77]) that calculates a time value [Page 23].

	Result of TIME(a) function
a is a time value or an alphanumeric value that matches the current time format	This time value
a is an alphanumeric value that does not match the current time format	Error message
a is a timestamp value [Page 24] or an alphanumeric value that matches the current timestamp format	The time value that is part of the timestamp
a is NULL value	NULL value [Page 22]

TIMESTAMP(a,b)

TIMESTAMP(a,b) is a function (extraction [Page 77]) that calculates a timestamp value [Page 24] comprising a date value [Page 23], time value [Page 23], and 0 microseconds.

	Result of TIMESTAMP(a,b) function
TIMESTAMP(a)	a must be a timestamp value or an alphanumeric value that matches the current timestamp format The result is this timestamp value.
TIMESTAMP(a,b)	a must be a date value and b a time value (or alphanumeric value that matches the current format for date and time values). The result is a timestamp value calculated from a date value, time value, and 0 microseconds.
a or b is the NULL value	NULL value [Page 22]

YEAR/MONTH/DAY(t)

YEAR(t), MONTH(t), and DAY(t) are functions (extraction [Page 77]) that extract the year, month, and day from the specified date or timestamp value.

t: date_or_timestamp_expression [Page 75]

	Result of the function
YEAR(t), MONTH(t), or DAY(t)	Numeric value (year, month, day)
t is NULL value	NULL value [Page 22]

Special function

There are certain special function [Page 55]s that are not restricted to specific data types.

Syntax

```
<special_function> ::=
  VALUE    ( <expression>, <expression>,...)
| GREATEST ( <expression>, <expression>,...)
| LEAST    ( <expression>, <expression>,...)
| DECODE   ( <check_expression>, <search_and_result_spec>,...[,
<default_expression> ] )
```

expression [Page 90], check_expression, search_and_result_spec, default_expression [Page 80]

VALUE(x,y,...) [Page 81], GREATEST(x,y,...), LEAST(x,y,...) [Page 81], DECODE(x,y,...,z) [Page 80]

DECODE(x,y(i),...,z)

DECODE(x,y(i),...,z) is a special function [Page 80] that decodes expressions [Page 90] in accordance with their values.

x	check_expression	Expression for which a comparison is to be carried out with the value y(i)
y(i)	search_and_result_spec	<search_and_result_spec> ::= <search_expression>, <result_expression> (y(i)=u(i),v(i), i=1,...) Combination of the comparison value u(i) and the value v(i) that is to replace this comparison value
z	default_expression	Optional default value
u(i)	search_expression	Comparison value that is to be replaced by v(i) if it matches x
v(i)	result_expression	Value that is to be replace u(i)

Both x and u(i) must have identical data types. The data types of v(i) and z must be comparable. The data types of u(i) and v(i) do not have to be comparable.

DECODE compares the values of x with the values u(i) consecutively. If a match is found, the result of DECODE is the value v(i) in the combination y(i)=u(i),v(i).

A match is present if x and u(i) are NULL values. The comparison of the special NULL value with any other value never results in a match.

If a match is not found, DECODE supplies the result of z. If z is not specified, the NULL value is the result of DECODE.

Model table: room [Page 85]

The room type identifiers are to be replaced in the output by an identifier declared in the DECODE function.

```
SELECT hno, price, DECODE (roomtype,
'SINGLE', 1, 'DOUBLE', 2, 'SUITE', 3) room_code
FROM room
```

GREATEST/LEAST(x,y,...)

GREATEST(x,y,...) or LEAST(x,y) is a special function [Page 80] that calculates the maximum or minimum value of all arguments.

The functions can be applied to any data type. The data types of the individual arguments must be comparable.

	Result of the function
At least one argument is the NULL value	NULL value [Page 22]
At least one argument is the special NULL value	Special NULL value [Page 22]

VALUE(x,y,...)

VALUE(x,y,...) is a special function [Page 80] that can be used to replace NULL values with a non-NULL value.

The arguments of the VALUE function must be comparable. The arguments are evaluated one after the other in the specified order.

	Result of the VALUE(x,y,...) function
One of the arguments is a non-NULL value	The first non-NULL value that occurs
Each argument is a special NULL value [Page 21]	Special NULL value
Each argument is a NULL value	NULL value

Model table: customer [Page 83]

The title does not occur in the output list. The word `COMPANY` is to be output for companies in the `FIRSTNAME` column instead of a NULL value.

```
SELECT VALUE(firstname, 'COMPANY') firstname, name FROM customer
```

Conversion function

A conversion function is a function [Page 55] that converts a value from one data type to another.

Syntax

```
<conversion_function> ::=
  NUM    ( <expression> )
| CHR    ( <expression>[, <unsigned_integer> ] )
| HEX    ( <expression> )
| CHAR   ( <expression>[, <datetimeformat> ] )
```

expression [Page 90], unsigned_integer [Page 38], datetimeformat [Page 53]

NUM(a) [Page 83], CHR(a,n) [Page 82], HEX(a) [Page 82], CHAR(a,t) [Page 82]

CHAR(a,t)

CHAR(a,t) is a function (conversion function [Page 81]) that converts the date values [Page 23], time values [Page 23], or timestamp values [Page 24] to a character string with the format for date values, time values, or timestamp values [Page 53] specified in t.

	Result of the CHAR(a,t) function
CHAR(a)	The current date and time format is used for the value a.
a is the NULL value	NULL value [Page 22]

CHR(a,n)

CHR(a,n) is a function (conversion function [Page 81]) that converts numbers into character strings.

The CHR function can only be applied to numeric values, character strings [Page 22], and boolean values [Page 24].

	Result of the CHR(a,n) function
a is a numeric value	Character string that matches the CHAR representation of the numeric value a. The code attribute of the character string corresponds to the code type of the computer.
a is character string	Identical character string
a is a boolean value	T, if a=TRUE F, if a=FALSE
a is not a numeric value, a character string, or a boolean value	Error message
CHR(a,n), n>=1	Output with the length attribute n
CHR(a)	The length attribute n is calculated as a function of the data type and the length of a.
CHR(a) and a if of the data type FLOAT(p)	If p=1, then n=6 If p>1, then n=p+6
CHR(a) and a is of the data type FIXED(p,s)	If p=s, then n=p+s If s=0, then n=p+1
a is the NULL value	NULL value [Page 22]
a is the special NULL value [Page 22]	Error message

HEX(a)

HEX(a) is a function (conversion function [Page 81]) that converts the argument a to hexadecimal notation. The function can be applied to any data type.

	Result of the HEX(a) function
a is the NULL value	NULL value [Page 22]
a is the special NULL value [Page 22]	Error message

NUM(a)

NUM(a) is a function (conversion function [Page 81]) that converts the argument a to a numeric value.

NUM can be applied to character string [Page 22]s with the code attributes ASCII or EBCDIC (see code tables [Page 25]), date value [Page 23]s, time value [Page 23]s, time stamp value [Page 24]s, and to numeric and boolean values (BOOLEAN [Page 24]).

	Result of the NUM(a) function
a is a character string and can be interpreted as a numeric value	Corresponding numeric value
a is a numeric value	Identical numeric value (unchanged)
a is a boolean value	1, if a=TRUE 0, if a=FALSE
a is a character string that cannot be interpreted as a numeric value; a is a character string that does not have the code attribute ASCII or EBCDIC; a is neither a numeric nor a Boolean value.	Error message

a is a character string that can be interpreted as a numeric value outside the range
`-9.99999999999999999999999999999999999E+62,`
`9.99999999999999999999999999999999999E+62`
	Special NULL value [Page 22]

a is the NULL value	NULL value [Page 22]
a is the special NULL value	Special NULL value

Model tables

customer [Page 83]

hotel [Page 84]

room [Page 85]

reservation [Page 86]

customer

The **`customer`** table contains a list of customers with the following information:

CNO	TITLE	NAME	FIRSTNAME	ZIP	CITY	ACCOUNT
3000	Mrs	Porter	Jenny	80335	New York	100.00
3100	Comp	DATASOFT	?	50933	Dallas	4813.50

3200	Mr	Porter	Martin	10969	Los Angeles	0.00
3300	Mrs	Peters	Sally	14165	Los Angeles	0.00
3400	Mr	Brown	Peter	40233	Hollywood	0.00
3500	Mr	Porter	Michael	70596	New York	0.00
3600	Mr	Howe	George	81737	New York	-315.40
3700	Mr	Randolph	Frank	22525	Los Angeles	0.00
3800	Mr	Peters	Joseph	10787	Los Angeles	650.00
3900	Mrs	Brown	Susan	13599	Los Angeles	-4167.79
4000	Mr	Jackson	Anthony	10785	Los Angeles	0.00
4100	Mr	Adams	Thomas	13355	Los Angeles	-416.88
4200	Mr	Griffith	Mark	81739	New York	0.00
4300	Comp	TOOLware	?	13629	Los Angeles	3770.50
4400	Mrs	Brown	Rose	40233	Hollywood	440.00

SQL statement for creating the table structure

```
CREATE TABLE customer
(cno         FIXED(4) KEY CONSTRAINT cno BETWEEN 1 AND 9999,
 title       CHAR(5) CONSTRAINT title IN ('Mr','Mrs','Comp.'),
 surname     CHAR(7) NOT NULL,
 firstname   CHAR(7),
 zip         CHAR(5) CONSTRAINT zip like '(0-9)(0-9)(0-9)(0-9)(0-9)',
 city        CHAR(12) NOT NULL,
 account     FIXED(7,2) CONSTRAINT account BETWEEN -10000 AND 10000)
```

hotel

The **hotel** table contains a list of hotels with the following information:

HNO	NAME	ZIP	CITY	ADDRESS
10	Congress	89477	Detroit	155 Beechwood Str.
20	Los Angeles	86159	Cincinnati	1499 Grove Street
30	Regency	48153	Portland	477 17th Avenue
40	Eight Avenue	21109	Los Angeles	112 8th Avenue
50	Lake Michigan	20097	Los Angeles	354 OAK Terrace 43
60	Airport	60313	New Orleans	650 C Parkway
70	Empire State	80805	New York	65 Yellowstone Dr.
80	Midtown	20251	Los Angeles	12 Barnard Street
90	Long Beach	45193	Long Beach	200 Yellowstone Dr.
100	Dallas	50668	Dallas	1980 34th Str.
110	Atlantic	81245	New York	111 78th Street
120	Sunshine	12249	Los Angeles	35 Broadway 77
130	Star	40223	Hollywood	13 Beechwood Place 2

140	River Boat	70469	New York	788 MAIN STREET 14
150	Indian Horse	69117	Santa Clara	16 MAIN STREET

SQL statement for creating the table structure

```
CREATE TABLE hotel
(hno     FIXED(4) KEY CONSTRAINT hno BETWEEN 1 AND 9999,
 name    CHAR(10) NOT NULL,
 zip     CHAR(5) CONSTRAINT zip like '(0-9)(0-9)(0-9)(0-9)(0-9)',
 city     CHAR(12) NOT NULL,
 address CHAR(25) NOT NULL)
```

room

The **room** table contains a list of rooms with the following information:

HNO	ROOMTYPE	MAX_FREE	PRICE
10	double	45	200.00
10	single	20	135.00
20	double	13	100.00
20	single	10	70.00
30	double	15	80.00
30	single	12	45.00
40	double	35	140.00
40	single	20	85.00
50	double	230	180.00
50	single	50	105.00
50	suite	12	500.00
60	double	39	200.00
60	single	10	120.00
60	suite	20	500.00
70	double	11	180.00
70	single	4	115.00
80	double	19	150.00
80	single	15	90.00
80	suite	5	400.00
90	double	145	150.00
90	single	45	90.00
90	suite	60	300.00
100	double	24	100.00
100	single	11	60.00
110	double	10	130.00
110	single	2	70.00

120	double	78	140.00
120	single	34	80.00
120	suite	55	350.00
130	double	300	270.00
130	single	89	160.00
130	suite	100	700.00
140	double	9	200.00
140	single	10	125.00
140	suite	78	600.00
150	double	115	190.00
150	single	44	100.00
150	suite	6	450.00

The room table is linked to the hotel [Page 84] table via the hotel number.

SQL statement for creating the table structure

```
CREATE TABLE room
(hno      FIXED(4) KEY,
 roomtype  CHAR(6) CONSTRAINT roomtype IN ('single', 'double', 'suite'),
 max_free FIXED(3) CONSTRAINT max_free>=0,
 price    FIXED(6,2) CONSTRAINT price BETWEEN 0.00 AND 1000.00)
```

reservation

The **reservation** table contains a list of rooms with the following information:

RNO	CNO	HNO	ROOMTYPE	ARRIVAL	DEPARTURE
100	3000	80	single	13.11.1998	15.11.1998
110	3000	100	double	24.12.1998	06.01.1999
120	3200	50	suite	14.11.1998	18.11.1998
130	3900	110	single	01.02.1999	03.02.1999
140	4300	80	double	12.04.1998	30.04.1998
150	3600	70	double	14.03.1999	24.03.1999
160	4100	70	single	12.04.1998	15.04.1998
170	4400	150	suite	01.09.1998	03.09.1998
180	3100	120	double	23.12.1998	08.01.1999
190	4300	140	double	14.11.1998	17.11.1998

A logical link between the customer [Page 83], hotel [Page 84], and room [Page 85] tables is established via the `reservation` table.

SQL statement for creating the table structure

```
CREATE TABLE reservation
(rno     FIXED(4) KEY CONSTRAINT rno BETWEEN 1 AND 9999,
 cno     FIXED(4) CONSTRAINT cno BETWEEN 1 AND 9999,
 hno     FIXED(4) CONSTRAINT hno BETWEEN 1 AND 9999,
 roomtype CHAR(6) CONSTRAINT roomtype IN ('single', 'double', 'SUITE'),
```

```
arrival DATE NOT NULL,
departure DATE CONSTRAINT departure > arrival)
```

Set function (set_function_spec)

There is a series of functions that can be applied to a set of values (rows) as an argument and supply a result. These functions are referred to as set functions (`set function spec`).

Syntax

```
<set_function_spec> ::= COUNT (*) | <distinct_function> | <all_function>
```

COUNT (*) [Page 89], distinct_function [Page 87], all_function [Page 88]

Explanation

Set functions operate across groups of values but only return one value. The result comprises one row. If a set function is used in a statement, a similar function must also be applied to each of the other columns in the request. This, however, does not apply to columns that were grouped using GROUP BY. In this case, the value of the set function can be defined for each group.

The argument of a DISTINCT function or an ALL function is a result table [Page 29] or a group (the result table can be grouped using a GROUP condition).

With the exception of the COUNT(*) function, no NULL value [Page 22]s are included in the calculation.

No locks are set for certain set functions, irrespective of the isolation level (see CONNECT-statement [Page 204]) specified when the user connected to the database.

Model table: customer [Page 83]

```
SELECT SUM(account) sum_account, MIN(account) min_account,
FIXED (AVG(account),7,2) avg_account,
MAX(account) max_acount, COUNT(*) number
FROM customer WHERE city = 'Los Angeles'
```

SUM_ACCOUNT	MIN_ACCOUNT	AVG_ACCOUNT	MAX_ACCOUNT	NUMBER
-164.17	-4167.79	-23.45	3770.50	7

DISTINCT function

The DISTINCT function is a set function [Page 87] that removes duplicated values and all NULL value [Page 22]s.

Syntax

```
<distinct_function>::= <set_function_name> ( DISTINCT <expression> )
```

set_function_name [Page 89], expression [Page 90]

Explanation

The argument of a DISTINCT function is a set of values that is calculated as follows:

1. A result table [Page 29] or group (the result table can be grouped with a GROUP condition) is formed.
2. The expression is applied to each row in this result table or group.
   The expression must not contain a set function.

3. All of the NULL values and duplicated values are removed (DISTINCT). Special NULL value [Page 22]s are not removed and two special NULL values are considered identical.

The DISTINCT function is executed taking into account the relevant set function name for this set of values.

	Result of the DISTINCT function
Set of values is empty and the DISTINCT function is applied to the entire result table	The set functions AVG, MAX, MIN, STDDEV, SUM, VARIANCE supply the NULL value as their result. The set function COUNT supplies the value 0.
There is no group to which the DISTINCT function can be applied.	The result is an empty table.
The set of values contains at least one special NULL value.	Special NULL value

Model table: customer [Page 83]

In how many cities do the customers live?

```
SELECT COUNT(DISTINCT city) number_cities FROM customer
```

NUMBER_CITIES
6

ALL function

The ALL function is a set function [Page 87] that removes the NULL value [Page 22]s.

Syntax

```
<all_function>::= <set_function_name> ( [ALL] <expression> )
```

set_function_name [Page 89], expression [Page 90]

Explanation

The argument of an ALL function is a set of values that is calculated as follows:

1. A result table [Page 29] or group (the result table can be grouped with a GROUP condition) is formed.
2. The expression is applied to each row in this result table or group.
   The expression must not contain a set function.
3. All NULL values are removed. Special NULL value [Page 22]s are not removed and two special NULL values are considered identical.

The ALL function is executed taking into account the relevant set function name for the set of values.

The result of an ALL function is independent of whether the keyword ALL is specified or not.

	Result of the ALL function
The set of values is empty and the ALL function is applied to the entire result table	The set functions AVG, MAX, MIN, STDDEV, SUM, VARIANCE supply the NULL value as their result. The set function COUNT supplies the value 0.

There is no group to which the ALL function can be applied.	The result is an empty table.
The set of values contains at least one special NULL value.	Special NULL value

Set function name

Set function name is the name of a function that can be specified in a DISTINCT function [Page 87] and an ALL function [Page 88].

Syntax

```
<set_function_name> ::= COUNT | MAX | MIN | SUM | AVG | STDDEV | VARIANCE
```

COUNT [Page 89], MAX, MIN [Page 90], SUM [Page 90], AVG [Page 89], STDDEV [Page 90], VARIANCE [Page 90]

AVG

AVG is a set function [Page 87].

The result of AVG is the arithmetical mean of the values of the argument.

AVG can only be applied to numeric values. The result has the data type FLOAT [Page 117](38).

Model table: customer [Page 83]

How many corporate customers are taken into account? What is the average value of their account balance?

```
SELECT COUNT(*) number, FIXED (AVG(account),7,2) avg_account
FROM customer WHERE firstname IS NULL
```

NUMBER	AVG_ACCOUNT
2	4292.00

COUNT

COUNT is a set function [Page 87].

`COUNT(*)` supplies the total number of values (rows in a result table or group).

`COUNT(DISTINCT <expression>)` supplies the total number of different values (number of values in the argument of the DISTINCT function [Page 87]).

`COUNT(ALL <expression>)` supplies the number of values that differ from the NULL value (number of values in the argument of the ALL function [Page 88])

The result has the data type FIXED [Page 117](10).

Model table: customer [Page 83]

How many customers are there?

```
SELECT COUNT (*) number FROM customer
```

NUMBER
15

MAX/MIN

MAX/MIN is a set function [Page 87].

The result of MAX is the largest value of the argument.

The result of MIN is the smallest value of the argument.

STDDEV

STDDEV is a set function [Page 87].

The result of STDDEV is the standard deviation of the values of the argument.

STDDEV can only be applied to numeric values. The result has the data type FLOAT [Page 117](38).

SUM

SUM is a set function [Page 87].

The result of SUM is the sum of the values of the argument.

SUM can only be applied to numeric values. The result has the data type FLOAT [Page 117](38).

VARIANCE

VARIANCE is a set function [Page 87].

The result of VARIANCE is the variance of the values of the argument.

VARIANCE can only be applied to numeric values. The result has the data type FLOAT [Page 117](38).

Expression

An expression specifies a value that is generated, if required, by applying arithmetic operators to values.

A distinction is made between the following arithmetic operators:

- Additive operators
  + Addition
  - Subtraction
- Multiplicative operators
  * Multiplication
  / Division
  DIV integer division
  MOD remainder after integer division

Syntax

```
<expression> ::= <term> | <expression> + <term> | <expression> - <term>
<expression_list> ::= (<expression>,...)
```

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

factor [Page 92]

Explanation

The arithmetic operators can only be applied to numeric data types.

	Result of an expression
`expression`	Value of any data type [Page 21]
`factor` supplies a NULL value	NULL value [Page 22]
`factor` supplies a special NULL value	Special NULL value [Page 22]
`expression` leads to a division by 0	Special NULL value
`expression` leads to an overflow of the internal temporary result	Special NULL value

If no parentheses are used, the operators have the following precedence:

1. The sign [Page 37] has a higher precedence than the additive and multiplicative operators.
2. The multiplicative operators have a higher precedence than the additive operators.
3. The multiplicative operators have different priorities.
4. The additive operators have different priorities.
5. Operators with the same precedence are evaluated from left to right.

Operands are fixed point numbers

Operand1 (a)	Operand2 (b)	Result
Fixed point number [Page 23] (p precision s number of decimal places	Fixed point number (p' precision s' number of decimal places)	Fixed point number (p" precision s" number of decimal places) or floating point number

The data type of the result depends on the operation as well as on the precision and number of decimal places of the operands.

Note that the data type of a column determines its name, and not the precision and number of decimal places in the current value.

Operands are fixed point numbers, operands are +, - , * or /

The result of addition, subtraction, and multiplication is generated from a temporary result which can have more than 38 valid digits. If the temporary result has no more than 38 valid digits, the final result is equal to the temporary result. Otherwise, a result is generated as a floating point number with 38 places. Decimal places are truncated if necessary.

Condition	Operator	Result
max(p-s,p'-s')	+, -	Fixed point number p"=max(p-s,p'-s')+max(s,s')+1 s"=max(s,s')
(p+p')<=38	*	Fixed point number p"=p+p' s"=s+s'
(p-s+s')<=38	/	Fixed point number p"=38

		s"=38-(p-s+s') Special NULL value, if b=0

Operands are integers, operators are DIV, MOD

Condition	Operator	Result
ABS [Page 56](a)<1E38 and ABS(b)<1E38 and b<>0	DIV	TRUNC [Page 61](a/b)
b=0	DIV	Special NULL value
ABS(a)>=1E38 and b<>0 or ABS(b)>=1E38	DIV	Error message
ABS(a)<1E38 and ABS(b)<1E38 and b<>0	MOD	a-b*(a DIV b)
b=0	MOD	a
ABS(a)>=1E38 and b<>0 or ABS(b)>=1E38	MOD	Error message

An operand is a floating point number

If an operand is a floating point number [Page 23], the result of the arithmetic operation is a floating point number.

factor

`factor` specifies how the values are determined that are to be linked in an expression [Page 90] by means of arithmetic operators.

Syntax

```
<factor> ::= [<sign>] <value_spec> | [<sign>] <column_spec>
| [<sign>] <function_spec> | [<sign>] <set_function_spec> | <expression>
```

sign [Page 37], value_spec [Page 52], column_spec [Page 51], function_spec [Page 55], set_function_spec [Page 87], expression [Page 90]

Predicate

A predicate is specified in a WHERE condition [Page 188] in a statement which is "true", "false", or "unknown". The result is generated by applying the predicate to a specific row in a result table (see result table name [Page 45]) or to a group of rows in a table that was formed by the GROUP clause [Page 188].

Syntax

```
<predicate> ::=
  <between_predicate> | <bool_predicate> | <comparison_predicate>
| <default_predicate> | <exists_predicate> | <in_predicate>
| <join_predicate> | <like_predicate> | <null_predicate>
| <quantified_predicate> | <rowno_predicate> | <sounds_predicate>
```

between_predicate [Page 94], bool_predicate [Page 95], comparison_predicate [Page 95], default_predicate [Page 97], exists_predicate [Page 97], in_predicate [Page 98], join_predicate [Page

99], like_predicate [Page 101], null_predicate [Page 104], quantified_predicate [Page 104], rowno_predicate [Page 106], sounds_predicate [Page 106]

Explanation

- Columns in a table with the same code attribute are comparable.
- Columns with different code attributes ASCII and EBCDIC (see code tables [Page 25]) can be compared.
- Columns with the code attributes ASCII and EBCDIC can be compared with date values [Page 23], time values [Page 23], or timestamp values [Page 24].
- LONG columns [Page 23] can only be used in the NULL predicate.

Model table: customer [Page 83]

Selection without a condition:

```
SELECT city, name, firstname FROM customer
```

CITY	NAME	FIRSTNAME
New York	Porter	Jenny
Dallas	DATASOFT	?
Los Angeles	Porter	Martin
Los Angeles	Peters	Sally
Hollywood	Brown	Peter
New York	Porter	Michael
New York	Howe	George
Los Angeles	Randolph	Frank
Los Angeles	Peters	Joseph
Los Angeles	Brown	Susan
Los Angeles	Jackson	Anthony
Los Angeles	Adams	Thomas
New York	Griffith	Mark
Los Angeles	TOOLware	?
Hollywood	Brown	Rose

Selection with restricting condition:

```
SELECT city, name, firstname FROM customer
WHERE city = 'Los Angeles'
```

CITY	NAME	FIRSTNAME
Los Angeles	Porter	Martin
Los Angeles	Peters	Sally
Los Angeles	Peters	Joseph
Los Angeles	Brown	Susan
Los Angeles	Jackson	Anthony
Los Angeles	Adams	Thomas

Los Angeles	TOOLware	?

BETWEEN predicate

The BETWEEN predicate [Page 92] checks whether a value is within a specified range.

Syntax

```
<between_predicate> ::= <expression> [NOT] BETWEEN <expression> AND
<expression>
```

expression [Page 90]

Explanation

Let x, y, and z be the results of the first, second, and third expression. The values x,y,z must be comparable.

	Result of the specified predicate
x BETWEEN y AND z	x>=y AND x<=z
x NOT BETWEEN y AND z	NOT(x BETWEEN y AND z)
x, y, or z are NULL value [Page 22]s	x [NOT] BETWEEN y AND z is undefined

Model table: customer [Page 83]

Finding customers with a credit balance between –420 and 0:

```
SELECT title, name, city, account FROM customer
WHERE account BETWEEN -420 AND 0
```

TITLE	NAME	CITY	ACCOUNT
Mr	Porter	Los Angeles	0.00
Mrs	Peters	Los Angeles	0.00
Mr	Brown	Hollywood	0.00
Mr	Porter	New York	0.00
Mr	Howe	New York	-315.40
Mr	Randolph	Los Angeles	0.00
Mr	Jackson	Los Angeles	0.00
Mr	Adams	Los Angeles	-416.88
Mr	Griffith	New York	0.00

You want to list the customers who have either a credit balance or a significant debit balance:

```
SELECT title, name, city, account FROM customer
WHERE account NOT BETWEEN -10 AND 0
```

TITLE	NAME	CITY	ACCOUNT
Mrs	Porter	New York	100.00
Comp	DATASOFT	Dallas	4813.50

Mr	Howe	New York	-315.40
Mr	Peters	Los Angeles	650.00
Mrs	Brown	Los Angeles	-4167.79
Mr	Adams	Los Angeles	-416.88
Comp	TOOLware	Los Angeles	3770.50
Mrs	Brown	Hollywood	440.00

Boolean predicate (bool_predicate)

Boolean values (BOOLEAN [Page 24]) are compared in a boolean predicate [Page 92].

Syntax

```
<bool_predicate> ::= <column_spec> [ IS [NOT] <TRUE | FALSE>]
```

column spec [Page 51]

Explanation

If only one column specification (`column_spec`) is specified, the syntax is identical to `<column_spec> IS TRUE`.

The `column_spec` must always denote a column with the data type BOOLEAN.

The following rules apply to the result of a boolean predicate:

Column value	IS TRUE	IS NOT TRUE	IS FALSE	IS NOT FALSE
false	false	true	true	false
undefined	undefined	undefined	undefined	undefined
true	true	false	false	true

Comparison predicate

A comparison predicate [Page 92] specifies a comparison between two values or lists of values.

Syntax

```
<comparison_predicate> ::= <expression> <comp_op> <expression>
| <expression> <comp_op> <subquery>
| <expression_list> <equal_or_not> (<expression_list>)
| <expression_list> <equal_or_not> <subquery>
```

expression [Page 90], expression_list [Page 90], subquery [Page 189]

The following operators are available for comparing two values:
<, >, <>, !=, =, <=, >= (comp_op [Page 97])

Value lists can only be compared with the = and <> operators (equal_or_not [Page 97]).

Explanation

The subquery must supply a result table (see result table name [Page 45]) that contains the same number of columns as the number of values on the left-hand side of the operator. The result table may contain no more than one row.

The list of values specified to the right of the `equal_or_not` operator (`expression_list`) must contain the same number of values as specified in the value list in front of the `equal_or_not` operator.

The JOIN predicate [Page 99] is a special case.

Comparing two values

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 one another.
- Numbers are compared to one another according to their algebraic values.
- Character strings are compared character by character.
  Any blanks (code attribute [Page 24] ASCII, EBCDIC, UNICODE) or binary zeros (code attribute BYTE) at the end of one or both of the character strings are removed.
  If the character strings have the different code attributes ASCII and EBCDIC, one of the two strings is converted implicitly so that they both have the same code attribute.
  If the character strings have the different code attributes ASCII or EBCDIC and UNICODE, the character string with the code attribute ASCII or EBCDIC is implicitly converted into a character string with the code attribute UNICODE.
  Two character strings are identical if they have the same characters in all positions.
  The relationship between two character strings that are not identical is defined by the first character that differs in a comparison from left to right. This comparison is performed in accordance with the code attribute selected for this column (ASCII, EBCDIC, UNICODE, or BYTE).
- If x or y are NULL value [Page 22]s, or if the result of the subquery is empty, (x `<comp_op>` y) is not defined.

Comparing two value lists

If a value list (`expression_list`) is specified on the left of the comparison operator `equal_or_not`, x is the value list that comprises the results of the values x1, x2, ..., xn in this 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 values y_1, y_2, ..., y_n.

- A value x_m must be comparable with the associated value y_m.
- x=y is true if for xm=ym for all m=1, ..., n.
- x<>y is true if at least xm<>ym for at least one m.
- (x `<equal_or_not>` y) is undefined (not known) 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 undefined.
- If one xm or one ym is a NULL value, or if the result of the subquery is empty, (x `<equal_or_not>` y) is undefined.

Model table: customer [Page 83]

Which customers are customers?

```
SELECT title, name FROM customer
WHERE title = 'Comp'
```

TITLE	NAME
Comp	DATASOFT
Comp	TOOLware

Comparison operators (comp_op)

Operators for comparing values (comp_op)

Syntax

```
<comp_op> ::= < | > | <> | != | = | <= | >=
| ¬= | ¬< | ¬> (for machines with EBCDIC code [Page 26])
| ~= | ~< | ~> (for machines with ASCII code [Page 25])
```

Comparison operators (equal_or_not)

Operators for comparing value lists (equal_or_not)

Syntax

```
<equal_or_not> ::= <> |=
| ¬= (for machines with EBCDIC code [Page 26])
| ~= (for machines with ASCII code [Page 25])
```

DEFAULT predicate

By specifying a DEFAULT predicate [Page 92], you can check whether a column contains the default value defined for this column.

Syntax

```
<default_predicate> ::= <column_spec> <comp_op> DEFAULT
```

column_spec [Page 51], comp_op [Page 97]

Explanation

A DEFAULT specification [Page 120] must be made for the specified column. This can be done in the following SQL statements:

- CREATE TABLE statement [Page 111]
- ALTER TABLE statement [Page 127]

If the column contains the NULL value [Page 22], <column_spec> <comp_op> DEFAULT is undefined.

The rules for comparing values or value lists, as defined under comparison predicate [Page 95], apply here.

EXISTS predicate

The EXISTS predicate [Page 92] checks whether a result table (see result table name [Page 45]) contains at least one row.

Syntax

```
<exists_predicate> ::= EXISTS <subquery>
```

subquery [Page 189]

Explanation

The truth content of an EXISTS predicate is either true or false.

The subquery generates a result table. If this result table contains at least one row, `EXISTS <subquery>` is true.

Model table customer [Page 83], reservation [Page 86]

Only select customers that have one or more reservations:

```
SELECT * FROM customer WHERE EXISTS
(SELECT * FROM reservation WHERE customer.cno = reservation.cno)
```

IN predicate

The IN predicate [Page 92] checks whether a value or a value list is contained in a specified set of values or value lists.

Syntax

```
<in_predicate> ::=
  <expression> [NOT] IN <subquery>
| <expression> [NOT] IN <expression_list>
| <expression_list> [NOT] IN <subquery>
| <expression_list> [NOT] IN (<expression_list>,...)
```

expression [Page 90], expression_list [Page 90], subquery [Page 189]

Explanation

The subquery must supply a result table (see result table name [Page 45]) that contains the same number of columns as the number of values specified by the expression on the left-hand side of the IN operator.

Each value list specified on the right-hand side of the IN operator must contain the same number of values as specified in the value list on the left-hand side of the IN operator.

- `x [NOT] IN S`, whereby x `<expression>` and S `<subquery>` or `<expression_list>`
  The value x and the values in S must be comparable.
- `x [NOT] IN S`, whereby x `<expression_list>` with the values x1, x2, ..., xn and S `<subquery>` (set of value lists s) or `(<expression_list>,...)` (Range of values lists s) with the value lists s: s1, s2, ..., sn
  A value xm must be comparable with all values sm.
  x=s is true if xm=sm, m=1,...,n
  x=s is false if there is at least one m for which xm=sm is false
  x=s is undefined if there is no m for which xm=sm is false and there is at least one m for which xm=sm is undefined.

The entry '------' in the list below means that no statement can be made if only the result of the comparison with one s is known.

	Result of the function x IN S
x=s is true for at least one s	true
x=s is true for all s	true
S contains NULL values and x=s is true for the remaining s	true
S is empty	false
x=s is false for at least one s	------
x=s is false for all s	false

S contains NULL values and x=s is false for the remaining s	undefined
x=s is not true for any s and is undefined for at least one value s	undefined

`x NOT IN S` has the same result as `NOT(x IN S)`

Model table: customer [Page 83]

Choosing all customers who are natural persons (not companies):

```
SELECT title, firstname, name, city FROM customer
WHERE title IN ('Mr','Mrs')
```

TITLE	FIRSTNAME	NAME	CITY
Mrs	Jenny	Porter	New York
Mr	Martin	Porter	Los Angeles
Mrs	Sally	Peters	Los Angeles
Mr	Peter	Brown	Hollywood
Mr	Michael	Porter	New York
Mr	George	Howe	New York
Mr	Frank	Randolph	Los Angeles
Mr	Joseph	Peters	Los Angeles
Mrs	Susan	Brown	Los Angeles
Mr	Anthony	Jackson	Los Angeles
Mr	Thomas	Adams	Los Angeles
Mr	Mark	Griffith	New York
Mrs	Rose	Brown	Hollywood

JOIN predicate

A JOIN predicate [Page 92] specifies a JOIN. A JOIN predicate can be specified with or without one or with two OUTER JOIN indicators.

Syntax

```
<join_predicate> ::=
<expression> [<outer_join_indicator>] <comp_op> <expression>
[<outer_join_indicator>]

<outer_join_indicator> ::= (+)
```

expression [Page 90], comp_op [Page 97]

Explanation

Each `expression` must contain a column specification [Page 51]. A column specification must exist for the first and second expression so that both specifications refer to different table names or reference names.

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 one another.

The rules outlined under comparison predicate [Page 95] apply here.

If at least one OUTER JOIN indicator is specified in a JOIN predicate of a search condition [Page 107], the corresponding table expression must be based on exactly two tables, or the following has to apply:

- OUTER JOIN indicators are only specified for one of the tables in the FROM clause [Page 185].
- All of the JOIN predicates in this table to just one other table contain the OUTER JOIN indicator.
- All other JOIN predicates contain no OUTER JOIN indicators.

If a JOIN requires more than two tables for the QUERY specification [Page 182] and if one of the rules above cannot be observed, a QUERY expression [Page 179] can also be used in the FROM clause.

Only those rows from the table that have a counterpart of the comparison operator in the JOIN predicate specified in the table are transferred to the result table.

The OUTER JOIN indicator must be specified on the side of the comparison operator where the other table is specified if each row in a table is to appear at least once in the result table.
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 formed from the columns in the other table.
Since the OUTER JOIN indicator can be specified on both sides of the comparison operator if the table expression [Page 185] is based on just two tables, it can be ensured that each line in both tables appears at least once in the result table.

The JOIN predicate is a special case of the comparison predicate [Page 95]. The number of JOIN predicates in a search condition is limited to 128.

JOIN predicate

Model tables customer [Page 83], reservation [Page 86]

Is there a reservation for the customer 'Porter'? If so, for what date?

```
SELECT reservation.rno, customer.name, reservation.arrival,
departure
FROM customer, reservation
WHERE customer.name = 'Porter' AND customer.cno = reservation.cno
```

RNO	NAME	ARRIVAL	DEPARTURE
100	Porter	13.11.1998	15.11.1998
110	Porter	24.12.1998	06.01.1999

Specifying an OUTER JOIN indicator

Model tables hotel [Page 84], reservation [Page 86]

List all the hotels in Chicago for which a reservation exists and those for which a reservation does not exist. Missing reservation numbers are assigned a NULL value.

```
SELECT hotel.hno, hotel.name, reservation.rno
FROM hotel, reservation
WHERE hotel.city = 'Chicago' AND hotel.hno = reservation.hno (+)
```

HNO	NAME	RNO
40	Eight Avenue	?
50	Lake Michigan	120
80	Midtown	100
80	Midtown	140

LIKE Predicate

A LIKE predicate [Page 92] is used to search for character strings [Page 22] that have a particular pattern. This pattern can be a certain character string or any sequence of characters (whose length may or may not be known).

Syntax

```
<like_predicate> ::= <expression> [NOT] LIKE <like_expression> [ESCAPE
<expression>]

<like_expression> ::= <expression> | '<pattern_element>...'
```

expression [Page 90], pattern_element [Page 102]

Explanation

The expression in the `like_expression` must supply an alphanumeric value or a date or time value.

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

	Result of x LIKE y
x or y are NULL value [Page 22]s	x LIKE y is undefined
x and y are non-NULL values	x LIKE y is either true or false
x can be split into substrings with the result that: A substring of x is a sequence of 0,1, or more contiguous characters, and each character of x belongs to exactly one substring. The number of substrings of x and y is identical. If the nth pattern element of y is a set of characters and the nth substring of x is a single character that is contained in the set of characters.	x LIKE y is true
x can be split into substrings with the result that: A substring of x is a sequence of 0,1, or more contiguous characters, and each character of x belongs to exactly one substring. The number of substrings of x and y is identical. If the nth pattern element of y is a sequence of characters and the nth substring of x is a sequence of 0 or more characters.	x LIKE y is true

If ESCAPE is specified, the corresponding expression in the LIKE predicate must supply an alphanumeric value that consists of just one character. If this escape character is contained in the LIKE expression, the following character is regarded as an independent character.

An escape character must be used if a search is to be performed for an <underscore>, '?', '%' or '*', or the hexadecimal value X'1E' or X'1F'.

Search for any character string with a minimum length of 1: `LIKE '%_'`
Search for a character string in which a fixed number of characters is known: `LIKE'_c_'`
Search for a character string with any number of characters, whereby the character string must contain an <underscore>: `LIKE '%:_%'ESCAPE':'`

Model table: customer [Page 83]

Customers whose name ends with 'FT':

```
SELECT name, city FROM customer
WHERE name LIKE '%FT'
```

NAME	CITY
DATASOFT	Dallas

Finding all customers whose names consist of six letters and begin with 'P':

```
SELECT name, city FROM customer
WHERE name LIKE 'P?????'
```

NAME	CITY
Porter	Los Angeles
Peters	Los Angeles

Finding all customers whose names have any lengths and begin with 'M':

```
SELECT name, city FROM customer
WHERE name LIKE 'M%'
```

NAME	CITY
Martin	Los Angeles
Masterstone	Los Angeles
Moore	New York
Melson	Los Angeles

Finding customers whose name contains an 'o' after the first letter:

```
SELECT name, city FROM customer
WHERE name LIKE '_%o%'
```

NAME	CITY
Porter	New York
Randolph	Los Angeles
Brown	Los Angeles
Jackson	Los Angeles

Pattern element

Element for specifying a comparison pattern (for a LIKE predicate [Page 101]). A comparison can be carried out with a string of characters or a set of characters.

Syntax

```
<pattern_element> ::= <match_string> | <match_set>
```

match_string [Page 103], match_set [Page 103]

Match string

If a match string is specified, this position in the search pattern can be replaced by any number of characters.

Syntax

```
<match_string> ::= % | * | X'1F'
```

Explanation

A (LIKE predicate [Page 101]) is used to search for character strings [Page 22] that have a certain pattern. `Match strings` can be used to specify the pattern (pattern_element [Page 102]).

Model table: customer [Page 83]

Finding all customers whose names have any lengths and begin with 'M':

```
SELECT name, city FROM customer
WHERE name LIKE 'M%'
```

NAME	CITY
Porter	Los Angeles
Peters	Los Angeles
Porter	New York
Jackson	Los Angeles

Match set

If a match set is specified, this position in the search pattern can be replaced by the exact number of characters specified in the match set.

Syntax

```
<match_set> ::= <underscore> | ? | X'1E' | <match_char>
| ([< ∧ (only for code type EBCDIC) | ~ (only for code type ASCII)| ¬ >]<match
class>...)

<match_char>     ::= any character [Page 35] except %, *, X'1F', underscore, ?, X'1E', (.
<match_class>    ::= <match_range> | <match_element>

<match_range>    ::= <match_element>-<match_element> (character range)

<match_element>  ::= any character except )
```

underscore [Page 41]

Explanation

A LIKE predicate [Page 101] is used to search for character strings [Page 22] that have a certain pattern. Match sets can be used to specify the pattern (pattern_element [Page 102]).

- `<underscore> | ? | X'1E'` : this position in the pattern can be replaced by any character.
- `match_char` : this position in the pattern can be replaced by the specified character itself.
- A sequence of character classes (`match_class`) can be negated by prefixing it with ∧ or ~ or ¬. You cannot negate each character class individually.

Model table: customer [Page 83]

Finding all customers whose names consist of six letters and begin with 'P':

```
SELECT name, city FROM customer
WHERE name LIKE 'P?????'
```

NAME	CITY
Porter	Los Angeles
Peters	Los Angeles

NULL predicate

By specifying a NULL predicate [Page 92], you can test whether the value is a NULL value [Page 22].

Syntax

```
<null_predicate> ::= <expression> IS [NOT] NULL
```

expression [Page 90]

Explanation

The truth content of a NULL predicate is either true or false.

	Result of the function x IS NULL
x is NULL value	true
x is a special NULL value [Page 22]	false

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

Quantified predicate

By specifying a quantity predicate [Page 92], you can compare a value or list of values to a set of values or value lists.

Syntax

```
<quantified_predicate> ::=
  <expression> <comp_op> <quantifier> <expression_list>
| <expression> <comp_op> <quantifier> <subquery>
| <expression_list> <equal_or_not> <quantifier> (<expression_list>,...)
| <expression_list> <equal_or_not> <quantifier> <subquery>
```

subquery [Page 189], expression_list [Page 90]

The following operators are available for comparing values:
<, >, <>, !=, =, <=, >= (comp_op [Page 97])

Value lists can only be compared with the = and <> operators (equal_or_not [Page 97]).

The quantified predicate can be qualified with ALL, SOME, or ANY (quantifier [Page 106]).

Explanation

The subquery must supply a result table (see result table name [Page 45]) that contains the same number of columns as the number of values specified by the expression or expression list on the left-hand side of the operator.

Each list of values specified to the right of the `equal_or_not` operator (`expression_list`) must contain the same number of values as specified in the value list in front of the `equal_or_not` operator.

- Let x be the result of the first expression and S the result of the subquery or sequence of values. S is a set of values s. The value x and the values in S must be comparable with each other.
- If a value list (`expression_list`) is specified on the left of the comparison operator `equal_or_not`, then let x be the value list comprising the results of the values x_1, x_2, ..., x_n of this value list. Let S be 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 values s_1, s_2, ..., s_n. A value xm must be comparable with all values sm.
  x=s is true if xm=sm, m=1,...,n
  x<>s is true if there is at least one m for which xm<>sm
  x `<equal or not>` s is undefined 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 undefined.
  If one xm or one ym is a NULL value, or if the result of the subquery is empty, x `<equal_or_not>` y is undefined.

The entry '------' in the list below means that no statement can be made if only the result of the comparison with one s is known.

x <compare> <quantifier> S, whereby compare ::= comp_op | equal_or_not

	quantifier ::= ALL	**quantifier ::= ANY | SOME**
S is empty	true	false
`x <compare> S` is true for at least one s from S	------	true
`x <compare> S` is true for all s from S	true	true
`x <compare> S` is not false for any value from S and is undefined for at least one value s	undefined	
S contains NULL values and `x <compare> S` is true for all other s	undefined	true
`x <compare> S` is false for at least one value s from S	false	------
`x <compare> S` is false for all s from S	false	false
`x <compare> S` is not true for any value s from S and is undefined for at least one value s		undefined
S contains NULL values and `x <compare> S` is false for all other s	false	undefined

Model table: hotel [Page 84]

List of hotels that have the same name as other cities in the base table.

```
SELECT name, city FROM hotel
WHERE name = ANY (SELECT city FROM hotel)
```

The subquery `SELECT city FROM hotel` determines the list of city names that are compared with the hotel names.

Quantifier

The quantified predicate [Page 104] can be qualified with ALL, SOME, or ANY.

Syntax

```
<quantifier> ::= ALL | SOME | ANY
```

ROWNO predicate

The ROWNO predicate [Page 92] restricts the number of lines in a result table (see result table name [Page 45]).

Syntax

```
<rowno_predicate> ::= ROWNO < <unsigned_integer>
| ROWNO < <parameter_spec>
| ROWNO <= <unsigned_integer>
| ROWNO <= <parameter_spec>
```

unsigned_integer [Page 38], parameter_spec [Page 51]

Explanation

A ROWNO predicate may only be used in a WHERE clause [Page 188] that belongs to a QUERY statement. The ROWNO predicate can be used like any other predicate [Page 92] in the WHERE clause if the following restrictions are observed:

- The ROWNO predicate must be linked to the other predicates by a logic AND
- The ROWNO predicate must not be negated
- The ROWNO predicate may not be used more than once in the WHERE clause

You can specify the maximum number of lines in the result table using an unsigned integer or a parameter specification. The specified value must allow the result table to contain at least one row. If more lines are found, they are simply ignored and do not lead to an error message.

If a ROWNO predicate and an ORDER clause [Page 190] are specified, only the first n result lines are searched and sorted. The result usually differs from that which would have been obtained if a ROWNO predicate had not been used and if the first n result rows had been considered.

If a ROWNO predicate and a set function [Page 87] are specified, the set function is only applied to the number of lines restricted by the ROWNO predicate.

SOUNDS predicate

A SOUNDS predicate [Page 92] is used to perform a phonetic comparison.

Syntax

```
<sound_predicate> ::= <expression> [NOT] SOUNDS [LIKE] <expression>
```

expression [Page 90]

Explanation

Specifying LIKE in the SOUNDS predicate has no effect.

The values in the expressions must be alphanumeric (code attribute ASCII, EBCDIC, see code tables [Page 25]).

A phonetic comparison between values is carried out according to the SOUNDEX algorithm. First, all vowels and some consonants are eliminated, then all consonants which are similar in sound are mapped to each other.

x [NOT] SOUNDS [LIKE] y

	Result of the predicate
x or y is the NULL value [Page 22]	`x SOUNDS y` is undefined
x and y are non-NULL values	`x SOUNDS y` is true or false
x and y are phonetically identical	`x SOUNDS y` is true

`x NOT SOUNDS y` has the same result as `NOT (x SOUNDS y)`.

See also:

SOUNDEX(x) [Page 69] string function

Search Condition (search_condition)

A `search_condition` links statements that can be true, false, or undefined. Rows in a table may be found that fulfill several conditions that are linked with AND or OR.

Syntax

```
<search_condition> ::= <boolean_term> | <search_condition> OR
<boolean_term>
```

```
<boolean_term> ::= <boolean_factor> | <boolean_term> AND <boolean_factor>
```

boolean factor [Page 108]: determine the boolean values (BOOLEAN [Page 24]) to be linked or their negation (`NOT`).

Explanation

Predicates in a WHERE clause [Page 188] are applied to the specified row or a group of rows in a table formed with the GROUP clause [Page 188]. The results are linked using the specified Boolean operators (AND, OR, NOT).

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 with the same precedence are evaluated from left to right.

NOT

x	**NOT(x)**
true	false
false	true
undefined	undefined

x AND y

x **y**	**false**	**undefined**	**true**

false	false	false	false
undefined	false	undefined	undefined
true	false	undefined	true

x OR y

x y	**false**	**undefined**	**true**
false	false	undefined	true
undefined	undefined	undefined	true
true	true	true	true

Model table: customer [Page 83]

Customers who live in New York or have a credit balance:

```
SELECT firstname, name, city, account FROM customer
WHERE city = 'New York' OR account > 0
```

FIRSTNAME	NAME	CITY	ACCOUNT
Jenny	Porter	New York	100.00
?	DATASOFT	Dallas	4813.50
George	Howe	New York	-315.40
Joseph	Peters	Los Angeles	650.00
Mark	Griffith	New York	0.00
?	TOOLware	Los Angeles	3770.50
Brown	Rose	Hollywood	440.00

Customers who live in New York and have a credit balance:

```
SELECT firstname, name, city, account FROM customer
WHERE city = 'New York' AND account > 0
```

FIRSTNAME	NAME	CITY	ACCOUNT
Jenny	Porter	New York	100.00

Boolean factor

Specifies how the Boolean values are determined that are to be linked in a search condition [Page 107] by AND or OR.

Syntax

```
<boolean_factor> ::= [NOT] <predicate> | [NOT] (<search_condition>)
```

predicate [Page 92], search_condition [Page 107]

SQL statement: overview

All SQL statements can be embedded in programming languages. Further information is available in the precompiler documentation. All SQL statements, with the exception of NEXT STAMP, can be specified interactively.

Comments [Page 110] can be specified for every SQL statement.

SQL statements for data definition [Page 110]

CREATE TABLE statement	DROP TABLE statement	ALTER TABLE statement RENAME TABLE statement RENAME COLUMN statement EXISTS TABLE statement
CREATE DOMAIN statement	DROP DOMAIN statement	
CREATE SEQUENCE statement	DROP SEQUENCE statement	
CREATE SYNONYM statement	DROP SYNONYM statement	RENAME SYNONYM statement
CREATE VIEW statement	DROP VIEW statement	RENAME VIEW statement
CREATE INDEX statement	DROP INDEX statement	ALTER INDEX statement RENAME INDEX statement
COMMENT ON statement		
CREATE TRIGGER statement	DROP TRIGGER statement	
CREATE DBPROC statement	DROP DBPROC statement	

SQL statements for authorization [Page 150]

CREATE USER statement	DROP USER statement	ALTER USER statement RENAME USER statement GRANT USER statement
CREATE USERGROUP statement	DROP USERGROUP statement	ALTER USERGROUP statement RENAME USERGROUP statement GRANT USERGROUP statement
CREATE ROLE statement	DROP ROLE statement	
ALTER PASSWORD statement	GRANT statement	REVOKE statement

SQL statements for data manipulation [Page 163]

INSERT statement	UPDATE statement	DELETE statement
NEXT STAMP statement	CALL statement	

SQL statements for data query [Page 174]

QUERY statement	SINGLE SELECT statement	EXPLAIN statement
SELECT DIRECT statement: searched	SELECT DIRECT statement: positioned	
SELECT ORDERED statement: searched	SELECT ORDERED statement: positioned	
OPEN CURSOR statement	FETCH statement	CLOSE statement

SQL statements for transaction [Page 202] management

CONNECT statement	SET statement	
COMMIT statement	ROLLBACK statement	SUBTRANS statement
LOCK statement	UNLOCK statement	RELEASE statement

SQL statements for statistics [Page 225] management

UPDATE STATISTICS statement	MONITOR statement	

Comment (sql_comment)

A comment (`sql_comment`) can be included for every SQL statement [Page 109].

Syntax

```
<sql_comment> ::= /*<comment text>*/
```

Explanation

You can enter any commend text.

```
CREATE TABLE person (cno FIXED(4), first name CHAR(7), last name
CHAR(7), account FIXED(7,2)) /*create table person*/
```

Data definition

The following sections contain an introduction to the data definition language (DDL) used by the database system.

SQL statements for data definition

CREATE TABLE statement [Page 111]	DROP TABLE statement [Page 127]	ALTER TABLE statement [Page 127] RENAME TABLE statement [Page 133] RENAME COLUMN statement [Page 133] EXISTS TABLE statement

		[Page 134]
CREATE DOMAIN statement [Page 134]	DROP DOMAIN statement [Page 134]	
CREATE SEQUENCE statement [Page 135]	DROP SEQUENCE statement [Page 136]	
CREATE SYNONYM statement [Page 136]	DROP SYNONYM statement [Page 136]	RENAME SYNONYM statement [Page 137]
CREATE VIEW statement [Page 137]	DROP VIEW statement [Page 141]	RENAME VIEW statement [Page 141]
CREATE INDEX statement [Page 141]	DROP INDEX statement [Page 142]	ALTER INDEX statement [Page 143] RENAME INDEX statement [Page 143]
COMMENT ON statement [Page 143]		
CREATE TRIGGER statement [Page 149]	DROP TRIGGER statement [Page 150]	
CREATE DBPROC statement [Page 145]	DROP DBPROC statement [Page 148]	

CREATE TABLE statement

A CREATE TABLE statement defines a base table (see Table [Page 29]).

Syntax

```
<create_table_statement> ::=
  CREATE TABLE <table_name>
(<column_definition>[,<table_description_element>,...])
  [IGNORE ROLLBACK] [<sample_definition>]
| CREATE TABLE <table_name> [(<table_description_element>,...)]
  [IGNORE ROLLBACK] [<sample_definition>] AS <query_expression>
[<duplicates_clause> ]
| CREATE TABLE <table_name> LIKE <table_name> [IGNORE ROLLBACK]

<table_description_element> ::= <column_definition> |
<constraint_definition> | <referential_constraint_definition> |
<key_definition> | <unique_definition>
```

table_name [Page 50], sample_definition [Page 113], query_expression [Page 179], duplicates_clause [Page 166], column_definition [Page 114], constraint_definition [Page 122], referential_constraint_definition [Page 123], key_definition [Page 126], unique_definition [Page 126]

SQL statement for creating a table called `person`:

```
CREATE TABLE person (cno FIXED(4), firstname CHAR(7), name
CHAR(7), account FIXED(7,2))
```

This CREATE TABLE statement comprises the keywords CREATE TABLE followed by the table name and (in parentheses) a list of column names, separated by commas. You can also define other criteria, such as a primary key, or referential integrity conditions.

Explanation

Executing a CREATE TABLE statement causes data that describes the table (or base table) to be stored in the catalog. This data is called metadata.

A CREATE TABLE statement can contain a maximum of 1024 column definitions. If a table is defined without a key, the database system creates a key column implicitly. In this case, up to 1023 additional columns can be defined.

A CREATE TABLE statement cannot contain more than one key definition.

The table name must not be identical with the name of an existing table of the current user.

The current user becomes the owner [Page 44] of the new table. In other words, he or she obtains the INSERT, UPDATE, DELETE, and SELECT privileges for this table. If the table is not a temporary table, the owner is also granted the INDEX, REFERENCES, and ALTER privileges.

Owner of a table

- **The table owner must be specified in front of the table name:** temporary tables are a special type of table. They only exist during a user session and are deleted with their entire contents afterwards. Temporary tables are identified by the owner TEMP in front of the table name.
  If a table name has an owner other than TEMP, the owner must be identical to the name of the current user and the user must have the status DBA or RESOURCE (see Users and Usergroups [Page 30]).
  The owner of the table is not specified: the result is the same as if the current user were the owner.

QUERY statement

- If a **QUERY expression is not specified**, the CREATE TABLE statement must contain at least one column definition.
- If a **query expression** is specified, a base table is defined with the same structure as the result table defined by the QUERY expression.
  If column definitions are specified, the column definition may only consist of a column name [Page 49] and the number of column definitions must be equal to the number of columns in the result table generated by the QUERY expression.
  The data_type [Page 114] of the i^{th} column in the base table is identical to that of the i^{th} column in the result table generated by the QUERY expression.
  The result table must not contain LONG columns [Page 23].
  If no column definitions are specified, the column names of the result table are used.
  The rows of the result table are implicitly inserted in the generated base table. The DUPLICATES clause [Page 166] can be used to determine how key collisions are handled.
  The QUERY expression is subject to certain restrictions that also apply to the INSERT statement [Page 164].

LIKE <table_name>

If `LIKE <table_name>` is specified, an empty base table is created which, from the point of view of the current user, has the same structure as the source table, that is, it has all the columns with the same column names and definitions as the source table. This view does not necessarily have to be identical to the actual structure of the source table, since the user may not know all the columns because of privilege limitations.

The specified table [Page 29] must be either a base table, a view table, or a synonym [Page 30]. The user must have at least one privilege for this table.

The current user is the owner of the base table.

If all the key columns of the table specified after LIKE are contained in the base table, they form the key columns in this table. Otherwise, the database system implicitly inserts a key column SYSKEY CHAR(8) BYTE which then represents the key for the base table.

DEFAULT specification [Page 120]s or CONSTRAINT definitions [Page 122] for columns that are copied to the base table also apply to the new base table.

IGNORE ROLLBACK

IGNORE ROLLBACK is optional and can only be specified for temporary tables. Temporary tables with this characteristic are not affected by the transaction mechanism; i.e., changes affecting these tables are not reversed by rolling back a transaction.

SQL statements for changing table properties

Adding, deleting columns, changing data types, changing the CONSTRAINT definition

ALTER TABLE statement [Page 127]

Renaming columns

RENAME COLUMN statement [Page 133]

Renaming tables

RENAME TABLE statement [Page 133]

SAMPLE definition

A SAMPLE definition defines the number of rows in a table that are to be used when statistics are updated.

Syntax

```
<sample_definition> ::= SAMPLE <unsigned_integer> ROWS
| SAMPLE <unsigned_integer> PERCENT
```

unsigned_integer [Page 38]

Explanation

The database system manages statistics for each base table. These statistics are used to determine the best strategy for executing an SQL statement. The statistics are stored in the catalog by the UPDATE STATISTICS statement [Page 226].

If a SAMPLE definition is specified in an UPDATE STATISTICS statement, it specifies the number of rows in the table that are to be used to calculate the statistics.

If a SAMPLE definition is not specified in an UPDATE STATISTICS statement and if it is not mandatory that all of the rows in the table be used to calculate the statistics, the database system uses the appropriate SAMPLE definition of the CREATE TABLE or ALTER TABLE statement.

The number of rows for which the UPDATE STATISTICS statement is to be executed can be defined by specifying a numeric or percentage value.

- If a SAMPLE definition is specified as a `PERCENT`, the unsigned integer must be between 1 and 100.
- If a SAMPLE definition is not defined, the database system uses the value 20000 ROWS.

SQL statements in which the SAMPLE definition can be used

CREATE TABLE statement [Page 111]

ALTER TABLE statement [Page 127]

UPDATE STATISTICS statement [Page 226]

Column definition

A column definition defines a column [Page 29] in a table. The name and data type of each column are defined by the column name and data type. The column names must be unique within a base table.

Syntax

```
<column_definition> ::= <column_name> <data_type> [<column_attributes>]
| <column_name> <domain_name> [<column_attributes>]
```

column_name [Page 49], data_type [Page 114], column_attributes [Page 119], domain_name [Page 44]

Explanation

If the [PRIMARY] KEY column attribute is specified, the CREATE TABLE statement [Page 111] must not contain a key definition [Page 126].

A column definition may only consist of a column name if a QUERY expression is used in the CREATE TABLE statement.

If a **column name and domain name** (the name of a value range) are specified, the domain name must identify an existing domain. The data type and the length of the domain are assigned to the specified column. If the domain has a constraint_definition [Page 122], the effect is the same as if the corresponding CONSTRAINT definition were specified in the column attribute of the column definition.

Columns, which are part of the key, or for which NOT NULL was defined, are called **NOT NULL columns**. A NULL value [Page 22] cannot be inserted in these columns.

- **Mandatory columns**: NOT NULL columns for which a DEFAULT specification [Page 120] has not been declared as a column attribute are called mandatory columns. Whenever rows are inserted, values must be specified for these columns.
- **Optional columns**: columns that are not mandatory are referred to as optional columns. A value does not have to be specified when a row is inserted in these columns. If a DEFAULT specification exists for the column, the default value is entered in the column. If there is no DEFAULT specification, a NULL value is entered in the column.

If an index [Page 30] is generated for an individual optional column, it does not contain the rows in which the NULL value is specified for this column. With certain queries, therefore, the most effective search strategy cannot be selected via this index. NOT NULL, therefore, should 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 specification should be considered, because its value is used instead of the NULL value. Rows with the DEFAULT value are contained in an index.

Memory requirements of a column value as a function of the data type [Page 118]

The memory requirements of all columns in a table must not exceed 8088 bytes.

Data type

As well as specifying the column name when you define columns [Page 114], you can also specify data types.

Syntax

```
<data_type> ::=
  CHAR[ACTER] [(<unsigned_integer>)] [ASCII | BYTE | EBCDIC | UNICODE]
| VARCHAR [(<unsigned_integer>)] [ASCII | BYTE | EBCDIC | UNICODE]
| LONG [VARCHAR] [ASCII | BYTE | EBCDIC | UNICODE]
| BOOLEAN
| FIXED (<unsigned_integer> [,<unsigned_integer>])
| FLOAT (<unsigned_integer>)
| INT[EGER] | SMALLINT
| DATE | TIME | TIMESTAMP
```

unsigned_integer [Page 38]

CHAR[ACTER] [Page 115], VARCHAR [Page 116], LONG[VARCHAR] [Page 116], BOOLEAN [Page 116], FIXED [Page 117], FLOAT [Page 117], INT[EGER] [Page 117], SMALLINT [Page 117], DATE [Page 118], TIME [Page 118], TIMESTAMP [Page 118]

Explanation

For the following character strings [Page 22], a code attribute [Page 24] can be entered as part of a column definition (`column_definition`), if required: CHAR[ACTER], VARCHAR, LONG[VARCHAR]

In addition to the data types defined above, the following data types are permitted in a column definition and are mapped as follows to the data types below:

Data Type	Is Mapped To
DEC[IMAL](p,s)	FIXED(p,s)
DEC[IMAL](p)	FIXED(p)
DEC[IMAL]	FIXED(5)
BINARY(p)	FIXED(p)
FLOAT	FLOAT(16)
FLOAT(39..64)	FLOAT(38)
DOUBLE PRECISION	FLOAT(38)
REAL(p)	FLOAT(p)
REAL	FLOAT(16)
LONG VARCHAR	LONG
SERIAL	FIXED(10) DEFAULT SERIAL
SERIAL(p)	FIXED(10) DEFAULT SERIAL(p)

See also:

Data type [Page 21]

Memory requirements of a column value as a function of the data type [Page 118]

CHAR[ACTER]

Definition

An alphanumerical column is defined. Specification of length attribute n is optional. If no other length attribute is specified, then n=1.

CHAR[ACTER] [(n)]: 0<n<=8000
CHAR[ACTER] [(n)] UNICODE: 0<n<=4000

The database system determines, in accordance with n, whether the values in the column are stored with a fixed or variable length. If you want to store the values in a variable length regardless of the value of n, you must enter a value for VARCHAR [Page 116].

Use

Specification of data type CHAR[ACTER] in column definition [Page 114], with a code attribute [Page 24], if required.

Integration

Data type (data_type) [Page 114]

VARCHAR

Definition

An alphanumerical column is defined. Specification of length attribute n is optional. If no other length attribute is specified, then n=1.

VARCHAR [(n)]: 0<n<=8000
VARCHAR [(n)] UNICODE: 0<n<=4000

Use VARCHAR (n) if the values in the column are to be stored with a variable length, irrespective of n.

Use

Specification of data type VARCHAR in column definition [Page 114], with a code attribute [Page 24], if required.

Integration

Data type (data_type) [Page 114]

LONG[VARCHAR]

Definition

An alphanumeric column is defined with any length (not for temporary tables).

LONG[VARCHAR]: A maximum of 2 GB of characters can be written in a LONG column.
LONG[VARCHAR] UNICODE: A maximum of 2 GB bytes can be written in a LONG column.

You can only give LONG columns [Page 23] NOT NULL or a DEFAULT specification [Page 120] as a column attribute [Page 119].

Use

Specification of data type LONG in column definition [Page 114], with a code attribute [Page 24], if required.

LONG columns can be used in the following SQL statements:
INSERT statement [Page 164], UPDATE statement [Page 169], NULL predicate [Page 104], and in selected columns [Page 183].

Integration

Data type (data_type) [Page 114]

BOOLEAN

Definition

BOOLEAN: The column is given the data type BOOLEAN [Page 24].

Use

Specifying the data type BOOLEAN when defining columns [Page 114]

Integration

Data type (data_type) [Page 114]

FIXED

Definition

FIXED (p,s): A column with a fixed point number [Page 23] with precision p and with s number of decimal places (0<p<=38, s<=p). If no s is specified, it is assumed that the decimal places are 0.

Use

Specifying the data type FIXED when defining columns [Page 114]

Integration

Data type (data_type) [Page 114]

FLOAT

Definition

FLOAT (p): A column with a floating point number [Page 23] with precision p (0<p<=38).

Use

Specifying the data type FLOAT when defining columns [Page 114]

Integration

Data type (data_type) [Page 114]

INT[EGER]

Definition

INT[EGERER]: This data type is the same as FIXED [Page 117](10.0). Its permitted values are between -2147483648 and 2147483647.

Use

Specifying the data type INT[EGER] when defining columns [Page 114]

Integration

Data type (data_type) [Page 114]

SMALLINT

Definition

SMALLINT: This data type is the same as FIXED [Page 117](5.0). Its permitted values are between -32768 and 32767.

Use

Specifying the data type SMALLINT when defining columns [Page 114]

Integration

Data type (data_type) [Page 114]

DATE

Definition

DATE: An alphanumeric column in which you can store date values [Page 23].

Use

Specifying the data type DATE when defining columns [Page 114]

Integration

Data type (data_type) [Page 114]

TIME

Definition

TIME: An alphanumeric column in which you can store time values [Page 23].

Use

Specifying the data type TIME when defining columns [Page 114]

Integration

Data type (data_type) [Page 114]

TIMESTAMP

Definition

TIMESTAMP: An alphanumeric column in which you can store time stamp values [Page 24].

Use

Specifying the data type TIMESTAMP when defining columns [Page 114]

Integration

Data type (data_type) [Page 114]

Memory requirements of a column value per data types

Data type [Page 114]

Column definition [Page 114]

Data Type	Memory Requirements of a Column Value in Bytes for This Data Type
FIXED [Page 117](p,s)	(p+1) DIV 2 + 2
FLOAT [Page 117](p)	(p+1) DIV 2 + 2
BOOLEAN [Page 116]	2
DATE [Page 118]	9
TIME [Page 118]	9
TIMESTAMP [Page 118]	21
LONG [Page 116]	9

CHAR [Page 115](n); n<=30	n+1
CHAR(n); 30<n<=254; key column	n+1
CHAR(n); 30<n<=254; not key column	n+2
CHAR(n); 254<n	n+3
CHAR(n) UNICODE; n<=15	2*n+1
CHAR(n) UNICODE; 15<n<=127; key column	2*n+1
CHAR(n) UNICODE; 15<n<=127; not key column	2*n+2
CHAR(n) UNICODE; 127<n	2*n+3
VARCHAR [Page 116](n); 30<n<=254; key column	n+1
VARCHAR(n); 30<n<=254; not key column	n+2
VARCHAR(n); 254<n	n+3
VARCHAR(n) UNICODE; 15<n<=127; key column	2*n+1
VARCHAR(n) UNICODE; 15<n<=127; not key column;	2*n+2
VARCHAR(n) UNICODE; 127<n	2*n+3

Column attributes

A column_definition [Page 114] can contain the column name and column attributes.

Syntax

```
<column_attributes> ::= [<key_or_not_null_spec>] [<default_spec>] [UNIQUE]
[<constraint_definition>]
[REFERENCES <referenced_table> [(<referenced_column>)] [<delete_rule>]]

<key_or_not_null_spec> ::= [PRIMARY] KEY | NOT NULL [WITH DEFAULT]
```

default_spec [Page 120], constraint_definition [Page 122], delete_rule [Page 124]

- A CONSTRAINT definition [Page 122] defines a condition that must be fulfilled by all the column values in the columns defined by the column definition [Page 114].
- `[REFERENCES <referenced_table> [(<referenced_column>)] [<delete_rule>]` has the same effect as specifying the referential CONSTRAINT definition [Page 123]
  `FOREIGN KEY [<referential_constraint_name>] (<referencing_column>)`
  `REFERENCES <referenced_table> [(<referenced_column>,...)]`
  `[<delete_rule>]`

`referenced_table` `referenced_column`	referenced table referenced column

Explanation

The [PRIMARY] KEY and UNIQUE column attributes must not be used together in a column definition.

If the [PRIMARY] KEY column attribute is specified, the CREATE TABLE statement [Page 111] must not contain a key definition [Page 126].

LONG data type: you may only specify NOT NULL or a DEFAULT specification [Page 120] as a column attribute for LONG [Page 116] columns.

UNIQUE

The UNIQUE column attribute determines the uniqueness of column values (see also CREATE INDEX statement [Page 141]).

KEY

If the KEY column attribute is specified, this column is part of the key of a table and is called the key column. All key columns must be the first columns specified for a table. The order of the key columns is relevant for the SELECT ORDERED statement. The database system ensures that the key values in a table are unique. The sum of the internal lengths of the key columns must not exceed 1024 bytes. The number of key columns in a table must be less than 512. To improve performance, the key should start with key columns which can assume many different values and which are to be used frequently in conditions with the "=" operator.

If a table is defined without a key column, the database system implicitly creates a key column SYSKEY CHAR(8) BYTE. This column is not visible with a SELECT *. However, it can be specified explicitly and has then the same function as a key column. The SYSKEY column can be used to obtain unique keys generated by the database system. The keys are in ascending order, thus reflecting the order of insertion in the table. The key values in the SYSKEY column are only unique within a table; i.e., the SYSKEY column in two different tables may contain the same values. If a unique key is desired across the entire database system, a key column of the data type CHAR(8) BYTE with the DEFAULT specification [Page 120] STAMP can be defined.

NOT NULL

NOT NULL must not be used together with the DEFAULT specification [Page 120] DEFAULT NULL.

NOT NULL WITH DEFAULT defines a default value that is dependent on the data type of the column. NOT NULL WITH DEFAULT must not be used with any of the DEFAULT specifications.

Column data type	DEFAULT value
CHAR [Page 115](n); VARCHAR [Page 116](n)	' '
CHAR(n) BYTE; VARCHAR(n) BYTE	X'00'
FIXED [Page 117](p,s), INT [Page 117], SMALLINT [Page 117], FLOAT [Page 117](p)	0
DATE [Page 118]	DATE
TIME [Page 118]	TIME
TIMESTAMP [Page 118]	TIMESTAMP
BOOLEAN [Page 116]	FALSE

DEFAULT specification(default_spec)

A DEFAULT specification is formed by specifying the keyword DEFAULT and a DEFAULT value. The maximum length of a default value is 254 characters.

Syntax

```
<default_spec> ::= DEFAULT <literal> | DEFAULT NULL
| DEFAULT USER | DEFAULT USERGROUP
| DEFAULT DATE | DEFAULT TIME | DEFAULT TIMESTAMP
| DEFAULT TRUE | DEFAULT FALSE
| DEFAULT TRANSACTION | DEFAULT STAMP
| DEFAULT SERIAL[(<unsigned_integer>)]
```

literal [Page 36], unsigned_integer [Page 38]

Explanation

If a DEFAULT specification has been made for a column, the default value (<literal>, NULL, USER,...) must be a value that can be inserted in the column.

DEFAULT specification	Explanation
DEFAULT <literal>	The literal must be comparable with the data type of the column.
DEFAULT USER	Supplies the user name of the current user and can only be specified for columns of the data type [VAR]CHAR(n) (n>=32).
DEFAULT USERGROUP	Supplies only members of a usergroup, the usergroup name, or the user name for users that do not belong to a usergroup. This DEFAULT specification can only be specified for columns of the data type [VAR]CHAR(n) (n>=32).
DEFAULT DATE	Supplies the current date [Page 23] and can only be specified for columns of the data type DATE.
DEFAULT TIME	Supplies the current time [Page 23] and can only be specified for columns of the data type TIME.
DEFAULT TIMESTAMP	Supplies the current timestamp [Page 24] and can only be specified for columns of the data type TIMESTAMP.
DEFAULT TRUE/DEFAULT FALSE	Can only be specified for columns of the data type BOOLEAN [Page 24].
DEFAULT TRANSACTION	Supplies the identification of the current transaction [Page 32] and can only be specified for columns of the data type CHAR(n) BYTE (n>=8).
DEFAULT STAMP	Supplies a value of eight characters in length that is unique within the database system and can only be specified for columns of the data type CHAR(n) BYTE (n>=8). If a table is defined without a key column, the database system implicitly creates a key column SYSKEY CHAR(8) BYTE. The key values in the SYSKEY column are only unique within a table; i.e., the SYSKEY column in two different tables may contain the same values. If a unique key is desired across the entire database system, a key column can be defined with the DEFAULT specification STAMP.
DEFAULT SERIAL [(<unsigned_integer)]	Supplies a number generator for positive integers and can only be specified for columns of the data type INTEGER, SMALLINT, and FIXED without decimal places (SERIAL [Page 28]). The first value generated by the generator can be defined by specifying an unsigned integer (must be greater than 0). If this definition is missing, 1 is defined as the first value. If the value 0 is inserted in this column by an INSERT statement, the current number generator value is supplied and not the value 0. Each table may not contain more than one column with the DEFAULT specification DEFAULT SERIAL.

CONSTRAINT definition

A CONSTRAINT definition defines an integrity condition (restrictions for column values, see data integrity [Page 33]) that must be fulfilled by all the rows **in one** table.

Syntax

```
<constraint_definition> ::= CHECK <search_condition>
| CONSTRAINT <search_condition>
| CONSTRAINT <constraint_name> CHECK <search_condition>
```

search condition [Page 107], constraint name [Page 43]

Simple constraint (for one column), model table customer [Page 83]

```
title CHAR (7) CONSTRAINT title IN ('Mr', 'Mrs', 'Comp')
```

Complex constraint (for several columns), model table reservation [Page 86]

```
arrival DATE NOT NULL
departure DATE CONSTRAINT departure > arrival
```

The system checks whether the arrival is before the departure.

Explanation

A CONSTRAINT definition defines an integrity condition that must be fulfilled by all the column values in the columns defined by the column definition [Page 114] with CONSTRAINT definition.

The CONSTRAINT definition in a column is checked when a row is inserted and a column changed that occurs in the CONSTRAINT definition. If the CONSTRAINT definition is violated, the INSERT or UPDATE statement fails.

When you define a constraint, you specify implicitly that the NULL value is not permitted as an input.

The search condition (`search_condition`) of the CONSTRAINT definition must not contain a subquery [Page 189].

The search condition of the CONSTRAINT definition must only contain column names in the form `<column name>`.

Constraint name

- No constraint name:
  The database system assigns a constraint name that is unique for the table in question.
- Constraint name is specified:
  The constraint name must be different to all other constraint names for this table.

Number of columns in a search condition

- Contains only one column name in the table:
  When the table is created (CREATE TABLE statement [Page 111]), you can check whether an additional DEFAULT value (default_spec [Page 120]) specified as a column attribute fulfills the search condition. If it is not true, the CREATE TABLE statement fails.
- Contains more than one column name in the table:
  When the table is created (CREATE TABLE statement), it is not possible to decide whether DEFAULT values of the table columns fulfill the search condition. In this case, an attempt to insert DEFAULT values in the table when an INSERT or UPDATE statement is executed may fail.

Referential CONSTRAINT definition

A referential CONSTRAINT definition defines an integrity condition (restrictions for columns values, see data integrity [Page 33]) that must be satisfied by all the rows in **two** tables. The resultant dependency between two tables affects changes to the rows contained in them.

Syntax

```
<referential_constraint_definition> ::=
FOREIGN KEY [<referential_constraint_name>] (<referencing_column>,...)
REFERENCES <referenced_table> [(<referenced_column>,...)] [<delete_rule>]
```

referential_constraint_name [Page 48], delete_rule [Page 124]

`referenced_table` `referenced_column`	Reference table, referenced column (table/column that is to be addressed)
`referencing_column`	Referencing column (column that establishes the link to the column that is to be addressed)

Dependency between the model tables customer [Page 83] and reservation [Page 86]. The referential CONSTRAINT definition is specified when the `reservation` table is defined. The `reservation` table is assigned a foreign key that corresponds to the key in the `customer` table.

```
CREATE TABLE reservation(rno FIXED (4) KEY, cno FIXED (4), hno
FIXED (4)
roomtype CHAR (6), arrival DATE, departure DATE,
FOREIGN KEY (cno) REFERENCES customer ON DELETE CASCADE
```

The defined relationship is assigned the name `customer_reservation`. The DELETE rule `ON DELETE CASCADE` specifies that deleting rows in the customer table causes the associated rows in the reservation table to be deleted automatically.

Explanation

A referential CONSTRAINT definition can be used in a CREATE TABLE statement [Page 111] or ALTER TABLE statement [Page 127]. The table specified in the corresponding statement (`table_name`) is referred to in the following sections as the referencing table.

The referencing columns are specified in the referential CONSTRAINT definition. The referencing columns must denote columns in the referencing table and must all be different. They are also called **foreign key columns**.

Referenced columns

- If no referencing columns are specified, the result is the same as if the key columns in the referenced table were specified in the defined sequence.
- If referenced columns are specified that are not the key in the referencing table, the referenced table must have a UNIQUE definition [Page 126] whose column names and sequence match those of the referenced columns.

Relationship between referenced and referencing columns:

- The number of referenced columns is equal to the number of referencing columns.
- 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.

The referencing table and the referenced table must be base tables, but not temporary base tables.

The current user must have the ALTER privilege for the referencing table and the REFERENCE privilege for the referenced table.

Name of a referential constraint

The name of a referential constraint [Page 48] can be specified after the keywords `FOREIGN KEY`.

- If the **name of a referential constraint is specified**, it must be different from all other names of referential constraints for the referencing table.
- If **no referential constraint name is specified**, the database system assigns a unique name (based on the referencing table).

Inserting and modifying rows in the referenced table

The following restrictions apply when rows in the referencing table are added or modified:

Let Z be an inserted or modified row. Rows can only be inserted or modified if one of the following conditions is fulfilled for the associated referenced table:

- Z is a matching row [Page 126]
- Z contains a NULL value in one of the referencing columns.
- The referential CONSTRAINT definition defines the DELETE rule ON DEFAULT SET DEFAULT, and Z contains the DEFAULT value in each referencing column.

Further terms

- DELETE rule [Page 124]
- CASCADE dependency [Page 125]
- Reference cycle [Page 125]
- A referential CONSTRAINT definition is **self-referencing** if the referenced and referencing tables are identical.
  With self-referencing referential CONSTRAINT definitions, the order in which a DELETE statement is processed can be important.
  Specifying CASCADE: all of the rows affected by the DELETE statement are first deleted irrespective of the referential CONSTRAINT conditions. All matching rows in the rows that have just been deleted are then also deleted. As a result, all of the matching rows in the previous deletion operation are deleted, etc.
  Specifying SET NULL or SET DEFAULT: all of the rows affected by the DELETE statement are first deleted irrespective of the referential CONSTRAINT conditions. Following this, SET NULL or SET DEFAULT is applied to the matching row.
- When rows are deleted from a referenced table, the number of rows deleted is entered in the third SQLERRD entry in the SQLCA database.
- When an INSERT or UPDATE statement is applied to a referencing table, the database uses a blocking behavior for the referenced table that corresponds to isolation level 1, irrespective of the isolation level defined for the current session.
  When a DELETE statement is applied to a referenced table, the database system uses a blocking behavior that corresponds to isolation level 3 (see CONNECT statement [Page 204]).

DELETE rule

The DELETE rule defines the effects that deleting a row in the referenced table has on the referencing table (see referential_constraint_definition [Page 123]). The DELETE rule is also used when column attributes [Page 119] are defined.

Syntax

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

Explanation

- No DELETE rule: deleting a row in the referenced table will fail if matching rows [Page 126] exist.
- ON DELETE CASCADE: if a row in the referenced table is deleted, all of the matching rows are deleted.
- ON DELETE RESTRICT: deleting a row in the referenced table will fails if matching rows exist.
- ON DELETE SET DEFAULT: if a row in the referenced table is deleted, the associated DEFAULT value is assigned to each referencing column for each matching row.
  A DEFAULT specification [Page 120] must exist for each referencing column.
- ON DELETE SET NULL: if a row in the referenced table is deleted, a NULL value is assigned to each referencing column of every matching row.
  None of these referencing tables may be a NOT NULL column.

CASCADE dependency

A table T* is CASCADE dependent on table T if a series of referential CONSTRAINT definitions [Page 123] R1, R2, ..., Rn (n>=1) exist where:

- T* is the referencing table of R1
- T is the referenced table of Rn
- All of the referential CONSTRAINT definitions use CASCADE from the DELETE rule [Page 124] or from the CASCADE option [Page 127].
- For i=1,...,n-1, n>1 is the referenced table of Ri is equal to the referencing table of Ri+1

Let R1 and R2 be two different referential CONSTRAINT definitions with the same referencing table S. T1 denotes the referenced table of R1, T2 denotes the referenced table of R2.

If T1 and T2 are identical, or if a table T exists so that T1 and T2 are CASCADE dependent on T, then R1 and R2 must both specify either CASCADE or RESTRICT.

There are different sequences of referential CONSTRAINT definitions that link the tables S and T. A DELETE statement on table T results in an action in table S. In order to ensure that the result of the DELETE statement does not depend on which of the two sequences of referential CONSTRAINT definitions is processed, the above restriction was selected for R1 and R2.

See also:

CASCADE option [Page 127]

Reference cycle

A reference cycle is a sequence of referential CONSTRAINT definitions [Page 123] R1, R2,...,Rn where n>1, so that the following applies:

- i=1,...,n-1 the referenced table of Ri is equal to the referencing table of Ri+1
- the reference table of Rn is the referencing table of R1

A reference cycle in which all of the referential CONSTRAINT definitions specify CASCADE (CASCADE dependency [Page 125]) is not allowed.

A reference cycle in which one referential CONSTRAINT definition does not specify CASCADE and all other referential CONSTRAINT definitions specify CASCADE is not allowed.

Matching row

A row in the referencing table is called a matching row of a row in the referenced table if the values of the corresponding referencing and referenced columns are identical.

A referential CONSTRAINT definition [Page 123] defines a 1:n relationship between two tables. This means that more than one matching row can exist for each row in the referenced table.

A row in the referenced table in a referenced column cannot be changed if at least one matching row exists.

Key definition

A key definition in a CREATE TABLE statement [Page 111] or an ALTER TABLE statement [Page 127] defines the key in a base table. The key definition is introduced by the keywords `PRIMARY KEY`.

Syntax

```
<key_definition> :: PRIMARY KEY (<column_name>,...)
```

column_name [Page 49]

SQL statement for creating a `person` table with a one-column primary key for the column `cno`:

```
CREATE TABLE person (cno FIXED(4), firstname CHAR(7), name
CHAR(7), account FIXED(7,2), PRIMARY KEY (cno))
```

Rows are inserted in the same way as in a base table without a key definition. Double entries for the customer number, however, are rejected.

Explanation

The column name must identify a column in the base table. The specified column names are key columns in the table.

A key column must not identify a column of the data type LONG [Page 116] and is always a NOT NULL column. The database system ensures that no key column has a NULL value and that no two rows of the table have the same values in all key columns.

The sum of the internal lengths of the key columns must not exceed 1024 characters.

UNIQUE definition

A UNIQUE definition in the CREATE TABLE statement [Page 111] defines the uniqueness of column value combinations.

Syntax

```
<unique_definition> ::= [CONSTRAINT <index_name>] UNIQUE
(<column_name>,...)
```

index_name [Page 45], column_name [Page 49]

Explanation

Specifying a UNIQUE definition in a CREATE TABLE statement has the same effect as specifying the CREATE TABLE statement without a UNIQUE definition, but with a CREATE INDEX statement [Page 141] with UNIQUE.

- **Index name specified**: the generated index is stored under this name in the catalog.

- **No index name and more than one column name specified**: the database system assigns the index a unique index name.

DROP TABLE statement

A DROP TABLE statement deletes a base table (see Table [Page 29]).

Syntax

```
<drop_table_statement> ::= DROP TABLE <table_name> [<cascade_option>]
```

table_name [Page 50], cascade_option [Page 127]

Explanation

The table name must be the name of an existing base table. The current user must be the owner of the base table.

All the metadata and rows in the base table and all the view tables (see table [Page 29]), indexes [Page 30], privileges [Page 31], synonyms [Page 30], and referential CONSTRAINT definitions [Page 123] derived from it are deleted.

CASCADE option RESTRICT: the DROP TABLE statement will fail if view tables or synonyms are based on the specified table.

No CASCADE option specified: the CASCADE value is accepted.

If all of the data that is linked to this base table by means of a referential_constraint_definition [Page 123] with a DELETE rule [Page 124], are processed according to the specified DELETE rule, a DELETE statement [Page 171] must first be executed for this base table and then the DROP TABLE statement.

CASCADE option

A CASCADE option determines the deletion behavior for objects (e.g. tables, users), i.e. it defined whether certain dependencies are to be taken into account when objects are deleted.

Syntax

```
<cascade_option> ::= CASCADE | RESTRICT
```

See also:

CASCADE dependency [Page 125]

ALTER TABLE statement

An ALTER TABLE statement changes the properties of a base table (see Table [Page 29]).

Syntax

```
<alter_table_statement> ::=
  ALTER TABLE <table_name> <add_definition>
| ALTER TABLE <table_name> <drop_definition>
| ALTER TABLE <table_name> <alter_definition>
| ALTER TABLE <table_name> <column_change_definition>
| ALTER TABLE <table_name> <modify_definition>
| ALTER TABLE <table_name> <referential_constraint_definition>
| ALTER TABLE <table_name> DROP FOREIGN KEY <referential_constraint_name>
| ALTER TABLE <table_name> <sample_definition>
```

table_name [Page 50], add_definition [Page 128], drop_definition [Page 130], alter_definition [Page 129], column_change_definition [Page 130], modify_definition [Page 131], referential_constraint_definition [Page 123], referential_constraint_name [Page 48], sample_definition [Page 113]

Explanation

The table name must be the name of an existing base table. The table must not be a temporary base table. The current user must have the ALTER privilege for the specified table.

- If a referential CONSTRAINT definition was specified, a new referential constraint is defined for the base table. The rules described in the referential CONSTRAINT definition [Page 123] apply.
- If DROP FOREIGN KEY was specified, the referential CONSTRAINT definition identified by the name of the referential constraint is dropped.
- If a SAMPLE definition is specified, a new number of rows is defined and is taken into account by the database system when the table statistics are calculated.

ADD definition

You can define additional table properties by specifying an ADD definition in the ALTER TABLE statement [Page 127].

Syntax

```
<add_definition> ::= ADD <column_definition>,...
| ADD (<column_definition>,...)
| ADD <constraint_definition>
| ADD <referential_constraint_definition>
| ADD <key_definition>
```

column_definition [Page 114], constraint_definition [Page 122], key_definition [Page 126], referential_constraint_definition [Page 123]

The following statement adds two columns to the customer [Page 83] table. The columns initially contain the NULL value [Page 22] in all rows.

```
ALTER TABLE customer ADD (telephone FIXED (8), street CHAR (15))
```

The new columns can be used straight away.

Explanation

Adding a column definition: ADD <column_definition>

A domain name [Page 44] can only be specified in a column definition [Page 114] if the domain was defined without a DEFAULT specification [Page 120].

You can extend the table specified in the ALTER TABLE statement to include these columns by specifying column definitions. These specifications must not exceed the maximum number of columns allowed and the maximum length of a row.

The memory requirements for each column are increased by one character (for normal memory requirements, see Memory requirements of a column value as a function of the data type [Page 118]), if the length described is less than 31 characters and the column does not have the data type VARCHAR [Page 116].

None of the newly defined columns may have the data type LONG [Page 116]. The column names specified must differ from each other and must not be identical to the names existing columns in the table.

The new columns contain the NULL value in all rows. If the NULL value violates a CONSTRAINT definition [Page 122] of the table, the ALTER TABLE statement will fail.

In every other respect, specifying a column definition has the same effect as specifying a column definition in a CREATE TABLE statement [Page 111].

- If view tables are defined for the specified table, and if alias names [Page 43] are defined for one of these view tables, and if the view tables reference the columns in the table with *, the ALTER TABLE statement will fail.
- If view tables are defined for the specified table, and if no alias names are defined, and if the view tables reference the columns in the table with *, this view table contains the columns added to the base table by the ADD definition.

Adding a CONSTANT definition: ADD <constant definition>

All of the rows in the table must satisfy the condition defined by the search condition [Page 107] of the CONSTRAINT definition [Page 122].

Adding a referential CONSTRAINT definition: ADD <referential_constraint_definition>

An integrity condition is defined for the table specified in the ALTER TABLE statement. The columns specified in the referential CONSTRAINT definition [Page 123] must be columns in the table. All of the rows in the table must satisfy the integrity condition defined by the referential CONSTRAINT definition.

Adding a key definition: ADD <key definition>

A key is defined for the table specified in the ALTER TABLE statement. At execution time, the table must only contain the key column SYSKEY generated by the database system. The columns specified in the key definition [Page 126] must be columns in the table and must satisfy the key properties (none of the columns may contain NULL value, and no two rows in the table may have the same values in all columns of the key definition). The new key is stored in the metadata of the table. The key column SYSKEY is omitted. This is an extremely lengthy procedure for tables with a large number of rows, since extensive copy operations are carried out.

ALTER definition

By specifying an ALTER definition in the ALTER TABLE statement [Page 127], you can change a CONSTRAINT definition [Page 122] or a key definition [Page 126] .

Syntax

```
<alter_definition> ::=
  ALTER CONSTRAINT <constraint_name> CHECK <search_condition>
| ALTER <key_definition>
```

constraint_name [Page 43], search_condition [Page 107], key_definition [Page 126]

Explanation

CONSTRAINT <constraint_name>

The constraint name must identify a CONSTRAINT definition in the table. If the specified search condition is not violated by any row in the table, it replaces the existing search condition of the CONSTRAINT definition. Otherwise, the ALTER TABLE statement fails.

<key_definition>

The key specified by the key definition replaces the current key in the table. The columns specified in the key definition must identify columns in the table and must have the key property (key definition [Page 126]).

If a column of the key to be replaced is a referenced column of a referential CONSTRAINT definition, the ALTER TABLE statement will fail.

With large tables, in particular, this may take more time, since extensive copy operations have to be carried out.

COLUMN change definition

You can modify the properties of a column by specifying a COLUMN change definition in the ALTER TABLE statement [Page 127].

Syntax

```
<column_change_definition> ::= COLUMN <column_name> NOT NULL
| COLUMN <column_name> DEFAULT NULL
| COLUMN <column_name> ADD <default_spec>
| COLUMN <column_name> ALTER <default_spec>
| COLUMN <column_name> DROP DEFAULT
```

column_name [Page 49], default_spec [Page 120]

Explanation

NOT NULL

`NOT NULL` can only be specified if the column contains no NULL value [Page 22]s. You cannot add a NULL value to the column once the ALTER TABLE statement has been successfully executed.

DEFAULT NULL

`DEFAULT NULL` allows a NULL value for the column. The system does not check whether a NULL value violates existing CONSTRAINT definitions [Page 122] in the table. For this reason, inserting the NULL value can fail when an INSERT or UPDATE statement is executed.

ADD <default_spec>

The column must not contain a DEFAULT specification before the ALTER TABLE statement is executed with `ADD <default spec>`. `ADD <default spec>` assigns a DEFAULT value to the column.

ALTER <default spec>

`ALTER <default spec>` changes the DEFAULT value assigned to the column. All of the rows that contain the old default value in the column remain unaltered.

DROP DEFAULT

`DROP DEFAULT` drops the DEFAULT specification of the column. If the column is the foreign key column of a referential CONSTRAINT definition [Page 123] with the DELETE RULE [Page 124] ON DELETE SET DEFAULT, the ALTER TABLE statement will fail.

DROP definition

You can delete table properties by specifying a DROP definition in the ALTER TABLE statement [Page 127].

Syntax

```
<drop_definition> ::= DROP <column_name>,... [<cascade_option>] [RELEASE
SPACE]
| DROP (<column_name>,...) [<cascade_option>] [RELEASE SPACE]
| DROP CONSTRAINT <constraint_name> | DROP PRIMARY KEY
```

column_name [Page 49], cascade_option [Page 127], constraint_name [Page 43]

Explanation

Dropping a column: DROP <column_name>

Each column name must be a column of the table identified by the ALTER TABLE statement. The column must be neither a key column nor a foreign key column of a referential CONSTRAINT definition [Page 123] of the table.

The columns are marked as dropped in the metadata of the table. A DROP definition does not automatically reduce the memory requirements of the underlying table. `RELEASE SPACE` forces the column values of the dropped columns to be dropped in every row in the table. With large tables, in particular, this may take more time, since extensive copy operations have to be carried out.

Any privileges and comments for the columns to be dropped are dropped as well.

If one of the columns to be dropped occurs as a selected column [Page 183] in a view definition, the specified column in the view table is dropped.
If this view table is used in the FROM condition of another view table, the described procedure is recursively applied to this view table.

- If one of the columns to be dropped occurs in the QUERY specification [Page 182] of a view definition and if no CASCADE condition [Page 127] is specified or if the CASCADE condition in the DROP is specified in the DROP definition, the view definition is dropped with all the view tables, privileges, and synonyms that depend on it.
- The ALTER TABLE statment will fail if one of the columns to be dropped appears in the QUERY specification of a view definition and the CASCADE condition RESTRICT is specified in the DROP definition.

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

All CONSTRAINT definitions [Page 122] that contain one of the dropped columns are dropped.

Dropping a constraint: DROP CONSTRAINT <constraint_name>

The constraint name must identify a CONSTRAINT definition in the table. The latter is then removed from the metadata of the table.

Dropping a key: DROP PRIMARY KEY

- The table must have a key defined by the user.
- The table must not contain more than 1023 columns.
- The maximum permissible length of a row must not exceed 8088 bytes.
- The key columns must not be a referenced column of a referential CONSTRAINT definition [Page 123].

The key is replaced by the key column SYSKEY generated by the database system. With large tables, in particular, this may take more time, since extensive copy operations have to be carried out.

MODIFY definition

You can modify data types and properties of table columns by specifying a MODIFY definition in the ALTER TABLE statement [Page 127].

Syntax

```
<modify_definition> ::= MODIFY (<column_name> [<data_type>]
[<column_attributes>]...)
```

column_name [Page 49], data_type [Page 114], column_attributes [Page 119]

The parentheses are not necessary if the MODIFY definition only contains one column name.

Explanation

Each column name must be a column of the base table specified in the ALTER TABLE statement.

Column attributes

Only the following column attributes are allowed:

- `NULL`
  NULL value [Page 22]
- `NOT NULL`
  If NOT NULL is specified, the table must not have any rows that contain a NULL value in the corresponding column. A NULL value can no longer be inserted into the column after being modified.
- DEFAULT specification [Page 120]
  The DEFAULT specification DEFAULT SERIAL is not permitted.
  If a DEFAULT specification is specified, it replaces an existing DEFAULT specification in the corresponding column. The new DEFAULT specification only affects subsequent INSERT statements and not affect rows that already exist in the table.

Data types

If a DEFAULT specification is not specified and if a DEFAULT specification is defined for the corresponding column, it must be compatible with the data type.

- **Code attribute ASCII or EBCDIC**: the corresponding column must have the data type DATE [Page 118], TIME [Page 118], or TIMESTAMP [Page 118] or the code attribute [Page 24] ASCII, EBCDIC, or UNICODE before it is modified.
- **Code attribute UNICODE**: A transformation from UNICODE to ASCII must be possible for the relevant column.
- **Code attribute BYTE**: the corresponding column must have the data type DATE, TIME, or TIMESTAMP or the code attribute ASCII, EBCDIC, or BYTE before it is modified.

Data type **CHAR(n)**, **VARCHAR(n)**: the corresponding column must have the data type CHAR [Page 115](n), VARCHAR [Page 116](n), DATE, TIME, or TIMESTAMP. In this case, the table must not contain a row in which the column has a value with a length greater than n.
If a column had the code attribute UNICODE before the ALTER TABLE statement was executed, and the new code attribute is not UNICODE, transformation to the new code attribute must be possible.

Data type **DATE**: the corresponding column must have the data type CHAR(n), VARCHAR(n), or DATE [Page 118]. This column must contain a date value in any of the date formats supported by the database system in all rows of the table.

Data type **FIXED(n,m)**: the corresponding column must have the data type FIXED [Page 117](n,m), FLOAT [Page 117], INT [Page 117], or SMALLINT [Page 117]. In this case, the table must not contain a row in which the column has a value with more than (n - m) integral or m fractional digits.

Data type **FLOAT(n)**: the corresponding column must have the data type FIXED(n,m), FLOAT(n), INT, or SMALLINT.

Data type **INT**: the corresponding column must have the data type FIXED(n,m), FLOAT(n), INT, or SMALLINT. In this case, the table must only contain rows in which this column has integral values in the range between -2147483648 and 2147483647.

Data type **SMALLINT**: the corresponding column must have the data type FIXED(n,m), FLOAT(n), INT, or SMALLINT. In this case, the table must only contain rows in which this column has integral values in the range between -32768 and 32767.

Data type **TIME**: the corresponding column must have the data type CHAR(n), VARCHAR(n), or TIME [Page 118]. This column must contain a time value in any of the time formats supported by the database system in all rows of the table.

Data type **TIMESTAMP**: the corresponding column must have the data type CHAR(n), VARCHAR(n), or TIMESTAMP [Page 118]. This column must contain a timestamp value in any of the timestamp formats supported by the database system in all rows of the table.

Column attribute **NULL**: a NULL value [Page 22] can be entered in the corresponding column with a subsequent INSERT or UPDATE statement.

If one of the columns identified by `column name` is contained in a search condition [Page 107] for the table, this column must also define a legal search condition after the data type has been modified.

Others

Depending on the type of modification, the MODIFY definition may result in the table having to be recopied and/or indexes rebuilt. In such a case, the runtime will be considerably long.

If a table is recopied and the table contains columns marked as deleted, then these columns are removed from the catalog and from the table rows, thus reducing the space requirements of the table.

RENAME TABLE statement

A RENAME TABLE statement changes the name of a base table (see Table [Page 29]).

Syntax

```
<rename_table_statement> ::=
RENAME TABLE <old_table_name> TO <new_table_name>

<old_table_name> ::= <table_name>
<new_table_name> ::= <identifier>
```

table_name [Page 50], identifier [Page 40]

Explanation

The old table name must identify a base table that is not a temporary table. The current user must be the owner of the table.

The new table name must not already be assigned to a base or view table or a private synonym [Page 30] of the current user.

The old table is assigned the name specified in the `new_table_name`. All of the properties of the table (e.g. privileges, indexes) remain unchanged. The definitions of view tables based on the old table name are adapted to the new name.

RENAME COLUMN statement

A RENAME COLUMN statement (`rename_column_statement`) changes the name of a table column.

Syntax

```
<rename_column_statement> ::=
RENAME COLUMN <table_name>.<column_name> TO <column_name>
```

table_name [Page 50], column_name [Page 49]

Explanation

The specified table [Page 29] must be a base or view table. The current user must be the owner of the table.

The specified table column is given a new name. If the column name of a view table (that was defined with this table) was derived from the column name of the base table, the old column name in the view table is replaced by the new name. If the new column name is identical with an existing column name of the view table, the RENAME COLUMN statement fails.

EXISTS TABLE statement

An EXISTS TABLE statement indicates whether a table exists or not.

Syntax

```
<exists_table_statement> ::= EXISTS TABLE <table_name>
```

table_name [Page 50]

Explanation

The specified table [Page 29] must be a base table, view table, or a synonym [Page 30].

The existence or non-existence of the specified table is indicated by the return code 0 or by the error message `-4004 UNKNOWN TABLE NAME`.

A table only exists for a user if the user has a privilege on this table.

CREATE DOMAIN statement

A CREATE DOMAIN statement defines a value range (domain) [Page 30].

Syntax

```
<create_domain_statement> ::= CREATE DOMAIN <domain_name> <data_type>
[<default_spec>] [<constraint_definition>]
```

domain_name [Page 44], data_type [Page 114], default_spec [Page 120], constraint_definition [Page 122]

Explanation

The CONSTRAINT definition must not contain a constraint name [Page 43].

The CREATE DOMAIN statement can be executed by all users with DBA status.

If the domain name is specified without an owner, the current user is assumed to be the owner. If you specify the domain name with an owner, this owner must be identical to the current user. In this case, the current user becomes the owner of the domain.

The name of the domain must differ from all other domain names of the current user.

If a domain is generated with a CONSTRAINT definition, the domain name is included in the search condition [Page 107] as a column name.

DROP DOMAIN statement

A DROP DOMAIN statement drops the definition of a domain [Page 30].

Syntax

```
<drop_domain_statement> ::= DROP DOMAIN <domain_name>
```

domain_name [Page 44]

Explanation

The domain name must identify an existing domain. The current user must be owner of the domain.

The metadata of the domain is dropped from the catalog. Dropping a domain has no effect on tables in which this domain was used to define columns.

CREATE SEQUENCE statement

The CREATE SEQUENCE statement defines a database object that supplies integer values (number generator). In the following description, this object is referred to as a sequence.

Syntax

```
<create_sequence_statement> ::= CREATE SEQUENCE [<owner>.]<sequence_name>
[INCREMENT BY <integer>] [START WITH <integer>]
[MAXVALUE <integer> | NOMAXVALUE] [MINVALUE <integer> | NOMINVALUE]
[CYCLE | NOCYCLE]
[CACHE <unsigned_integer> | NOCACHE]
[ORDER | NOORDER]
```

owner [Page 44], sequence_name [Page 49], integer [Page 39], unsigned_integer [Page 38]

Explanation

The sequence names can be specified in any order.

The current user must have RESOURCE or DBA status. If an owner is specified, it must identify the current user. The current user becomes the owner of the sequence.

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

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. The value 1 is used if no value is assigned.

START WITH

Defines the first sequence value. If no value is specified, the value specified for MAXVALUE or –1 is used for descending sequences and the value specified for MINVALUE or 1 for ascending sequences.

MINVALUE

Defines the smallest value generated by the sequence. If no value is defined for MINVALUE, the smallest integer value that can be represented with 38 digits is used.

MAXVALUE

Defines the largest value generated by the sequence. If no value is defined for MAXVALUE, the largest integer value that can be represented with 38 digits is used.

CYCLE/NOCYCLE

CYCLE: MINVALUE is produced for ascending sequences after MAXVALUE has been assigned. MAXVALUE is produced for ascending sequences after MINVALUE has been assigned.

NOCYCLE: a request for a sequence value fails if the end of the sequence has already been reached, i.e. if MAXVALUE has been assigned for ascending sequences or MINVALUE for descending sequences.

If neither CYCLE nor NOCYCLE is specified, NOCYCLE is assumed.

CACHE/NOCACHE

CACHE: access to the sequence can be accelerated because the defined number of sequence values is held in the memory.

NOCACHE: no sequence values are defined beforehand.

Sequence values can be specified using CURRVAL and NEXTVAL (see value_spec [Page 52]). In this way, you can interrogate or increase the current counter value.

DROP SEQUENCE statement

A DROP SEQUENCE statement drops a number generator (sequence).

Syntax

```
<drop_sequence_statement> ::= DROP SEQUENCE [<owner>.]<sequence_name>
```

owner [Page 44], sequence_name [Page 49]

Explanation

The current user must be the owner. The sequence name must identify a sequence of the current user.

The metadata of the sequence is removed from the catalog.

CREATE SYNONYM statement

The CREATE SYNONYM statement defines a synonym [Page 30] (alternative name) of a table name [Page 50].

Syntax

```
<create_synonym_statement> ::= CREATE [PUBLIC] SYNONYM
[<owner>.]<synonym_name> FOR <table_name>
```

owner [Page 44], synonym_name [Page 49], table_name [Page 50]

Explanation

The table name must not denote a temporary base table (see Table [Page 29]). The user must have a privilege for the specified table. The current user must be the owner.

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.

PUBLIC

If PUBLIC is specified, the synonym name must not be identical to the name of a synonym defined with PUBLIC. A synonym is generated that can be accessed by all users.

If PUBLIC is not specified, a private synonym is generated that is only known by the current user. In this case, the synonym name must not be identical to the name of an existing base table, view table, or a private synonym of the current user. If a synonym with the same name and the PUBLIC attribute exists, it cannot be accessed by the current user until the private synonym has been dropped.

DROP SYNONYM statement

The DROP SYNONYM statement drops a synonym [Page 30] (alternative name) of a table name [Page 50].

Syntax

```
<drop_synonym_statement> ::= DROP [PUBLIC] SYNONYM [<owner>.]<synonym_name>
```

owner [Page 44], synonym_name [Page 49]

Explanation

The specified synonym name must identify an existing synonym of the current user.

If PUBLIC is specified, the synonym identified by the synonym name must be defined as PUBLIC.

The synonym definition is removed from the set of table name synonyms available to the user.

RENAME SYNONYM statement

The RENAME SYNONYM statement changes the name of a synonym [Page 30].

Syntax

```
<rename_synonym_statement> ::= RENAME [PUBLIC] SYNONYM <old_synonym_name>
TO <new_synonym_name>
```

old/new_synonym_name [Page 49]

Explanation

The old synonym name must have been generated by the current user.

If PUBLIC is specified, the old must be defined as PUBLIC.

A table of the current user with the new synonym name must not exist already.

CREATE VIEW statement

The CREATE VIEW statement defines a view table (see Table [Page 29]). A view table never actually exists physically. Instead, it is formed from the rows of the underlying base table(s) when this view table is specified in an SQL statement.

Syntax

```
<create_view_statement> ::= CREATE [OR REPLACE] VIEW <table_name>
[(<alias_name>,...)] AS <query_expression> [WITH CHECK OPTION]
```

table_name [Page 50], alias_name [Page 43], query_expression [Page 179]

Explanation

When the CREATE VIEW statement is executed, metadata that describes the view table is stored in the catalog.

The view table is always identical to the table that would be obtained as the result of the QUERY expression. The QUERY expression must not contain a parameter specification [Page 51]. The QUERY expression must not reference a temporary table or a result table name [Page 45].

The table expressions [Page 185] of the QUERY specification [Page 182] in the QUERY statement of the CREATE VIEW statement must not contain a QUERY expression.

If a column selected [Page 183] by the QUERY statement is of the data type LONG, the FROM clause [Page 185] must contain exactly one table name that is based on exactly one base table.

The user must have the SELECT privilege for all columns occurring in the view definition. The user is the owner of the view table and has at least the SELECT privilege for it. The user may grant the SELECT privilege for any columns in the view table derived from columns for which the user is authorized to grant the SELECT privilege to others. The user has the INSERT, UPDATE, or DELETE privilege when he has the corresponding privileges for the tables on which the view table is based, and when the view table is updateable. The user may only grant these privileges to others if he or she is authorized to grant the corresponding privilege for all tables on which the view table is based.

OR REPLACE

If OR REPLACE is not specified, the table name must not be identical to the name of an existing view table.

If OR REPLACE is specified, the 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. The database system then attempts to adapt privileges granted for the existing view table to the new view definition, with the result that the privileges for the view table usually remain unchanged. Privileges are only removed implicitly if conflicts occur that cannot be resolved by the database system. If there are major discrepancies between the two view definitions, the CREATE VIEW statement may fail in the following case: the CREATE VIEW statement of a view table based on the existing view table cannot be executed correctly for the new view definition.

Alias names

The column names of the view table must be unique. Otherwise, alias names must be specified for the result table generated by the QUERY expression. The number of alias names must be equal to the number of columns in the result table generated by the QUERY expression. If no alias names are specified, the column names of the result table generated by the QUERY expression are applied to the view table. The column descriptions for the view table are taken from the corresponding columns in the QUERY expression. The FROM clause of the QUERY expression can contain one or more tables.

WITH CHECK OPTION

If the CREATE VIEW statement contains a WITH CHECK OPTION, the owner of the view table must have the INSERT, UPDATE, or DELETE privilege for the view table.

Specifying WITH CHECK OPTION has the effect that the INSERT statement or UPDATE statement issued on the view table does not create any rows that could not be selected subsequently via the view table; i.e. the search condition [Page 107] 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 updateable view table V1, only those rows that can be selected using V can be inserted or altered using V1.

Further terms and information

- Complex view table [Page 138]
- Updateable view table [Page 139]
- INSERT privilege for owners of the view table [Page 139]
- UPDATE privilege for owners of the view table [Page 139]
- DELETE privilege for owners of the view table [Page 140]
- If DISTINCT was specified (see DISTINCT specification [Page 183]), a SELECT ORDERED statement: searched) [Page 197] cannot be executed on the defined view table.
- Updateable join view table [Page 140]
- SELECT DIRECT and SELECT ORDERED statements cannot be executed on complex view tables or join view tables.

Complex view table

A view table (see CREATE VIEW statement [Page 137]) is a complex view table if it satisfies one of the following conditions:

- The definition of the view table contains DISTINCT or GROUP BY or HAVING.
- The CREATE VIEW statement contains EXCEPT, INTERSECT, or UNION.
- The search condition [Page 107] in the QUERY expression [Page 179] of the CREATE VIEW statement contains a subquery [Page 189].
- The CREATE VIEW statement contains an outer join, that is an OUTER JOIN indicator in a JOIN predicate [Page 99] of the search condition.

Updateable View Table

A view table (see CREATE VIEW statement [Page 137]) is called updateable if it is not a complex view table [Page 138], and if it is not based on a complex view table.

For join view tables, that is, view tables whose FROM clause [Page 185] contains more than one table or join table, the following additional conditions must be satisfied:

- Each base table on which the view table is based has a key defined by the user.
- Referential CONSTRAINT definitions [Page 123] must exist between the base tables on which the view table is based.
- One of the base tables, on which the view table is based, is not a referenced table of a referential CONSTRAINT definition for a different base table of the view table. This table is the **key table** of the view table.
- For each base table on which the view table is based, there is a sequence of referential CONSTRAINT definitions so that the respective base table can be accessed from the key table.
- The referential CONSTRAINT definitions must be reflected as a JOIN predicate [Page 99] in the search condition [Page 107] of the CREATE VIEW statement, that is, the condition "key column = foreign key column" must exist for every column in each referential CONSTRAINT definition.
- The CREATE VIEW statement must contain either the primary key or foreign key column from each referential CONSTRAINT definition as the selected column [Page 183], but not both.
- The view table must be defined with WITH CHECK OPTION.

INSERT privilege for the owner of the view table

The owner [Page 44] of the view table (see CREATE VIEW statement [Page 137]) has the INSERT privilege, i.e. he or she can specify the view table as a table in which insertion is to be made in the INSERT statement if the following conditions are satisfied:

- The view table is updateable (updateable view table [Page 139]).
- The owner of the view table has the INSERT privilege for all tables in the FROM clause [Page 185] of the CREATE VIEW statement.
- The selected columns [Page 183] of the CREATE VIEW statement consist of table columns or column names, not expressions [Page 90] with more than one column name.
- The CREATE VIEW statement contains every mandatory column from all tables of the FROM clause as the selected column.

UPDATE privilege for the owner of the view table

The owner [Page 44] of the view table (see CREATE VIEW statement [Page 137]) has the UPDATE privilege for a column in the view table, i.e. he or she can specify the column as a the column to be updated in an UPDATE statement if the following conditions are satisfied:

- The view table is updateable (updateable view table [Page 139]).
- The owner of the view table has the UPDATE privilege for the table columns or the column name that defines the column.
- The column is defined by specifying table columns or by means of a column name, but not by an expression [Page 90] with more than one column name.

DELETE privilege for the owner of the view table

The owner [Page 44] of the view table (see CREATE VIEW statement [Page 137]) has the DELETE privilege for the view table, i.e. he or she can specify the view table as a table in which entries are to be deleted in the DELETE statement if the following conditions are satisfied:

- The view table is updateable (updateable view table [Page 139]).
- The owner of the view table has the DELETE privilege for all tables in the FROM clause [Page 185] of the CREATE VIEW statement.

Updateable join view table

It is assumed that the definition of the join table V (see CREATE VIEW statement [Page 137]) in the FROM clause [Page 185] contains the base tables T1,...,Tn (n>1).

- Let Ti and Tj be two base tables selected by V. Let Rij be a referential CONSTRAINT definition [Page 123] of Ti and Tj in which Ti is the referencing table and Tj the referenced table.
  Let PKj1,...,PKjm be the key columns of Tj.
  Let Fki1,...,Fkim be the corresponding foreign key columns of Ti.
  The referential CONSTRAINT definition is **relevant** for V if the JOIN predicate [Page 99] (PKj1=Fki1 AND ... AND PKjm=FKjm) is part of the search condition [Page 107] of V.
- Let Ti and Tj be two base tables selected by V and Rij a referential CONSTRAINT definition of Ti and Tj that is relevant for V.
  Ti is the **predecessor** of Tj (Ti<Tj) if Rij is the only referential CONSTRAINT definition of Ti and Tj that is relevant for V.
- Let Rij be a referential CONSTRAINT definition that is relevant for V.
  Rij defines a **1 : 1 relationship** between Ti and Tj if the foreign key columns of Rij make up the key columns of Tj.
- Let Rij be a referential CONSTRAINT definition that is relevant to V and s a key column of Tj or a foreign key column of this referential CONSTRAINT definition of Ti. The **column s can be derived from V** if exactly one of the following conditions is satisfied:
  - s is a selected column [Page 183] of V.
  – a key column or a foreign key column s' of a referential CONSTRAINT definition that is relevant to V exists that can be derived from V and the JOIN predicate s=s' is part of the search condition of V.
- A column v of V **corresponds** to a column s of a base table T if one of the following conditions is satisfied
  - v is the i^{th} column of V and s is the i^{th} selected column of V
  - v corresponds to a key column PK of Tj of a referential CONSTRAINT definition Rij that is relevant to V and s is the foreign key column of Ti that is assigned to PK
  - v corresponds to a foreign key column FK of Ti of a referential CONSTRAINT definition Rij that is relevant to V and s is the key column of Tj that is assigned to FK.

A **join view table V is updateable** if the following conditions are satisfied:

- Each base table Ti (1 <= i <= n) has a key defined by the user.
- The database system must be able to determine a processing sequence for the underlying base tables; i.e. an order $T_{i1},\ldots,Tin$ of the tables T1,…,Tn must exist so that j < k can be deduced from Tij<Tik. The columns of V from which the key columns of Ti1 can be derived make up the key of V. Ti1 is called the key table of V. The order of the tables does not have to be unique.
- Starting with a row in the key table of V, it must be possible to assign each underlying base table exactly one row; that is, there is a sequence of tables Ti1,...,Tij for each table Tij (1 <= j <= n) such that Ti1 < .. < Tij
  This sequence is unique for each base table referred to by V.
- It must be possible to derive the key columns and foreign key columns of all referential CONSTRAINT definitions relevant to V from the columns of V.

- The join predicates needed to recognize the relevance of a referential CONSTRAINT definition must be specified in parts of the search condition defined with the WITH CHECK OPTION. If the view definition only contains base tables, this means that the view table must be defined WITH CHECK OPTION. If a view table V is derived from a view table V' and if V' was defined WITH CHECK OPTION, then V inherits the CHECK OPTION for the part of the qualification passed on by V'.

DROP VIEW statement

The DROP VIEW statement drops a view table (see Table [Page 29]).

Syntax

```
<drop_view_statement> ::= DROP VIEW <table_name> [<cascade_option>]
```

table name [Page 50], cascade option [Page 127]

Explanation

The table name must identify an existing view table.

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

The metadata of the view table and all dependent synonyms [Page 30], view tables, and privilege [Page 31]s are dropped. The tables on which the view table was created remain unaffected.

If the CASCADE option RESTRICT is specified and other view tables or synonyms based on this view table exist, the DROP VIEW statement will fail.

RENAME VIEW statement

A RENAME VIEW statement changes the name of a view table (see Table [Page 29]).

Syntax

```
<rename_view_statement> ::= RENAME VIEW <old_table_name> TO
<new_table_name>

<old_table_name> ::= <table_name>
<new_table_name> ::= <table_name>
```

table_name [Page 50]

Explanation

The `old_table_name` must be a view table. The current user must be the owner of the view table.

The `new_table_name` must not yet be used for a table of the current user.

The CREATE VIEW statement [Page 137] of the `old_table_name` view table is adapted to the new name. The result of this adaptation can be retrieved from the table DOMAIN.VIEWDEFS.

The definitions of view tables based on the `old_table_name` are adapted to the new name.

CREATE INDEX statement

The CREATE INDEX statement creates an index [Page 30] of a base table (see table [Page 29]).

Syntax

```
<create_index_statement> ::=
  CREATE [UNIQUE] INDEX <table_name>.<column_name> [ASC | DESC]
```

```
| CREATE [UNIQUE] INDEX <index_name> ON <table_name> (<column_name> [ASC |
DESC],...)
```

table_name [Page 50], column_name [Page 49], index_name [Page 45]

Explanation

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 specified table must be an existing base table, and not a temporary table.

If an index was created across exactly one column, a further one-column index cannot be created for this column.

The column defined by the `column_name` must be a column in the specified table. This column must not be a LONG column [Page 23]. All of the column name pairs must be different.

The current user must have the INDEX privilege for the columns.

The sum of the internal lengths of the identified columns must not exceed 1024 characters.

Indexes provide access to the table data via non-key columns. Maintaining these indexes, however, can be quite complex in the case of the INSERT, UPDATE, and DELETE statements.

`CREATE [UNIQUE] INDEX <index_name> ON <table_name> (<column_name> [ASC | DESC],...)`

If you use this statement, you must make sure that you do not use more than 16 column names. The index name must not be identical with an existing index name of the table. A maximum of 255 indexes can be created for each table using this statement.

If the only difference between the defined index and an index that already exists for the table is the index name, the CREATE INDEX statement will fail.

UNIQUE

If UNIQUE is specified, the database system ensures that no two rows of the specified table have the same values in the indexed columns. NULL value [Page 22]s in one-column indexes are considered to be non-identical.

ASC | DESC

The index values are stored in ascending or descending order. If the specification of ASC or DESC is omitted, ASC is implicitly assumed.

DROP INDEX statement

The DROP INDEX statement drops an index [Page 30] for a base table (see table [Page 29]). The metadata of the index is dropped from the catalog. The storage space occupied by the index is released.

Syntax

```
<drop_index_statement> ::=
  DROP INDEX <table_name>.<column_name>
| DROP INDEX <index_name> [ON <table_name>]
```

table_name [Page 50], column_name [Page 49], index_name [Page 45]

Explanation

The specified index must exist and the specified table name must be the name of an existing base table.

`ON <table_name>` is not necessary if the index name identifies an index unambiguously.

The current user must be the owner of the specified table or have the INDEX privilege for it.

ALTER INDEX statement

The ALTER INDEX statement determines how an index [Page 30] is used in data queries.

Syntax

```
<alter_index_statement> ::= ALTER INDEX <index_name> [ON <table_name>]
ENABLE
| ALTER INDEX <index_name> [ON <table_name>] DISABLE
```

index_name [Page 45], table_name [Page 50]

Explanation

When a CREATE INDEX statement [Page 141] is executed, an index is generated across the specified columns. This index is modified accordingly for all of the following SQL statements for data manipulation (INSERT statement [Page 164], UPDATE statement [Page 169], DELETE statement [Page 171]). With all other SQL statements in which individual rows in a table are specified, the database system can use this index to speed up the search for these rows.

When an ALTER INDEX statement is executed with **DISABLE**, the index cannot be used for this search but is still modified with the INSERT, UPDATE, or DELETE statements.

If an ALTER INDEX statement is executed with **ENABLE**, the index can be used again for the search.

RENAME INDEX statement

The RENAME INDEX statement changes the name of an index [Page 30].

Syntax

```
<rename_index_statement> ::= RENAME INDEX <old_index_name> [ON
<table_name>] TO <new_index_name>

<old_index_name> ::= <index_name>
<new_index_name> ::= <index_name>
```

table_name [Page 50], index_name [Page 45]

Explanation

The specified table name must be the name of an existing base table (see table [Page 29]).

The index identified by the `old_index_name` must exist. `ON <table_name>` is not necessary if this index is unique.

The current user must be the owner of the specified table or have the INDEX privilege for it.

The new index name must not be identical to an existing index name for the table.

COMMENT ON statement

The COMMENT ON statement creates, alters, or drops a comment for a database object stored in the database catalog.

Syntax

```
<comment_on_statement> ::= COMMENT ON <object_spec> IS <comment>

<object_spec> ::= see explanation
<comment> ::= <string_literal> | <parameter_name>
```

string_literal [Page 36], parameter_name [Page 46]

Explanation

Comments can be specified for the following database objects:

<object_spec> ::=	Explanation
COLUMN <table_name>.<column_name> table_name [Page 50], column_name [Page 49]	The column must exist in the specified table. The current user must be the owner of the table. The comment for this column can be interrogated by selecting the system table DOMAIN.COLUMNS [Page 213].
DBPROC[EDURE] <dbproc_name> dbproc_name [Page 44]	dbproc_name must identify an existing database procedure [Page 33] whose owner is the current user. A comment is stored for the DB procedure. The comment can be interrogated by selecting the system table DOMAIN.DBPROCEDURES [Page 215].
DOMAIN <domain_name> domain_name [Page 44]	domain_name must specify a domain [Page 30] of the current user. The comment for this domain can be interrogated by selecting the system table DOMAIN.DOMAINS [Page 216].
FOREIGN KEY <table_name>.<referential_constraint_name> referential constraint name [Page 48]	referential_constraint_name must specify a referential_CONSTRAINT definition [Page 123] for the specified table of the current owner. The comment for this referential CONSTRAINT definition can be interrogated by selecting the system table DOMAIN.FOREIGNKEYS [Page 217].
INDEX <index_name> ON <table_name> INDEX <table_name>.<column_name> index_name [Page 45]	index_name must specify an index [Page 30] and column name a column [Page 29] (for which a single-column index exists) in the specified table. The current user must be the owner of the table. The comment for this index can be interrogated by selecting the system table DOMAIN.INDEXES [Page 218].
SEQUENCE <sequence_name> sequence_name [Page 49]	An existing sequence must be specified using sequence_name. The current user must be the owner of the sequence. The comment for this sequence can be interrogated by selecting the system table DOMAIN.SEQUENCES [Page 220].
[PUBLIC] SYNONYM <synonym_name> synonym_name [Page 49]	synonym_name must specify a synonym [Page 30] of the current user. If PUBLIC is specified, the synonym must have the PUBLIC attribute. The comment for this synonym can be interrogated by selecting the system table DOMAIN.SYNONYMS [Page 221].
TABLE <table_name> table_name [Page 50]	The specified table [Page 29] must identify a base or view table of the current user that is not a temporary table. The comment for this table can be interrogated by selecting the system table DOMAIN.TABLES [Page 222].

`TRIGGER <trigger_name> ON <table_name>` trigger_name [Page 51]	The specified trigger name must identify a trigger [Page 34] of the specified table. The current user must be the owner of the table. A comment is stored for the trigger and can be interrogated by selecting the system table DOMAIN.TRIGGERS [Page 223].
`USER <user_name>` user_name [Page 43]	The specified user [Page 30] must identify an existing user whose owner is the current user. The comment for this user can be interrogated by selecting the system table DOMAIN.USERS [Page 223].
`USERGROUP <usergroup_name>` usergroup_name [Page 43]	The specified usergroup [Page 30] must identify an existing usergroup whose owner is the current user. The comment for this usergroup can be interrogated by selecting the system table DOMAIN.USERS [Page 223].
`<parameter_name>` parameter_name [Page 46]	The corresponding variable must contain one of the values listed in the table. The values must be encapsulated in inverted commas. Example: `'COLUMN <table_name>.<column_name>'`

CREATE DBPROC statement

The CREATE DBPROC statement defines a database procedure [Page 33].

Syntax

```
<create_dbproc_statement> ::= CREATE DBPROC <procedure_name>
[(<formal_parameter>,..)] AS <routine>

<formal_parameter> ::=
  IN <argument> <data_type>
| OUT <argument> <data_type>
| INOUT <argument> <data_type>

<argument> ::= <identifier>
```

procedure_name [Page 44], data_type [Page 114], identifier [Page 40], routine [Page 146]

The database procedure determines the average price for single rooms in hotels that are located within the specified zip code range.

```
CREATE DBPROC avg_price (IN zip CHAR(5), OUT avg_price FIXED(6,2))
AS
    VAR total FIXED(10,2); price FIXED(6,2); hotels INTEGER;
TRY
  SET total = 0; SET hotels = 0
  SELECT price FROM tours.room,tours.hotel WHERE zip = :zip AND
  room.hno = hotel.hno AND roomtype = 'SINGLE';
    WHILE $rc = 0 DO BEGIN
      FETCH INTO :price;
      SET total = total + price;
      SET hotels = hotels + 1;
    END;
```

```
CATCH
  IF $rc <> 100 THEN STOP ($rc, 'Unexpected error');
  IF hotels > 0 THEN SET avg_price = total / hotels;
  ELSE STOP (100, 'No hotel found');
```

Explanation

A database procedure is a subroutine that runs on the SAP DB server. SAP DB provides a language (special SQL syntax that has been extended to include variables, control structures, and troubleshooting measures) that can be used to define database procedures and triggers [Page 34].

The current user is the owner of a database procedure. He or she has the EXECUTE privilege to execute the procedure.

Parameter

When an application invokes the database procedure with the CALL statement [Page 173], it exchanges data via parameters that are defined by means of the formal parameters. A formal parameter of the database procedure usually corresponds to a variable in the application.

IN | OUT | INOUT

The parameter mode (IN | OUT | INOUT) specifies the direction in which the data is transferred when the procedure is invoked.

IN: defines an input parameter, i.e. the value of the variable is transferred to the database procedure when the procedure is invoked.

OUT: defines an output parameter, i.e. the value of the formal parameter is transferred from the database procedure to the variable after the procedure has terminated.

INOUT: defines an input/output parameter that combines the IN and OUT functions.

Argument

By specifying an argument, you assign a name to a formal parameter of the database procedure. This parameter name can then be used as a variable in expressions and assignments in the database procedure.

Data Type

Only BOOLEAN, CHAR[ACTER], DATE, FIXED, FLOAT, INT[EGER], NUMBER, REAL, SMALLINT, TIME, TIMESTAMP, and VARCHAR can be used as the data type of a formal parameter of a database procedure.

Further information

routine [Page 146]

Routine

The part of the CREATE DBPROC [Page 145] or CREATE TRIGGER-statement [Page 149] referred to as the routine is the implementation of the database procedure [Page 33] or trigger [Page 34]. . It consists of optional variable declarations and statements.

Syntax

```
<routine> ::= [<local variables>] <statement_list>;

<local_variables> ::=VAR <local_variable_list>;
<local_variable_list> ::= <local_variable> | <local_variable_list>;
<local_variable>
<local_variable> ::= <variable_name> <data_type>
<variable_name> ::= <identifier>
```

```
<statement_list> ::= <statement> | <statement_list> ; <statement>
```

identifier [Page 40], data_type [Page 114], statement [Page 147]

Explanation

Variables

The local variables of the database procedure must be declared explicitly by specifying a data type before they are used. Only BOOLEAN, CHAR[ACTER], DATE, FIXED, FLOAT, INT[EGER], NUMBER, REAL, SMALLINT, TIME, TIMESTAMP, and VARCHAR are permitted as data types data types [Page 114]. Once they have been declared, the variables can be used in any SQL and other statements.

Every database procedure has the variables $RC, $ERRMSG, and $COUNT implicitly.

The **$RC variable** returns a numeric error code after an SQL statement has been executed. The value 0 means that the SQL statement was successfully executed.
In parallel with $RC, the **$ERRMSG variable** returns an explanation of the error containing a maximum of 80 characters.
The number of lines processed in an SQL statement is indicated by the **$COUNT variable**.

Variables can be assigned a value with the `assignment_statement` (see statement [Page 147]).

Restrictions

The statement list must not contain more than 255 SQL statements.

statement

`statement` is a syntax element that is used in a routine [Page 146]. The statements specified in the `statement` syntax description can be used to define a database procedure (see CREATE DBPROC statement [Page 145]) or trigger (see CREATE TRIGGER statement [Page 149]).

Syntax

```
<statement> ::= BEGIN <statement_list> END
| BREAK | CONTINUE | CONTINUE EXECUTION
| <if_statement> | <while_statement> | <assignment_statement>
| STOP (<expression> [,<expression>] )
| TRY <statement_list>; CATCH <statement>
| <routine_sql_statement>

<statement_list> ::= <statement> | <statement_list> ; <statement>

<if_statement> ::= IF <search_condition> THEN <statement> [ELSE
<statement>]

<while_statement> ::= WHILE <search_condition> DO <statement>

<assignment_statement> ::= SET <variable_name> = <expression>

<routine_sql_statement> ::=
<call_statement> | <close_statement> | <create_table_temp>
| <declare_cursor_statement> | <delete_statement> | <fetch_statement>
| <insert_statement> | <lock_statement>
| <select_statement> | <named_select_statement> | <single_select_statement>
| <select_direct_statement:_searched> |
<select_direct_statement:_positioned>
| <select_ordered_statement:_searched> | <select_ordered
statement:_positioned>
| <subtrans_statement> | <update_statement>

<variable_name> ::= <identifier>
```

<create_table_temp> :: = <create_table_statement> for creating temporary tables, that is the table_name [Page 50] in the CREATE TABLE statement, must have the format TEMP.<identifier>.

expression [Page 90], search_condition [Page 107], call_statement [Page 173], close_statement [Page 195], create_table_statement [Page 111], declare_cursor_statement [Page 175]; delete_statement [Page 171], fetch_statement [Page 193], insert_statement [Page 164], lock_statement [Page 209], select_statement [Page 178], named_select_statement [Page 177], single_select_statement [Page 196], select_direct_statement:_searched [Page 196], select_direct_statement:_positioned [Page 197], select_ordered_statement:_searched [Page 197], select_ordered_statement:_positioned [Page 199], subtrans_statement [Page 208], update_statement [Page 169], identifier [Page 40]

Explanation

Variables specified in a routine [Page 146] can be assigned a value with the assignment statement.

Control Structures

The **IF statement** first evaluates the search condition [Page 107]. If this is fulfilled, the statement specified in the THEN branch is executed. Otherwise, the statement in the ELSE branch (if defined) is executed.

The **WHILE statement** enables statements to be repeated in response to certain conditions. The statement is executed as long as the specified search condition is true. The condition is checked, in particular, before the statement is executed for the first time. This means that the statement may not be executed at all. By specifying **BREAK**, you can exit the loop straight away, without checking the condition. If **CONTINUE** is specified in the loop, the condition is revaluated immediately and the loop is processed again or exited, depending on the result.

Troubleshooting

If an SQL error occurs in the statement list between **TRY** and **CATCH**, the system branches directly to the statement that follows CATCH. The actual troubleshooting routine can be programmed in this statement. If **CONTINUE EXECUTE** is executed here, the system jumps directly to the point after the statement that triggered the error.

The database procedure is interrupted immediately when the **STOP** function is invoked. The value of the first parameter of the STOP function is the return or error message that the application receives as the result of the database procedure call. An error text can also be returned.

SQL Statements (routine_sql_statement)

The tables in the SQL statements of the database procedure must always be complete, i.e. with the owner specified. In the case of SELECT statements, a full statement of the table name in the FROM clause [Page 185] is sufficient.

The CALL statement **cannot** be used in SAP DB version 7.2 and can only be used as of version 7.3.0.18 for SAP DB version 7.3.

Restrictions

The statement list must not contain more than 255 SQL statements.

The length of an SQL statement (routine sql statement) must not exceed approx. 8 KB.

DROP DBPROC statement

The DROP DBPROC statement drops a database procedure [Page 33].

Syntax

<drop_dbproc_statement> ::= DROP DBPROC <dbproc_name>

dbproc_name [Page 44]

Explanation

The specified database procedure name must identify an existing database procedure of the current user.

The metadata of the database procedure is dropped.

CREATE TRIGGER statement

The CREATE TRIGGER statement defines a trigger [Page 34] for a base table (see Table [Page 29]).

Syntax

```
<create_trigger_statement> ::= CREATE TRIGGER <trigger_name> FOR
<table_name>
AFTER <trigger_event,..> EXECUTE (<routine>) [WHENEVER <search_condition> ]

<trigger_event> :: INSERT | UPDATE [(<column_list>)] | DELETE
<column_list> ::= <column_name> | <column_list>,<column_name>
```

trigger_name [Page 51], table_name [Page 50], search_condition [Page 107], routine [Page 146], column_name [Page 49]

The trigger ensures that the hotel number in the `room` table is also changed when a hotel number is changed in the `hotel` table.

```
CREATE TRIGGER hotel_update FOR hotel AFTER UPDATE EXECUTE (
TRY
  IF NEW.hnr <> OLD.hnr
  THEN UPDATE reisen.raum SET hnr = :NEW.hnr WHERE hnr = :OLD.hnr;
CATCH
  IF $rc <> 100
  THEN STOP ($rc, 'unexpected error');
)
```

Explanation

A trigger is a special type of database procedure [Page 33] that is assigned to a base table. This database procedure cannot be executed explicitly with the CALL statement [Page 173], but rather automatically by SAP DB when defined events (`trigger_events`) for the table occur.

SAP DB provides a language (special SQL syntax that has been extended to include variables, control structures, and troubleshooting measures) that can be used to define database procedures and triggers.

The specified synonym name must identify an existing base table of the current user.

Trigger Event

The trigger event defines what triggers the trigger. The trigger is always invoked if the triggering event has been processed correctly.

INSERT: the INSERT trigger event causes the trigger to be executed for each row inserted in the table.

UPDATE: the UPDATE event causes the trigger to be executed for each modification made to a row in the table. If a column list is specified, the trigger is only called if one of the columns in the column list was modified.

DELETE: the DELETE trigger event causes the trigger to be executed for every row deleted from the table.

A maximum of one trigger can be defined for each trigger event in each table.

Trigger Routine

Each **INSERT trigger** implicitly has a corresponding variable `NEW.<column_name>` for each column in the table. When the trigger is executed, this variable has the value of the corresponding column in the inserted row. NEW is optional.

Each **UPDATE trigger** implicitly has a corresponding variable `NEW.<column_name>` and `OLD.<column_name>` for each column in the table. When the trigger is executed, the `OLD.<column_name>` variable has the value of the corresponding column in front of and `NEW.<column_name>` after the change in the row. NEW is optional.

Each **DELETE trigger** implicitly has a corresponding variable `OLD.<column_name>` for each column in the table. When the trigger is executed, this variable has the value of the corresponding column in the deleted row. OLD is optional.

:NEW and **:OLD** must always be used with a colon in SQL statements that are used in triggers and that belong to the `routine_sql_statement` [Page 147]s
(For example: `UPDATE reisen.raum SET hnr = :NEW.hnr WHERE hnr = :OLD.hnr`).

NEW and **OLD** must always be used **without** a colon in SQL statements that are used in triggers and that do not belong to the `routine_sql_statements`
(For example: `IF NEW.hnr <> OLD.hnr`).

See also:

routine [Page 146]

If the trigger is terminated by STOP with an error number not equal to zero, the entire SQL statement that triggered the trigger fails.

The SUBTRANS statement [Page 208] is not allowed in a trigger.

If a **WHENEVER** statement is specified, the trigger is only executed if the search condition [Page 107] is fulfilled. The condition must not contain a subquery [Page 189] nor any set function [Page 87]s.

DROP TRIGGER statement

A DROP TABLE statement deletes a trigger [Page 34] for a table.

Syntax

```
<drop_trigger_statement> ::= DROP TRIGGER <trigger_name> OF <table_name>
```

trigger_name [Page 51], table_name [Page 50]

Explanation

The specified table name must identify an existing table of the current user.

The specified trigger name must identify an existing trigger of the table.

The metadata of the trigger is dropped.

Authorization

SQL statements for authorization

CREATE USER statement [Page 151]	DROP USER statement [Page 155]	ALTER USER statement [Page 156] RENAME USER statement

		[Page 158] GRANT USER statement [Page 158]
CREATE USERGROUP statement [Page 153]	DROP USERGROUP statement [Page 155]	ALTER USERGROUP statement [Page 157] RENAME USERGROUP statement [Page 158] GRANT USERGROUP statement [Page 159]
CREATE ROLE statement [Page 160]	DROP ROLE statement [Page 160]	
ALTER PASSWORD statement [Page 159]	GRANT statement [Page 161]	REVOKE statement [Page 162]

CREATE USER statement

The CREATE USER statement defines a user [Page 30]. The existence and the properties of the user are recorded in the catalog in the form of metadata.

Syntax

```
<create_user_statement> ::= CREATE USER <user_name> PASSWORD <password>
[<user_mode>] [PERMLIMIT <unsigned_integer>] [TEMPLIMIT <unsigned_integer>]
[TIMEOUT <unsigned_integer>] [COSTWARNING <unsigned_integer>] [COSTLIMIT
<unsigned_integer>] [[NOT] EXCLUSIVE]
| CREATE USER <user_name> PASSWORD <password> LIKE <source_user>
| CREATE USER <user_name> PASSWORD <password> USERGROUP <usergroup_name>
```

user_name [Page 43], user_mode [Page 152], unsigned_integer [Page 38], usergroup_name [Page 43]

Explanation

The current user must be a DBA user. The user is the owner of the created user.

The specified user name must not be identical to the name of an existing user, usergroup, or role.

The password must be specified when an database session is started. It ensures that only authorized users can access the database system.

Unsigned integers must always be greater than 0.

User class (user_mode [Page 152])

If no user class or if STANDARD is specified, PERMLIMIT must not be used.

PERMLIMIT may be used if the user class DBA or RESOURCE was specified.

PERMLIMIT

If PERMILIMIT is specified, the disk space available for users' private tables is limited. This value is specified in units of 8 KB.

If PERMLIMIT is not specified, the user has unlimited space (within the limits of the sizes of the data devspaces specified during the installation) for storing private tables.

TEMPLIMIT

If PERMILIMIT is specified, the disk space available to this user for building temporary result tables, temporary base tables, and for execution plans is limited. This value is specified in units of 8 KB.

If TEMPLIMIT is not specified, the user has unlimited space (within the limits of the sizes of the data devspaces specified during the installation).

TIMEOUT

The TIMEOUT value defines the maximum time that may elapse between the completion of an SQL statement and the issuing of the next SQL statement. This value is the maximum value which can be specified as a TIMEOUT value in the CONNECT statement.

The TIMEOUT value is specified in seconds and must be between 30 and 32400.

COSTWARNING/COSTLIMIT

COSTWARNING and COSTLIMIT limit costs by preventing users from executing QUERY statements or INSERT statements in the form of INSERT...SELECT... beyond a specified degree of complexity.

Before these SQL statements are executed, the costs expected to result from this statement are estimated. This SELECT costs estimate can be output with the EXPLAIN statement [Page 201]. In interactive mode, the estimated SELECT cost value is compared with the COSTWARNING and COSTLIMIT values specified for the user.

The COSTWARNING and COSTLIMIT values are ignored with QUERY statements or INSERT statements in the form INSERT...SELECT... which are embedded in a programming language.

COSTWARNING: specifies the estimated SELECT cost value beyond which the user receives a warning. In this case, the user is asked whether the SQL statement is to be executed.

COSTLIMIT: specifies the estimated SELECT cost value beyond which the SQL statement is not executed.

The COSTLIMIT value must be greater than the COSTWARNING value.

EXCLUSIVE

EXCLUSIVE: prevents the user from opening two different database sessions simultaneously.

NOT EXCLUSIVE: allows the user to open several database sessions simultaneously.

If the EXCLUSIVE condition is not specified, EXCLUSIVE is assumed implicitly (without NOT).

LIKE

The current user must have owner authorization over the source user.

If the source user **is not a member of a usergroup**, the new user receives the same user class and values for PERMLIMIT, TEMPLIMIT, TIMEOUT, COSTWARNING, COSTLIMIT, and EXCLUSIVE as the source user. The new user receives all the privileges that the source user was granted by other users.

If the source user is a **member of a usergroup**, a new member is created in this usergroup with the new user name.

USERGROUP

The user issuing the SQL statement must be the owner of the specified usergroup. The new user then becomes a member of this usergroup.

User mode

User mode is used to specify the user class or status of the defined user when a user [Page 30] is created (CREATE USER statement [Page 151]). The user class specifies the operations that the defined user can execute.

Syntax

```
<user_mode> ::= DBA | RESOURCE | STANDARD
```

Explanation

If no user class is specified, the STANDARD class is assumed implicitly.

DBA

The specified user is authorized to define private data and grant privileges for this data to other users. The user can define additional users. DBA status may only be assigned by the SYSDBA that was created when the database system was installed.

RESOURCE

The specified user is authorized to define private data and grant privileges for these objects to other users.

STANDARD

The specified user can only access private data, which was created by other users and for which he or she has the appropriate privileges, as well as view tables, synonyms, and temporary tables.

Dependencies

The user classes are hierarchically ordered as follows:

- The user class RESOURCE encompasses all the rights of STANDARD users.
- The user class DBA encompasses all the rights of RESOURCES users.
- The SYSDBA user can create DBA users. He or she has owner rights over all users. Otherwise, the SYSDBA has the same function and the same rights as a DBA user, i.e. whenever a DBA user is allowed to execute an SQL statement, this also applies to a SYSDBA user.

See also:

Users and user groups [Page 30]

usergroup_mode [Page 154]

CREATE USERGROUP statement

The CREATE USERGROUP statement defines a user group [Page 30].

Syntax

```
<create_usergroup_statement> ::= CREATE USERGROUP <usergroup_name>
[<usergroup_mode>] [PERMLIMIT <unsigned_integer>] [TEMPLIMIT
<unsigned_integer>]
[TIMEOUT <unsigned_integer>] [COSTWARNING <unsigned_integer>] [COSTLIMIT
<unsigned_integer>] [[NOT] EXCLUSIVE]
```

usergroup_name [Page 43], usergroup_mode [Page 154], unsigned_integer [Page 38]

Explanation

The current user must be a DBA user.

The specified usergroup name must not be identical to the name of an existing user, usergroup, or role.

Several users who are members of this usergroup can be defined using CREATE USER statement [Page 151]. All private objects created by members of the usergroup are identified by the usergroup name. The owner of a private object is the group, not the user who created the object. Each user can work with any private object of the group, as if this user were the owner of the object. Privileges can only be granted or revoked from the group. A privilege cannot be granted or revoked from a single member of the group.

Unsigned integers must always be greater than 0.

User class of the usergroups (usergroup_mode [Page 154])

If no user class is specified for the usergroup or if STANDARD is specified, PERMLIMIT must not be used.

PERMLIMIT

If PERMILIMIT is specified, the disk space available for private tables of users in this usergroup is limited. This value is specified in units of 8 KB.

If PERMLIMIT is not specified, the user has unlimited space (within the limits of the sizes of the data devspaces specified during the installation) for storing private tables.

TEMPLIMIT

If PERMILIMIT is specified, the disk space available to users in this usergroup for building temporary result tables, temporary base tables, and for execution plans is limited. This value is specified in units of 8 KB.

If TEMPLIMIT is not specified, the user has unlimited space (within the limits of the sizes of the data devspaces specified during the installation).

TIMEOUT

The TIMEOUT value defines the maximum time that may elapse between the completion of an SQL statement and the issuing of the next SQL statement. This value is the maximum value which can be specified as a TIMEOUT value in the CONNECT statement.

The TIMEOUT value is specified in seconds and must be between 0 and 32400.

COSTWARNING/COSTLIMIT

COSTWARNING and COSTLIMIT limit costs by preventing users in this usergroup from executing QUERY statements or INSERT statements in the form of INSERT...SELECT... beyond a specified degree of complexity.

Before these SQL statements are executed, the costs expected to result from this statement are estimated. This SELECT costs estimate can be output with the EXPLAIN statement [Page 201]. In interactive mode, the estimated SELECT cost value is compared with the COSTWARNING and COSTLIMIT values specified for the user.

The COSTWARNING and COSTLIMIT values are ignored with QUERY statements or INSERT statements in the form INSERT...SELECT... which are embedded in a programming language.

COSTWARNING: specifies the estimated SELECT cost value beyond which the user receives a warning. In this case, the user is asked whether the SQL statement is to be executed.

COSTLIMIT: specifies the estimated SELECT cost value beyond which the SQL statement is not executed.

The COSTLIMIT value must be greater than the COSTWARNING value.

EXCLUSIVE

EXCLUSIVE: prevents users in this usergroup from opening two different database sessions simultaneously.

NOT EXCLUSIVE: allows the user to open several database sessions simultaneously.

If the EXCLUSIVE condition is not specified, EXCLUSIVE is assumed implicitly (without NOT).

Usergroup name

Usergroup mode is used to specify the user class or status of the defined usergroup when a usergroup [Page 30] is created (CREATE USERGROUP statement [Page 153]).

Syntax

```
<usergroup_mode> ::= RESOURCE | STANDARD
```

Explanation

If no user class is specified, the STANDARD class is assumed implicitly.

RESOURCE

A user in the specified usergroup is authorized to define private data and grant privileges for these objects to other users.

STANDARD

A user in the specified usergroup can only access private data, which was created by other users and for which he or she has the appropriate privileges, as well as view tables, synonyms, and temporary tables.

The user class RESOURCE encompasses all the rights of STANDARD users.

See also:

Users and user groups [Page 30]

user_mode [Page 152]

DROP USER statement

A DROP USER statement drops a user [Page 30] definition. The metadata of the user to be dropped is dropped from the catalog.

Syntax

```
<drop_user_statement> ::= DROP USER <user_name> [<cascade_option>]
```

user_name [Page 43], cascade_option [Page 127]

Explanation

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

The specified user must not be logged onto the database system when the DROP USER statement is executed.

- If the user to be dropped does not belong to a usergroup and is the owner of synonyms or tables, and if the CASCADE option **RESTRICT** was specified, the DROP USER statement fails.
- If **no** CASCADE option or the CASCADE option **CASCADE** is specified, all the synonyms and tables of the user to be dropped, as well as all indexes, privileges, view tables, etc. based on these objects, are dropped.

Dropping a user with the user class [Page 152] DBA does not affect any users that were created by this user. The SYSDBA user then becomes the new owner of these users.

DROP USERGROUP statement

The DROP USERGROUP statement drops a usergroup [Page 30] definition. The metadata of the usergroup to be dropped is dropped from the catalog.

Syntax

```
<drop_usergroup_statement> ::= DROP USERGROUP <usergroup_name>
[<cascade_option>]
```

usergroup_name [Page 43], cascade_option [Page 127]

Explanation

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

The users in this usergroup must not be logged onto the database system when the DROP USERGROUP statement is executed.

- If the usergroup to be dropped does not belong to a usergroup and is the owner of synonyms or tables, and if the CASCADE option **RESTRICT** was specified, the DROP USERGROUP statement fails.

- If **no** CASCADE option or the CASCADE option **CASCADE** is specified, all the synonyms and tables of the usergroup to be dropped, as well as all indexes, privileges, view tables, etc. based on these objects, are dropped.

ALTER USER statement

The ALTER USER statement alters the properties assigned to a user [Page 30].

Syntax

```
<alter_user_statement> ::= ALTER USER <user_name> [<user_mode>]
[PERMLIMIT <unsigned_integer> | PERMLIMIT NULL]
[TEMPLIMIT <unsigned_integer> | TEMPLIMIT NULL]
[TIMEOUT <unsigned_integer> | TIMEOUT NULL]
[COSTWARNING <unsigned_integer> | COSTWARNING NULL]
[COSTLIMIT <unsigned_integer> | COSTLIMIT NULL]
[DEFAULT ROLE ALL [EXCEPT <role_name>]
| DEFAULT ROLE NONE
| DEFAULT ROLE <role_name> [IDENTIFIED BY <password>]]
[[NOT] EXCLUSIVE]
```

user_name [Page 43], user_mode [Page 152], unsigned_integer [Page 38], role_name [Page 48], password [Page 46]

Explanation

At least one of the optional clauses must be specified.

The specified user name must identify a defined user, who is not a member of a usergroup.

The current user must have owner authorization over the user whose properties are to be altered.

The specified user must not be logged onto the database system when the ALTER USER statement is executed.

User class (user_mode)

- **DBA**: specifies that the user is to be assigned the user class DBA. The DBA user class can only be granted by the SYSDBA.
- **RESOURCE**: specifies that the user is to be assigned the user class RESOURCE. If the user had DBA status before, owner authorization for all users he or she created is revoked. The SYSDBA user then becomes the new owner.
- **STANDARD**: specifies that the user is removed from the current user class and loses the right to create base tables. All the base tables created by the user are dropped.
- **No user class**: if no user class is specified, the user class remains unchanged.

NULL

If the NULL value is specified, the value defined previously is cancelled.

DEFAULT ROLE

DEFAULT ROLE defines which of the roles assigned to the user is activated automatically when a session is opened.

- **ALL**: all the roles assigned to the user are activated when a session is opened. EXCEPT can be used to exclude specified roles from activation.
- **NONE**: none of the roles is activated when a user session is opened.
- **Role name specified**: the roles specified here must exist and be assigned to the user. They are automatically activated when a user session is opened.

PERMLIMIT, TEMPLIMIT, TIMEOUT, COSTWARNING, COSTLIMIT, and [NOT] EXCLUSIVE are described under the CREATE USER statement [Page 151].

The PERMLIMIT specification may only be altered if the new value is greater than the current space requirement of all private tables.

ALTER USERGROUP statement

The ALTER USERGROUP statement alters the properties assigned to a usergroup [Page 30].

Syntax

```
<alter_usergroup_statement> ::= ALTER USERGROUP <usergroup_name>
[<usergroup_mode>]
[PERMLIMIT <unsigned_integer> | PERMLIMIT NULL]
[TEMPLIMIT <unsigned_integer> | TEMPLIMIT NULL]
[TIMEOUT <unsigned_integer> | TIMEOUT NULL]
[COSTWARNING <unsigned_integer> | COSTWARNING NULL] [COSTLIMIT
<unsigned_integer> | COSTLIMIT NULL]
[DEFAULT ROLE ALL [EXCEPT <role_name>]
| DEFAULT ROLE NONE
| DEFAULT ROLE <role_name> [IDENTIFIED BY <password>]]
[[NOT] EXCLUSIVE]
```

usergroup_name [Page 43], usergroup_mode [Page 154], unsigned_integer [Page 38]

Explanation

At least one of the optional clauses must be specified.

The specified usergroup must identify a defined usergroup.

The current user must have owner authorization over the usergroup whose properties are to be altered.

The members of the specified usergroup must not be logged onto the database system when the ALTER USERGROUP statement is executed.

User class of the usergroup (usergroup_mode)

- **RESOURCE**: specifies that the usergroup is to be assigned the user class RESOURCE.
- **STANDARD**: specifies that the usergroup is removed from the current user class and loses the right to create base tables. All the base tables created by the usergroup are dropped.
- **No user class**: if no user class is specified, the user class remains unchanged.

NULL

If the NULL value is specified, the value defined previously is cancelled.

DEFAULT ROLE

DEFAULT ROLE defines which of the roles assigned to the usergroup is activated automatically when a session is opened by a group member.

- **ALL**: all the roles assigned to the usergroup are activated when a session is opened. EXCEPT can be used to exclude specified roles from activation.
- **NONE**: none of the roles is activated when a session is opened by a member of the usergroup.
- **Role name specified**: the roles specified here must exist and be assigned to the usergroup. They are automatically activated when a session of a group member is opened.

PERMLIMIT, TEMPLIMIT, TIMEOUT, COSTWARNING, COSTLIMIT, and [NOT] EXCLUSIVE are described under the CREATE USERGROUP statement [Page 153].

The PERMLIMIT specification may only be altered if the new value is greater than the current space requirement of all private tables.

RENAME USER statement

The RENAME USER statement changes the name of a user [Page 30].

Syntax

```
<rename_user_statement> ::= RENAME USER <user_name> TO <new_user_name>
```

user_name [Page 43]

Explanation

The user to be modified must exist. The user name must identify the current user or a user over whom the current user has the owner privilege.

The new user name must not be identical to the name of an existing user, usergroup, or role.

If the name of the user to be modified is different from that of the current user, the user that is to be modified must not be logged onto the database system when the RENAME USER statement is executed. Otherwise, the current transaction is terminated with COMMIT before executing the RENAME USER statement is executed.

The database system automatically adapts the objects that are dependent on the modified user to the new user name.

RENAME USERGROUP statement

The RENAME USERGROUP statement changes the name of a usergroup [Page 30].

Syntax

```
<rename_usergroup_statement> ::= RENAME USERGROUP <usergroup_name> TO
<new_usergroup_name>
```

usergroup_name [Page 43]

Explanation

The usergroup to be modified must exist. The name of the usergroup must identify a usergroup for which the current user has the owner privilege.

The new usergroup name must not be identical to that of an existing user, usergroup, or role.

The members of this usergroup must not be logged onto the database system when the RENAME USERGROUP statement is executed.

The database system automatically adapts the objects that are dependent on the modified usergroup to the new usergroup name.

GRANT USER statement

The GRANT USER statement grants another user the owner privilege that the SYSDBA or a DBA user has over a user [Page 30].

Syntax

```
<grant_user_statement> ::= GRANT USER <granted_users> [FROM <user_name>] TO
<user_name>

<granted_users> ::= <user_name>,...| *
```

user_name [Page 43]

Explanation

The current user must be a DBA user.

The user names specified after the FROM and TO keywords must be different and must identify DBA users. If `FROM <user_name>` is not specified, the current user is assumed implicitly.

The users specified after the GRANT USER keywords must exist and must not be a member of a usergroup. They must also have the user class [Page 152] RESOURCE or STANDARD. The FROM user must have the owner privilege for these users. If * is specified, the GRANT USER statement affects all users for which the FROM user has the owner privilege.

The FROM user grants the TO user the owner privileges which the FROM user has over the specified users. These rights are revoked from the FROM user. In particular, the TO user is granted the right to drop any specified user and to alter the user class and other properties of this user.

GRANT USERGROUP statement

The GRANT USERGROUP statement grants another user the owner privilege that the SYSDBA or a DBA user has over a usergroup [Page 30].

Syntax

```
<grant_usergroup_statement> ::= GRANT USERGROUP <granted_usergroups> [FROM
<user_name>] TO <user_name>

<granted_usergroups> ::= <usergroup_name>,...| *
```

user_name [Page 43], usergroup_name [Page 43]

Explanation

The current user must be a DBA user.

The user names specified after the FROM and TO keywords must be different and must identify DBA users. If `FROM <user name>` is not specified, the current user is assumed implicitly.

The usergroup name must identify a usergroups for which the FROM user has the owner privilege. An asterisk * stands for all usergroups for which the FROM user has the owner privilege.

The FROM user grants the TO user the owner authorization which the FROM user has over the specified usergroups. These rights are revoked from the FROM user. In particular, the TO user is granted the right to drop any usergroup, to alter the user class and properties of this usergroup, and to drop or create group members.

ALTER PASSWORD statement

The ALTER PASSWORD statement is required to alter a user's password.

Syntax

```
<alter_password_statement> ::= ALTER PASSWORD <old_password> TO
<new_password>
| ALTER PASSWORD <user_name> <new_password>

<old_password> ::= <password>
<new_password> ::= <password>
```

user_name [Page 43], password [Page 46]

Explanation

The old password must match the password entered in the catalog for the current user.

If the `user_name` is specified, the current user must be the SYSDBA.

The new password must be specified in the CONNECT statement [Page 204] the next time the user starts a session.

CREATE ROLE statement

The CREATE ROLE statement defines a role [Page 31].

Syntax

```
<create_role_statement> ::= CREATE ROLE <role_name> [IDENTIFIED BY
<password>]
```

role_name [Page 48], password [Page 46]

Explanation

A role combines a set of privileges [Page 31] that can be assigned to users, usergroups [Page 30], or other roles by specifying the role name in the GRANT statement [Page 161]. The role is empty initially after the CREATE ROLE statement has been executed. Privileges must be assigned to the role using the GRANT statement.

The existence and the properties of the role are recorded in the catalog in the form of metadata. The current user is the owner of the role.

The current user must be a DBA.

The role name must not be the same as the name of an existing role, user, or usergroup.

Roles can be assigned to a user or usergroup by executing the ALTER USER statement [Page 156] or ALTER USERGROUP statement [Page 157]. These roles are then activated when a session is opened. Alternatively, roles can be activated within a session by means of the SET statement [Page 206]. If a role is activated in a session, the current user of the session has all the privileges assigned to the role.

Note that roles are not active while data definition [Page 110] commands are being executed.

If a password is defined for the role, users who are assigned the role can only activate it by specifying the password in the SET statement.

DROP ROLE statement

The DROP ROLE statement drops a role [Page 31].

Syntax

```
<drop_role_statement> ::= DROP ROLE <role_name>
```

role_name [Page 48]

Explanation

The current user must be the owner of the role.

The metadata of the role to be dropped is dropped from the catalog.

GRANT statement

The GRANT statement assigns privileges [Page 31] for tables, individual columns and roles, the SELECT privilege for a sequence, and the execution privilege for a database procedure.

Syntax

```
<grant_statement> ::= GRANT <priv_spec>,... TO <grantee>,... [WITH GRANT
OPTION]
| GRANT EXECUTE ON <dbproc_name> TO <grantee>,...
| GRANT SELECT ON <sequence_name> TO <grantee>,... [WITH GRANT OPTION]
```

priv_spec [Page 161], grantee [Page 162], dbproc_name [Page 44], sequence_name [Page 49]

Explanation

The privileges in the privilege specification are assigned to the users [Page 30], user groups [Page 30], and roles [Page 31] specified in the grantee list.

WITH GRANT OPTION

Users or usergroups identified as grantees are allowed to pass on their privileges to other users. The current user must have the authorization to pass on these privileges.

The WITH GRANT OPTION cannot be specified if `grantee` identifies a role.

GRANT EXECUTE ON

GRANT EXECUTE ON allow the user identified by `grantee` to execute the specified database procedure [Page 33]. The current user must be the owner of the database procedure.

GRANT SELECT ON

GRANT SELECT ON allows the user identified by `grantee` to execute the specified sequence.

Privilege specification (priv_spec)

A privilege specification (`priv_spec`) defines a role or a set of privileges [Page 31] for specific tables.

Syntax

```
<priv_spec> ::= ALL [PRIV[ILEGES]] ON [TABLE] <table_name>,...
| <privilege>,... ON [TABLE] <table_name>,... | <role_name>
```

table_name [Page 50], privilege [Page 47], role_name [Page 48]

Explanation

These tables must not be temporary base tables.

The user [Page 30] must have the authorization to grant (GRANT statement [Page 161]) and revoke (REVOKE statement [Page 162]).privileges for the specified tables. For base tables, the owner of the table has this authorization.

In the case of view tables (see Table [Page 29]), the owner may not always be authorized to assign or revoke all privileges. The database determines the privileges that a user can assign or revoke for a view table when the table is created. The result depends on the type of table and on the user's privileges for the tables selected in the view table. The owner of a table can interrogate the privileges that he or she is allowed to grant or revoke by selecting the system table DOMAIN.PRIVILEGES.

A list of all the privileges that can be granted is provided in the privilege type [Page 47].

If a role is defined as a privilege specification, it must exist and the current user must be the owner of the role.

ALL [PRIV[ILEGES]]

All of the privileges that the user can grant for tables are granted (GRANT statement) or revoked (REVOKE statement) for the specified users, usergroups, and roles.

If a user who is not the owner of the table specifies ALL in a REVOKE statement, all of the privileges he or she has granted to the specified user for this table are revoked.

grantee

In `grantee` the user [Page 30] (or several users) or role [Page 31] is specified for which privileges are to be granted with the GRANT statement [Page 161] or revoked with the REVOKE statement [Page 162].

Syntax

```
<grantee> ::= PUBLIC | <user_name> | <usergroup_name> | <role_name>
```

user_name [Page 43], usergroup_name [Page 43], role_name [Page 48]

Explanation

A user in the `grantee` list must not be identical to the user name of the current user or the name of the owner of the table. A user in the `grantee` list must not denote a member of a usergroup [Page 30].

Roles

If a role is assigned to a user or usergroup, it extends the set of roles which can be activated for this user or usergroup. The user activates the role either with the SET statement [Page 206] or by including the role in the set of roles automatically activated when a session was opened with the ALTER USER statement [Page 156] or ALTER USERGROUP statement [Page 157].

A cycle may not be created when a role is assigned to a role, that is

- a role may not be assigned to itself.
- if a role R1 is assigned to a role R2, R2 may not be assigned to R1.
- if a role R1 is assigned to a role R2 and R2 is assigned to a role R3, R3 may not be assigned to either R2 or R1.
- etc.

PUBLIC

The listed privileges are granted to all users, both to current ones and to any created later.

A role cannot be assigned to PUBLIC.

REVOKE statement

The REVOKE statement revokes privileges [Page 31].

Syntax

```
<revoke statement> ::= REVOKE <priv spec>,... FROM <grantee>,...
[<cascade_option>]
| REVOKE EXECUTE ON <dbproc_name> FROM <grantee>,...
| REVOKE SELECT ON <sequence_name> FROM <grantee>,... [<cascade_option>]
```

priv_spec [Page 161], grantee [Page 162], cascade_option [Page 127], dbproc_name [Page 44], sequence_name [Page 49]

Explanation

The owner of a table can revoke the privileges granted for this table from any user [Page 30].

If a user is not the owner of the table, he may only revoke the privileges he has granted.

If the SELECT privilege was granted for a table any specified column names, REVOKE SELECT (<column name>,...) can be used to revoke the SELECT privilege (see privilege type [Page 47]) for the specified columns; the SELECT privilege for table columns that have not been specified remains unchanged. The same is true for the UPDATE, REFERENCES, and SELUPD privileges.

The REVOKE statement can cascade; i.e. revoking a privilege from one user can result in this privilege being revoked from other users who have received the privilege from the user in question.

Let U1, U2, and U3 be users.
U1 grants U2 the privilege set P WITH GRANT OPTION.
U1 grants U3 the privilege set P' (P'<=P).

If U1 revokes the privilege set P'' (P''<=P) from user U2, the privilege set (P'*P'') is revoked implicitly from user U3.

- Whenever the SELECT privilege is revoked from the owner of a view table for a selected that does not occur in the `table_expression` of the view definition (CREATE VIEW statement [Page 137]), the column defined by `select_column` is dropped from the view table.
  If this view table is used in the FROM clause [Page 185] of another view table, the described procedure is applied recursively to this view table.
- If the SELECT privilege is revoked from the owner of a view table for a column or table occurring in the `table_expression` of the view definition, the view table is dropped, along with all view tables (see Table [Page 29]), privileges [Page 31], and synonyms [Page 30] based on this view table, if no CASCADE option [Page 127] or the CASCADE option CASCADE is specified. The REVOKE statement will fail if the CASCADE option RESTRICT is specified.

REVOKE EXECUTE

If REVOKE EXECUTE is specified, the authorization to execute the database procedure [Page 33] is revoked from the user identified by `grantee`. The authorization for execution can only be revoked by the owner of the database procedure.

Data manipulation

The following sections provide an introduction to the data manipulation language (DML) used by the database system.

Every SQL statement that manipulates data implicitly sets an EXCLUSIVE lock (see transactions [Page 202]) for each inserted, updated, or deleted row.

Whenever a user's transaction holds too many row locks on a table, the database system attempts to convert these row locks into a table lock. If this causes collisions with other locks, further row locks are requested. This means that table locks are obtained without waiting periods. The limit beyond which the database system attempts to transform row locks into table locks depends on the installation parameter MAXLOCKS, which indicates the maximum number of possible lock entries allowed for a transaction.

SQL statements for data manipulation

INSERT statement [Page 164]	UPDATE statement [Page 169]	DELETE statement [Page 171]
NEXT STAMP statement [Page 173]	CALL statement [Page 173]	

INSERT statement

The INSERT statement creates new rows in a table [Page 29].

Syntax

```
<insert_statement> ::=
  INSERT [INTO] <table_name> [(<column_name>,...)] VALUES
(<insert_expression>,...) [<duplicates_clause>]
| INSERT [INTO] <table_name> [(<column_name>,...)] <query_expression>
[<duplicates_clause>]
| INSERT [INTO] <table_name> SET <set_insert_clause>,...
[<duplicates_clause>]

<insert_expression> ::= <extended_expression> | <subquery>
```

table_name [Page 50], column_name [Page 49], extended_expression [Page 168], duplicates_clause [Page 166], subquery [Page 189] query_expression [Page 179], set_insert_clause [Page 168]

Explanation

The table name must denote an existing base table, view table (see Table [Page 29]), or a synonym [Page 30].

If column names or a SET INSERT clause are specified, all of the specified column names must be columns in the table. If the table was defined without a key (i.e. if the SYSKEY column is created internally by the database), the SYSKEY column must not occur in the sequence of column names or in a SET INSERT clause. A column must not occur more than once in a sequence of column names or in more than one SET INSERT clause.

The user must have the INSERT privilege for the table identified by the table name. If the table name identifies a view table, even the owner of the view table may not have the INSERT privilege because the view table cannot be changed.

A specified column (identified by `column name` or the column name in the `set_insert_clause`) is a **target column**. Target columns can be specified in any order.

- If a column name and SET INSERT clause are not specified, the result is the same as if a sequence of columns were specified that contains all of the columns in the table in the order specified in the CREATE TABLE statement [Page 111] or CREATE VIEW statement [Page 137]. In this case, every table column defined by the user is a target column.
- The number of expressions (`extended_expressions`) must be equal to the number of target columns. The i^{th} expression is assigned to the i^{th} column name.
- Expressions (`extended_expression`) and subqueries (`subquery`) can be specified simultaneously.
- You can enter one or more subqueries, but no more than 64 subqueries.
- Values for any number of target columns can be supplied in a subquery.
- The specified subqueries may only supply a single result line.
- The number of selected columns [Page 183] specified in the QUERY expression must be identical to the number of target columns.
- All mandatory columns of the table identified by table name must be target columns.
- If the table name identifies a view table, rows are inserted into the base table(s), on which the view table is based. In this case, the target columns of the specified table name correspond to columns of the base tables, on which the view table is based. In the following paragraphs, the term target column always refers to the corresponding column in the base table.

Further information

- Data type of the target column and data type of the value to be inserted [Page 165]
- Join view table in INSERT statement [Page 166]

- QUERY expression in INSERT statement [Page 166]
- DUPLICATES clause [Page 166]
- Constraint definition in INSERT statement [Page 167]
- Trigger in INSERT statement [Page 167]
- Syntax extension of INSERT statement [Page 167]

If a QUERY expression is specified in the INSERT statement, none of the LONG columns may be a target column.

In the case of the INSERT statement, the third entry of SQLERRD in the SQLCA is set to the number of inserted rows.

If errors occur while inserting rows, the INSERT statement fails, leaving the table unmodified.

Data type of the target column and inserted value

Let C be a target column and v a value that is not equal to the NULL value.

v is to be inserted in C by means of an INSERT statement [Page 164].
v is to be used to modify C with an UPDATE statement [Page 169].

Target column C	Required value for v
Numeric column	Number within the permissible range of C. INSERT statement: if v is the result of a QUERY expression, decimal places are rounded off if necessary. UPDATE statement: if v is the result of an expression that does not comprise one single numeric literal [Page 37], decimal places are rounded off if necessary.
Alphanumeric column with the code attribute ASCII or EBCDIC	Character string [Page 22] whose length is not greater than the length attribute of C. Subsequent blanks are ignored when the length of v is calculated. 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 before it is assigned.
Alphanumeric column with the code attribute BYTE	Hexadecimal character string whose length is not greater than the length attribute of C. Subsequent binary zeros are ignored when the length of v is calculated. If the length of v is shorter than the length attribute of C, then v is lengthened by the corresponding number of binary zeros.
Data type DATE	Date value [Page 23] in the current date format.
Data type TIME	Time value [Page 23] in the current time format.
Data type TIMESTAMP	Timestamp value [Page 24] in the current timestamp format.
Data type BOOLEAN	One of the values TRUE or FALSE (BOOLEAN [Page 24])

Join View Table in INSERT Statement

If the table name does **not identify a join view table** in an INSERT statement [Page 164] (see CREATE VIEW statement [Page 137]), and a row containing the key of the row to be inserted already exists, the result will depend on the DUPLICATES clause [Page 166]. The INSERT statement will fail if no DUPLICATES clause is specified.

If the table name identifies a **join view table**, a row is inserted into each base table on which the view table is based. If the key table of the view table already contains a row with the key of the row to be inserted, the INSERT statement will fail. If any row in a base table, which is not the key table of the view table, already contains the key of the row to be inserted, the INSERT statement will fail if the row to be inserted does not match the existing row.

QUERY Expression in INSERT Statement

If a QUERY expression [Page 179] is specified in the INSERT statement [Page 164] , the specified table must not be a join view table.

The QUERY expression defines a result table (see result table name [Page 45]) whose i^{th} column is assigned to the i^{th} target column. A row is formed from each row in the result table and inserted in the base table. Each of these rows has the following contents:

- Each base table column that is a target column of the INSERT statement contains the value of the corresponding column in the current result table row.

If a **QUERY expression is not specified** in the INSERT statement, exactly one row is inserted in the specified table. The inserted row has the following contents:

- Each base table column that is a target column of the INSERT statement contains the value assigned to the respective target column.

The following still applies to the inserted row(s):

- All columns of the base table that are not target columns of the INSERT statement and for which a DEFAULT specification [Page 120] exists contain the DEFAULT value.
- All columns of the base table that are not target columns of the INSERT statement and for which no DEFAULT specification exists contain the NULL value [Page 22].

DUPLICATES clause

The DUPLICATES clause can be used to determine how key collisions are handled.

Syntax

```
<duplicates_clause> ::= REJECT DUPLICATES | IGNORE DUPLICATES | UPDATE
DUPLICATES
```

Explanation

SQL statements in which the DUPLICATES clause is used

CREATE TABLE statement [Page 111]

`REJECT DUPLICATES` or no DUPLICATES clause	The CREATE TABLE statement fails if key collisions occur.
`IGNORE DUPLICATES`	Any rows that key collisions on insertion are ignored.

UPDATE DUPLICATES	Any rows that key collisions on insertion overwrite the rows with which they collide.

INSERT statement [Page 164]

If there is already a row in the base table with the key of the row to be inserted, the following cases must be distinguished:

REJECT DUPLICATES or no DUPLICATES clause	The INSERT statement fails.
IGNORE DUPLICATES	The new row is not inserted and processing of the INSERT statement is continued.
UPDATE DUPLICATES	The existing row is overwritten by the new row and processing of the INSERT statement is continued.

If there is more than one key collision for the same key for an INSERT statement with UPDATE DUPLICATES and QUERY expression specification, it is impossible to predict the content of the respective base table row once the INSERT statement has been completed.

If, for an INSERT statement with IGNORE DUPLICATES and a QUERY expression, more than one row in the result table produces the same base table key, and if this key did not exist before in the base table, it is impossible to predict the row that will be inserted in the table.

If the table name specified in the INSERT statement identifies a table without a user-defined key, the DUPLICATES clause has no effect.

Constraint Definition in INSERT Statement

If CONSTRAINT definitions [Page 122] exist for base tables in which rows are to be inserted with the INSERT statement [Page 164], each row that is to be inserted is checked against the CONSTRAINT definition. The INSERT statement fails if this is not the case for at least one row.

If at least one of the base tables in which rows are to be inserted with the INSERT statement is the referencing table of a referential CONSTRAINT definition [Page 123], the database system checks each row to be inserted to determine whether the foreign key resulting from the row exists as a key or as a value of an index defined with UNIQUE (see CREATE INDEX statement [Page 141]) in the corresponding referenced table. The INSERT statement fails if this is not the case for at least one row.

Trigger in INSERT Statement

If triggers [Page 34] that are to be executed after an INSERT statement [Page 164] were defined for base tables in which rows are to be inserted with the INSERT statement, they are executed accordingly. The INSERT statement will fail if one of these triggers fails.

Syntax Extension of INSERT Statement

The INSERT statement [Page 164] creates new rows in a table [Page 29]. In SAP DB version 7.3.00.19, the **bold** elements have been added to the syntax of this statement.

Syntax

```
<insert statement> ::=
  INSERT [INTO] <table name> [(<column name>,...)] VALUES
(<extended expression>,...) [<duplicates_clause>]
| INSERT [INTO] <table_name> [(<column_name>,...)] VALUES
(<insert_expression>,...) [<duplicates_clause>]
| INSERT [INTO] <table_name> [(<column_name>,...)] <query expression>
[<duplicates clause>]
| INSERT [INTO] <table name> SET <set insert clause>,...
[<duplicates clause>]
```

```
<insert_expression> ::= <extended_expression> | <subquery>
```

Extended expression

An extended expression can be specified by means of an expression [Page 90] or one of the keywords DEFAULT or STAMP.

Syntax

```
<extended_expression> ::= <expression> | DEFAULT | STAMP
```

expression [Page 90]

Explanation

- Expression
  INSERT statement [Page 164]: an expression in an INSERT statement must not contain a column specification [Page 51].
  The value specified by a parameter specification [Page 51] in an expression is the value of the parameter identified by the specification. If an indicator parameter is specified with a negative value, the value defined by the parameter specification [Page 51] is a NULL value.
- Keyword DEFAULT
  DEFAULT denotes the value used as the DEFAULT for the column.
- STAMP key word
  The database system is able to generate unique values. This is a serial number that starts with X'000000000001'. The values are assigned in ascending order. It cannot be ensured that a sequence of values is uninterrupted. The STAMP key word supplies the next value generated by the database system.
  The STAMP keyword can be used in the INSERT statement [Page 164] or in the UPDATE statement [Page 169], but only for columns of the data type CHAR(n) BYTE with n>=8 (default specification [Page 120]).
  If the user wants to find out the generated value before it is applied to a column, he or she must use the following **SQL statement**:
  NEXT STAMP statement [Page 173]

See also:

SET INSERT condition [Page 168]

SET UPDATE condition [Page 171]

SET INSERT clause

SET INSERT clause

Syntax

```
<set_insert_clause> ::= <column_name> = <extended_value_spec>
```

column_name [Page 49], extended_value_spec [Page 52]

Explanation

SQL statements in which the SET INSERT clause is used

INSERT statement [Page 164]

UPDATE Statement

The UPDATE statement changes column values in table rows.

Syntax

```
<update_statement> ::=
  UPDATE [OF] <table_name> [<reference_name>] SET <set_update_clause>,...
[KEY <key_spec>,...] [WHERE <search_condition>]
| UPDATE [OF] <table_name> [<reference_name>] (<column_name>,...) VALUES
(<extended_value_spec>,...) [KEY <key_spec>,...] [WHERE <search_condition>]
| UPDATE [OF] <table_name> [<reference_name>] SET <set_update_clause>,...
WHERE CURRENT OF <result_table_name>
| UPDATE [OF] <table_name> [<reference_name>] (<column_name>,...) VALUES
(<extended_value_spec>,...) WHERE CURRENT OF <result_table_name>
```

table name [Page 50], reference name [Page 48], set update clause [Page 171], key spec [Page 55], search condition [Page 107], column name [Page 49], extended value spec [Page 52], result table name [Page 45]

Explanation

The table name must denote an existing base table, view table (see Table [Page 29]), or a synonym [Page 30].

Columns whose values are to be updated are called **target columns**. One or more target columns and new values for these columns are specified after the table name and reference name (if necessary).

- All target columns must identify columns in the specified table, and each target column may only be listed once.
- The number of specified values (extended_value_spec [Page 52]) must be identical to the number of target columns. The i^{th} value specification is assigned to the i^{th} target column.
- The current user must have the UPDATE privilege for each target column in the specified table. If the table name identifies a view table, even the owner of the view table may not be able to update column values because the view table is not updateable.
- If the specified table is a view table, column values are only updated in rows in the base tables on which the view table is based. In this case, the target columns of the specified table correspond to columns in the base tables on which the view table is based. In the following paragraphs, the term target column always refers to the corresponding column in the base table.

Data type of the target column and data type of the value to be inserted [Page 165]

The following specifications determine the rows in the table that are updated:

- Optional sequence of key specifications [Page 55] and optional search condition [Page 107]
  Key specification and no search condition: a row with the specified key values already exists. The corresponding values are assigned to the target columns in this row. No rows are updated if a row with the specified key values does not exist.
  Key specification and a search condition: a row containing the specified key values exists. The search condition is applied to this row. If the search condition is satisfied, the corresponding values are assigned to the target columns of this row. No rows are updated if a row with the specified key values does not exist or if a search condition applied to a row is not fulfilled.
  No key specification and a search condition: the search condition is applied to each row in the specified table. The corresponding values are assigned to the target columns of all rows that satisfy the search condition.

- If **CURRENT OF** is used, i.e. if the cursor position in the result table [Page 45] (`result_table_name`) is specified: no rows are updated if the cursor is not positioned on a row in the result table.
- If none of the above specifications were made, all of the rows in the specified table are updated.
- If no row is found that satisfies the conditions defined by the optional clauses, the following message appears: `100 row not found`.

Even values in key columns that were defined by a user in a CREATE TABLE statement [Page 111] or ALTER TABLE statement [Page 127] can be updated. The implicit key column SYSKEY, if created, cannot be updated.

If the table name specifies a join view table (see CREATE VIEW statement [Page 137]), columns may exist that can only be updated in conjunction with other columns (determine the column combination for a column or a join view table [Page 171]). To update the value of the relevant column, a value must be specified for all of the columns in the column combination. This is true of all target columns that fulfill the following conditions:

- The target columns are located in a base table that is not a key table of the join view table and does not have a 1 : 1 relationship with the key table of the join view table.
- The target columns are foreign key columns of a referential CONSTRAINT definition [Page 123] that is relevant for the join view table.

CURRENT OF

If CURRENT OF is specified, the table name in the FROM clause [Page 185] of the QUERY statement, with which the result table was built, must be identical to the table name in the UPDATE statement.

If CURRENT OF is specified and the cursor is positioned on a row in 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 result table row was formed. This procedure requires that the result table was specified with FOR UPDATE. It is impossible to predict whether or not the updated values in the corresponding row are visible the next time the same row of the result table is accessed.

Reasons for an UPDATE statement failure

If CONSTRAINT definitions [Page 122] exist for base tables in which rows were updated with the UPDATE statement, each row that was updated is checked against the CONSTRAINT definitions. The UPDATE statement fails if this is not the case for at least one modified row.

For each row in which the value of foreign key columns has been updated with the UPDATE statement, the database system checks whether the respective resulting foreign key exists as a key or as a value of an index defined with UNIQUE (see CREATE INDEX statement [Page 141]) in the corresponding referenced table. The UPDATE statement fails if this is the case for at least one modified row.

For each row in which the value of a referenced column of a referential CONSTRAINT definition [Page 123] is to be updated using the UPDATE statement, the database system checks whether there are rows in the corresponding foreign key table that contain the old column values as foreign keys. The UPDATE statement fails if this is the case for at least one row.

If triggers [Page 34] that are to be executed after an UPDATE statement were defined for base tables in which rows are to be updated with the UPDATE statement, they are executed accordingly. The UPDATE statement will fail if one of these triggers fails.

Further information

The update statement can only be used to assign a value to columns with 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 database tools.

In the case of the UPDATE statement, the third entry of SQLERRD in the SQLCA is set to the number of updated rows. Rows are also counted as updated when the old value was overwritten with a new but identical value.

If errors occur while rows are updated, the UPDATE statement fails, leaving the table unchanged.

SET UPDATE clause

SET UPDATE clause

Syntax

```
<set_update_clause> ::= <column_name> = <extended_expression>
| <column_name>,... = (<extended_expression>,...)
| (<column_name>,...) = (<extended_expression>,...)
| <column_name> = <subquery>
| (<column_name>,...) = <subquery>
```

column_name [Page 49], extended_expression [Page 168], subquery [Page 189]

Explanation

The expression of a SET UPDATE clause must not contain a set function [Page 87].

The subquery must produce a result table with at most one row. The number of columns must be equal to the number of target columns specified.

SQL statement in which the SET UPDATE clause is used

UPDATE statement [Page 169]

Column combination for a given column of a join view table

To determine the combination of columns for a given column v in the join view table V, use the following procedure:

1. Determine the base table T_j containing the column which corresponds to v.
2. Determine the unique table sequence Til...Tik that contains Tj.
3. Determine the last table Til in this sequence that has a 1:1 relationship with the key table.
4. The columns of V, which correspond to the foreign key columns of Til of the referential CONSTRAINT definition between Til and Til+I that is relevant for V, are elements of the column combination.
5. All columns of V which correspond to columns of the tables Til+I...Tik are elements of the column combination.

To update the column value of the column v in an UPDATE statement [Page 169], a value must be specified for each of the columns of the column combination.

DELETE statement

The DELETE statement deletes rows in a table.

Syntax

```
<delete_statement> ::=
  DELETE [FROM] <table_name> [<reference_name>] [KEY <key_spec>,...] [WHERE
<search_condition>]
| DELETE [FROM] <table_name> [<reference_name>] WHERE CURRENT OF
<result_table_name>
```

table name [Page 50], reference name [Page 48], key spec [Page 55], search condition [Page 107], result table name [Page 45]

Explanation

The table name must denote an existing base table, view table (see Table [Page 29]), or a synonym [Page 30].

The current user must have the DELETE privilege for the specified table. If the table name identifies a view table, even the owner of the view table may not have the DELETE privilege because the view table cannot be changed.

Table name identifies a view table: the rows of the tables on which the view tables are based are deleted.

Table name identifies a join view table: only the following rows are deleted:

- Rows in the key table of the join view table
- Rows in base tables on which the view table is based and that have a 1:1: relationship with the key table.

The following specifications determine the rows in the table that are deleted:

- Optional sequence of key specifications [Page 55] and optional search condition [Page 107]
  Key specification and no search condition: a row with the specified key values already exists. This row is deleted. No rows are deleted if a row with the specified key values does not exist.
  Key specification and a search condition: a row containing the specified key values exists. The search condition is applied to this row. If the search condition is satisfied, then the row is deleted. No rows are deleted if a row with the specified key values does not exist or if a search condition applied to a row is not fulfilled.
  No key specification and a search condition: the search condition is applied to each row in the specified table. All rows for which the search condition is satisfied are deleted.
- If **CURRENT OF** is used, i.e. if the cursor position in the result table [Page 45] (`result_table_name`) is specified: no rows are deleted if the cursor is not positioned on a row in the result table.
- If none of the above specifications were made, all of the rows in the specified table are deleted.
- If no row is found that satisfies the conditions defined by the optional clauses, the following message appears: `100 row not found`.

CURRENT OF

If CURRENT OF is specified, the table name in the FROM clause [Page 185] of the QUERY statement, with which the result table was built, must be identical to the table name in the DELETE statement.

If CURRENT OF is specified and the cursor is positioned on a row in the result table, the corresponding row is deleted. The corresponding row is the row of the table specified in the FROM clause of the QUERY statement, from which the result table row was formed. This procedure requires that the result table was specified with FOR UPDATE. Afterwards, the cursor is positioned behind the result table row. It is impossible to predict whether or not the updated values in the corresponding row are visible the next time the same row of the result table is accessed.

DELETE rule

For each row deleted in the course of the DELETE statement which originates from a referenced table of at least one referential CONSTRAINT definition [Page 123], one of the following actions is carried out - depending on the DELETE rule [Page 124] of the referential constraint definition:

- DELETE CASCADE: all matching rows in the corresponding foreign key table are deleted.
- DELETE RESTRICT: if there are matching rows in the corresponding foreign key table, the DELETE statement fails.
- DELETE SET NULL: the NULL value is assigned to the respective foreign key columns of all matching rows in the corresponding foreign key table.
- DELETE SET DEFAULT: the DEFAULT value that was set with a DEFAULT specification [Page 120] is assigned to the respective foreign key columns of all matching rows in the corresponding foreign key table.

Trigger

If triggers [Page 34] that are to be executed after a DELETE statement were defined for base tables from which rows are to be deleted with the DELETE statement, they are executed accordingly. The DELETE statement will fail if one of these triggers fails.

Further information

In the case of the DELETE statement, the third entry of SQLERRD in the SQLCA is set to the number of deleted rows. If this counter has the value -1, either a significant part of the table or the entire table was deleted by the DELETE statement.

If errors occur in the course of the DELETE statement, the statement fails, leaving the table unchanged.

NEXT STAMP statement

The NEXT STAMP statement supplies a unique key that was generated by the database system.

Syntax

```
<next_stamp_statement> ::= NEXT STAMP [INTO] <parameter_name>
```

parameter_name [Page 46]

Explanation

The database system is able to generate unique values. This is a serial number that starts with X'0000000000001'. The values are assigned in ascending order. It cannot be ensured that a sequence of values is uninterrupted. These values can be stored in a column with the data type CHAR [Page 115](n) BYTE with n>=8.

The NEXT STAMP statement assigns the next key generated by the database system to the variable denoted by `parameter name`.

The NEXT STAMP statement can only be embedded in a programming language and cannot be used in interactive mode.

The keyword STAMP can be also used in an INSERT statement [Page 164] or an UPDATE statement [Page 169] if the next value is to be generated by the database system and stored in a column without the user knowing the value.

CALL statement

The CALL statement causes a database procedure [Page 33] to be executed.

Syntax

```
<call_statement> ::= CALL <dbproc_name> [(<expression>,...)] [WITH COMMIT]
```

dbproc_name [Page 44], expression [Page 90]

Explanation

The specified database procedure name must identify an existing database procedure.

The current user must have the EXECUTE privilege for the database procedure.

The number of expressions must be equal to the number of formal parameters for the database procedure.

The data type of the i^{th} expression must be compatible with the data type of the i^{th} formal parameter for the database procedure.

If the i^{th} formal parameter for the database procedure has the OUT or INOUT mode (see CREATE DBPROC statement [Page 145]), the corresponding expression must be a parameter specification [Page 51].

WITH COMMIT

If WITH COMMIT is specified, the current transaction [Page 32] is terminated with COMMIT after the database procedure has been executed correctly. If execution of the database procedure fails, the current transaction is terminated with ROLLBACK.

Data query

The following sections provide an introduction to the query language (also referred to as the data retrieval language) used by the database system.

SQL statements for data queries

QUERY statement [Page 174]	SINGLE SELECT statement [Page 196]	EXPLAIN statement [Page 201]
SELECT DIRECT statement: searched [Page 196]	SELECT DIRECT statement: positioned [Page 197]	
SELECT ORDERED statement: searched [Page 197]	SELECT ORDERED statement: positioned [Page 199]	
OPEN CURSOR statement [Page 192]	FETCH statement [Page 193]	CLOSE statement [Page 195]

QUERY statement

A QUERY statement specifies a result table that can be ordered (see result table name [Page 45]). There are four different ways of formulating a QUERY statement.

Syntax

```
<query_statement> ::= <declare_cursor_statement> |
<recursive_declare_cursor_statement>
| <named_select_statement> | <select_statement>
```

declare_cursor_statement [Page 175], recursive_declare_cursor_statement [Page 176], named_select_statement [Page 177], select_statement [Page 178]

Explanation

A QUERY statement generates a named/unnamed result table [Page 175].

A distinction is made between the following QUERY statements:

Basic types of QUERY statement	
DECLARE CURSOR statement	A named result table is defined. The table is generated with an OPEN CURSOR statement [Page 192].

Recursive DECLARE CURSOR statement	This statement can be used to generate bills of material.
SELECT statement (`named_select_statement`)	A named result table is defined and generated.
SELECT statement (`select_statement`)	An unnamed result table is defined and generated.

The SELECT statement (`named_select_statement`) and SELECT statement (`select_statement`) are subject to the rules that were specified for the DECLARE CURSOR statement [Page 175] and those that were specified for the OPEN CURSOR statement.

The order of rows in the result table depends on the internal search strategies of the system and is arbitrary. The only reliable means of sorting the result rows is to specify an ORDER clause [Page 190].

Updateable result table

A result table or the underlying base tables are updateable if the QUERY statement satisfies the following conditions:

- See the section entitled "Updateable result table" in the SELECT statement (named_select_statement) [Page 177] or SELECT statement (select_statement) [Page 178]
- The result table is a named result table, i.e. it must not have been generated by a SELECT statement (`select_statement`).

Named/unnamed result table

The difference between a named result table and an unnamed result table (see result table name [Page 45]) is that the unnamed result table cannot be specified in a FROM clause [Page 185] or in CURRENT OF <`result_table_name`> of a subsequent SQL statement.

QUERY statement	
DECLARE CURSOR statement (declare_cursor_statement) [Page 175]	A named result table is generated. The column names of a result table defined by this QUERY statement do not have to be unique.
SELECT statement (select_statement) [Page 178]	A unnamed result table is generated. The column names of a result table defined by this QUERY statement do not have to be unique.
SELECT statement (named_select_statement) [Page 177]	A named result table is generated. The column names of a result table generated by this QUERY statement must be unique.

DECLARE CURSOR statement

The DECLARE CURSOR statement defines a named result table (see named/unnamed result table [Page 175]) with the name `result_table_name`.

Syntax

```
<declare_cursor_statement> ::= DECLARE <result_table_name> CURSOR FOR
<select_statement>
```

result_table_name [Page 45], select_statement [Page 178]

Explanation

An OPEN CURSOR statement [Page 192] with the name of the result table is required to actually generate the result table defined with a DECLARE CURSOR statement.

See also:

SELECT statement (select_statement) [Page 178]

Recursive DECLARE CURSOR statement

The recursive DECLARE CURSOR statement can be used to receive bills of material by means of a command.

Syntax

```
<recursive_declare_cursor_statement> ::= DECLARE <result_table_name> CURSOR
FOR
WITH RECURSIVE <reference_name> (<alias_name>,...) AS
(<initial_select> UNION ALL <recursive_select>) <final_select>

<initial_select> ::= <query_spec>
<recursive_select> ::= <query_spec>
<final_select> ::= <select_statement>
```

result_table_name [Page 45], reference_name [Page 48], alias_name [Page 43], query_spec [Page 182], select_statement [Page 178]

```
DECLARE C CURSOR FOR
WITH RECURSIVE PX (MAJOR, MINOR, NUMBER, MAINMAJOR) AS
  (SELECT W,X,Y,W FROM T WHERE W = 'aaa' UNION ALL
   SELECT W,X,Y,MAINMAJOR FROM T, PX WHERE MINOR = T.W)
 SELECT MAINMAJOR,MINOR,NUMBER FROM PX ORDER BY NUMBER
```

Explanation

- The QUERY specification [Page 182] **`initial_select`** is executed and the result is entered in a temporary result table whose name is defined by specifying the reference name. The column names contained in it receive the names from the list of alias names. The number of output columns in the QUERY specification must be identical to the number of alias names.
- The QUERY specification **`recursive select`** should comprise a SELECT statement that contains at least the reference name in the FROM clause [Page 185] and one JOIN predicate [Page 99] between this table and a different table from the FROM clause.
  The QUERY specification `recursive_select` is repeated until it does not produce a result. The respective results are (logically) entered in the temporary result table whose name is defined by the reference name. This table is extended continuously. It is ensured, however, that the results of the n^{th} execution are used for the $n+1^{th}$ execution to avoid an endless loop.
- The SELECT statement `final_select` must only contain one QUERY expression that comprises a QUERY specification [Page 182].
  This is a SELECT statement across the table with the specified reference name in which the following elements can be used: set function names [Page 89], GROUP clause [Page 188], HAVING clause [Page 189], ORDER clause [Page 190], LOCK option [Page 191]

If a result table name [Page 45] with the specified reference name existed before the recursive DECLARE CURSOR statement was executed, the corresponding cursor is closed implicitly.

SELECT statement (named_select_statement)

A SELECT statement (`named_select_statement`) defines and creates a result table with the name `result_table_name` (see named/unnamed result table [Page 175]).

Syntax

```
<named_select_statement> ::= <named_query_expression>
[<order_clause>] [<update_clause>] [<lock_option>] [FOR REUSE]
```

named_query_expression [Page 181], order_clause [Page 190], update_clause [Page 191], lock_option [Page 191]

Explanation

An OPEN CURSOR statement [Page 192] is not permitted for result tables created with this SELECT statement.

The SELECT statement (`named_select_statement`) is subject to the rules that were specified for the DECLARE CURSOR statement [Page 175] and those that were specified for the OPEN CURSOR statement.

Depending on the search strategy, either all the rows in the result table are searched when the SELECT statement (`named_select_statement`) is executed and the result table is physically generated, or each next result table row is searched when a FETCH statement [Page 193] is executed, without being physically stored. This must be taken into account for the time behavior of FETCH statements.

Updateable result table

A result table or the underlying base tables are updateable if the QUERY statement satisfies the following conditions:

- The QUERY expression (`named_query_expression`) must only comprise one QUERY specification (named_query_spec) [Page 184].
- Only one base table or one updateable view table may be specified in the FROM clause [Page 185] of the QUERY specification (`named_query_spec`).
- The DISTINCT keyword (see DISTINCT specification [Page 183]), a GROUP clause [Page 188], or HAVING clause [Page 189] must **not** be specified.
- Expressions must not contain a set function (set_function_spec) [Page 87].
- See also the section entitled "Updateable result table" under QUERY statement [Page 174].

ORDER clause

The ORDER clause [Page 190] specifies a sort sequence for a result table.

UPDATE clause

An UPDATE clause [Page 191] can only be specified for updateable result tables. For updateable result tables, a position within a particular result table always corresponds to a position in the underlying tables and thus, ultimately, to a position in one or more base tables.

If an UPDATE clause was specified, the base tables can be updated using the position in the result table (`CURRENT OF <result_table_name>`) by means of an UPDATE statement [Page 169] or a DELETE statement [Page 171]. The position in a base table can be used to issue a SELECT DIRECT statement [Page 197] or a SELECT ORDERED statement [Page 199], or a LOCK statement [Page 209] can be used to request a lock for the row concerned in each base table involved.

LOCK option

The LOCK option [Page 191] determines which locks are to be set on the read rows.

FOR REUSE

If the result table is to be specified in the FROM clause of a subsequent QUERY statement [Page 174], the table should be specified with FOR REUSE keywords. If FOR REUSE is not specified, the reusability of the result table depends on internal system strategies.

Since specifying FOR REUSE increases the response times of some QUERY statements, it should only be specified if it is required to reuse the result table.

See also:

SELECT statement (select_statement) [Page 178]

SELECT statement (select_statement)

A SELECT statement (`select_statement`) defines and creates an unnamed result table (see named/unnamed result table [Page 175]).

Syntax

```
<select_statement> ::= <query_expression> [<order_clause>]
[<update_clause>] [<lock_option>] [FOR REUSE]
```

query_expression [Page 179], order_clause [Page 190], update_clause [Page 191], lock_option [Page 191]

Explanation

An OPEN CURSOR statement [Page 192] is not permitted for result tables created with this SELECT statement.

The SELECT statement (`select_statement`) is subject to the rules that were specified for the DECLARE CURSOR statement [Page 175] and those that were specified for the OPEN CURSOR statement.

Depending on the search strategy, either all the rows in the result table are searched when the SELECT statement (`select_statement`) is executed and the result table is physically generated, or each next result table row is searched when a FETCH statement [Page 193] is executed, without being physically stored. This must be taken into account for the time behavior of FETCH statements.

Updateable result table

A result table or the underlying base tables are updateable if the QUERY statement satisfies the following conditions:

- The QUERY statement [Page 174] comprises a DECLARE CURSOR statement.
- The QUERY expression (`query_expression`) must only comprise one QUERY specification (query_spec) [Page 182].
- Only one base table or one updateable view table may be specified in the FROM clause [Page 185] of the QUERY specification (`query_spec`).
- The DISTINCT keyword (see DISTINCT specification [Page 183]), a GROUP clause [Page 188], or HAVING clause [Page 189] must **not** be specified.
- Expressions must not contain a set function (set_function_spec) [Page 87].
- See also the section entitled "Updateable result table" under QUERY statement [Page 174].

ORDER clause

The ORDER clause [Page 190] specifies a sort sequence for a result table.

UPDATE clause

An UPDATE clause [Page 191] can only be specified for updateable result tables. For updateable result tables, a position within a particular result table always corresponds to a position in the underlying tables and thus, ultimately, to a position in one or more base tables.

If an UPDATE clause was specified, the base tables can be updated using the position in the result table (`CURRENT OF <result_table_name>`) by means of an UPDATE statement [Page 169] or a DELETE statement [Page 171]. The position in a base table can be used to issue a SELECT DIRECT statement [Page 197] or a SELECT ORDERED statement [Page 199], or a LOCK statement [Page 209] can be used to request a lock for the row concerned in each base table involved.

LOCK option

The LOCK option [Page 191] determines which locks are to be set on the read rows.

FOR REUSE

If the result table is to be specified in the FROM clause of a subsequent QUERY statement [Page 174], the table should be specified with FOR REUSE keywords. If FOR REUSE is not specified, the reusability of the result table depends on internal system strategies.

Since specifying FOR REUSE increases the response times of some QUERY statements, it should only be specified if it is required to reuse the result table.

See also:

SELECT statement (named_select_statement) [Page 177]

QUERY expression (query expression)

QUERY expressions are required to generate an unordered result table in a SELECT statement [Page 178].

Syntax

```
<query_expression> ::= <query_term> | <query_expression> UNION [ALL]
<query_term> | <query_expression> EXCEPT [ALL] <query_term>
```

query_term [Page 180]

Explanation

If the QUERY expressions consists of only one QUERY specification (query spec) [Page 182] (specified in `query term`), the result of the expression is the unchanged result of the QUERY specification.

If a QUERY expression consists of more than one QUERY specification, the number of selected columns in all QUERY specifications of the QUERY expression must be the same. The respective i^{th} selected columns of the QUERY specifications must be comparable.

Column type (select column)	
Numeric columns	Are comparable. If all i^{th} selected columns are numeric columns, the i^{th} column of the result table is a numeric column.
Alphanumerical column, code attribute [Page 24] BYTE	Are comparable.
Alphanumerical column, code attribute ASCII, EBCDIC, UNICODE	Are comparable. Are also comparable with date, time, and timestamp values.
All i^{th} columns are date values [Page 23]	The i^{th} column of the result table is a date value.

All i[th] columns are time values [Page 23]	The i[th] column of the result table is a time value.
All i[th] columns are timestamp values [Page 24]	The i[th] column of the result table is a timestamp value.
Columns of the type BOOLEAN [Page 24]	Are comparable.
All i[th] columns are of the type BOOLEAN	The i[th] column of the result table is of the type BOOLEAN.
Columns of any other data type (not mentioned above)	The i[th] 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.

Column names in the result table

The names of the result table columns are formed from the names of the selected columns of the first QUERY specification.

Let T1 be the left operand of UNION, EXCEPT, or INTERSECT (defined in `query term`). 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 have identical values in each column. NULL values [Page 22] are assumed to be identical. Special NULL values [Page 22] are assumed to be identical.
- UNION: R contains all rows from T1 and T2.
- EXCEPT: R contains all rows from T1 which have no duplicate rows in T2.
- INTERSECT: R contains all rows from T1 which have a duplicate row in T2. A row from T2 can only be a duplicate row of exactly one row from T1. More than one row from T1 cannot have the same duplicate row in T2.
- DISTINCT is implicitly assumed for the QUERY expressions belonging to T1 and T2 if ALL is not specified. All duplicate rows are removed from R.

If parentheses are missing, then INTERSECT will be evaluated before UNION and EXCEPT. UNION and EXCEPT have the same precedence and will be evaluated from left to right in the case that parentheses are missing.

QUERY term (query_term)

A QUERY term (`query_term`) is part of the syntax in a QUERY expression (`query_expression` or `named_query_expression`).

Syntax

```
<query_term> ::= <query_primary> | <query_term> INTERSECT [ALL]
<query_primary>

<query_primary> ::= <query_spec> | (<query_expression>)
```

query_spec [Page 182], query_expression [Page 179]

Explanation

See QUERY expression [Page 179] or QUERY expression (named_query_expression) [Page 181]

QUERY expression (named query expression)

QUERY expressions (`named_query_expressions`) are required to generate an unordered result table in a SELECT statement (named_select_statement) [Page 177].

Syntax

```
<named_query_expression> ::= <named_query_term> | <named_query_expression>
UNION [ALL] <query_term> | <named_query_expression> EXCEPT [ALL]
<query_term>
```

named_query_term [Page 182], query_term [Page 180]

Explanation

If the QUERY expressions consists of more than one QUERY specification (query_spec) [Page 182] (specified in `named_query_term` or in `query_term`), the only the first QUERY specification in the QUERY expression may be a QUERY specification (`named_query_spec`).

If the QUERY expressions consists of only one QUERY specification (named_query_spec) [Page 184] (specified in `named_query_term`), the result of the expression is the unchanged result of the QUERY specification.

If a QUERY expression consists of more than one QUERY specification, the number of selected columns in all QUERY specifications of the QUERY expression must be the same. The respective i^{th} selected columns of the QUERY specifications must be comparable.

Column type (select column)	
Numeric columns	Are comparable. If all i^{th} selected columns are numeric columns, the i^{th} column of the result table is a numeric column.
Alphanumerical column, code attribute [Page 24] BYTE	Are comparable.
Alphanumeric columns, code attribute ASCII, EBCDIC, UNCODE	Are comparable. Are also comparable with date, time, and timestamp values.
All i^{th} columns are date values [Page 23]	The i^{th} column of the result table is a date value.
All i^{th} columns are time values [Page 23]	The i^{th} column of the result table is a time value.
All i^{th} columns are timestamp values [Page 24]	The i^{th} column of the result table is a timestamp value.
Columns of the type BOOLEAN [Page 24]	Are comparable.
All i^{th} columns are of the type BOOLEAN	The i^{th} column of the result table is of the type BOOLEAN.
Columns of any other data type (not mentioned above)	The i^{th} 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.

Column names in the result table

The names of the result table columns are formed from the names of the selected columns of the first QUERY specification.

Let T1 be the left operand of UNION, EXCEPT, or INTERSECT (defined in `named_query_term`). 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 have identical values in each column. NULL values [Page 22] are assumed to be identical. Special NULL values [Page 22] are assumed to be identical.
- UNION: R contains all rows from T1 and T2.
- EXCEPT: R contains all rows from T1 which have no duplicate rows in T2.
- INTERSECT: R contains all rows from T1 which have a duplicate row in T2. A row from T2 can only be a duplicate row of exactly one row from T1. More than one row from T1 cannot have the same duplicate row in T2.
- DISTINCT is implicitly assumed for the QUERY expressions belonging to T1 and T2 if ALL is not specified. All duplicate rows are removed from R.

If parentheses are missing, then INTERSECT will be evaluated before UNION and EXCEPT. UNION and EXCEPT have the same precedence and will be evaluated from left to right in the case that parentheses are missing.

QUERY term (named query term)

A QUERY term (`named_query_term`) is part of the syntax in a QUERY expression (`named_query_expression`).

Syntax

```
<named_query_term> ::= <named_query_primary> | <named_query_term> INTERSECT
[ALL] <query_primary>

<named_query_primary> ::= <named_query_spec> | (<named_query_expression>)
<query_primary> ::= <query_spec> | (<query_expression>)
```

named_query_spec [Page 184], named_query_expression [Page 181], query_spec [Page 182], query_expression [Page 179]

Explanation

See QUERY expression (named_query_expression) [Page 181]

QUERY specification (query_spec)

QUERY specifications (`query_spec`) are required to generate an unordered result table in a SELECT statement [Page 178]. A QUERY specification is part of the syntax in a QUERY term (query_term) [Page 180] or QUERY term (named_query_term) [Page 182].

Syntax

```
<query_spec> ::= SELECT [<distinct_spec>] <select_column>,...
<table_expression>
```

distinct_spec [Page 183], select_column [Page 183], table_expression [Page 185]

Explanation

A QUERY specification 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 [Page 185].

DISTINCT function (distinct spec)

The DISTINCT specification (`distinct_spec`) is specified in a QUERY specification (query_spec) [Page 182], QUERY specification (named_query_spec) [Page 184], or SINGLE SELECT statement [Page 196] to remove duplicate rows.

Syntax

```
<distinct_spec> ::= DISTINCT | ALL
```

Explanation

A row is a duplicate of another row if both have identical values in each column. NULL values [Page 22] are assumed to be identical. Special NULLvalues [Page 22] are also assumed to be identical.

DISTINCT: all duplicate rows are removed from the result table.

ALL: no duplicate rows are removed from the result table.

If no DISTINCT specification is specified, no duplicate rows are removed from the result table.

Selected column (select_column)

Selected columns (`select_column`) must be specified in a QUERY specification (query_spec) [Page 182] or a QUERY specification (named_query_spec) [Page 184] to specify a result table.

The sequence of selected columns defines the columns in the result table. The columns in the result table are produced from the columns of the temporary result table and by the ROWNO columns or STAMP columns, if these exist. The columns of the temporary result table are determined by the FROM clause [Page 185] of the table expression [Page 185]. The order of the column names in the temporary result table is determined by the order of the table names in the FROM clause.

Syntax

```
<select_column> ::= <table_columns> | <derived_column> | <rowno_column> |
<stamp_column>

<table_columns> ::= * | <table_name>.* | <reference_name>.*
<derived_column> ::= <expression> [ [AS] <result_column_name>] |
<result_column_name> = <expression>
<rowno_column> ::= ROWNO [<result_column_name>] | <result_column_name> =
ROWNO
<stamp_column> ::= STAMP [<result_column_name>] | <result_column_name> =
STAMP

<result_column_name> ::= <identifier>
```

table_name [Page 50], reference_name [Page 48], expression [Page 90], identifier [Page 40]

Explanation

Every column name that is specified as a selected column must uniquely denote a column in a QUERY specification (query_spec) [Page 182] of the underlying tables. If necessary, the column name must be qualified with the table name.

The specification of a column with the data type LONG [Page 116] in a selected column is only valid in the uppermost sequence of select columns in a QUERY statement [Page 174], SINGLE SELECT statement [Page 196], SELECT DIRECT statement [Page 196], or a SELECT ORDERED statement [Page 197] if the DISTINCT specification [Page 183] was not used there.

The specification of a column with the data type LONG in a selected column is only valid in the uppermost sequence of select columns in a CREATE VIEW statement [Page 137] which is based on exactly one base table.

If a selected column contains a set function (set_function_spec) [Page 87], the sequence of selected columns to which the selected column belongs must not contain any table columns, and every column

name occurring in an expression [Page 90] must denote a grouping column, or the expression must consist of grouping columns.

- Specifying table columns in a selected column is a quick way of specifying the result table columns.
  - Specifying a selected column of the type * is a quick way of specifying all temporary result table columns.
    Columns for which the user has not the SELECT privilege and the implicitly generated column SYSKEY are not passed.
  - Specifying `<table_name>.*` or `<reference_name>.*` is quick way of specifying all the columns in 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 column names in the underlying table corresponds to the order determined when the underlying table is defined. Columns for which the user has not the SELECT privilege and the implicitly generated column SYSKEY are not passed.
- Specifying **`derived column`** in a selected column defines a column in the result table. If a column of the result table has the form `<expression> [AS] <result_column_name>` or the form `<result_column_name> = <expression>`, this result column receives the name `result_column_name`.
  If no `<result_column_name>` is specified and the expression [Page 90] is a column specification that denotes a column in the temporary result table, the column in the result table receives the column name of the temporary result table.
  If no `<result_column_name>` is specified and the expression is not a column specification, the column receives the name `EXPRESSION_`, where "_" denotes a number with a maximum of three digits, starting with `EXPRESSION1`, `EXPRESSION2`, etc.
- A ROWNO column may only be used in a selected column that belongs to a QUERY statement. If a ROWNO column is specified, a column with the data type FIXED [Page 117](10) is generated with the name ROWNO. It contains the values 1, 2, 3,... which the numbers of the result table rows.
  If the ROWNO column was specified in the form `ROWNO <result_column_name>` or the form `<result_column_name> = ROWNO`, this result column receives the name `result_column_name`.
  A ROWNO column>must not be ordered by using ORDER BY.
- A STAMP column may only be specified in a selected column that belongs to a QUERY-expression [Page 179] of an INSERT statement [Page 164].
  The database system is able to generate unique values. This is a serial number that starts with X'000000000001'. The values are assigned in ascending order. It cannot be ensured that a sequence of values is uninterrupted.
  If a STAMP column is specified, the next value of the data type CHAR [Page 115](8) BYTE generated by the database system is produced for each row in the temporary result table.

Each column of a result table has exactly the same data type, the same length, the same precision, and the same scale as the `derived_column` or the column underlying the table columns.

This does not apply to the data types DATE and TIMESTAMP. To enable the representation of any date and time format, the length of the result table column is set to the maximum length required for the representation of a date value [Page 23] (length 10) or a timestamp value [Page 24] (length 26).

QUERY specification (named_query_spec)

QUERY specifications (`named_query_spec`) are required to generate an unordered result table in a SELECT statement (named_select_statement) [Page 177]. A QUERY specification is part of the syntax in a QUERY term (named_query_term) [Page 182].

Syntax

```
<named_query_spec> ::= SELECT [<distinct_spec>] <result_table_name>
(<select_column>,...) <table_expression>
```

distinct_spec [Page 183], result_table_name [Page 45], select_column [Page 183], table_expression [Page 185]

Explanation

A QUERY specification specifies a result table with the name `result_table_name`. The result table is generated from a temporary result table. The temporary result table is the result of the table expression [Page 185].

Table expression

A table expression specifies a single or a simple or grouped result table (see result table name [Page 45]).

Syntax

```
<table_expression> ::= <from_clause> [<where_clause>] [<group_clause>]
[<having_clause>]
```

from_clause [Page 185], where_clause [Page 188], group_clause [Page 188], having_clause [Page 189]

Explanation

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 condition and the table is the result of the last specified clause. The temporary result table contains all of the columns in all the tables listed in the FROM clause.

The order of the GROUP and HAVING clauses is random.

FROM clause

A FROM clause specifies a table in a table expression [Page 185] that is formed from one or more tables.

Syntax

```
<from_clause> ::= FROM <from_table_spec>,...
```

from_table_spec [Page 186]

Explanation

The FROM clause specifies a table. This table can be derived from several base, view, and result tables (see Table [Page 29]). The number of underlying tables in a FROM clause is equal to the total number of underlying tables in each FROM TABLE specification. The number of underlying tables in a FROM clause must not exceed 64.

The user must have the SELECT privilege for each specified table or for at least one column in the specified table.

The result of a FROM clause is a table that is generated from the specified tables as follows:

- If the FROM clause comprises a single FROM TABLE specification, the result is the specified table.
- If the FROM clause contains more than one FROM TABLE specification, 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. From a mathematical perspective, this is the Cartesian product of all the tables.

This rule describes the effect of the FROM clause, not its actual implementation.

FROM TABLE specification (from_table_spec)

Each FROM TABLE specification (`from_table_spec`) in a FROM clause [Page 185] specifies no, one, or any number of table identifiers.

Syntax

```
<from_table_spec> ::= <table_name> [<reference_name>]
| <result_table_name> [<reference_name>]
| (<query_expression>) [<reference_name>]
| <joined_table>
```

table_name [Page 50], reference_name [Page 48], result_table_name [Page 45], query_expression [Page 179], joined_table [Page 187]

Explanation

Reference name

If a FROM TABLE specification does not contain a reference name, the table name or result table name is the table identifier.

If a FROM TABLE specification contains a reference name, the reference name is the table identifier.

Each reference name must be different from each identifier [Page 40] that specifies a table name. If a result table name is a table identifier, there must not be any table identifiers in the form `<table_name>` equal to `[<owner.]<result_table_name>`, where owner [Page 44] is the current user. Each table identifier must differ from any other table identifier.

The validity range of the table identifiers is the entire QUERY specification [Page 182] within which the FROM TABLE specification is used. If column names are to be qualified within the QUERY specification, table identifiers must be used for this purpose.

Reference names are essential for formulating JOIN conditions within a table. For example, `FROM HOTEL, HOTEL X` defines a reference name X for the second occurrence of the HOTEL table. Reference names are also necessary sometimes to formulate correlated subqueries [Page 189]. Similarly, a reference name is required if a column in the result of a QUERY expression can only be identified uniquely by specifying the reference name.

Number of underlying tables

If a FROM TABLE specification denotes a base table, result table, or the result of a QUERY expression, the number of tables underlying this FROM TABLE specification is equal to 1.

If a FROM TABLE specification denotes a complex view table, the number of tables underlying this FROM TABLE specification is equal to 1.

If a FROM TABLE specification denotes a view table that is not a complex view table, the number of underlying tables is equal to the number of tables underlying the FROM condition [Page 185] of the view table.

If a FROM TABLE specification denotes a JOINED TABLE, the number of tables underlying this FROM TABLE specification is equal to the total number of underlying tables of the FROM TABLE specifications contained in it.

QUERY expression (query_expression)

A FROM TABLE specification that contains a QUERY expression specifies a table identifier only if a reference name is specified.

If a FROM TABLE specification contains a QUERY expression, a result table is built that matches this QUERY expression. This result table obtains a system-internal name that collides neither with an unnamed nor a named result table. While the FROM condition is being processed, the result of the

QUERY expression is used in the same way as a named result table and is deleted implicitly after processing.

A table expression [Page 185] containing at least one OUTER JOIN indicator (see JOIN predicate [Page 99]) or OUTER JOIN TYPE (LEFT | RIGHT | FULL) (see joined_table [Page 187]) is subject to strict restrictions if it is to be based on more than two tables. For this reason, a QUERY expression is frequently required to formulate a QUERY specification [Page 182] that is to be based on at least three tables and in which at least one OUTER JOIN indicator is used in a JOIN predicate.

JOINED TABLE

A FROM TABLE specification containing a JOINED TABLE (joined_table [Page 187]) specifies the number of table identifiers that are specified by the FROM TABLE specifications it contains.

Joined table

JOINED TABLE (`joined table`) can be specified as part of a FROM TABLE specification (from table spec) [Page 186].

Syntax

```
<joined_table> ::=
<from_table_spec> CROSS JOIN <from_table_spec>
| <from_table_spec> [INNER] JOIN <from_table_spec> <join_spec>
| <from_table_spec> [<LEFT | RIGHT | FULL> [OUTER]] JOIN <from_table_spec>
<join_spec>

<join_spec> ::= ON <search_condition> | USING (<column_name>,...)
```

from_table_spec [Page 186], search_condition [Page 107], column_name [Page 49]

Explanation

If a FROM TABLE specification comprises a JOINED TABLE, the result is generated as follows:

Let FT1 be the set of all rows in the table specified by the first FROM TABLE specification. Let FT2 be the set of all rows in the table specified by the second FROM TABLE specification.

- If JOINED TABLE is specified as **CROSS JOIN**, a table is created that comprises all possible combinations of FT1 and FT2. From a mathematical perspective, the Cartesian product of the two tables is calculated.
- If JOINED TABLE is specified with the keyword JOIN without the optional keywords INNER, LEFT, RIGHT, FULL, or OUTER, the JOIN type is assumed to be INNER.

Let T be the set of result rows consisting of all possible combinations of FT1 and FT2. Each result row satisfies the JOIN specification for this set.

- If JOINED TABLE is specified with the JOIN type **INNER**, the result is the set T.
- If JOINED TABLE is specified with the JOIN type **LEFT**, the result is the set T plus the rows from FT1 that are not in T. The result columns that are not formed from FT1 are assigned the NULL value.
- If JOINED TABLE is specified with the JOIN type **RIGHT**, the result is the set T plus the rows from FT2 that are not in T. The result columns that are not formed from FT2 are assigned the NULL value.
- If JOINED TABLE is specified with the JOIN type **FULL**, the result is the set T plus the rows from FT1 that are added by the JOIN types LEFT and RIGHT.

The rules specified for the WHERE condition [Page 188] apply to the JOIN specification (`join_spec`) `ON <search_condition>`.

If the JOIN specification (`join_spec`) `USING (<column_name>,...)` is specified, the column names must denote columns that are contained in both FT1 and FT2 and for which the user has the SELECT privilege. Specifying the JOIN specification `USING (<column_name>,...)` means the

same as comparison predicates [Page 95] between the specified columns in FT1 and FT2 linked with AND. = is used as a comparison operator (comp_op [Page 97]).

WHERE clause

The WHERE clause specifies the conditions for building a result table (see result table name [Page 45]).

Syntax

```
<where_clause> ::= WHERE <search_condition>
```

search_condition [Page 107]

Explanation

The search condition is applied to each row in the temporary result table formed by the FROM clause [Page 185]. The result of the WHERE clause is a table that only contains those rows from the result table for which the search condition is true.

The search condition may only contain column specifications for which the user has the SELECT privilege.

Expressions [Page 90] in the search condition must not contain a set function (set_function_spec) [Page 87].

Each column specification [Page 51] directly contained in the search condition must uniquely identify a column from the tables specified in the FROM clause of the table expression [Page 185]. If necessary, the column name must be qualified with the table identifier. If reference names are defined in the FROM clause for table names, they must be used as table identifiers in the search condition.

In the case of correlated subqueries [Page 189], a column specification can identify a column in a table that was specified in a FROM clause of a different table expression in the QUERY specification [Page 182].

Each subquery [Page 189] in the search condition is usually evaluated only once. In the case of a correlated subquery, the subquery is executed for each row in the result table generated by the FROM clause.

GROUP clause

The GROUP clause specifies grouping criteria for a result table (see result table name [Page 45]).

Syntax

```
<group_clause> ::= GROUP BY <expression>,...
```

expression [Page 90]

Explanation

Each column name specified in the GROUP clause must identify a `result_column_name` in the selected columns [Page 183] of the QUERY specification [Page 182] or uniquely identify a column in the tables on which the QUERY specification is based. If necessary, the column name must be qualified with the table identifier.

The GROUP clause allows the functions SUM [Page 90], AVG [Page 89], MAX/MIN [Page 90], COUNT [Page 89], STDDEV [Page 90], and VARIANCE [Page 90] 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 [Page 22] in a grouping column are combined to form a group. The same is true for the special NULL value [Page 22].

GROUP BY generates one row for each group in the result table. The selected columns in the QUERY specification, therefore, may only contain those grouping columns and operations on grouping columns, as well as those expressions [Page 90] that use the functions SUM, AVG, MAX/MIN, COUNT, STDDEV, and VARIANCE.

If no rows satisfy the conditions indicated in the WHERE clause [Page 188] and a GROUP clause was specified, the result table is empty.

HAVING clause

The HAVING clause specifies the properties of a group.

Syntax

```
<having_clause> ::= HAVING <search_condition>
```

search_condition [Page 107]

Explanation

Each expression [Page 90] in the search condition that does not occur in the argument of a set function (set_function_spec) [Page 87] must identify a grouping column.

If the HAVING clause is used without a GROUP clause [Page 188], the result table built so far is regarded as a group.

The search condition is applied to each group in the result table. The result of the HAVING clause is a table that only contains those groups for which the search condition is true.

Subquery

A subquery specifies a result table (see result table name [Page 45]) that can be used in certain predicates and for updating column values.

Syntax

```
<subquery> ::= (<query_expression>)
```

query_expression [Page 179]

Explanation

Subqueries can be used in a SET UPDATE condition [Page 171] of an UPDATE statement [Page 169]. In this case, the subquery must produce a result table that contains a maximum of one row.

Subqueries can be used in an INSERT statement (INSERT statement) [Page 164]).

Subqueries can be used in the following predicates:

Comparison predicate [Page 95]
EXISTS predicate [Page 97]
IN predicate [Page 98]
Quantified predicate [Page 104]

Correlated subquery

Certain predicates can contain subqueries. These subqueries, in turn, can contain other subqueries, etc. A subquery [Page 189] with further subqueries is the higher-level subquery of the subqueries it contains.

- The search condition [Page 107] of a subquery can contain column names that belong to tables that are contained in higher levels in the FROM clause [Page 185]. This type of subquery is called a **correlated subquery**.
- Tables that are used in subqueries in this way are called **correlated tables**. No more than 16 correlated tables are allowed within an SQL statement.
- Columns that are used in subqueries in this way are called **correlated columns**. A total of 64 correlated columns can be used in an SQL statement.

If the qualifying table name or reference name does not clearly identify a table at 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 searched. 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 in 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 to uniquely qualify the higher-level temporary result row.

Model tables hotel [Page 84] and room [Page 85].
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

The ORDER clause specifies a sort sequence for a result table (see result table name [Page 45]).

Syntax

```
<order_clause> ::= ORDER BY <sort_spec>,...

<sort_spec> ::= <unsigned_integer> [ASC | DESC] | <expression> [ASC | DESC]
```

unsigned_integer [Page 38], expression [Page 90]

Explanation

The sort columns specified in the ORDER clause determine the sequence of the sort criteria.

A number n specified in the sorting specification (`sort_spec`) identifies the n^{th} column in the result table. n must be less than or equal to the number of columns in the result table.

The maximum number of sort specifications in the sort criterion is 128.

ASC/DESC

ASC: the values are sorted in ascending order.

DESC: the values are sorted in descending order.

The default setting is ASC.

Further information

If a QUERY expression [Page 179] consists of more than one QUERY specification [Page 182], sort specifications must be specified in the form `<unsigned_integer> [ASC | DESC]`.

If a QUERY specification was specified with DISTINCT, the total of the internal lengths of all sorting columns must not exceed 1016 characters; otherwise it can comprise 1020 characters.

Column names in the sort specifications must be columns in the tables of the FROM clause [Page 185] or a result_column_name in the selected columns [Page 183] of the QUERY specification.

If DISTINCT or a set function [Page 87] in a selected column was used, the sort specification must identify a column in the result table.

Values are compared in accordance with the rules for the comparison predicate [Page 95]. For sorting purposes, NULL value [Page 22]s are greater than non-NULL values, and special NULL value [Page 22]s are greater than non-NULL values but less than NULL values.

UPDATE clause

The UPDATE clause specifies that a result table (see result table name [Page 45]) is to be updateable.

Syntax

```
<update_clause> ::= FOR UPDATE [OF <column_name>,...]
```

column_name [Page 49]

Explanation

The specified column names must identify columns in the tables underlying the QUERY specification [Page 182]. They do not have to occur in a selected column [Page 183].

The QUERY statement that contains the UPDATE clause must generate an updateable result table.

The UPDATE clause is a prerequisite for using the result table with CURRENT OF <result_table_name> in the UPDATE statement [Page 169], DELETE statement [Page 171], LOCK statement [Page 209], SELECT DIRECT statement [Page 197], and SELECT ORDERED statement [Page 199]. The UPDATE clause is meaningless for other forms of the above mentioned SQL statements, as well as in interactive mode.

All columns of the underlying base tables are updateable if the user has the corresponding privileges, irrespective of whether they were specified as a column name [Page 49].

For performance reasons, it is recommended to specify column names only if the cursor is to be used in an UPDATE statement.

Assume that the a column x fulfills the following conditions:

- x is contained in the primary key or an index
- x is contained in the search condition [Page 107] of the QUERY statement
- x is contained in a SET UPDATE clause [Page 171] of the UPDATE statement in the type x = <expression>, where the expression [Page 90] contains the column x.

If all of the conditions are fulfilled, it is essential that you specify the column x as a column name in the UPDATE clause.

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

LOCK option

The LOCK option requests a lock for each selected row.

Syntax

```
<lock_option> ::=
  WITH LOCK [(NOWAIT)] [EXCLUSIVE | OPTIMISTIC] [ISOLATION LEVEL
<unsigned_integer>]
```

unsigned_integer [Page 38] may only have the values 0, 1, 2, 3, 10, 15, 20, or 30

Explanation

EXCLUSIVE

An exclusive lock (see transactions [Page 202]) is defined. As long as the locked row has not been updated or deleted, the exclusive lock can be cancelled using the UNLOCK statement [Page 211].

OPTIMISTIC

OPTIMISTIC defines an optimistic lock on rows. This lock only makes sense when it is used together with the isolation levels 0, 1, 10, and 15. An update operation of the current user on a row which has been locked by this user using an optimistic lock is only performed if this row has not been updated in the meantime by a concurrent transaction. If this row has been changed in the meantime by a concurrent transaction, the update operation of the current user is rejected. The optimistic lock is released in both cases. If the update operation was successful, an exclusive lock is set for this row. If the update operation was not successful, it should be repeated after reading the row again with or without optimistic lock. In this way, it can be ensured that the update is done to the current state and that no modifications made in the meantime are lost.

The request of an optimistic lock only collides with an exclusive lock. Concurrent transactions do not collide with an optimistic lock.

SHARE lock

If neither EXCLUSIVE nor OPTIMISTIC is specified, a SHARE lock on rows is thus defined. If a SHARE lock was set on a row, no concurrent transaction can modify this row.

ISOLATION LEVEL

The locks are set independently of the ISOLATION specification (`isolation_spec`) of the CONNECT statement [Page 204]. The isolation level of the LOCK option can identify a higher or lower value than that in the CONNECT statement. The meaning of the various isolation levels is explained in the description of the CONNECT statement.

If an isolation level is specified by the LOCK option, it is only valid for the duration of the SQL statement which contains the LOCK option specification. Afterwards, the isolation level that was specified in the CONNECT statement is applicable again. In the case of a SELECT statement (named_select_statement) [Page 177], SELECT statement (select_statement) [Page 178], or an OPEN CURSOR statement [Page 192], for which the result table is not actually physically generated, the specified isolation level is valid for this SQL statement and all FETCH statements [Page 193] that refer to the result table. The isolation level that was specified in the CONNECT statement is applicable for other SQL statements that were executed in the meantime.

NOWAIT

If `(NOWAIT)` is specified, the database system does not wait until another user has released a data object. Instead, it returns a message if a collision occurs. If there is no collision, the requested lock is set.

If `(NOWAIT)` is not specified and a collision occurs, the system waits for the locked data object to be released (but only as long as is specified by the installation parameter REQUEST_TIMEOUT).

OPEN CURSOR statement

An OPEN CURSOR statement generates the result table defined under the specified name with a DECLARE CURSOR statement [Page 175].

Syntax

```
<open_cursor_statement> ::= OPEN <result_table_name>
```

result_table_name [Page 45]

Explanation

Existing result tables are implicitly deleted when a result table is generated with the same name.

All result tables generated within the current transaction are implicitly deleted at the end of the transaction using the ROLLBACK statement [Page 208].

All result tables are implicitly deleted at the end of the session using the RELEASE statement [Page 212]. A CLOSE statement [Page 195] can be used to delete them explicitly beforehand.

If the name of a result table is identical with that of a base table, view table (see Table [Page 29]), or a synonym [Page 30], these tables cannot be accessed as long as the result table exists.

At any given time when a result table is processed, 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.

Depending on the search strategy, either all the rows in the result table are searched when the OPEN CURSOR statement is executed and the result table is physically generated, or each next result table row is searched when a FETCH statement [Page 193] is executed, without being physically stored. This must be considered for the time behavior of OPEN CURSOR statements and FETCH statements.

If the result table is empty, the return code `100 - row not found -` is set.

The number of rows in the result table is returned in the SQLCA in the third entry of SQLERRD. If this counter has the value -1, there is at least one result row.

FETCH statement

The FETCH statement assigns the values of the current row in a result table (see result table name [Page 45]) to parameters.

Syntax

```
<fetch_statement> ::=
FETCH [FIRST | LAST | NEXT | PREV | <position> | SAME]
[<result_table_name>] INTO <parameter_spec>,...

<position> ::= POS (<unsigned_integer>) | POS (<parameter_spec>)
| ABSOLUTE <integer> | ABSOLUTE <parameter_spec>
| RELATIVE <integer> | RELATIVE <parameter_spec>
```

result_table_name [Page 45], parameter_spec [Page 51], unsigned_integer [Page 38], integer [Page 39]

Explanation

If no result table name is specified, the FETCH statement refers to the last unnamed result table that was generated (see named/unnamed result table [Page 175]).

Depending on the search method, either all the rows in the result table are searched when the OPEN CURSOR statement [Page 192], the SELECT statement (select_statement) [Page 178], or the SELECT statement (named_select_statement) [Page 177] is executed and the result table is generated, or each subsequent row in the result table is searched when a FETCH statement is executed, but they are not physically stored. This must be taken into account for the time behavior of FETCH statements. Depending on the isolation level selected, this can also cause locking problems with a FETCH, e.g. return code `500 - LOCK REQUEST TIMEOUT`.

Row not found

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:

- The result table is empty.
- C is positioned on or after the last result table row, and FETCH or FETCH NEXT is specified.

- C is positioned on or before the first row of the result table and FETCH PREV is specified.
- FETCH is specified with a position which does not lie within the result table.

FIRST | LAST | NEXT | PREV

- FETCH FIRST or FETCH LAST: the result table is not empty. C is positioned in the first or last row of the result table and the values of this row are assigned to the parameters.
- FETCH or FETCH NEXT: C is positioned before a row in the result table. C is positioned in this row and the values of this row are assigned to the parameters.
- FETCH or FETCH NEXT: C is positioned in a row that is not the last row in the result table. C is positioned in the next row and the values in this row are assigned to the parameters.
- FETCH PREV: C is positioned behind a row in the result table. C is positioned in this row and the values of this row are assigned to the parameters.
- FETCH PREV: C is positioned in a row that is not the first row in the result table. C is positioned in the previous row and the values in this row are assigned to the parameters.

Position: POS

Regardless of an ORDER clause [Page 190], there is an implicit order of the rows in a result table. This can be displayed by specifying a ROWNO column as a selected column [Page 183]. The specified position refers to this internal numbering.

If a position is defined with POS, the parameter specification must denote a positive integer.

If a position that is less than or equal to the number of rows in the result table was defined with POS, C is set to the corresponding row and the values of this row are assigned to the parameters. If a position that is greater than the number of rows in the result table was specified, the message `100 - ROW NOT FOUND` is output.

Position: ABSOLUTE

Let x be the value of the integer or parameter specification specified with the position. Let `abs_x` be the absolute value of x.

- FETCH ABSOLUTE and x is : FETCH ABSOLUTE is the same as a FETCH POS.
- FETCH ABSOLUTE and x=0: the return code `100 - row not found` is set.
- FETCH ABSOLUTE and x is negative: C is set after the last row of the result table where FETCH PREV is executed `abs_x` times. The last row found is the result of the SQL statement. This description refers to the logic and not the flow of the statement. If `abs_x` is larger than the number of rows in the result table, the `message 100 - ROW NOT FOUND` is output.

Position: RELATIVE

Let x be the value of the integer or parameter specification specified with the position. Let `abs_x` be the absolute value of x.

- FETCH RELATIVE and x is positive: FETCH NEXT is executed x times from the current position in the result table C.
- FETCH RELATIVE and x=0: corresponds to a FETCH SAME.
- FETCH RELATIVE and x is negative: FETCH PREV is executed `abs_x` times starting from C. This description refers to the logic and not the flow of the statement. The return code `100 - row not found` is output if one of the conditions in the section "row not found" is fulfilled.

FETCH SAME

The last row found in the result table is output again.

Parameter specification

The parameter specification [Page 51] specified in **position** must denote an integer.

The remaining parameters in the parameter specification are output parameters. The parameter identified by the n^{th} parameter specification corresponds to the n^{th} 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. You must specify an indicator name [Page 45] in order to assign NULL values [Page 22] or special NULL values [Page 22].

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 unchanged.

Let p be a parameter and v the corresponding value in the current row of the result table.

- v is a number: p must be a numeric parameter and v must be within the permissible range of p.
- v is a character string: p must be an alphanumeric parameter.

Further information

If no FOR REUSE was specified in the QUERY statement (see SELECT statement [Page 178]), subsequent INSERT, UPDATE, or DELETE statements that refer to the underlying base table and are executed by the current user or by other users can cause several executions of a FETCH statement to denote different rows in the result table, even though the same position was specified.

You can prevent other users from making changes by executing a LOCK statement [Page 209] for the entire table or by using the isolation level 2, 3, 15, 20, or 30 with the CONNECT statement [Page 204] or the LOCK option [Page 191] of the QUERY statement.
FOR REUSE must be specified if this is not possible or if the user makes changes to this table. Changes made in the meantime are not visible in this case.

If a result table that was physically created contains LONG columns [Page 23] and if the isolation levels 0, 1, and 15 are used, consistency between the content of the LONG columns and that of the other columns is not ensured. If the result table was not physically generated, consistency is not ensured at isolation level 0 only. For this reason, it is advisable to ensure consistency by using a LOCK statement or the isolation levels 2, 3, 20, or 30.

CLOSE statement

The CLOSE statement deletes a result table (see result table name [Page 45]).

Syntax

```
<close_statement> ::= CLOSE [<result_table_name>]
```

result_table_name [Page 45]

Explanation

- If the name of a result table is specified, this result table is deleted. This name can be used to denote another result table.
- If no result table name is specified, an existing unnamed result table is deleted.

An unnamed result table is implicitly deleted by the next SELECT statement [Page 178].

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

All result tables generated within the current transaction are implicitly deleted at the end of the transaction using the ROLLBACK statement [Page 208].

All result tables are implicitly deleted at the end of the session using the RELEASE statement [Page 212].

SINGLE SELECT statement

A SINGLE SELECT statement specifies a result table with one row (see result table name [Page 45]) and assigns the values in this row to parameters.

Syntax

```
<single_select_statement> ::=
SELECT [<distinct_spec>] <select_column>,...
INTO <parameter_spec>,... FROM <from_table_spec>,...
[<where_clause>] [<group_clause>] [<having_clause>] [<lock_option>]
```

distinct_spec [Page 183], select_column [Page 183], parameter_spec [Page 51], from_table_spec [Page 186], where_clause [Page 188], group_clause [Page 188], having_clause [Page 189], lock_option [Page 191]

Explanation

The number of rows in the result table must not be greater than one. If the result table is empty or contains more than one row, corresponding messages or error codes are issued and no values are assigned to the parameters specified in the parameter specifications. The return code `100 - row not found -` is set if the result table is empty.

If the result table contains just one row, the values of this row are assigned to the corresponding parameters. The FETCH statement [Page 193] rules apply for assigning the values to the parameters.

The order of the GROUP and HAVING clauses is random.

A LONG column [Page 23] can only be specified in a selected column in the uppermost sequence of selected columns in a SINGLE SELECT statement if the DISTINCT specification [Page 183] DISTINCT was not used there.

SELECT DIRECT statement (select direct statement: searched)

The SELECT DIRECT statement (`select direct statement: searched`) selects a row in a table. A specified key value is used for the selection.

Syntax

```
<select_direct_statement:_searched> ::= SELECT DIRECT <select_column>,...
INTO <parameter_spec>,... FROM <table_name> KEY <key_spec>,...
[<where_clause>] [<lock_option>]
```

select_column [Page 183], parameter_spec [Page 51], table_name [Page 50], key_spec [Page 55], where_clause [Page 188], lock_option [Page 191]

Explanation

The SELECT DIRECT statement is used to access a particular row in a table directly by specifying the key columns. For tables defined without key columns, there is the implicitly generated column SYSKEY CHAR(8) BYTE which contains a key generated by the database system. The table column SYSKEY, therefore, can be used in the SELECT DIRECT statement to access a specific table row.

The user must have the SELECT privilege for the selected columns or for the entire table.

If a row with the specified key values is found and if a search condition [Page 107] specified for this row is true, the corresponding column values are assigned to the parameters. The FETCH statement [Page 193] rules apply for assigning the values to the parameters.

If there is no row with the specified key values, or if a row with the specified key values does exist but a search condition defined for this row is not true, the return code `100 - ROW NOT FOUND -` is issued and no values are assigned to the parameters specified in the parameter specifications.

`INTO <parameter_spec>` is not necessary in interactive mode.

The LOCK option determines which lock is set for the read row.

A LONG column can only be specified in a selected column in the uppermost sequence of selected columns in this SELECT DIRECT statement.

See also:

SELECT DIRECT statement (select_direct_statement:_positioned) [Page 197]

SELECT DIRECT statement (select_direct_statement:_positioned)

The SELECT DIRECT statement (`select_direct_statement:_positioned`) selects a row in a table. A position in a result table is used for the selection.

Syntax

```
<select_direct_statement:_positioned> ::= SELECT DIRECT <select_column>,...
INTO <parameter_spec>,... FROM <table_name>
WHERE CURRENT OF <result_table_name> [<lock_option>]
```

select_column [Page 183], parameter_spec [Page 51], table_name [Page 50], result_table_name [Page 45], lock_option [Page 191]

Explanation

The table name in this SELECT DIRECT statement must be identical to that in the FROM clause [Page 185] of the QUERY statement that generated the result table.

The result table must have been specified with FOR UPDATE.

If the cursor is positioned on a row of the result table, then column values are selected from the corresponding row and are assigned to parameters. The corresponding row is the row of the table specified in the FROM condition of the QUERY statement, from which the result table row was formed. The FETCH statement [Page 193] rules apply for assigning the values to the parameters.

If the cursor is not positioned on a row in the result table, an error message is issued and no values are assigned to the parameters specified in the parameter specifications.

`INTO <parameter_spec>` is not necessary in interactive mode.

The LOCK option determines which lock is set for the read row.

A LONG column can only be specified in a selected column in the uppermost sequence of selected columns in this SELECT DIRECT statement.

See also:

SELECT DIRECT statement (select_direct_statement:_searched) [Page 196]

SELECT ORDERED statement (select_ordered_statement:_searched)

The SELECT ORDERED statement (`select_ordered_statement:_searched`) selects the first or last row, or, in relation to a certain position, the next or previous row in an ordered table. The order is defined by a key or by an index [Page 30]. The position is defined by specified key values and index values.

Syntax

```
<select_ordered_statement:_searched> ::=
<select_ordered_format1:_searched> | <select_ordered_format2:_searched>


<select_ordered_format1:_searched> ::= SELECT <FIRST | LAST>
<select_column>,...
INTO <parameter_spec>,... FROM <table_name>
[<pos_spec>] [<where_clause>] [<lock_option>]

<select_ordered_format2:_searched> ::= SELECT <NEXT | PREV>
<select_column>,...
INTO <parameter_spec>,... FROM <table_name>
```
[<index_pos_spec>] KEY <key_spec>,... `[<where_clause>] [<lock_option>]`

<pos_spec> ::= INDEX <column_name> | INDEXNAME <index_name>
| <index_pos_spec> [KEY <key_spec>,...] | KEY <key_spec>,...

select_column [Page 183], parameter_spec [Page 51], table_name [Page 50], where_clause [Page 188], lock_option [Page 191], index_pos_spec [Page 199], key_spec [Page 55], column_name [Page 49], index_name [Page 45]

Explanation

The SELECT ORDERED statement is used to access the first or last row of an order defined by the key or a secondary key, or to access the previous or next row starting at a specified position. For tables defined without key columns, there is the implicitly generated column SYSKEY CHAR(8) BYTE which contains a key generated by the database system. The table column SYSKEY, therefore, can be used in the SELECT ORDERED statement to access a specific table row. In a table, the order defined by the ascending values of SYSKEY corresponds to the order of insertions made to the table.

- If no index name is specified with `INDEX <column_name>` or `INDEXNAME <index_name>` and no index position is specified with (`index_pos_spec`), the order is defined by the key.
- If an index name is specified with `INDEX <column_name>` or `INDEXNAME <index_name>` or an index position is specified with, the order is defined by the secondary key and the key. The ascending key order then is the second sort criterion.

The position in the table can be specified explicitly by means of an index position and key (`key_spec`). The table does not have to contain a row with the position values.

FIRST | LAST

FIRST (LAST) produces a search for the first (last) row in the ordered table which satisfies the specified WHERE condition [Page 188] and which, in relation to the order, is greater (less) than or equal to the position.

NEXT | PREV

NEXT (PREV) produces a search in ascending (descending) order for the next row which satisfies the specified WHERE condition, starting at the specified position. If no WHERE condition is specified, the result is the row which is next according to order and position.

Single-column index

If an index name is specified with `INDEX <column_name>` or `INDEXNAME <index_name>` or if an index position is specified and the index associated with it only contains one column, the rows with NULL values [Page 22] in the index column are not taken into account for the SELECT ORDERED statement. In such a case, the result of the SELECT ORDERED statement can never be a row with a NULL value in the index column. This state is indicated by a warning.

Further information

The user must have the SELECT privilege for the selected columns or for the entire table.

The SELECT ORDERED statement cannot be used for view tables which have been defined by SELECT DISTINCT or which have more than one underlying base table.

The column name in the position specification (pos_spec) must denote an indexed column.

INTO <parameter_spec> is not necessary in interactive mode.

A LONG column [Page 23] can only be specified in a selected column in the uppermost sequence of selected columns in a SELECT ORDERED statement.

The LOCK option [Page 191] determines which lock is set for the read row.

Result

If a row was found that satisfies the specified conditions, the corresponding column values are assigned to the parameters. The FETCH statement [Page 193] rules apply for assigning the values to the parameters.

If the specified table does not contain a row that satisfies the specified conditions, the return code 100 - ROW NOT FOUND - is issued and no values are assigned to the parameters specified in the parameter specifications.

See also:

SELECT ORDERED statement (select_ordered_statement:_positioned) [Page 199]

Index position specification (index_pos_spec)

The index position specification (index_pos_spec) is a syntax element in the following SQL statements:

SELECT ORDERED statement (select_ordered_statement:_searched) [Page 197]

SELECT ORDERED statement (select_ordered_statement:_positioned) [Page 199]

Syntax

```
<index_pos_spec> ::= INDEX <column_name> = <value_spec>
| INDEXNAME <index_name> VALUES (<value_spec>,...)
```

column_name [Page 49], value_spec [Page 52], index_name [Page 45]

Explanation

The column name in the index position specification must denote an indexed column.

SELECT ORDERED statement (select_ordered_statement:_positioned)

The SELECT ORDERED statement (select_ordered_statement:_positioned) selects the first or last row, or, in relation to a certain position, the next or previous row in an ordered table. The order is defined by a key or by an index. The position is defined by a cursor position.

Syntax

```
<select_ordered_statement:_positioned> ::=
<select_ordered_format1:_positioned> | <select_ordered_format2:_positioned>

<select_ordered_format1:_positioned> ::=
  SELECT <FIRST | LAST> <select_column>,... INTO <parameter_spec>,...
FROM <table_name> [INDEX <column_name> | INDEXNAME <index_name>]
WHERE CURRENT OF <result_table_name> [<lock_option>]
| SELECT <FIRST | LAST> <select_column>,... INTO <parameter_spec>,...
FROM <table_name> [index_pos_spec]
WHERE CURRENT OF <result_table_name> [<lock_option>]

<select_ordered_format2:_positioned> ::=
SELECT <NEXT | PREV> <select_column>,...INTO <parameter_spec>,...
```

```
FROM <table_name> [<index_pos_spec>]
WHERE CURRENT OF <result_table_name> [<lock_option>]
```

select_column [Page 183], parameter_spec [Page 51], table_name [Page 50], column_name [Page 49], index_name [Page 45], result_table_name [Page 45], lock_option [Page 191], index_pos_spec [Page 199]

Explanation

The SELECT ORDERED statement is used to access the first or last row of an order defined by the key or a secondary key, or to access the previous or next row starting at a specified position.

The result table must have been specified with FOR UPDATE.

The user must have the SELECT privilege for the selected columns or for the entire table.

The table name in the SELECT ORDERED statement must be identical to that in the FROM clause [Page 185] of the QUERY statement that generated the result table.

- If no index name is specified with `INDEX <column_name>` or `INDEXNAME <index_name>` and no index position is specified with (`index_pos_spec`), the order is defined by the key.
- If an index name is specified with `INDEX <column_name>` or `INDEXNAME <index_name>` or an index position is specified with, the order is defined by the secondary key and the key. The ascending key order then is the second sort criterion.

The position within the table is defined by the optional index position specification and by a key value, whereby the key value is determined by the cursor position.

FIRST | LAST

FIRST (LAST) produces a search for the first (last) row which, in relation to the order, is greater (less) than or equal to the position.

NEXT | PREV

NEXT (PREV) produces a search in ascending (descending) order for the next row, starting at the specified position.

Single-column index

If an index name is specified with `INDEX <column_name>` or `INDEXNAME <index_name>` or if an index position is specified and the index associated with it only contains one column, the rows with NULL values [Page 22] in the index column are not taken into account for the SELECT ORDERED statement. In such a case, the result of the SELECT ORDERED statement can never be a row with a NULL value in the index column.

Further information

`INTO <parameter_spec>` is not necessary in interactive mode.

The column name must denote an indexed column.

The LOCK option [Page 191] determines which lock is set for the read row.

A LONG column [Page 23] can only be specified in a selected column in the uppermost sequence of selected columns in this SELECT ORDERED statement.

Result

If the cursor is positioned on a row of the result table and a row was found which satisfies the specified conditions, then the corresponding column values are assigned to the parameters. The FETCH statement [Page 193] rules apply for assigning the values to the parameters.

If the cursor is not positioned on a row in the result table, an error message is issued and no values are assigned to the parameters specified in the parameter specifications.

See also:

SELECT ORDERED statement (select_ordered_statement:_searched) [Page 197]

EXPLAIN statement

The EXPLAIN statement describes the search strategy used internally by the database system for a QUERY statement [Page 174] or SINGLE SELECT statement [Page 196] (statements for searching for certain rows in specific tables). This statement indicates in particular whether and in which form key columns or indexes are used for the search.

Syntax

```
<explain_statement> ::=
  EXPLAIN [(<result_table_name>)] <query_statement>
| EXPLAIN [(<result_table_name>)] <single_select_statement>
```

result_table_name [Page 45], query_statement [Page 174], single_select_statement [Page 196]

Explanation

The EXPLAIN statement can be used to check the effect of creating or deleting indexes (see index [Page 30]) on the choice of search strategy for the specified SQL statement. It is also possible to estimate the time needed by the database system to process the specified SQL statement. The specified QUERY or SINGLE SELECT statement is not executed while the EXPLAIN statement is being executed.

You will find information on the Optimizer functions in Optimizer: SAP DB 7.3 [Extern].

As a result of the EXPLAIN statement, a result table is generated (see Result table name [Page 45]). This result table may be named. If the optional name specification is missing, the result table is given the name SHOW. The result table has the following structure:

Structure of the EXPLAIN result table

OWNER	CHAR [Page 115](64)
TABLENAME	CHAR(64)
COLUMN_OR_INDEX	CHAR(64)
STRATEGY	CHAR(40)
PAGECOUNT	CHAR(10)
O	CHAR (1)
D	CHAR (1)
T	CHAR (1)
M	CHAR (1)

The sequence in which the SELECT is processed is described by the order of the rows in the result table.

STRATEGY

The STRATEGY column shows which search strategy/ies is/are used and whether a result table is generated. A result table is physically generated if the column STRATEGY contains `RESULT IS COPIED` in the last result row.

COLUMN_OR_INDEX

The COLUMN_OR_INDEX column shows which key column or indexed column or which index is used for the strategy.

PAGECOUNT

The PAGECOUNT column shows which sizes are assumed for the tables or, in the case of certain strategies, for the indexes. These sizes influence the choice of the search strategy.

The assumed sizes are updated using the UPDATE STATISTICS statement [Page 226] and can be requested by selecting the system table OPTIMIZERSTATISTICS. The current sizes of tables or indexes can be checked by selecting the TABLESTATISTICS and INDEXSTATISTICS system tables. If there are large discrepancies between the values contained in the OPTIMIZERSTATISTICS and TABLESTATISTICS, the UPDATE STATISTICS statement should be performed for this table.

If the system discovers during a search in a table that the values determined by the last UPDATE STATISTICS statement are extremely low, a row is entered in the SYSUPDSTATWANTED system table that contains the table name. In all other cases, rows are entered in this system table that describe columns in tables. The UPDATE STATISTICS statement should be executed for tables and columns in tables that are described in the SYSUPDSTATWANTED system table.

The last row contains the estimated SELECT cost value in the PAGECOUNT column. The specifications for COSTLIMIT and COSTWARNING in the CREATE USER, CREATE USERGROUP, ALTER USER, and ALTER USERGROUP statements refer to this estimated SELECT cost value.

O, D, T, M

The columns O, D, T, and M are used for support purposes and are not explained here.

Transactions

A transaction [Page 32] is a sequence of SQL statements that are handled by the database system as a basic unit, in the sense that any modifications made to the database by the SQL statements are either all reflected in the state of the database, or else none of the database modifications are retained.

The first transaction is opened when a session [Page 32] is opened with the CONNECT statement [Page 204]. The transaction is concluded with the COMMIT statement [Page 207] or the ROLLBACK statement [Page 208]. When a transaction is successfully concluded with a COMMIT statement, all of the changes to the database are retained. If a transaction is aborted using a ROLLBACK statement, on the other hand, or if it is aborted in another way, all of the changes to the database made by the transaction are rolled back.

Both the COMMIT and ROLLBACK statements open a new transaction implicitly.

Locks

Since the database system permits concurrent transactions on the same database objects, locks on rows, tables, and the catalog are necessary to isolate individual transactions. These locks are set either implicitly by the database system while an SQL statement is being processed or explicitly using the LOCK statement [Page 209]. The locks are assigned to the transaction that contains the SQL statement or LOCK statement. The database system distinguishes between **SHARE locks** and **exclusive locks**, which refer either to rows or tables, and **optimistic row locks**. In addition, there are special locks for the metadata of the catalog. These locks, however, are always set implicitly.

- **SHARE** locks: once a SHARE lock is assigned to a transaction for a particular data object, other transactions can access the object but cannot modify it.
- **Exclusive** locks: 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.

EXCLUSIVE locks for rows that have not yet been modified and SHARE locks on rows can be released by the UNLOCK statement [Page 211] before the end of the transaction.

The locks assigned to a transaction are usually released at the end of the transaction, making the respective database objects accessible again to other transactions.

The following table contains an overview of the possible parallel locks. EXCL stands for an exclusive lock and SHARE for a SHARE lock.

	A transaction has an					
	EXCL	SHARE	EXCL	SHARE	EXCL	SHARE
Can another transaction	lock on a table		lock on a line		lock on the system catalog	
lock the table with an EXCLUSIVE lock?	No	No	No	No	No	Yes
lock the table with a SHARE lock?	No	Yes	No	Yes	No	Yes
lock a line in the table with an EXCLUSIVE lock?	No	No	---	---	No	Yes
lock the locked line with an EXCLUSIVE lock?	---	---	No	No	---	---
lock a different line with an EXCLUSIVE lock?	---	---	Yes	Yes	---	---
lock a line in the table with a SHARE lock?	No	Yes	---	---	No	Yes
lock the locked line with a SHARE lock?	---	---	No	Yes	---	---
lock a different line with a SHARE lock?	---	---	Yes	Yes	---	---
change the table definition in the system cataolg?	No	No	No	No	No	No
read the table definition in the system catalog?	Yes	Yes	Yes	Yes	No	Yes

A lock collision exists in the cases which are marked with "No"; i.e., after having requested a lock within a transaction, the user must 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.

Subtransactions

The SQL statements SUBTRANS BEGIN, SUBTRANS END, and SUBTRANS ROLLBACK (SUBTRANS statement [Page 208]) subdivide a transaction into additional basic units. These can be nested as often as necessary and in any form. Unlike transactions, however, modifications made by subtransactions can be reversed by a the ROLLBACK statement or a SUBTRANS ROLLBACK of a higher-level subtransaction, even after the subtransaction has been concluded with SUBTRANS END.

SQL statements for transaction management

CONNECT statement [Page 204]	SET statement [Page 206]	
COMMIT statement [Page 207]	ROLLBACK statement [Page 208]	SUBTRANS statement [Page 208]
LOCK statement [Page 209]	UNLOCK statement [Page	RELEASE statement [Page

	211]	212]

CONNECT statement

A CONNECT statement opens a database session [Page 32] and a transaction [Page 32] for a database user.

Syntax

```
<connect_statement> ::=
  CONNECT <parameter_name> IDENTIFIED BY <parameter_name>
[<connect_option>...]
| CONNECT <parameter_name> IDENTIFIED BY <password> [<connect_option>...]
| CONNECT <user_name> IDENTIFIED BY <parameter_name> [<connect_option>...]
| CONNECT <user_name> IDENTIFIED BY <parameter_name> [<connect_option>...]

<connect_option> ::=
  SQLMODE <INTERNAL | ANSI | DB2 | ORACLE>
| ISOLATION LEVEL <unsigned_integer> | TIMEOUT <unsigned_integer>
| TERMCHAR SET <termchar_set_name>
```

parameter_name [Page 46], password [Page 46], user_name [Page 43], unsigned_integer [Page 38], termchar_set_name [Page 50]

Explanation

If the parameter name/user name and parameter name/password combination is valid, the user [Page 30] opens a database session and gains access to the database. As a result, he or she is the current user in this session.

A transaction is opened implicitly (see transactions [Page 202]). The COMMIT statement [Page 207] or ROLLBACK statement [Page 208] ends a transaction and opens a new one implicitly. At the end of each transaction, all locks assigned to the transaction are released, providing they are not maintained by a KEEP LOCK. The isolation specification in the CONNECT statement is applied to each new transaction.

Each CONNECT option may only be specified once.

SQL mode

The specification `SQLMODE <INTERNAL | ANSI | DB2 | ORACLE>` can be used to select the SQL mode [Page 34]. The default SQL mode is INTERNAL.

The CONNECT option `SQLMODE <INTERNAL | ANSI | DB2 | ORACLE>` is not allowed in programs. The appropriate precompiler option must be used to specify an SQLMODE other than INTERNAL.

Locks / ISOLATION LEVEL

The unsigned integer after ISOLATION LEVEL keywords may only have the values 0, 1, 2, 3, 10, 15, 20, and 30.

Locks (see transactions [Page 202]) can be requested either implicitly or explicitly. Explicit lock requests are made with the LOCK statement [Page 209]. The specified isolation level determines whether a lock is requested implicitly or explicitly. The length of time for which an implicit SHARE lock is maintained also depends on the isolation level. Exclusive locks set implicitly cannot be released within a transaction. Explicit lock requests are possible with every isolation level.

- **ISOLATION LEVEL 0**: rows can be read without requesting SHARE locks; i.e. no SHARE locks are requested implicitly. 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.

- **ISOLATION LEVEL 1 or 10**: 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 R is released and a SHARE lock is assigned to the transaction for the row R2.
  For data retrieval using a QUERY statement [Page 174], the database system ensures that, at the time each row is read, no exclusive lock has been assigned to other transactions for the given row. It is impossible to predict, however, 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.
- **ISOLATION LEVEL 15**: for all SQL statements that read exactly one table row using the key, ISOLATION LEVEL 15 is equivalent to ISOLATION LEVEL 1 or 10.
  With all other SQL statements, the description of ISOLATION LEVEL 1 also applies to ISOLATION LEVEL 15, with the exception that a SHARE lock is set for all the tables addressed by the SQL statement before they are processed. When the SQL statement generates a result table that is not physically stored, these locks are not released until the end of the transaction or until the result table is closed. Otherwise, they are released immediately after the SQL statement has been processed.
  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.
- **ISOLATION LEVEL 2 or 20**: all the tables addressed by the SQL statement are locked in SHARE mode prior to processing. When the SQL statement generates a result table that is not physically stored, these locks are not released until the end of the transaction or until the result table is closed. Otherwise, they are released immediately after the SQL statement has been processed. This table SHARE lock is not assigned to the transaction with SQL statements in which just one table row is processed that is determined by key specifications [Page 55] or by `CURRENT OF <result_table_name>`.
  In addition, an implicit SHARE lock is assigned to the transaction for each row read while an SQL statement is being processed. These SHARE locks can only be released with the UNLOCK statement [Page 211] or by ending the transaction.
  When rows are inserted, updated or deleted, implicit exclusive locks are assigned to the transaction for the rows concerned. These cannot be released until the end of the transaction.
- **ISOLATION LEVEL 3 or 30**: 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. This table SHARE lock is not assigned to the transaction with SQL statements in which just one table row is processed that is determined by key specifications or by `CURRENT OF <result_table_name>`.
  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.
- The default isolation level is ISOLATION LEVEL 1.

The selected isolation level 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 far as consistency is concerned, **various phenomena** can arise through concurrent access to the same database:

- Phenomenon 1: **Dirty Read Phenomenon**
  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 that never actually existed.
- Phenomenon 2: **Non-Repeatable Read Phenomenon**
  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, it either receives the modified row or a message indicating that the row no longer exists.

- Phenomenon 3: **Phantom Phenomenon**
  A transaction T1 executes an SQL statement S that reads a set M of rows that fulfill a search condition [Page 107]. A transaction T2 then uses the INSERT statement or the UPDATE statement to create at least one additional row that satisfies the search condition. If S is subsequently re-executed within T1, the set of read rows will differ from M.

The following table indicates the phenomena that can occur with each isolation level:

	ISO 0	ISO 1	ISO 2	ISO 3
Dirty Read	+	-	-	-
Non Repeatable Read	+	+	-	-
Phantom	+	+	+	-

The lower the value of the isolation level, the higher the degree of concurrency and the lower the guaranteed consistency. This means that a compromise between concurrency and consistency that best suits the requirements of the application at hand must always be found.

TIMEOUT

The TIMEOUT value defines the maximum period of inactivity during a database session. The inactivity period is the time interval between the completion of an SQL statement and the next SQL statement. The session is terminated with a ROLLBACK WORK RELEASE when the specified maximum inactivity period is exceeded.

TIMEOUT values are specified in seconds. The specified TIMEOUT value must be less than or equal to the defined maximum TIMEOUT value.

- A TIMEOUT value created for a user is always a maximum TIMEOUT value.
- A TIMEOUT value defined for a usergroup is the maximum TIMEOUT value for all of the members of this group.
- For all other users, the installation parameter SESSION_TIMEOUT represents the maximum TIMEOUT value.

If no TIMEOUT value is specified, the database uses the maximum TIMEOUT value or the SESSION_TIMEOUT value, depending on which is smaller. The SESSION_TIMEOUT value is determined when the database system is installed.

If the TIMEOUT value is set to 0, the inactivity period is not monitored. This can result in a situation where database resources are not available again even though the associated application was concluded or aborted without a RELEASE statement [Page 212].

Users with the NOT EXCLUSIVE attribute

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.

TERMCHAR SET

Terminal character set name [Page 50]

User Manual: SAP DB → Definition of Terms → Language Support (Termchar Sets [Extern]

SET statement

The SET statement alters the properties of a session [Page 32].

Syntax

```
<set_statement> ::= SET ROLE ALL [EXCEPT <role_name>] | SET ROLE NONE
| SET ROLE <role_name> [IDENTIFIED BY <password>]
| SET ISOLATION LEVEL <unsigned_integer>
```

role_name [Page 48], password [Page 46], unsigned_integer [Page 38]

Explanation

SET ROLE

DEFAULT ROLE in the ALTER USER statement [Page 156] or ALTER USERGROUP statement [Page 157] specifies which of the roles [Page 31] assigned to the current user or user group is active in the user session or group member session. If a role is active, the current user has all the privileges that are included in the role.

If a role that is activated automatically when a session is opened is assigned to the current user with the ALTER USER statement or ALTER USERGROUP statement, it is deactivated when the SET statement is executed if it is not identified by the SET ROLE specification in the SET statement.

- **ALL**: all roles assigned to the current user are active. EXCEPT can be used to exclude specified roles from activation.
- **NONE**: none of the roles is active.
- **Role name specified**: the roles specified here must exist and be assigned to the current user. If a password exists for the role, it must be defined in the SET statement in addition to the owner of the role.
  The role identified with role name is activated.

ISOLATION LEVEL

The isolation level specified in the CONNECT statement [Page 204] defines when locks are set implicitly and how long they are retained. A LOCK option [Page 191] can be defined for individual SQL statements for data retrieval in order to change the lock behavior for this SQL statement.

The SET statement with isolation level changes the lock behavior for all subsequent SQL statements of the current session.

COMMIT statement

A COMMIT statement terminates the current transaction and starts a new one (see transactions [Page 202]).

Syntax

```
<commit_statement> ::= COMMIT [WORK] [KEEP <lock_statement>]
```

lock_statement [Page 209]

Explanation

The COMMIT statement terminates the current transaction. This means that the modifications executed within the transaction are recorded and are thus visible to concurrent users as well.

The COMMIT statement implicitly opens a new transaction. Any locks set, either implicitly or explicitly, within the new transaction are assigned to this transaction. The isolation level specification declared in the CONNECT statement [Page 204] controls the setting of locks in the new transaction.

LOCK statement

The LOCK statement [Page 209] must not contain a WAIT option.

- If a LOCK statement is not specified, the locks assigned to the transaction are released.

- If a LOCK statement is specified, the locks contained in it are maintained after the transaction has ended and are assigned to the new transaction that is opened implicitly. For this purpose, however, the locks specified in the LOCK statement must be assigned to the terminating transaction. Any locks assigned to the terminating transaction that are not specified in the LOCK statement are released.

ROLLBACK statement

The ROLLBACK statement cancels the current transaction and starts a new transaction (see transactions [Page 202]).

Syntax

```
<rollback_statement> ::= ROLLBACK [WORK] [KEEP <lock_statement>]
```

lock_statement [Page 209]

Explanation

The ROLLBACK statement cancels the current transaction. This means that any modifications made within the transaction are reversed.

The ROLLBACK statement implicitly opens a new transaction. Any locks set, either implicitly or explicitly, within the new transaction are assigned to this transaction. The isolation level specification declared in the CONNECT statement [Page 204] controls the setting of locks in the new transaction.

All result tables generated within the current transaction are implicitly deleted at the end of the transaction using the ROLLBACK statement.

LOCK statement

The LOCK statement [Page 209] must not contain a WAIT option.

- If a LOCK statement is not specified, the locks assigned to the transaction are released.
- If a LOCK statement is specified, the locks contained in it are maintained after the transaction has ended and are assigned to the new transaction that is opened implicitly. For this purpose, however, the locks specified in the LOCK statement must be assigned to the terminating transaction. Any locks assigned to the terminating transaction that are not specified in the LOCK statement are released.

SUBTRANS statement

The SUBTRANS statement divides a transaction into units known as subtransactions (see transactions [Page 202]).

Syntax

```
<subtrans_statement> ::= SUBTRANS BEGIN | SUBTRANS END | SUBTRANS ROLLBACK
```

Explanation

SUBTRANS BEGIN

A subtransaction [Page 32] is opened, i.e. the database records the current point in the transaction. This can be followed by any sequence of SQL statements. If this sequence does not contain an additional SUBTRANS BEGIN, all database modifications performed since the SUBTRANS BEGIN can be reversed using a SUBTRANS ROLLBACK.

The sequence, however, can also contain additional SUBTRANS BEGIN statements that open additional subtransactions. This means several nested subtransactions may be open at the same time.

SUBTRANS END

A subtransaction is closed, i.e. the database system "forgets" the point in the transaction recorded with SUBTRANS BEGIN. An open subtransaction must exist for this purpose. If more than one open subtransaction exists, the last opened subtransaction is closed; i.e. it is no longer considered to be an open subtransaction.

SUBTRANS ROLLBACK

SUBTRANS ROLLBACK reverses all database modifications performed within a subtransaction and then closes the subtransaction. Any database modifications performed by any subtransactions within the subtransaction are reversed, irrespective of whether they were ended with SUBTRANS END or SUBTRANS ROLLBACK. All result tables generated within the subtransaction are closed.

An open subtransaction must exist for this purpose. If more than one open subtransaction exists, the last opened subtransaction is rolled back. The subtransaction concerned is then no longer considered open.

Further information

The SUBTRANS statement does not affect locks assigned to the transaction. In particular, SUBTRANS END and SUBTRANS ROLLBACK do not release any locks.

The SUBTRANS statement is particularly useful in keeping the effects of subroutines or database procedures [Page 33] atomic; i.e. it ensures that they either fulfil all their tasks or else have no effect. To this end, a SUBTRANS BEGIN is issued initially. If the subroutine succeeds in fulfilling its task, it is ended with a SUBTRANS END; in the event of an error, a SUBTRANS ROLLBACK is used to reverse all the modifications performed by the subroutine.

The COMMIT statement [Page 207] and the ROLLBACK statement [Page 208] close any open subtransactions implicitly.

LOCK statement

The LOCK statement assigns a lock to the current transaction (see transactions [Page 202]).

Syntax

```
<lock_statement> ::=
LOCK [(WAIT) | (NOWAIT)] <lock_spec> IN SHARE MODE
LOCK [(WAIT) | (NOWAIT)] <lock_spec> IN EXCLUSIVE MODE
LOCK [(WAIT) | (NOWAIT)] <lock_spec> IN SHARE MODE <lock_spec> IN EXCLUSIVE
MODE
LOCK [(WAIT) | (NOWAIT)] <row_spec>... OPTIMISTIC

<lock_spec> ::= TABLE <table_name>,... | <row_spec>...
| TABLE <table_name>,... <row_spec>...
```

row_spec [Page 211], table_name [Page 50]

Explanation

The specified table cannot be a base table, a view table (see table [Page 29]), nor a synonym [Page 30]. If the table name identifies a view table, 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 requires the UPDATE, DELETE or INSERT privilege.

<row_spec>...

A `<row_spec>...` creates a lock for the table row denoted by the key values or a position in a result table.

Specifying a `row_spec` requires that the specified table have a key column; that is, if the table name identifies a view table, this must be modifiable.

TABLE <table_name>,...

If `TABLE <table_name>,...` is specified, a lock is created for the table in question.

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

SHARE

SHARE defines a SHARE lock for the listed objects. If a SHARE lock is set, no concurrent transaction can modify the locked objects.

EXCLUSIVE

EXCLUSIVE defines an exclusive lock for the listed objects. If an exclusive lock is set, no concurrent transaction can modify the locked objects. Concurrent transactions can only read-access the locked objects in isolation level 0.

Exclusive locks for rows that have not been modified yet can be released using the UNLOCK statement [Page 211] before the transaction ends.

OPTIMISTIC

OPTIMISTIC defines an optimistic lock on rows. This lock only makes sense when it is used together with the isolation levels 0, 1, 10, and 15. An update operation of the current user on a row which has been locked by this user using an optimistic lock is only performed if this row has not been updated in the meantime by a concurrent transaction. If this row has been changed in the meantime by a concurrent transaction, the update operation of the current user is rejected. The optimistic lock is released in both cases. If the update operation was successful, an exclusive lock is set for this row. If the update operation was not successful, it should be repeated after reading the row again with or without optimistic lock. In isolation level 0, an explicit lock must be specified for the new read operation. In this way, it can be ensured that the update is done to the current state and that no modifications made in the meantime are lost.

The request of an optimistic lock only collides with an exclusive lock. Concurrent transactions do not collide with an optimistic lock.

- 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.

See also:

Transactions [Page 202]

Locks can be requested either **implicitly or explicitly**. Explicit lock requests are made with the LOCK statement. Whether a lock is requested implicitly and how long it remains assigned to the transaction depends on the isolation level specification in the CONNECT statement [Page 204].

SHARE locks and exclusive locks set on single table rows which have not yet been updated can be released within a transaction. Exclusive locks on updated table rows or table locks cannot be released within a transaction.

The locks assigned to a transaction by the LOCK statement are usually released when the transaction ends provided that the COMMIT statements [Page 207] or the ROLLBACK statements [Page 208] that end a transaction do not contain a LOCK statement.

WAIT/NOWAIT

- If (NOWAIT) is specified, the database does not wait for a lock to be released by another transaction. Instead, it issues an error message if a lock collision occurs. If there is no collision, the requested lock is set.

- In the event of a lock collision, if either the WAIT option is omitted or (WAIT) is specified, the system waits for locks to be released, until the period specified by the installation parameter REQUEST_TIMEOUT has elapsed.

If the database system has to wait too long for locks to be released when setting explicit or implicit locks, it issues a return code to this effect. The user can then respond to this return code, e.g., by terminating the transaction. In this case, the database system does not execute an implicit ROLLBACK WORK.

Deadlock

Whenever the database system recognizes a deadlock caused by explicit or implicit locks, it ends the transaction with an implicit ROLLBACK WORK.

Reproducibility

If reproducible results are needed to read 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 statements or implicitly by using the isolation level 3. This ensures that other users cannot modify the table. To ensure the reproducibility of the SQL statement SELECT DIRECT [Page 196], it is sufficient to implicitly or explicitly lock the row to be read in SHARE mode.

Number of locks

The fewer objects 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 locks that have been set should be released as soon as possible.

If a transaction explicitly or implicitly requests too many row locks (SHARE or exclusive locks) on a table, the database system tries to obtain a table lock instead. If this causes collisions with other locks, the database system continues to request row locks. This means that table locks are obtained without waiting periods. The limit beyond which the system attempts to transform row locks into table locks depends on the installation parameter MAXLOCKS.

ROW specification (row spec)

The ROW specification (`row_spec`) is a syntax element in a LOCK statement [Page 209] or an UNLOCK statement [Page 211].

Syntax

```
<row_spec> ::= ROW <table_name> KEY <key_spec>,...
| ROW <table_name> CURRENT OF <result_table_name>
```

table_name [Page 50], result_table_name [Page 45]

Explanation

For tables defined without key columns, the implicit key column SYSKEY CHAR(8) BYTE can be used in a key specification.

If `CURRENT OF <result_table_name>` is specified, the result table must have been specified with FOR UPDATE.

UNLOCK statement

The UNLOCK statement releases locks on rows.

Syntax

```
<unlock_statement> ::= UNLOCK <row_spec>... IN SHARE MODE
| UNLOCK <row_spec>... IN EXCLUSIVE MODE
| UNLOCK <row_spec>... IN SHARE MODE <row_spec>... IN EXCLUSIVE MODE
| UNLOCK <row_spec>... OPTIMISTIC
```

row_spec [Page 211]

Explanation

SHARE locks, optimistic locks, and exclusive locks (see transactions [Page 202]) set for single table rows that have not yet been updated can be released within a transaction using the UNLOCK statement.

Exclusive locks are created by inserting, updating, or deleting a row or are set in the same way as optimistic locks, by specifying LOCK options [Page 191] in SELECT statements and by LOCK statements [Page 209]. As long as the locked row has not been updated or deleted, the exclusive lock can be cancelled using the UNLOCK statement.

The UNLOCK statement does not fail if the specified lock does not exist or cannot be released.

RELEASE statement

The RELEASE statement terminates a user's transaction [Page 32] and session [Page 32].

Syntax

```
<release_statement> ::= COMMIT [WORK] RELEASE | ROLLBACK [WORK] RELEASE
```

Explanation

Ending a session using a RELEASE statement implicitly deletes all result tables, the data stored in temporary base tables, and the metadata of these tables.

COMMIT WORK RELEASE

The current transaction is aborted without opening a new one. The user session is ended.

If the database system has to reverse the current transaction implicitly, COMMIT WORK RELEASE fails, and a new transaction is opened. The user session is not ended in this case.

ROLLBACK WORK RELEASE

The current transaction is aborted without opening a new one. Any database modifications performed during the current transaction are undone. The user session is ended. ROLLBACK WORK RELEASE has the same effect as a ROLLBACK statement [Page 208] followed by COMMIT WORK RELEASE.

If the accounting function in the database system is activated, information on the session is added to the table SYSACCOUNT.

System Tables

This section describes the system tables that are available in all SQL modes [Page 34]. The owner [Page 44] of these system tables is the user [Page 30] DOMAIN. The owner must be specified when these tables all SQL modes other than INTERNAL.

COLUMNS [Page 213]

CONNECTEDUSERS [Page 214]

COLUMNS

This system table [Page 212] contains the columns in all the tables, views, synonyms, and result tables for which the current user has privileges.

COLUMNS

OWNER	CHAR(32)	Name of the owner of the database object
TABLENAME	CHAR(32)	Name of the database object
COLUMNNAME	CHAR(32)	Name of the column
MODE	CHAR(3)	Type of column (KEY | MAN | OPT)
DATATYPE	CHAR(10)	Data type of the column (BOOLEAN | CHAR | DATE | FIXED | FLOAT | INTEGER | LONG | SMALLINT | TIME | TIMESTAMP)

CODETYPE	CHAR(8)	Code attribute of the column (ASCII | EBCDIC | BYTE | UNICODE)
LEN	FIXED(4)	Length or precision of the column
DEC	FIXED(3)	Number of decimal places in the columns of data type FIXED
COLUMNPRIVILEGES	CHAR(8)	Privileges of the current user for the column
DEFAULT	CHAR(254)	DEFAULT value for the column
DOMAINOWNER	CHAR(32)	Name of the owner of the domain
DOMAINNAME	CHAR(32)	Name of the domain
POS	FIXED(4)	Original position of the column in the table
KEYPOS	FIXED(4)	Original position of the key column in the table
CREATEDATE	DATE	Creation date of the column
CREATETIME	TIME	Creation time of the column
ALTERDATE	DATE	Alter date of the column
ALTERTIME	TIME	Alter date of the column
TABLETYPE	CHAR(8)	Type of table
COMMENT	LONG	Comment on the column

CONNECTEDUSERS

This system table [Page 212] contains all of the users that are currently logged on.

CONNECTEDUSERS

USERNAME	CHAR(32)	User name
TERMID	CHAR(18)	Terminal identification
SESSION	FIXED(10)	Session
CATALOG_CACHE_SIZE	FIXED(10)	Size of the cache for catalog information in this database session
DBPROC_CACHE_SIZE	FIXED(10)	Size of the cache for processing database procedures

CONNECTPARAMETERS

This system table [Page 212] contains information on session-specific parameters.

CONNECTPARAMETERS

SQLMODE	CHAR(8)	SQL mode
ISOLEVEL	FIXED(10)	Isolation level
TIMEOUT	FIXED(10)	Session timeout value
TERMCHARSETNAME	CHAR(32)	Name of the character set used in this database session

CONSTRAINTS

This system table [Page 212] contains the CONSTRAINT definitions for all the tables for which the current user has privileges.

CONSTRAINTS

OWNER	CHAR(32)	Name of the owner of the table
TABLENAME	CHAR(32)	Name of the table with the CONSTRAINT definition
CONSTRAINTNAME	CHAR(32)	Name of the CONSTRAINT definition
DEFINITION	LONG	Text of the CONSTRAINT definition

DBPROCEDURES

This system table [Page 212] contains all the database procedures for which the current user has privileges.

DBPROCEDURES

OWNER	CHAR(32)	Name of the owner of the database procedure
DBPROCNAME	CHAR(32)	Name of the database procedure
PACKAGE	CHAR(32)	Package containing the database procedure
PARAMETER	FIXED(4)	Number of parameters for the database procedure
CREATEDATE	DATE	Creation date of the database procedure
CREATETIME	TIME	Creation time of the database procedure
EXCECUTION_KIND	CHAR(6)	Execution location of the database procedure (INPROC / LOCAL / REMOTE)
SQL_SUPPORT	CHAR(3)	Database procedure may contain SQL statements (YES / NO)
REMOTE_LOCATION	CHAR(132)	Execution location
COMMENT	LONG	Comment on the database procedure

DBPROCPARAMS

This system table [Page 212] contains all the parameters of a database procedures for which the current user has privileges.

DBPROCPARAMS

OWNER	CHAR(32)	Name of the owner of the database procedure
DBPROCNAME	CHAR(32)	Name of the database procedure
PARAMETERNAME	CHAR(32)	Name of the parameter
POS	FIXED(4)	Original position of the parameter in the database procedure
IN/OUT-TYPE	CHAR(6)	Type of parameter (in / out)
DATATYPE	CHAR(10)	Data type of the parameter (BOOLEAN | CHAR | DATE | FIXED | FLOAT | TIME | TIMESTAMP)
CODETYPE	CHAR(8)	Code attribute of the column (ASCII | EBCDIC | BYTE | UNICODE)

LEN	FIXED(4)	Length or precision of the parameter
DEC	FIXED(3)	Number of decimal places of the parameter with data type FIXED
CREATEDATE	DATE	Creation date of the database procedure
CREATETIME	TIME	Creation time of the database procedure

DOMAINCONSTRAINTS

This system table [Page 212] contains the CONSTRAINT definition of a domain.

DOMAINCONSTRAINTS

OWNER	CHAR(32)	Name of the owner of the domain
DOMAINNAME	CHAR(32)	Name of the domain
CONSTRAINTNAME	CHAR(32)	Name of the CONSTRAINT definition
DEFINITION	LONG	Text of the CONSTRAINT definition

DOMAINS

This system table [Page 212] contains all domains.

DOMAINS

OWNER	CHAR(32)	Name of the owner of the domain
DOMAINNAME	CHAR(32)	Name of the domain
DATATYPE	CHAR(10)	Data type of the domain (BOOLEAN | CHAR | DATE | FIXED | FLOAT | INTEGER | LONG | SMALLINT | TIME | TIMESTAMP)
CODETYPE	CHAR(8)	Code attribute of the domain (ASCII | EBCDIC | BYTE | UNICODE)
LEN	FIXED(4)	Length or precision of the domain
DEC	FIXED(3)	Number of decimal places in domains of data type FIXED
DEFAULT	CHAR(254)	Default value of the domain
DEFINITION	LONG	Domain definition text
CREATEDATE	DATE	Creation date of the domain
CREATETIME	TIME	Creation time of the domain
COMMENT	LONG	Comment on the domain

FOREIGNKEYCOLUMNS

This system table [Page 212] contains the columns of all referential CONSTRAINT definitions for which the current user has privileges.

FOREIGNKEYCOLUMNS

OWNER	CHAR(32)	Name of the owner of the table

TABLENAME	CHAR(32)	Name of the table
COLUMNNAME	CHAR(32)	Name of the column
FKEYNAME	CHAR(32)	Name of the referential CONSTRAINT definition
REFOWNER	CHAR(32)	Name of the owner of the referenced table
REFTABLENAME	CHAR(32)	Name of the referenced table
REFCOLUMNNAME	CHAR(32)	Name of the column of the referenced table
RULE	CHAR(18)	Rules for deleting the table
CREATEDATE	DATE	Creation date of the referential CONSTRAINT definition
CREATETIME	TIME	Creation time of the referential CONSTRAINT definition
COMMENT	LONG	Comment on the referential CONSTRAINT definition

FOREIGNKEYS

This system table [Page 212] contains all the referential CONSTRAINT definitions for which the current user has privileges.

FOREIGNKEYS

OWNER	CHAR(32)	Name of the owner of the table
TABLENAME	CHAR(32)	Name of the table
FKEYNAME	CHAR(32)	Name of the referential CONSTRAINT definition
RULE	CHAR(18)	Rules for deleting the table
CREATEDATE	DATE	Creation date of the referential CONSTRAINT definition
CREATETIME	TIME	Creation time of the referential CONSTRAINT definition
COMMENT	LONG	Comment on the referential CONSTRAINT definition

INDEXCOLUMNS

This system table [Page 212] contains detailed information on the indexes for which the current user has privileges.

INDEXECOLUMNS

OWNER	CHAR(32)	Name of the owner of the index
TABLENAME	CHAR(32)	Name of the table
INDEXNAME	CHAR(32)	Index name
TYPE	CHAR(6)	Index type (UNIQUE | NULL)
COLUMNNAME	CHAR(32)	Name of the column
SORT	CHAR(4)	Sort sequence for the column (ASC | DESC)
COLUMNNO	FIXED(4)	Number of the column in the index
DATATYPE	CHAR(10)	Data type of the domain (BOOLEAN | CHAR | DATE | FIXED | FLOAT | INTEGER | LONG | SMALLINT | TIME | TIMESTAMP)

LEN	FIXED(4)	Length or precision of the definition of the column
DISTINCTVALUES	FIXED(10)	Number of different values in the column
PAGECOUNT	FIXED(10)	Number of pages for the index
+AVGLISTLENGTH	FIXED(10)	Average number of rows with the same value, determined for all the columns used in the index
CREATEDATE	DATE	Creation date of the index
CREATETIME	TIME	Creation time of the index
INDEX_USED	FIXED(10)	Frequency of use in search operations
COMMENT	LONG	Comment on the index

INDEXES

This system table [Page 212] contains all the indexes for which the current user has privileges.

INDEXES

OWNER	CHAR(32)	Name of the owner of the index
TABLENAME	CHAR(32)	Name of the table
INDEXNAME	CHAR(32)	Index name
TYPE	CHAR(6)	Index type (UNIQUE | NULL)
CREATEDATE	DATE	Creation date of the index
CREATETIME	TIME	Creation time of the index
INDEX_USED	FIXED(10)	Frequency of use in search operations
DISABLED	CHAR(3)	Index not active for search operations (YES | NO)
COMMENT	LONG	Comment on the index

LOCKS

This system table [Page 212] contains information on locks that have been set.

LOCKS

SESSION	FIXED(10)	Database session identification
TRANSCOUNT	FIXED(10)	Transaction identification in the database session
PROCESS	FIXED(10)	Process identification in the database session
USERNAME	CHAR(32)	User name
DATE	DATE	Creation date of the lock
TIME	TIME	Creation time of the lock
TERMID	CHAR(18)	User terminal identification
LASTWRITE	CHAR(10)	Elapsed time since the last SQL statement for data manipulation
LOCKMODE	CHAR(14)	Type of lock
LOCKSTATE	CHAR(10)	Status of lock

APPLPROCESS	FIXED(10)	Process identification on the client hardware
APPLNODE	CHAR(64)	Client hardware identification
OWNER	CHAR(32)	table owner
TABLENAME	CHAR(32)	Name of the table
TABLEID	CHAR BYTE(8)	Table identification
ROWIDLENGTH	FIXED(3)	Length of the key of the locked row
ROWIDHEX	CHAR BYTE(64)	Start of the key of the locked row in hexadecimal notation
ROWID	CHAR(128)	Start of the key of the locked row

MAPCHARSETS

This system table [Page 212] contains all MAPCHAR SETs.

MAPCHARSETS

MAPCHARSETNAME	CHAR(32)	Name of the MAPCHAR SET
CODE	CHAR(8)	Code attribute for which the MAPCHAR SET was defined (ASCII | EBCDIC)
INTERN	CHAR BYTE(1)	The original form in hexadecimal notation
MAP_CODE	CHAR BYTE(2)	Target form in hexadecimal notation
MAP_CHARACTER	CHAR(2)	Target form in plain text notation

PACKAGES

This system table [Page 212] describes all the packages for which the current user has privileges.

PACKAGES

OWNER	CHAR(32)	Name of the owner of the package
PACKAGE	CHAR(32)	Package name
CREATEDATE	DATE	Creation date of the package
CREATETIME	TIME	Creation time of the package
EXCECUTION_KIND	CHAR(6)	Execution location of the database procedure (INPROC / LOCAL / REMOTE)
SQL_SUPPORT	CHAR(3)	Database procedure may contain SQL statements (YES / NO)
REMOTE_LOCATION	CHAR(132)	Execution location
COMMENT	LONG	Comment on the package

ROLEPRIVILEGES

This system table [Page 212] describes the privileges and roles that were assigned to roles for which the current user has privileges.

Structure

ROLEPRIVILEGES

OWNER	CHAR(32)	Name of the owner of the object that was assigned to the role
TABLENAME	CHAR(32)	Name of the table for which privileges were assigned to the role
ROLE	CHAR(32)	Name of the assigned role
GRANTEE	CHAR(32)	Name of the role to which an assignment was made
PRIVILEGES	CHAR(30)	Assigned privileges
GRANTOR	CHAR(32)	Name of the user who assigned the privileges or role
CREATEDATE	DATE	Date on which the privilege was assigned
CREATETIME	TIME	Time at which the privilege was assigned

ROLES

This system table [Page 212] describes the roles for which the current user has privileges.

ROLES

OWNER	CHAR(32)	Name of the owner of role
ROLE	CHAR(32)	Role name
PASSWORD_REQUIRED	CHAR(3)	Password required to activate the role (YES / NO)
GRANTED	CHAR(3)	Role was assigned to current user (YES / NO)
CREATEDATE	DATE	Creation date of the role
CREATEDATE	TIME	Creation time of the role

SEQUENCES

This system table [Page 212] contains all the sequences for which the current user has privileges.

SEQUENCES

OWNER	CHAR(32)	Name of the owner of the sequence
SEQUENCE_NAME	CHAR(32)	Name of the sequence
MIN_VALUE	FIXED(38)	Minimum value of the sequence
MAX_VALUE	FIXED(38)	Maximum value of the sequence
INCREMENT_BY	FIXED(38)	Value by which the sequence is increased
CYCLE_FLAG	CHAR(1)	Does the sequence start at the minimum value again when the maximum value is reached?
ORDER_FLAG	CHAR(1)	Are the sequence values assigned in the order of the request?
CACHE_SIZE	FIXED(38)	Number of sequence values that are loaded to the cache simultaneously
LAST_NUMBER	FIXED(38)	Last sequence value stored

CREATEDATE	DATE	Creation date of the sequence
CREATETIME	TIME	Creation time of the sequence
COMMENT	LONG	Comment on the sequence

SESSION_ROLES

This system table [Page 212] contains the roles that are active in the current session.

SESSION_ROLES

ROLE	CHAR(32)	Role name

SYNONYMS

This system table [Page 212] contains all the synonyms for which the current user has privileges.

SYNONYMS

OWNER	CHAR(32)	Name of the owner of the synonym
SYNONYMNAME	CHAR(32)	Name of the synonym
PUBLIC	CHAR(3)	Synonym is PUBLIC (YES | NO)
TABLEOWNER	CHAR(32)	Name of the owner of the table
TABLENAME	CHAR(32)	Name of the table
CREATEDATE	DATE	Creation date of the synonym
CREATETIME	TIME	Creation time of the synonym
COMMENT	LONG	Comment on the synonym

TABLEPRIVILEGES

This system table [Page 212] describes the privileges that the current user has for tables.

TABLEPRIVILEGES

TABLE_OWNER	CHAR(32)	Type of table (TABLE | VIEW | SYNONYM | RESULT)
TABLE_NAME	CHAR(32)	Name of the table
GRANTOR	CHAR(32)	Name of the user who assigned the privileges
GRANTEE	CHAR(32)	Name of the user or role to which an assignment was made
PRIVILEGES	CHAR(30)	Assigned privileges
IS_GRANTABLE	CHAR(3)	Privilege to grant the privilege (YES / NO)

TABLES

This system table [Page 212] contains all the tables (base tables, views, synonyms, result tables) for which the current user has privileges.

TABLES

OWNER	CHAR(32)	Name of the owner of the table
TABLENAME	CHAR(32)	Name of the table
PRIVILEGES	CHAR(30)	Privileges of the current user for the table
TYPE	CHAR(8)	Type of table (TABLE | VIEW | SYNONYM | RESULT)
CREATEDATE	DATE	Creation date of the table
CREATETIME	TIME	Creation time of the table
UPDSTATDATE	DATE	Date at which the UPDATE STATISTICS statement was last carried out on the table
UPDSTATTIME	TIME	Time at which the UPDATE STATISTICS statement was last carried out on the table
ALTERDATE	DATE	Change date of the table
ALTERTIME	TIME	Change time of the table
UNLOADED	CHAR(3)	Table is unloaded (YES | NO)
SAMPLE_PERCENT	FIXED(3)	Percentage share of the table to be used to update statistics
SAMPLE_ROWS	FIXED(10)	Number of lines in the table to be used to update statistics
COMMENT	LONG	Comment on the table
TABLEID	CHAR(8) BYTE	Table identification in hexadecimal notation

TERMCHARSETS

This system table [Page 212] contains all the terminal character sets.

TERMCHARSETS

TERMCHARSETNAME	CHAR(32)	The name of the terminal character set
CODE	CHAR(8)	The code attribute for which the terminal character set was defined (ASCII | EBCDIC)
STATE	CHAR(8)	The terminal character set is active (ENABLED | DISABLED)
INTERN	CHAR BYTE(1)	The original form in hexadecimal notation
EXTERN	CHAR BYTE(1)	The terminal character set in hexadecimal notation
COMMENT	CHAR(8)	A comment on the terminal character set.

TRIGGERPARAMS

This system table [Page 212] contains all the parameters of a trigger for which the current user has privileges.

TRIGGERPARAMS

OWNER	CHAR(32)	Name of the owner of a table
TABLENAME	CHAR(32)	Name of the table
TRIGGERNAME	CHAR(32)	Trigger name
PARAMETERNAME	CHAR(32)	Name of the parameter
POS	FIXED(4)	Original position of the parameter in the trigger
NEW/OLD-TYPE	CHAR(3)	Parameter version (NEW | OLD)
DATATYPE	CHAR(10)	Data type of the parameter (BOOLEAN | CHAR | DATE | FIXED | FLOAT | TIME | TIMESTAMP)
CODETYPE	CHAR(8)	Code attribute of the column (ASCII | EBCDIC | BYTE | UNICODE)
LEN	FIXED(4)	Length or precision of the parameter
DEC	FIXED(3)	Number of decimal places for parameters with data type FIXED
CREATEDATE	DATE	Creation date of the trigger
CREATETIME	TIME	Creation time of the trigger

TRIGGERS

This system table [Page 212] contains all the triggers for which the current user has privileges.

TRIGGERS

OWNER	CHAR(32)	Name of the owner of the table
TABLENAME	CHAR(32)	Name of the table for which the trigger was defined
TRIGGERNAME	CHAR(32)	Trigger name
INSERT	CHAR(3)	Trigger type
UPDATE	CHAR(3)	Trigger type
DELETE	CHAR(3)	Trigger type
CREATEDATE	DATE	Creation date of the trigger
CREATETIME	TIME	Creation time of the trigger
DEFINITION	LONG	Trigger definition text
COMMENT	LONG	Comment on the trigger

USERS

This system table [Page 212] contains all users.

USERS

OWNER	CHAR(32)	Name of the owner of the user
GROUPNAME	CHAR(32)	Group name
USERNAME	CHAR(32)	User name
USERMODE	CHAR(8)	Class of the user (SYSDBA | DBA | RESOURCE | STANDARD)
CONNECTMODE	CHAR(8)	Type of connection (MULTIPLE | SINGLE)

PERMLIMIT	FIXED(10)	PERMLIMIT value
TEMPLIMIT	FIXED(10)	TEMPLIMIT value
MAXTIMEOUT	FIXED(10)	TIMEOUT value
COSTWARNING	FIXED(10)	COSTWARNING value
COSTLIMIT	FIXED(10)	COSTLIMIT value
CREATEDATE	DATE	Creation date of the user
CREATETIME	TIME	Creation time of the user
ALTERDATE	DATE	Change date of the user
ALTERTIME	TIME	Change time of the user
PWCREADATE	DATE	Creation date of the password
PWCREATIME	TIME	Creation time of the password
SERVERDB	CHAR(18)	Name of the serverdb
SERVERNODE	CHAR(64)	Server node name of the serverdb
USER_ID	FIXED(10)	User identification
COMMENT	LONG	Comment on the user

VERSIONS

This system table [Page 212] describes the versions of the SAP DB database software.

VERSIONS

KERNEL	CHAR(40)	Version of the SAP DB database software
RUNTIMEENVIRONMENT	CHAR(40)	Version of the runtime environment

VIEWCOLUMNS

This system table [Page 212] contains all the view tables for which the current user has privileges.

VIEWCOLUMNS

OWNER	CHAR(32)	Name of the owner of the view table
VIEWNAME	CHAR(32)	Name of the view table
VIEWCOLUMNNAME	CHAR(32)	Name of the column of the view table
TABLEOWNER	CHAR(32)	Name of the owner of the table to which the used column belongs
TABLENAME	CHAR(32)	Name of the table to which the used column belongs
COLUMNNAME	CHAR(32)	Name that the column originally had in the table
CREATEDATE	DATE	Date on which the view table was created
CREATETIME	TIME	Time at which the view table was created
COMMENT	LONG	Comment on the view table column

VIEWDEFS

This system table [Page 212] contains the definitions of the view tables for which the current user has privileges.

VIEWDEFS

OWNER	CHAR(32)	Name of the owner of the view table
VIEWNAME	CHAR(32)	Name of the view table
LEN	FIXED(4)	Length of the view table definition
DEFINITION	LONG	Text in the view table definition

VIEWS

This system table [Page 212] contains all the view tables for which the current user has privileges.

VIEWS

OWNER	CHAR(32)	Name of the owner of the view table
VIEWNAME	CHAR(32)	Name of the view table
PRIVILEGES	CHAR(30)	Privileges of the user for the view table
TYPE	CHAR(8)	Type of table
CREATEDATE	DATE	Date on which the view table was created
CREATETIME	TIME	Time at which the view table was created
UPDSTATDATE	DATE	Last date on which the UPDATE STATISTICS statement was carried out on the view table
UPDSTATTIME	TIME	Time at which the UPDATE STATISTICS statement was carried out on the view table
ALTERDATE	DATE	Date on which the view table was changed
ALTERTIME	TIME	Time at which the view table was changed
UNLOADED	CHAR(3)	View table is unloaded (YES | NO)
COMMENT	LONG	Comment on the view table

Statistics

The database system addresses disks in units of 8KB. The term ‘page’ is used to refer to these units.

SQL statements for statistics management

UPDATE STATISTICS statement [Page 226]	MONITOR statement [Page 235]

UPDATE STATISTICS statement

The UPDATE STATISTICS statement defines the storage requirements of tables and indexes as well as the value distribution of columns, and stores this information in the database catalog.

Syntax

```
<update statistics statement> ::=
  UPDATE STAT[ISTICS] COLUMN <table name>.<column name> [ESTIMATE
[<sample definition>]]
| UPDATE STAT[ISTICS] COLUMN (<column name>,...) FOR <table name> [ESTIMATE
[<sample definition>]]
| UPDATE STAT[ISTICS] COLUMN (*) FOR <table name> [ESTIMATE
[<sample definition>]]
| UPDATE STAT[ISTICS] <table name> [ESTIMATE [<sample definition>]]
| UPDATE STAT[ISTICS] [<owner>.][<identifier>]* [ESTIMATE
[<sample definition>]]
```

table_name [Page 50], column_name [Page 49], sample_definition [Page 113], owner [Page 44], identifier [Page 40]

Explanation

When the UPDATE STATISTICS statement is executed, information on the table, such as the number of rows, the number of pages used, the sizes of indexes, the value distribution within columns or indexes, etc., is stored in the database catalog. These values are used by the Optimizer to determine the best strategy for executing SQL statements.

You will find information on the Optimizer functions in Optimizer: SAP DB 7.3 [Extern].

The UPDATE STATISTICS statement implicitly performs a COMMIT statement [Page 207] for each base table; i.e. the transaction within which the UPDATE STATISTICS statement has been executed is closed.

When a CREATE INDEX statement [Page 141] is executed, the above-mentioned information is stored in the catalog for the index as well as for the base table for which this index is being defined. No information is stored for other indexes defined on this base table.

The statistical values stored in the database catalog can be retrieved by selecting the system table OPTIMIZERSTATISTICS. Each row of the table describes statistical values of indexes, columns or the size of a table:

OPTIMIZERSTATISTICS

OWNER	CHAR (32)	Owner of the table for which statistical information is available
TABLENAME	CHAR (32)	Name of a table for which statistical information is available
INDEXNAME	CHAR (32)	Name of an index for which statistical information is available
COLUMNNAME	CHAR (32)	Name of a column for which statistical information is available
DISTINCTVALUES	FIXED (10)	Number of different values if the current row describes a column or a single-column index; otherwise, the number of rows in a table
PAGECOUNT	FIXED (10)	Number of pages used by an index if the current row describes an index; number of pages in a base table if the current row describes a table; otherwise; NULL
+AVGLISTLENGTH	FIXED (10)	Average number of keys in an index list if the current row describes an index; otherwise, NULL

<table name>

If a table name is specified, the table must be a non-temporary base table and the user must have a privilege for it.

<column name>

If a column name is specified, this column must exist in specified table.

If * is specified, all columns in the table are assumed.

<identifier>*

Specifying `<identifier>*` has the same effect as issuing the UPDATE STATISTICS statement for all base tables for which the current user has a privilege, and whose table name begins with the `identifier`.

UPDATE STATISTICS *

The SYSDBA can use UPDATE STATISTICS * to execute the UPDATE STATISTICS statement for all base tables, even if the SYSDBA has not been assigned a privilege for these tables.

ESTIMATE

- If ESTIMATE and a `sample definition` are specified, the database system estimates the statistical values by selecting data at random. The number of random selects can be given as number of rows or as percentage. For a specification of 50% or more, all rows are analyzed. The runtime of the UPDATE STATISTICS statement can be considerably reduced by specifying ESTIMATE. In most cases, the precision of the statistical values determined is sufficient.
- If ESTIMATE is specified without a `sample definition`, the database system estimates the statistical values by selecting data at random. The number of selects was defined with the CREATE TABLE statement or an ALTER TABLE statement by means of a sample_definition [Page 113] for the specified table. For a specification of 50% or more, all rows are analyzed. The runtime of the UPDATE STATISTICS statement can be considerably reduced by specifying ESTIMATE. In most cases, the precision of the statistical values determined is sufficient.
- If ESTIMATE is not specified, the database system determines exact statistical values by considering the complete data of the table. For large tables, the runtime can be considerably long.

See also:

Statistics system tables [Page 227]

Statistical system tables

This section describes the statistical system tables. During the complete installation of database system, system tables are created in the database catalog. These system tables can be used to select information on the configuration, structures, and sizes of database objects.

The owner [Page 44] of these statistical system tables is the user [Page 30] SYSDBA. You do not need to specify the owner to access these tables.

DATADEVSPACES [Page 228]

DBPARAMETERS [Page 228]

INDEXSTATISTICS [Page 228]

LOCKLISTSTATISTICS [Page 230]

SERVERDBSTATISTICS [Page 231]

TABLESTATISTICS [Page 232]

TRANSACTIONS [Page 234]

USERSTATISTICS [Page 235]

See also:

UPDATE STATISTICS statement [Page 226]

DATADEVSPACES

This statistical system table [Page 227] contains information on the occupied data devspaces.

DATADEVSPACES

DEVSPACENAME	CHAR (64)	logical name of the data devspace
DEVSPACESIZE	FIXED (10)	Size of the devspace in pages
USEDPAGES	FIXED (10)	Number of devspace pages used
PCTUSEDPAGES	FIXED (10)	Percentage share of used pages
UNUSEDPAGES	FIXED (10)	Number of unused pages
PCTUNUSED	FIXED (10)	Percentage share of unused pages

DBPARAMETERS

This statistical system table [Page 227] contains the parameters defined for the database instance with the Database Manager tool.

DBPARAMETERS

DESCRIPTION	CHAR (18)	description of how to interpret the column VALUE
VALUE	CHAR (64)	Value

INDEXSTATISTICS

This statistical system table [Page 227] contains information about structure and size of indexes.

INDEXSTATISTICS

OWNER	CHAR (32)	Owner of a table
TABLENAME	CHAR (32)	table name
INDEXNAME	CHAR (32)	index name (NULL for unnamed indexes)
COLUMNNAME	CHAR (32)	name of an inverted column
DESCRIPTION	CHAR (40)	Description of how to interpret the following columns (see *Column DESCRIPTION* table)
CHAR_VALUE	CHAR (12)	Alphanumeric value
NUMERIC_VALUE	FIXED (10)	Numeric value

DESCRIPTION column

Value	Explanation
Root pno	NUMERIC_VALUE contains the page number of the B* tree root
FILETYPE	CHAR_VALUE contains the B* tree type
Used pages	NUMERIC_VALUE contains the number of pages used by the index
Index pages	NUMERIC_VALUE contains the number of B* tree index pages used by the index
Leaf pages	NUMERIC_VALUE contains the number of leaf pages used by the index
Index levels	NUMERIC_VALUE contains the number of B* tree index levels
Space used in all pages(%)	NUMERIC_VALUE contains the percentage of the B* tree root page used
Space used in root page(%)	NUMERIC_VALUE contains the percentage of the B* tree root page used
Space used in index pages(%)	NUMERIC_VALUE contains the percentage of the B* tree index pages used
Space used in index pages(%) min	NUMERIC_VALUE contains the minimum percentage of the B* tree index pages used
Space used in index pages(%) max	NUMERIC_VALUE contains the maximum percentage of the B* tree index pages used
Space used in leaf pages(%)	NUMERIC_VALUE contains the percentage of the B* tree leaf pages used
Space used in leaf pages(%) min	NUMERIC_VALUE contains the minimum percentage of the B* tree leaf pages used
Space used in leaf pages(%) max	NUMERIC_VALUE contains the maximum percentage of the B* tree leaf pages used
Secondary keys (index lists)	NUMERIC_VALUE contains the number of different values in the indexed columns
Avg secondary key length	NUMERIC_VALUE contains the average length of the index values
Min secondary key length	NUMERIC_VALUE contains the minimum length of the index values
Max secondary key length	NUMERIC_VALUE contains the maximum length of the index values
Avg separator length	NUMERIC_VALUE contains the average length of a B* tree separator
Min separator length	NUMERIC_VALUE contains the minimum length of the separator
Max separator length	NUMERIC_VALUE contains the maximum length of the separator
Primary keys	NUMERIC_VALUE contains the number of rows in the tables identified by OWNER and TABLENAME

Avg primary keys per list	NUMERIC_VALUE contains the average number of keys per index list
Min primary keys per list	NUMERIC_VALUE contains the minimum number of keys per index list
Max primary keys per list	NUMERIC_VALUE contains the maximum number of keys per index list
Values with selectivity <= 1%	NUMERIC_VALUE contains the number of index lists with a selectivity <= 1%
Values with selectivity <= 5%	NUMERIC_VALUE contains the number of index lists with a selectivity between 5% and 1%
Values with selectivity <= 10%	NUMERIC_VALUE contains the number of index lists with a selectivity between 10% and 5%
Values with selectivity <= 25%	NUMERIC_VALUE contains the number of index lists with a selectivity between 10% and 25%.
Values with selectivity > 25%	NUMERIC_VALUE contains the number of index lists with a selectivity > 25%

LOCKLISTSTATISTICS

This statistical system table [Page 227] contains information on the lock list usage

LOCKLISTSTATISTICS

DESCRIPTION	CHAR (40)	Description of how to interpret the content of the VALUE column (see *Column DESCRIPTION* table)
VALUE	CHAR (12)	Value

DESCRIPTION column

Value	Explanation
MAX LOCKS	VALUE contains the number of available locks in the lock list
TRANS LIST REGIONS	VALUE contains the number of semaphores for transactions
TABLE LIST REGIONS	VALUE contains the number of semaphores for tables
ROW LIST REGIONS	VALUE contains the number of the semaphore for rows
ENTRIES	VALUE contains an internal measure (entries) for the lock list size
USED ENTRIES	VALUE contains the number of entries for locks and lock requests
USED ENTRIES(%)	VALUE contains the percentage of entries used for locks and lock requests
AVG USED ENTRIES	VALUE contains the average number of entries used for locks and lock requests

AVG USED ENTRIES(%)	VALUE contains the average percentage of entries used for locks and lock requests
MAX USED ENTRIES	VALUE contains the maximum number of entries for locks and lock requests
MAX USED ENTRIES(%)	VALUE contains the maximum percentage of entries used for locks and lock requests
LOCK ESCALATION VALUE	VALUE contains the number of table rows from which the lock rows are converted into table locks (lock escalation)
LOCK ESCALATIONS	VALUE contains the number of lock escalations
LOCK COLLISIONS	VALUE contains the number of lock requests that could not be satisfied (immediately)
DEADLOCKS	VALUE contains the number of situations in which at least two transactions collided due to existing or requested locks with the result that this collision could only be solved by terminating a transaction implicitly.
SQL REQUEST TIMEOUTS	VALUE contains the number of lock requests that could not be satisfied because they had exceeded the maximum wait time
TRANSACTIONS HOLDING LOCKS	VALUE contains the number of transactions with assigned locks
TRANSACTIONS REQUESTING LOCKS	VALUE contains the number of transactions requesting locks
CHECKPOINT WANTED	If the VALUE column contains the value TRUE, the lock list is closed, i.e. no exclusive lock can be assigned to a transaction without exclusive lock because a checkpoint was requested
SHUTDOWN WANTED	If the column VALUE contains the value TRUE, the lock list is closed because a shutdown was requested

SERVERDBSTATISTICS

This statistical system table [Page 227] contains information on the database instance usage

SERVERDBSTATISTICS

SERVERDBSIZE	FIXED (10)	Database instance size in pages
MAXDATAPAGENO	FIXED (10)	Largest page number of the database instance
MAXPERM	FIXED (10)	Number of pages of the database instance that can be used for non-temporary objects
USEDPERM	FIXED (10)	Number of pages of the database instance used for non-temporary objects
PCTUSEDPERM	FIXED (10)	Percentage of pages used for non-temporary objects
USEDTMP	FIXED (10)	Number of database instance pages used for temporary objects

PCTUSEDTMP	FIXED (10)	Percentage of pages used for temporary objects
UNUSED	FIXED (10)	Number of unused pages
PCTUNUSED	FIXED (10)	Percentage share of unused pages
UPDATEDPERM	FIXED (10)	Number of modified pages for permanent objects
SERVERDBFULL	CHAR (3)	YES: database instance full NO: database instance not full
LOGSIZE	FIXED (10)	Size of the log area in pages
USEDLOG	FIXED (10)	Number of log pages used
PCTUSEDLOG	FIXED (10)	Percentage of log pages used
PCTLOGNOTSAVED	FIXED (10)	Number of log pages not backed up
PCTLOGNOTSAVED	FIXED (10)	Percentage of log pages not yet saved
LOGSINCEBACKUP	FIXED (10)	Number of log pages written since the last complete or incremental data backup
RESERVEDREDO	FIXED (10)	Number of pages in the data devspace reserved for recovery (redo area)
LOGSEGMENTSIZE	FIXED (10)	Size of a log segment in pages
SAVEPOINTS	FIXED (10)	Number of savepoints executed
CHECKPOINTS	FIXED (10)	Number of checkpoints executed

TABLESTATISTICS

This statistical system table [Page 227] contains information about structure and size of Basis tables.

TABLESTATISTICS

OWNER	CHAR (32)	Table owner
TABLENAME	CHAR (32)	Table name
DESCRIPTION	CHAR (40)	Description of how to interpret the following columns (see *Column DESCRIPTION* table)
CHAR_VALUE	CHAR (12)	Alphanumeric value
NUMERIC_VALUE	FIXED (10)	Numeric value

DESCRIPTION column

Value	Explanation
Root pno	NUMERIC_VALUE contains the page number of the B* tree root
FILETYPE	CHAR_VALUE contains the B* tree type
Used pages	NUMERIC_VALUE contains the number of pages used by the table
Index pages	NUMERIC_VALUE contains the number of pages used by the table in the B* tree index

Leaf pages	NUMERIC_VALUE contains the number of leaf pages used by the table
Index levels	NUMERIC_VALUE contains the number of B* tree index levels
Space used in all pages(%)	NUMERIC_VALUE contains the percentage of the B* tree root page used
Space used in root page(%)	NUMERIC_VALUE contains the percentage of the B* tree root page used
Space used in index pages(%)	NUMERIC_VALUE contains the percentage of the B* tree index pages used
Space used in index pages(%) min	NUMERIC_VALUE contains the minimum percentage of the B* tree index pages used
Space used in index pages(%) max	NUMERIC_VALUE contains the maximum percentage of the B* tree index pages used
Space used in leaf pages(%)	NUMERIC_VALUE contains the percentage of the B* tree leaf pages used
Space used in leaf pages(%) min	NUMERIC_VALUE contains the minimum percentage of the B* tree leaf pages used
Space used in leaf pages(%) max	NUMERIC_VALUE contains the maximum percentage of the B* tree leaf pages used
Rows	NUMERIC_VALUE contains the number of table rows
Avg rows per page	NUMERIC_VALUE contains the average number of rows per page
Min rows per page	NUMERIC_VALUE contains the minimum number of rows per page
Min rows per page	NUMERIC_VALUE contains the maximum number of rows per page
Avg row length	NUMERIC_VALUE contains the average length of rows
Min row length	NUMERIC_VALUE contains the minimum length of rows
Max row length	NUMERIC_VALUE contains the maximum length of rows
Avg key length	NUMERIC_VALUE contains the average length of keys
Min key length	NUMERIC_VALUE contains the minimum length of keys
Max key length	NUMERIC_VALUE contains the maximum length of keys
Avg separator length	NUMERIC_VALUE contains the average length of the separator
Min separator length	NUMERIC_VALUE contains the minimum length of the separator
Max separator length	NUMERIC_VALUE contains the maximum length of the separator

Defined LONG columns	NUMERIC_VALUE contains the number of defined columns with the data type LONG
Avg LONG column length	NUMERIC_VALUE contains the average length of LONG columns
Min LONG column length	NUMERIC_VALUE contains the minimum length of LONG columns
Max LONG column length	NUMERIC_VALUE contains the maximum length of LONG columns
LONG column pages	NUMERIC_VALUE contains the number of pages of all the LONG columns in the table
Avg pages per LONG column	NUMERIC_VALUE contains the average number of pages in the table per LONG column
Min pages per LONG column	NUMERIC_VALUE contains the smallest LONG column in the table in pages
Max pages per LONG column	NUMERIC_VALUE contains the largest LONG column in the table in pages

TRANSACTIONS

This statistical system table [Page 227] contains information on the active transactions of a database instance.

TRANSACTIONS

SESSION	FIXED (10)	Database session identification
TRANSCOUNT	CHAR (20)	Transaction identification
PROCESS	FIXED (10)	User process identification
USERNAME	CHAR (32)	User name
CONNECTDATE	DATE	Date of start of session
CONNECTTIME	TIME	Time of start of session
TERMID	CHAR (18)	Terminal identification
REQTIMEOUT	CHAR (10)	Remaining time until REQUEST_TIMEOUT in seconds
LASTWRITE	CHAR (10)	Elapsed time since last write request in timeout intervals
LOCKMODE	CHAR (14)	Type of lock
LOCKSTATE	CHAR (10)	Status of lock
REQMODE	CHAR (14)	Type of lock request
REQSTATE	CHAR (10)	Status of lock request
APPLPROCESS	FIXED (10)	Identifier of application process
APPLNODEID	CHAR (64)	Identifier of client machine of application process

USERSTATISTICS

This statistical system table [Page 227] contains information on users' resource usage.

USERSTATISTICS

USERNAME	CHAR (32)	User name
USERMODE	CHAR (8)	User class
PERMLIMIT	FIXED (10)	Maximum number of pages that can be used for permanent objects
PERMCOUNT	FIXED (10)	Number of pages currently used for permanent objects
TEMPLIMIT	FIXED (10)	Maximum number of pages that can be used for temporary objects
TEMPCOUNT	FIXED (10)	Number of pages currently used for temporary objects

MONITOR statement

The MONITOR statement can be used to initialize counters for monitoring the database with 0.

Syntax

```
<monitor statement> ::= MONITOR INIT
```

Explanation

The database system always records result counters. These are initialized with 0 when the system is started. The MONITOR statement can be used to reset them to 0.

The event counters recorded by the database system can be retrieved by selecting monitor system tables [Page 235].

Monitor system tables

This section describes the monitor system tables that are created during installation by user [Page 30] SYSDBA. You do not need to specify the owner [Page 44] to access the tables.

The monitor system tables provide results for user SYSDBA and for all users with database user class DBA. The error message `100 - ROW NOT FOUND -` is output for non-authorized users.

The monitor system tables are structured as follows:

DESCRIPTION	CHAR (40)
VALUE	FIXED (20)

Each row contains a counter value that is described by the value contained in the DESCRIPTION column.

The following monitor system tables are provided:

MONITOR_CACHES [Page 236]

MONITOR_LOAD [Page 238]

MONITOR_LOCK [Page 240]

MONITOR_LOG [Page 241]

MONITOR_PAGES [Page 241]

MONITOR_ROW [Page 243]

MONITOR_TRANS [Page 244]

MONITOR_VTRACE [Page 244]

The monitor system table MONITOR [Page 244] contains all the values in the listed monitor system tables.

See also:

MONITOR statement [Page 235]

MONITOR_CACHES

This monitor system table [Page 235] contains information about the operations performed on the different caches.

DESCRIPTION column

Value	Explanation
`Data cache accesses`	Number of accesses to data pages in the data cache
`Data cache accesses successful`	Number of successful accesses to data pages in the data cache
`Data cache accesses unsuccessful`	Number of unsuccessful accesses to data pages in the data cache
`Data cache hit rate (%)`	Percentage of successful accesses to data pages in the data cache
`File directory cache accesses`	Number of accesses to the file directory cache
`File directory cache accesses successful`	Number of successful accesses to the file directory cache
`File directory cache accesses unsuccessful`	Number of unsuccessful accesses to the file directory cache
`File directory cache hit rate (%)`	Percentage of successful accesses to the file directory cache
`FBM cache accesses`	Number of accesses to the Free Block Management cache
`FBM cache accesses successful`	Number of successful accesses to the Free Block Management cache
`FBM cache accesses unsuccessful`	Number of unsuccessful accesses to the Free Block Management cache
`FBM cache hit rate (%)`	Percentage of successful accesses to the Free Block Management cache
`Converter cache accesses`	Number of accesses to the converter cache
`Converter cache accesses successful`	Number of successful accesses to the converter cache
`Converter cache accesses unsuccessful`	Number of unsuccessful accesses to the converter cache

Converter cache hit rate (%)	Percentage of successful accesses to the converter cache
USM cache accesses	Number of accesses to the User Space Management cache
USM cache accesses successful	Number of successful accesses to the User Space Management cache
USM cache accesses unsuccessful	Number of unsuccessful accesses to the User Space Management cache
USM cache hit rate (%)	Percentage of successful accesses to the User Space Management cache
Rollback cache accesses	Number of accesses to the rollback cache
Rollback cache accesses successful	Number of successful accesses to the rollback cache
Rollback cache accesses unsuccessful	Number of unsuccessful accesses to the rollback cache
Rollback cache hit rate (%)	Percentage of successful accesses to the rollback cache
Catalog cache accesses	Number of accesses to the session-specific catalog cache
Catalog cache accesses successful	Number of successful accesses to the session-specific catalog cache
Catalog cache accesses unsuccessful	Number of unsuccessful accesses to the session-specific catalog cache
Catalog cache hit rate (%)	Percentage of successful accesses to the session-specific catalog cache
Sequence cache accesses	Number of accesses to the sequence cache
Sequence cache accesses successful	Number of successful accesses to the sequence cache
Sequence cache accesses unsuccessful	Number of unsuccessful accesses to the sequence cache
Sequence cache hit rate (%)	Percentage of successful accesses to the sequence cache
Data cache logpage accesses	Number of accesses to log pages in the data cache
Data cache logpage accesses successful	Number of successful accesses to log pages in the data cache
Data cache logpage accesses unsuccessful	Number of unsuccessful accesses to log pages in the data cache
Data cache logpage hit rate (%)	Percentage of successful accesses to log pages in the data cache

If the Intel memory extension PSE36 is used on Windows NT, the table contains the following additional information:

DESCRIPTION column

Value	Explanation

PSE36 data cache accesses	Number of accesses to the PSE36 data cache
PSE36 data cache accesses successful	Number of successful accesses to the PSE36 data cache
PSE36 data cache accesses unsuccessful	Number of unsuccessful accesses to the PSE36 data cache
PSE36 data cache hit rate (%)	Percentage of successful accesses to the PSE36 data cache

MONITOR_LOAD

This monitor system table [Page 235] contains information on executed SQL statements and access methods.

DESCRIPTION column

Value	Explanation
SQL commands	Number of executed SQL statements
PREPAREs	Number of parsed SQL statements
EXECUTEs	Number of executions of previously parsed SQL statements
COMMITs	Number of executed COMMIT statements
ROLLBACKs	Number of executed ROLLBACK statements
LOCKs and UNLOCKs	Number of executed LOCK and UNLOCK statements
SUBTRANS BEGINs	Number of SQL statements for opening a subtransaction
SUBTRANS ENDs	Number of SQL statements for concluding a subtransaction
SUBTRANS ROLLBACKs	Number of SQL statements for resetting a subtransaction
CREATEs	Number of executed SQL statements for creating database objects
ALTERs	Number of executed SQL statements for altering database objects
DROPs	Number of executed SQL statements for dropping database objects
SELECTs and FETCHes	Number of executed SQL statements for accessing data
SELECTs and FETCHes, rows read	Number of rows considered for accessing data
SELECTs and FETCHes, rows qual	Number of rows considered for the access of data satisfying conditions
INSERTs	Number of executed SQL statement for inserting rows
INSERTs, rows inserted	Number of rows inserted

UPDATEs	Number of executed SQL statements for updating rows
UPDATEs, rows read	Number of rows considered for updating data
UPDATEs, rows updated	Number of rows updated
DELETEs	Number of executed SQL statements for deleting rows
DELETEs, rows read	Number of rows considered for deleting data
DELETEs, rows deleted	Number of rows deleted
Internal DBPROC calls	Number of DBPROCEDUREs executed
Internal trigger calls	Number of trigger calls
Primary key accesses	Number of search operations with direct access via the key
Primary key accesses, rows read	Number of rows read by direct access via the key
Primary key accesses, rows qual	Number of rows read by direct access using the key, satisfying conditions
Primary key range accesses	Number of search operations with accesses within a range of keys
Primary key range accesses, rows read	Number of rows read within a range of keys
Primary key range accesses, rows qual	Number of rows read within a range of keys, satisfying conditions
Index accesses	Number of search operations with accesses to an index
Index accesses, rows read	Number of rows directly accessed using an index
Index accesses, rows qual	Number of rows indirectly accessed using an index, satisfying conditions
Index range accesses	Number of search operations using an index range
Index range accesses, rows read	Number of rows indirectly accessed using an index range
Index range accesses, rows qual	Number of rows indirectly accessed using an index range, satisfying conditions
Isolated index accesses	Number of search operations completely or partially satisfied by an index without accessing the corresponding base table rows
Isolated index accesses, rows read	Number of keys accessed within the search operations denoted in ISOLATED INDEX ACCESSES
Isolated index accesses, rows qual	Number of keys accessed within the search operations denoted in ISOLATED INDEX ACCESSES, satisfying conditions
Isolated index range accesses	Number of search operations using a part of an index with values within a range without accessing the rows of the base table

Isolated index range accesses, rows read	Number of primary/secondary keys accessed within the search operations denoted by ISOLATED INDEX RANGE ACCESSES
Isolated index range accesses, rows qual	Number of primary/secondary keys accessed within the search operations denoted by ISOLATED INDEX RANGE ACCESSES, satisfying conditions
Table scans	Number of search operations through the whole base table
Table scans, rows read	Number of rows accessed within search operations through the whole base table
Table scans, rows qual	Number of rows accessed within search operations through the whole base table, satisfying conditions
Isolated index scans	Number of search operations for which a complete index was accessed without accessing rows of the base table
Isolated index scans, rows read	Number of index rows accessed within the search operations described under ISOLATED INDEX SCANS
Isolated index scans, rows qual	Number of index rows accessed within the search operations described under ISOLATED INDEX SCANS, satisfying conditions
Memory sorts / sort&merge	Number of sorting operations in the main memory to build temporary indexes
Memory sorts / sort&merge, rows read	Number of rows read to build temporary indexes
Sorts by insertion	Number of sorting operations by inserts
Sorts by insertion, rows inserted	Number of rows inserted during the sorting operation

MONITOR_LOCK

This monitor system table [Page 235] contains information on the lock management operations performed by the database system.

DESCRIPTION column

Value	Explanation
Lock list avg used entries	Average number of entries in the lock list
Lock list max used entries	Maximum number of entries in the lock list
Lock list collisions	Number of lock collisions
Lock list escalations	Number of lock escalations
Lock list inserted row entries	Number of inserted row locks
Lock list inserted table entries	Number of inserted table locks

Detected deadlocks	Number of situations in which at least two transactions collided due to existing or requested locks with the result that this collision could only be solved by terminating a transaction implicitly.
Request timeouts	Number of lock requests that could not be satisfied because they had exceeded the maximum wait time

MONITOR_LOG

This monitor system table [Page 235] contains information on the logging operations (writing log information) performed by the database system.

DESCRIPTION column

Value	Explanation
Log page physical reads	Number of physically read log pages
Log page physical writes	Number of physically written log pages
Log queue pages	Size of the log queue in pages
Log queue max used pages	Maximum number of used log queue pages
Log queue inserts	Number of insert operations in the log queue
Log queue overflows	Number of log queue overflows
Log queue group commits	Number of group COMMITs
Log queue waits for log page write	Number of waiting times for log write operations
Log queue max waits per log page	Maximum number of waiting times per log page
Log queue avg waits per log page	Average number of waiting times per log page

MONITOR_PAGES

This monitor system table [Page 235] contains information on page accesses.

DESCRIPTION column

Value	Explanation
Virtual reads	Number of virtual read operations
Virtual writes	Number of virtual write operations
Physical reads	Number of physical read operations
Physical writes	Number of physical write operations
Catalog virtual read	Number of virtual catalog read operations
Catalog virtual writes	Number of virtual catalog write operations
Catalog physical reads	Number of physical catalog read operations

Catalog	physical writes	Number of physical catalog write operations
FBM page	physical reads	Number of physically read Free Block Management pages
FBM page	physical writes	Number of physically written Free Block Management pages
Converter page	physical reads	Number of physically read converter pages
Converter page	physical writes	Number of physically written converter pages
USM page	physical reads	Number of physically read User Space Management pages
USM page	physical writes	Number of physically written User Space Management pages
Perm page	virtual reads	Number of virtually read permanent pages
Perm page	virtual writes	Number of virtually written permanent pages
Perm page	physical reads	Number of physically read permanent pages
Perm page	physical writes	Number of physically written permanent pages
Temp page	virtual reads	Number of virtually read temporary pages
Temp page	virtual writes	Number of virtually written temporary pages
Temp page	physical reads	Number of physically read temporary pages
Temp page	physical writes	Number of physically written temporary pages
LONG page	virtual reads	Number of virtually read pages for LONG columns
LONG page	virtual writes	Number of virtually written pages for LONG columns
LONG page	physical reads	Number of physically read pages for LONG columns
LONG page	physical writes	Number of physically written pages for LONG columns
Leaf page	virtual reads	Number of virtually read leaf pages
Leaf page	virtual writes	Number of virtually written leaf pages
Leaf page	physical reads	Number of physically read leaf pages
Leaf page	physical writes	Number of physically written leaf pages
Level1 page	virtual reads	Number of virtually read index pages at level 1
Level1 page	virtual writes	Number of virtually written index pages at level 1
Level1 page	physical reads	Number of physically read index pages at level 1
Level1 page	physical writes	Number of physically written index pages at level 1
Level2 page	virtual reads	Number of virtually read index pages at level 2
Level2 page	virtual writes	Number of virtually written index pages at level 2
Level2 page	physical reads	Number of physically read index pages at level 2

Level2 page physical writes	Number of physically written index pages at level 2
Level3 page virtual reads	Number of virtually read index pages at level 3
Level3 page virtual writes	Number of virtually written index pages at level 3
Level3 page physical reads	Number of physically read index pages at level 3
Level3 page physical writes	Number of physically written index pages at level 3

MONITOR_ROW

This monitor system table [Page 235] contains information on operations at row level.

DESCRIPTION column

Value	Explanation
BD add record perm	Number of rows inserted in permanent tables
BD add record temp	Number of rows inserted in temporary tables
BD repl record perm	Number of rows updated in permanent tables
BD repl record temp	Number of rows updated in temporary tables
BD del record perm	Number of rows deleted from permanent tables
BD del record temp	Number of rows deleted from temporary tables
BD get record perm	Number of rows selected from permanent tables specifying the key
BD get record temp	Number of rows selected from temporary tables specifying the key
BD next record perm	Number of rows selected from permanent tables specifying the predecessor key
BD next record temp	Number of rows selected from temporary tables specifying the predecessor key
BD prev record perm	Number of rows selected from permanent tables specifying the successor key
BD prev record temp	Number of rows selected from temporary tables specifying the successor key
BD select direct record	Number of rows selected specifying the key
BD select next record	Number of rows selected specifying the predecessor key
BD select prev record	Number of rows selected specifying the successor key
BD add to index list perm	Number of insert operations in permanent indexes
BD add to index list temp	Number of insert operations in temporary indexes

BD del from index list perm	Number of delete operations from permanent indexes
BD del from index list temp	Number of delete operations from temporary indexes
BD get index list perm	Number of accesses to permanent indexes
BD get index list temp	Number of accesses to temporary indexes

MONITOR_TRANS

This monitor system table [Page 235] contains information on transactions.

DESCRIPTION column

Value	Explanation
SQL commands	Number of SQL statements
Write transactions	Number of transactions with modifying operations
KB calls	Number of requests via the internal communications interface

MONITOR_VTRACE

This monitor system table [Page 235] contains information on vtrace output.

DESCRIPTION column

Value	Explanation
Trace I/O operations	Number of vtrace output operations
Trace I/O operations locked	Number of delayed vtrace output operations

MONITOR

This table is a combination of all other monitor system tables described under monitor system tables [Page 235].

MONITOR

TYPE	CHAR (8)
DESCRIPTION	CHAR (40)
VALUE	FIXED (20)

Restrictions

Maximum values

Identifier length	32 characters
Precision of numeric values	38 digits
Number of tables	unlimited
Internal length of a table row	8088 bytes
Length of a LONG column	2147483647 bytes
Number of columns per table (with KEY)	1024
Number of columns per table (without KEY)	1023
Number of primary key columns per table	512
Sum of internal lengths of all key columns	1024 bytes
Number of named indexes per table	255
Number of indexes, comprising more than one column, per table	255
Number of columns in an index	16
Sum of internal lengths of all columns belonging to an index	1024 bytes
Number of referential CONSTRAINT defintions (foreign key dependencies) per table	unlimited
Number of columns in a referential CONSTRAINT definition	16
Number of triggers per table	3
Number of rows per table	unlimited
Number of result columns in a SELECT instruction	1023
Number of join tables in a SELECT statement	64
Number of join conditions in a WHERE clause of a SELECT statement	128
Number of correlated columns in an SQL statement	64
Number of correlated columns in an SQL statement	16
Number of the columns in an ORDER or GROUP clause	128
Length of the columns in an ORDER or GROUP clause	1020 bytes
Number of parameters in an SQL statement	2000
Length of an SQL statement (Configuaration parameter _PACKET_SIZE less administration overhead)	min. 16000 bytes

Syntax List

The syntax notation [Page 246] used in this document is BNF.

The syntax rules are specified in the following form:

```
Clause ::=
  Rule
```

If you want an explanation of the syntax rules, you can use the clause link to go to the relevant part of the Reference Manual. As a result, you exit the syntax list itself.

If further syntax rules are required for the individual syntax modules, you can access these by selecting the relevant links in the left-hand part of the syntax rules, or by directly selecting the syntax module in the alphabetical syntax list.

Syntax Notation

This documentation uses the BNF syntax notation with the following conventions:

	Explanation
KEYWORDS	Keywords are shown in uppercase letters for the sake of clarity. They can be entered in uppercase or lowercase letters.
<xyz>	Terms in angle brackets are placeholders for syntactical units explained in this document. Do not use angle brackets when entering an SQL statement.
clause ::= rule	Clauses are the building blocks of SQL statements. Rules describe how these building blocks are put together to form more complex clauses and also dictate the notation that is used.
$clause_1$ $clause_2$	The two clauses are written one after the other, separated by at least one blank.
[clause]	Optional clause. This clause can be ignored. Do not use square brackets when entering an SQL statement.
Clause1 | clause2 | ... | $clause_n$	Alternative clauses. You can use exactly one of these clauses.
Clause,...	The clause can be repeated as often as required. The individual repetitions must be written one after the other and separated by a comma and any number of blanks.
Clause...	The clause can be repeated as often as required. The individual repetitions must be written directly one after the other without a separating comma or blank.

add_definition

```
<add_definition [Page 128]> ::=
  ADD <column_definition [Page 250]>,...
| ADD (<column_definition>,...)
| ADD <constraint_definition [Page 251]>
| ADD <referential_constraint_definition [Page 275]>
| ADD <key_definition [Page 264]>
```

alias_name

<alias_name [Page 43]> ::=
  <identifier [Page 262]>

all_function

<all_function [Page 88]> ::=
  <set_function_name [Page 281]> ([ALL] <expression [Page 258]>)

alter_definition

<alter_definition [Page 129]> ::=
  ALTER CONSTRAINT <constraint_name [Page 251]> CHECK <search_condition [Page 280]>
| ALTER <key_definition [Page 264]>

alter_index_statement

<alter_index_statement [Page 143]> ::=
  ALTER INDEX <index_name [Page 263]> [ON <table_name [Page 286]>] ENABLE
| ALTER INDEX <index_name> [ON <table_name>] DISABLE

alter_password_statement

<alter_password_statement [Page 159]>::=
  ALTER PASSWORD <old_password [Page 272]> TO <new_password>
| ALTER PASSWORD <user_name [Page 288]> <new_password>

alter_table_statement

<alter_table_statement [Page 127]> ::=
  ALTER TABLE <table_name [Page 286]> <add_definition [Page 246]>
| ALTER TABLE <table_name> <drop_definition [Page 256]>
| ALTER TABLE <table_name> <alter_definition [Page 247]>
| ALTER TABLE <table_name> <column_change_definition [Page 249]>
| ALTER TABLE <table_name> <modify_definition [Page 267]>
| ALTER TABLE <table_name> <referential_constraint_definition [Page 275]>
| ALTER TABLE <table_name> DROP FOREIGN KEY <referential_constraint_name [Page 275]>
| ALTER TABLE <table_name> <sample_definition [Page 279]>

alter_user_statement

<alter_user_statement [Page 156]> ::=
  ALTER USER <user_name [Page 288]> [<user_mode [Page 288]>]
    [PERMLIMIT <unsigned_integer [Page 287]> | PERMLIMIT NULL]
    [TEMPLIMIT <unsigned_integer> | TEMPLIMIT NULL]
    [TIMEOUT <unsigned_integer> | TIMEOUT NULL]
    [COSTWARNING <unsigned_integer> | COSTWARNING NULL]
    [COSTLIMIT <unsigned_integer> | COSTLIMIT NULL]
    [DEFAULT ROLE ALL [EXCEPT <role_name [Page 278]>] | DEFAULT ROLE NONE
    | DEFAULT ROLE <role_name> [IDENTIFIED BY <password [Page 272]>]]
    [[NOT] EXCLUSIVE]

alter_usergroup_statement

```
<alter_usergroup_statement [Page 157]> ::=
  ALTER USERGROUP <usergroup_name [Page 288]> [<usergroup_mode [Page 288]>]
    [PERMLIMIT <unsigned_integer [Page 287]> | PERMLIMIT NULL]
    [TEMPLIMIT <unsigned_integer> | TEMPLIMIT NULL]
    [TIMEOUT <unsigned_integer> | TIMEOUT NULL]
    [COSTWARNING <unsigned_integer> | COSTWARNING NULL]
    [COSTLIMIT <unsigned_integer> | COSTLIMIT NULL]
    [DEFAULT ROLE ALL [EXCEPT <role_name [Page 278]>] | DEFAULT ROLE NONE
    | DEFAULT ROLE <role_name> [IDENTIFIED BY <password [Page 272]>]]
    [[NOT] EXCLUSIVE]
```

argument

```
<argument [Page 145]> ::=
  <identifier [Page 262]>
```

arithmetic_function

```
<arithmetic_function [Page 56]> ::=
  TRUNC ( <expression [Page 258]>[, <expression>] )
| ROUND ( <expression>[, <expression>] )
| NOROUND ( <expression> )
| FIXED ( <expression>[, <unsigned_integer [Page 287]> [, <unsigned_integer] ] )
| FLOAT ( <expression>[, <unsigned_integer> ] )
| CEIL ( <expression> )
| FLOOR ( <expression> )
| SIGN ( <expression> )
| ABS ( <expression> )
| POWER ( <expression>, <expression> )
| EXP ( <expression> )
| SQRT ( <expression> )
| LN ( <expression> )
| LOG ( <expression>, <expression> )
| PI
| LENGTH ( <expression> )
| INDEX ( <string_spec [Page 285]>, <string_spec> [,<expression>[, <expression>]
] )
```

assignment_statement

```
<assignment_statement [Page 147]> ::=
  SET <variable_name [Page 288]> = <expression [Page 258]>
```

between_predicate

```
<between_predicate [Page 94]> ::=
  <expression [Page 258]> [NOT] BETWEEN <expression> AND <expression>
```

boolean_factor

<boolean_factor [Page 108]> ::=
  [NOT] <predicate [Page 273]>
| [NOT] (<search_condition [Page 280]>)

boolean_term

<boolean_term [Page 107]> ::=
  <boolean_factor [Page 249]>
| <boolean_term> AND <boolean_factor>

call_statement

<call_statement [Page 173]> ::=
  CALL <dbproc_name [Page 254]> [(<expression [Page 258]>,...)] [WITH COMMIT]

cascade_option

<cascade_option [Page 127]> ::=
  CASCADE
| RESTRICT

character

<character [Page 35]> ::=
  <digit [Page 256]>
| <letter [Page 265]>
| <extended_letter [Page 259]>
| <hex_digit [Page 262]>
| <language_specific_character [Page 265]>
| <special_character [Page 283]>

close_statement

<close_statement [Page 195]> ::=
  CLOSE [<result_table_name [Page 278]>]

column_attributes

<column_attributes [Page 119]> ::=
  [<key_or_not_null_spec [Page 264]>] [<default_spec [Page 255]>] [UNIQUE]
[<constraint_definition [Page 251]>]
    [REFERENCES <referenced_table [Page 275]> [(<referenced_column [Page 275]>)]
[<delete_rule [Page 255]>]]

column_change_definition

<column_change_definition [Page 127]> ::=
  COLUMN <column_name [Page 250]> NOT NULL

```
| COLUMN <column_name> DEFAULT NULL
| COLUMN <column_name> ADD <default_spec [Page 255]>
| COLUMN <column_name> ALTER <default_spec>
| COLUMN <column_name> DROP DEFAULT
```

column_definition

```
<column_definition [Page 114]> ::=
  <column_name [Page 250]> <data_type [Page 254]> [<column_attributes [Page 249]>]
| <column_name> <domain_name [Page 256]> [<column_attributes>]
```

column_list

```
<column_list [Page 149]> ::=
  <column_name [Page 250]>
| <column_list>, <column_name>
```

column_name

```
<column_name [Page 49]> ::=
  <identifier [Page 262]>
```

column_spec

```
<column_spec [Page 51]> ::=
  <column_name [Page 250]>
| <table_name [Page 286]>.<column_name>
| <reference_name [Page 275]>.<column_name>
| <result_table_name [Page 278]>.<column_name>
```

comment

```
<comment [Page 143]> ::=
  <string_literal [Page 285]>
| <parameter_name [Page 272]>
```

comment_on_statement

```
<comment_on_statement [Page 143]> ::=
  COMMENT ON <object_spec [Page 271]> IS <comment [Page 250]>
```

commit_statement

```
<commit_statement [Page 207]> ::=
  COMMIT [WORK] [KEEP <lock_statement [Page 266]>]
```

comp_op

<comp_op [Page 97]> ::=
  <
| >
| <>
| !=
| =
| <=
| >=
| ¬= | ¬< | ¬> (for machines with EBCDIC code)
| ~= | ~< | ~> (for machines with ASCII code)

comparison_predicate

<comparison_predicate [Page 95]> ::=
  <expression [Page 258]> <comp_op [Page 250]> <expression>
| <expression> <comp_op> <subquery [Page 285]>
| <expression_list [Page 258]> <equal_or_not [Page 258]> (<expression_list>)
| <expression_list> <equal_or_not> <subquery>

connect_option

<connect_option [Page 204]> ::=
  SQLMODE <INTERNAL | ANSI | DB2 | ORACLE>
| ISOLATION LEVEL <unsigned_integer [Page 287]>
| TIMEOUT <unsigned_integer>
| TERMCHAR SET <termchar_set_name [Page 286]>

connect_statement

<connect_statement [Page 204]> ::=
  CONNECT <parameter_name [Page 272]> IDENTIFIED BY <parameter_name>
[<connect_option [Page 251]>...]
| CONNECT <parameter_name> IDENTIFIED BY <password [Page 272]>
[<connect_option>...]
| CONNECT <user_name [Page 288]> IDENTIFIED BY <parameter_name>
[<connect_option>...]
| CONNECT <user_name> IDENTIFIED BY <password> [<connect_option>...]

constraint_definition

<constraint_definition [Page 122]> ::=
  CHECK <search_condition [Page 280]>
| CONSTRAINT <search_condition>
| CONSTRAINT <constraint_name [Page 251]> CHECK <search_condition>

constraint_name

<constraint_name [Page 43]> ::=
  <identifier [Page 262]>

conversion_function

<conversion_function [Page 81]> ::=
  NUM (<expression [Page 258]>)
| CHR (<expression>[, <unsigned_integer [Page 287]>])
| HEX (<expression>)
| CHAR (<expression>[, <datetimeformat [Page 254]>])

create_dbproc_statement

<create_dbproc_statement [Page 145]> ::=
  CREATE DBPROC <procedure_name [Page 273]> [(<formal_parameter [Page 260]>,..)] AS
<routine [Page 279]>

create_domain_statement

<create_domain_statement [Page 134]> ::=
  CREATE DOMAIN <domain_name [Page 256]> <data_type [Page 254]> [<default_spec [Page 255]>] [<constraint_definition [Page 251]>]

create_index_statement

<create_index_statement [Page 141]> ::=
  CREATE [UNIQUE] INDEX <table_name [Page 286]>.<column_name [Page 250]> [ASC | DESC]
| CREATE [UNIQUE] INDEX <index_name [Page 263]> ON <table_name> (<column_name> [ASC | DESC],...)

create_role_statement

<create_role_statement [Page 160]> ::=
  CREATE ROLE <role_name [Page 278]> [IDENTIFIED BY <password [Page 272]>]

create_sequence_statement

<create_sequence_statement [Page 135]> ::=
  CREATE SEQUENCE [<owner [Page 272]>.]<sequence_name [Page 281]>
    [INCREMENT BY <integer [Page 264]>]
    [START WITH <integer>]
    [MAXVALUE <integer> | NOMAXVALUE]
    [MINVALUE <integer> | NOMINVALUE]
    [CYCLE | NOCYCLE]
    [CACHE <unsigned_integer [Page 287]> | NOCACHE]
    [ORDER | NOORDER]

create_table_statement

<create_table_statement [Page 111]> ::=
  CREATE TABLE <table_name [Page 286]> (<column_definition>
    [,<table_description_element [Page 285]>,...]) [IGNORE ROLLBACK]
[<sample_definition [Page 279]>]
| CREATE TABLE <table_name> [(<table_description_element>,...)]
    [IGNORE ROLLBACK] [<sample_definition>] AS <query_expression [Page 274]>
[<duplicates_clause [Page 258]>]
| CREATE TABLE <table_name> LIKE <table_name> [IGNORE ROLLBACK]

create_table_temp

<create_table_temp [Page 284]> :: =
  <create_table_statement [Page 252]> for creating temporary tables,
    that is, the table_name [Page 286] in the CREATE TABLE statement
    must have the format TEMP.<identifier [Page 262]>.

create_trigger_statement

<create_trigger_statement [Page 149]> ::=
  CREATE TRIGGER <trigger_name [Page 286]> FOR <table_name [Page 286]> AFTER
<trigger_event [Page 286],..>
    EXECUTE (<routine [Page 279]>) [WHENEVER <search_condition [Page 280]>]

create_user_statement

<create_user_statement [Page 151]> ::=
  CREATE USER <user_name [Page 288]> PASSWORD <password [Page 272]>
    [<user_mode [Page 288]>]
    [PERMLIMIT <unsigned_integer [Page 287]>]
    [TEMPLIMIT <unsigned_integer>]
    [TIMEOUT <unsigned_integer>]
    [COSTWARNING <unsigned_integer>]
    [COSTLIMIT <unsigned_integer>]
    [[NOT] EXCLUSIVE]
| CREATE USER <user_name> PASSWORD <password> LIKE <source_user [Page 283]>
| CREATE USER <user_name> PASSWORD <password> USERGROUP <usergroup_name [Page
288]>

create_usergroup_statement

<create_usergroup_statement [Page 153]> ::=
  CREATE USERGROUP <usergroup_name [Page 288]>
    [<usergroup_mode [Page 288]>]
    [PERMLIMIT <unsigned_integer [Page 287]>]
    [TEMPLIMIT <unsigned_integer>]
    [TIMEOUT <unsigned_integer>]
    [COSTWARNING <unsigned_integer>]
    [COSTLIMIT <unsigned_integer>]
    [[NOT] EXCLUSIVE]

create_view_statement

<create_view_statement [Page 137]> ::=
  CREATE [OR REPLACE] VIEW <table_name [Page 286]> [(<alias_name [Page 247]>,...)]
    AS <query_expression [Page 274]> [WITH CHECK OPTION]

data_type

<data_typ [Page 114]> ::=
  CHAR[ACTER] [(<unsigned_integer [Page 287]>)] [ASCII | BYTE | EBCDIC | UNICODE]
| VARCHAR [(<unsigned_integer>)] [ASCII | BYTE | EBCDIC | UNICODE]
| LONG [VARCHAR] [ASCII | BYTE | EBCDIC | UNICODE]
| BOOLEAN
| FIXED (<unsigned_integer> [,<unsigned_integer>])
| FLOAT (<unsigned_integer>)
| INT[EGER]
| SMALLINT
| DATE
| TIME
| TIMESTAMP

date_function

<date_function [Page 73]> ::=
  ADDDATE (<date_or_timestamp_expression [Page 254]>, <expression [Page 258]>)
| SUBDATE (<date_or_timestamp_expression>, <expression>)
| DATEDIFF (<date_or_timestamp_expression>, <date_or_timestamp_expression>
)
| DAYOFWEEK (<date_or_timestamp_expression>)
| WEEKOFYEAR (<date_or_timestamp_expression>)
| DAYOFMONTH (<date_or_timestamp_expression>)
| DAYOFYEAR (<date_or_timestamp_expression>)
| MAKEDATE (<expression>, <expression>)
| DAYNAME (<date_or_timestamp_expression>)
| MONTHNAME (<date_or_timestamp_expression>)

date_or_timestamp_expression

<date_or_timestamp_expression [Page 75]> ::=
  <expression [Page 258]>

datetimeformat

<datetimeformat [Page 53]> ::=
  EUR
| INTERNAL
| ISO
| JIS
| USA

dbproc_name

<dbproc_name [Page 44]> ::=
  [<owner [Page 272]>.]<procedure_name [Page 273]>

declare_cursor_statement

<declare_cursor_statement [Page 175]> ::=
  DECLARE <result_table_name [Page 278]> CURSOR FOR <select_statement [Page 281]>

default_predicate

<default_predicate [Page 97]> ::=
  <column_spec [Page 250]> <comp_op [Page 250]> DEFAULT

default_spec

<default_spec [Page 120]> ::=
  DEFAULT <literal [Page 265]>
| DEFAULT NULL
| DEFAULT USER
| DEFAULT USERGROUP
| DEFAULT DATE
| DEFAULT TIME
| DEFAULT TIMESTAMP
| DEFAULT TRUE
| DEFAULT FALSE
| DEFAULT TRANSACTION
| DEFAULT STAMP
| DEFAULT SERIAL[(<unsigned_integer [Page 287]>)]

delete_rule

<delete_rule [Page 124]> ::=
  ON DELETE CASCADE
| ON DELETE RESTRICT
| ON DELETE SET DEFAULT
| ON DELETE SET NULL

delete_statement

<delete_statement [Page 171]> ::=
  DELETE [FROM] <table_name [Page 286]> [<reference_name [Page 275]>]
    [KEY <key_spec [Page 264]>,...] [WHERE <search_condition [Page 280]>]
| DELETE [FROM] <table_name> [<reference_name>]
    WHERE CURRENT OF <result_table_name [Page 278]>

Delimiter token

<delimiter_token [Page 41]> ::=
  (|) | , | . | + | - | * | / | < | > | <> | != | = | <= | >=
| ¬= | ¬< | ¬> (for machines with EBCDIC code)
| ~= | ~< | ~> (for machines with ASCII code)

derived_column

<derived_column [Page 183]> ::=
  <expression [Page 258]> [[AS] <result_column_name [Page 278]>]
| <result_column_name> = <expression>

digit

<digit [Page 35]> ::=
  0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

digit_sequence

<digit_sequence [Page 38]> ::=
  <digit [Page 256]>...

distinct_function

<distinct_function [Page 183]>::=
  <set_function_name [Page 281]> (DISTINCT <expression [Page 258]>)

distinct_spec

<distinct_spec [Page 183]> ::=
  DISTINCT
| ALL

domain_name

<domain_name [Page 44]> ::=
  [<owner [Page 272]>.]<identifier [Page 262]>

double_quotes

<double_quotes [Page 41]> ::=
  "

drop_dbproc_statement

<drop_dbproc_statement [Page 148]> ::=
  DROP DBPROC <dbproc_name [Page 254]>

drop_definition

<drop_definition [Page 130]> ::=
  DROP <column_name [Page 250]>,... [<cascade_option [Page 249]>] [RELEASE SPACE]
| DROP (<column_name>,...) [<cascade_option>] [RELEASE SPACE]
| DROP CONSTRAINT <constraint_name [Page 251]>
| DROP PRIMARY KEY

drop_domain_statement

<drop_domain_statement [Page 134]> ::=
  DROP DOMAIN <domain_name [Page 256]>

drop_index_statement

<drop_index_statement [Page 142]> ::=
  DROP INDEX <table_name>.<column_name [Page 250]>
| DROP INDEX <index_name [Page 263]> [ON <table_name [Page 286]>]

drop_role_statement

<drop_role_statement [Page 160]> ::=
  DROP ROLE <role_name [Page 278]>

drop_sequence_statement

<drop_sequence_statement [Page 136]> ::=
  DROP SEQUENCE [<owner [Page 272]>.]<sequence_name [Page 281]>

drop_synonym_statement

<drop_synonym_statement [Page 136]> ::=
  DROP [PUBLIC] SYNONYM [<owner [Page 272]>.]<synonym_name [Page 285]>

drop_table_statement

<drop_table_statement [Page 127]> ::=
  DROP TABLE <table_name [Page 286]> [<cascade_option [Page 249]>]

drop_trigger_statement

<drop_trigger_statement [Page 150]> ::=
  DROP TRIGGER <trigger_name [Page 286]> OF <table_name [Page 286]>

drop_user_statement

<drop_user_statement [Page 155]> ::=
  DROP USER <user_name [Page 288]> [<cascade_option [Page 249]>]

drop_usergroup_statement

<drop_usergroup_statement [Page 155]> ::=
  DROP USERGROUP <usergroup_name [Page 288]> [<cascade_option [Page 249]>]

drop_view_statement

<drop_view_statement [Page 141]> ::=
  DROP VIEW <table_name [Page 286]> [<cascade_option [Page 249]>]

duplicates_clause

<duplicates_clause [Page 166]> ::=
  REJECT DUPLICATES
| IGNORE DUPLICATES
| UPDATE DUPLICATES

equal_or_not

<equal_or_not [Page 97]> ::=
  <>
| =
| ¬= (for machines with EBCDIC code)
| ~= (for machines with ASCII code)

exists_predicate

<exists_predicate [Page 97]> ::=
  EXISTS <subquery [Page 285]>

exists_table_statement

<exists_table_statement [Page 134]> ::=
  EXISTS TABLE <table_name [Page 286]>

explain_statement

<explain_statement [Page 201]> ::=
  EXPLAIN [(<result_table_name [Page 278]>)] <query_statement [Page 274]>
| EXPLAIN [(<result_table_name>)] <single_select_statement [Page 282]>

exponent

<exponent [Page 38]> ::=
  [<sign [Page 282]>][[<digit [Page 256]>]<digit>]<digit>

expression

<expression [Page 90]> ::=
  <term [Page 286]>
| <expression> + <term>
| <expression> - <term>

expression_list

<expression_list [Page 90]> ::=
  (<expression [Page 258]>,...)

extended_expression

<extended_expression [Page 168]> ::=
  <expression [Page 258]>
| DEFAULT
| STAMP

extended_letter

<extended_letter [Page 35]> ::=
  #
| @
| $

extended_value_spec

<extended_value_spec [Page 52]> ::=
  <value_spec [Page 288]>
| DEFAULT
| STAMP

extraction_function

<extraction_function [Page 77]> ::=
  YEAR (<date_or_timestamp_expression [Page 254]>)
| MONTH (<date_or_timestamp_expression>)
| DAY (<date_or_timestamp_expression>)
| HOUR (<time_or_timestamp_expression [Page 286]>)
| MINUTE (<time_or_timestamp_expression>)
| SECOND (<time_or_timestamp_expression>)
| MICROSECOND (<expression [Page 258]>)
| TIMESTAMP (<expression>[, <expression>])
| DATE (<expression>)
| TIME (<expression>)

factor

<factor [Page 92]> ::=
  [<sign [Page 282]>] <value_spec [Page 288]>
| [<sign>] <column_spec [Page 250]>
| [<sign>] <function_spec [Page 261]>
| [<sign>] <set_function_spec [Page 282]>
| <expression [Page 258]>

fetch_statement

<fetch_statement [Page 193]> ::=
  FETCH [FIRST | LAST | NEXT | PREV | <position [Page 273]> | SAME]
[<result_table_name [Page 278]>]
    INTO <parameter_spec [Page 272]>,...

final_select

<final_select [Page 176]> ::=
  <select_statement [Page 281]>

first_character

<first_character [Page 41]> ::=
  <letter [Page 265]>
| <extended_letter [Page 259]>
| <language_specific_character [Page 265]>

first_password_character

<first_password_character [Page 46]> ::=
  <letter [Page 265]>
| <extended_letter [Page 259]>
| <language_specific_letter [Page 265]>
| <digit [Page 256]>

fixed_point_literal

<fixed_point_literal [Page 37]> ::=
  [<sign [Page 282]>]<digit_sequence [Page 256]>[.<digit_sequence>]
| [sign]<digit_sequence>.
| [sign].<digit_sequence>

floating_point_literal

<floating_point_literal [Page 38]> ::=
  <mantissa [Page 266]>E<exponent [Page 258]>
| <mantissa>e<exponent>

formal_parameter

<formal_parameter [Page 145]> ::=
  IN <argument [Page 248]> <data_type [Page 254]>
| OUT <argument> <data_type>
| INOUT <argument> <data_type>

from_clause

<from_clause [Page 185]> ::=
  FROM <from_table_spec [Page 260]>,...

from_table_spec

<from_table_spec [Page 186]> ::=
  <table_name [Page 286]> [<reference_name [Page 275]>]
| <result_table_name [Page 278]> [<reference_name>]
| (<query_expression [Page 274]>) [<reference_name>]
| <joined_table [Page 264]>

function_spec

<function_spec [Page 55]> ::=
  <arithmetic_function [Page 248]>
| <trigonometric_function [Page 286]>
| <string_function [Page 284]>
| <date_function [Page 254]>
| <time_function [Page 286]>
| <extraction_function [Page 259]>
| <special_function [Page 283]>
| <conversion_function [Page 252]>

grant_statement

<grant_statement [Page 161]> ::=
  GRANT <priv_spec [Page 273]>,... TO <grantee [Page 261]>,... [WITH GRANT OPTION]
| GRANT EXECUTE ON <dbproc_name [Page 254]> TO <grantee>,...
| GRANT SELECT ON <sequence_name [Page 281]> TO <grantee>,... [WITH GRANT OPTION]

grant_user_statement

<grant_user_statement [Page 158]> ::=
  GRANT USER <granted_users [Page 261]> [FROM <user_name [Page 288]>] TO <user_name>

grant_usergroup_statement

<grant_usergroup_statement [Page 159]> ::=
  GRANT USERGROUP <granted_usergroups [Page 261]> [FROM <user_name [Page 288]>] TO <user_name>

granted_usergroups

<granted_usergroups [Page 159]> ::=
  <usergroup_name [Page 288]>,...
| *

granted_users

<granted_users [Page 158]> ::=
  <user_name [Page 288]>,...
| *

grantee

<grantee [Page 162]> ::=
  PUBLIC
| <user_name [Page 288]>
| <usergroup_name [Page 288]>
| <role_name [Page 278]>

group_clause

<group_clause [Page 188]> ::=
  GROUP BY <expression [Page 258]>,...

having_clause

<having_clause [Page 189]> ::=
  HAVING <search_condition [Page 280]>

hex_digit

<hex_digit [Page 36]> ::=
  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_seq [Page 37]> ::=
  <hex_digit [Page 262]><hex_digit>
| <hex_digit_seq><hex_digit><hex_digit>

hex_literal

<hex_literal [Page 37]> ::=
  x''
| X''
| x'<hex_digit_seq [Page 262]>'
| X'<hex_digit_seq>'

hours

<hours [Page 77]> ::=
  <expression [Page 258]>

identifier

<identifier [Page 40]> ::=
  <simple_identifier [Page 282]>
| <double quotes [Page 256]><special_identifier [Page 283]><double_quotes>

identifier_tail_character

<identifier_tail_character [Page 41]> ::=
  <letter [Page 265]>
| <extended_letter [Page 259]>
| <language_specific_character [Page 265]>
| <digit [Page 256]>
| <underscore [Page 287]>

if_statement

<if_statement [Page 147]> ::=
  IF <search_condition [Page 280]> THEN <statement [Page 284]> [ELSE <statement>]

in_predicate

<in_predicate [Page 98]> ::=
  <expression [Page 258]> [NOT] IN <subquery [Page 285]>
| <expression> [NOT] IN <expression_list [Page 258]>
| <expression_list> [NOT] IN <subquery>
| <expression_list> [NOT] IN (<expression_list>,...)

index_name

<index_name [Page 45]> ::=
  <identifier [Page 262]>

index_pos_spec

<index_pos_spec [Page 199]> ::=
  INDEX <column_name [Page 250]> = <value_spec [Page 288]>
| INDEXNAME <index_name [Page 263]> VALUES (<value_spec>,...)

indicator_name

<indicator_name [Page 45]> ::=
  <parameter_name [Page 272]>

initial_select

<initial_select [Page 176]> ::=
  <query_spec [Page 274]>

insert_expression

<insert_expression [Page 164]> ::=
  <extended_expression [Page 259]>
| <subquery [Page 285]>

insert_statement

<insert_statement [Page 164]> ::=
  INSERT [INTO] <table_name [Page 286]> [(<column_name [Page 250]>,...)]
    VALUES (<insert_expression [Page 263]>,...) [<duplicates_clause [Page 258]>]
| INSERT [INTO] <table_name> [(<column_name>,...)]
    <query_expression [Page 274]> [<duplicates_clause>]
| INSERT [INTO] <table_name> SET <set_insert_clause [Page 282]>,...
[<duplicates_clause>]

integer

<integer [Page 39]> ::=
  [sign [Page 282]]<unsigned_integer [Page 287]>

join_predicate

<join_predicate [Page 99]> ::=
  <expression [Page 258]> [<outer_join_indicator [Page 272]>] <comp_op [Page 250]>
<expression> [<outer_join_indicator>]

join_spec

<join_spec> ::=
  ON <search_condition [Page 280]>
| USING (<column_name [Page 250]>,...)

joined_table

<joined_table [Page 187]> ::=
  <from_table_spec [Page 260]> CROSS JOIN <from_table_spec>
| <from_table_spec> [INNER] JOIN <from_table_spec> <join_spec [Page 264]>
| <from_table_spec> [<LEFT | RIGHT | FULL> [OUTER]] JOIN
    <from_table_spec> <join_spec>

key_definition

<key_definition [Page 126]> ::
  PRIMARY KEY (<column_name [Page 250]>,...)

key_or_not_null_spec

<key_or_not_null_spec [Page 119]> ::=
  [PRIMARY] KEY
| NOT NULL [WITH DEFAULT]

key_spec

<key_spec [Page 55]> ::=
  <column_name [Page 250]> = <value_spec [Page 288]>

key_word

<key_word [Page 39]> ::=
  <not_reserved_key_word [Page 269]>
| <reserved_keyword [Page 277]>

language_specific_character

<language_specific_character [Page 36]> ::=
  <every letter that occurs in a Northern, Central, or Southern European language
    and is not included in the <letter [Page 265]> list>
| <for UNICODE-enabled databases: every character that is not included in the ASCII code list from 0 to 127>

letter

<letter [Page 35]> ::=

```
  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

<like_expression [Page 101]> ::=
  <expression [Page 258]>
| '<pattern_element [Page 272]>...'

like_predicate

<like_predicate [Page 101]> ::=
  <expression [Page 258]> [NOT] LIKE <like_expression [Page 265]> [ESCAPE <expression>]

literal

<literal [Page 36]> ::=
  <string_literal [Page 285]>
| <numeric_literal [Page 269]>

local_variable

<local_variable [Page 146]> ::=
  <variable_name [Page 288]> <data_type [Page 254]>

local_variable_list

<local_variable_list [Page 146]> ::=
  <local_variable [Page 265]>
| <local_variable_list>;<local_variable>

local_variables

<local_variables [Page 146]> ::=
  VAR <local_variable_list [Page 265]>;

lock_option

<lock_option [Page 191]> ::=
  WITH LOCK [(NOWAIT)] [EXCLUSIVE | OPTIMISTIC] [ISOLATION LEVEL
<unsigned_integer [Page 287]>]

lock_spec

<lock_spec [Page 209]> ::=
  TABLE <table_name [Page 286]>,...
| <row_spec [Page 279]>...
| TABLE <table_name>,... <row_spec>...

lock_statement

<lock_statement [Page 209]> ::=
  LOCK [(WAIT) | (NOWAIT)] <lock_spec [Page 266]> IN SHARE MODE
| LOCK [(WAIT) | (NOWAIT)] <lock_spec> IN EXCLUSIVE MODE
| LOCK [(WAIT) | (NOWAIT)] <row_spec [Page 279]>... OPTIMISTIC
| LOCK [(WAIT) | (NOWAIT)] <lock_spec> IN SHARE MODE <lock_spec> IN
EXCLUSIVE MODE

mantissa

<mantissa [Page 38]> ::=
  <fixed_point_literal [Page 260]>

mapchar_set_name

<mapchar_set_name [Page 46]> ::=
  <identifier [Page 262]>

match_char

<match_char [Page 103]> ::=
  Every character except
    %
  | *
  | X'1F'
  | <underscore [Page 287]>

```
  | ?
  | X'1E'
  | (
```

match_class

<match_class [Page 103]> ::=
  <match_range [Page 267]>
| <match_element [Page 267]>

match_element

<match_element [Page 103]> ::=
  Every character except
)

match_range

<match_range [Page 103]> ::=
  <match_element [Page 267]>-<match_element>

Match set

<match_set [Page 103]> ::=
  <underscore [Page 287]>
| ?
| X'1E'
| <match_char [Page 266]>
| ([<
      ∧ (only for code type EBCDIC)
    | ~ (only for code type ASCII)
    | ¬
      >]<match_class [Page 267]>...)

match_string

<match_string [Page 103]> ::=
  %
| *
| X'1F'

minutes

<minutes [Page 77]> ::=
  <expression [Page 258]>

modify_definition

<modify_definition [Page 131]> ::=
  MODIFY (<column_name [Page 250]> [<data_type [Page 254]>] [<column_attributes [Page 249]>]...)

monitor_statement

<monitor_statement [Page 235]> ::=
  MONITOR INIT

named_query_expression

<named_query_expression [Page 181]> ::=
  <named_query_term [Page 268]>
| <named_query_expression> UNION [ALL] <query_term [Page 274]>
| <named_query_expression> EXCEPT [ALL] <query_term>

named_query_primary

<named_query_primary [Page 182]> ::=
  <named_query_spec [Page 268]>
| (<named_query_expression [Page 268]>)

named_query_spec

<named_query_spec [Page 184]> ::=
  SELECT [<distinct_spec [Page 256]>] <result_table_name [Page 278]> (<select_column [Page 280]>,...) <table_expression [Page 285]>

named_query_term

<named_query_term [Page 182]> ::=
  <named_query_primary [Page 268]>
| <named_query_term> INTERSECT [ALL] <query_primary [Page 274]>

named_select_statement

<named_select_statement [Page 177]> ::=
  <named_query_expression [Page 268]> [<order_clause [Page 272]>] [<update_clause [Page 287]>] [<lock_option [Page 266]>] [FOR REUSE]

new_index_name

<new_index_name [Page 143]> ::=
  <index_name [Page 263]>

new_table_name

<new_table_name [Page 133]> ::=
  <table_name [Page 286]>

next_stamp_statement

<next_stamp_statement [Page 173]> ::=
  NEXT STAMP [INTO] <parameter_name [Page 272]>

null_predicate

<null_predicate [Page 104]> ::=
  <expression [Page 258]> IS [NOT] NULL

numeric_literal

<numeric_literal [Page 37]> ::=
  <fixed_point_literal [Page 260]>
| <floating_point_literal [Page 260]>

not_reserved_key_word

<not_reserved_key_word [Page 40]> ::=

ACCOUNTING	ACTION	ACTIVATE	ADD	ADD_MONTHS
AFTER	ANALYZE	AND	ANSI	APPEND
AS	ASC	AT	AUDIT	AUTOSAVE
BACKUP_PAGES	BAD	BEFORE	BEGIN	BEGINLOAD
BEGINPROC	BETWEEN	BLOCKSIZE	BOTH	BREAK
BUFFER	BUFFERPOOL	BY		
CACHE	CACHELIMIT	CACHES	CALL	CANCEL
CASCADE	CAST	CATALOG	CATCH	CHECKPOINT
CLEAR	CLOSE	CLUSTER	COLD	COMMENT
COMMIT	COMPLETE	COMPUTE	CONFIG	CONNECT
CONSTRAINTS	CONTAINER	CONTINUE	COPY	COSTLIMIT
COSTWARNING	CREATE	CURRENT_DATE	CURRENT_TIME	CURRENT_TIME
CURRVAL	CURSOR	CYCLE		
DATA	DAYS	DB2	DBA	DBFUNCTION
DBPROC	DBPROCEDURE	DB_ABOVE_LIMIT	DB_BELOW_LIMIT	DECLARE
DEGREE	DESC	DESCRIBE	DESTPOS	DEVICE
DEVSPACE	DIAGNOSE	DISABLE	DIV	DO
DOMAIN	DOMAINDEF	DROP	DSETPASS	DUPLICATES
DW_COMPRESS	DW_DISPLACE	DYNAMIC		

EDITPROC	ELSE	ENABLE	END	ENDLOAD
ENDPOS	ERROR	ESCAPE	ESTIMATE	EUR
EVENT	EXCLUSIVE	EXECUTE	EXPLAIN	EXPLICIT
EXTRACT				
FALSE	FETCH	FILE	FIRSTPOS	FNULL
FORCE	FOREIGN	FORMAT	FREAD	FREEPAGE
FVERSION	FWRITE			
GET	GRANT	GRANTED		
HEXTORAW	HIGH	HOLD	HOURS	
IDENTIFIED	IF	IMPLICIT	IN	INCREMENT
INDEXNAME	INDICATOR	INIT	INITRANS	INOUT
INPROC	INSTANCE	INSTR	IS	ISO
ISOLATION				
JIS				
KEEP				
LABEL	LANGUAGE	LASTPOS	LAST_DAY	LEADING
LEVEL	LIKE	LOAD	LOCAL	LOCK
LOGFULL	LOG_ABOVE_LIMIT	LOW		
MAXTRANS	MAXVALUE	MEDIANAME	MEDIUM	MICROSECONDS
MINUS	MINUTES	MINVALUE	MODE	MODIFY
MONITOR	MONTHS	MONTHS_BETWEEN		
NEW_TIME	NEXTVAL	NEXT_DAY	NLSSORT	NLS_DATE_FOR
NLS_DATE_LANGUAGE	NLS_LANGUAGE	NLS_SORT	NOCACHE	NOCYCLE
NOLOG	NOMAXVALUE	NOMINVALUE	NONE	NOORDER
NOREWIND	NORMAL	NOSORT	NOWAIT	NUMBER
NVL				
OBID	OFF	ONLY	OPEN	OPTIMISTIC
OPTIMIZE	OPTION	OR	ORACLE	OUT
OUTER	OVERWRITE			
PACKAGE	PAGES	PARAM	PARSE	PARSEID
PARTICIPANTS	PASSWORD	PATTERN	PCTFREE	PCTUSED
PERCENT	PERMLIMIT	PIPE	POS	PRECISION
PRIV	PRIVILEGES	PROC	PROCEDURE	PSM
PUBLIC				
QUICK				
RANGE	RAW	RAWTOHEX	READ	RECURSIVE
REFERENCES	REFRESH	RELEASE	REMOTE	RENAME
RESOURCE	REST	RESTART	RESTORE	RESTRICT
REUSE	REVOKE	RFETCH	ROLE	ROLLBACK

ROW	ROWNUM	ROWS		
SAME	SAMPLE	SAVE	SAVEPOINT	SCHEMA
SEARCH	SECONDS	SEGMENT	SELECTIVITY	SEQUENCE
SERVERDB	SESSION	SHARE	SHUTDOWN	SNAPSHOT
SOUNDS	SOURCEPOS	SQLID	SQLMODE	STANDARD
START	STARTPOS	STAT	STATE	STOP
STORAGE	SUBPAGES	SUBTRANS	SWITCH	SYNONYM
SYSDATE				
TABID	TABLEDEF	TABLESPACE	TAPE	TEMP
TEMPLIMIT	TERMCHAR	THEN	TIMEOUT	TO_CHAR
TO_DATE	TO_NUMBER	TRACE	TRAILING	TRANSFILE
TRIGGER	TRIGGERDEF	TRUE	TRY	TYPE
UNIQUE	UNKNOWN	UNLOAD	UNLOCK	UNTIL
USA	USAGE	USERID		
VALIDPROC	VARCHAR2	VARYING	VERIFY	VERSION
VIEW	VSIZE	VTRACE		
WAIT	WHENEVER	WHILE	WORK	WRITE
YEARS				

object_spec

<object_spec [Page 143]> ::=
  COLUMN <table_name [Page 286]>.<column_name [Page 250]>
| DBPROC[EDURE] <dbproc_name [Page 254]>
| DOMAIN <domain_name [Page 256]>
| FOREIGN KEY <table_name>.<referential_constraint_name [Page 275]>
| INDEX <index_name [Page 263]> ON <table_name>
| INDEX <table_name>.<column_name>
| SEQUENCE <sequence_name [Page 281]>
| [PUBLIC] SYNONYM <synonym_name [Page 285]>
| TABLE <table_name>
| TRIGGER <trigger_name [Page 286]> ON <table_name>
| USER <user_name [Page 288]>
| USERGROUP <usergroup_name [Page 288]>
| <parameter_name [Page 272]>

old_index_name

<old_index_name [Page 143]> ::=
  <index_name [Page 263]>

old_table_name

<old_table_name [Page 133]> ::=
  <table_name [Page 286]>

open_cursor_statement

<open_cursor_statement [Page 192]> ::=
  OPEN <result_table_name [Page 278]>

order_clause

<order_clause [Page 190]> ::=
  ORDER BY <sort_spec [Page 283]>,...

outer_join_inidicator

<outer_join_inidicator [Page 99]> ::=
  (+)

owner

<owner [Page 44]> ::=
  <user_name [Page 288]>
| <usergroup_name [Page 288]>
| TEMP

parameter_name

<parameter_name [Page 46]> ::=
  :<identifier [Page 262]>
| :<identfier>(<identifier>)
| :<identfier>(.<identifier>.)

parameter_spec

<parameter_spec [Page 51]> ::=
  <parameter_name [Page 272]> [<indicator_name [Page 263]>]

password

<password [Page 46]> ::=
  <identifier [Page 262]>
| <first_password_character [Page 260]>[<identifier_tail_character [Page 262]>...]

pattern_element

<pattern_element [Page 102]> ::=
  <match_string [Page 267]>
| <match_set [Page 267]>

pos_spec

<pos_spec [Page 197]> ::=
  INDEX <column_name [Page 250]>

```
| INDEXNAME <index_name [Page 263]>
| <index_pos_spec [Page 263]> [KEY <key_spec [Page 264]>,...]
| KEY <key_spec>,...
```

position

```
<position [Page 193]> ::=
  POS (<unsigned_integer [Page 287]>)
| POS (<parameter_spec [Page 272]>)
| ABSOLUTE <integer [Page 264]>
| ABSOLUTE <parameter_spec>
| RELATIVE <integer>
| RELATIVE <parameter_spec>
```

predicate

```
<predicate [Page 92]> ::=
  <between_predicate [Page 248]>
| <bool_predicate [Extern]>
| <comparison_predicate [Page 251]>
| <default_predicate [Page 255]>
| <exists_predicate [Page 258]>
| <in_predicate [Page 263]>
| <join_predicate [Page 264]>
| <like_predicate [Page 265]>
| <null_predicate [Page 269]>
| <quantified_predicate [Page 274]>
| <rowno_predicate [Page 279]>
| <sound_predicate [Page 283]>
```

priv_spec

```
<priv_spec [Page 161]> ::=
  ALL [PRIV[ILEGES]] ON [TABLE] <table_name [Page 286]>,...
| <privilege [Page 273]>,... ON [TABLE] <table_name>,...
| <role_name [Page 278]>
```

privilege

```
<privilege [Page 47]> ::=
  INSERT
| UPDATE [(<column_name [Page 250]>,...)]
| SELECT [(<column_name>,...)]
| SELUPD [(<column_name>,...)]
| DELETE
| INDEX
| ALTER
| REFERENCES [(<column_name>,...)]
```

procedure_name

```
<procedure_name [Page 44]> ::=
  <identifier [Page 262]>
```

quantified_predicate

<quantified_predicate [Page 104]> ::=
  <expression [Page 258]> <comp_op [Page 250]> <quantifier [Page 274]> <expression_list [Page 258]>
| <expression> <comp_op> <quantifier> <subquery [Page 285]>
| <expression_list> <equal_or_not [Page 258]> <quantifier>
(<expression_list>,...)
| <expression_list> <equal_or_not> <quantifier> <subquery>

quantifier

<quantifier [Page 106]> ::=
  ALL
| SOME
| ANY

query_expression

<query_expression [Page 179]> ::=
  <query_term [Page 274]>
| <query_expression> UNION [ALL] <query_term>
| <query_expression> EXCEPT [ALL] <query_term>

query_primary

<query_primary [Page 180]> ::=
  <query_spec [Page 274]>
| (<query_expression [Page 274]>)

query_spec

<query_spec [Page 182]> ::=
  SELECT [<distinct_spec [Page 256]>] <select_column [Page 280]>,... <table_expression [Page 285]>

query_statement

<query_statement [Page 174]> ::=
  <declare_cursor_statement [Page 254]>
| <recursive_declare_cursor_statement [Page 275]>
| <named_select_statement [Page 268]>
| <select_statement [Page 281]>

query_term

<query_term [Page 180]> ::=
  <query_primary [Page 274]>
| <query_term> INTERSECT [ALL] <query_primary>

recursive_declare_cursor_statement

<recursive_declare_cursor_statement [Page 176]> ::=
  DECLARE <result_table_name [Page 278]> CURSOR FOR WITH RECURSIVE <reference_name [Page 275]>
    (<alias_name [Page 247]>,...) AS (<initial_select [Page 263]> UNION ALL
<recursive_select [Page 275]>) <final_select [Page 260]>

recursive_select

<recursive_select [Page 176]> ::=
  <query_spec [Page 274]>

reference_name

<reference_name [Page 48]> ::=
  <identifier [Page 262]>

referenced_column

<referenced_column [Page 123]> ::=
  <column_name [Page 250]>

referenced_table

<referenced_table [Page 123]> ::=
  <table_name [Page 286]>

referencing_column

<referencing_column [Page 123]> ::=
  <column_name [Page 250]>

referential_constraint_definition

<referential_constraint_definition [Page 123]> ::=
  FOREIGN KEY [<referential_constraint_name [Page 275]>] (<referencing_column [Page 275]>,...)
    REFERENCES <referenced_table [Page 275]> [(<referenced_column [Page 275]>,...)]
[<delete_rule [Page 255]>]

referential_constraint_name

<referential_constraint_name [Page 48]> ::=
  <identifier [Page 262]>

regular_token

<regular_token [Page 39]> ::=
  <literal [Page 265]>
| <key_word [Page 264]>
| <identifier [Page 262]>
| <parameter_name [Page 272]>

release_statement

<release_statement [Page 212]> ::=
  COMMIT [WORK] RELEASE
| ROLLBACK [WORK] RELEASE

rename_column_statement

<rename_column_statement [Page 133]> ::=
  RENAME COLUMN <table_name [Page 286]>.<column_name [Page 250]> TO <column_name>

rename_index_statement

<rename_index_statement [Page 143]> ::=
  RENAME INDEX <old_index_name [Page 271]> [ON <table_name [Page 286]>] TO
<new_index_name [Page 268]>

rename_synonym_statement

<rename_synonym_statement [Page 137]> ::=
  RENAME [PUBLIC] SYNONYM <old_synonym_name [Page 285]> TO <new_synonym_name>

rename_table_statement

<rename_table_statement [Page 133]> ::=
  RENAME TABLE <old_table_name [Page 271]> TO <new_table_name [Page 268]>

rename_user_statement

<rename_user_statement [Page 158]> ::=
  RENAME USER <user_name [Page 288]> TO <new_user_name [Page 288]>

rename_usergroup_statement

<rename_usergroup_statement [Page 158]> ::=
  RENAME USERGROUP <usergroup_name [Page 288]> TO <new_usergroup_name [Page 288]>

rename_view_statement

<rename_view_statement [Page 141]> ::=
  RENAME VIEW <old_table_name [Page 271]> TO <new_table_name [Page 268]>

reserved_key_word

<reserved_key_word [Page 40]> ::=

ABS	ABSOLUTE	ACOS	ADDDATE	ADDTIME
ALL	ALPHA	ALTER	ANY	ASCII
ASIN	ATAN	ATAN2	AVG	
BINARY	BIT	BOOLEAN	BYTE	
CEIL	CEILING	CHAR	CHARACTER	CHECK
CHR	COLUMN	CONCAT	CONNECTED	CONSTRAINT
COS	COSH	COT	COUNT	CROSS
CURDATE	CURRENT	CURTIME		
DATABASE	DATE	DATEDIFF	DAY	DAYNAME
DAYOFMONTH	DAYOFWEEK	DAYOFYEAR	DBYTE	DEC
DECIMAL	DECODE	DEFAULT	DEGREES	DELETE
DIGITS	DIRECT	DISTINCT	DOUBLE	
EBCDIC	EXCEPT	EXISTS	EXP	EXPAND
FIRST	FIXED	FLOAT	FLOOR	FOR
FROM	FULL			
GRAPHIC	GREATEST	GROUP		
HAVING	HEX	HOUR		
IFNULL	IGNORE	INDEX	INITCAP	INNER
INSERT	INT	INTEGER	INTERNAL	INTERSECT
INTO				
JOIN				
KEY				
LAST	LCASE	LEAST	LEFT	LENGTH
LFILL	LINK	LIST	LN	LOCALSYSDBA
LOCATE	LOG	LOG10	LONG	LONGFILE
LOWER	LPAD	LTRIM		
MAKEDATE	MAKETIME	MAPCHAR	MAX	MBCS
MICROSECOND	MIN	MINUTE	MOD	MONTH
MONTHNAME				
NATURAL	NCHAR	NEXT	NOROUND	NO
NOT	NOW	NULL	NUM	NUMERIC
OBJECT	OF	ON	ORDER	
PACKED	PI	POWER	PREV	PRIMARY
RADIANS	REAL	REFERENCED	REJECT	RELATIVE
REPLACE	RFILL	RIGHT	ROUND	ROWID
ROWNO	RPAD	RTRIM		

SECOND	SELECT	SELUPD	SERIAL	SET
SHOW	SIGN	SIN	SINH	SMALLINT
SOME	SOUNDEX	SPACE	SQRT	STAMP
STATISTICS	STDDEV	SUBDATE	SUBSTR	SUBSTRING
SUBTIME	SUM	SYSDBA		
TABLE	TAN	TANH	TIME	TIMEDIFF
TIMESTAMP	TIMEZONE	TO	TOIDENTIFIER	TRANSACTION
TRANSLATE	TRIM	TRUNC	TRUNCATE	
UCASE	UID	UNICODE	UNION	UPDATE
UPPER	USER	USERGROUP	USING	UTCDIFF
VALUE	VALUES	VARCHAR	VARGRAPHIC	VARIANCE
WEEK	WEEKOFYEAR	WHERE	WITH	
YEAR				
ZONED				

result_column_name

<result_column_name [Page 183]> ::=
  <identifier [Page 262]>

result_table_name

<result_table_name [Page 45]> ::=
  <identifier [Page 262]>

revoke_statement

<revoke_statement [Page 162]> ::=
  REVOKE <priv_spec [Page 273]>,... FROM <grantee [Page 261]>,... [<cascade_option [Page 249]>]
| REVOKE EXECUTE ON <dbproc_name [Page 254]> FROM <grantee>,...
| REVOKE SELECT ON <sequence_name [Page 281]> FROM <grantee>,...
[<cascade_option>]

role_name

<role_name [Page 48]> ::=
  <identifier [Page 262]>

rollback_statement

<rollback_statement [Page 208]> ::=
  ROLLBACK [WORK] [KEEP <lock_statement [Page 266]>]

routine

<routine [Page 146]> ::=
  [<local_variables [Page 266]>] <statement_list [Page 284]>;

routine_sql_statement

<routine_sql_statement [Page 147]> ::=
  <call_statement [Page 249]>
| <close_statement [Page 249]>
| <create_table_temp [Page 253]>
| <declare_cursor_statement [Page 254]>
| <delete_statement [Page 255]>
| <fetch_statement [Page 259]>
| <insert_statement [Page 263]>
| <lock_statement [Page 266]>
| <select_statement [Page 281]>
| <named_select_statement [Page 268]>
| <single_select_statement [Page 282]>
| <select_direct_statement:_searched [Page 280]>
| <select_direct_statement:_positioned [Page 280]>
| <select_ordered_statement:_searched [Page 281]>
| <select_ordered statement:_positioned [Page 281]>
| <subtrans_statement [Page 285]>
| <update_statement [Page 287]>

rowno_column

<rowno_column [Page 183]> ::=
  ROWNO [<result_column_name [Page 278]>]
| <result_column_name> = ROWNO

rowno_predicate

<rowno_predicate [Page 106]> ::=
  ROWNO < <unsigned_integer [Page 287]>
| ROWNO < <parameter_spec [Page 272]>
| ROWNO <= <unsigned_integer [Page 287]>
| ROWNO <= <parameter_spec [Page 272]>

sample_definition

<sample_definition [Page 113]> ::=
  SAMPLE <unsigned_integer [Page 287]> ROWS
| SAMPLE <unsigned_integer> PERCENT

search_and_result_spec

<search_and_result_spec [Page 80]> ::=
  <search_expression [Page 258]>, <result_expression>

search_condition

<search_condition [Page 107]> ::=
  <boolean_term [Page 249]>
| <search_condition> OR <boolean_term>

seconds

<seconds [Page 77]> ::=
  <expression [Page 258]>

select_column

<select_column [Page 183]> ::=
  <table_columns [Page 285]>
| <derived_column [Page 255]>
| <rowno_column [Page 279]>
| <stamp_column [Page 284]>

select_direct_statement:_positioned

<select_direct_statement:_positioned [Page 197]> ::=
  SELECT DIRECT <select_column [Page 280]>,... INTO <parameter_spec [Page 272]>,...
    FROM <table_name [Page 286]> WHERE CURRENT OF <result_table_name [Page 278]>
[<lock_option [Page 266]>]

select_direct_statement:_searched

<select_direct_statement:_searched [Page 196]> ::=
  SELECT DIRECT <select_column [Page 280]>,... INTO <parameter_spec [Page 272]>,...
    FROM <table_name [Page 286]> KEY <key_spec [Page 264]>,...[<where_clause [Page 289]>] [<lock_option [Page 266]>]

select_ordered_format1:_positioned

<select_ordered_format1:_positioned [Page 199]> ::=
  SELECT <FIRST | LAST> <select_column [Page 280]>,... INTO <parameter_spec [Page 272]>,...
    FROM <table_name [Page 286]> [INDEX <column_name [Page 250]> | INDEXNAME <index_name [Page 263]>]
    WHERE CURRENT OF <result_table_name [Page 278]> [<lock_option [Page 266]>]
| SELECT <FIRST | LAST> <select_column>,... INTO <parameter_spec>,...
    FROM <table_name> [index_pos_spec [Page 263]]
    WHERE CURRENT OF <result_table_name> [<lock_option>]

select_ordered_format1:_searched

<select_ordered_format1:_searched [Page 197]> ::=
  SELECT <FIRST | LAST> <select_column [Page 280]>,... INTO <parameter_spec [Page 272]>,...

```
    FROM <table_name [Page 286]>[<pos_spec [Page 272]>] [<where_clause [Page 289]>]
[<lock_option [Page 266]>]
```

select_ordered_format2:_positioned

```
<select_ordered_format2:_positioned [Page 199]>::=
  SELECT <NEXT | PREV> <select_column [Page 280]>,...INTO <parameter_spec [Page
272]>,...
    FROM <table_name [Page 286]> [<index_pos_spec [Page 263]>]
    WHERE CURRENT OF <result_table_name [Page 278]> [<lock_option [Page 266]>]
```

select_ordered_format2:_searched

```
<select_ordered_format2:_searched [Page 197]> ::=
  SELECT <NEXT | PREV> <select_column [Page 280]>,...INTO <parameter_spec [Page
272]>,...
    FROM <table_name [Page 286]>[<index_pos_spec [Page 263]>]
    KEY <key_spec [Page 264]>,... [<where_clause [Page 289]>] [<lock_option [Page 266]>]
```

select_ordered_statement:_positioned

```
<select_ordered_statement:_positioned [Page 199]> ::=
  <select_ordered_format1:_positioned [Page 280]>
| <select_ordered_format2:_positioned [Page 281]>
```

select_ordered_statement:_searched

```
<select_ordered_statement:_searched [Page 197]> ::=
  <select_ordered_format1:_searched [Page 280]>
| <select_ordered_format2:_searched [Page 281]>
```

select_statement

```
<select_statement [Page 178]> ::=
  <query_expression [Page 274]> [<order_clause [Page 272]>] [<update_clause [Page 287]>]
[<lock_option [Page 266]>]
    [FOR REUSE]
```

sequence_name

```
<sequence_name [Page 49]> ::=
  <identifier [Page 262]>
```

set_function_name

```
<set_function_name [Page 89]> ::=
  COUNT
| MAX
| MIN
| SUM
| AVG
```

```
| STDDEV
| VARIANCE
```

set_function_spec

<set_function_spec [Page 87]> ::=
  COUNT (*)
| <distinct_function [Page 256]>
| <all_function [Page 247]>

set_insert_clause

<set_insert_clause [Page 168]> ::=
  <column_name [Page 250]> = <extended_value_spec [Page 259]>

set_statement

<set_statement [Page 206]> ::=
  SET ROLE ALL [EXCEPT <role_name [Page 278]>]
| SET ROLE NONE
| SET ROLE <role_name> [IDENTIFIED BY <password [Page 272]>]
| SET ISOLATION LEVEL <unsigned_integer [Page 287]>

set_update_clause

<set_update_clause [Page 171]> ::=
  <column_name [Page 250]> = <extended_expression [Page 259]>
| <column_name>,... = (<extended_expression>,...)
| (<column_name>,...) = (<extended_expression>,...)
| <column_name> = <subquery [Page 285]>
| (<column_name>,...) = <subquery>

sign

<sign [Page 37]> ::=
  +
| -

simple_identifier

<simple_identifier [Page 40]> ::=
  <first_character [Page 260]>[<identifier_tail_character [Page 262]>...]

single_select_statement

<single_select_statement [Page 196]> ::=
  SELECT [<distinct_spec [Page 256]>] <select_column [Page 280]>,... INTO
<parameter_spec [Page 272]>,...
    FROM <from_table_spec [Page 260]>,...[<where_clause [Page 289]>] [<group_clause
[Page 262]>]
    [<having_clause [Page 262]>] [<lock_option [Page 266]>]

sort_spec

<sort_spec [Page 190]> ::=
  <unsigned_integer [Page 287]> [ASC | DESC]
| <expression [Page 258]> [ASC | DESC]

sound_predicate

<sound_predicate [Page 106]> ::=
  <expression [Page 258]> [NOT] SOUNDS [LIKE] <expression>

source_user

<source_user [Page 151]> ::=
  <user_name [Page 288]>

special_character

<special_character [Page 36]> ::=
  every character except
    <digit [Page 256]>
  | <letter [Page 265]>
  | <extended_letter [Page 259]>
  | <hex_digit [Page 262]>
  | <language_specific_character [Page 265]>
  | <character for the end of a line in a file>

special_function

<special_function [Page 80]> ::=
  VALUE (<expression [Page 258]>, <expression>,...)
| GREATEST (<expression>, <expression>,...)
| LEAST (<expression>, <expression>,...)
| DECODE (<check_expression [Page 258]>, <search_and_result_spec [Page 279]>,...[,
<default_expression [Page 258]>])

special_identifier

<special_identifier [Page 41]> ::=
  <special_identifier_character [Page 283]>...

special_identifier_character

<special_identifier_character [Page 41]> ::=
  any character.

sql_comment

```
<sql_comment [Page 110]> ::=
  /*<commend text>*/
```

stamp_column

```
<stamp_column [Page 183]> ::=
  STAMP [<result_column_name [Page 278]>]
| <result_column_name> = STAMP
```

statement

```
<statement [Page 147]> ::=
  BEGIN <statement_list [Page 284]> END
| BREAK
| CONTINUE
| CONTINUE EXECUTION
| <if_statement [Page 263]>
| <while_statement [Page 289]>
| <assignment_statement [Page 248]>
| STOP (<expression [Page 258]> [,<expression>] )
| TRY <statement_list>; CATCH <statement>
| <routine_sql_statement [Page 279]>
```

statement_list

```
<statement_list [Page 147]> ::=
  <statement [Page 284]>
| <statement_list> ; <statement>
```

string_function

```
<string_function [Page 62]> ::=
  <string_spec [Page 285]> || <string_spec>
| <string_spec> & <string_spec>
| SUBSTR ( <string_spec>, <expression [Page 258]>[, <expression>] )
| LFILL ( <string_spec>, <string_literal [Page 285]> [,<unsigned_integer [Page 287]> ] )
| RFILL ( <string_spec>, <string_literal> [,<unsigned_integer> ] )
| LPAD ( <string_spec>, <expression>, <string_literal> [,<unsigned_integer>
] )
| RPAD ( <string_spec>, <expression>, <string_literal> [,<unsigned_integer>
] )
| TRIM ( <string_spec>[, <string_spec> ] )
| LTRIM ( <string_spec>[, <string_spec> ] )
| RTRIM ( <string_spec>[, <string_spec> ] )
| EXPAND ( <string_spec>, <unsigned_integer> )
| UPPER ( <string_spec> )
| LOWER ( <string_spec> )
| INITCAP ( <string_spec> )
| REPLACE ( <string_spec>, <string_spec>[, <string_spec> ] )
| TRANSLATE ( <string_spec>, <string_spec>, <string_spec> )
| MAPCHAR ( <string_spec>[, <unsigned_integer> ] [, <mapchar_set_name [Page
266]> ] )
| ALPHA ( <string_spec>[, <unsigned_integer> ] )
| ASCII ( <string_spec> )
```

```
| EBCDIC ( <string_spec> )
| SOUNDEX ( <string_spec> )
```

string_literal

<string_literal [Page 36]> ::=
  ''
| '<character [Page 249]>...'
| <hex_literal [Page 262]>

string_spec

<string_spec [Page 55]> ::=
  <expression [Page 258]>

subquery

<subquery [Page 189]> ::=
  (<query_expression [Page 274]>)

subtrans_statement

<subtrans_statement [Page 208]> ::=
  SUBTRANS BEGIN
| SUBTRANS END
| SUBTRANS ROLLBACK

synonym_name

<synonym_name [Page 49]> ::=
  <identifier [Page 262]>

table_columns

<table_columns [Page 183]> ::=
  *
| <table_name [Page 286]>.*
| <reference_name [Page 275]>.*

table_description_element

<table_description_element [Page 111]> ::=
  <column_definition [Page 250]>
| <constraint_definition [Page 251]>
| <referential_constraint_definition [Page 275]>
| <key_definition [Page 264]>
| <unique_definition [Page 287]>

table_expression

<table_expression [Page 185]> ::=
  <from_clause [Page 260]> [<where_clause [Page 289]>] [<group_clause [Page 262]>]
[<having_clause [Page 262]>]

table_name

<table_name [Page 50]> ::=
  [<owner [Page 272]>.]<identifier [Page 262]>

term

<term [Page 90]> ::=
  <factor [Page 259]>
| <term> * <factor>
| <term> / <factor>
| <term> DIV <factor>
| <term> MOD <factor>

termchar_set_name

<termchar_set_name [Page 50]> ::=
  <identifier [Page 262]>

time_expression

<time_expression [Page 77]> ::=
  <expression [Page 258]>

time_or_timestamp_expression

<time_or_timestamp_expresion [Page 77]> ::=
  <expression [Page 258]>

time_function

<time_function [Page 76]> ::=
  ADDTIME (<time_or_timestamp_expression [Page 286]>, <time_expression [Page 286]>)
| SUBTIME (<time_or_timestamp_expression>, <time_expression>)
| TIMEDIFF (<time_or_timestamp_expression>, <time_or_timestamp_expression>
)
| MAKETIME (<hours [Page 262]>, <minutes [Page 267]>, <seconds [Page 280]>)

trigger_event

<trigger_event [Page 149]> ::
  INSERT
| UPDATE [(<column_list [Page 250]>)]
| DELETE

trigger_name

<trigger_name [Page 51]> ::=
  <identifier [Page 262]>

trigonometric_function

<trigonometric_function [Page 62]> ::=
  COS (<expression [Page 258]>)
| SIN (<expression>)
| TAN (<expression>)
| COT (<expression>)

```
| COSH ( <expression> )
| SINH ( <expression> )
| TANH ( <expression> )
| ACOS ( <expression> )
| ASIN ( <expression> )
| ATAN ( <expression> )
| ATAN2 ( <expression>, <expression> )
| RADIANS ( <expression> )
| DEGREES ( <expression> )
```

underscore

<underscore [Page 41]> ::=

```
  _
```

unique_definition

<unique_definition [Page 126]> ::=

```
  [CONSTRAINT <index_name [Page 263]>] UNIQUE (<column_name [Page 250]>,...)
```

unlock_statement

<unlock_statement [Page 211]> ::=

```
  UNLOCK <row_spec [Page 279]>... IN SHARE MODE
| UNLOCK <row_spec>... IN EXCLUSIVE MODE
| UNLOCK <row_spec>... IN SHARE MODE <row_spec>... IN EXCLUSIVE MODE
| UNLOCK <row_spec>... OPTIMISTIC
```

unsigned_integer

<unsigned_integer [Page 38]> ::=

```
  <numeric_literal [Page 269]>
```

update_clause

<update_clause [Page 191]> ::=

```
  FOR UPDATE [OF <column_name [Page 250]>,...]
```

update_statement

<update_statement [Page 169]> ::=

```
  UPDATE [OF] <table_name [Page 286]> [<reference_name [Page 275]>] SET
<set_update_clause [Page 282]>,...
    [KEY <key_spec [Page 264]>,...] [WHERE <search_condition [Page 280]>]
| UPDATE [OF] <table_name> [<reference_name>] (<column_name [Page 250]>,...)
    VALUES (<extended_value_spec [Page 259]>,...) [KEY <key_spec>,...]
    [WHERE <search_condition>]
| UPDATE [OF] <table_name> [<reference_name>] SET <set_update_clause>,...
    WHERE CURRENT OF <result_table_name [Page 278]>
| UPDATE [OF] <table_name> [<reference_name>] (<column_name>,...)
    VALUES (<extended_value_spec>,...) WHERE CURRENT OF <result_table_name>
```

update_statistics_statement

<update_statistics_statement [Page 226]> ::=

```
  UPDATE STAT[ISTICS] COLUMN <table_name [Page 286]>.<column_name [Page 250]>
```

```
    [ESTIMATE [<sample_definition [Page 279]>]]
| UPDATE STAT[ISTICS] COLUMN (<column_name>,...) FOR <table_name>
    [ESTIMATE [<sample_definition>]]
| UPDATE STAT[ISTICS] COLUMN (*) FOR <table_name>
    [ESTIMATE [<sample_definition>]]
| UPDATE STAT[ISTICS] <table_name> [ESTIMATE [<sample_definition>]]
| UPDATE STAT[ISTICS] [<owner [Page 272]>.][<identifier [Page 262]>]* [ESTIMATE
[<sample_definition>]]
```

user_mode

```
<user_mode [Page 152]> ::=
  DBA
| RESOURCE
| STANDARD
```

user_name

```
<user_name [Page 43]> ::=
  <identifier [Page 262]>
```

usergroup_mode

```
<usergroup_mode [Page 154]> ::=
  RESOURCE
| STANDARD
```

usergroup_name

```
<usergroup_name [Page 43]> ::=
  <identifier [Page 262]>
```

value_spec

```
<value_spec [Page 52]> ::=
  <literal [Page 265]>
| <parameter_spec [Page 272]>
| NULL
| USER
| USERGROUP
| LOCALSYSDBA
| SYSDBA [(<user_name [Page 288]>)]
| SYSDBA [(<usergroup_name [Page 288]>)]
| [<owner>.]<sequence_name [Page 281]>.NEXTVAL
| [<owner [Page 272]>.]<sequence_name>.CURRVAL
| DATE
| TIME
| TIMESTAMP
| TIMEZONE
| TRUE
| FALSE
| TRANSACTION
| UTCDIFF
```

variable_name

```
<variable_name [Page 147]> ::=
  <identifier [Page 262]>
```

where_clause

<where_clause [Page 188]> ::=
  WHERE <search_condition [Page 280]>

while_statement

<while_statement [Page 147]> ::=
  WHILE <search_condition [Page 280]> DO <statement [Page 284]>