Reference Manual 7.2 (BC)

HELP.BCDBADAREFERENZ
_73

Release 4.6C

SAP™

Copyright

Icons

Icon	Meaning
	Caution
	Example
	Note
	Recommendation
	Syntax
	Tip

Contents

Reference Manual 7.2 (BC)

Reference Manual 7.2 (BC)

This document outlines the syntax and semantics of the SQL statements, as used by the SAP DB database system (version 7.2).

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

This document 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 the 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 the 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 but must not be separated by commas or blanks.

Concepts

The following terms are explained here:

Data type

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

- NULL value (undefined value)

  Special value, whose comparison with all other values is always undefined.

- Special NULL value [Page 111]

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

- Non-NULL value

  Character string [Page 16], LONG column [Page 17], number [Page 18], date value [Page 19], time value [Page 20], timestamp value [Page 21], BOOLEAN [Page 22]

See also:

Using data types in SQL statements: data type [Page 251]

Character string

A character string is a data type [Page 15] that consists of a series of alphanumeric characters.

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

Character strings with the same code attribute	These character strings can be compared to each other.
Character strings with the code attributes ASCII, EBCDIC	These character strings have the EBCDIC and ASCII code attributes and can be compared with date [Page 19], time [Page 20], and timestamp values [Page 21].

See also:

Code tables [Page 44]

LONG column

A LONG column is a data type [Page 15] 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 16] or other data types.

Number

A number is a special data type [Page 15]. There are fixed point and floating point numbers:

- Fixed point numbers

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

- Floating point numbers

  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.

All numbers can be compared to one another.

Date value

The date value date type [Page 15] is a special character string [Page 16].

A date value can be compared to other date values and to character strings with the code attributes ASCII and EBCDIC.

See also:

Date and time format [Page 114]

Time value

The time value date type [Page 15] is a special character string [Page 16].

A time value can be compared to other time values and to character strings with the code attributes ASCII and EBCDIC.

See also:

Date and time format [Page 114]

Timestamp value

The timestamp value date type [Page 15] is a special character string [Page 16]. A timestamp consists of a date [Page 19] and time value [Page 20] and a microsecond specification.

A timestamp value can be compared to other timestamp values and to character strings with the code attributes ASCII and EBCDIC.

See also:

Date and time format [Page 114]

BOOLEAN

Boolean is a data type [Page 15] that can only assume one of the states TRUE or FALSE and the NULL value.

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

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 256] 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 [Page 251] FIXED(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 (such as different value and parameter lengths, NULL value, special NULL value, etc.) that may have occurred when the values were assigned. 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:

Data type [Page 15]

Parameter name [Page 98]

Indicator name [Page 96]

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 26] 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 93], 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 106]

Result table name [Page 94]

CREATE TABLE statement [Page 246]

CREATE VIEW statement [Page 291]

Column

All values in a table [Page 25] column have the same data type [Page 15]. 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 16] 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 18]) have the same mantissa length.
  All numbers in a **fixed point column** (fixed point number [Page 18]) have the same format; i.e. 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 104]

Using column definitions in SQL statements: column definition [Page 250]

Domain

Domain definitions enable ranges of values to be defined and designated for table columns [Page 26].

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

If a domain was defined without an owner [Page 93], 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 92]

CREATE DOMAIN statement [Page 283]

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 95] and table name [Page 106] must be unique.

See also:

CREATE INDEX statement [Page 302]

Synonym

A synonym is another name for a table [Page 25].

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 105]

CREATE SYNONYM statement [Page 288]

Users and Usergroups

User names and passwords are defined when the database system is installed.

- DBM user

  responsible for controlling, monitoring, and backing up the database system.

- Database administrator - SYSDBA

  installs the database system, i.e. creates users etc. The database administrator is the owner of the system tables and is granted the privilege of defining other administrators.

- DOMAIN user

  maintains the system tables. His or her name is always DOMAIN. Any password can be selected.

  Information on installing the system is provided in the documentation entitled *R/3 Database Manager*.

There are four hierarchical groups of users in WARM database mode:

- STANDARD users

  can only access existing tables for which they have received privileges. They can define synonyms and view tables for these tables. A STANDARD user can only create temporary tables.

- RESOURCE users

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

- Database administrators (DBA)

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

- Database administrator - SYSDBA

  installs the system. The system database administrator has all the rights of a DBA. In addition, he or she can create users with DBA status.

  Each database only has one SYSDBA.

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 name [Page 89]

Usergroup name [Page 88]

CREATE USER statement [Page 318]

CREATE USERGROUP statement [Page 321]

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 99]

GRANT statement [Page 337]

Role

A role is a collection of privileges [Page 32].

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 102]

CREATE ROLE statement [Page 335]

Serverdb

A serverdb, i.e. a database instance, consists of the catalog and the user data.

- The **catalog** comprises metadata which stores the definitions of database objects such as base tables [Page 25], view tables [Page 25], synonyms [Page 29], value ranges [Page 27], indexes [Page 28], users and usergroups [Page 30].
  The catalog consists of several parts. One part comprises information on the installed database system and the metadata with the user and usergroup definitions. This part is not assigned to a user or usergroup.
  The catalog contains a part for each user or usergroup with the metadata for the objects created by the user or usergroup in question. This includes metadata on base tables, view tables, etc.
  Users can only access the metadata of other users or usergroups if they have been granted the necessary privileges.
- The **user data** is all of the rows in all of the base tables.

Transaction

A transaction is a sequence of database operations that form a unit with regard to data backup and synchronization.

Transactions are closed with COMMIT or ROLLBACK:

- **COMMIT**: all modifications made to the serverdb within the transaction are kept.
- **ROLLBACK**: all of the modifications to the serverdb made by the transaction are reversed, even those that were terminated with the SUBTRANS END statement (subtransactions [Page 36]).

Modifications terminated with COMMIT cannot be reversed with ROLLBACK.

COMMIT and ROLLBACK implicitly open a new transaction.

Locks

The database system draws a distinction between SHARE locks and exclusive locks:

- **SHARE** locks prevent locked tables or table rows from being modified by other users while still allowing read access.
- **Exclusive** locks prevent other users from both reading and modifying the locked objects, while the user who specified the lock can modify these objects.

Tables and table rows are locked in a transaction with a lock mode that is determined when the user logs onto the database system (CONNECT statement [Page 412]) or can be set with the LOCK option [Page 392].

See also:

Transactions [Page 409]

COMMIT statement [Page 417]

ROLLBACK statement [Page 418]

Subtransaction

The purpose of closed, nested transactions (subtransactions) is to let a series of database operations within a transaction [Page 35] 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 409]

SUBTRANS statement [Page 419]

COMMIT statement [Page 417]

ROLLBACK statement [Page 418]

Session

When users are defined, they are assigned a password. To be able to work with a database instance, users must specify their user name and password.

If the user name and password are valid, the user is given access to the database instance. By doing so, the user opens a session and the first transaction [Page 35].

A user can only work with the database instance within a session. A session is always terminated explicitly by the user.

The user name required to obtain access to the database instance is referred to as the "current user" if the user is not a member of a usergroup. If the user is a member of a usergroup, the name of the usergroup is the "current user".

See also:

Users and usergroups [Page 30]

Password [Page 97]

CONNECT statement [Page 412]

SET statement [Page 416]

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 90]).
- **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 267]).
- **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 268] 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 256]).
- **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 100]).
- **Database procedures and triggers**: complex integrity rules that require access to further tables can be formulated with database procedures [Page 39] or triggers [Page 41].

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 41].

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 columns [Page 17]. 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 36] that can be reset without interfering with transaction control in the database application.

See also:

Name of a database procedure [Page 91]

Triggers

While database procedures [Page 39] 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 108]

Backup concept

In error situations that do not involve storage medium failures, the database system automatically restores the last consistent state of the database on restart. This means that all effects of committed transactions [Page 35] are preserved, while the effects of transactions open at the time of the error are cancelled.

If a failure occurs on a storage medium, a backup version of the serverdb [Page 34] must be loaded. This backup version can be a full and/or incremental backup. To be able to restore the current state of the database, the logs must be imported into the database system after the relevant backups have been loaded. By doing so, you can ensure that the last consistent state of the database is restored.

Further information is available in the R/3 documentation entitled:

R/3 Database Manager (DBMGUI)

R/3 Database Manager (DBMCLI)

BC – Computing Center Management System

SQL mode (SQLMODE)

The database system can execute correct database applications and applications that are written in accordance with one of the following definitions:

- INTERNAL (internal database definition)
- ANSI Standard (ANSI X3.135-1992, Entry SQL)
- Definition DB2 Version 4
- Definition ORACLE7

The database system is able to check whether new applications comply with one of the above definitions. This means, in particular, that any extension above and beyond the selected definition is considered incorrect. Support for DDL statements in other SQL modes, however, is limited.

When connecting to the database system, one of the above-mentioned definitions or the INTERNAL SQL mode can be selected.

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 412]

Code tables

The database system uses one of the following codes internally:

- ASCII code [Page 45] pursuant to ISO 8859/1
- EBCDIC code [Page 47] CCSID 500, codepage 500

This code can be specified as a code attribute in the column definition when the data type [Page 251] for the column is defined.

ASCII code

ASCII code pursuant to ISO 8859/1.2:

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

ASCII code

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

May be used by the operating system.

EBCDIC code

EBCDIC code CCSID 500, codepage 500:

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

EBCDIC code

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

Basic elements

Character

A `character` is an element of a character string [Page 16] or keyword [Page 72].

Syntax

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

digit [Page 51], letter [Page 52], extended letter [Page 53], hex digit [Page 54], language-specific character [Page 55], special character [Page 56]

Digit

A `digit` is a character [Page 50].

Syntax

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

Letter

A `letter` is a character [Page 50].

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 50].

Syntax

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

Hex digit

A `hex digit` is a character [Page 50].

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 50] is any letter that occurs in a northern, southern, or central European language and is not contained in the list of letters [Page 52].

German umlauts: ä, ö, ü

French letters with a “grave” accent.

Special character

A `special character` is any character [Page 50] that is not contained in the following list:

- digit [Page 51]
- letter [Page 52]
- extended letter [Page 53]
- hex digit [Page 54]
- language-specific character [Page 55]
- Characters that indicate the end of a line in a file

Literal

A `literal` is an unknown data object that is defined fully by virtue of its value (specifies a non-NULL value, see data type [Page 15]). Literal values cannot be modified. A distinction is made between string literals and numeric literals.

Syntax

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

string literal [Page 58], numeric literal [Page 61]

String literals
```
'69190 Walldorf'
'Anthony Smith'
```

Numeric literals
```
+0.58498
1E160
-765E-04
```

Explanation

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

A string literal [Page 58] of the type **`'<character>...'`** or `''` is only valid for a value referring to an alphanumeric column [Page 26] with the code attribute ASCII or EBCDIC. A hex literal [Page 59] is only valid for a value referring to a column with the code attribute BYTE (column definition [Page 250]).

A string literal (string literal [Page 58]) of the type `''`, **`x''`** and **`X''`**, and string literals that only contain blanks are **not** the same as the NULL value.

String literal

A string literal [Page 57] 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 50], hex literal [Page 59]

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

Hex literal

String literal [Page 58] that contains a value in hexadecimal notation.

Syntax

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

Hex digit seq [Page 60]

```
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 54]

Numeric literal

A numeric literal [Page 57] is a number [Page 18] represented as a fixed or floating point number.

Syntax

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

Fixed point literal [Page 62], floating point literal [Page 65]

Fixed point literal

Numeric literal [Page 61] that specifies a number [Page 18] as a fixed point number.

Syntax

```
<fixed point literal> ::=   [<sign>]<digit sequence>[.<digit sequence>]
| [sign]<digit sequence>. | [sign].<digit sequence>
```

sign [Page 63], digit sequence [Page 64]

```
+123.12
1234.
```

Sign

Sign

Syntax

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

Digit sequence

Sequence of digits

Syntax

```
<digit sequence> ::= <digit>...
```

digit [Page 51]

Floating point literal

Numeric literal [Page 61] that specifies a number [Page 18] as a floating point number.

Syntax

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

mantissa [Page 66], exponent [Page 67]

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

Mantissa

Mantissa

Syntax

```
<mantissa> ::= <fixed point literal>
```

Fixed point literal [Page 62]

Exponent

Exponent

Syntax

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

Sign [Page 63], digit [Page 51]

Unsigned integer

An `unsigned integer` is a special numeric literal [Page 61].

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

Integer

An `unsigned integer` is a special numeric literal [Page 61]. An unsigned integer can be represented in any way but must be a positive integer.

Syntax

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

Sign [Page 63], unsigned integer [Page 68]

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 71], delimiter token [Page 85]

```
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 83] within a special identifier [Page 84] are represented by two consecutive quotes.

Regular token

Normal character set [Page 70] (regular token)

Syntax

```
<regular token> ::= <literal> | <keyword> | <identifier> | <parameter
name>
```

Literal [Page 57], key word [Page 72], identifier [Page 78], parameter name [Page 98]

```
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 73], reserved keyword [Page 76]

Explanation

Keywords can be entered in uppercase/lowercase characters.

Reserved keywords must not be used in simple identifiers [Page 79]. Reserved keywords, however, can be specified in the form of special identifiers [Page 84].

Not reserved keyword

Keywords [Page 72] (`not reserved keywords`). If possible, these key words should not be used to designate objects.

ACCOUNTING	ACTION	ACTIVATE	ADD	ADD_MONTHS
AFTER	ANALYZE	AND	ANSI	APPEND
AS	ASC	AT	AUDIT	AUTOSAVE
AUTO_NEW_TAPE	AUTO_OFF	AUTO_ON	AUTO_STANDBY_OFF	AUTO_STANDBY_ON
BACKUP_PAGES	BAD	BAD_PAGE	BEFORE	BEGIN
BEGINLOAD	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	CONVERTER_SCAN	COPY
COSTLIMIT	COSTWARNING	CREATE	CURRENT_DATE	CURRENT_TIME
CURRENT_TIMESTAMP	CURRVAL	CURSOR	CYCLE	
DATA	DAYS	DB2	DBA	DBFUNCTION
DBPROC	DBPROCEDURE	DB_ABOVE_LIMIT	DB_BELOW_LIMIT	DC_LOG_HITRATE
DC_OMSLOG_HITRATE	DC_OMS_HITRATE	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	END_RESTART	END_SHUTDOWN	ERROR	ESCAPE

Not reserved keyword

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	LOW			
MAXTRANS	MAXVALUE	MEDIANAME	MEDIUM	MICROSECONDS
MINUS	MINUTES	MINVALUE	MODE	MODIFY
MONITOR	MONTHS	MONTHS_BETWEEN		
NEW_TIME	NEXTVAL	NEXT_DAY	NLSSORT	NLS_DATE_FORMAT
NLS_DATE_LANGUAGE	NLS_LANGUAGE	NLS_SORT	NO	NOCACHE
NOCYCLE	NOLOG	NOMAXVALUE	NOMINVALUE	NONE
NOORDER	NOREWIND	NORMAL	NOSORT	NOWAIT
NUMBER	NVL			
OBID	OFF	OMSLOG_MIN_FREE	OMSLOG_READ_CNT	OMSLOG_READ_MAX
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	START_RESTART	START_SHUTDOWN	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
UPD_STAT_WANTED	USA	USAGE	USERID	
VALIDPROC	VARCHAR2	VARYING	VERIFY	VERSION
VIEW	VSIZE	VTRACE		
WAIT	WARNING	WHENEVER	WHILE	WORK
WRITE				
YEARS				

Reserved keyword

Reserved keywords [Page 72] (`reserved keywords`). These key words must not be used to designate objects.

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	ININTCAP	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	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	
VALUE	VALUES	VARCHAR	VARGRAPHIC	VARIANCE
WEEK	WEEKOFYEAR	WHERE	WITH	
YEAR				
ZONED				

Identifier

Identifier A distinction is made between simple identifiers and special identifiers.

Syntax

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

Simple identifier [Page 79], double quotes [Page 83], special identifier [Page 84]

Explanation

Identifiers can be entered in uppercase/lowercase characters.

Reserved keywords [Page 76] 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 78]. The first character in a simple identifier must not be a digit or underscore [Page 82].

Syntax

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

First character [Page 80], identifier tail character [Page 81]

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 84] (in double quotes).

First character

First character in a simple identifier [Page 79].

Syntax

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

Letter [Page 52], extended letter [Page 53], language-specific character [Page 55]

Identifier tail character

Character allowed after the first character in a simple identifier [Page 79] or password [Page 86].

Syntax

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

Letter [Page 52], extended letter [Page 53], language-specific character [Page 55], digit [Page 51], underscore [Page 82]

Underscore

Underscore

Syntax

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

Double quotes

Quotation marks (double quotes)

Syntax

```
<double quotes> ::= "
```

Special identifier

Special identifier [Page 78]. 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 50], that can be linked in any sequence

Delimiter token

Delimiter token [Page 70]

Syntax

```
<delimiter token> ::=
   ( | ) | , | . | + | - | * | / | < | > | <> | != | = | <= | >=
| ¬= | ¬< | ¬> (for machines with EBCDIC code [Page 47])
| ~= | ~< | ~> (for machines with ASCII code [Page 45])
```

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 87]	alias name
Column name [Page 104]	column name
Constraint name [Page 90]	constraint name
Name of a database procedure [Page 91]	dbproc name
Domain name [Page 92]	domain name
Index name [Page 95]	index name
Indicator name [Page 96]	indicator name
Owner [Page 93]	owner
Parameter name [Page 98]	parameter name
Password [Page 97]	password
Reference name [Page 101]	reference name
Name of a referential constraint [Page 100]	referential constraint name
Result table name [Page 94]	result table name
Role name [Page 102]	role name
Sequence name [Page 103]	sequence name
Synonym name [Page 105]	synonym name
Table name [Page 106]	table name
Terminal character set name [Page 107]	termchar set name
Trigger name [Page 108]	trigger name
Usergroup name [Page 88]	usergroup name
User name [Page 89]	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 (`alias_name`) is a new column name [Page 104] 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 78]

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 291]

Recursive DECLARE CURSOR statement [Page 363]

Usergroup name

A usergroup name identifies a usergroup [Page 30].

Syntax

```
<usergroup name> ::= <identifier>
```

identifier [Page 78]

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 325]

User name

A user name defines a user [Page 30].

Syntax

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

identifier [Page 78]

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 318]

Constraint name

A constraint name defines a condition (data integrity [Page 38]) that must be satisfied by the rows in a table.

Syntax

```
<constraint name> ::= <identifier>
```

identifier [Page 78]

Explanation

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

SQL statement for defining a constraint name

CONSTRAINT definition [Page 258]

CREATE TABLE statement [Page 246]

ALTER TABLE statement [Page 271]

Name of a database procedure (dbproc name)

The name of a database procedure (`dbproc name`) designates a database procedure [Page 39].

Syntax

```
<dbproc name> ::= [<owner>.]<procedure name>
<procedure name> ::= <identifier>
```

owner [Page 93], identifier [Page 78]

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 308]

CALL statement [Page 357]

DROP DBPROC statement [Page 313]

Domain name

A domain name identifies a domain [Page 27] in a table column [Page 26].

Syntax

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

owner [Page 93], identifier [Page 78]

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 283]

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 89], usergroup name [Page 88]

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 106], 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 25]).

Syntax

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

identifier [Page 78]

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 359]

Index name

An index name identifies an index [Page 28].

Syntax

```
<index name> ::= <identifier>
```

identifier [Page 78]

Explanation

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

SQL statement for creating an index

CREATE INDEX statement [Page 302]

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 98]

Explanation

The indicator variable of a parameter [Page 24] provides information on any irregularities (e.g. NULL value, difference value and parameter lengths).

Password

Users require a password to connect to the database server (start a session [Page 37]).

Syntax

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

identifier [Page 78], identifier tail character [Page 81]

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

letter [Page 52], extended letter [Page 53], language specific letter [Page 55], digit [Page 51]

Explanation

Passwords are truncated after 18 characters.

SQL statement for defining a user password

CREATE USER statement [Page 318]

SQL statement for changing a user password

ALTER PASSWORD statement [Page 334]

Parameter name

A parameter name identifies a parameter [Page 24] (host variable) in an application containing SQL statements from the database system.

Syntax

```
<parameter name> ::= :<identifier> | :<identifier>(<identifier>) |
:<identifier>(.<identifier>.)
```

identifier [Page 78]

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

A privilege type identifies a certain privilege [Page 32].

Syntax

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

column name [Page 104]

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 250] or referential constraint definition [Page 260].

SQL statement for granting or revoking privileges

GRANT statement [Page 337]

REVOKE statement [Page 340]

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 38]). This integrity condition specifies deletion and existence dependencies between two tables.

Syntax

```
<referential constraint name> ::= <identifier>
```

identifier [Page 78]

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 260]

CREATE TABLE statement [Page 246]

ALTER TABLE statement [Page 271]

Reference name

The reference name is an identifier that is declared for a certain validity range and associated with exactly one table [Page 25]. The scope of this declaration is the entire SQL statement.

Syntax

```
<reference name> ::= <identifier>
```

identifier [Page 78]

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 33].

Syntax

```
<role name> ::= <identifier>
```

identifier [Page 78]

Explanation

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

SQL statement for defining a role

CREATE ROLE statement [Page 335]

SQL statement for granting a role

GRANT statement [Page 337]

SQL statement for activating a role

ALTER USER statement [Page 326]

ALTER USERGROUP statement [Page 328]

SET statement [Page 416]

SQL statement for dropping a role

DROP ROLE statement [Page 336]

Sequence name

A sequence name identifies a sequence of values.

Syntax

```
<sequence name> ::= <identifier>
```

identifier [Page 78]

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 285]

Column name

A column name identifies a column [Page 26].

Syntax

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

identifier [Page 78]

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 246]

CREATE VIEW statement [Page 291]

ALTER TABLE statement [Page 271]

QUERY statement [Page 359]

Synonym name

A synonym name identifies a synonym [Page 29] for a table name.

Syntax

```
<synonym name> ::= <identifier>
```

identifier [Page 78]

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 288]

Table name

A table name identifies a table [Page 25].

Syntax

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

owner [Page 93], identifier [Page 78]

Explanation

The database system uses certain table names for internal purposes. The identifiers [Page 78] 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 93] 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 246]

CREATE VIEW statement [Page 291]

CREATE SYNONYM statement [Page 288]

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 78]

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 412]

See also:

R/3 Database Manager (DBMGUI)

R/3 Database Administration (CONTROL)

Trigger name

A trigger name identifies a trigger [Page 41] that is defined for a table.

Syntax

```
<trigger name> ::= <identifier>
```

identifier [Page 78]

Explanation

SQL statements for creating and dropping a trigger

CREATE TRIGGER statement [Page 314]

DROP TRIGGER statement [Page 316]

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 104], table name [Page 106], reference name [Page 101], result table name [Page 94]

Explanation

For all names consisting of several identifiers [Page 78] 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 98], indicator name [Page 96]

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 111].
- **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.

Special NULL value

A special NULL value is a special data type [Page 15] 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 390]. If an overflow occurs in an arithmetic operation or a division by 0 at another point, the SQL statement is abnormally terminated.

As far as sorting is concerned, the special NULL value is greater than all non-NULL values, but less than 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 113]

Explanation

- DEFAULT keyword
  DEFAULT identifies the default value for the column in a CREATE TABLE statement [Page 246] or ALTER TABLE statement [Page 271]. 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 342]
  UPDATE statement [Page 349]
  The DEFAULT keyword can be used in a DEFAULT predicate [Page 222].

- STAMP keyword
  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 keyword 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 256]):
  INSERT statement [Page 342]
  UPDATE statement [Page 349]
  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 356]

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
```

`literal`	Literal [Page 57]
`parameter spec`	Parameter spec [Page 110]
`NULL`	NULL value [Page 15]
`USER`	Current user name [Page 89]
`USERGROUP`	Name of the usergroup [Page 88] 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 serverdb [Page 34]
`SYSDBA [(<user name>)]` `| SYSDBA [(<usergroup name>)]`	SYSDBA of the serverdb
`[<owner>.]<sequence name>.NEXTVAL`	Next value generated for the specified sequence name [Page 103] (of the owner [Page 93] in question).
`[<owner>.]<sequence name>.CURRVAL`	Value generated last for the specified sequence name using `[<owner>.]<sequence name>.NEXTVAL`.
`DATE`	Current date [Page 19]
`TIME`	Current time [Page 20]
`TIMESTAMP`	Current timestamp [Page 21]
`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 22]
`TRANSACTION`	Identification of the current transaction [Page 35]. This is a value of data type CHAR(10) BYTE.

Date and time format (datetimeformat)

The date and time format (`datetimeformat`) specifies the way in which date values [Page 19], time values [Page 20], and timestamp values [Page 21] 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/ISO	'HH.MM.SS'	'14.30.08'
INTERNAL	'HHHHMMSS'	'00143008'
JIS	'HH:MM:SS'	'14:30:08'
USA	'HH:MM AM (PM)'	'2:30 PM'

In all time formats, the identifier of the hour must consist of at least one digit. In the 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/ISO/JIS/USA	'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.

Specifying a string (string spec)

Only expressions [Page 209] 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 containing the specified key values does not have to exist.

Syntax

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

column name [Page 104], value spec [Page 113]

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 119], trigonometric function [Page 139], string function [Page 141], date function [Page 163], time function [Page 170], extraction function [Page 177], special function [Page 184], conversion function [Page 188]

Explanation

The arguments and results of the functions are numeric, alphanumeric or Boolean values that are subject to certain restrictions. LONG columns [Page 17] are not allowed as arguments.

Arithmetic function

An arithmetic function (`arithmetic_function`) is a function 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 209], unsigned integer [Page 68], string spec [Page 116]

TRUNC(a,n) [Page 138], ROUND(a,n) [Page 135], NOROUND(a) [Page 132], FIXED(a,p,s) [Page 123], FLOAT(a,s) [Page 124], CEIL(a) [Page 121], FLOOR(a) [Page 125], SIGN(a) [Page 136], ABS(a) [Page 120], POWER(a,n) [Page 134], EXP(a) [Page 122], SQRT(a) [Page 137], LN(a) [Page 130], LOG(a,b) [Page 131], PI [Page 133], LENGTH(a) [Page 128], INDEX(a,b,p,s) [Page 126]

ABS(a)

ABS(a) is an arithmetic function [Page 119] that determines the unsigned value (absolute value) of the number a.

	Result of the ABS(a) function
a is NULL value	NULL value [Page 15]
a is special NULL value	Special NULL value [Page 15]

CEIL(a)

CEIL(a) is an arithmetic function [Page 119] 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 15]
a is special NULL value	Special NULL value [Page 15]

EXP(a)

EXP(a) is an arithmetic function [Page 119] 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 15]
a is special NULL value	Special NULL value [Page 15]

FIXED(a,p,s)

FIXED(a,p,s) is an arithmetic function [Page 119] that is used to output the number a in a format of the data type FIXED(p,s) (see data type [Page 251]).

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 15]
a is the NULL value	NULL value (see data type [Page 15])
a is special NULL value	Special NULL value

FLOAT(a,s)

FLOAT(a,s) is an arithmetic function [Page 119] that outputs the number a in a format of the data type FLOAT(s) (see data type [Page 251]). It is rounded to s places if necessary.

	Result of the FLOAT(a,s) function
a is NULL value	NULL value (see data type [Page 15])
a is special NULL value	Special NULL value [Page 15]

FLOOR(a)

FLOOR(a) is an arithmetic function [Page 119] 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 15]
a is special NULL value	Special NULL value [Page 15]

INDEX(a,b,p,s)

INDEX(a,b,p,s) is an arithmetic function [Page 119] 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 15]
p or s is special NULL value [Page 15]	Error message

Model table: customer [Page 194]

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

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

SURNAME	POSITION_AR
Barth	2
GIAG	0
Starke	3
Steger	0
Braun	0
Schwarz	5
Maler	0
Wenzel	0
Muschel	0
Rietz	0
Schulze	0
Tisch	0
Meyer	0
DATA_KG	0

Braun	0

LENGTH(a)

LENGTH(a) is an arithmetic function [Page 119] 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 (see data type [Page 15])
a is special NULL value	Special NULL value [Page 15]

Model table: customer [Page 194]

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
```

SURNAME	LENGTH
GIAG	4
Barth	5
Braun	5
Braun	5
Maler	5
Meyer	5
Rietz	5
Tisch	5
Starke	6
Steger	6
Wenzel	6
DATA_KG	7
Muschel	7
Schulze	7
Schwarz	7

LN(a)

LN(a) is an arithmetic function [Page 119] that calculates the natural logarithm of the number a.

	Result of the LN(a) function
a is NULL value	NULL value [Page 15]
a is special NULL value	Special NULL value [Page 15]

LOG(a,b)

LOG(a,b) is an arithmetic function [Page 119] 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 15]
b is special NULL value	Special NULL value [Page 15]

NOROUND(a)

NOROUND(a) is an arithmetic function [Page 119] 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 15]
a is special NULL value	Special NULL value [Page 15]

PI

PI is an arithmetic function [Page 119] that outputs the value π.

POWER(a,n)

POWER(a,n) is an arithmetic function [Page 119] 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 15]
a is special NULL value	Special NULL value [Page 15]

ROUND(a,n)

ROUND(a,n) is an arithmetic function [Page 119] 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 138](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 15]
a is special NULL value	Special NULL value [Page 15]

SIGN(a)

SIGN(a) is an arithmetic function [Page 119] 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 15]
a is special NULL value	Special NULL value [Page 15]

SQRT(a)

SQRT(a) is an arithmetic function [Page 119] 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 15]
a is special NULL value	Special NULL value [Page 15]

TRUNC(a,n)

TRUNC(a,n) is an arithmetic function [Page 119] 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 18]
a is fixed point number	Fixed point number [Page 18]
a is the NULL value	NULL value [Page 15]
a is special NULL value	Special NULL value [Page 15]

Trigonometric function

A trigonometric function is a function 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 209]

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 15]
`<expression>` is special NULL value	Special NULL value [Page 15]
`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 $-\pi<=a<=+\pi$ and $-\pi<=b<=+\pi$
`ASIN(a)`	Arc sine of number a

Trigonometric function

`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 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 116], expression [Page 209], string literal [Page 58], unsigned integer [Page 68], mapchar set name [Page 150]

Concatenation (x||y and x&y) [Page 161], SUBSTR(x,a,b) [Page 156], LFILL(x,a,n) [Page 146], RFILL(x,a,n) [Page 152], LPAD(x,a,y,n) [Page 147], RPAD(x,a,y,n) [Page 153], TRIM(x,y) [Page 159], LTRIM(x,y) [Page 148], RTRIM(x,y) [Page 154], EXPAND(x,n) [Page 144], UPPER(x)/LOWER(x) [Page 160], INITCAP(x) [Page 145], REPLACE(x,y,z) [Page 151], TRANSLATE(x,y,z) [Page 158], MAPCHAR(x,n,i) [Page 149], ALPHA(x,n) [Page 142], ASCII(x)/EBCDIC(x) [Page 143], SOUNDEX(x) [Page 155]

ALPHA(x,n)

ALPHA(x,n) is a string function [Page 141] that enables a character x in ASCII or EBCDIC code (code tables [Page 44]) to be converted to a different one or two-character representation in the DEFAULTMAP (mapchar_set_name [Page 150]). ALPHA(x,n) is used to define the sort sequence.

The function ALPHA(x,n) uses the MAPCHAR(x,n,i) [Page 149] function internally (where i is the DEFAULTMAP) and also performs a conversion to uppercase letters (UPPER(x) [Page 160]).

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, for example, if the German umlaut "ü" is to be treated as "UE" for 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 141] that converts a character string [Page 16] x in ASCII code [Page 45] to EBCDIC code [Page 47] 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 (see data type [Page 15])

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 141] that inserts as many blanks (code attribute ASCII, EBCDIC (code tables [Page 44])) or binary zeros (code attribute BYTE) to the end of a character string [Page 16] 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 (see data type [Page 15])

See also:

LFILL(x,a,n) [Page 146]

RFILL(x,a,n) [Page 152]

INITCAP(x)

INITCAP(x) is a string function [Page 141] that changes a character string [Page 16] 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 52] nor digits [Page 51].

	Result of the INITCAP(x) function
x is NULL value	NULL value (see data type [Page 15])

Model table: customer [Page 194]

Standardizing the notation for names

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

NAME	NEW_NAME
DATASOFT	Datasoft
TOOLware	Toolware

LFILL(x,a,n)

LFILL(x,a,n) is a string function [Page 141] that inserts the character string a at the start of the character string [Page 16] 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 (see data type [Page 15])
a or n is the NULL value	Error message

Model table: customer [Page 194]

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

FIRSTNAME	NAME	CITY
Martin	Randolph	Los Angeles
Sally	Smith	Los Angeles
Joseph	Peters	Los Angeles
Susan	Baker	Los Angeles
Anthony	Jenkins	Los Angeles
Thomas	Adams	Los Angeles

RFILL(x,a,n) [Page 152]

EXPAND(x,n) [Page 144]

LPAD(x,a,y,n)

LPAD(x,a,y,n) is a string function [Page 141] that inserts the character string y at the start of the character string [Page 16] 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 128](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 (see data type [Page 15])
a is the special NULL value [Page 111]	Error message

Model table: customer [Page 194]

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 153]

LTRIM(x,y)

LTRIM(x,y) is a string function [Page 141] that removes all of the characters specified in the character string y from the start of the character string [Page 16] 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 44])) or binary zeros (code attribute BYTE) are removed from x.
x is NULL value	NULL value (see data type [Page 15])

See also:

TRIM(x,y) [Page 159]

RTRIM(x,y) [Page 154]

MAPCHAR(x,n,i)

MAPCHAR(x,n,i) is a string function [Page 141] 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 45] and EBCDIC code [Page 47] 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 150]) to convert the character string x.

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 (see data type [Page 15])

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
```

MAPCHAR SET name

MAPCHAR SETs were introduced to enable letters to be converted form one format to another. MAPCHAR SETs allow individual, country-specific letters to be represented by one or two internationally used letters (e.g. "ü" as "ue").

When the database instance is configured, these country-specific letters can be converted and stored under the name DEFAULTMAP. The conversion rules in the DEFAULTMAP can be changed if necessary.

Further MAPCHAR SETs can also be defined using the Database Manager.

If the MAPCHAR [Page 149] function is to be used, the name of the MAPCHAR SET (`mapchar_set_name`) must be specified. Otherwise, the MAPCHAR SET with the name DEFAULTMAP is used.

REPLACE(x,y,z)

REPLACE(x,y,z) is a string function [Page 141] that replaces the character string y in the character string [Page 16] 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 (see data type [Page 15])
y is NULL value	x remains unchanged
z is NULL value	The character string y in x is deleted

Model table: hotel [Page 195]

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 158]

RFILL(x,a,n)

RFILL(x,a,n) is a string function [Page 141] that inserts the character string a at the end of the character string [Page 16] 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 (see data type [Page 15])
a or n is the NULL value	Error message

See also:

LFILL(x,a,n) [Page 146]

EXPAND(x,n) [Page 144]

RPAD(x,a,y,n)

RPAD(x,a,y,n) is a string function [Page 141] that inserts the character string y at the end of the character string [Page 16] 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 128](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 (see data type [Page 15])
a is the special NULL value [Page 111]	Error message

See also:

LPAD(x,a,y,n) [Page 147]

RTRIM(x,y)

RTRIM(x,y) is a string function that removes the blanks (code attribute ASCII, EBCDIC (code tables [Page 44])) or binary zeros (code attribute BYTE) from the end of the character string [Page 16] 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 (see data type [Page 15])

See also:

TRIM(x,y) [Page 159]

LTRIM(x,y) [Page 148]

SOUNDEX(x)

SOUNDEX(x) is a string function [Page 141] that converts a character string [Page 16] 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 (see data type [Page 15])

The SOUNDS predicate [Page 239] 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 141] that outputs part of x (character string [Page 16] 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 ath 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 ath character to the last (n^{th}) character.
b is an unsigned integer [Page 68]	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 (see data type [Page 15])

Model table: customer [Page 194]

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	Dallas
P. Brown	Hollywood
M. Jackson	Washington
G. Howe	New York
F. Miller	Chicago
R. Brown	Hollywood
S. Murphy	Denver
B. Smith	San Francisco

A. Jones	Hollywood
F. O'Brien	Washington
P. Johnson	Chicago
E. Thomas	Denver

TRANSLATE(x,y,z)

TRANSLATE(x,y,z) is a string function [Page 141] that replaces the ith character of the character string [Page 16] y with the ith 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 (see data type [Page 15])
y is NULL value	x remains unchanged

Model table: customer [Page 194]

Each occurrence of the ith letter in the first character string is replaced by the ith 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
Randolph	Randolph
Smith	Smith
Peters	Pitirs
Baker	Bokir
Jenkins	Jinkins
Adams	Adoms

See also:

REPLACE(x,y,z) [Page 151]

TRIM(x,y)

TRIM(x,y) is a string function [Page 141] that removes all of the characters specified in the character string y from the start of the character string [Page 16] 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 44])) 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 (see data type [Page 15])

Model table: customer [Page 194]

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

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

See also:

LTRIM(x,y) [Page 148]

RTRIM(x,y) [Page 154]

UPPER/LOWER(x)

UPPER(x) and LOWER(x) are string functions [Page 141] that convert a character string [Page 16] to uppercase/lowercase letters.

	Result of the UPPER(x) or LOWER(x) function
x is NULL value	NULL value (see data type [Page 15])

Model table: hotel [Page 195]

Refining searches by specifying uppercase/lowercase letters

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

HNO	NAME	ZIP	CITY	ADDRESS
30	Flora	48153	New Jersey	158 Gardiner Street

Concatenation

A concatenation x||y or x&y is a string function [Page 141] that supplies the following results for x (character string [Page 16] 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 ASCII, EBCDIC (code tables [Page 44])) or binary zeros (code attribute BYTE).
x or y is the NULL value	NULL value (see data type [Page 15])

Columns with the same code attribute can be concatenated.

Columns with different code attributes (ASCII and EBCDIC) can be concatenated together and with date values [Page 19], time values [Page 20], and timestamp values [Page 21].

Model table: customer [Page 194]

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

NAME	ADDRESS
Porter	New York, NY 10580
DATASOFT	Dallas, TX 75243
Randolph	Los Angeles, CA 90018
Smith	Los Angeles, CA 90011
Brown	Hollywood, CA 90029
Jackson	Washington, DC 20037
Howe	New York, NY 10019
Miller	Chicago, IL 60601
Peters	Los Angeles, CA 90013
Baker	Los Angeles, CA 90008
Jenkins	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 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 169], expression [Page 209]

ADDDATE(t,a)/SUBDATE(t,a) [Page 164], DATEDIFF(t,s) [Page 165], DAYOFWEEK(t), WEEKOFYEAR(t), DAYOFMONTH(t), DAYOFYEAR(t) [Page 167], MAKEDATE(a,b) [Page 168], DAYNAME(t), MONTHNAME(t) [Page 166]

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 114] (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 163] that calculate a date in the future or past.

t: date or timestamp expression [Page 169]

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 19] or time stamp value [Page 21]
t or a is NULL value	NULL value (see data type [Page 15])
a is special NULL value [Page 15]	Error message

Model table: reservation [Page 198]

Increasing a reservation date by two days

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

ARRIVAL	ARRIVAL2	RNO
02/01/1999	02/03/1999	130

DATEDIFF(t,s)

DATEDIFF(t,s) is a date function [Page 163] that calculates the number of days between a start and end date.

t and s: date or timestamp expression [Page 169]

	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 19] in the timestamp value [Page 21] are used to calculate DATEDIFF(t,s).
t or s is NULL value	NULL value (see data type [Page 15])

Model table: reservation [Page 198]

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

ARRIVAL	DEPARTURE	DIFFERENCE	RNO
02/01/1999	02/03/1999	2	130

DAYNAME/MONTHNAME(t)

DAYNAME(t) and MONTHNAME(t) are date functions [Page 163] that supply the weekday or month of the specified day.

Value t: date or timestamp expression [Page 169]

	Result of the function
DAYNAME(t) or MONTHNAME(t)	Character string [Page 16] that supplies the name of a weekday (Sunday to Saturday) or month (January to December).
t is NULL value	NULL value (see data type [Page 15])

DAYOFWEEK/WEEKOFYEAR/DAYOFMONTH/DAYOFYEAR(t)

DAYOFWEEK(t)/WEEKOFYEAR(t)/DAYOFMONTH(t)/DAYOFYEAR(t) are date functions [Page 163] that calculate the following:

t: date or timestamp expression [Page 169]

	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 (see data type [Page 15])

MAKEDATE(a,b)

MAKEDATE(a,b) is a date function [Page 163] that supplies a date value [Page 19] from a year and day value.

The values a and b (expression [Page 209]) 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 (see data type [Page 15])
a or b is the special NULL value [Page 15]	Error message

`MAKEDATE(1999,49)` in the INTERNAL date format [Page 114] outputs '19990218'

Date or timestamp expression

When used in a function, this argument must supply a date value [Page 19], timestamp value [Page 21], or an alphanumeric value that matches the current date or timestamp format [Page 114].

Time function

A time function is a function 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 176], time expression [Page 175], hours, minutes, seconds [Page 174]

ADDTIME/SUBTIME(t,a) [Page 171], TIMEDIFF(t,s) [Page 173], MAKETIME(h,m,s) [Page 172]

A variety of date and time formats [Page 114] (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 170] that calculate a time/timestamp value in the past or future.

t: time or timestamp expression [Page 176]
a: time expression [Page 175]

	Result of the ADDTIME(t,a)/SUBTIME(t,a) function
Addition of a to t/ subtraction of a from t	Time value [Page 20] or time stamp value [Page 21]
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 (see data type [Page 15])

MAKETIME(h,m,s)

MAKETIME(h,m,s) is a time function [Page 170] that calculates a time value [Page 20] from the total number of hours, minutes, and seconds [Page 174].

	Result of the MAKETIME(h,m,s) function
h,m,s is NULL value	NULL value (see data type [Page 15])
h,m,s is special NULL value [Page 15]	Error message

TIMEDIFF(t,s)

TIMEDIFF(t,s) is a time function [Page 170] that calculates the time value between a start and end time.

t and s: time or timestamp expression [Page 176]
Both arguments must have the same data type, i.e. they must be either a time value [Page 20] or a timestamp value [Page 21].

	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 19] 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 (see data type [Page 15])

Hours/minutes/seconds

When used in a function, each of these arguments must be an integer greater than or equal to 0. Decimal places are truncated.

Time expression

When used in a function, this argument must supply a time value [Page 20] or an alphanumeric value that is in the current time format [Page 114].

Time or timestamp expression

When used in a function, this argument must supply a time value [Page 20], timestamp value [Page 21], or an alphanumeric value that matches the current time or timestamp format [Page 114].

Extraction function

An extraction function is a function that extracts part of a date value [Page 19], time value [Page 20], or timestamp value [Page 21] or calculates a date, time, or timestamp.

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 169], time or timestamp expression [Page 176], expression [Page 209]

YEAR(t), MONTH(t), DAY(t) [Page 183], HOUR(t), MINUTE(t), SECOND(t) [Page 179], MICROSECOND(a) [Page 180], TIMESTAMP(a,b) [Page 182], DATE(a) [Page 178], TIME(a) [Page 181]

A variety of date and time formats [Page 114] (ISO, USA, EUR, JIS, INTERNAL) are available for processing date and time values.

DATE(a)

DATE(a) is a function (extraction [Page 177]) that calculates a date value [Page 19].

	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 21] 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 18]	Date value that is equal to the xth day after December 31, 0000 (x=TRUNC(a) [Page 138])
a is NULL value	NULL value (see data type [Page 15])
a is special NULL value [Page 15]	Error message

HOUR/MINUTE/SECOND(t)

HOUR(t), MINUTE(t), and SECOND(t) are functions (extraction [Page 177]) that extract the hours, minutes, or seconds from the specified time or timestamp value.

t: time or timestamp expression [Page 176]

	Result of the function
HOUR(t), MINUTE(t), or SECOND(t)	Numeric value (hours, minutes, and seconds)
t is NULL value	NULL value (see data type [Page 15])

MICROSECOND(a)

MICROSECOND(a) is a function (extraction [Page 177]) that extracts the microseconds from a.

The value a must be a timestamp value [Page 21] 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 (see data type [Page 15])

TIME(a)

TIME(a) is a function (extraction [Page 177]) that calculates a time value [Page 20].

	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 21] 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 (see data type [Page 15])

TIMESTAMP(a,b)

TIMESTAMP(a,b) is a function (extraction [Page 177]) that calculates a timestamp value [Page 21] comprising a date value [Page 19], time value [Page 20], 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 (see data type [Page 15])

YEAR/MONTH/DAY(t)

YEAR(t), MONTH(t), and DAY(t) are functions (extraction [Page 177]) that extract the year, month, and day from the specified date or timestamp value.

t: date or timestamp expression [Page 169]

	Result of the function
YEAR(t), MONTH(t), or DAY(t)	Numeric value (year, month, day)
t is NULL value	NULL value (see data type [Page 15])

Special function

There are certain special functions 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 209], check expression, search and result spec, default expression [Page 185]

VALUE(x,y,...) [Page 187], GREATEST(x,y,...), LEAST(x,y,...) [Page 186], DECODE(x,y,...,z) [Page 185]

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

DECODE(x,y(i),...,z) is a special function [Page 184] that decodes expressions [Page 209] in accordance with their values.

x	check expression	Expression for which a comparison is to be carried out with the values in 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 196]

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 184] 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 (see data type [Page 15])
At least one argument is the special NULL value [Page 15]	Special NULL value

VALUE(x,y,...)

VALUE(x,y,...) is a special function [Page 184] that can be used to replace NULL values (see data type [Page 15]) 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 15]	Special NULL value
Each argument is a NULL value	NULL value

Model table: customer [Page 194]

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 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 209], unsigned integer [Page 68], datetimeformat [Page 114]

NUM(a) [Page 192], CHR(a,n) [Page 190], HEX(a) [Page 191], CHAR(a,t) [Page 189]

CHAR(a,t)

CHAR(a,t) is a function (conversion function [Page 188]) that converts the date values [Page 19], time values [Page 20], or timestamp values [Page 21] to a character string with the format for date values, time values, or timestamp values [Page 114] 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 (see data type [Page 15])

CHR(a,n)

CHR(a,n) is a function (conversion function [Page 188]) that converts numbers into character strings.

The CHR function can only be applied to numeric values, character strings [Page 16], and Boolean values [Page 22].

	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 (see data type [Page 15])
a is the special NULL value [Page 15]	Error message

HEX(a)

HEX(a) is a function (conversion function [Page 188]) 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 (see data type [Page 15])
a is the special NULL value [Page 15]	Error message

NUM(a)

NUM(a) is a function (conversion function [Page 188]) that converts the argument a to a numeric value.

NUM can be applied to character strings [Page 16] with the code attributes ASCII or EBCDIC (see code tables [Page 44]), date values [Page 19], time values [Page 20], timestamp values [Page 21], numeric and Boolean values [Page 22].

	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 dos 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 `-9.999999999999999999999999999999999999E+62`, `9.999999999999999999999999999999999999E+62`	Special NULL value [Page 15]
a is the NULL value	NULL value (see data type [Page 15])
a is the special NULL value	Special NULL value

Model Tables

Customer

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

CNO	TITLE	NAME	FIRSTNAME	ZIP	CITY	ACCOUNT
3000	Mrs	Porter	Jenny	10580	New York	100.00
3100	Comp	DATASOFT	?	75243	Dallas	4813.50
3200	Mr	Randolph	Martin	90018	Los Angeles	0.00
3300	Mrs	Smith	Sally	90011	Los Angeles	0.00
3400	Mr	Brown	Peter	90029	Hollywood	0.00
3500	Mr	Jackson	Michael	20037	Washington	0.00
3600	Mr	Howe	George	10019	New York	-315.40
3700	Mr	Miller	Frank	60601	Chicago	0.00
3800	Mr	Peters	Joseph	90013	Los Angeles	650.00
3900	Mrs	Baker	Susan	90008	Los Angeles	-4167.79
4000	Mr	Jenkins	Anthony	90005	Los Angeles	0.00
4100	Mr	Adams	Thomas	90014	Los Angeles	-416.88
4200	Mr	Griffith	Mark	10575	New York	0.00
4300	Comp	TOOLware	?	90002	Los Angeles	3770.50
4400	Mrs	Brown	Rose	90025	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'),
 name 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	48226	Detroit	155 Beechwood Str.
20	Long Island	45211	Cincinnati	1499 Grove Street
30	Regency	97213	Portland	477 17th Avenue
40	Eight Avenue	60601	Chicago	112 8th Avenue
50	Lake Michigan	60615	Chicago	354 OAK Terrace 43
60	Airport	70112	New Orleans	650 C Parkway
70	Empire State	10019	New York	65 Yellowstone Dr.
80	Midtown	60607	Chicago	12 Barnard Street
90	Long Beach	90804	Long Beach	200 Yellowstone Dr.
100	Dallas	75225	Dallas	1980 34th Str.
110	Atlantic	10570	New York	111 78th Street
120	Sunshine	90018	Los Angeles	35 Broadway 77
130	Star	90030	Hollywood	13 Beechwood Place 2
140	River Boat	20019	Washington	788 MAIN STREET 14
150	Indian Horse	95054	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 195] 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	11/13/1998	11/15/1998
110	3000	100	double	12/24/1998	01/06/1999
120	3200	50	suite	11/14/1998	11/18/1998
130	3900	110	single	02/01/1999	02/03/1999
140	4300	80	double	04/12/1998	04/30/1998
150	3600	70	double	03/14/1999	03/24/1999
160	4100	70	single	04/12/1998	04/05/1998
170	4400	150	suite	09/01/1998	09/03/1998
180	3100	120	double	12/23/1998	01/08/1999
190	4300	140	double	11/14/1998	11/17/1998

A logical link between the customer [Page 194], hotel [Page 195], and room [Page 196] 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 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 204], distinct function [Page 200], all function [Page 201]

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 25] or a group (the result table can be grouped using a GROUP condition).

NULL values (see data type [Page 15]), with the exception of the COUNT(*) function, are not included in the calculation.

No locks are set for certain set functions, irrespective of the isolation level (see CONNECT-statement [Page 412]) specified when the user connected to the database.

Model table: customer [Page 194]

```
SELECT SUM(account) sum_account, MIN(account) min_account,
FIXED (AVG(account),7,2) avg_account,
MAX(account) max_account, 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 199] that removes duplicated values and all NULL values (see data type [Page 15]).

Syntax

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

set function name [Page 202], expression [Page 209]

Explanation

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

1. A result table [Page 25] 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 values [Page 15] are not removed, two special NULL values are regarded as 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 194]

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 199] that removes the NULL values (see data type [Page 15]).

Syntax

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

set function name [Page 202], expression [Page 209]

Explanation

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

1. A result table [Page 25] 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 values [Page 15] are not removed, two special NULL values are regarded as 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, and 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 200] and an ALL function [Page 201].

Syntax

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

COUNT [Page 204], MAX, MIN [Page 205], SUM [Page 207], AVG [Page 203], STDDEV [Page 206], VARIANCE [Page 208]

AVG

AVG is a set function [Page 199].

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(38).

Model table: customer [Page 194]

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 199].

`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 200]).

`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 201])

The result has the data type FIXED(10).

Model table: customer [Page 194]

How many customers are there?

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

NUMBER
15

MAX/MIN

MAX/MIN is a set function [Page 199].

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 199].

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(38).

SUM

SUM is a set function [Page 199].

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(38).

VARIANCE

VARIANCE is a set function [Page 199].

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(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 212]

Explanation

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

	Result of an expression
`expression`	Value with any data type [Page 15]
`factor` supplies a NULL value	NULL value (see data type [Page 15])
`factor` supplies a special NULL value	Special NULL value [Page 15]
`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 priority:

1. The sign [Page 63] has a higher priority than the additive and multiplicative operators.
2. The multiplicative operators have a higher priority than the additive operators.
3. Multiplicative operators have the same priority.
4. Additive operators have the same priority.
5. Operators with the same precedence are evaluated from left to right.

Expression

Operands are fixed point numbers

Operand1 (a)	Operand2 (b)	Result
Fixed point number [Page 18] (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 120](a)<1E38 and ABS(b)<1E38 and b<>0	DIV	TRUNC [Page 138](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 18], the result of the arithmetic operation is a floating point number.

Factor

Specifies how the values are determined that are to be linked in an expression [Page 209] by means of arithmetic operators.

Syntax

```
<factor> ::= [<sign>] <value spec> | [<sign>] <column spec>
| [<sign>] <function spec> | [<sign>] <set function spec> |
<expression>
```

sign [Page 63], value spec [Page 113], column spec [Page 109], function spec [Page 118], set function spec [Page 199], expression [Page 209]

Predicate

A predicate is specified in a WHERE condition [Page 385] 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 94]) or to a group of rows in a table that was formed by the GROUP clause [Page 386].

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 215], bool predicate [Page 217], comparison predicate [Page 218], default predicate [Page 222], exists predicate [Page 223], in predicate [Page 224], join predicate [Page 226], like predicate [Page 228], null predicate [Page 234], quantified predicate [Page 235], rowno predicate [Page 238], sounds predicate [Page 239]

Explanation

- Columns in a table with the same code attribute are comparable.
- Columns with different code attributes ASCII and EBCDIC (see code tables [Page 44]) can be compared.
- Columns with the code attributes ASCII and EBCDIC can be compared with date values [Page 19], time values [Page 20], or timestamp values [Page 21].
- LONG columns [Page 17] can only be used in the NULL predicate.

Model table: customer [Page 194]

Selection without a condition:

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

CITY	NAME	FIRSTNAME
New York	Porter	Jenny
Dallas	DATASOFT	?
Los Angeles	Randolph	Martin
Los Angeles	Smith	Sally
Hollywood	Brown	Peter
Washington	Jackson	Michael
New York	Howe	George
Chicago	Miller	Frank
Los Angeles	Peters	Joseph

Predicate

Los Angeles	Baker	Susan
Los Angeles	Jenkins	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	Randolph	Martin
Los Angeles	Smith	Sally
Los Angeles	Peters	Joseph
Los Angeles	Baker	Susan
Los Angeles	Jenkins	Anthony
Los Angeles	Adams	Thomas
Los Angeles	TOOLware	?

BETWEEN predicate

The BETWEEN predicate [Page 213] checks whether a value is within a specified range.

Syntax

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

expression [Page 209]

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 values (see data type [Page 15])	x [NOT] BETWEEN y AND z is undefined

Model table: customer [Page 194]

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	Randolph	Los Angeles	0.00
Mrs	Smith	Los Angeles	0.00
Mr	Brown	Hollywood	0.00
Mr	Jackson	Washington	0.00
Mr	Howe	New York	-315.40
Mr	Miller	Chicago	0.00
Mr	Jenkins	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
```

BETWEEN predicate

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	Baker	Los Angeles	-4167.79
Mr	Adams	Los Angeles	-416.88
Comp	TOOLware	Los Angeles	3770.50
Mrs	Brown	Hollywood	440.00

Boolean predicate

Boolean values (BOOLEAN [Page 22]) are compared in a Boolean predicate [Page 213].

Syntax

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

column spec [Page 109]

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 213] 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 209], expression list [Page 209], subquery [Page 388]

The following operators are available for comparing two values:
<, >, <>, !=, =, <=, >= (comp op [Page 220])

Value lists can only be compared with the = and <> operators (equal or not [Page 221]).

Explanation

The subquery must supply a result table (see result table name [Page 94]) 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 226] 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 ASCII, EBCDIC, see code tables [Page 44]) or binary zeros (code attribute BYTE) at the end of one or both of the character strings are removed.
  If the character strings have different code attributes (ASCII and EBCDIC), one of the two strings is converted implicitly so that they both have the same code attribute.
  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, or BYTE).
- If x or y are NULL values (see data type [Page 15]), 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 194]

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 47])
| ~= | ~< | ~> (for machines with ASCII code [Page 45])
```

Comparison operators (equal or not)

Operators for comparing value lists (`equal or not`)

Syntax

```
<equal or not> ::= <> |=
| ¬=  (for machines with EBCDIC code [Page 47])
| ~=  (for machines with ASCII code [Page 45])
```

DEFAULT predicate (default predicate)

By specifying a DEFAULT predicate [Page 213], 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 109], comp op [Page 220]

Explanation

A DEFAULT specification [Page 256] must be made for the specified column. This can be done in the following SQL statements:

- CREATE TABLE statement [Page 246]
- ALTER TABLE statement [Page 271]

If the column contains the NULL value (see data type [Page 15]), `<column spec> <comp op> DEFAULT` is undefined.

The rules for comparing values or value lists, as defined under comparison predicate [Page 218], apply here.

EXISTS predicate

The EXISTS predicate [Page 213] (`exists predicate`) checks whether a result table (see result table name [Page 94]) contains at least one row.

Syntax

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

subquery [Page 388]

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 194], reservation [Page 198]

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 213] 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 209], expression list [Page 209], subquery [Page 388]

Explanation

The subquery must supply a result table (see result table name [Page 94]) 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 194]

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	Randolph	Los Angeles
Mrs	Sally	Smith	Los Angeles
Mr	Peter	Brown	Hollywood
Mr	Michael	Jackson	Washington
Mr	George	Howe	New York
Mr	Frank	Miller	Chicago
Mr	Joseph	Peters	Los Angeles
Mrs	Susan	Baker	Los Angeles
Mr	Anthony	Jenkins	Los Angeles
Mr	Thomas	Adams	Los Angeles
Mr	Mark	Griffith	New York
Mrs	Rose	Brown	Hollywood

JOIN predicate

A JOIN predicate [Page 213] 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 209], comp op [Page 220]

Explanation

Each `expression` must contain a column specification [Page 109]. 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 218] apply here.

If at least one OUTER JOIN indicator is specified in a JOIN predicate of a SEARCH condition [Page 240], 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 380].
- 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 374] and if one of the rules above cannot be observed, a QUERY expression [Page 368] 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 379] 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 type of comparison predicate [Page 218]. The number of JOIN predicates in a SEARCH condition is limited to 128.

JOIN predicate

Model tables customer [Page 194], reservation [Page 198]

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	11/13/1998	11/15/1998
110	Porter	12/24/1998	01/06/1999

Specifying an OUTER JOIN indicator

Model tables hotel [Page 195], reservation [Page 198]

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 213] is used to search for character strings [Page 16] 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 209], pattern element [Page 231]

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 values (see data type [Page 15])	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 194]

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	New York
Peters	Los Angeles

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

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

FIRSTNAME	CITY
Martin	Los Angeles
Michael	Washington
Mark	New York

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	Hollywood
Jackson	Washington

Pattern element

Element for specifying a comparison pattern (for a LIKE predicate [Page 228]). 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 232], match set [Page 233]

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 228]) is used to search for character strings [Page 16] that have a certain pattern. `Match strings` can be used to specify the pattern (pattern element [Page 231]).

Model table: customer [Page 194]

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

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

NAME	CITY
Martin	Washington
Michael	Washington
Porter	New York
Jackson	Washington

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 50] 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 82]

Explanation

A LIKE predicate [Page 228] is used to search for character strings [Page 16] that have a certain pattern. `Match sets` can be used to specify the pattern (pattern element [Page 231]).

- `<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 classes`) can be negated by prefixing it with ∧ or ~ or ¬. You cannot negate each character class individually.

Model table: customer [Page 194]

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	New York
Peters	Los Angeles

NULL predicate

By specifying a NULL predicate [Page 213], you can test whether the value is NULL value (see data type [Page 15]).

Syntax

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

expression [Page 209]

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 15]	false

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

Quantified predicate

By specifying a quantity predicate [Page 213], 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 388], expression list [Page 209]

The following operators are available for comparing values:
<, >, <>, !=, =, <=, >= (comp op [Page 220])

Value lists can only be compared with the = and <> operators (equal or not [Page 221]).

The quantified predicate can be qualified with ALL, SOME, or ANY (quantifier [Page 237]).

Explanation

The subquery must supply a result table (see result table name [Page 94]) 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 195]

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 235]) can be qualified with ALL, SOME, or ANY.

Syntax

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

ROWNO predicate

The ROWNO predicate [Page 213] restricts the number of lines in a result table (see result table name [Page 94]).

Syntax

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

unsigned integer [Page 68], parameter spec [Page 110]

Explanation

A ROWNO predicate may only be used in a WHERE clause [Page 385] that belongs to a QUERY statement. The ROWNO predicate can be used like any other predicate [Page 213] 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 390] 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 199] are specified, the set function is only applied to the number of lines restricted by the ROWNO predicate.

SOUNDS predicate

A SOUNDS predicate [Page 213] is used to perform a phonetic comparison.

Syntax

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

expression [Page 209]

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 44]).

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 a NULL value (see data type [Page 15])	`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 155] string function

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 242]: determine the boolean values to be linked or their negation (`NOT`).

Explanation

Predicates in a WHERE clause [Page 385] are applied to the specified row or a group of rows in a table formed with the GROUP clause [Page 386]. 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 194]

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 240] by AND or OR.

Syntax

```
<boolean factor> ::= [NOT] <predicate> | [NOT] (<search condition>)
```

predicate [Page 213], search condition [Page 240]

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.

SQL statements for data definition [Page 245]

CREATE TABLE statement	DROP TABLE statement	ALTER TABLE statement RENAME TABLE 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 317]

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 341]

SQL statement: overview

INSERT statement	UPDATE statement	DELETE statement
NEXT STAMP statement	CALL statement	

SQL statements for data query [Page 358]

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 409] management

CONNECT statement	SET statement	
COMMIT statement	ROLLBACK statement	SUBTRANS statement
LOCK statement	UNLOCK	RELEASE statement

SQL statements for statistics [Page 457] management

UPDATE STATISTICS statement	MONITOR statement	

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 246]	DROP TABLE statement [Page 269]	ALTER TABLE statement [Page 271] RENAME TABLE statement [Page 280] EXISTS TABLE statement [Page 282]
CREATE DOMAIN statement [Page 283]	DROP DOMAIN statement [Page 284]	
CREATE SEQUENCE statement [Page 285]	DROP SEQUENCE statement [Page 287]	
CREATE SYNONYM statement [Page 288]	DROP SYNONYM statement [Page 289]	RENAME SYNONYM statement [Page 290]
CREATE VIEW statement [Page 291]	DROP VIEW statement [Page 300]	RENAME VIEW statement [Page 301]
CREATE INDEX statement [Page 302]	DROP INDEX statement [Page 303]	ALTER INDEX statement [Page 304] RENAME INDEX statement [Page 305]
COMMENT ON statement [Page 306]		
CREATE TRIGGER statement [Page 314]	DROP TRIGGER statement [Page 316]	
CREATE DBPROC statement [Page 308]	DROP DBPROC statement [Page 313]	

CREATE TABLE statement

A CREATE TABLE statement creates a base table (see Table [Page 25]).

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 106], sample definition [Page 249], query expression [Page 368], duplicates clause [Page 346], constraint definition, [Page 258]

column definition [Page 250], constraint definition [Page 258], referential constraint definition [Page 260], key definition [Page 267], unique definition [Page 268]

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 93] 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 created 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 104] 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 251] of the ith column in the base table is identical to that of the ith column in the result table generated by the QUERY expression.
  The result table must not contain LONG columns [Page 17].
  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 346] 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 342].

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, i.e., 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 25] must be a base table, view table, or a synonym [Page 29]. 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 specifications [Page 256] or CONSTRAINT definitions [Page 258] 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 271]

Renaming columns

RENAME COLUMN statement [Page 281]

Renaming tables

RENAME TABLE statement [Page 280]

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 68]

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 458].

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 246]

ALTER TABLE statement [Page 271]

UPDATE STATISTICS statement [Page 458]

Column definition

A column definition defines a column in a table. The name and data type of each column are defined by the column name and data type [Page 15]. 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 104], data type [Page 251], column attributes [Page 254], domain name [Page 92]

Explanation

If the [PRIMARY] KEY column attribute is specified, the CREATE TABLE statement [Page 246] must not contain a key definition [Page 267].

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 258], 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 cannot be inserted in these columns.

- **Mandatory columns**: NOT NULL columns for which a DEFAULT specification [Page 256] 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 28] 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 253]

The memory requirements of all columns in a table must not exceed 8088 bytes.

Data type

A column definition [Page 250] can be specified with the column name, data types [Page 15], and a code attribute (ASCII, BYTE, EBCDIC).

Syntax

```
<data type> ::=
  CHAR[ACTER] [(<unsigned integer>)] [ASCII | BYTE | EBCDIC]
| VARCHAR [(<unsigned integer>)] [ASCII | BYTE | EBCDIC]
| LONG [VARCHAR] [ASCII | BYTE | EBCDIC]
| BOOLEAN
| FIXED (<unsigned integer> [,<unsigned integer>])
| FLOAT (<unsigned integer>)
| INT[EGER] | SMALLINT
| DATE | TIME | TIMESTAMP
```

unsigned integer [Page 68]

Explanation

Data Type	
CHAR[ACTER] [(n)]	Alphanumeric column with the length attribute n, 0<n<=8000 If no length attribute is specified, it is assumed that n=1. The database system determines, in accordance with n, whether the values in the column are stored with a fixed or variable length.
VARCHAR [(n)]	Alphanumeric column with the length attribute n, 0<n<=8000 If no length attribute is specified, it is assumed that n=1. Use VARCHAR (n) if the values in the column are to be stored with a variable length, irrespective of n.
LONG	Alphanumeric column with any length (not for temporary tables) LONG columns can only contain NOT NULL or a DEFAULT specification [Page 256] as a column attribute [Page 254]. LONG columns can be used in the following SQL statements: INSERT statement [Page 342], UPDATE statement [Page 349], NULL predicate [Page 234], and in selected columns [Page 376].
BOOLEAN	Column that can only have a NULL value (see data type [Page 15]) and the values TRUE and FALSE
FIXED(p,s)	Fixed point column with precision p and s decimal places (0<p<=38, s<=p). If no s is specified, it is assumed that the decimal places are 0.
INT[EGER]	Corresponds to FIXED(10,0) with permissible values between -2147483648 and 2147483647
SMALLINT	Corresponds to FIXED(5,0) with permissible values between -32768 and 32767

Explanation

FLOAT(p)	Floating point column with precision p (0<p<=38)
DATE	Alphanumeric column in which date values [Page 19] are stored.
TIME	Alphanumeric column in which time values [Page 20] are stored.
TIMESTAMP	Alphanumeric column in which timestamp values [Page 21] are stored.

Code Attribute	Column Values
No code attribute	Code attribute defined when the database system was installed
ASCII	ASCII code [Page 45] pursuant to ISO 8859/1
BYTE	Code neutral, i.e. the column values are not converted by the database system
EBCDIC	EBCDIC code [Page 47] according to CCSID 500, codepage 500

In addition to these data types, 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:

Memory Requirements of a Column Value as a Function of the Data Type [Page 253]

Memory Requirements of a Column Value per Data Types

Data type [Page 251]

Column definition [Page 250]

Data Type	Memory Requirements of a Column Value in Bytes for This Data Type
CHAR(n); VARCHAR(n); key columns	n+1
CHAR(n); no key column; n<=30	n+1
CHAR(n); no key column; 30<n<=254	n+2
CHAR(n); no key column; 254<n	n+3
VARCHAR(n); no key column; n<=254	n+2
VARCHAR(n); no key column; 254<n	n+3
LONG	9
FIXED(p,s)	(p+1) DIV 2 + 2
FLOAT(p)	(p+1) DIV 2 + 2
BOOLEAN	2
DATE	9
TIME	9
TIMESTAMP	21

Column attributes

A column definition [Page 250] 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 256], constraint definition [Page 258], delete rule [Page 263]

- A CONSTRAINT definition [Page 258] defines a condition that must be fulfilled by all the column values in the columns defined by the column definition [Page 250].
- `[REFERENCES <referenced table> [(<referenced column>)] [<delete rule>]`
  has the same effect as specifying the referential CONSTRAINT definition [Page 260]
  `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 246] must not contain a key definition [Page 267].

LONG data type [Page 251]: only NOT NULL or a DEFAULT specification [Page 256] may be specified as a column attribute for LONG columns.

UNIQUE

The UNIQUE column attribute determines the uniqueness of column values (see also CREATE INDEX statement [Page 302]).

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 256] STAMP can be defined.

NOT NULL

NOT NULL must not be used together with the DEFAULT specification [Page 256] DEFAULT NULL.

NOT NULL WITH DEFAULT defines a default value that is independent of the data type of the column. NOT NULL WITH DEFAULT must not be used with any DEFAULT specification.

Data Type	DEFAULT Value
`[VAR]CHAR(n)`	`' '`
`[VAR]CHAR(n) BYTE`	`X'00'`
`FIXED(p,s), INT[EGER], SMALLINT, FLOAT(p)`	`0`
`DATE`	`DATE`
`TIME`	`TIME`
`TIMESTAMP`	`TIMESTAMP`
`BOOLEAN`	`FALSE`

DEFAULT specification

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 57], unsigned integer [Page 68]

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 19] and can only be specified for columns of the data type DATE.
`DEFAULT TIME`	Supplies the current time [Page 20] and can only be specified for columns of the data type TIME.
`DEFAULT TIMESTAMP`	Supplies the current timestamp [Page 21] 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 22].
`DEFAULT TRANSACTION`	Supplies the identification of the current transaction [Page 35] 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 23]). 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 38]) 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 240], constraint name [Page 90]

Simple constraint (for one column), model table customer [Page 194]

```
title CHAR (7) CONSTRAINT title IN ('Mr', 'Mrs', 'Comp')
```

Complex constraint (for several columns), model table reservation [Page 198]

```
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 250] 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 of the CONSTRAINT definition must not contain a subquery [Page 388].

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 from all other constraint names in the 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 246]), you can check whether an additional DEFAULT value (default spec [Page 256]) 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 38]) 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 100], delete rule [Page 263]

`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 194] and reservation [Page 198]. 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 246] or ALTER TABLE statement [Page 271]. 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 268] 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 100] 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 266]
- 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 263]
- CASCADE dependency [Page 264]
- Reference cycle [Page 265]
- 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 412]).

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 260]). The DELETE rule is also used when column attributes [Page 254] 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 266] 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 256] 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 260] 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 263] or from the CASCADE option [Page 270].
- 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 270]

Reference cycle

A reference cycle is a sequence of referential CONSTRAINT definitions [Page 260] 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 264]) 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 260] 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 246] or an ALTER TABLE statement [Page 271] 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 104]

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 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 246] defines the uniqueness of column value combinations.

Syntax

```
<unique definition> ::= [CONSTRAINT <index name>] UNIQUE (<column
name>,...)
```

index name [Page 95], column name [Page 104]

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 302] 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 25]).

Syntax

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

table name [Page 106], cascade option [Page 270]

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 of the metadata and rows in the base table, as well as view definitions [Page 25], indexes [Page 28], privileges [Page 32], synonyms [Page 29], and referential CONSTRAINT definitions [Page 260] derived from them 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 260] with a DELETE rule [Page 263], are processed according to the specified DELETE rule, a DELETE statement [Page 354] 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 264]

ALTER TABLE statement

An ALTER TABLE statement (`alter_table_statement`) changes the properties of a base table (see Table [Page 25]).

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> <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>
```

add definition [Page 272], drop definition [Page 274], alter definition [Page 276], modify definition [Page 278], referential constraint definition [Page 260], referential constraint name [Page 100], sample definition [Page 249]

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 260] 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 (`add_definition`) in the ALTER TABLE statement [Page 271].

Syntax

```
<add_definition> ::= ADD <column_definition>,...
| ADD (<column_definition>,...)
| ADD <constraint_definition>
| ADD <referential_constraint_definition>
| ADD <key_definition>
```

column definition [Page 250], constraint definition [Page 258], key definition [Page 267], referential constraint definition [Page 260]

The following statement adds two columns to the customer [Page 194] table. The columns initially contain the NULL value in all rows.

```
ALTER TABLE customer ADD (telephone FIXED (8), street CHAR
(15))
```

The new columns can be used immediately.

Explanation

Adding a column definition: ADD <column definition>

A domain name [Page 92] can only be specified in a column definition [Page 250] if the domain was defined without a DEFAULT specification [Page 256].

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 space requirements for each column are increased by one character (for normal space requirements, see Space Requirements of a Column Value as a Function of the Data Type [Page 253]), if the length described is less than 31 characters and the column does not have the data type VARCHAR.

None of the newly defined columns may have the data type LONG. 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 258] 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 246].

- If view tables are defined for the specified table, and if alias names [Page 87] 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 CONSTRAINT definition: ADD <constraint_definition>

All of the rows in the table must satisfy the condition defined by the SEARCH condition [Page 240] of the CONSTRAINT definition [Page 258].

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 260] 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 267] 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, as it involves a large number of copy operations.

DROP definition

You can delete table properties by specifying a DROP definition in the ALTER TABLE statement [Page 271].

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 104], cascade option [Page 270], constraint name [Page 90]

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 260] 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 376] 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 374] of a view definition and if no CASCADE condition [Page 270] 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.
- If one of the columns to be dropped occurs in the QUERY specification of a view definition and if no CASCADE condition 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.

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

All CONSTRAINT definitions [Page 258] 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 260].

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.

ALTER definition

By specifying an ALTER definition (`alter_definition`) in the ALTER TABLE statement [Page 271], you can change the properties of a column or a CONSTRAINT definition [Page 258].

Syntax

```
<alter_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
ALTER CONSTRAINT <constraint_name> CHECK <search_condition>
ALTER <key_definition>
```

column name [Page 104], default spec [Page 256], constraint name [Page 90], search condition [Page 240], key definition [Page 267]

Explanation

NOT NULL

`NOT NULL` can only be specified if the column does not contain any NULL values (see data type [Page 15]). 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 258] 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 260] with the DELETE RULE [Page 263] ON DELETE SET DEFAULT, the ALTER TABLE statement will fail.

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 267]).

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, as it involves a large number of copy operations.

MODIFY definition

You can modify data types and properties of table columns by specifying a MODIFY definition in the ALTER TABLE statement [Page 271].

Syntax

```
<modify definition> ::= MODIFY (<column name> [<data type>] [<column
attributes>]...)
```

column name [Page 104], data type [Page 251], column attributes [Page 254]

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`
- `NOT NULL`
  If NOT NULL is specified, the table must not have any rows that contain a NULL value (see data type [Page 15]) in the corresponding column. A NULL value can no longer be inserted into the column after being modified.
- DEFAULT specification [Page 256]
  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, TIME, or TIMESTAMP or the code attribute ASCII or EBCDIC before it is modified.
- **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 **[VAR]CHAR(n)**: the corresponding column must have the data type [VAR]CHAR(m), 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.

Data type **DATE**: the corresponding column must have the data type [VAR]CHAR(m) or DATE. 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(p,s), FLOAT, INTEGER, or SMALLINT. 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(p,s), FLOAT, INTEGER, or SMALLINT.

Data type **INTEGER**: the corresponding column must have the data type FIXED(p,s), FLOAT, INTEGER, 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(p,s), FLOAT, INTEGER, 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 [VAR]CHAR(m) or TIME. 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 [VAR]CHAR(m) or TIMESTAMP. 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 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 240] 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 25]).

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 106], identifier [Page 78]

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 29] 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 TABLE statement changes the name of a table column.

Syntax

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

table name [Page 106], column name [Page 104]

Explanation

The specified table [Page 25] 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 106]

Explanation

The specified table [Page 25] must be a base table, view table, or a synonym [Page 29].

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 27].

Syntax

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

domain name [Page 92], data type [Page 251], default spec [Page 256], constraint definition [Page 258]

Explanation

The CONSTRAINT definition must not contain a constraint name [Page 90].

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 240] as a column name.

DROP DOMAIN statement

A DROP DOMAIN statement drops the definition of a domain [Page 27].

Syntax

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

domain name [Page 92]

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 generates 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 93], sequence name [Page 103], integer [Page 69], unsigned integer [Page 68]

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 113]). 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 93], sequence name [Page 103]

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 29] (alternative name) of a table name [Page 106].

Syntax

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

owner [Page 93], synonym name [Page 105], table name [Page 106]

Explanation

The table name must not denote a temporary base table (see Table [Page 25]). The user must have a privilege for 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 29] (alternative name) of a table name [Page 106].

Syntax

```
<drop synonym statement> ::= DROP [PUBLIC] SYNONYM [<owner>.]<synonym
name>
```

owner [Page 93], synonym name [Page 105]

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 29].

Syntax

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

old/new synonym name [Page 105]

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 generates a view table (see Table [Page 25]). 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 106], alias name [Page 87], query expression [Page 368]

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 110]. The QUERY expression must not reference a temporary table or a result table name [Page 94].

The table expressions [Page 379] of the QUERY specification [Page 374] in the QUERY statement of the CREATE VIEW statement must not contain a QUERY expression.

If a column selected [Page 376] by the QUERY statement is of the data type LONG, the FROM clause [Page 380] 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 the 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 240] 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 293]
- Updateable view table [Page 294]
- INSERT privilege for owners of a view table [Page 295]
- UPDATE privilege for owners of a view table [Page 296]
- DELETE privilege for owners of a view table [Page 297]
- If DISTINCT was specified (see DISTINCT specification [Page 375]), a SELECT ORDERED statement: searched) [Page 402] cannot be executed on the defined view table.
- Updateable join view table [Page 298]
- 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 291]) 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 240] in the QUERY expression [Page 368] of the CREATE VIEW statement contains a subquery [Page 388].
- The CREATE VIEW statement contains an outer join, that is an OUTER JOIN indicator in a JOIN predicate [Page 226] of the SEARCH condition.

Updateable view table

A view table (see CREATE VIEW statement [Page 291]) is called updateable if it is not a complex view table [Page 293], and if it is not based on a complex view table.

For join view tables, i.e. view tables whose FROM clause [Page 380] 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 260] 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 226] in the SEARCH condition [Page 240] of the CREATE VIEW statement, i. e. 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 376], but not both.
- The view table must be defined with WITH CHECK OPTION.

INSERT privilege for owners of a view table

The owner [Page 93] of the view table (see CREATE VIEW statement [Page 291]) 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 294]).
- The owner of the view table has the INSERT privilege for all tables in the FROM clause [Page 380] of the CREATE VIEW statement.
- The selected columns [Page 376] of the CREATE VIEW statement consist of table columns or column names, not expressions [Page 209] 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 owners of a view table

The owner [Page 93] of the view table (see CREATE VIEW statement [Page 291]) 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 294]).
- 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 209] with more than one column name.

DELETE privilege for owners of a view table

The owner [Page 93] of the view table (see CREATE VIEW statement [Page 291]) 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 294]).
- The owner of the view table has the DELETE privilege for all tables in the FROM clause [Page 380] 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 291]) in the FROM clause [Page 380] 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 260] 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 226] (PKj1=Fki1 AND ... AND PKjm=FKjm) is part of the SEARCH condition [Page 240] 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 376] 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 ith column of V and s is the ith 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},…,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; i.e. 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 25]).

Syntax

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

table name [Page 106], cascade option [Page 270]

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 29], view tables, and privileges 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 25]).

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 106]

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 291] 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 28] of a base table (see table [Page 25]).

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 106], column name [Page 104], index name [Page 95]

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 be of the data type LONG. 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 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 values (see data type [Page 15]) in single-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 28] for a base table (see table [Page 25]). 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 106], column name [Page 104], index name [Page 95]

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 (`alter_index_statement`) determines how an index [Page 28] 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 95], table name [Page 106]

Explanation

When a CREATE INDEX statement [Page 302] 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 342], UPDATE statement [Page 349], DELETE statement [Page 354]). 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 28].

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 106], index name [Page 95]

Explanation

The specified table name must be the name of an existing base table (see table [Page 25]).

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 catalog.

Syntax

```
<comment on statement> ::= COMMENT ON <object spec> IS <comment>
```

```
<object spec> ::= see explanation
<comment> ::= <string literal> | <parameter name>
```

string literal [Page 58], parameter name [Page 98]

Explanation

Comments can be specified for the following database objects:

<object spec> ::=	Explanation
`COLUMN <table name>.<column name>` table name [Page 106], column name [Page 104]	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 429].
`DBPROC[EDURE] <dbproc name>` dbproc name [Page 91]	`dbproc name` must identify an existing database procedure [Page 39] 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 434].
`DOMAIN <domain name>` domain name [Page 92]	`domain name` must specify a domain [Page 27] of the current user. The comment for this domain can be interrogated by selecting the system table DOMAIN.DOMAINS [Page 437].
`FOREIGN KEY <table name>.<referential constraint name>` referential constraint name [Page 100]	`referential constraint name` must specify a referential CONSTRAINT definition [Page 260] 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 438].

`INDEX <index name> ON <table name>`
`INDEX <index name.<column name>`
index name [Page 95]
	`index name` must specify an index [Page 28] and `column name` a column [Page 26] (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 439].

`[PUBLIC] SYNONYM <synonym name>` synonym name [Page 105]	`synonym name` must specify a synonym [Page 29] 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 447].
`TABLE <table name>` table name [Page 106]	The specified table [Page 25] 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 449].
`TRIGGER <trigger name> ON <table name>` Trigger name [Page 108]	The specified trigger name must identify a trigger [Page 41] 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 452].
`USER <user name>` user name [Page 89]	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 453].
`USERGROUP <usergroup name>` Usergroup name [Page 88]	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 453].
`<parameter name>` parameter name [Page 98]	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 39].

Syntax

```
<create dbproc statement> ::= CREATE DBPROC <dbproc name> [(<formal
parameter>,..)] AS <routine>

<formal parameter> ::=
  IN <argument> <data type>
| OUT <argument> <data type>
| INOUT <argument> <data type>

<argument> ::= <identifier>
```

dbproc name [Page 91], data type [Page 251], identifier [Page 78], routine [Page 310]

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 41].

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 357], 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 310]

Routine

The part of the CREATE DBPROC [Page 308] or CREATE TRIGGER-statement [Page 314] referred to as the routine is the implementation of the database procedure [Page 39] or trigger [Page 41]. It comprises 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 78], data type [Page 251], statement [Page 311]

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. 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 311]).

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 310]. The statements specified in the `statement` syntax description can be used to define a database procedure (see CREATE DBPROC statement [Page 308]) or trigger (see CREATE TRIGGER statement [Page 314]).

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> ::= <close 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>

<variable name> ::= <identifier>
```

expression [Page 209], search condition [Page 240], close statement [Page 398], delete statement [Page 354], fetch statement [Page 395], insert statement [Page 342], lock statement [Page 421], select statement [Page 366], named select statement [Page 364], single select statement [Page 399], select direct statement: searched [Page 400], select direct statement: positioned [Page 401], select ordered statement: searched [Page 402], select ordered statement: positioned [Page 405], subtrans statement [Page 419], identifier [Page 78]

Explanation

Variables specified in a routine [Page 310] can be assigned a value with the `assignment statement`.

Control structures

The **IF statement** first evaluates the SEARCH condition [Page 240]. 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

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 380] is sufficient.

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 39].

Syntax

```
<drop dbproc statement> ::= DROP DBPROC <dbproc name>
```

dbproc name [Page 91]

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 creates a trigger [Page 41] for a base table (see Table [Page 25]).

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 108], table name [Page 106], search condition [Page 240], routine [Page 310], column name [Page 104]

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.hno <> OLD.hno
  THEN UPDATE tours.room SET hno = NEW.hno WHERE hno =
OLD.hno;
CATCH
  IF $rc <> 100
  THEN STOP ($rc, 'Unexpected error');
)
```

Explanation

A trigger is a special type of database procedure [Page 39] that is assigned to a base table. This database procedure cannot be executed explicitly with the CALL statement [Page 357], 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 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 **DELTE 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.

See also:

routine [Page 310]

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 419] is not allowed in a trigger.

If a **WHENEVER** statement is specified, the trigger is only executed if the SEARCH condition [Page 240] is fulfilled. The condition must not contain a subquery [Page 388] or set function [Page 199].

DROP TRIGGER statement

A DROP TABLE statement deletes a trigger [Page 41] for a table.

Syntax

```
<drop trigger statement> ::= DROP TRIGGER <trigger name> OF <table
name>
```

trigger name [Page 108], table name [Page 106]

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 318]	DROP USER statement [Page 324]	ALTER USER statement [Page 326] RENAME USER statement [Page 330] GRANT USER statement [Page 332]
CREATE USER GROUP statement [Page 321]	DROP USER GROUP statement [Page 325]	ALTER USER GROUP statement [Page 328] RENAME USER GROUP statement [Page 331] GRANT USER GROUP statement [Page 333]
CREATE ROLE statement [Page 335]	DROP ROLE statement [Page 336]	
ALTER PASSWORD statement [Page 334]	GRANT statement [Page 337]	REVOKE statement [Page 340]

CREATE USER statement

The CREATE USER statement is used to create 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 89], user mode [Page 320], unsigned integer [Page 68], usergroup name [Page 88]

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 320])

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 407]. 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 318]). 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 usergroups [Page 30]

usergroup name [Page 323]

CREATE USERGROUP statement

The CREATE USERGROUP statement is used to create a usergroup [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 88], usergroup mode [Page 323], unsigned integer [Page 68]

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 318]. 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 (user mode [Page 323])

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 407]. 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 321]).

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 320]

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 89], cascade option [Page 270]

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 320] 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 88], cascade option [Page 270]

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 (`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 89], user mode [Page 320], unsigned integer [Page 68], role name [Page 102], password [Page 97]

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 user group.

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 deleted.
- **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 canceled.

DEFAULT ROLE

DEFAULT ROLE defines which of the roles assigned to the user are 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 are 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 318].

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 (`alter_usergroup_statement`) alters the properties assigned to a user group [Page 30].

Syntax

```
<alter_usergroup_statement> ::= ALTER USERGROUP <user_group_name>
[<user_ group_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 group name [Page 88], user group mode [Page 323], unsigned integer [Page 68]

Explanation

At least one of the optional clauses must be specified.

The specified user group must identify a defined user group.

The current user must have owner authorization over the user group whose properties are to be altered.

The members of the specified user group must not be logged onto the database system when the ALTER USER GROUP statement is executed.

User class of the user group (user group mode)

- **RESOURCE**: specifies that the user group is to be assigned the user class RESOURCE.
- **STANDARD**: specifies that the user group is removed from the current user class and loses the right to create base tables. All the base tables created by the user group 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 canceled.

DEFAULT ROLE

DEFAULT ROLE defines which of the roles assigned to the user group are activated automatically when a session is opened by a group member.

- **ALL**: All the roles assigned to the user group are activated when a session is opened. EXCEPT can be used to exclude specified roles from activation.
- **NONE**: None of the roles are activated when a session is opened by a member of the user group.

- **Role name specified**: The roles specified here must exist and be assigned to the user group. 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 USER GROUP statement [Page 321].

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 89]

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 88]

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 89]

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 320] 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 then 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 89], usergroup name [Page 88]

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 (`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 89], password [Page 97]

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 412] the next time the user starts a session.

CREATE ROLE statement

The CREATE ROLE statements defines a role [Page 33].

Syntax

```
<create role statement> ::= CREATE ROLE <role name> [IDENTIFIED BY
<password>]
```

role name [Page 102], password [Page 97]

Explanation

A role combines a set of privileges [Page 32] that can be assigned to users, usergroups [Page 30], or other roles by specifying the role name in the GRANT statement [Page 337]. 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 326] or ALTER USERGROUP statement [Page 328]. 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 416]. 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 245] 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 33].

Syntax

```
<drop role statement> ::= DROP ROLE <role name>
```

role name [Page 102]

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 32] for tables, individual columns and roles, 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>,...
```

priv spec [Page 338], grantee [Page 339], dbproc name [Page 91]

Explanation

The privileges in the privilege specification are assigned to the users [Page 30], user groups [Page 30], and roles [Page 33] 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

GRANT EXECUTE allows the user identified by `grantee` to execute the specified database procedure [Page 39]. The current user must be the owner of the database procedure.

Privilege specification (priv spec)

A privilege specification (`priv spec`) defines a role or a set of privileges [Page 32] for specific tables.

Syntax

```
<priv spec> ::= ALL [PRIV[ILEGES]] ON [TABLE] <table name>,...
| <privilege>,... ON [TABLE] <table name>,... | <role name>
```

table name [Page 106], privilege [Page 99], role name [Page 102]

Explanation

These tables must not be temporary base tables.

The user [Page 30] must have the authorization to grant (GRANT statement [Page 337]) and revoke (REVOKE statement [Page 340]).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 25]), 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 99].

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

The user [Page 30] (or several users) or role [Page 33] is specified for which privileges are to be granted with the GRANT statement [Page 337] or revoked with the REVOKE statement [Page 340].

Syntax

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

user name [Page 89], usergroup name [Page 88], role name [Page 102]

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 416] or by including the role in the set of roles automatically activated when a session was opened with the ALTER USER statement [Page 326] or ALTER USERGROUP statement [Page 328].

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 32].

Syntax

```
<revoke statement> ::= REVOKE <priv spec>,... FROM <grantee>,...
[<cascade option>]
| REVOKE EXECUTE ON <dbproc name> FROM <grantee>,...
```

priv spec [Page 338], grantee [Page 339], cascade option [Page 270], dbproc name [Page 91]

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 99]) 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 291]), the column defined by `select column` is dropped from the view table.
  If this view table is used in the FROM clause [Page 380] 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 25]), privileges [Page 32], and synonyms [Page 29] based on this view table, if no CASCADE option [Page 270] 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 39] 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 409]) 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 342]	UPDATE statement [Page 349]	DELETE statement [Page 354]
NEXT STAMP statement [Page 356]	CALL statement [Page 357]	

INSERT statement

The INSERT statement creates new rows in a table [Page 25].

Syntax

```
<insert statement> ::=
  INSERT [INTO] <table name> [(<column name>,...)] VALUES
(<extended expression>,...) [<duplicates clause>]
| INSERT [INTO] <table name> [(<column name>,...)] <query expression>
[<duplicates clause>]
| INSERT [INTO] <table name> SET <set insert clause>,...
[<duplicates clause>]
```

table name [Page 106], column name [Page 104], extended expression [Page 345], duplicates clause [Page 346], query expression [Page 368], set insert clause [Page 347]

Explanation

The table name must denote an existing base table, view table (see Table [Page 25]), or a synonym [Page 29].

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 INERST condition. A column must not occur more than once in a sequence of column names or in more than one SET INSERT condition.

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 updated.

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 246] or CREATE VIEW statement [Page 291]. 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 ith expression is assigned to the ith column name.
- The number of selected columns [Page 376] 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.

Data type of the target column and data type of the value to be inserted [Page 348]

Join view table

If the table name does **not identify a join view table** (see CREATE VIEW statement [Page 291]) and a row containing the key of the row to be inserted already exists, the result will depend on the DUPLICATES clause [Page 346]. 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 statement

If a QUERY expression [Page 368] is specified in the INSERT statement, the specified table must not be a join view table.

The QUERY expression defines a result table (se result table name [Page 94]) whose i^{th} column is assigned to the ith 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 256] 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 (see data type [Page 15]).

DUPLICATES clause

See DUPLICATES clause [Page 346]

CONSTRAINT definitions

If CONSTRAINT definitions [Page 258] exist for base tables in which rows are to be inserted with the INSERT statement, 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 260], 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 302]) in the corresponding referenced table. The INSERT statement fails if this is not the case for at least one row.

Triggers

If triggers [Page 41] that are to be executed after an INSERT statement 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.

Further information

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.

Extended expression

An extended expression can be specified by means of an expression [Page 209] or one of the keywords DEFAULT or STAMP.

Syntax

```
<extended expression> ::= <expression> | DEFAULT | STAMP
```

expression [Page 209]

Explanation

- Expression
  INSERT statement [Page 342]: an expression in an INSERT statement must not contain a column specification [Page 109].
  The value specified by a parameter specification [Page 110] 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 110] 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 342] or in the UPDATE statement [Page 349], but only for columns of the data type CHAR(n) BYTE with n>=8 (default specification [Page 256]).
  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 356]

See also:

SET INSERT condition [Page 347]

SET UPDATE condition [Page 352]

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 246]

`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 342]

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.

SET INSERT clause

SET INSERT clause

Syntax

```
<set insert clause> ::= <column name> = <extended value spec>
```

column name [Page 104], extended value spec [Page 112]

Explanation

SQL statements in which the SET INSERT clause is used

INSERT statement [Page 342]

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 342].
v is to be used to modify C with an UPDATE statement [Page 349].

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 61], decimal places are rounded off if necessary.
Alphanumeric column with the code attribute ASCII or EBCDIC	Character string [Page 16] 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 19] in the current date format.
Data type TIME	Time value [Page 20] in the current time format.
Data type TIMESTAMP	Timestamp value [Page 21] in the current timestamp format.
Data type BOOLEAN	One of the values TRUE or FALSE (BOOLEAN [Page 22])

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 106], reference name [Page 101], set update clause [Page 352], key spec [Page 117], search condition [Page 240], column name [Page 104], extended value spec [Page 112], result table name [Page 94]

Explanation

The table name must identify an existing base table, view table (see Table [Page 25]), or a synonym [Page 29].

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 112]) must be identical to the number of target columns. The ith value specification is assigned to the ith 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 348]

The following specifications determine the rows in the table that are updated:

- Optional sequence of key specifications [Page 117] and optional SEARCH condition [Page 240]
  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 94] (`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 246] or ALTER TABLE statement [Page 271] 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 291]), 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 353]). 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 260] that is relevant for the join view table.

CURRENT OF

If CURRENT OF is specified, the table name in the FROM clause [Page 380] 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 258] 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 302])

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 260] 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 41] 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 INSERT 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 104], extended expression [Page 345], subquery [Page 388]

Explanation

The expression of a SET UPDATE clause must not contain a set function [Page 199].

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 349]

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 349], 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 106], reference name [Page 101], key spec [Page 117], search condition [Page 240], result table name [Page 94]

Explanation

The table name must identify an existing base table, view table (see Table [Page 25]), or a synonym [Page 29].

The current user must have the DELETE privilege for the specified table. If the specified table name identifies a view table, even the owner of the view table might not have the DELETE privilege because the view table is not updateable.

Table name identifies a view table: the rows are deleted from the base tables, on which the view table is based.

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 117] and optional SEARCH condition [Page 240]
  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 94] (`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 380] 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 260], one of the following actions is carried out - depending on the DELETE rule [Page 263] 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 256] is assigned to the respective foreign key columns of all matching rows in the corresponding foreign key table.

Trigger

If triggers [Page 41] 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 98]

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(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 342] or an UPDATE statement [Page 349] 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 39] to be executed.

Syntax

```
<call statement> ::= CALL <dbproc name> [(<expression>,...)] [WITH
COMMIT]
```

dbproc name [Page 91], expression [Page 209]

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 ith expression must be compatible with the data type of the ith formal parameter for the database procedure.

If the ith formal parameter for the database procedure has the OUT or INOUT mode (see CREATE DBPROC statement [Page 308]), the corresponding expression must be a parameter specification [Page 110].

WITH COMMIT

If WITH COMMIT is specified, the current transaction [Page 35] 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 359]	SINGLE SELECT statement [Page 399]	EXPLAIN statement [Page 407]
SELECT DIRECT statement: searched [Page 400]	SELECT DIRECT statement: positioned [Page 401]	
SELECT ORDERED statement: searched [Page 402]	SELECT ORDERED statement: positioned [Page 405]	
OPEN CURSOR statement [Page 394]	FETCH statement [Page 395]	CLOSE statement [Page 398]

QUERY statement

A QUERY statement specifies a result table that can be ordered (see result table name [Page 94]). 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 362], recursive declare cursor statement [Page 363], named select statement [Page 364], select statement [Page 366]

Explanation

A QUERY statement generates a named/unnamed result table [Page 361].

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 394].
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 362] 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 390].

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 364] or SELECT statement (select statement) [Page 366]
- 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 94]) is that the unnamed result table cannot be specified in a FROM clause [Page 380] or in CURRENT OF <`result table name`> of a subsequent SQL statement.

QUERY statement	
DECLARE CURSOR statement [Page 362]	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 366]	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 364]	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 361]) with the name `result table name`.

Syntax

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

result table name [Page 94], select statement [Page 366]

Explanation

An OPEN CURSOR statement [Page 394] 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 366]

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 94], reference name [Page 101], alias name [Page 87], query spec [Page 374], select statement [Page 366]

```
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 374] **`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 380] and one JOIN predicate [Page 226] 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 nth 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 374].
  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 202], GROUP clause [Page 386], HAVING clause [Page 387], ORDER clause [Page 390], LOCK option [Page 392]

If a result table name [Page 94] 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 361]).

Syntax

```
<named select statement> ::= <named query expression>
[<order clause>] [<update clause>] [<lock option>] [FOR REUSE]
```

named query expression [Page 371], order clause [Page 390], update clause [Page 391], lock option [Page 392]

Explanation

An OPEN CURSOR statement [Page 394] 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 362] 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 395] 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 378].
- Only one base table or one updateable view table may be specified in the FROM clause [Page 380] of the QUERY specification (`named query spec`).
- The DISTINCT keyword (see DISTINCT specification [Page 375]), a GROUP clause [Page 386], or HAVING clause [Page 387] must **not** be specified.
- Expressions must not contain a set function (set function spec) [Page 199].
- See also the section entitled "Updateable result table" under QUERY statement [Page 359].

ORDER clause

The ORDER clause [Page 390] specifies a sort sequence for a result table.

UPDATE clause

An UPDATE clause [Page 391] 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

349] or a DELETE statement [Page 354]. The position in a base table can be used to issue a SELECT DIRECT statement [Page 401] or a SELECT ORDERED statement [Page 405], or a LOCK statement [Page 421] can be used to request a lock for the row concerned in each base table involved.

LOCK option

The LOCK option [Page 392] 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 359], 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 366]

SELECT statement (select statement)

A SELECT statement (`select statement`) defines and creates an unnamed result table (see named/unnamed result table [Page 361]).

Syntax

```
<select statement> ::= <query expression> [<order clause>]
[<update clause>] [<lock option>] [FOR REUSE]
```

query expression [Page 368], order clause [Page 390], update clause [Page 391], lock option [Page 392]

Explanation

An OPEN CURSOR statement [Page 394] 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 362] 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 395] 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 359] comprises a DECLARE CURSOR statement.
- The QUERY expression (`query expression`) must only comprise one QUERY specification (query spec) [Page 374].
- Only one base table or one updateable view table may be specified in the FROM clause [Page 380] of the QUERY specification (`query spec`).
- The DISTINCT keyword (see DISTINCT specification [Page 375]), a GROUP clause [Page 386], or HAVING clause [Page 387] must **not** be specified.
- Expressions must not contain a set function (set function spec) [Page 199].
- See also the section entitled "Updateable result table" under QUERY statement [Page 359].

ORDER clause

The ORDER clause [Page 390] specifies a sort sequence for a result table.

UPDATE clause

An UPDATE clause [Page 391] 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 349] or a DELETE statement [Page 354]. The position in a base table can be used to issue a SELECT DIRECT statement [Page 401] or a SELECT ORDERED statement [Page 405], or a LOCK statement [Page 421] can be used to request a lock for the row concerned in each base table involved.

LOCK option

The LOCK option [Page 392] 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 359], 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 364]

QUERY expression (query expression)

QUERY expressions are required to generate an unordered result table in a SELECT statement [Page 366].

Syntax

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

query term [Page 370]

Explanation

If the QUERY expressions consists of only one QUERY specification (query spec) [Page 374] (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 ith column of the result table is a numeric column.
Alphanumeric columns, code attribute BYTE	Are comparable.
Alphanumeric columns, code attribute ASCII, EBCDIC	Are comparable. Are also comparable with date, time, and timestamp values.
All i^{th} columns are date values [Page 19]	The ith column of the result table is a date value.
All i^{th} columns are time values [Page 20]	The ith column of the result table is a time value.
All i^{th} columns are timestamp values [Page 21]	The ith column of the result table is a timestamp value.
Columns of the type BOOLEAN [Page 22]	Are comparable.
All i^{th} columns are of the type BOOLEAN	The ith column of the result table is of the type BOOLEAN.
Columns of any other data type (not mentioned above)	The ith column of the result table is an alphanumeric column. Comparable columns with differing code attributes are converted.

If columns are comparable but have different lengths, the corresponding column of the result table has the maximum length of the underlying columns.

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 (see data type [Page 15]) are assumed to be identical. Special NULL values [Page 111] 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

A 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 374], query expression [Page 368]

Explanation

See QUERY expression [Page 368] or QUERY expression (named query expression) [Page 371]

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 364].

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 373], query term [Page 370]

Explanation

If the QUERY expressions consists of more than one QUERY specification (query spec) [Page 374] (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 378] (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 ith column of the result table is a numeric column.
Alphanumeric columns, code attribute BYTE	Are comparable.
Alphanumeric columns, code attribute ASCII, EBCDIC	Are comparable. Are also comparable with date, time, and timestamp values.
All i^{th} columns are date v values [Page 19]	The ith column of the result table is a date value.
All i^{th} columns are time values [Page 20]	The ith column of the result table is a time value.
All i^{th} columns are timestamp values [Page 21]	The ith column of the result table is a timestamp value.
Columns of the type BOOLEAN [Page 22]	Are comparable.
All i^{th} columns are of the type BOOLEAN	The ith column of the result table is of the type BOOLEAN.
Columns of any other data type (not mentioned above)	The ith column of the result table is an alphanumeric column. Comparable columns with differing code attributes are converted.

If columns are comparable but have different lengths, the corresponding column of the result table has the maximum length of the underlying columns.

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 (see data type [Page 15]) are assumed to be identical. Special NULL values [Page 111] are assumed to be identical.
- UNION: R contains all rows of 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 378], named query expression [Page 371], query spec [Page 374], query expression [Page 368]

Explanation

See QUERY expression (named query expression) [Page 371]

QUERY specification (query spec)

QUERY specifications (`query spec`) are required to generate an unordered result table in a SELECT statement [Page 366]. A QUERY specification is part of the syntax in a QUERY term (query term) [Page 370] or QUERY term (named query term) [Page 373].

Syntax

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

distinct spec [Page 375], select column [Page 376], table expression [Page 379]

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 379].

DISTINCT function (distinct spec)

The DISTINCT specification (`distinct spec`) is specified in a QUERY specification (query spec) [Page 374], QUERY specification (named query spec) [Page 378], or SINGLE SELECT statement [Page 399] 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 (see data type [Page 15]) are assumed to be identical, as are special NULL values [Page 111].

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 374] or a QUERY specification (named query spec) [Page 378] 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 380] of the table expression [Page 379]. 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 106], reference name [Page 101], expression [Page 209], identifier [Page 78]

Explanation

Every column name that is specified as a selected column must uniquely denote a column in a QUERY specification (query spec) [Page 374] 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 in a selected column is only valid in the uppermost sequence of select columns in a QUERY statement [Page 359], SINGLE SELECT statement [Page 399], SELECT DIRECT statement [Page 400], or a SELECT ORDERED statement [Page 402] if the DISTINCT specification [Page 375] 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 291] which is based on exactly one base table.

If a selected column contains a set function (set function spec) [Page 199], 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 209] 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 209] 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(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 368] of an INSERT statement [Page 342].
  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(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 19] (length 10) or a timestamp value [Page 21] (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 364]. A QUERY specification is part of the syntax in a QUERY term (named query term) [Page 373].

Syntax

```
<named query spec> ::= SELECT [<distinct spec>] <result table name>
(<select column>,...) <table expression>
```

distinct spec [Page 375], result table name [Page 94], select column [Page 376], table expression [Page 379]

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 379].

Table expression

A table expression specifies a single or a simple or grouped result table (see result table name [Page 94]).

Syntax

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

from clause [Page 380], where clause [Page 385], group clause [Page 386], having clause [Page 387]

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 379] that is formed from one or more tables.

Syntax

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

from table spec [Page 381]

Explanation

The FROM clause specifies a table. This table can be derived from several base, view, and result tables (see Table [Page 25]). 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

Each FROM TABLE specification (`from table spec`) in a FROM clause [Page 380] 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 106], reference name [Page 101], result table name [Page 94], query expression [Page 368], joined table [Page 383]

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 78] 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 93] 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 374] 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 389]. 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 380] 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.

FROM TABLE specification

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 379] containing at least one OUTER JOIN indicator (see JOIN predicate [Page 226]) or OUTER JOIN TYPE (LEFT | RIGHT | FULL) (see joined table [Page 383]) 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 374] 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 383]) specifies the number of table identifiers that are specified by the FROM TABLE specifications it contains.

Jointed table

JOINED TABLE (`joined table`) can be specified as part of a FROM TABLE specification (from table spec) [Page 381].

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 381], search condition [Page 240], column name [Page 104]

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 385] 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 218] between the specified columns in FT1 and FT2 linked with AND. = is used as a comparison operator (comp op [Page 220]).

WHERE clause

The WHERE clause specifies the conditions for building a result table (see result table name [Page 94]).

Syntax

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

search condition [Page 240]

Explanation

The SEARCH condition is applied to each row in the temporary result table formed by the FROM clause [Page 380]. 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 209] in the SEARCH condition must not contain a set function (set function spec) [Page 199].

Each column specification [Page 109] directly contained in the SEARCH condition must uniquely identify a column from the tables specified in the FROM clause of the table expression [Page 379]. 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 389], 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 374].

Each subquery [Page 388] 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 94]).

Syntax

```
<group clause> ::= GROUP BY <expression>,...
```

expression [Page 209]

Explanation

Each column name specified in the GROUP clause must identify a `result column name` in the selected columns [Page 376] of the QUERY specification [Page 374] 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 207], AVG [Page 203], MAX/MIN [Page 205], COUNT [Page 204], STDDEV [Page 206], and VARIANCE [Page 208] to be applied not only to entire result tables but also to groups of rows within a result table. A group is defined by the grouping columns specified in GROUP BY. All rows of a group have the same values in the grouping columns. Rows containing the NULL value in a grouping column (see data type [Page 15]) are combined to form a group. The same is true for the special NULL value [Page 111].

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 209] that use the functions SUM, AVG, MAX/MIN, COUNT, STDDEV, and VARIANCE.

If no rows satisfy the conditions indicated in the WHERE clause [Page 385] 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 240]

Explanation

Each expression [Page 209] in the SEARCH condition that does not occur in the argument of a set function (set function spec) [Page 199] must identify a grouping column.

If the HAVING clause is used without a GROUP clause [Page 386], 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 94]) that can be used in certain predicates and for updating column values.

Syntax

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

query expression [Page 368]

Explanation

Subqueries can be used in a SET UPDATE condition [Page 352] of an UPDATE statement [Page 349]. In this case, the subquery must produce a result table that contains a maximum of one row.

Subqueries can be used in the following predicates:

Comparison predicate [Page 218]
EXISTS predicate [Page 223]
IN predicate [Page 224]
Quantified predicate [Page 235]

Correlated subquery

Certain predicates can contain subqueries. These subqueries, in turn, can contain other subqueries, etc. A subquery [Page 388] with further subqueries is the higher-level subquery of the subqueries it contains.

- The SEARCH condition [Page 240] of a subquery can contain column names that belong to tables that are contained in higher levels in the FROM clause [Page 380]. 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 195] and room [Page 196].
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 94]).

Syntax

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

```
<sort spec> ::= <unsigned integer> [ASC | DESC] | <expression> [ASC | DESC]
```

unsigned integer [Page 68], expression [Page 209]

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 nth 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 368] consists of more than one QUERY specification [Page 374], 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 380] or a `result column name` in the selected columns [Page 376] of the QUERY specification.

If DISTINCT or a set function [Page 199] 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 218]. For sorting purposes, NULL values are greater than non-NULL values, and special NULL values are greater than non-NULL values but less than NULL values (see data type [Page 15]).

UPDATE clause

The UPDATE clause specifies that a result table (see result table name [Page 94]) is to be updateable.

Syntax

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

column name [Page 104]

Explanation

The specified column names must identify columns in the tables underlying the QUERY specification [Page 374]. They do not have to occur in a selected column [Page 376].

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 349], DELETE statement [Page 354], LOCK statement [Page 421], SELECT DIRECT statement [Page 401], and SELECT ORDERED statement [Page 405]. 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 104].

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 240] of the QUERY statement
- x is contained in a SET UPDATE clause [Page 352] of the UPDATE statement in the type `x = <expression>`, where the expression [Page 209] 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] [ISOLATION LEVEL <unsigned integer>]
WITH LOCK [(NOWAIT)] OPTIMISTIC [ISOLATION LEVEL <unsigned integer>]
```

unsigned integer [Page 68] may only have the values 0, 1, 2, 3, 10, 15, 20, or 30

Explanation

EXCLUSIVE

An exclusive lock (see transactions [Page 409]) 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 425].

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 412]. 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 364], SELECT statement (select statement) [Page 366], or an OPEN CURSOR statement [Page 394], 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 395] 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 362].

Syntax

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

result table name [Page 94]

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 closed at the end of the transaction using the ROLLBACK statement [Page 418].

All result tables are implicitly deleted at the end of the session using the RELEASE statement [Page 426]. A CLOSE statement [Page 398] 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 25]), or a synonym [Page 29], 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 395] 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 94]) to parameters.

Syntax

```
<fetch statement> ::=
FETCH [FIRST | LAST | NEXT | PREV] [<result table name>] INTO
<parameter spec>,...
FETCH [<position>] [<result table name>] INTO <parameter spec>,...
FETCH [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 94], parameter spec [Page 110], unsigned integer [Page 68], integer [Page 69]

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 361]).

Depending on the search strategy, either all the rows in the result table are searched when the OPEN CURSOR statement [Page 394], SELECT statement (select statement) [Page 366], or SELECT statement (named select statement) [Page 364] is executed and the result table is physically generated, or each next result table row is searched when a FETCH statement executed, without being 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 390], 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 376]. 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 110] specified in **position** must denote an integer.

The remaining parameters in the parameter specification are output parameters. The parameter identified by the nth parameter specification corresponds to the nth value in the current result table row. If the number of columns in this row exceeds the number of specified parameters, the column values for which no corresponding parameters exist are ignored. If the number of

columns in the row is less than the number of specified parameters, no values are assigned to the remaining parameters. An indicator name [Page 96] must be specified to assign NULL values or special NULL values (see data type [Page 15]).

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 366]), 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 421] for the entire table or by using the isolation level 2, 3, 15, 20, or 30 with the CONNECT statement [Page 412] or the LOCK option [Page 392] 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 17] 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 94]).

Syntax

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

result table name [Page 94]

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 366].

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 418].

All result tables are implicitly deleted at the end of the session using the RELEASE statement [Page 426].

SINGLE SELECT statement

A SINGLE SELECT statement specifies a result table with one row (see result table name [Page 94]) 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 375], select column [Page 376], parameter spec [Page 110], from table spec [Page 381], where clause [Page 385], group clause [Page 386], having clause [Page 387], lock option [Page 392]

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 395] rules apply for assigning the values to the parameters.

The order of the GROUP and HAVING clauses is random.

A LONG column [Page 17] 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 375] 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 376], parameter spec [Page 110], table name [Page 106], key spec [Page 117], where clause [Page 385], lock option [Page 392]

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 240] specified for this row is true, the corresponding column values are assigned to the parameters. The FETCH statement [Page 395] 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 401]

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 376], parameter spec [Page 110], table name [Page 106], result table name [Page 94], lock option [Page 392]

Explanation

The table name in this SELECT DIRECT statement must be identical to that in the FROM clause [Page 380] 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 395] 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 400]

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 28]. 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 376], parameter spec [Page 110], table name [Page 106], where clause [Page 385], lock option [Page 392], index pos spec [Page 404], key spec [Page 117], column name [Page 104], index name [Page 95]

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 385] 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 an index position is specified, and the associated index only contains one column, the rows with NULL values in the index column (see data type [Page 15]) 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 17] can only be specified in a selected column in the uppermost sequence of selected columns in a SELECT ORDERED statement.

The LOCK option [Page 392] 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 395] 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 405]

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 402]

SELECT ORDERED statement (select ordered statement: positioned) [Page 405]

Syntax

```
<index pos spec> ::= INDEX <column name> = <value spec>
| INDEXNAME <index name> VALUES (<value spec>,...)
```

column name [Page 104], value spec [Page 113], index name [Page 95]

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 376], parameter spec [Page 110], table name [Page 106], column name [Page 104], index name [Page 95], result table name [Page 94], lock option [Page 392], index pos spec [Page 404]

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 380] 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.

SELECT ORDERED statement (select ordered statement: positioned)

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 an index position is specified, and the associated index only contains one column, the rows with NULL values in the index column (see data type [Page 15]) 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 392] determines which lock is set for the read row.

A LONG column [Page 17] 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 395] 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 402]

EXPLAIN statement

The EXPLAIN statement describes the search strategy used internally by the database system for a QUERY statement [Page 359] or SINGLE SELECT statement [Page 399] (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 94], query statement [Page 359], single select statement [Page 399]

Explanation

The EXPLAIN statement can be used to check the effect of creating or deleting indexes (see index [Page 28]) 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.

A result table (see result table name [Page 94]) is generated. 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 result table

OWNER	CHAR(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 458] 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 35] 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 37] is opened with the CONNECT statement [Page 412]. The transaction is concluded with the COMMIT statement [Page 417] or the ROLLBACK statement [Page 418]. 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 421]. 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 425] 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.

Can another Transaction	A transaction has a(n)					
	EXCL	SHARE	EXCL	SHARE	EXCL	SHARE
	Lock on a table		Lock on a row		Lock on the system catalog	
set an EXCLUSIVE lock on the table?	No	No	No	No	No	Yes
set a SHARE lock on the table?	No	Yes	No	Yes	No	Yes
set an EXCLUSIVE lock on any row in the table?	No	No	---	---	No	Yes
set an EXCLUSIVE lock on the locked table?	---	---	No	No	---	---
set an EXCLUSIVE lock on a different row?	---	---	Yes	Yes	---	---
set a SHARE lock onany row in the table?	No	Yes	---	---	No	Yes
set a SHARE lock on the locked row?	---	---	No	Yes	---	---
set a SHARE lock on a different row?	---	---	Yes	Yes	---	---
change the table definition in the system catalog?	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 419]) 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 412]	SET statement [Page 416]	
COMMIT statement [Page 417]	ROLLBACK statement [Page 418]	SUBTRANS statement [Page 419]
LOCK statement [Page 421]	UNLOCK [Page 425]	RELEASE statement [Page 426]

CONNECT statement

A CONNECT statement opens a session [Page 37] and a transaction [Page 35] 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 <password> [<connect option>...]

<connect option> ::=
  SQLMODE <INTERNAL | ANSI | DB2 | ORACLE>
| ISOLATION LEVEL <unsigned integer> | TIMEOUT <unsigned integer>
| TERMCHAR SET <termchar set name>
```

parameter name [Page 98], password [Page 97], user name [Page 89], unsigned integer [Page 68], termchar set name [Page 107]

Explanation

If the parameter name/user name and parameter name/password combination is valid, the user [Page 30] opens a 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 409]). The COMMIT statement [Page 417] or ROLLBACK statement [Page 418] 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

SQL mode (SQLMODE) [Page 43]

The specification `SQLMODE <INTERNAL | ANSI | DB2 | ORACLE>` can be used to select the SQL mode. 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 409]) can be requested either implicitly or explicitly. Explicit lock requests are made with the LOCK statement [Page 421]. 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 359], 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 117] 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 425] 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 240]. 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 426].

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

The ASCII code as per ISO 8859/1 or the EBCDIC code CCSID 500 is used throughout the database system. Since ASCII and EBCDIC code contains characters that have a different hexadecimal representation on some terminals, the keywords TERMCHAR SET can be used to define terminal character sets (terminal character set name [Page 107]) that converts the terminal representation of a character to the code used in the database system for inputs and outputs. The CONNECT statement can be used to select one of the defined terminal character sets which is then used for conversion purposes during the session. If no or an unsuitable terminal character set is selected, characters in the database that are to be output may not be displayed correctly on the terminal.

Set statement

The SET statement alters the properties of a session [Page 37].

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 102], password [Page 97], unsigned integer [Page 68]

Explanation

SET ROLE

DEFAULT ROLE in the ALTER USER statement [Page 326] or ALTER USERGROUP statement [Page 328] specifies which of the roles [Page 33] 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 412] defines when locks are set implicitly and how long they are retained. A LOCK option [Page 392] 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 409]).

Syntax

```
<commit statement> ::= COMMIT [WORK] [KEEP <lock statement>]
```

lock statement [Page 421]

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 412] controls the setting of locks in the new transaction.

LOCK statement

The LOCK statement [Page 421] 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 409]).

Syntax

```
<rollback statement> ::= ROLLBACK [WORK] [KEEP <lock statement>]
```

lock statement [Page 421]

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 412] 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 421] 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 subunits (see transactions [Page 409]).

Syntax

```
<subtrans statement> ::= SUBTRANS BEGIN | SUBTRANS END | SUBTRANS
ROLLBACK
```

Explanation

SUBTRANS BEGIN

A subtransaction [Page 36] 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 39] 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 417] and the ROLLBACK statement [Page 418] close any open subtransactions implicitly.

LOCK statement

The LOCK statement assigns a lock to the current transaction (see transactions [Page 409]).

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 424], table name [Page 106]

Explanation

The specified table [Page 25] must not be a base table, view table, or a synonym [Page 29]. 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 has a key column; i.e. if the table name identifies a view table, this must be updateable.

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 425] 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 409]

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 412].

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 a LOCK statement are normally released once this transaction is ended, provided that the COMMIT statement [Page 417] or ROLLBACK statement [Page 418] ending the transaction does 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 400], 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 421] or an UNLOCK statement [Page 425].

Syntax

```
<row spec> ::= ROW <table name> KEY <key spec>,...
| ROW <table name> CURRENT OF <result table name>
```

table name [Page 106], result table name [Page 94]

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 424]

Explanation

SHARE locks, optimistic locks, and exclusive locks (see transactions [Page 409]) 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 392] in SELECT statements and by LOCK statements [Page 421]. 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 35] and session [Page 37].

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 session of the user 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 418] 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. These system tables belong to the DOMAIN user. The DOMAIN user name must be placed in front of the name of the system table in all SQL modes other than INTERNAL.

COLUMNS

Use

This table contains the columns in all the tables, views, synonyms, and result tables for which the current user has privileges.

Structure

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)
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

Integration

System tables [Page 427]

CONNECTEDUSERS

Use

This table contains all of the users that are currently logged on.

Structure

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 session
DBPROC_CACHE_SIZE	FIXED(10)	Size of the cache for processing database procedures

Integration

System tables [Page 427]

CONNECTPARAMETERS

Use

This table contains information on session-specific parameters.

Structure

CONNECTPARAMETERS

SQLMODE	CHAR(8)	SQLMODE
ISOLEVEL	FIXED(10)	ISOLATION LEVEL
TIMEOUT	FIXED(10)	Session timeout value
TERMCHARSETNAME	CHAR(32)	Name of the character set used in this session

Integration

System tables [Page 427]

CONSTRAINTS

Use

This table contains the CONSTRAINT definitions for all the tables for which the current user has privileges.

Structure

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

Integration

System tables [Page 427]

DBPROCEDURES

Use

This table contains all the database procedures for which the current user has privileges.

Structure

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 type 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

Integration

System tables [Page 427]

DBPROCPARAMS

Use

This table contains all the parameters of a database procedures for which the current user has privileges.

Structure

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)
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

Integration

System tables [Page 427]

DOMAINCONSTRAINTS

Use

This table contains the CONSTRAINT definition of a domain.

Structure

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

Integration

System tables [Page 427]

DOMAINS

Use

This table contains all the domains.

Structure

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)
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

Integration

System tables [Page 427]

FOREIGNKEYS

Use

This table contains all the referential CONSTRAINT definitions for which the current user has privileges.

Structure

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

Integration

System tables [Page 427]

INDEXES

Use

This table contains all the indexes for which the current user has privileges.

Structure

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

Integration

System tables [Page 427]

LOCKS

Use

This table contains information on locks that have been set.

Structure

LOCKS

SESSION	FIXED(10)	Session identification
TRANSCOUNT	FIXED(10)	Transaction identification in the session
PROCESS	FIXED(10)	Process identification on the database server
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

Integration

System tables [Page 427]

MAPCHARSETS

Use

This table contains all of the MAPCHAR SETs.

Structure

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)	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

Integration

System tables [Page 427]

PACKAGES

Use

This table describes all the packages for which the current user has privileges.

Structure

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 type 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

Integration

System tables [Page 427]

ROLEPRIVILEGES

Use

This table 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

Integration

System tables [Page 427]

ROLES

Use

This table describes the roles for which the current user has privileges.

Structure

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

Integration

System tables [Page 427]

SEQUENCES

Use

This table contains all the sequences for which the current user has privileges.

Structure

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

Integration

System tables [Page 427]

SESSION_ROLES

Use

This table contains the roles that are active in the current session.

Structure

SESSION_ROLES

ROLE	CHAR(32)	Role name

Integration

System tables [Page 427]

SYNONYMS

Use

This table contains all the synonyms for which the current user has privileges.

Structure

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

Integration

System tables [Page 427]

TABLEPRIVILEGES

Use

This table describes the privileges that the current user has for tables.

Structure

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)

Integration

System tables [Page 427]

TABLES

Use

This table contains all the tables (base tables, views, synonyms, result tables) for which the current user has privileges.

Structure

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

Integration

System tables [Page 427]

TERMCHARSETS

Use

This table contains all the terminal character sets.

Structure

TERMCHARSETS

TERMCHARSETNAME	CHAR(32)	Name of the terminal character set
CODE	CHAR(8)	Code attribute for which the terminal character set was defined (ASCII | EBCDIC)
STATE	CHAR(8)	Terminal character set is activated (ENABLED | DISABLED)
INTERN	CHAR BYTE(1)	Original form in hexadecimal notation
EXTERN	CHAR BYTE(1)	Terminal code in hexadecimal notation
COMMENT	CHAR(8)	Comment on the terminal character set

Integration

System tables [Page 427]

TRIGGERPARAMS

Use

This table contains all the parameters of a trigger for which the current user has privileges.

Structure

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)
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

Integration

System tables [Page 427]

TRIGGERS

Use

This table contains all the triggers for which the current user has privileges.

Structure

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

Integration

System tables [Page 427]

USERS

Use

This table contains all the users.

Structure

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 the user
ALTERTIME	TIME	Change time 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

Integration

System tables [Page 427]

VERSIONS

Use

This table describes the versions of the SAP DB database server.

Structure

VERSIONS

KERNEL	CHAR(40)	Version of the SAP DB database server
RUNTIMEENVIRONMENT	CHAR(40)	Version of the runtime environment

Integration

System tables [Page 427]

VIEWDEFS

Use

This table contains the definitions of the view tables for which the current user has privileges.

Structure

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 of the view table definition

Integration

System tables [Page 427]

VIEWS

Use

This table contains all the view tables for which the current user has privileges.

Structure

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	Creation date of the view table
CREATETIME	TIME	Creation time of the view table
UPDSTATDATE	DATE	Date at which the UPDATE STATISTICS statement was last carried out on the view table
UPDSTATTIME	TIME	Time at which the UPDATE STATISTICS statement was last carried out on the view table
ALTERDATE	DATE	Alter date of the view table
ALTERTIME	TIME	Alter time of the view table
UNLOADED	CHAR(3)	View table is unloaded (YES | NO)
COMMENT	LONG	Comment on the view table

Integration

System tables [Page 427]

Statistics

The database system addresses disks in units of 8KB. In this section, the term page is used to refer to these units.

SQL statements for statistics management

UPDATE STATISTICS statement [Page 458]	MONITOR statement [Page 475]

See also:

Information on the Optimizer is available in the R/3 documentation entitled *Computing Center Management System*

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 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 106], column name [Page 104], sample definition [Page 249], owner [Page 93], identifier [Page 78]

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 catalog. These values are used by the Optimizer to determine the best strategy for executing SQL statements.

The UPDATE STATISTICS statement implicitly performs a COMMIT statement [Page 417] 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 302] 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 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 249] 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:

Statistical system tables [Page 460]

Statistical system tables

During the complete installation of database system, a set of system tables is created on the Serverdb. These system tables can be used to select information on the configuration, structures, and sizes of database objects.

The owner [Page 93] of these tables is the SYSDBA created when the system was configured. You do not need to specify the owner to access these tables.

See also:

UPDATE STATISTICS statement [Page 458]

DATADEVSPACES

Use

This table contains information on the occupied data devspaces.

Structure

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

Integration

Statistics system tables [Page 460]

DBPARAMETERS

Use

This table contains the parameters defined for the database instance with the Database Manager tool.

Structure

DBPARAMETERS

DESCRIPTION	CHAR (18)	description of how to interpret the column VALUE
VALUE	CHAR (64)	value

Integration

Statistics system tables [Page 460]

INDEXSTATISTICS

Use

This table contains information about structure and size of indexes.

Structure

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%

Integration

Statistics system tables [Page 460]

LOCKLISTSTATISTICS

Use

This table contains information on the lock list usage

Structure

LOCKLISTSTATISTICS

DESCRIPTION	CHAR (40)	Description of how to interpret the content of the VALUE column (see *Column DESCRIPTION* table)
VALUE	CHAR (12)	value

Column DESCRIPTION

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

Integration

Statistics system tables [Page 460]

SERVERDBSTATISTICS

Use

This table contains information on the serverdb usage

Structure

SERVERDBSTATISTICS

SERVERDBSIZE	FIXED (10)	Serverdb size in pages
MAXDATAPAGENO	FIXED (10)	Largest page number of the SERVERDB
MAXPERM	FIXED (10)	Number of pages of the serverdb that can be used for non-temporary objects
USEDPERM	FIXED (10)	Number of pages of the SERVERDB used for non-temporary objects
PCTUSEDPERM	FIXED (10)	Percentage of pages used for non-temporary objects
USEDTMP	FIXED (10)	Number of serverdb 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: serverdb full NO: serverdb 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 backed up
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
LOGSEGMENTSIZE	FIXED (10)	Size of a log segment in pages
SAVEPOINTS	FIXED (10)	Number of savepoints executed
CHECKPOINTS	FIXED (10)	Number of checkpoints executed

Integration

Statistics system tables [Page 460]

TABLESTATISTICS

Use

This table contains information about structure and size of base tables.

Structure

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

Integration

Statistics system tables [Page 460]

TRANSACTIONS

Use

This table contains information on active transactions in a serverdb.

Structure

TRANSACTIONS

SESSION	FIXED (10)	User 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

Integration

Statistics system tables [Page 460]

USERSTATISTICS

Use

This table contains information on the resources used by users.

Structure

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

Integration

Statistics system tables [Page 460]

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 counters for the internal events kept by the database system can be retrieved by selecting system tables. These tables are created by the SYSDBA during the installation. They produce results for users with DBA status. The error message `100 - ROW NOT FOUND` - is output for non-authorized users. You do not need to specify the owner [Page 93] to access the tables.

See also:

Monitor system tables [Page 476]

Monitor system tables

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 477]

MONITOR_LOAD [Page 480]

MONITOR_LOCK [Page 483]

MONITOR_LOG [Page 484]

MONITOR_PAGES [Page 485]

MONITOR_ROW [Page 487]

MONITOR_TRANS [Page 489]

MONITOR_VTRACE [Page 490]

The monitor system table MONITOR [Page 491] contains all the values in the listed monitor system tables.

See also:

MONITOR statement [Page 475]

MONITOR_CACHES

Use

This table contains information about the operations performed on the different caches.

Structure

Column DESCRIPTION

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

Integration

Monitor system tables [Page 476]

MONITOR_LOAD

Use

This table contains information on executed SQL statements and access methods.

Structure

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

Integration

Monitor system tables [Page 476]

MONITOR_LOCK

Use

This table contains information on the lock management operations performed by the database system.

Structure

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

Integration

Monitor system tables [Page 476]

MONITOR_LOG

Use

This table contains information on the logging operations (writing log information) performed by the database system.

Structure

DESCRIPTION column

Value	Explanation
`Log page physical reads`	Number of log pages physically read
`Log page physical writes`	Number of log pages physically written
`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

Integration

Monitor system tables [Page 476]

MONITOR_PAGES

Use

This table contains information on page accesses.

Structure

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

Integration

Monitor system tables [Page 476]

MONITOR_ROW

Use

This table contains information on operations at row level.

Structure

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

Integration

Monitor system tables [Page 476]

MONITOR_TRANS

Use

This table contains information on transactions.

Structure

DESCRIPTION column

Value	Explanation
`SQL commands`	Number of SQL statements
`Write transactions`	Number of transactions with modifying operations
`KB calls`	Number of KB requests (requests via the internal communications interface)

Integration

Monitor system tables [Page 476]

MONITOR_VTRACE

Use

This table contains information on the vtrace output.

Structure

DESCRIPTION column

Value	Explanation
`Trace I/O operations`	Number of vtrace output operations
`Trace I/O operations locked`	Number of delayed vtrace output operations

Integration

Monitor system tables [Page 476]

MONITOR

Use

This table is a combination of all other monitor system tables described under monitor system tables [Page 476].

Structure

MONITOR

TYPE	CHAR (8)
DESCRIPTION	CHAR (40)
VALUE	FIXED (20)

Integration

Monitor system tables [Page 476]

Restrictions

Maximum values

Identifier length	32 characters
Precision of numeric values	38 digits
Number of tables	unlimited
Internal length of a table row	8088 characters
Length of a LONG column	2147483647 characters
Columns per table (with KEY)	1024 columns
Columns per table (without KEY)	1023 columns
Number of key tables	512 columns
Sum of internal lengths of all key columns	1024 characters
Number of named indexes per table	256
Number of indexes, comprising more than one column, per table	256
Number of columns in an index	16
Sum of internal lengths of all columns belonging to an index	1024 characters
Number of referential CONSTRAINT definitions per base table	unlimited
Number of columns in a referential CONSTRAINT definition	16
Number of triggers per base table	3
Number of rows per table	unlimited
Number of result columns in a SELECT	1023 columns
Number of join tables in SELECT	64 tables
Number of join conditions in a WHERE condition of a SELECT statement	128
Number of correlated columns in an SQL statement	64
Number of correlated tables in an SQL statement	16
Number of columns in an ORDER or GROUP condition	128
Length of the columns in an ORDER or GROUP condition	1020 characters
Number of parameters in an SQL statement	2000 parameters

Length of an SQL statement (configuration parameter _PACKET_SIZE less administration overhead)	min. 16000