SAP HANA Database - SQL ĭ ′Y

SAP HANA Appliance Software SPS 04

-
- ь
- SAP Ð ĬU ƀ P ᵖ
- ì ĭ

目录

符号表示

本参考手册使用 BNF（巴科斯范式），用来定义编程语言，描述 SQL 的符号技术。BNF 描述的是使用一组利用一系列符号的产生式规则的语法。

< > 尖括号用来包含SQL语言的句法元素（BNF非终结符）的名字。

::= 定义符号提供出现在产生式左侧的元素的定义。

[ ] 方括号用来标示公式中的可选元素。可选元素可能是特定的或者省略的。

{ } 公式中的括号组元素。重复元素（零个或多个元素）可以在括号符号内指定。

| 替代符表示公式中的后半部分是前半部分的替代。

... 省略号表明元素可能重复出现任意次。如果省略号出现在组元素之后，表示用括号括起来的组元素重复；如果只出现在单个元素之后，只有该元素重复。

!! 引入普通的英语文本。语法元素的定义无法用BNF表达时使用。

引言

本章节描述了 SAP HANA 数据库的 SQL 语言实施，解释了 SQL 的特性，以及如何管理注释和保留字。

SQL

SQL 代表结构化查询语言，是标准化的连通关系数据库的语言。SQL 用来获取、存储和操作数据库中的信息。

SQL 语言执行以下任务：

- 集合（schema）的定义和操纵
- 操作数据
- 系统管理
- 会话管理
- 事务管理

支持的语言和代码页

SAP HANA 数据库支持 Unicode 允许使用 Unicode 标准中的所有语言，以及无限制的 7 位 ACSII 代码页。

注释

你可以给你的 SQL 语句添加注释来增加可读性和可维护性。SQL 语句中注释的分隔如下：

- 双连字符“-”。所有在双连字符之后直到行尾的内容都被 SQL 解析器认为是注释。
- “/*”和“*\/”。这种类型的注释用来注释多行内容。所有在引号符“/*”和关闭符“*\/”之间 的文字都会被 SQL 解析器忽略。

标识符

<标识符> ::= <简单标识符> | <双引号><特殊标识符><双引号>

<简单标识符> ::= <字母> [{<字母或数字>|<下划线>}, ...]

<双引号> ::= "

<特殊标识符> ::= 任意单词

<字母> ::= 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

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

<字母或数字> ::= <字母> | <数字>

<下划线> ::= _

标识符用来表示 SQL 语句中的名字，包括表名、视图名、同义字、列名、索引名、函数名、存储过程名、用户名、角色名等等。有两种类型的标识符：未定标识符和分隔标识符。

- 未分隔的表名和列名必须以字母开头，不能包含除数字或者下划线以外的符号。
- 分隔标识符用分隔符、双引号关闭，然后标识符可以包含任何字符包括特殊字符。例如，"AB$%CD" 是一个有效的标识符。
- 限制：
    - "_SYS_"专门为数据库引擎保留，因此不允许出现在集合对象的名字开头。
    - 角色名和用户名必须以未分隔符指定。
    - 标识符最大长度为 127 字母。

单引号

单引号是用来分隔字符串，使用两个单引号就可以代表单引号本身。

双引号

用双引号分隔标识符，使用两个双引号可以代表双引号本身。

SQL 保留字

保留字对于 SAP HANA 数据库的 SQL 解析器有着特殊含义，不能成为用户自定义的名字。保留字不能在 SQL 语句中使用为集合对象名。如果有必要，你可以使用双引号限定表或列名绕过这个限制。

下表列出了所有现在和未来 SAP HANA 数据库的保留字。

ALL	ALTER	AS	BEFORE
BEGIN	BOTH	CASE	CHAR
CONDITION	CONNECT	CROSS	CUBE
CURRENT_CONNECTION	CURRENT_DATE	CURRENT_SCHEMA	CURRENT_TIME
CURRENT_TIMESTAMP	CURRENT_USER	CURRENT_UTCDATE	CURRENT_UTCTIME
CURRENT_UTCTIMESTAMP	CURRVAL	CURSOR	DECLARE
DISTINCT	ELSE	ELSEIF	ELSIF
END	EXCEPT	EXCEPTION	EXEC
FOR	FROM	FULL	GROUP
HAVING	IF	IN	INNER
INOUT	INTERSECT	INTO	IS
JOIN	LEADING	LEFT	LIMIT
LOOP	MINUS	NATURAL	NEXTVAL
NULL	ON	ORDER	OUT
PRIOR	RETURN	RETURNS	REVERSE
RIGHT	ROLLUP	ROWID	SELECT
SET	SQL	START	SYSDATE
SYSTIME	SYSTIMESTAMP	SYSUUID	TOP
TRAILING	UNION	USING	UTCDATE
UTCTIME	UTCTIMESTAMP	VALUES	WHEN
WHERE	WHILE	WITH	

表 1.保留字

数据类型

本节描述了 SAP HANA 数据库中使用的数据类型。

数据类型指定一个数值的特性。一个特殊值 NULL 包含在每个数据类型中表示值的缺省。下表显示了 SAP HANA 数据库中提供的内置的数据类型。

数据类型的分类

SAP HANA 数据库中数据类型的特点可分为如下：

分类	数据类型
时间类型	DATE, TIME, SECONDDATE, TIMESTAMP
数字类型	TINYINT, SMALLINT, INTEGER, BIGINT, SMALLDECIMAL, DECIMAL, REAL, DOUBLE
字符串类型	VARCHAR, NVARCHAR, ALPHANUM, SHORTTEXT
二进制类型	VARBINARY
大对象类型	BLOB, CLOB, NCLOB, TEXT

表 2. 数据类型的分类

日期时间类型

- DATE
  DATE 数据类型由年、月、日信息组成，表示一个日期值。DATA 类型的默认格式为‘YYYY-MM-DD’。YYYY 表示年，MM 表示月而 DD 表示日。时间值的范围从 0001-01-01 至 9999-12-31。
- TIME
  TIME 数据类型由小时、分钟、秒信息组成，表示一个时间值。TIME 类型的默认格式为‘HH24:MI:SS’。HH24 表示从 0 至 24 的小时数，MI 代表 0 至 59 的分钟值而 SS 表示 0 至 59 的秒。
- SECONDDATE
  SECONDDATE 数据类型由年、月、日、小时、分钟和秒来表示一个日期和时间值。SECONDDATE 类型的默认格式为‘YYYY-MM-DD HH24:MI:SS’。YYYY 代表年，MM 代表月份，DD 代表日，HH24 表示小时，MI 表示分钟，SS 表示秒。日期值的范围从 0001-01-01 00:00:01 至 9999-12-31 24:00:00。
- TIMESTAMP
  TIMESTAMP 数据类型由日期和时间信息组成。其默认格式为‘YYYY-MM-DD HH24:MI:SS.FF7’。FFn 代表含有小数的秒，其中 n 表示小数部分的数字位数。时间戳的范围从 0001-01-01 00:00:00.0000000 至 9999-12-31 23:59:59.9999999。

有关日期时间类型支持的格式，请参阅以下的表 4、表 5、表 6 和表 7。

Date 格式

下面的日期/时间格式可以使用在把一个字符串解析为日期/时间类型以及把日期/时间类型解析成字符串时。请注意，时间戳的格式是日期和时间的结合以及对秒的小数部分的额外支持。

格式	描述	例子
YYYY-MM-DD	默认格式。	INSERT INTO TBL VALUES ('1957-06-13');
YYYY/MM/DD YYYY/MM-DD YYYY-MM/DD	YYYY从0001到9999，MM从1到12，DD从1至31。如果年份少于四位数，月份少于两位数，或者日期少于两位数，值会补充一个或多个零。例如，两位数年份45会保存为0045，一位数的月份9保存为09，一位数的日期2被保存为02。	INSERT INTO TBL VALUES ('1957-06-13'); INSERT INTO TBL VALUES ('1957/06/13'); INSERT INTO TBL VALUES ('1957/06-13'); INSERT INTO TBL VALUES ('1957-06/13');
YYYYMMDD	ABAP 数据类型, DATS格式	INSERT INTO TBL VALUES ('19570613');
MON	月份名字的缩写 (JAN. ~ DEC.)	INSERT INTO TBL VALUES (TO_DATE('2040-Jan-10', 'YYYY-MON-DD')); INSERT INTO TBL VALUES (TO_DATE('Jan-10', 'MON-DD'));
MONTH	月份名字. (JANUARY - DECEMBER).	INSERT INTO TBL VALUES (TO_DATE('2040-January-10', 'YYYY-MONTH-DD')); INSERT INTO TBL VALUES (TO_DATE('January-10', 'MONTH-DD'));
RM	月份罗马数字 (I-XII; JAN = I).	INSERT INTO TBL VALUES (TO_DATE('2040-I-10', 'YYYY-RM-DD')); INSERT INTO TBL VALUES (TO_DATE('I-10', 'RM-DD'));
DDD	一年中的一天(1-366).	INSERT INTO TBL VALUES (TO_DATE('204', 'DDD')); INSERT INTO TBL VALUES (TO_DATE('2001-204','YYYY-DDD'));

表 4：支持的日期格式

Time 格式

格式	描述	例子
HH24:MI:SS	默认格式	
HH:MI[:SS][AM|PM] HH12:MI[:SS][AM|PM] HH24:MI[:SS]	HH从0至23，MI从0至59，SS从0至59，FFF从0只59。 如果指定了一位数的小时、分钟、秒，则0将插入该值。例如，9:9:9将保存为09:09:09. HH12表示12小时制，HH24表示24小时制。 指定AM或PM作为前缀表示时间值为中午前或中午后。	INSERT INTO TBL VALUES ('23:59:59'); INSERT INTO TBL VALUES ('3:47:39 AM'); INSERT INTO TBL VALUES ('9:9:9 AM'); INSERT INTO TBL VALUES (TO_TIME('11:59:59','HH12:MI:SS');

表 5：支持的时间格式

Timestamp 格式

格式	描述	例子
YYYY-MM-DD HH24:MI:SS.FF7	默认格式	
FF [1..7]	秒的小数部分范围为1至7，FF参数指定了返回的时间值小数部分的位数。 如果未指定一个数值，则将使用默认值。 portion of the date time value returned. If a digit is not specified, the default value is used.	INSERT INTO TBL VALUES (TO_TIMESTAMP('2011-05-11 12:59.999','YYYY-MM-DD HH:SS.FF3'));

表 6：支持的时间戳格式

额外格式

格式	描述	例子
D	日期数	TO_CHAR(CURRENT_TIMESTAMP,'D')
DAY	日期名 (MONDAY - SUNDAY)	TO_CHAR(CURRENT_TIMESTAMP,'DAY')
DY	日期名的缩写(MON - SUN)	TO_CHAR(CURRENT_TIMESTAMP,'DY')
MON	月份名的缩写(JAN - DEC)	TO_CHAR(CURRENT_TIMESTAMP,'MON')
MONTH	月份的全名(JANUARY - DECEMBER)	TO_CHAR(CURRENT_TIMESTAMP,'MONTH')
RM	月份罗马数字(I - XII); I is for January)	TO_CHAR(CURRENT_TIMESTAMP,'RM')
Q	季度名(1, 2, 3, 4)	TO_CHAR(CURRENT_TIMESTAMP,'Q')
W	一个月中的星期(1-5)	TO_CHAR(CURRENT_TIMESTAMP,'W')
WW	一年中的星期 (1-53)	TO_CHAR(CURRENT_TIMESTAMP,'WW')

表 7：DateTime 额外格式

日期/时间类型支持的函数：

- **ADD_DAYS**
- **ADD_MONTHS**
- **ADD_SECONDS**
- **ADD_YEARS**
- **COALESCE**
- **CURRENT_DATE**
- **CURRENT_TIME**
- **CURRENT_TIMESTAMP**
- **CURRENT_UTCDATE**
- **CURRENT_UTCTIME**
- **CURRENT_UTCTIMESTAMP**
- **DAYNAME**
- **DAYOFMONTH**
- **DAYOFYEAR**
- **DAYS_BETWEEN**
- **EXTRACT**
- **GREATEST**
- **GREATEST**
- **HOUR**
- **IFNULL**
- **ISOWEEK**
- **LAST_DAY**
- **LEAST**
- **LOCALTOUTC**
- **MINUTE**
- **MONTH**
- **MONTHNAME**
- **NEXT_DAY**

- **NULLIF**
- **QUARTER**
- **SECOND**
- **SECONDS_BETWEEN**
- **TO_CHAR**
- **TO_DATE**
- **TO_DATS**
- **TO_NCHAR**
- **TO_TIME**
- **TO_TIMESTAMP**
- **UTCTOLOCAL**
- **WEEK**
- **WEEKDAY**
- **YEAR**

数字类型

- TINYINT
  TINYINT 数据类型存储一个 8 位无符号整数。TINYINT 的最小值是 0，最大值是 255。
- SMALLINT
  SMALLINT 数据类型存储一个 16 位无符号整数。SMALLINT 的最小值为-32,768 ，最大值为 32, 767。
- INTEGER
  INTEGER 数据类型存储一个 32 位有符号整数。INTEGER 的最小值为-2,147,483,648 ，最大值为 2,147,483,647。
- BIGINT
  BIGINT 数据类型存储一个 64 位有符号整数。INTEGER 的最小值为-9,223,372,036,854,775,808，最大值为 9, 223,372,036,854,775,807。
- DECIMAL（精度，小数位数）或 DEC（p, s）
  DECIMAL (p, s) 数据类型指定了一个精度为 p 小数位数为 s 的定点小数。精度是有效位数的总数，范围从 1 至 34。
  小数位数是从小数点到最小有效数字的数字个数，范围从-6,111 到 6,176，这表示位数指定了十进制小数的指数范围从 10^{-6111} 至 10^{6176}。如果没有指定小数位数，则默认值为 0。
  当数字的有效数字在小数点的右侧时，小数位数为正；有效数字在小数点左侧时，小数位数为负。
  例子：0.0000001234 (1234 x 10-10) 精度为 4，小数位数 10。 1.0000001234 (10000001234 x 10-10) 精度为 11，小数位数为 10。 1234000000 (1234x106) 精度为 4，小数位数为-6。
  当未指定精度和小数位数，DECIMAL 成为浮点小数。这种情况下，精度和小数位数可以在上文描述的范围内不同，根据存储的数值，1-34 的精度和 6111-6176 的小数位数。
- SMALLDECIMAL

SMALLDECIMAL 是一个浮点十进制数。精度和小数位数可以在范围有所不同，根据存储的数值，1-16 的精度以及-369-368 的小数位数。SMALLDECIMAL 只支持列式存储。
DECIMAL 和 SMALLDECIMAL 都是浮点十进制数。举例来说，一个十进制列可以存储 3.14，3.1415，3.141592 同时保持它们的精度。
DECIMAL（p, s）是 SQL 对于定点十进制数的标准标记。例如，3.14, 3.1415,3.141592 存储在 decimal(5, 4)列中为 3.1400, 3.1415, 3.1416，各自保持其精度（5）和小数位数（4）。

- REAL
  REAL 数据类型定义一个 32 位单精度浮点数。
- DOUBLE
  DOUBLE 数据类型定义一个 64 位的单精度浮点数，最小值为-1.79769 x 10308，最大值为 1.79769x10308，DOUBLE 最小的正数为 2.2207x10-308，最大的负数为-2.2207x10-308。
- FLOAT(n)
  FLOAT 数据类型定义一个 32 位或 64 位的实数，n 指定有效数字的个数，范围可以从 1 至 53。
  当你使用 FLOAT(n)数据类型时，如果 n 比 25 小，其会变成 32 位的实数类型；如果 n 大于等于 25，则会成为 64 的 DOUBLE 数据类型。如果 n 没有声明，默认变成 64 位的 double 数据类型。

字符类型

字符类型用来存储包含字符串的值。VARCHAR 类型包含 ASCII 字符串，而 NVARCHAR 用来存储 Unicode 字符串。

- VARCHAR
  VARCHAR (n) 数据类型定义了一个可变长度的 ASCII 字符串，n 表示最大长度，是一个 1 至 5000 的整数值。
- NVARCHAR
  NVARCHAR (n) 数据类型定义了一个可变长度的 Unicode 字符串，n 表示最大长度，是一个 1 至 5000 的整数值。
- ALPHANUM
  ALPHANUM (n) 数据类型定义了一个可变长度的包含字母数字的字符串，n 表示最大长度，是一个 1 至 127 的整数值。
- SHORTTEXT
  SHORTTEXT (n) 数据类型定义了一个可变长度的字符串，支持文本搜索和字符搜索功能。这不是一个标准的 SQL 类型。选择一列 SHORTTEXT (n) 列会生成一个 NVARCHAR (n)类型的列。
  <shorttext_type> ::= SHORTTEXT '(' int_const ')' <elem_list_shorttext>
  <elem_list_shorttext> ::= <elem_shorttext> [... ',' <elem_shorttext>]
  <elem_shorttext> ::= <fulltext_elem> | SYNC[HRONOUS]

二进制类型

二进制类型用来存储二进制数据的字节。

- VARBINARY
  VARBINARY 数据类型用来存储指定最大长度的二进制数据，以字节为单位，n 代表最大长度，是一个 1 至 5000 的整数。

大对象（LOB）类型

LOB（大对象）数据类型，CLOB，NCLOB 和 BLOB，用来存储大量的数据例如文本文件和图像。一个 LOB 的最大大小为 2GB。

- BLOB
  BLOB 数据类型用来存储大二进制数据。
- CLOB
  CLOB 数据类型用来存储大 ASCII 字符数据。
- NCLOB
  NCLOB 数据类型用来存储大 Unicode 字符对象。
- TEXT
  TEXT 数据类型指定支持文本搜索功能，这不是一个独立的 SQL 类型。选择一列 TEXT 列会生成一个 NCLOB 类型的列。

  <text_type> ::= TEXT <opt_fulltext_elem_list_text>
  <opt_fulltext_elem_list_text> ::=
  <fulltext_elem_text> [... ',' <fulltext_elem_text>]
  <fulltext_elem_text> ::= <fulltext_elem>
  | [SYNC[HRONOUS]
  | [ASYNC[HRONOUS] FLUSH [QUEUE]
  EVERY <n> MINUTES [[OR] AFTER <m> DOCUMENTS]]

TEXT 和 SHORTTEXT 共同语法规则

<fulltext_elem> ::= LANGUAGE COLUMN <column_name>

| LANGUAGE DETECTION '(' <str_const_list> ')'

| MIME TYPE COLUMN <column_name>

| FUZZY SEARCH INDEX [ON|OFF]

| PHRASE INDEX RATIO [ON|OFF]

| CONFIGURATION <str_const>

| SEARCH ONLY [ON|OFF]

| FAST PREPROCESS [ON|OFF]

LOB 类型用于存储和检索大量的数据。LOB 类型支持以下操作：

- Length(n)以字节形式返回 LOB 的长度。
- LIKE 可以用来搜索 LOB 列。

LOB 类型有如下限制：

- LOB 列不能出现在 ORDER BY 或 GROUP BY 子句中。
- LOB 列不能出现在 FROM 子句作为联接谓词。
- 不能作为谓词出现在 WHERE 子句中，除了 LIKE，CONTAINS，=或<>。
- LOB 列不能出现在 SELECT 子句作为一个聚合函数的参数。
- LOB 列不能出现在 SELECT DISTINCT 语句中。
- LOB 列不能用于集合操作，除了 EXCEPT， UNION ALL 是个例外。
- LOB 列不能作为主键。
- LOB 列不能使用 CREATE INDEX 语句。
- LOB 列不能使用统计信息更新语句。

SQL 数据类型和列存储数据类型之间的映射

	SQL Type	Column Store Type
Integer Types	TINYINT, SMALLINT, INT	CS_INT
	BIGINT	CS_FIXED(18,0)
Approximate Types	REAL	CS_FLOAT
	DOUBLE	CS_DOUBLE
	FLOAT	CS_DOUBLE
	FLOAT(p)	CS_FLOAT, CS_DOUBLE
Decimal Types	DECIMAL	CS_DECIMAL_FLOAT
	DECIMAL(p,s)	CS_FIXED(p-s,s)
	SMALLDECIMAL	CS_SDFLOAT
Character Types	VARCHAR	**CS_STRING**,CS_ALPHANUM,CS_UNITDECFLOAT,CS_DATE,CS_TIME
	NVARCHAR	**CS_STRING**,CS_ALPHANUM,CS_UNITDECFLOAT
	CLOB, NCLOB	CS_STRING
	ALPHANUM	CS_ALPHANUM
Binary Types	BLOB	CS_RAW
	VARBINARY	CS_RAW
Date/Time Types	DATE	**CS_DAYDATE**,CS_DATE
	TIME	**CS_SECONDTIME**,CS_TIME
	TIMESTAMP	**CS_LONGDATE**,CS_DATE,CS_SECONDDATE
	SECONDDATE	CS_SECONDDATE

数据类型转换

本节描述 SAP HANA 数据库中允许的类型转换。

- 显式类型转换
  表达式结果的类型，例如一个字段索引，一个字段函数或者文字可以使用如下函数进行转换：CAST, TO_ALPHANUM, TO_BIGINT, TO_VARBINARY, TO_BLOB, TO_CLOB, TO_DATE, TO_DATS, TO_DECIMAL, TO_DOUBLE, TO_INTEGER, TO_INT, TO_NCLOB, TO_NVARCHAR, TO_REAL, TO_SECONDDATE, TO_SMALLINT, TO_TINYINT, TO_TIME, TO_TIMESTAMP, TO_VARCHAR。
- 隐式类型转换
  当给定的一系列运算符/参数类型不符合其所期望的类型，SAP HANA 数据库就会执行类型转换。这种转换仅仅发生在相关的转换可供使用，并且使得运算符/参数类型可执行。举例来说，BIGINT 和 VARCHAR 之间的比较是通过把 VARCHAR 隐式转换成 BIGINT 进行的。显式转换可以全部用于隐式转换，除了 TIME 和 TIMESTAMP 数据类型。TIME 和 TIMESTAMP 可以使用 TO_TIME(TIMESTAMP)以及 TO_TIMESTAMP(TIME)相互转换。
- 例子

Input Expression	Transformed Expression with Implicit Conversion
BIGINT > VARCHAR	BIGINT > BIGINT(VARCHAR)
BIGINT > DECIMAL	DECIMAL(BIGINT) > DECIMAL
TIMESTAMP > DATE	TIMESTAMP > TIMESTAMP(DATE)
DATE > TIME	Error because there is no conversion available between DATE and TIME

表 8： 隐式类型转换例子

在下表中：

- 方框中“OK”表示允许的数据类型转换，没有任何检查。
- 方框中”CHK”表示数据类型转换只有在数据是有效的目标类型时才执行。
- 方框中”-”表示不允许该数据类型转换。

如下显示的规则同时适用于隐式和显示转换，除了 TIME 至 TIMESTAMP 的转换。TIME 类型只能通过显示转换 TO_TIMESTAMP 或者 CAST 函数执行。

Target/ Source	tinyint	smallint	integer	bigint	decimal	decimal(p,s)	smalldecimal	real	double	varchar	nvarchar
tinyint	-	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
smallint	CHK	-	OK	OK	OK	OK	OK	OK	OK	OK	OK
integer	CHK	CHK	-	OK	OK	OK	OK	OK	OK	OK	OK
bigint	CHK	CHK	CHK	-	OK	CHK	CHK	CHK	OK	OK	OK
decimal	CHK	CHK	CHK	CHK	-	CHK	CHK	CHK	OK	OK	OK
decimal(p,s)	CHK	CHK	CHK	CHK	CHK	CHK	CHK	CHK	CHK	CHK	OK
smalldecimal	CHK	CHK	CHK	CHK	OK	CHK	-	CHK	CHK	OK	OK
real	CHK	CHK	CHK	CHK	OK	CHK	CHK	-	OK	OK	OK
double	CHK	CHK	CHK	CHK	CHK	CHK	CHK	CHK	-	OK	OK
varchar	CHK	CHK	CHK	CHK	CHK	CHK	CHK	CHK	CHK	-	OK
nvarchar	CHK	CHK	CHK	CHK	CHK	CHK	CHK	CHK	CHK	CHK	-

表 9a：数据类型转换表

Target/ Source	time	date	seconddate	timestamp	varchar	nvarchar
time	-	-	-	-	OK	OK
date	-	-	OK	OK	OK	OK
seconddate	time	date	-	timestamp	OK	OK
timestamp	time	date	seconddate	-	OK	OK
varchar	CHK	CHK	CHK	CHK	-	OK
nvarchar	CHK	CHK	CHK	CHK	CHK	-

表 9b：数据类型转换表

Target/ Source	varbinary	alphanum	varchar	nvarchar
varbinary	-	-	-	-
alphanum	-	-	OK	OK
varchar	OK	OK	-	OK
nvarchar	OK	OK	CHK	-

表 9c：数据类型转换表

数据类型优先顺序

本节介绍 SAP HANA 数据库实施的数据类型的优先级。数据类型优先级指定较低优先级的类型转换为较高优先级的类型。

Highest	TIMESTAMP
	SECONDDATE
	DATE
	TIME
	DOUBLE
	REAL
	DECIMAL
	SMALLDECIMAL
	BIGINT
	INTEGER
	SMALLINT
	TINYINT
	NCLOB
	NVARCHAR
	CLOB
	VARCHAR
	BLOB
Lowest	VARBINARY

类型常量

常量是表示一个特定的固定数值的符号。

- 字符串常量
  字符串常量括在单引号中。
  - 'Brian'
  - '100'

Unicode 字符串的格式与字符串相似，但前面有一个 N 标识符（N 代表 SQL-92 标准中的国际语言）。N 前缀必须是大写。

  - N'abc'

```
SELECT 'Brian' "character string 1", '100' "character string 2", N'abc' "unicode st
ring" FROM DUMMY;

character string 1, character string 2, unicode string
Brian, 100, abc
```

- 数字常量
  数字常量用没有括在单引号中的数字字符串表示。数字可能包含小数点或者科学计数。
  - 123
  - 123.4
  - 1.234e2

十六进制数字常量是十六进制数的字符串，含有前缀 0x。

  - 0x0abc

```
SELECT 123 "integer", 123.4 "decimal1", 1.234e2 "decimal2", 0x0abc "hexadecimal" FR
OM DUMMY;

integer, decimal1, decimal2, hexadecimal
123, 123.4, 123.4, 2748
```

- 二进制字符串常量
  二进制字符串有前缀 X，是一个括在单引号中的十六进制数字字符串。
  - X'00abcd'
  - x'dcba00'

```
SELECT X'00abcd' "binary string 1", x'dcba00' "binary string 2" FROM DUMMY;

binary string 1, binary string 2
00ABCD, DCBA00
```

- 日期、时间、时间戳常量
  日期、时间、时间戳各自有如下前缀：
  - date'2010-01-01'
  - time'11:00:00.001'
  - timestamp'2011-12-31 23:59:59'

```
SELECT date'2010-01-01' "date", time'11:00:00.001' "time", timestamp'2011-12-31 23:
59:59' "timestamp" FROM DUMMY;
```

```
date, time, timestamp
2010-01-01, 11:00:00, 2011-12-31 23:59:59.0
```

谓词

谓词由组合的一个或多个表达式，或者逻辑运算符指定，并返回以下逻辑/真值中的一个：TRUE、FALSE、或者 UNKNOW。

比较谓词

两个值使用比较谓词进行比较，并返回 TRUE，FALSE 或 UNKNOW。

语法：

<comparison_predicate> ::=

<expression> { = | != | <> | > | < | >= | <= } [ANY | SOME| ALL] { <expression_li

st> | <subquery> }

<expression_list> ::= <expression>, ...

表达式可以是简单的表达式如字符、日期或者数字，也可以是标量子查询。

ANY，SOME – 当指定了 ANY 或者 SOME 关键字，如果子查询返回至少一个结果或者 expression_list 为真，比较操作返回真。ALL - 当指定了 ALL 时，如果子查询返回了所有结果或者 expression_list 为真，比较操作返回真。

范围谓词

值将在给定范围中进行比较。

语法：

<range_predicate> ::= <expression1> [NOT] BETWEEN <expression2> AND <expression3>

BETWEEN …AND … - 当指定了范围谓词时，如果 expression1 在 expression2 和 expression3 指定的范围内时，结果返回真；如果 expression2 比 expression3 小，则只返回真。

In 谓词

一个值与一组指定的值比较。如果 expression1 的值在 expression_list（或子查询）中，结果返回真。

语法：

<in_predicate> ::= <expression> [NOT] IN { <expression_list> | <subquery> }

Exists 谓词

如果子查询返回非空结果集，结果为真；返回空结果集，结果则为假。

Like 谓词

Like 用来比较字符串，Expression1 与包含在 expression2 中的模式比较。通配符（%）和（_）可以用在比较字符串 expression2 中。

NULL 谓词

当指定了谓词 IS NULL，值可以与 NULL 比较。如果表达式值为 NULL， 则 IS NULL 返回值为真；如果指定了谓词 IS NOT NULL，值不为 NULL 时返回值为真。

语法：

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

CONTAINS 谓词

CONTAINS 谓词用来搜索子查询中文本匹配的字符串。

语法：

<contains_function> ::= CONTAINS '(' <contains_columns> ',' <search_string>')'

| CONTAINS '(' <contains_columns> ',' <search_string> ',' <se

arch_specifier> ')'

<contains_columns> ::= '*' | <column_name> | '(' <columnlist> ')'

<search_string> ::= <string_const>

<search_specifier> ::= <search_type> <opt_search_specifier2_list>

| <search_specifier2_list>

<opt_search_specifier2_list> ::= empty

| <search_specifier2_list>

<search_type> ::= <exact_search> | <fuzzy_search> | <linguistic_search>

<search_specifier2_list> ::= <search_specifier2>

| <search_specifier2_list> ',' <search_specifier2>

<search_specifier2> := <weights> | <language>

<exact_search> ::= EXACT

<fuzzy_search> ::= FUZZY

| FUZZY '(' <float_const> ')'

| FUZZY '(' <float_const> ',' <additional_params> ')'

<linguistic_search> ::= LINGUISTIC

<weights> ::= WEIGHT '(' <float_const_list> ')'

<language> :: LANGUAGE '(' <string_const> ')'

<additional_params> ::= <string_const>

search_string

使用自由式字符串搜索格式（例如，Peter "Palo Alto"或 Berlin -"SAP LABS"）。

search_specifier

如果没有指定 search_specifier，EXACT 为默认值。

EXACT

对于那些在 search_attributes 中精确匹配 searchterms 的记录，EXACT 返回真。

FUZZY

对于那些在 search_attributes 相似匹配 searchterms 的记录，FUZZY 返回真(例如，以某种程度忽略拼写错误)。

Float_const

如果省略 float_const，则默认值为 0.8。可以通过定义列式存储联接视图支持的参数 FUZZINESSTHRESHOLD 来覆盖默认值。

WEIGHT

如果定义了 weights 列表，则必须与<contains_columns>中的列数量一样。

LANGUAGE

LANGUAGE 在搜索字符串的预处理中使用，并且作为搜索前的过滤。只返回匹配搜索字符串的文档和定义的语言。

LINGUISTIC

对于那些在 searchattribute 中出现的 searchterms 字符变量，LINGUISTIC 返回真。

限制：如果在 where 条件中定义了多个 CONTAINS，那么只有其中的一个由<contains_columns>列表中的不止一列组成。

CONTAINS 只对列式存储表适用（简单表和联接视图）。

例子:

精确搜索：

select * from T where contains(column1, 'dog OR cat') -- EXACT is implicit select

* from T where contains(column1, 'dog OR cat', EXACT)

select * from T where contains(column1, '"cats and dogs"') -- phrase search

模糊搜索：

select * from T where contains(column1, 'catz', FUZZY(0.8))

语言搜索

select * from T where contains(column1, 'catz', LINGUISTIC)

自由式搜索：

自由式搜索是对于多列的搜索。

select * from T where CONTAINS((column1,column2,column3), 'cats OR dogz', FUZZY(0.

7))

select * from T where CONTAINS((column1,column2,column3), 'cats OR dogz', FUZZY(0.

7))

操作符

你可以在表达式中使用操作符进行算术运算。操作符可以用来计算、比较值或者赋值。

一元和二元操作符

操作符	操作	格式	描述
一元	一元操作符适用于单个操作数或者单值表达式。 s	operator operand	unary plus operator(+) unary negation operator(-) logical negation(NOT)
二元	二元表达式适用于两个操作数或者两个值的表达式。	operand1 operator operand2	multiplicative operators (*, /) additive operators (+,-) comparison operators (=,!=,<,>,<=,>=) logical operators (AND, OR)

表 10. 一元和二元操作符

操作符优先级

一个表达式可以使用多个操作符。如果操作符大于一个，则 SAP HANA 数据库会根据操作符优先级评估它们。你可以通过使用括号改变顺序，因为在括号内的表达式会第一个评估。

如果没有使用括号，则操作符优先级将根据下表。请注意，SAP HANA 数据库对于优先级相同的操作符将从左至右评估操作符。

优先级	操作符	操作
Highest	()	parentheses
	+, -	unary positive and negative operation
	*, /	multiplication, division
	+, -	addition, subtraction
	||	concatenation
	=, !=, <, >, <=, >=, IS NULL, LIKE, BETWEEN	comparsion
	NOT	logical negation
	AND	conjunction
Lowest	OR	disjunction

算术操作符

你可以使用算术操作符来执行数学运算，如加法、减法、乘法和除法，以及负数。

操作符	描述
-<expression>	取反。如果表达式为NULL, 则结果也为NULL。
<expression> + <expression>	加法。如果其中一表达式为NULL，则结果也为NULL。
<expression> - <expression>	减法。如果其中一表达式为NULL，则结果也为NULL。
<expression> * <expression>	乘法。如果其中一表达式为NULL，则结果也为NULL。
<expression> / <expression>	除法。如果其中一表达式为NULL，或者第二个表达式为0，则返回错误。

表 12. 算术操作符

字符串操作符

级联操作符结合两项类似字符串、表达式或者常量到一项中。

操作符	描述
<expression> || <expression>	级联字符串。如果其中一字符串为NULL，则返回NULL

表 13. 级联操作符

对于 VARCHAR 或者 NVARCHAR 类型字符串，前导或者后置空格将保留。如果其中一字符串类型为 NVARCHAR，则结果也为 NVARCHAR 并且限制在 5000 个字母，VARCHAR 联接的最大长度也限制在 5000 个字母。

比较操作符

语法：

<comparison_operation> ::= <expression1> <comparison_operator> <expression2>

操作符	描述	例子
=	等于	SELECT * FROM students WHERE id = 25;
>	大于	SELECT * FROM students WHERE id > 25;
<	小于	SELECT * FROM students WHERE id < 25;
>=	大于等于	SELECT * FROM students WHERE id >= 25;
<=	小于等于	SELECT * FROM students WHERE id <= 25;
!=, <>	不等于	SELECT * FROM students WHERE id != 25; SELECT * FROM students WHERE id <> 25;

表 14. 比较操作符

逻辑操作符

搜索条件可以使用 AND 或者 OR 操作符结合，你也可以使用 NOT 操作符否定条件。

操作符	语法	描述
AND	WHERE condition1 AND condition2	When using AND, the combined condition is TRUE if both conditions are TRUE, FALSE if either condition is FALSE, and UNKNOWN otherwise.
OR	WHERE condition1 OR condition2	When using OR, the combined condition is TRUE if either condition is TRUE, FALSE if both conditions are FALSE, and UNKNOWN otherwise.
NOT	WHERE NOT condition	The NOT operator is placed before a condition to negate the condition. The NOT condition is TRUE if condition is FALSE, FALSE if condition is TRUE, and UNKNOWN if condition is UNKNOWN.

表 15：逻辑操作符

集合操作符

本节中所述的操作符对两个或更多个查询的结果执行集合操作。

操作符	返回值
UNION	Combines the results of two or more select statements or query expressions
UNION ALL	Combines the results of two or more select statements or query expressions, including all duplicate rows.
INTERSECT	Combines the results of two or more select statements or query expressions, and returns all common rows.
EXCEPT	Takes output from the first query and then removes rows selected by the second query.

表 16：集合操作符

表达式

表达式是可以用来计算并返回值的子句。

语法：

<expression> ::=

<case_expression>
| <function_expression>
| <aggregate_expression>
| (<expression>)
| (<subquery>)
| - <expression>
| <expression> <operator> <expression>
| <variable_name>
| <constant>
| [<correlation_name>.]<column_name>

Case 表达式

Case 表达式允许用户使用 IF ... THEN ... ELSE 逻辑，而不用在 SQL 语句中调用存储过程。

语法：

<case_expression> ::=

CASE <expression>
WHEN <expression> THEN <expression>, ...
[ELSE <expression>]
{ END | END CASE }

如果位于 CASE 语句后面的表达式和 WHEN 后面的表达式相等，则 THEN 之后的表达式将作为返回值；否则返回 ELSE 语句之后的表达式，如果存在的话。

Function 表达式

SQL 内置的函数可以作为表达式使用。

语法：

<function_expression> ::= <function_name> (<expression>, ...)

Aggregate 表达式

Aggregate 表达式利用 aggregate 函数计算同一列中多行值。

语法:

<aggregate_expression> ::= COUNT(*) | <agg_name> ([ALL | DISTINCT] <expression>

)

<agg_name> ::= COUNT | MIN | MAX | SUM | AVG | STDDEV | VAR

聚合名	描述
COUNT	Counts the number of rows returned by a query. COUNT(*) returns the number of rows, regardless of the value of those rows and including duplicate values. COUNT(<expression>) returns the number of non-NULL values for that expression returned by the query.
MIN	Returns the minimum value of expression.
MAX	Returns the maximum value of expression.
SUM	Returns the sum of expression.
AVG	Returns the arithmetical mean of expression.
STDDEV	Returns the standard deviation of given expression as the square root of VARIANCE function.
VAR	Returns the variance of expression as the square of standard deviation.

表达式中的子查询

子查询是在括号中的 SELECT 语句。SELECT 语句可以包含一个，有且仅有一个选择项。当作为表达式使用时，标量子查询允许返回零个或一个值。

语法：

<scalar_subquery_expression> ::= (<subquery>)

在最高级别的 SELECT 中的 SELECT 列表，或者 UPDATE 语句中的 SET 条件中，你可以在任何能使用列名的地方使用标量子查询。不过，scalar_subquery 不可以在 GROUP BY 条件中使用。

例子：

以下语句返回每个部门中的员工数，根据部门名字分组：

```
SELECT DepartmentName, COUNT(*), 'out of',
(SELECT COUNT(*) FROM Employees)
FROM Departments AS D, Employees AS E
WHERE D.DepartmentID = E.DepartmentID
GROUP BY DepartmentName;
```

SQL 函数

介绍

本章将介绍 SAP HANA 数据库提供的 SQL 函数。

- **Data Type Conversion Functions**
- **DateTime Functions**
- **Number Functions**
- **String Functions**
- **Miscellaneous Functions**

数据类型转换函数

数据类型转换函数用来把参数从一个数据类型转换为另一个数据类型，或者测试转换是否可行。

CAST

语法：

CAST (expression AS data_type)

语法元素：

Expression – 被转换的表达式。Data type – 目标数据类型。TINYINT | SMALLINT |

INTEGER | BIGINT | DECIMAL | SMALLDECIMAL | REAL | DOUBLE | ALPHANUM | VARCHAR | NVARCHAR | DAYDATE | DATE | TIME | SECONDDATE | TIMESTAMP。

描述：

返回转换为提供的数据类型的表达式。

例子：

SELECT CAST (7 AS VARCHAR) "cast" FROM DUMMY;

Cast
7

TO_ALPHANUM

语法：

TO_ALPHANUM (value)

描述：

将给定的 value 转换为 ALPHANUM 数据类型。

例子：

SELECT TO_ALPHANUM ('10') "to alphanum" FROM DUMMY;

to alphanum
10

TO_BIGINT
语法：

TO_BIGINT (value)

描述：

将 value 转换为 BIGINT 类型。

例子：

SELECT TO_BIGINT ('10') "to bigint" FROM DUMMY;

to bigint
10

TO_BINARY
语法：

TO_BINARY (value)

描述：

将 value 转换为 BINARY 类型。

例子：

SELECT TO_BINARY ('abc') "to binary" FROM DUMMY;

to binary
616263

TO_BLOB
语法：

TO_BLOB (value)

描述：

将 value 转换为 BLOB 类型。参数值必须是二进制字符串。

例子：

SELECT TO_BLOB (TO_BINARY('abcde')) "to blob" FROM DUMMY;

to blob
abcde

TO_CHAR
语法：

TO_CHAR (value [, format])

描述：

将 value 转换为 CHAR 类型。如果省略 format 关键字，转换将会使用 Date Formats 中说明的日期格式模型。

例子：

SELECT TO_CHAR (TO_DATE('2009-12-31'), 'YYYY/MM/DD') "to char" FROM DUMMY;

to char
2009/12/31

TO_CLOB
语法：

TO_CLOB (value)

描述:

将 value 转换为 CLOB 类型。

例子：

SELECT TO_CLOB ('TO_CLOB converts the value to a CLOB data type') "to clob" FROM DUMMY;

to clob
TO_CLOB converts the value to a CLOB data type

TO_DATE
语法：

TO_DATE (d [, format])

描述：

将日期字符串 d 转换为 DATE 数据类型。如果省略 format 关键字，转换将会使用 Date Formats 中说明的日期格式模型。

例子：

SELECT TO_DATE('2010-01-12', 'YYYY-MM-DD') "to date" FROM DUMMY;

to date
2010-01-12

TO_DATS
语法：

TO_DATS (d)

描述：

将字符串 d 转换为 ABAP 日期字符串，格式为”YYYYMMDD”。

例子：

SELECT TO_DATS ('2010-01-12') "abap date" FROM DUMMY;

abap date
20100112

TO_DECIMAL
语法：

TO_DECIMAL (value [, precision, scale])

描述：

将 value 转换为 DECIMAL 类型。

精度是有效数字的总数，范围为 1 至 34。小数位数是从小数点到最小有效数字的数字个数，范围从-6,111 到 6,176，这表示位数指定了十进制小数的指数范围从 10-6111 至 106176。如果没有指定小数位数，则默认值为 0。当数字的有效数字在小数点的右侧时，小数位数为正；有效数字在小数点左侧时，小数位数为负。

当未指定精度和小数位数，DECIMAL 成为浮点小数。这种情况下，精度和小数位数可以在上文描述的范围内不同，根据存储的数值，精度为 1-34 以及小数位数为 6111-6176。

例子：

SELECT TO_DECIMAL(7654321.888888, 10, 3) "to decimal" FROM DUMMY

to decimal
7654321.889

TO_DOUBLE
语法：

TO_DOUBLE (value)

描述：

将 value 转换为 DOUBLE（双精度）数据类型。

例子：

SELECT 3*TO_DOUBLE ('15.12') "to double" FROM DUMMY;

to double
45.36

TO_INT
语法：

TO_INT (value)

描述：

将 value 转换为 INTEGER 类型。

例子：

SELECT TO_INT('10') "to int" FROM DUMMY;

to int
10

TO_INTEGER
语法：

TO_INTEGER (value)

描述：

将 value 转换为 INTEGER 类型。

例子：

SELECT TO_INT('10') "to int" FROM DUMMY;

to int
10

TO_NCHAR
语法：

TO_NCHAR (value [, format])

描述：

将 value 转换为 NCHAR Unicode 字符类型。如果省略 format 关键字，转换将会使用 Date Formats 中说明的日期格式模型。

例子：

SELECT TO_NCHAR (TO_DATE('2009-12-31'), 'YYYY/MM/DD') "to nchar" FROM DUMMY;

to nchar
2009/12/31

TO_NCLOB
语法：

TO_NCLOB (value)

描述：

将 value 转换为 NCLOB 数据类型。

例子：

SELECT TO_NCLOB ('TO_NCLOB converts the value to a NCLOB data type') "to nclob" FROM DUMMY;

to nclob
TO_NCLOB converts the value to a NCLOB data type

TO_NVARCHAR
语法：

TO_NVARCHAR (value [,format])

描述：

将 value 转换为 NVARCHAR Unicode 字符类型。如果省略 format 关键字，转换将会使用 Date Formats 中说明的日期格式模型。

例子：

SELECT TO_NVARCHAR(TO_DATE('2009/12/31'), 'YY-MM-DD') "to nchar" FROM DUMMY;

to nchar
09-12-31

TO_REAL
语法：

TO_REAL (value)

描述：

将 value 转换为实数（单精度）数据类型。

例子：
SELECT 3*TO_REAL ('15.12') "to real" FROM DUMMY;

to real
45.36000061035156

TO_SECONDDATE
语法：

TO_SECONDDATE (d [, format])

描述：

将 value 转换为 SECONDDATE 类型。如果省略 format 关键字，转换将会使用 Date Formats 中说明的日期格式模型。

例子：

SELECT TO_SECONDDATE ('2010-01-11 13:30:00', 'YYYY-MM-DD HH24:MI:SS') "to seconddate" FROM DUMMY;

to seconddate
2010-01-11 13:30:00.0

TO_SMALLDECIMAL
语法：

TO_SMALLDECIMAL (value)

描述：

将 value 转换为 SMALLDECIMAL 类型。

例子：

SELECT TO_SMALLDECIMAL(7654321.89) "to smalldecimal" FROM DUMMY;

to smalldecimal
7654321.89

TO_SMALLINT
语法：

TO_SMALLINT (value)

描述：

将 value 转换为 SMALLINT 类型。

例子：

SELECT TO_SMALLINT ('10') "to smallint" FROM DUMMY;

to smallint
10

TO_TIME
语法：
TO_TIME (t [, format])

描述：

将时间字符串 t 转换为 TIME 类型。如果省略 format 关键字，转换将会使用 Date Formats 中说明的日期格式模型。

例子：

SELECT TO_TIME ('08:30 AM', 'HH:MI AM') "to time" FROM DUMMY;

to time
08:30:00

TO_TIMESTAMP
语法：

TO_TIMESTAMP (d [, format])

描述：

将时间字符串 t 转换为 TIMESTAMP 类型。如果省略 format 关键字，转换将会使用 Date Formats 中说明的日期格式模型。

例子：

SELECT TO_TIMESTAMP ('2010-01-11 13:30:00', 'YYYY-MM-DD HH24:MI:SS') "to timestamp"

FROM DUMMY;

to timestamp
2010-01-11 13:30:00.0

TO_TINYINT
语法：

TO_TINYINT (value)

描述：

将 value 转换为 TINYINT 类型。

例子:

SELECT TO_TINYINT ('10') "to tinyint" FROM DUMMY;

to tinyint
10

TO_VARCHAR
语法：

TO_VARCHAR (value [, format])

描述:

将给定 value 转换为 VARCHAR 字符串类型。如果省略 format 关键字，转换将会使用 Date Formats 中说明的日期格式模型。

例子：

SELECT TO_VARCHAR (TO_DATE('2009-12-31'), 'YYYY/MM/DD') "to char" FROM DUMMY;

to char
2009/12/31

日期时间函数

ADD_DAYS
语法:

ADD_DAYS (d, n)

描述：

计算日期 d 后 n 天的值。

例子：

SELECT ADD_DAYS (TO_DATE ('2009-12-05', 'YYYY-MM-DD'), 30) "add days" FROM DUMMY;

add days
2010-01-04

ADD_MONTHS
语法：

ADD_MONTHS (d, n)

描述：

计算日期 d 后 n 月的值。

例子：

SELECT ADD_MONTHS (TO_DATE ('2009-12-05', 'YYYY-MM-DD'), 1) "add months" FROM DUMMY

;

add months
2010-01-05

ADD_SECONDS
语法：

ADD_SECONDS (t, n)

描述：

计算时间 t 后 n 秒的值。

例子：

SELECT ADD_SECONDS (TO_TIMESTAMP ('2012-01-01 23:30:45'), 60*30) "add seconds" FROM

DUMMY;

add seconds
2012-01-02 00:00:45.0

ADD_YEARS
语法：

ADD_YEARS (d, n)

描述：

计算日期 d 后 n 年的值。

例子：

SELECT ADD_YEARS (TO_DATE ('2009-12-05', 'YYYY-MM-DD'), 1) "add years" FROM DUMMY;

add years
2010-12-05

CURRENT_DATE
语法：

CURRENT_DATE

描述：

返回当前本地系统日期。

例子：

SELECT CURRENT_DATE "current date" FROM DUMMY;

current date
2010-01-11

CURRENT_TIME
语法：

CURRENT_TIME

描述：

返回当前本地系统时间。

例子：

SELECT CURRENT_TIME "current time" FROM DUMMY;

current time
17:37:37

CURRENT_TIMESTAMP
语法：

CURRENT_TIMESTAMP

描述：

返回当前本地系统的时间戳信息。

例子：

SELECT CURRENT_TIMESTAMP "current timestamp" FROM DUMMY;

current timestamp
2010-01-11 17:38:48.802

CURRECT_UTCDATE
语法：

CURRENT_UTCDATE

描述：

返回当前 UTC 日期。UTC 代表协调世界时，也被称为格林尼治标准时间（GMT）。

例子：

SELECT CURRENT_UTCDATE "Coordinated Universal Date" FROM DUMMY;

Coordinated Universal Time
2010-01-11

CURRENT_UTCTIME
语法：

CURRENT_UTCTIME

描述：

返回当前 UTC 时间。

例子：

SELECT CURRENT_UTCTIME "Coordinated Universal Time" FROM DUMMY;

Coordinated Universal Time
08:41:19

CURRENT_UTCTIMESTAMP
语法：

CURRENT_UTCTIMESTAMP

描述：

返回当前 UTC 时间戳。

例子：

SELECT CURRENT_UTCTIMESTAMP "Coordinated Universal Timestamp" FROM DUMMY;

Coordinated Universal Timestamp
2010-01-11 08:41:42.484

DAYNAME
语法：

DAYNAME (d)

描述：

返回一周中日期 d 的英文名。

例子：

SELECT DAYNAME ('2011-05-30') "dayname" FROM DUMMY;

dayname
MONDAY

DAYOFMONTH
语法：

DAYOFMONTH (d)

描述：

返回一个月中日期 d 的整数数字。

例子：

SELECT DAYOFMONTH ('2011-05-30') "dayofmonth" FROM DUMMY;

dayofmonth
30

DAYOFYEAR
语法：

DAYOFYEAR (d)

描述:

返回一年中代表日期 d 的整数数字。

例子：

SELECT DAYOFYEAR ('2011-05-30') "dayofyear" FROM DUMMY;

dayofyear
150

DAYS_BETWEEN
语法：

DAYS_BETWEEN (d1, d2)

描述：

计算 d1 和 d2 之间的日期数。

例子：

SELECT DAYS_BETWEEN (TO_DATE ('2009-12-05', 'YYYY-MM-DD'), TO_DATE('2010-01-05', 'YYYY-MM-DD')) "days between" FROM DUMMY;

days between
31

EXTRACT
语法：

EXTRACT ({YEAR | MONTH | DAY | HOUR | MINUTE | SECOND} FROM d)

描述：

返回日期 d 中指定的时间日期字段的值。

例子：

SELECT EXTRACT (YEAR FROM TO_DATE ('2010-01-04', 'YYYY-MM-DD')) "extract" FROM DUMMY;

extract
2010

HOUR
语法：

HOUR (t)

描述：

返回时间 t 中表示小时的整数。

例子：

SELECT HOUR ('12:34:56') "hour" FROM DUMMY;

hour
12

ISOWEEK
语法：

ISOWEEK (d)

描述：

返回日期 d 的 ISO 年份和星期数。星期数前缀为字母 W。另请阅 WEEK。

例子：

SELECT ISOWEEK (TO_DATE('2011-05-30', 'YYYY-MM-DD')) "isoweek" FROM DUMMY;

Isoweek
2011-W22

LAST_DAY
语法：

LAST_DAY (d)

描述：

返回包含日期 d 的月的最后一天日期。

例子：

SELECT LAST_DAY (TO_DATE('2010-01-04', 'YYYY-MM-DD')) "last day" FROM DUMMY;

last day
2010-01-31

LOCALTOUTC
语法：

LOCALTOUTC (t, timezone)

描述：

将 timezone 下的本地时间 t 转换为 UTC 时间。

例子：

SELECT LOCALTOUTC (TO_TIMESTAMP('2012-01-01 01:00:00', 'YYYY-MM-DD HH24:MI:SS'), '

EST') "localtoutc" FROM DUMMY;

localtoutc
2012-01-01 06:00:00.0

MINUTE
语法:

MINUTE(t)

描述：

返回时间 t 中表示分钟的数字。

例子：

SELECT MINUTE ('12:34:56') "minute" FROM DUMMY;

minute
34

MONTH
语法：

MONTH(d)

描述：

返回日期 d 所在月份的数字。

例子：

SELECT MONTH ('2011-05-30') "month" FROM DUMMY;

month
5

MONTHNAME
语法：

MONTHNAME(d)

描述：

返回日期 d 所在月份的英文名。

例子：

SELECT MONTHNAME ('2011-05-30') "monthname" FROM DUMMY;

monthname
MAY

NEXT_DAY
语法：

NEXT_DAY (d)

描述：

返回日期 d 的第二天。

例子：

SELECT NEXT_DAY (TO_DATE ('2009-12-31', 'YYYY-MM-DD')) "next day" FROM DUMMY;

next day
2010-01-01

NOW
语法：

NOW ()

描述：

返回当前时间戳。

例子：

SELECT NOW () "now" FROM DUMMY;

now
2010-01-01 16:34:19.894

QUARTER
语法：

QUARTER (d, [, start_month])

描述：

返回日期 d 的年份，季度。第一季度由 start_month 定义的月份开始，如果没有定义 start_month，第一季度假设为从一月开始。

例子：

SELECT QUARTER (TO_DATE('2012-01-01', 'YYYY-MM-DD'), 2) "quarter" FROM DUMMY;

quarter
2011-Q4

SECOND
语法：

SECOND (t)

描述：

返回时间 t 表示的秒数。

例子：

SELECT SECOND ('12:34:56') "second" FROM DUMMY;

second
56

SECONDS_BETWEEN
语法：

SECONDS_BETWEEN (d1, d2)

描述：

计算日期参数 d1 和 d2 之间的秒数，语义上等同于 d2-d1。

例子：

SELECT SECONDS_BETWEEN ('2009-12-05', '2010-01-05') "seconds between" FROM DUMMY;

seconds between
2678400

UTCTOLOCAL
语法：

UTCTOLOCAL (t, timezone)

描述：

将 UTC 时间值转换为时区 timezone 下的本地时间。

例子：

SELECT UTCTOLOCAL (TO_TIMESTAMP('2012-01-01 01:00:00', 'YYYY-MM-DD HH24:MI:SS'), 'EST') "utctolocal" FROM DUMMY;

utctolocal
2011-12-31 20:00:00.0

WEEK
语法：

WEEK (d)

描述：

返回日期 d 所在星期的整数数字。另请参阅 ISOWEEK。

例子：

SELECT WEEK (TO_DATE('2011-05-30', 'YYYY-MM-DD')) "week" FROM DUMMY;

week
23

WEEKDAY
语法：

WEEKDAY (d)

描述：

返回代表日期 d 所在星期的日期数字。返回值范围为 0 至 6，表示 Monday(0)至 Sunday(6)。

例子：

SELECT WEEKDAY (TO_DATE ('2010-12-31', 'YYYY-MM-DD')) "week day" FROM DUMMY;

week day
4

YEAR
语法：

YEAR (d)

描述：

返回日期 d 所在的年份数。

例子：

SELECT YEAR (TO_DATE ('2011-05-30', 'YYYY-MM-DD')) "year" FROM DUMMY;

year
2011

数字函数

数字函数接受数字或者带有数字的字符串作为输入，返回数值。当数字字符的字符串作为输入时，在计算结果前，自动执行字符串到数字的隐式转换。

ABS

语法：

ABS (n)

描述：

返回数字参数 n 的绝对值。

例子：

SELECT ABS (-1) "absolute" FROM DUMMY;

absolute
1

ACOS

语法：

ACOS (n)

描述：

返回参数 n 的反余弦，以弧度为单位，值为-1 至 1。

例子：

SELECT ACOS (0.5) "acos" FROM DUMMY;

acos
1.0471975511965979

ASIN
语法：

ASIN (n)

描述：

返回参数 n 的反正弦值，以弧度为单位，值为-1 至 1。

例子：

SELECT ASIN (0.5) "asin" FROM DUMMY;

asin
0.5235987755982989

ATAN
语法：

ATAN (n)

描述：

返回参数 n 的反正切值，以弧度为单位，n 的范围为无限。

例子：

SELECT ATAN (0.5) "atan" FROM DUMMY;

atan
0.4636476090008061

ATAN2
语法：

ATAN2 (n, m)

描述：

返回两数 n 和 m 比率的反正切值，以弧度为单位。这和 ATAN(n/m)的结果一致。

例子：

SELECT ATAN2 (1.0, 2.0) "atan2" FROM DUMMY;

atan2
0.4636476090008061

BINTOHEX
语法：

BINTOHEX (expression)

描述：

将二进制值转换为十六进制。

例子：

SELECT BINTOHEX('AB') "bintohex" FROM DUMMY;

bintohex
4142

BITAND
语法：

BITAND (n, m)

描述：

对参数 n 和 m 的位执行 AND 操作。n 和 m 都必须是非负整数。BITAND 函数返回 BIGINT 类型的结果。

例子：

SELECT BITAND (255, 123) "bitand" FROM DUMMY;

bitand
123

CEIL
语法：

CEIL(n)

描述：

返回大于或者等于 n 的第一个整数。

例子：

SELECT CEIL (14.5) "ceiling" FROM DUMMY;

ceiling
15

COS
语法：

COS (n)

描述：

返回参数 n 的余弦值，以弧度为单位。

例子：

SELECT COS (0.0) "cos" FROM DUMMY;

cos
1.0

COSH
语法：

COSH (n)

描述：

返回参数 n 的双曲余弦值。

例子：

SELECT COSH (0.5) "cosh" FROM DUMMY;

cosh
1.1276259652063807

COT
语法：

COT (n)

描述：

计算参数 n 的余切值，其中 n 以弧度表示。

例子：

SELECT COT (40) "cot" FROM DUMMY;

cot
-0.8950829176379128

EXP
语法：

EXP (n)

描述：

返回以 e 为底，n 为指数的计算结果。

例子：

SELECT EXP (1.0) "exp" FROM DUMMY;

exp
2.718281828459045

FLOOR
语法：

FLOOR (n)

描述：

返回不大于参数 n 的最大数。

例子:

SELECT FLOOR (14.5) "floor" FROM DUMMY;

floor
14

GREATEST
语法：

GREATEST (n1 [, n2]...)

描述：

返回参数 n1,n2,…最大数。

例子：

SELECT GREATEST ('aa', 'ab', 'ba', 'bb') "greatest" FROM DUMMY;

greatest
bb

HEXTOBIN
语法：

HEXTOBIN (value)

描述：

将十六进制数转换为二进制数。

例子：

SELECT HEXTOBIN ('1a') "hextobin" FROM DUMMY;

hextobin
1A

LEAST
语法：

LEAST (n1 [, n2]...)

描述：

返回参数 n1,n2,…最小数。

例子：

SELECT LEAST('aa', 'ab', 'ba', 'bb') "least" FROM DUMMY;

least
aa

LN
语法：

LN (n)

描述：

返回参数 n 的自然对数。

例子：

SELECT LN (9) "ln" FROM DUMMY;

ln
2.1972245773362196

LOG
语法：

LOG (b, n)

描述：

返回以 b 为底，n 的自然对数值。底 b 必须是大于 1 的正数，且 n 必须是正数。

例子：

SELECT LOG (10, 2) "log" FROM DUMMY;

log
0.30102999566398114

MOD
语法：

MOD (n, d)

描述：

返回 n 整除 b 的余数值。

当 n 为负时，该函数行为不同于标准的模运算。

以下列举了 MOD 函数返回结果的例子

如果 d 为零，返回 n。
如果 n 大于零，且 n 小于 d，则返回 n。
如果 n 小于零，且 n 大于 d，则返回 n。

在上文提到的其他情况中，利用 n 的绝对值除以 d 的绝对值来计算余数。如果 n 小于 0，则 MOD 返回的余数为负数；如果 n 大于零，MOD 返回的余数为正数。

例子：

SELECT MOD (15, 4) "modulus" FROM DUMMY;

modulus
3

SELECT MOD (-15, 4) "modulus" FROM DUMMY;

modulus
-3

POWER
语法：

POWER (b, e)

描述：

计算以 b 为底，e 为指数的值。

例子：

SELECT POWER (2, 10) "power" FROM DUMMY;

power
1024.0

ROUND
语法：

ROUND (n [, pos])

描述：

返回参数 n 小数点后 pos 位置的值。

例子：

SELECT ROUND (16.16, 1) "round" FROM DUMMY;

round
16.2

SELECT ROUND (16.16, -1) "round" FROM DUMMY;

round
20

SIGN
语法：

SIGN (n)

描述：

返回 n 的符号（正或负）。如果 n 为正，则返回 1；n 为负，返回-1，n 为 0 返回 0。

例子：

SELECT SIGN (-15) "sign" FROM DUMMY;

sign
-1

SIN
语法：

SIN (n)

描述：

返回参数 n 的正弦值，以弧度为单位。

例子：

SELECT SIN (3.141592653589793/2) "sine" FROM DUMMY;

sine
1.0

SINH
语法：

SINH (n)

描述：

返回 n 的双曲正弦值，以弧度为单位。

例子：

SELECT SINH (0.0) "sinh" FROM DUMMY;

sinh
0.0

SQRT
语法：

SQRT (n)

描述：

返回 n 的平方根。

例子：

SELECT SQRT (2) "sqrt" FROM DUMMY;

sqrt
1.4142135623730951

TAN
语法：

TAN (n)

描述：

返回 n 的正切值，以弧度为单位。

例子：

SELECT TAN (0.0) "tan" FROM DUMMY;

tan
0.0

TANH
语法：

TANH (n)

描述：

返回 n 的双曲正切值，以弧度为单位。

例子：

SELECT TANH (1.0) "tanh" FROM DUMMY;

tanh
0.7615941559557649

UMINUS
语法：

UMINUS (n)

描述：

返回 n 的负值。

例子：

SELECT UMINUS(-765) "uminus" FROM DUMMY;

uminus
765

字符串函数

ASCII

语法：

ASCII(c)

描述：

返回字符串 c 中第一个字节的 ASCII 值。

SELECT ASCII('Ant') "ascii" FROM DUMMY;

ascii
65

CHAR

语法：

CHAR (n)

描述：

返回 ASCII 值为数字 n 的字符。

例子：

SELECT CHAR (65) || CHAR (110) || CHAR (116) "character" FROM DUMMY;

character
Ant

CONCAT

语法：

CONCAT (str1, str2)

描述：

返回位于 str1 后的 str2 联合组成的字符串。级联操作符(||)与该函数作用相同。

例子：

SELECT CONCAT ('C', 'at') "concat" FROM DUMMY;

concat
Cat

LCASE
语法：

LCASE(str)

描述：

将字符串 str 中所有字符转换为小写。

注意：LCASE 函数作用与 LOWER 函数相同。

例子：

SELECT LCASE ('TesT') "lcase" FROM DUMMY;

lcase
test

LEFT
语法：

LEFT (str, n)

描述：

返回字符串 str 开头 n 个字符/位的字符。

例子：

SELECT LEFT ('Hello', 3) "left" FROM DUMMY;

left
Hel

LENGTH
语法：

LENGTH(str)

描述：

返回字符串 str 中的字符数。对于大对象(LOB)类型，该函数返回对象的字节长度。

例子：

SELECT LENGTH ('length in char') "length" FROM DUMMY;

length
14

LOCATE
语法：

LOCATE (haystack, needle)

描述：

返回字符串 haystack 中子字符串 needle 所在的位置。如果未找到，则返回 0。

例子：

SELECT LOCATE ('length in char', 'char') "locate" FROM DUMMY;

Locate
11

SELECT LOCATE ('length in char', 'length') "locate" FROM DUMMY;

Locate
1

SELECT LOCATE ('length in char', 'zchar') "locate" FROM DUMMY;

Locate
0

LOWER
语法：

LOWER (str)

描述：

将字符串 str 中所有字符转换为小写。

注意：LOWER 函数作用与 LCASE 相同。

例子：

SELECT LOWER ('AnT') "lower" FROM DUMMY;

lower
ant

LPAD
语法：

LPAD (str, n [, pattern])

描述：

从左边开始对字符串 str 使用空格进行填充，达到 n 指定的长度。如果指定了 pattern 参数，字符串 str 将按顺序填充直到满足 n 指定的长度。

例子：

SELECT LPAD ('end', 15, '12345') "lpad" FROM DUMMY;

lpad
123451234512end

LTRIM
语法：

LTRIM (str [, remove_set])

描述：

返回字符串 str 截取所有前导空格后的值。如果定义了 remove_set，LTRIM 从起始位置移除字符串 str 包含该集合中的字符，该过程持续至到达不在 remove_set 中的字符。

注意：remove_set 被视为字符集合，而非搜索字符。

例子：

SELECT LTRIM ('babababAabend','ab') "ltrim" FROM DUMMY;

ltrim
Aabend

NCHAR
语法：

NCHAR (n)

描述：

返回整数 n 表示的 Unicode 字符。

例子：

SELECT NCHAR (65) "nchar" FROM DUMMY;

nchar
A

REPLACE
语法：

REPLACE (original_string, search_string, replace_string)

描述：

搜索 original_string 所有出现的 search_string，并用 replace_string 替换。

如果 original_string 为空， 则返回值也为空。
如果 original_string 中两个重叠的子字符串与 search_string 匹配，只有第一个会被替换。
如果 original_string 未出现 search_string，则返回未修改的 original_string。
如果 original_string，search_string 或者 replace_string 为 NULL，则返回值也为 NULL。

例子：

SELECT REPLACE ('DOWNGRADE DOWNWARD','DOWN', 'UP') "replace" FROM DUMMY;

replace
UPGRADE UPWARD

RIGHT
语法：

RIGHT(str, n)

描述：

返回字符串 str 中最右边的 n 字符/字节。

例子：

SELECT RIGHT('HI0123456789', 3) "right" FROM DUMMY;

right
789

RPAD
语法：

RPAD (str, n [, pattern])

描述：

从尾部开始对字符串 str 使用空格进行填充，达到 n 指定的长度。如果指定了 pattern 参数，字符串 str 将按顺序填充直到满足 n 指定的长度。

例子：

SELECT RPAD ('end', 15, '12345') "right padded" FROM DUMMY;

right padded
end123451234512

RTRIM
语法：

RTRIM (str [,remove_set])

描述：

返回字符串 str 截取所有后置空格后的值。如果定义了 remove_set，RTRIM 从尾部位置移除字符串 str 包含该集合中的字符，该过程持续至到达不在 remove_set 中的字符。

注意：remove_set 被视为字符集合，而非搜索字符。

例子：

SELECT RTRIM ('endabAabbabab','ab') "rtrim" FROM DUMMY;

rtrim
endabA

SUBSTR_AFTER
语法：

SUBSTR_AFTER (str, pattern)

描述：

返回 str 中位于 pattern 第一次出现位置后的子字符串。

如果 str 不包含 pattern 子字符串，则返回空字符串。
如果 pattern 为空字符串，则返回 str。
如果 str 或者 pattern 为 NULL，则返回 NULL。

例子：

SELECT SUBSTR_AFTER ('Hello My Friend','My ') "substr after" FROM DUMMY;

substr after
Friend

SUBSTR_BEFORE
语法：

SUBSTR_BEFORE (str, pattern)

描述：

返回 str 中位于 pattern 第一次出现位置前的子字符串。

如果 str 不包含 pattern 子字符串，则返回空字符串。
如果 pattern 为空字符串，则返回 str。
如果 str 或者 pattern 为 NULL，则返回 NULL。

例子：

SELECT SUBSTR_BEFORE ('Hello My Friend','My') "substr before" FROM DUMMY;

substr before
Hello

SUBSTRING
语法：

SUBSTRING (str, start_position [, string_length])

描述：

返回字符串 str 从 start_position 开始的子字符串。SUBSTRING 可以返回 start_position 起的剩余部分字符或者作为可选，返回由 string_length 参数设置的字符数。

如果 start_position 小于 0，则被视为 1。
如果 string_length 小于 1，则返回空字符串。

例子：

SELECT SUBSTRING ('1234567890',4,2) "substring" FROM DUMMY;

substring
45

TRIM
语法：

TRIM ([[LEADING | TRAILING | BOTH] trim_char FROM] str)

描述：

返回移除前导和后置空格后的字符串 str。截断操作从起始(LEADING)、结尾(TRAILING)或者两端(BOTH)执行。

如果 str 或者 trim_char 为空，则返回 NULL。
如果没有指定可选项，TRIM 移除字符串 str 中两端的子字符串 trim_char。
如果没有指定 trim_char，则使用单个空格。

例子：

SELECT TRIM ('a' FROM 'aaa123456789aa') "trim both" FROM DUMMY;

trim both
123456789

SELECT TRIM (LEADING 'a' FROM 'aaa123456789aa') "trim leading" FROM DUMMY;

trim leading
123456789aa

UCASE
语法：

UCASE (str)

描述：

将字符串 str 中所有字符转换为大写。

注意：UCASE 函数作用与 UPPER 函数相同。

例子：

SELECT UCASE ('Ant') "ucase" FROM DUMMY;

ucase
ANT

UNICODE
语法：

UNICODE(c)

描述：

返回字符串中首字母的 UnIcode 字符码数字；如果首字母不是有效编码，则返回 NULL。

例子：

SELECT UNICODE ('#') "unicode" FROM DUMMY;

unicode
35

UPPER
语法：

UPPER (str)

描述：

将字符串 str 中所有字符转换为大写。

注意：UPPER 函数作用与 UCASE 相同。

例子：

SELECT UPPER ('Ant') "uppercase" FROM DUMMY;

uppercase
ANT

杂项函数

COALESCE
语法：

COALESCE (expression_list)

描述：

返回 list 中非 NULL 的表达式。Expression_list 中必须包含至少两个表达式，并且所有表达式都是可比较的。如果所有的参数都为 NULL，则结果也为 NULL。

例子：

CREATE TABLE coalesce_example (ID INT PRIMARY KEY, A REAL, B REAL);

INSERT INTO coalesce_example VALUES(1, 100, 80);

INSERT INTO coalesce_example VALUES(2, NULL, 63);

INSERT INTO coalesce_example VALUES(3, NULL, NULL);

SELECT id, a, b, COALESCE (a, b*1.1, 50.0) "coalesce" FROM coalesce_example;

1 100.0 80.0 100.0

2 NULL 63.0 69.30000305175781

3 NULL NULL 50.0

CURRENT_CONNECTION
语法：

CURRENT_CONNECTION

描述：

返回当前连接 ID。

例子：

SELECT CURRENT_CONNECTION "current connection" FROM DUMMY;

current connection
2

CURRENCT_SCHEMA
语法：

CURRENT_SCHEMA

描述：

返回当前数据集合名。

例子：

SELECT CURRENT_SCHEMA "current schema" FROM DUMMY;

current schema
SYSTEM

CURRENT_USER
语法：

CURRENT_USER

描述：

返回当前语句上下文的用户名，即当前授权堆栈顶部的用户名。

例子：

```
-- example showing basic function operation using SYSTEM user
SELECT CURRENT_USER "current user" FROM DUMMY;

current user
SYSTEM

-- definer-mode procedure declared by USER_A
CREATE PROCEDURE USER_A.PROC1 LANGUAGE SQLSCRIPT SQL SECURITY DEFINER AS
BEGIN
SELECT CURRENT_USER "current user" FROM DUMMY;
END;

-- USER_B executing USER_A.PROC1
CALL USER_A.PROC1;

current user
USER_A

-- invoker-mode procedure declared by USER_A
CREATE PROCEDURE USER_A.PROC2 LANGUAGE SQLSCRIPT SQL SECURITY INVOKER AS
BEGIN
SELECT CURRENT_USER "current user" FROM DUMMY;
END;

-- USER_B is executing USER_A.PROC
CALL USER_A.PROC2;

current user
USER_B
```

GROUPING_ID

语法：

```
GROUPING_ID(column_name_list)
```

描述：

GROUPING_ID 函数可以使用 GROUPING SETS 返回单个结果集中的多级聚集。GROUPING_ID 返回一个整数识别每行所在的组集合。GROUPING_ID 每一列必须是 GROUPING SETS 中的元素。

通过把生成的位矢量从 GROUPING SETS 转换为十进制数，将位矢量视作二进制数，分配 GROUPING_ID。组成位矢量后，0 分配给 GROUPING SETS 指定的每一列，否则根据 GROUPING SETS 出现的顺序分配 1。通过将位矢量作为二进制数处理，该函数返回一个整型值作为输出。

例子：

```
SELECT customer, year, product, SUM(sales),
GROUPING_ID(customer, year, product)
FROM guided_navi_tab
GROUP BY GROUPING SETS (
(customer, year, product),
(customer, year),
(customer, product),
(year, product),
(customer),
(year),
(product));
```

```
CUSTOMER YEAR PRODUCT SUM(SALES) GROUPING_ID(CUSTOMER,YEAR,PRODUCT)
1 C1 2009 P1 100 0
2 C1 2010 P1 50 0
3 C2 2009 P1 200 0
4 C2 2010 P1 100 0
5 C1 2009 P2 200 0
6 C1 2010 P2 150 0
7 C2 2009 P2 300 0
8 C2 2010 P2 150 0
9 C1 2009 a 300 1
10 C1 2010 a 200 1
11 C2 2009 a 500 1
12 C2 2010 a 250 1
13 C1 a P1 150 2
14 C2 a P1 300 2
15 C1 a P2 350 2
16 C2 a P2 450 2
17 a 2009 P1 300 4
18 a 2010 P1 150 4
```

19 a 2009 P2 500 4
20 a 2010 P2 300 4
21 C1 a a 500 3
22 C2 a a 750 3
23 a 2009 a 800 5
24 a 2010 a 450 5
25 a a P1 450 6
26 a a P2 800 6

IFNULL

语法：

IFNULL (expression1, expression2)

描述：

返回输入中第一个不为 NULL 的表达式。

如果 expression1 不为 NULL，则返回 expression1。

如果 expression2 不为 NULL，则返回 expression2。

如果输入表达式都为 NULL，则返回 NULL。

例子：

SELECT IFNULL ('diff', 'same') "ifnull" FROM DUMMY;

ifnull
diff

SELECT IFNULL (NULL, 'same') "ifnull" FROM DUMMY;

ifnull
same

SELECT IFNULL (NULL, NULL) "ifnull" FROM DUMMY;

ifnull
NULL

MAP

语法：

MAP (expression, search1, result1 [, search2, result2] ... [, default_result])

描述：

在搜索集合中搜索 expression，并返回相应的结果。

如果未找到 expression 值，并且定义了 default_result，则 MAP 返回 default_result。

如果未找到 expression 值，并且未定义 default_result，MAP 返回 NULL。

注意：

搜索值和相应的结果总是以搜索-结果方式提供。

例子：

SELECT MAP(2, 0, 'Zero', 1, 'One', 2, 'Two', 3, 'Three', 'Default') "map" FROM DUMMY;

map
Two

SELECT MAP(99, 0, 'Zero', 1, 'One', 2, 'Two', 3, 'Three', 'Default') "map" FROM DUMMY;

map
Default

SELECT MAP(99, 0, 'Zero', 1, 'One', 2, 'Two', 3, 'Three') "map" FROM DUMMY;

map
NULL

NULLIF

语法：

NULLIF (expression1, expression2)

描述：

NULLIF 比较两个输入表达式的值，如果第一个表达式等于第二个，NULLIF 返回 NULL。

如果 expression1 不等于 expression2，NULLIF 返回 expression1。

如果 expression2 为 NULL，NULLIF 返回 expression1。

例子：

SELECT NULLIF ('diff', 'same') "nullif" FROM DUMMY;

nullif
diff

SELECT NULLIF('same', 'same') "nullif" FROM DUMMY;

nullif
NULL

SESSION_CONTEXT

语法：

SESSION_CONTEXT(session_variable)

描述：

返回分配给当前用户的 session_variable 值。

访问的 session_variable 可以是预定义或者用户自定义。预定义的会话变量可以通过客户端设置的有'APPLICATION', 'APPLICATIONUSER'以及'TRACEPROFILE'。

会话变量可以定义或者修改通过使用命令 SET [SESSION] <variable_name> = <value>，使用 UNSET [SESSION] <variable_name>取消设置。

SESSION_CONTEXT 返回最大长度为 512 字符的 NVARCHAR 类型。

例子：

读取会话变量：

SELECT SESSION_CONTEXT('APPLICATION') "session context" FROM DUMMY;

session context
HDBStudio

SESSION_USER

语法：

SESSION_USER

描述：

返回当前会话的用户名。

例子：

```
-- example showing basic function operation using SYSTEM user
SELECT SESSION_USER "session user" FROM DUMMY;

session user
SYSTEM

-- definer-mode procedure declared by USER_A
CREATE PROCEDURE USER_A.PROC1 LANGUAGE SQLSCRIPT SQL SECURITY DEFINER AS
BEGIN
SELECT SESSION_USER "session user" FROM DUMMY;
END;

-- USER_B is executing USER_A.PROC
CALL USER_A.PROC1;

session user
USER_B

-- invoker-mode procedure declared by USER_A
CREATE PROCEDURE USER_A.PROC2 LANGUAGE SQLSCRIPT SQL SECURITY INVOKER AS
BEGIN
SELECT SESSION_USER "session user" FROM DUMMY;
END;

-- USER_B is executing USER_A.PROC
CALL USER_A.PROC2;

session user
USER_B
```

SYSUUID

语法：

```
SYSUUID
```

描述：

返回 SAP HANA 连接实例的 SYSUUID。

例子：

```
SELECT SYSUUID FROM DUMMY;

SYSUUID
4DE3CD576C79511BE10000000A3C2220
```

SQL 语句

本章描述 SAP HANA 数据库支持的 SQL 语句。

- Schema Definition and Manipulation Statements
- Data Manipulation Statements
- System Management Statements
- Session Management Statements
- Transaction Management Statements
- Access Control Statements
- Data Import Export Statements

集合定义和操纵语句

ALTER AUDIT POLICY

语法：

ALTER AUDIT POLICY <policy_name> <audit_mode>

语法元素：

<policy_name> ::= <identifier>

被改变的审计策略名：

<audit_mode> ::= ENABLE | DISABLE

Audit_mode 启用或禁用审计策略。

ENABLE
启用审计策略。

DISABLE
禁用审计策略。

描述：

ALTER AUDIT POLICY 语句启用或禁用审计策略。<policy_name>必须定义一个已存在的审计策略名。

只有拥有系统权限 AUDIT ADMIN 的数据库用户允许改变审计策略。每个拥有该权限的数据库用户可以修改任意的审计策略，无论是否由该用户创建。

新建的审计策略默认为禁用，并且不会发生任何审计。因此，必须启动该审计策略来执行审计。

审计策略可以视需要禁用和启用。

配置参数：

以下审计的配置参数存储在文件 global.ini，在审计配置部分：

global_auditing_state ('true' / 'false')
无论启动的审计策略数量多少，审计只会在配置参数 global_auditing_state 设置为 true 时启用，默认值 false。

default_audit_trail_type ('SYSLOGPROTOCOL' / 'CSVTEXTFILE')
指定如何存储审计结果。
SYSLOGPROTOCOL：使用系统 syslog。
CSVTEXTFILE：审计信息值以逗号分隔存储在一个文本文件中。

default_audit_trail_path
指定 CSVTEXTFILE 存储的文件路径。

如果用户拥有需要的系统权限，参数可以在监控视图 M_INIFILE_CONTENTS 中选择。这些只有在显示设置的情况下才可见。

系统表和监控视图：

AUDIT_POLICY：显示所有审计策略和状态。
M_INIFILE_CONTENTS：显示数据库系统配置参数。

只有拥有系统权限 CATALOG READ, DATA ADMIN 或 INIFILE ADMIN 的用户可以查看 M_INIFILE_CONTENTS 视图中的内容，对于其他用户则为空。

例子：

该例子中你需要先利用如下语句创建名为 priv_audit 的审计权限：

CREATE AUDIT POLICY priv_audit AUDITING SUCCESSFUL GRANT PRIVILEGE, REVOKE PRIVILEGE, GRANT ROLE, REVOKE ROLE LEVEL CRITICAL;

现在你可以启用审计策略：

ALTER AUDIT POLICY priv_audit ENABLE;

你也可以禁用该审计策略：

ALTER AUDIT POLICY priv_audit DISABLE;

ALTER FULLTEXT INDEX

语法：

ALTER FULLTEXT INDEX <index_name> <alter_fulltext_index_option>

语法元素：

<index_name> ::= <identifier>

被修改的全文索引标识符：

<alter_fulltext_index_option> ::= <fulltext_parameter_list> | <queue_command> QUEUE

定义了全文索引的参数或者全文索引队列的状态是否应该修改。后者只对异步显式全文索引可见。

<fulltext_parameter_list> ::= <fulltext_parameter> [, ...]

修改的全文索引参数列表：

<fulltext_parameter> ::= FUZZY SEARCH INDEX <on_off>
| PHRASE INDEX RATIO <index_ratio>
| <change_tracking_elem>

<on_off> ::= ON | OFF

FUZZY SEARCH INDEX
使用模糊搜索索引。

PHRASE INDEX RATIO
定义短语索引比率。

<index_ratio> ::= <float_literal>

定义短语索引比率的百分比，值必须为 0.0 与 1.0 之间。

SYNC[HRONOUS]
改变全文索引至同步模式。

ASYNC[HRONOUS]
改变全文索引至异步模式。

<flush_queue_elem> ::= EVERY <integer_literal> MINUTES
| AFTER <integer_literal> DOCUMENTS
| EVERY <integer_literal> MINUTES OR AFTER <integer_literal>

DOCUMENTS

当使用异步索引时，你可以利用 flush_queue_elem 定义更新全文索引的时间。

<queue_command> ::= FLUSH | SUSPEND | ACTIVATE

FLUSH
利用正在处理队列的文件更新全文索引。

SUSPEND
暂停全文索引处理队列。

ACTIVATE
激活全文索引处理队列。

描述：

使用该命令，你可以修改全文索引的参数或者索引处理队列的状态。队列是用来激活全文索引以异步方式工作的机制，例如插入操作被阻塞，直到文件处理完。

ALTER FULLTEXT INDEX <index_name> <fulltext_elem_list>语句修改全文索引的参数。

ALTER FULLTEXT INDEX <index_name> <queue_parameters>语句修改异步全文索引的处理队列状态。

例子：

ALTER FULLTEXT INDEX i1 PHRASE INDEX RATIO 0.3 FUZZY SEARCH INDEX ON

在上述例子中，对于全文索引’i1’，短文索引比率设为 30,，并且启用了模糊搜索索引。

ALTER FULLTEXT INDEX i2 SUSPEND QUEUE

暂停全文索引’i2’的队列

ALTER FULLTEXT INDEX i2 FLUSH QUEUE

利用队列中已处理的文件更新全文索引’i2’。

ALTER INDEX
ALTER INDEX <index_name> REBUILD

语法元素：

<index_name>::= <identifier>

定义重建的索引名。

描述：

ALTER INDEX 语句重建索引。

例子：

以下例子重建索引 idx。

ALTER INDEX idx REBUILD;

ALTER SEQUENCE
语法：

ALTER SEQUENCE <sequence_name> [<alter_sequence_parameter_list>]

[RESET BY <reset_by_subquery>]

语法元素：

<sequence_name> ::= <identifier>

被修改的序列名。

<alter_sequence_parameter_list> ::= <alter_sequence_parameter>, ...

<alter_sequence_parameter> ::= <sequence_parameter_restart_with>
| <basic_sequence_parameter>

<sequence_parameter_restart_with> ::= RESTART WITH <restart_value>

<basic_sequence_parameter> ::= INCREMENT BY <increment_value>
| MAXVALUE <maximum_value>
| NO MAXVALUE
| MINVALUE <minimum_value>
| NO MINVALUE
| CYCLE
| NO CYCLE

RESTART WITH
序列的起始值。如果你没有指定 RESTART WITH 子句的值，将使用当前序列值。

<restart_value> ::= <integer_literal>

由序列生成器提供的第一个值为 0 至 4611686018427387903 之间的整数。

INCREMENT BY
序列增量值。

<increment_value> ::= <integer_literal>

利用一个整数增加或者减少序列的值。

MAXVALUE
定义序列生成的最大值。

<maximum_value> ::= <integer_literal>

一个正整数定义序列可生成的最大数值，必须为 0 至 4611686018427387903 之间。

NO MAXVALUE
使用 NO MAXVALUE 指令，递增序列的最大值将为 4611686018427387903，递减序列的最大值为-1。

MINVALUE
定义序列生成的最小值。

<minimum_value> ::= <integer_literal>

一个正整数定义序列可生成的最小数值，必须为 0 至 4611686018427387903 之间。

NO MINVALUE
使用 NO MINVALUE 指令，递增序列的最小值将为 1，递减序列的最小值为-
4611686018427387903。

CYCLE
使用 CYCLE 指令，序列在到达最大值或最小值后将会重新开始。

NO CYCLE
使用 NO CYCLE 指令，序列在到达最大值或最小值后将不会重新开始。

<reset_by_subquery> ::= <subquery>

系统重启期间，系统自动执行 RESET BY 语句，并且将用 RESET BY 子查询确定的值重启序列。

关于子查询的详情，请参阅 Subquery。

描述：

ALTER SEQUENCE 语句用来修改序列生成器的参数。

例子：

在下面的例子中，你把序列 seq 的起始序列值改为 2。

```
ALTER SEQUENCE seq RESTART WITH 2;
```

在下面的例子中，你把序列 seq 的最大值改为 100，并且没有最小值。

```
ALTER SEQUENCE seq MAXVALUE 100 NO MINVALUE;
```

在下面的例子中，你把序列 seq 的增量改为 2，并且限制为"no cycle"。

```
ALTER SEQUENCE seq INCREMENT BY 2 NO CYCLE;
```

在下面的例子中，你首先创建表 r，有一列 a。然后你将序列 seq 的 reset-by 子查询修改为列 a 包含的最大值。

```
CREATE TABLE r (a INT);
ALTER SEQUENCE seq RESET BY SELECT MAX(a) FROM r;
```

ALTER TABLE

语法：

```
ALTER TABLE <table_name>
{
<add_column_clause>
| <drop_column_clause>
| <alter_column_clause>
| <add_primary_key_clause>
| <drop_primary_key_clause>
| <preload_clause>
| <table_conversion_clause>
| <move_clause>
| <add_range_partition_clause>
| <move_partition_clause>
| <drop_range_partition_clause>
| <partition_by_clause>
| <disable_persistent_merge_clause>
| <enable_persistent_merge_clause>
| <enable_delta_log>
| <disable_delta_log>
| <enable_automerge>
| <disable_automerge>
}
```

语法元素：

```
<table_name> ::= [<schema_name>.]<identifier>
<add_column_clause> ::= ADD ( <column_definition> [<column_constraint>], ...
)
<drop_column_clause> ::= DROP ( <column_name>, ... )
<alter_column_clause> ::= ALTER ( <column_definition> [<column_constraint>], ..

. )
<column_definition> ::= <column_name> <data_type> [<column_store_data_type>]
[<ddic_data_type>] [DEFAULT <default_value>] [GENERATED ALWAYS AS <expression>]
<column_constraint> ::= NULL
| NOT NULL
| UNIQUE [BTREE | CPBTREE]
| PRIMARY KEY [BTREE | CPBTREE]

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

<data_type> ::= DATE | TIME | SECONDDATE | TIMESTAMP | TINYINT | SMALLINT | INTEGER | BIGINT | SMALLDECIMAL | DECIMAL | REAL | DOUBLE| VARCHAR | NVARCHAR | ALPHANUM | SHORTTEXT | VARBINARY | BLOB| CLOB | NCLOB | TEXT

<column_store_data_type> ::= CS_ALPHANUM | CS_INT | CS_FIXED | CS_FLOAT | CS_DOUBLE | CS_DECIMAL_FLOAT | CS_FIXED(p-s, s) | CS_SDFLOAT
| CS_STRING | CS_UNITEDECFLOAT | CS_DATE | CS_TIME
| CS_FIXEDSTRING | CS_RAW | CS_DAYDATE | CS_SECONDTIME | CS_LONGDATE | CS_SECONDDATE

<ddic_data_type> ::= DDIC_ACCP | DDIC_ALNM | DDIC_CHAR | DDIC_CDAY | DDIC_CLNT | DDIC_CUKY
| DDIC_CURR | DDIC_D16D | DDIC_D34D | DDIC_D16R | DDIC_D34R | DDIC_D16S | DDIC_D34S
| DDIC_DATS | DDIC_DAY | DDIC_DEC | DDIC_FLTP | DDIC_GUID
| DDIC_INT1 | DDIC_INT2 | DDIC_INT4 | DDIC_INT8 | DDIC_LANG | DDIC_LCHR | DDIC_MIN
|DDIC_MON| DDIC_LRAW | DDIC_NUMC | DDIC_PREC | DDIC_QUAN | DDIC_RAW
| DDIC_RSTR | DDIC_SEC | DDIC_SRST | DDIC_SSTR | DDIC_STRG | DDIC_STXT | DDIC_TIMS| DDIC_UNIT
| DDIC_UTCM | DDIC_UTCL | DDIC_UTCS | DDIC_TEXT | DDIC_VARC | DDIC_WEEK

<default_value> ::= NULL | <string_literal> | <signed_numeric_literal> | <uns

igned_numeric_literal>

<signed_numeric_literal> ::= [<sign>] <unsigned_numeric_literal>

<sign> ::= + | -

<unsigned_numeric_literal> ::= <exact_numeric_literal> | <approximate_numeric_literal>
<exact_numeric_literal> ::= <unsigned_integer> [<period> [<unsigned_integer>]]| <period>
<unsigned_integer>

<period> ::= .

<approximate_numeric_literal> ::= <mantissa> E <exponent>
<mantissa> ::= <exact_numeric_literal>
<exponent> ::= <signed_integer>
<signed_integer> ::= [<sign>] <unsigned_integer>
<unsigned_integer> ::= <digit>...

DEFAULT

DEFAULT 定义了 INSERT 语句没有为列提供值情况下，默认分配的值。

可供使用的数据类型有 DATE, TIME, SECONDDATE, TIMESTAMP, TINYINT, SMALLINT, INTEGER, BIGINT, SMALLDECIMAL, DECIMAL, REAL, DOUBLE, VARCHAR, NVARCHAR, ALPHANUM, SHORTTEXT, VARBINARY, BLOB, CLOB, NCLOB and TEXT。

可供使用的列存储数据类型有 CS_ALPHANUM, CS_INT, CS_FIXED, CS_FLOAT, CS_DOUBLE,CS_DECIMAL_FLOAT, CS_FIXED(p-s,s), CS_SDFLOAT, CS_STRING, CS_UNITEDECFLOAT,CS_DATE, CS_TIME,CS_FIXEDSTRING, CS_RAW, CS_DAYDATE, CS_SECONDTIME, CS_LONGDATE, and CS_SECONDDATE。

可供使用的 DDIC 数据类型有 DDIC_ACCP, DDIC_ALNM, DDIC_CHAR, DDIC_CDAY, DDIC_CLNT, DDIC_CUKY,DDIC_CURR, DDIC_D16D, DDIC_D34D, DDIC_D16R, DDIC_D34R, DDIC_D16S, DDIC_D34S, DDIC_DATS,DDIC_DAY, DDIC_DEC, DDIC_FLTP, DDIC_GUID, DDIC_INT1, DDIC_INT2, DDIC_INT4, DDIC_INT8,DDIC_LANG, DDIC_LCHR, DDIC_MIN, DDIC_MON, DDIC_LRAW, DDIC_NUMC, DDIC_PREC, DDIC_QUAN,DDIC_RAW, DDIC_RSTR, DDIC_SEC, DDIC_SRST, DDIC_SSTR, DDIC_STRG, DDIC_STXT,DDIC_TIMS,DDIC_UNIT, DDIC_UTCM, DDIC_UTCL, DDIC_UTCS, DDIC_TEXT, DDIC_VARC, DDIC_WEEK。

GENERATED ALWAYS AS
指定在运行时生成的列值的表达式。

<column_constraint> ::= NULL
| NOT NULL
| UNIQUE [BTREE | CPBTREE]
| PRIMARY KEY [BTREE | CPBTREE]

NULL | NOT NULL
NOT NULL 禁止列的值为 NULL。
如果指定了 NULL，将不被认为是常量，其表示一列可能含有空值，默认为 NULL。

UNIQUE
指定为唯一键的列。
一个唯一的复合键指定多个列作为唯一键。有了 unique 约束，多行不能在同一列中具有相同的值。

PRIMARY KEY
主键约束是 NOT NULL 约束和 UNIQUE 约束的组合，其禁止多行在同一列中具有相同的值。

BTREE | CPBTREE
指定索引类型。当列的数据类型为字符串、二进制字符串、十进制数或者约束是一个组合键，或非唯一键，默认的索引类型为 CPBTREE，否则使用 BTREE。
为了使用 B+-树索引，必须使用 BTREE 关键字；对于 CPB+-树索引，需使用 CPBTREE 关键字。
B+-树是维护排序后的数据进行高效的插入、删除和搜索记录的树。

CPB+-树表示压缩前缀 B+-树，是基于 pkB-tree 树。CPB+树是一个非常小的索引，因为它使用“部分键”，只是索引节点全部键的一部分。对于更大的键，CPB+-树展现出比 B+-树更好的性能。
如果省略了索引类型，SAP HANA 数据库将考虑列的数据类型选择合适的索引。

ALTER
增加一列的长度是可以做到的。当在列式存储中尝试修改列的定义，不会返回错误，因为在数据库中没有做任何的检查。错误可能发生在选择列时，数据不符合新定义的数据类型。ALTER 仍未遵照数据类型转换规则。

增加 NOT NULL 约束至已存在的列是允许的，如果

表为空或者
表有数据时定义了默认值

<add_primary_key_clause> ::= ADD [CONSTRAINT <constraint_name>] PRIMARY KEY
(<column_name>, ...)
<constraint_name> ::= <identifier>

ADD PRIMARY KEY
增加一列主键。

PRIMARY KEY
主键约束是 NOT NULL 约束和 UNIQUE 约束的组合，其禁止多行在同一列中具有相同的值。

CONSTRAINT
指定约束名。

<drop_primary_key_clause> ::= DROP PRIMARY KEY

DROP PRIMARY KEY
删除主键约束。

<preload_clause> ::= PRELOAD ALL | PRELOAD (<column_name>) | PRELOAD NONE

PRELOAD
设置/移除给定表或列的预载标记。PRELOAD ALL 为表中所有列设置预载标记，PRELOAD (<column_name>)为指定的列设置标记，PRELOAD NONE 移除所有列的标记。其结果是这些表在索引服务器启动后自动加载至内存中。预载标记的当前状态在系统表 TABLES，列 PRELOAD 中可见，可能值为('FULL', 'PARTIALLY', 'NO')；在系统表 TABLE_COLUMNS，列 PRELOAD,可能值为('TRUE', 'FALSE')。

<table_conversion_clause> ::= [ALTER TYPE] {ROW [THREADS <number_of_threads>] | COLUMN [THREADS <number_of_threads> [BATCH <batch_size>]]}

ALTER TYPE ROW | COLUMN
该命令用于将表存储

从行转换为列或从列转换为行。

THREADS <number_of_threads>
指定进行表转换的并行线程数。线程数目的最佳值应设置为可用的 CPU 内核的数量。

Default：默认值为 param_sql_table_conversion_parallelism，即在 ndexserver.ini 文件中定义的 CPU 内核数。

BATCH <batch_size>
指定批插入的行数，默认值为最佳值 2,000,000。插入表操作在每个<batch_size>记录插入后立即提交，可以减少内存消耗。BATCH 可选项只可以在表 从行转换为列时使用。然而，大于 2,000,000 的批大小可能导致高内存消耗，因此不建议修改该值。

通过复制现有的表中的列和数据，可以从现有的表创建一个不同存储类型的新表。该命令用来把表从行转换为列或从列转换为行。如果源表是行式存储，则新建的表为列式存储。

<move_clause> ::= MOVE [PARTITION <partition_number>] TO [LOCATION]<host_port> [PHYSICAL] |

MOVE [PARTITION <partition_number>] PHYSICAL

MOVE 将表移动至分布式环境中的另一个位置。端口号是内部索引服务器的端口号，3xx03。

如果你有一个分区表，你可以通过指定可选的分区号只移动个别部分，移动分区表时，没有指定分区号会导致错误。

PHYSICAL 关键字只适用列式存储表。行式存储表总是物理移动。

如果指定了可选关键字 PHYSICAL，持久存储立即移动至目标主机。否则，此举将创建一个新主机里面的持久层链接指向旧主机持久层。该链接如果没有 TO<host_port>部分，将在下次合并或者移动中删除。

PHYSICAL 移动操作没有指定 TO <host_port>时，将移除从上次移动中仍然存在的持久层链接。

LOCATION 仅支持向后兼容

<add_range_partition_clause> ::= ADD <range_partition_clause>
<range_partition_clause> ::= PARTITION <lower_value> <= VALUES < <upper_value>
| PARTITION <value_or_values> = <target_value>
| PARTITION OTHERS
<lower_value> ::= <string_literal> | <numeric_literal>
<upper_value> ::= <string_literal> | <numeric_literal>
<target_value> ::= <string_literal> | <numeric_literal>

ADD PARTITION
为一个分区表添加分区，使用 RANGE, HASH RANGE, ROUNDROBIN RANGE 关键字。当添加分区至一张按范围分区的表时，如果需要的话，可以对其余分区进行重新分区。

<drop_range_partition_clause> ::= DROP <range_partition_clause>
<range_partition_clause> ::= PARTITION <lower_value> <= VALUES < <upper_value>
| PARTITION <value_or_values> = <target_value>
| PARTITION OTHERS
<lower_value> ::= <string_literal> | <numeric_literal>
<upper_value> ::= <string_literal> | <numeric_literal>
<target_value> ::= <string_literal> | <numeric_literal>

DROP PARTITION
删除根据 RANGE, HASH RANGE, ROUNDROBIN RANGE 分区的表的分区。

<partition_clause> ::= PARTITION BY <hash_partition> [,<range_partition> | ,<hash_partition>]
| PARTITION BY <range_partition>
| PARTITION BY <roundrobin_partition> [,<range_partition>]
<hash_partition> ::=HASH (<partition_expression>[, ...]) PARTITIONS { <num_partitions> | GET_NUM_SERVERS() }
<range_partition> ::= RANGE (<partition_expression>) (<range_spec>)
<roundrobin_partition> ::= ROUNDROBIN PARTITIONS {<num_partitions> |GET_NUM_SERVERS()}
<range_spec> ::= {<from_to_spec> | <single_spec>[,...] } [, PARTITION OTHERS]
<from_to_spec> ::= PARTITION <lower_value> <= VALUES < <upper_value>
<single_spec> ::= PARTITION VALUE <single_value>
<partition_expression> ::= <column_name>| YEAR(<column_name>) | MONTH(<column_name>)

PARTITION BY
使用 RANGE, HASH RANGE, ROUNDROBIN RANGE 对表进行分区。关于表分区自居，请参见 CREATE TABLE。

<merge_partition_clause> ::= MERGE PARTITIONS

MERGE PARTITIONS
合并分区表的所有部分至非分区表。

<disable_persistent_merge_clause> ::= DISABLE PERSISTENT MERGE

DISABLE PERSISTENT MERGE
指导合并管理器对于给定表，使用内存进行合并而非持久合并。

<enable_persistent_merge_clause> ::= ENABLE PERSISTENT MERGE

ENABLE PERSISTENT MERGE
指导合并管理器对于给定表使用持久合并。

<enable_delta_log> ::= ENABLE DELTA LOG
启动表日志记录。启用之后，你必须执行一个保存点以确保所有的数据都保存，并且你必须执行数据备份，否则你不能恢复这些数据。

<enable_delta_log> ::= DISABLE DELTA LOG

DISABLE DELTA LOG
禁用表日志记录。如果禁用，不会记录该表的日志。当完成一个保存点对于该表的修改只会写到数据域，这会导致当索引服务器终止时，提交的事务丢失。

仅在初次加载中使用该命令！

<enable_delta_log> ::= ENABLE AUTOMERGE
指导合并管理器处理表。

<enable_delta_log> ::= DISABLE AUTOMERGE

DISABLE AUTOMERGE
指导合并管理器忽略该表。

例子：

表 t 已创建，列 b 的默认值设为 10。

```
CREATE TABLE t (a INT, b INT);
ALTER TABLE t ALTER (b INT DEFAULT 10);
```

列 c 添加至表 t。

```
ALTER TABLE t ADD (c NVARCHAR(10) DEFAULT 'NCHAR');
```

创建表 t 的主键约束 prim_key。

```
ALTER TABLE t ADD CONSTRAINT prim_key PRIMARY KEY (a, b);
```

表 t 类型转换为列式 (COLUMN)。

```
ALTER TABLE t COLUMN;
```

设置列 b 和 c 的预载标记

```
ALTER TABLE t PRELOAD (b, c);
```

表 t 使用 RANGE 分区，并且添加另一分区。

```
ALTER TABLE t PARTITION BY RANGE (a) (PARTITION VALUE = 1, PARTITION OTHERS);
ALTER TABLE t ADD PARTITION 2 <= VALUES < 10;
```

表 t 的会话类型改为 HISTORY

```
ALTER TABLE t CREATE HISTORY;
```

禁用表 t 的日志记录。

```
ALTER TABLE t DISABLE DELTA LOG;
```

CREATE AUDIT POLICY

语法：

```
CREATE AUDIT POLICY <policy_name> AUDITING <audit_status_clause>
<audit_action_list> LEVEL <audit_level>
```

语法元素：

```
<audit_status_clause> ::= SUCCESSFUL | UNSUCCESSFUL | ALL

<audit_action_list> ::= <audit_action_name>[,<audit_action_name>]...

<audit_action_name> ::=
GRANT PRIVILEGE | REVOKE PRIVILEGE
| GRANT STRUCTURED PRIVILEGE | REVOKE STRUCTURED PRIVILEGE
| GRANT ROLE | REVOKE ROLE
| GRANT ANY | REVOKE ANY
| CREATE USER | DROP USER
| CREATE ROLE | DROP ROLE
| ENABLE AUDIT POLICY | DISABLE AUDIT POLICY
| CREATE STRUCTURED PRIVILEGE
| DROP STRUCTURED PRIVILEGE
| ALTER STRUCTURED PRIVILEGE
| CONNECT
| SYSTEM CONFIGURATION CHANGE

| SET SYSTEM LICENSE
| UNSET SYSTEM LICENSE

<audit_level> ::=
EMERGENCY
| ALERT
| CRITICAL
| WARNING
| INFO
```

描述：

CREATE AUDIT POLICY 语句创建新的审计策略。该审计策略可以稍后启动，并将导致指定的审计活动发生的审计。
只有拥有系统权限 AUDIT ADMIN 用户才可以新建审计策略。
指定的审计策略名必须和已有的审计策略名不同。
审计策略定义将被审计的审计活动。现有的审计策略需要被激活，从而进行审计。

<audit_status_clause>定义，成功或不成功或执行指定的审计活动进行审核。

以下的审计活动是可供使用的。它们被分在不同的组里。一组审计活动可以组合成一个审计策略，不同组的审计行动不能组合成审计策略。

GRANT PRIVILEGE	1	审计授予用户或角色的权限
REVOKE PRIVILEGE	1	审计撤销用户或角色的权限
GRANT STRUCTURED PRIVILEGE	1	审计授予用户的结构/分析权限
REVOKE STRUCTURED PRIVILEGE	1	审计撤销用户的结构/分析权限
GRANT ROLE	1	审计授予用户或角色的角色
REVOKE ROLE	1	审计撤销用户或角色的角色
GRANT ANY	1	审计授予用户或角色的权限、结构/分析权限或角色
REVOKE ANY	1	审计撤销用户或角色的权限、结构/分析权限或角色
CREATE USER	2	审计用户创建
DROP USER	2	审计用户删除
CREATE ROLE	2	审计角色创建
DROP ROLE	2	审计角色删除
CONNECT	3	审计连接到数据库的用户
SYSTEM CONFIGURATION CHANGE	4	审计系统配置的 更改(e.g. INIFILE)
ENABLE AUDIT POLICY	5	审计审核策略激活
DISABLE AUDIT POLICY	5	审计审核策略停用
CREATE STRUCTURED PRIVILEGE	6	审计结构化/分析权限创建
DROP STRUCTURED PRIVILEGE	6	审计结构化/分析权限删除

ALTER STRUCTURED PRIVILEGE	6	审计结构化/分析权限更改
SET SYSTEM LICENSE	7	审计许可证安装
UNSET SYSTEM LICENSE	7	审计许可证删除

每一个审计分配分配至审计级别，可能的值按照重要性递减有：
EMERGENCY, ALERT, CRITICAL, WARNING, INFO

为了使得审计活动发生，审计策略必须建立并启动，global_auditing_state（见下文）必须设置为 true。

配置参数：

目前，审计的配置参数存储在 global.ini，部分审计配置如下：

global_auditing_state ('true' / 'false')激活/关闭所有审计，无论有多少审计策略可用和启动。默认为 false，代表没有审计会发生。
default_audit_trail_type ('SYSLOGPROTOCOL' / 'CSVTEXTFILE')定义如何存储审计结果。
SYSLOGPROTOCOL 为默认值。
CSVTEXTFILE 应只用于测试目的。
default_audit_trail_path 指定 存储文件的位置，CSVTEXTFILE 已经选定的情况下。

对于所有的配置参数，可以在视图 M_INIFILE_CONTENTS 中选择，如果当前用户具有这样做所需的权限。但是目前这些参数只有在被显式设置后才可见。这意味着，它们将对新安装的数据库实例不可见。

系统和监控视图

AUDIT_POLICY：显式所有审计策略和其状态
M_INIFILE_CONTENTS：显示审计有关的配置参数

只有数据库用户具有 CATALOG READ, DATA ADMIN 或 INIFILE ADMIN 权限可以在视图 M_INIFILE_CONTENTS 查看任意信息，对于其他用户，该视图为空。

例子

新建的名为 priv_audit 的审计策略将审计有关成功授予和撤销权限和角色的命令，该审计策略有中等审计级别 CRITICAL.。
该策略必须显式启动(见 alter_audit_policy)，使得审计策略的审计发生。

CREATE AUDIT POLICY priv_audit AUDITING SUCCESSFUL GRANT PRIVILEGE, REVOKE PRIVILEGE, GRANT ROLE, REVOKE ROLE LEVEL CRITICAL;

CREATE FULLTEXT INDEX

语法：

CREATE FULLTEXT INDEX <index_name> ON <tableref> '(' <column_name> ')' [<fulltext_parameter_list>]

定义全文索引名：

<fulltext_parameter_list> ::= <fulltext_parameter> [, ...]

<fulltext_parameter> ::= LANGUAGE COLUMN <column_name>
| LANGUAGE DETECTION '(' <string_literal_list> ')'
| MIME TYPE COLUMN <column_name>
| <change_tracking_elem>
| FUZZY SEARCH INDEX <on_off>
| PHRASE INDEX RATIO <on_off>
| CONFIGURATION <string_literal>
| SEARCH ONLY <on_off>
| FAST PREPROCESS <on_off>

<on_off> ::= ON | OFF

LANGUAGE COLUMN
指定文件语言的列

LANGUAGE DETECTION
语言检测设置的语言集合。

MIME TYPE COLUMN
指定文件 mime-type 的列

FUZZY SEARCH INDEX
指定是否使用模糊搜索

PHRASE INDEX RATIO
指定**短语索引比率的百分比，值必须为 0.0 和 1.0 之间。**

CONFIGURATION
自定义配置文件的文本分析路径。

SEARCH ONLY
如果设为 ON，将不存储原始文件内容。

FAST PREPROCESS
如果设为 ON，将使用快速处理，例如语言搜索将不可用。

<change_tracking_elem> ::= SYNC[HRONOUS]| ASYNC[HRONOUS] [FLUSH [QUEUE]
<flush_queue_elem>]

SYNC
指定是否创建同步全文索引

ASYNC
指定是否创建异步全文索引

<flush_queue_elem> ::= EVERY <integer_literal> MINUTES
| AFTER <integer_literal> DOCUMENTS
| EVERY <integer_literal> MINUTES OR AFTER <integer_literal> DOCUMENTS

指定如果使用异步索引，更新全文索引的时机。

描述：

CREATE FULLTEXT INDEX 语句对给定表创建显式全文索引。

例子：

CREATE FULLTEXT INDEX i1 ON A(C) FUZZY SEARCH INDEX OFF
SYNC
LANGUAGE DETECTION ('EN','DE','KR')

上面的例子在表 A 的列 C 创建名为‘i1’的全文索引，未使用模糊搜索索引，语言检测设置的语言集合由'EN','DE' 和 'KR'组成。

CREATE INDEX
语法：

CREATE [UNIQUE] [BTREE | CPBTREE] INDEX <index_name> ON <table_name>
(<column_name_order>, ...) [ASC | DESC]

语法元素：

<index_name> ::= [<schema_name>.]<identifier>
<column_name_order> ::= <column_name> [ASC | DESC]

UNIQUE
用来创建唯一性索引。当创建索引和记录添加到表中将进行重复检查。

BTREE | CPBTREE
用来选择使用的索引类型。
为了使用 B+-树索引，必须使用 BTREE 关键字；对于 CPB+-树索引，需使用 CPBTREE 关键字。

B+-树是维护排序后的数据进行高效的插入、删除和搜索记录的树。

CPB+-树表示压缩前缀 B+-树，是基于 pkB-tree 树。CPB+树是一个非常小的索引，因为它使用“部分

键”，只是索引节点全部键的一部分。对于更大的键，CPB+-树展现出比 B+-树更好的性能。
如果省略了索引类型，SAP HANA 数据库将考虑列的数据类型选择合适的索引。

ASC | DESC
指定以递增或递减方式创建索引。
这些关键字只能在 btree 索引使用，并且对每一列只能使用一次。

描述：

CREATE INDEX 语句创建索引。

例子：

表 t 创建后，以递增方式对表 t 的 b 列创建 CBPTREE 索引 idx。

```
CREATE TABLE t (a INT, b NVARCHAR(10), c NVARCHAR(20));
CREATE INDEX idx ON t(b);
```

以递增方式对表 t 的 a 列创建 CBPTREE 索引 idx，以递减顺序对 b 列创建：

```
CREATE CPBTREE INDEX idx1 ON t(a, b DESC);
```

以递减方式对表 t 的 a 列和 c 列创建 CPBTREE 索引 idx2。

```
CREATE INDEX idx2 ON t(a, c) DESC;
```

以递增顺序对表 t 的 b 列和 c 列创建唯一性 CPBTREE 索引 idx3。

```
CREATE UNIQUE INDEX idx3 ON t(b, c);
```

以递增顺序对表 t 的 a 列创建唯一性 BTREE 索引 idx4。

```
CREATE UNIQUE INDEX idx4 ON t(a);
```

CREATE SCHEMA

语法：

```
CREATE SCHEMA <schema_name> [OWNED BY <user_name>]
```

语法元素：

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

OWNED BY
指定数据集合所有者名字。如果省略，当前用户将为数据集合所有者。

描述：

CREATE SCHEMA 语句在当前数据库创建数据集合。

例子：

CREATE SCHEMA my_schema OWNED BY system;

CREATE SEQUENCE
语法：

CREATE SEQUENCE <sequence_name> [<common_sequence_parameter_list>] [RESET BY <subquery>]

语法元素：

<sequence_name> ::= <identifier>
<common_sequence_parameter_list> ::= <common_sequence_parameter>, ...
<common_sequence_parameter> ::= <sequence_parameter_start_with>
| <basic_sequence_parameter>
<basic_sequence_parameter> ::= INCREMENT BY n
| MAXVALUE n
| NO MAXVALUE
| MINVALUE n
| NO MINVALUE
| CYCLE
| NO CYCLE
<sequence_parameter_start_with> ::= START WITH n

INCREMENT BY
定义上个分配值递增到下一个序列值的量，默认值为 1。指定一个负的值来生成一个递减的序列。INCREMENT BY 值为 0，将返回错误。

START WITH
定义起始序列值。如果未定义 START WITH 子句值，递增序列将使用 MINVALUE，递减序列将使用 MAXVALUE。
定义序列可生成的最大数值，必须为 0 至 4611686018427387903 之间。

NO MAXVALUE
使用 NO MAXVALUE 指令，递增序列的最大值将为 4611686018427387903，递减序列的最大值为-1。

MINVALUE
定义序列可生成的最小数值，必须为 0 至 4611686018427387903 之间。

NO MINVALUE
使用 NO MINVALUE 指令，递增序列的最小值将为 1，递减序列的最小值为-4611686018427387903。

CYCLE
使用 CYCLE 指令，序列在到达最大值或最小值后将会重新开始。

NO CYCLE
使用 NO CYCLE 指令，序列在到达最大值或最小值后将不会重新开始。

RESET BY
系统重启期间，系统自动执行 RESET BY 语句，并且将用 RESET BY 子查询确定的值重启序列。如果未指定 RESET BY，序列值将持久地存储在数据库中。在数据库重启过程中，序列的下一个值将由已保存的序列值生成。

描述：

CREATE SEQUENCE 语句用来创建序列。

序列用来为多个用户生成唯一整数。CURRVAL 用来获取序列的当前值，NEXTVAL 则用来获取序列的下一个值。CURRVAL 只有在会话中调用 NEXTVAL 才有效。

例子：

例子 1：
序列 seq 已创建，使用 CURRVAL 和 NEXTVAL 从序列中读取值。

```
CREATE SEQUENCE seq START WITH 11;
```

NEXTVAL 返回 11:

```
SELECT seq.NEXTVAL FROM DUMMY;
```

CURRVAL 返回 11:

```
SELECT seq.CURRVAL FROM DUMMY;
```

例子 2：

如果序列用来在表 R 的 A 列上创建唯一键，在数据库重启后，通过自动分配列 A 的最大值到序列，创建一个唯一键值，语句如下：

```
CREATE TABLE r (a INT);
CREATE SEQUENCE s RESET BY SELECT IFNULL(MAX(a), 0) + 1 FROM r;
SELECT s.NEXTVAL FROM DUMMY;
```

CREATE SYNONYM

语法：

```
CREATE [PUBLIC] SYNONYM <synonym_name> FOR <object_name>
```

语法元素：

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

<object_name> ::= <table_name>
| <view_name>
| <procedure_name>
| <sequence_name>
```

描述：

CREATE SYNONYM 为表、视图、存储过程或者序列创建备用名称。

你可以使用同义词把函数和存储过程重新指向不同的表、视图或者序列，而不需要重写函数或者过程。

可选项 PUBLIC 允许创建公用同义词。任何用户可以访问公用同义词，但只有拥有基本对象合适权限的用户可以访问基本对象。

例子：

```
CREATE SYNONYM a_synonym FOR a;
```

CREATE TABLE

语法：

```
CREATE [<table_type>] TABLE <table_name> <table_contents_source>
```

[<logging_option> | <auto_merge_option> | <partition_clause> | <location_clause>]

语法元素：

<table_name> ::= [<schema_name>.]<identifier>
<schema_name> ::= <identifier>

有关集合名字和标识符的说明，请参阅 Identifiers。

table_type:

<table_type> ::= COLUMN
| ROW
| HISTORY COLUMN
| GLOBAL TEMPORARY
| LOCAL TEMPORARY

ROW, COLUMN
如果大多数的访问是通过大量元组，而只选择少数几个属性，应使用基于列的存储。如果大多数访问选择一些记录的全部属性，使用基于行的存储是最好的。SAP HANA 数据库使用组合启用两种方式的的存储和解释。你可以为每张表指定组织类型，默认值为 ROW。

HISTORY COLUMN
利用特殊的事务会话类型称为'HISTORY'创建表。具有会话类型'HISTORY'的表支持"时间旅行"，对历史状态的数据库查询的执行是可能的。

时间旅行可以以如下方式完成：
会话级别时间旅行：
SET HISTORY SESSION TO UTCTIMESTAMP = <utc_timestamp>
SET HISTORY SESSION TO COMMIT ID = <commit_id>
<utc_timestamp> ::= <string_literal> <commit_id> ::= <unsigned_integer>

数据库会话可以设置回到一个特定时间点。该语句的 COMMIT ID 变量接受 commitid 作为参数。commitid 参数的值必须存在系统表 SYS.TRANSACTION_HISTORY 的 COMMIT_ID 列，否则将抛出异常。COMMIT_ID 在使用用户定义的快照时是有用的。用户自定义的快照可以通过存储在提交阶段分配至事务的 commitid 来获得。Commitid 可以通过在事务提交后执行以下查询来读取：

SELECT LAST_COMMIT_ID FROM M_TRANSACTIONS
WHERE CONNECTION_ID = CURRENT_CONNECTION;

该语句的 TIMESTAMP 变量接受时间戳作为参数。在系统内部，时间戳用来在系统表 SYS.TRANSACTION_HISTORY，commit_time 接近给定的时间戳的地方，查询一对

(commit_time,commit_id)，准确的说，选择最大 COMMIT_TIME 小于等于给定时间戳的对；如果没有找到这样的对，将会抛出异常。然后会话将使用 COMMIT_ID 变量确定的 commit-id 恢复。要终止恢复会话切换到当前会话中，必须在数据库连接上执行明确的 COMMIT 或 ROLLBACK。

语句级别时间旅行：

<subquery> AS OF UTCTIMESTAMP <utc_timestamp>
<subquery> AS OF COMMIT ID <commit_id>

为了能使 commitid 与提交时间关联，需维护系统表 SYS.TRANSACTION_HISTORY，存储每个为历史表提交数据的事务的额外信息。有关设置会话级别时间旅行的详细信息，请参阅 SET HISTORY SESSION，关于<subquery>的信息，请参阅 Subquery。

注意：

当会话恢复时，自动提交必须关闭（否则会抛出包含相应错误消息的异常）。

非历史表在恢复会话中总显示其当前快照。

只有数据查询语句(select)可以在恢复会话中使用。

历史表必须有主键。

会话类型可以通过系统表 SYS.TABLES 的 SESSION_TYPE 列检查。

GLOBAL TEMPORARY

表定义全局可见，但数据只在当前会话中可见。表在会话结束后截断。

全局临时表的元数据是持久的，表示该元数据一直存在直到表被删除，元数据在会话间共享。临时表中的数据是会话特定的，代表只有全局临时表的所有者才允许插入、读取、删除数据，存在于会话持续期间，并且当会话结束时，全局临时表中的数据会自动删除。全局临时表只有当表中没有任何数据才能被删除。

全局临时表支持的操作：

1. Create without a primary key
2. Rename table
3. Rename column
4. Truncate
5. Drop
6. Create or Drop view on top of global temporary table
7. Create synonym
8. Select

9. Select into or Insert
10. Delete
11. Update
12. Upsert or Replace

LOCAL TEMPORARY
表的定义和数据只在当前会话可见，该表在会话结束时被截断。

元数据在会话间共享，并且是会话特定的，代表只有本地临时表的所有者才允许查看。临时表中的数据是会话特定的，代表只有本地临时表的所有者才允许插入、读取、删除数据，存在于会话持续期间，并且当会话结束时，本地临时表中的数据会自动删除。

本地临时表支持的操作：
1. Create without a primary key
2. Truncate
3. Drop
4. Select
5. Select into or Insert
6. Delete
7. Update
8. Upsert or Replace

table_contents_source:

<table_contents_source> ::= (<table_element>, ...)
| <like_table_clause> [WITH [NO] DATA]
| [(<column_name>, ...)] <as_table_subquery> [WITH [NO] DATA]]

<table_element> ::= <column_definition> [<column_constraint>]
| <table_constraint> (<column_name>, ...)

<column_definition> ::= <column_name> <data_type> [<column_store_data_type>] [<ddic_data_type>] [DEFAULT <default_value>] [GENERATED ALWAYS AS <expression>]

<column_name> ::= <identifier>

<data_type> ::= DATE | TIME | SECONDDATE | TIMESTAMP | TINYINT | SMALLINT | INTEGER | BIGINT | SMALLDECIMAL | DECIMAL | REAL | DOUBLE| VARCHAR | NVARCHAR | ALPHANUM | SHORTTEXT | VARBINARY |BLOB | CLOB | NCLOB | TEXT

<column_store_data_type> ::= CS_ALPHANUM | CS_INT | CS_FIXED | CS_FLOAT| CS_DOUBLE | CS_DECIMAL_FLOAT | CS_FIXED(p-s, s) | CS_SDFLOAT| CS_STRING | CS_UNITEDECFLOAT | CS_DATE |

CS_TIME | CS_FIXEDSTRING | CS_RAW | CS_DAYDATE | CS_SECONDTIME | CS_LONGDATE | CS_SECONDDATE

<ddic_data_type> ::= DDIC_ACCP | DDIC_ALNM | DDIC_CHAR | DDIC_CDAY | DDIC_CLNT | DDIC_CUKY | DDIC_CURR | DDIC_D16D | DDIC_D34D | DDIC_D16R | DDIC_D34R | DDIC_D16S | DDIC_D34S | DDIC_DATS | DDIC_DAY | DDIC_DEC | DDIC_FLTP | DDIC_GUID | DDIC_INT1 | DDIC_INT2 | DDIC_INT4 | DDIC_INT8 | DDIC_LANG | DDIC_LCHR | DDIC_MIN | DDIC_MON| DDIC_LRAW | DDIC_NUMC | DDIC_PREC | DDIC_QUAN | DDIC_RAW | DDIC_RSTR | DDIC_SEC | DDIC_SRST | DDIC_SSTR | DDIC_STRG | DDIC_STXT | DDIC_TIMS | DDIC_UNIT| DDIC_UTCM | DDIC_UTCL | DDIC_UTCS | DDIC_TEXT | DDIC_VARC | DDIC_WEEK

<default_value> ::= NULL | <string_literal> | <signed_numeric_literal>| <unsigned_numeric_literal>

<signed_numeric_literal> ::= [<sign>] <unsigned_numeric_literal>

<sign> ::= + | -

<unsigned_numeric_literal> ::= <exact_numeric_literal> | <approximate_numeric_literal>

<exact_numeric_literal> ::= <unsigned_integer> [<period> [<unsigned_integer>]]
| <period> <unsigned_integer>

<period> ::= .

<approximate_numeric_literal> ::= <mantissa> E <exponent>

<mantissa> ::= <exact_numeric_literal>

<exponent> ::= <signed_integer>

<signed_integer> ::= [<sign>] <unsigned_integer>

<unsigned_integer> ::= <digit>...

DEFAULT
DEFAULT 定义了 INSERT 语句没有为列提供值情况下，默认分配的值。

列定义中的 DATA TYPE

可供使用的数据类型有：DATE, TIME, SECONDDATE, TIMESTAMP, TINYINT, SMALLINT, INTEGER,BIGINT,SMALLDECIMAL, DECIMAL, REAL, DOUBLE, VARCHAR, NVARCHAR, ALPHANUM, SHORTTEXT,VARBINARY,BLOB, CLOB, NCLOB 和 TEXT。

可供使用的列式存储数据类型有：CS_ALPHANUM, CS_INT, CS_FIXED, CS_FLOAT,CS_DOUBLE,CS_DECIMAL_FLOAT, CS_FIXED(p-s,s), CS_SDFLOAT, CS_STRING,

CS_UNITEDECFLOAT,CS_DATE, CS_TIME,CS_FIXEDSTRING, CS_RAW, CS_DAYDATE, CS_SECONDTIME, CS_LONGDATE, 和 CS_SECONDDATE。

可供使用的 DDIC 数据类型有：DDIC_ACCP, DDIC_ALNM, DDIC_CHAR, DDIC_CDAY, DDIC_CLNT, DDIC_CUKY,DDIC_CURR, DDIC_D16D, DDIC_D34D, DDIC_D16R, DDIC_D34R, DDIC_D16S, DDIC_D34S, DDIC_DATS,DDIC_DAY, DDIC_DEC, DDIC_FLTP, DDIC_GUID, DDIC_INT1, DDIC_INT2, DDIC_INT4, DDIC_INT8,DDIC_LANG, DDIC_LCHR, DDIC_MIN, DDIC_MON, DDIC_LRAW, DDIC_NUMC, DDIC_PREC, DDIC_QUAN,DDIC_RAW, DDIC_RSTR, DDIC_SEC, DDIC_SRST, DDIC_SSTR, DDIC_STRG, DDIC_STXT, DDIC_TIMS,DDIC_UNIT, DDIC_UTCM, DDIC_UTCL, DDIC_UTCS, DDIC_TEXT, DDIC_VARC, DDIC_WEEK。

GENERATED ALWAYS AS
指定在运行时生成的列值的表达式。

<column_constraint> ::= NULL
| NOT NULL
| UNIQUE [BTREE | CPBTREE]
| PRIMARY KEY [BTREE | CPBTREE]

NULL | NOT NULL

NOT NULL 禁止列的值为 NULL。

如果指定了 NULL，将不被认为是常量，其表示一列可能含有空值，默认为 NULL。

UNIQUE
指定为唯一键的列。

一个唯一的复合键指定多个列作为唯一键。有了 unique 约束，多行不能在同一列中具有相同的值。

PRIMARY KEY
主键约束是 NOT NULL 约束和 UNIQUE 约束的组合，其禁止多行在同一列中具有相同的值。

BTREE | CPBTREE
指定索引类型。当列的数据类型为字符串、二进制字符串、十进制数或者约束是一个组合键，或非唯一键，默认的索引类型为 CPBTREE，否则使用 BTREE。

为了使用 B+-树索引，必须使用 BTREE 关键字；对于 CPB+-树索引，需使用 CPBTREE 关键字。

B+-树是维护排序后的数据进行高效的插入、删除和搜索记录的树。

CPB+-树表示压缩前缀 B+-树，是基于 pkB-tree 树。CPB+树是一个非常小的索引，因为它使用“部分

键”，只是索引节点全部键的一部分。对于更大的键，CPB+-树展现出比 B+-树更好的性能。
如果省略了索引类型，SAP HANA 数据库将考虑列的数据类型选择合适的索引。

<table_constraint> ::= UNIQUE [BTREE | CPBTREE]| PRIMARY KEY [BTREE | CPBTREE]

定义了表约束可以使用在表的一列或者多列上。有两种表约束，它们是：

<like_table_clause> ::= LIKE <like_table_name>
<like_table_name> ::= <table_name>

创建与 like_table_name 定义相同的表。表中所有列的定义和默认值拷贝自 like_table_name。当提供了可选项 WITH DATA，数据将从指定的表填充，不过，默认值为 WITH NO DATA。

<as_table_subquery> ::= AS '(<subquery>)

创建表并且用<subquery>计算的数据填充。使用该子句只拷贝 NOT NULL 约束。如果指定了 column_names，该指定的 column_names 将覆盖<subquery>中的列名。

WITH [NO] DATA 指定数据拷贝自<subquery> 或 <like_table_clause>。

<logging_option> ::= LOGGING| NO LOGGING [RETENTION <retention_period>]
<retention_period> ::= <unsigned_integer>

LOGGING | NO LOGGING
LOGGING (默认值)指定激活记录表日志。
NO LOGGING 指定停用记录表日志。一张 NO LOGGING 表意味表定义是持久的且全局可见的，数据则为临时和全局的。

RETENTION
指定以秒为单位，NO LOGGING 列式表的保留时间。在指定的保留期限已过，如果使用物理内存的主机达到 80%，上述表将被删除。

<auto_merge_option> ::= AUTO MERGE | NO AUTO MERGE

AUTO MERGE | NO AUTO MERGE
AUTO MERGE (默认值)指定触发自动增量合并。

<partition_clause> ::= PARTITION BY <hash_partition> [, <range_partition> | , <hash_partition>]
| PARTITION BY <range_partition>
| PARTITION BY <roundrobin_partition> [,<range_partition>]
<hash_partition> ::= HASH (<partition_expression> [, ...]) PARTITIONS {<num_partitions> |

GET_NUM_SERVERS()}
<range_partition> ::= RANGE (<partition_expression>) (<range_spec>, ...)

<roundrobin_partition> ::= ROUNDROBIN PARTITIONS {<num_partitions> | GET_NUM_SERVERS()} [, <range_partition>]
<range_spec> ::= {<from_to_spec> | <single_spec>} [, ...] [, PARTITION OTHERS]
<from_to_spec> ::= PARTITION <lower_value> <= VALUES < <upper_value>
<single_spec> ::= PARTITION VALUE <target_value>
<partition_expression> ::= <column_name> | YEAR(<column_name>) | MONTH(<column_name>)
<lower_value> ::= <string_literal> | <numeric_literal>
<upper_value> ::= <string_literal> | <numeric_literal>
<target_value> ::= <string_literal> | <numeric_literal>
<num_partitions> ::= <unsigned_integer>

GET_NUM_SERVERS()函数返回服务器数量。

PARTITION OTHERS 表示分区定义中未指定的其余值成为一个分区。

确定分区创建所在的索引服务器是可能的。如果你指定了 LOCATION，在这些实例中循环创建分区。列表中的重复项将被移除。如果你在分区定义中精确指定实例数为分区数，则每个分区将分配至列表中各自实例。所有索引列表中的服务器必须属于同一个实例。
如果指定了 no location，将随机创建分区。如果分区数与服务器数匹配-例如使用 GET_NUM_SERVERS()-可以确保多个 CREATE TABLE 调用以同样的方式分配分区。对于多级别分区的情况，其适用于第一级的分区数。这个机制是很有用的，如果创建彼此语义相关的多张表。

<location_clause> ::= AT [LOCATION] {'<host>:<port>' | ('<host>:<port>', ...)}
<host> ::= <string_literal>
<port> ::= <unsigned_integer>

AT LOCATION
表可以创建在 host:port 指定的位置，位置列表可以在创建分配至多个实例的分区表时定义。当位置列表没有提供<partition_clause>，表将创建至指定的第一个位置中。
如果没有提供位置信息，表将自动分配到一个节点中。此选项可用于在分布式环境中的行存储和列存储表

描述：

CREATE TABLE 创建一张表。表中未填充数据，除了当<as_table_subquery> 或 <like_table_clause>和 WITH DATA 可选项一起使用。

例子：

创建了表 A，整数列 A 和 B。列 A 具有主键约束。

CREATE TABLE A (A INT PRIMARY KEY, B INT);

创建了分区表 P1，日期列 U。列 U 具有主键约束并且作为 RANGE 分区列使用。

CREATE COLUMN TABLE P1 (U DATE PRIMARY KEY) PARTITION BY RANGE (U) (PARTITION '2010-02-03' <= VALUES < '2011-01-01', PARTITION VALUE = '2011-05-01');

创建了分区表 P2，整数列 I，J 和 K。列 I，J 组成键约束，并且作为哈希分区列使用。列 K 则为子哈希分区列。

CREATE COLUMN TABLE P2 (I INT, J INT, K INT, PRIMARY KEY(I, J)) PARTITION BY HASH (I, J) PARTITIONS 2, HASH (K) PARTITIONS 2;

创建表 C1，与表 A 的定义相同，记录也相同。

CREATE COLUMN TABLE C1 LIKE A WITH DATA;

创建表 C2，与表 A 的列数据类型和 NOT NULL 约束相同。表 C2 没有任何数据。

CREATE TABLE C2 AS (SELECT * FROM A) WITH NO DATA;

CREATE TRIGGER

语法：

```
CREATE TRIGGER <trigger_name> <trigger_action_time> <trigger_event>
ON <subject_table_name> [REFERENCING <transition_list>][<for_each_row>]
BEGIN
[<trigger_decl_list>]
[<proc_handler_list>]
<trigger_stmt_list>
END
```

语法元素：

<trigger_name> ::= <identifier>

你创建的触发器名称。

<subject_table_name> ::= <identifier>

你创建的触发器所在表的名称。

关于 identifier **更多的信息，参见** Identifiers。

<trigger_action_time> ::= BEFORE | AFTER

指定触发动作发生的 时间。

BEFORE

操作主题表之前执行触发。

AFTER
操作主题表之后执行触发。

<trigger_event> ::= INSERT | DELETE | UPDATE

定义激活触发器活动的数据修改命令

<transition_list> ::= <transition> | <transition_list> , <transition>
<transition> ::= <trigger_transition_old_or_new> <trigger_transition_var_or_table> <trans_var_name>|
<trigger_transition_old_or_new> <trigger_transition_var_or_table> AS <trans_var_name>

当声明了 触发器转变变量，触发器可以访问 DML 正在修改的记录。
当执行了行级别触发器，<trans_var_name>.<column_name>代表触发器正在修改的记录的相应列。
这里，<column_name>为主题表的列名。参见转换变量的例子。

<trigger_transition_old_or_new> ::= OLD | NEW
<trigger_transition_var_or_table> ::= ROW
<trans_var_name> ::= <identifier>

OLD

你可以访问触发器 DML **的旧记录**，取而代之的是更新的旧记录或删除的旧记录。

UPDATE 触发器和 DELETE 触发器可以有 OLD ROW 事务变量。

NEW

你可以访问触发器 DML **的新记录**，取而代之的是插入的新记录或更新的新记录。

UPDATE 触发器和 DELETE 触发器可以有 NEW ROW 事务变量。

只支持事务变量。
事务表不被支持。
如果你将’TABLE’作为<trigger_transition_var_or_table>，你将看到不支持功能错误。

<for_each_row> ::= FOR EACH ROW

触发器将以逐行方式调用。
默认模式为执行行级别触发器，没有 FOR EACH ROW 语法。
目前，逐语句触发不被支持。

<trigger_decl_list> ::= DECLARE <trigger_decl>| <trigger_decl_list> DECLARE <trigger_decl>

<trigger_decl> ::= <trigger_var_decl> | <trigger_condition_decl>

<trigger_var_decl> ::= <var_name> CONSTANT <data_type> [<not_null>] [<trigger_default_assign>] ;
| <var_name> <data_type> [<not_null>] [<trigger_default_assign>] ;

<var_name> ::= <identifer>

<data_type> ::= DATE | TIME | SECONDDATE | TIMESTAMP | TINYINT | SMALLINT | INTEGER| BIGINT | SMALLDECIMAL | DECIMAL | REAL | DOUBLE| VARCHAR | NVARCHAR | ALPHANUM | SHORTTEXT |VARBINARY | BLOB | CLOB| NCLOB | TEXT

<not_null> ::= NOT NULL

<trigger_default_assign> ::= DEFAULT <expression>| := <expression>

<trigger_condition_decl> ::= <condition_name> CONDITION ;| <condition_name> CONDITION FOR <sql_error_code> ;

<condition_name> ::= <identifier>

<sql_error_code> ::= SQL_ERROR_CODE <int_const>

trigger_decl_list
你可以声明触发器变量或者条件。
声明的变量可以在标量赋值中使用或者在 SQL 语句触发中引用。
声明的条件名可以在异常处理中引用。

CONSTANT
当指定了 CONSTANT 关键字，你不可以在触发器执行时修改变量。

<proc_handler_list> ::= <proc_handler>
| <proc_handler_list> <proc_handler>

<proc_handler> ::= DECLARE EXIT HANDLER FOR <proc_condition_value_list> <trigger_stmt>

<proc_condition_value_list> ::= <proc_condition_value>
| <proc_condition_value_list> , <proc_condition_value>

<proc_condition_value> ::= SQLEXCEPTION
| SQLWARNING
| <sql_error_code>
| <condition_name>

异常处理可以声明捕捉已存的 SQL 异常，特定的错误代码或者条件变量声明的条件名称。

<trigger_stmt_list> ::= <trigger_stmt>| <trigger_stmt_list> <trigger_stmt>

<trigger_stmt> ::= <proc_block>
| <proc_assign>
| <proc_if>
| <proc_loop>
| <proc_while>
| <proc_for>
| <proc_foreach>
| <proc_signal>
| <proc_resignal>
| <trigger_sql>

触发器主体的语法是过程体的语法的一部分。
参见 SAP HANA Database SQLScript guide 中的 create procedure 定义。
触发器主体的语法遵循过程体的语法，即嵌套块(proc_block)，标量变量分配(proc_assign)， if 块(proc_if),， loop 块(proc_loop)，for 块(proc_for)，for each 块(proc_foreach)，exception signal(proc_signal)，exception resignal(proc_resignal)，和 sql 语句(proc_sql)。

<proc_block> ::= BEGIN
[<trigger_decl_list>]
[<proc_handler_list>]
<trigger_stmt_list>
END ;

你可以以嵌套方式添加另外的'BEGIN ... END;'块。

<proc_assign> ::= <var_name> := <expression> ;

var_name 为变量名，应该事先声明。

<proc_if> ::= IF <condition> THEN <trigger_stmt_list>
[<proc_elsif_list>]
[<proc_else>]
END IF ;

<proc_elsif_list> ::= ELSEIF <condition> THEN <trigger_stmt_list>

<proc_else> ::= ELSE <trigger_stmt_list>

关于 condition 的说明，参见 SELECT 的<condition>。

你可以使用 IF ... THEN ... ELSEIF... END IF 控制执行流程与条件。

<proc_loop> ::= LOOP <trigger_stmt_list> END LOOP ;

<proc_while> ::= WHILE <condition> DO <trigger_stmt_list> END WHILE ;

<proc_for> ::= FOR <column_name> IN [<reverse>] <expression> <DDOT_OP> <expression>

DO <trigger_stmt_list>
END FOR ;

<column_name> ::= <identifier>

<reverse> ::= REVERSE
<DDOT_OP> ::= ..

<proc_foreach> ::= FOR <column_name> AS <column_name> [<open_param_list>] DO
<trigger_stmt_list>
END FOR ;

<open_param_list> ::= (<expr_list>)

<expr_list> ::= <expression> | <expr_list> , <expression>

<proc_signal> ::= SIGNAL <signal_value> [<set_signal_info>] ;

<proc_resignal> ::= RESIGNAL [<signal_value>] [<set_signal_info>] ;

<signal_value> ::= <signal_name> | <sql_error_code>

<signal_name> ::= <identifier>

<set_signal_info> ::= SET MESSEGE_TEXT = '<message_string>'

<message_string> ::= <identifier>

SET MESSEGE_TEXT
如果你为自己的消息设置 SET MESSEGE_TEXT，当触发器执行时指定的错误被抛出，消息传递给用户。

SIGNAL 语句显式抛出一个异常。
用户定义的范围（10000一19999）将被允许发行错误代码。
RESIGNAL 语句在异常处理中对活动语句抛出异常。
如果没有指定错误代码，RESIGNAL 将抛出已捕捉的异常。

<trigger_sql> ::= <select_into_stmt>
| <insert_stmt>
| <delete_stmt>
| <update_stmt>
| <replace_stmt>
| <upsert_stmt>

对于 insert_stmt 的详情，参见 INSERT。
对于 delete_stmt 的详情，参见 DELETE。
对于 update_stmt 的详情，参见 UPDATE。
对于 replace_stmt 和 upsert_stmt 的详情，参见 REPLACE | UPSERT。

<select_into_stmt> ::= SELECT <select_list> INTO <var_name_list>
<from_clause >
[<where_clause>]
[<group_by_clause>]
[<having_clause>]
[{<set_operator> <subquery>, ... }]
[<order_by_clause>]
[<limit>]

<var_name_list> ::= <var_name> | <var_name_list> , <var_name>

<var_name> ::= <identifier>

关于 select_list, from_clause, where_clause, group_by_clause, having_clause,set_operator, subquery, order_by_clause, limit 的详情，参见 SELECT。

var_name 是预先声明的标量变量名。
你可以分配选中项至标量变量中。

描述：

CREATE TRIGGER 语句创建触发器。
触发器是一种特殊的存储过程，在对表发生事件时自动执行。
CREATE TRIGGER 命令定义一组语句，当给定操作(INSERT/UPDATE/DELETE)发生在给定对象(主题表)上执行。

只有对于给定的<subject_table_name>拥有 TRIGGER 权限的数据库用户允许创建表的触发器。
当前触发器限制描述如下：

- 不支持 INSTEAD_OF 触发器。
- 访问触发器定义的主题表在触发器主体中是不允许的，这代表对于触发器所在的表的任何 insert/update/delete/replace/select 操作是不支持的。

- 只支持行级别触发器，不支持语句级别触发器。行级别触发器表示触发活动只有在每次行改变时执行。语句级别触发器代表触发活动在每次语句执行时运行。语法'FOR EACH ROW'表示行式执行触发，为默认模式；即时没有指定'FOR EACH ROW'，仍然为行级别触发器。
- 不支持事务表(OLD/NEW TABLE)。当触发 SQL 语句想要引用正在被触发器触发事件如 insert/update/delete 修改的数据，事务变量/表将是触发器主体中 SQL 语句访问新数据和旧数据的方式。事务变量只在行级别触发器中使用，而事务表则在语句级别触发器中使用。
- 不支持从节点到多个主机或表中的分区表上执行触发器。
- 表只能为每个 DML 操作有一个触发器，可能是插入触发器、更新触发器和删除触发器，并且它们三个可以一起激活。
  因此，一张表总共最多有三个触发器。
- 不支持的触发器动作（而存储过程支持）：

resultset assignment(select resultset assignment to tabletype),
exit/continue command(execution flow control),
cursor open/fetch/close(get each record data of search result by cursor and access record in loop),
procedure call(call another proecedure),
dynomic sql execution(build SQL statements dynamically at runtime of SQLScript),
return(end SQL statement execution)

系统和监控视图：

TRIGGER 为触发器的系统视图：

系统视图 TRIGGER 显示：

SCHEMA_NAME, TRIGGER_NAME, TRIGGER_OID, OWNER_NAME,
OWNER_OID,SUBJECT_TABLE_SCHEMA,SUBJECT_TABLE_NAME, TRIGGER_ACTION_TIME,
TRIGGER_EVENT, TRIGGERED_ACTION_LEVEL,DEFINITION

例子：

你先需要触发器定义的表：

```
CREATE TABLE TARGET ( A INT);
```

你也需要表触发访问和修改：

```
CREATE TABLE SAMPLE ( A INT);
```

以下为创建触发器的**例子：**

```
CREATE TRIGGER TEST_TRIGGER
AFTER INSERT ON TARGET FOR EACH ROW
BEGIN
DECLARE SAMPLE_COUNT INT;
```

```
SELECT COUNT(*) INTO SAMPLE_COUNT FROM SAMPLE;
IF :SAMPLE_COUNT = 0
THEN
INSERT INTO SAMPLE VALUES(5);
ELSEIF :SAMPLE_COUNT = 1
THEN
INSERT INTO SAMPLE VALUES(6);
END IF;
END;
```

触发器 TEST_TRIGGER 将在任意记录插入至 TARGET 表后执行。
由于在第一次插入中，表 SAMPLE 记录数为 0，触发器 TEST_TRIGGER 将插入 5 至表中。
在第二次插入 TARGET 表时，触发器插入 6，因为其计数为 2。

```
INSERT INTO TARGET VALUES (1);
SELECT * FROM SAMPLE;
5

INSERT INTO TARGET VALUES (2);
SELECT * FROM SAMPLE;
5
6
```

以下是创建触发器的更多例子。FOR/WHILE **例子：**

```
CREATE TABLE TARGET ( A INT);
CREATE TABLE SAMPLE ( A INT);
CREATE TRIGGER TEST_TRIGGER_WHILE_UPDATE
AFTER UPDATE ON TARGET
BEGIN
DECLARE found INT := 1;
DECLARE val INT := 1;
WHILE :found <> 0 DO
SELECT count(*) INTO found FROM sample WHERE p = :val;
IF :found = 0 THEN
INSERT INTO sample VALUES(:val,100000);
END IF;
val := :val + 1;
END WHILE;
END;
CREATE TABLE TARGET ( A INT);
CREATE TABLE control_tab(id INT PRIMARY KEY, name VARCHAR(30), payment INT);
CREATE TABLE message_box(message VARCHAR(200), log_time TIMESTAMP);
```

```
CREATE TRIGGER TEST_TRIGGER_FOR_INSERT
AFTER INSERT ON TARGET
BEGIN
DECLARE v_id INT := 0;
DECLARE v_name VARCHAR(20) := '';
DECLARE v_pay INT := 0;
DECLARE v_msg VARCHAR(200) := '';
DELETE FROM message_box;
FOR v_id IN 100 .. 103 DO
SELECT name, payment INTO v_name, v_pay FROM control_tab WHERE id = :v_id;
v_msg := :v_name || ' has ' || TO_CHAR(:v_pay);
INSERT INTO message_box VALUES (:v_msg, CURRENT_TIMESTAMP);
END FOR;
END;
```

Handler 例子：

```
CREATE TABLE TARGET ( A INT);
CREATE TABLE MYTAB (I INTEGER PRIMARY KEY);
CREATE TRIGGER MYTRIG_SQLEXCEPTION
AFTER INSERT ON TARGET
BEGIN
DECLARE EXIT HANDLER FOR SQLEXCEPTION SELECT ::SQL_ERROR_CODE, ::SQL_ERROR_MESS
AGE FROM DUMMY;
INSERT INTO MYTAB VALUES (1);
INSERT INTO MYTAB VALUES (1); -- expected unique violation error: 301
-- not reached
END;
CREATE TRIGGER MYTRIG_SQL_ERROR_CODE
AFTER UPDATE ON TARGET
BEGIN
DECLARE EXIT HANDLER FOR SQL_ERROR_CODE 301 SELECT ::SQL_ERROR_CODE, ::SQL_ERRO
R_MESSAGE FROM DUMMY;
INSERT INTO MYTAB VALUES (1);
INSERT INTO MYTAB VALUES (1); -- expected unique violation error: 301
-- not reached
END;
CREATE TRIGGER MYTRIG_CONDITION
AFTER DELETE ON TARGET
BEGIN
DECLARE MYCOND CONDITION FOR SQL_ERROR_CODE 301;
DECLARE EXIT HANDLER FOR MYCOND SELECT ::SQL_ERROR_CODE, ::SQL_ERROR_MESSAGE FR
OM DUMMY;
INSERT INTO MYTAB VALUES (1);
INSERT INTO MYTAB VALUES (1); -- expected unique violation error: 301
-- not reached
```

```
END;
```

SIGNAL/RESIGNAL

```
CREATE TABLE TARGET ( A INT);
CREATE TABLE MYTAB (I INTEGER PRIMARY KEY);
CREATE TRIGGER MYTRIG_SIGNAL
AFTER INSERT ON TARGET
BEGIN
DECLARE MYCOND CONDITION FOR SQL_ERROR_CODE 10001;
DECLARE EXIT HANDLER FOR MYCOND SELECT ::SQL_ERROR_CODE, ::SQL_ERROR_MESSAGE FR
OM DUMMY;
INSERT INTO MYTAB VALUES (1);
SIGNAL MYCOND SET MESSAGE_TEXT = 'my error';
-- not reached
END;
CREATE TRIGGER MYTRIG_RESIGNAL
AFTER UPDATE ON TARGET
BEGIN
DECLARE MYCOND CONDITION FOR SQL_ERROR_CODE 10001;
DECLARE EXIT HANDLER FOR MYCOND RESIGNAL;
INSERT INTO MYTAB VALUES (1);
SIGNAL MYCOND SET MESSAGE_TEXT = 'my error';
-- not reached
END;
```

事务变量**例子：**

```
CREATE TABLE TARGET ( A INT, B VARCHAR(10));
CREATE TABLE SAMPLE_OLD ( A INT, B VARCHAR(10));
CREATE TABLE SAMPLE_NEW ( A INT, B VARCHAR(10));
INSERT INTO TARGET VALUES ( 1, 'oldvalue');
CREATE TRIGGER TEST_TRIGGER_VAR_UPDATE
AFTER UPDATE ON TARGET
REFERENCING NEW ROW mynewrow, OLD ROW myoldrow
FOR EACH ROW
BEGIN
INSERT INTO SAMPLE_new VALUES(:mynewrow.a, :mynewrow.b)
INSERT INTO SAMPLE_old VALUES(:myoldrow.a, :myoldrow.b)
END;
UPDATE TARGER SET b = 'newvalue' WHERE A = 1;
SELECT * FROM SAMPLE_NEW;
1, 'newvalue'
SELECT * FROM SAMPLE_OLD;
1, 'oldvalue'
```

CREATE VIEW

语法：

CREATE VIEW <view_name> [(<column_name>, ...)] AS <subquery>

语法元素：

<view_name> ::= [<schema_name>.]<identifier>
<schema_name> ::= <identifier>
<column_name> ::= <identifier>

描述：

CREATE VIEW 可以有效地根据 SQL 结果创建虚拟表，这不是真正意义上的表，因为它本身不包含数据。

当列名一起作为视图的名称，查询结果将以列名显示。如果列名被省略，查询结果自动给出一个适当的列名。列名的数目必须与从<subquery>返回的列数目相同。

支持视图的更新操作，如果满足以下条件：

视图中的每一列必须映射到单个表中的一列。
如果基表中的一列有 NOT NULL 约束，且没有默认值，该列必须包含在可插入视图的列中。更新操作没有该条件。
例如在 SELECT 列表，必须不包含聚合或分析的功能，以下是不允许的：
. 在 SELECT 列表中的 TOP, SET, DISTINCT 操作
. GROUP BY, ORDER BY 子句
在 SELECT 列表中必须不能含有子查询。
必须不能包含序列值(CURRVAL, NEXTVAL)。
必须不能包含作为基视图的列视图。

如果基视图或表是可更新的，符合上述条件的基础上，基视图或表的视图是可更新的。

例子：

选择表 a 中的所有数据创建视图 v：

CREATE VIEW v AS SELECT * FROM a;

DROP AUDIT POLICY

语法：

DROP AUDIT POLICY <policy_name>

语法元素：

<policy_name> ::= <identifier>

描述：

DROP AUDIT POLICY 语句删除审计策略。<policy_name>必须指定已存在的审计策略。

只有拥有系统权限 AUDIT ADMIN 的数据库用户允许删除审计策略。每个拥有该权限的数据库用户可以删除任意的审计策略，无论是否由该用户创建。

即使审计策略被删除了，可能发生的是，定义在已删除的审计策略中的活动将被进一步审计；如果也启用了其他审计策略和定义了审计活动。

暂时关闭审计策略时，可以禁用而不用删除。

系统和监控视图：

AUDIT_POLICY：显示所有审计策略和状态。

M_INIFILE_CONTENTS：显示数据库系统配置参数。

只有拥有系统权限 CATALOG READ, DATA ADMIN 或 INIFILE ADMIN 的用户可以查看 M_INIFILE_CONTENTS 视图中的内容，对于其他用户则为空。

例子：

假设使用如下语句前，审计策略已创建：

```
CREATE AUDIT POLICY priv_audit AUDITING SUCCESSFUL GRANT PRIVILEGE, REVOKE PRIVILEGE, GRANT ROLE, REVOKE ROLE LEVEL CRITICAL;
```

现在，该审计策略必须删除：

```
DROP AUDIT POLICY priv_audit;
```

DROP FULLTEXT INDEX

语法：

```
DROP FULLTEXT INDEX <fulltext_index_name>
```

语法元素：

```
fulltext_index_name ::= <identifier>
```

定义将删除的索引。

描述：

DROP FULLTEXT INDEX 语句移除全文索引。

例子：

DROP FULLTEXT INDEX idx;

DROP INDEX

语法：

DROP INDEX <index_name>

语法元素：

index_name ::= <identifier>

索引名 identifier 定义了将删除的索引。

描述：

DROP INDEX 语句移除索引。

例子：

DROP INDEX idx;

DROP SCHEMA

语法：

DROP SCHEMA <schema_name> [<drop_option>]

语法元素：

<drop_option> ::= CASCADE | RESTRICT

Default = RESTRICT

有限制的删除操作将删除对象当其没有依赖对象时。如果存在依赖对象，将抛出错误。级联删除将删除对象以及其依赖对象。

描述：

DROP SCHEMA 语句移除数据集合。

例子：

创建集合 my_schema，表 my_schema.t，然后 my_schema 将使用 CASCADE 选项删除。

```
CREATE SCHEMA my_schema;
CREATE TABLE my_schema.t (a INT);
DROP SCHEMA my_schema CASCADE;
```

DROP SEQUENCE

语法：

DROP SEQUENCE <sequence_name> [<drop_option>]

语法元素：

<drop_option> ::= CASCADE | RESTRICT

Default = RESTRICT

级联删除将删除对象以及其依赖对象。当未指定 CASCADE 可选项，将执行非级联删除对象，不会删除依赖对象，而是使依赖对象 (VIEW, PROCEDURE) 无效。

无效的对象可以重新验证当一个对象有同样的集合，并且已创建了对象名。对象 ID，集合名以及对象名为重新验证依赖对象而保留。

有限制的删除操作将删除对象当其没有依赖对象时。如果存在依赖对象，将抛出错误。

描述：

DROP SEQUENCE 语句移除序列。

例子：

```
DROP SEQUENCE s;
```

DROP SYNONYM

语法：

DROP [PUBLIC] SYNONYM <synonym_name> [<drop_option>]

语法元素：

<synonym_name> ::= <identifier>

<drop_option> ::= CASCADE | RESTRICT

Default = RESTRICT

级联删除将删除对象以及其依赖对象。当未指定 CASCADE 可选项，将执行非级联删除操作，不会删除依赖对象，而是使依赖对象 (VIEW, PROCEDURE) 无效。

无效的对象可以重新验证当一个对象有同样的集合，并且已创建了对象名。对象 ID，集合名以及对象名为重新验证依赖对象而保留。

有限制的删除操作将删除对象当其没有依赖对象时。如果存在依赖对象，将抛出错误。

描述：

DROP SYNONYM 删除同义词。可选项 PUBLIC 允许删除公用同义词。

例子：

表 a 已创建，然后为表 a 创建同义词 a_synonym 和公用同义词 pa_synonym：

```
CREATE TABLE a (c INT);
CREATE SYNONYM a_synonym FOR a;
CREATE PUBLIC SYNONYM pa_synonym FOR a;
```

删除同义词 a_synonym 和公用同义词 pa_synonym

```
DROP SYNONYM a_synonym;
DROP PUBLIC SYNONYM pa_synonym;
```

DROP TABLE

语法：

DROP TABLE <table_name> [<drop_option>]

语法元素：

<table_name> ::= [<schema_name>.]<identifier>
<schema_name> ::= <identifier>
<drop_option> ::= CASCADE | RESTRICT

Default = RESTRICT

级联删除将删除对象以及其依赖对象。当未指定 CASCADE 可选项，将执行非级联删除操作，不会删除依赖对象，而是使依赖对象 (VIEW, PROCEDURE) 无效。

无效的对象可以重新验证当一个对象有同样的集合，并且已创建了对象名。对象 ID，集合名以及对象名为重新验证依赖对象而保留。

有限制的删除操作将删除对象当其没有依赖对象时。如果存在依赖对象，将抛出错误。

描述：

DROP TABLE 语句删除表。

例子：

创建表 A，然后删除。

```
CREATE TABLE A(C INT);
DROP TABLE A;
```

DROP TRIGGER

Syntax

DROP TRIGGER <trigger_name>

Syntax Elements

<trigger_name> ::= <identifier>

将删除的触发器名称。

描述：

DROP TRIGGER 语句删除一个触发器。

只有在触发器所作用表上有 TRIGGER 权限的数据库用户才允许删除该表的触发器。

例子：

对于这个例子，你需要先创建一个名为 test_trigger 的触发器，如下：

```
CREATE TABLE TARGET ( A INT);
CREATE TABLE SAMPLE ( A INT);
CREATE TRIGGER TEST_TRIGGER
```

```
AFTER UPDATE ON TARGET
BEGIN
INSERT INTO SAMPLE VALUES(3);
END;
```

现在你可以删除触发器：

```
DROP TRIGGER TEST_TRIGGER;
```

DROP VIEW

语法：

```
DROP VIEW <view_name> [<drop_option>]
```

语法元素：

```
<view_name> ::= [<schema_name>.]<identifier>
<schema_name> ::= <identifier>
<drop_option> ::= CASCADE | RESTRICT
```

Default = RESTRICT

级联删除将删除对象以及其依赖对象。当未指定 CASCADE 可选项，将执行非级联删除操作，不会删除依赖对象，而是使依赖对象 (VIEW, PROCEDURE) 无效。

无效的对象可以重新验证当一个对象有同样的集合，并且已创建了对象名。对象 ID，集合名以及对象名为重新验证依赖对象而保留。

有限制的删除操作将删除对象当其没有依赖对象时。如果存在依赖对象，将抛出错误。

描述：

DROP VIEW 语句删除视图。

例子：

表 t 已创建，然后选择表 a 中的所有数据创建视图 v：

```
CREATE TABLE t (a INT);
CREATE VIEW v AS SELECT * FROM t;
```

删除视图 v：

DROP VIEW v;

RENAME COLUMN

语法：

RENAME COLUMN <table_name>.<old_column_name> TO <new_column_name>

语法元素：

<old_column_name> ::= <identifier>
<new_column_name> ::= <identifier>

描述：

RENAME COLUMN 语句修改列名：

关于列名的信息，请参阅 Identifiers。

例子：

创建表 B:

CREATE TABLE B (A INT PRIMARY KEY, B INT);

显示表 B 中列名的列表：

SELECT COLUMN_NAME, POSITION FROM TABLE_COLUMNS WHERE SCHEMA_NAME = CURRENT_SCHEMA AND TABLE_NAME = 'B' ORDER BY POSITION;

列 A 重名为 C：

RENAME COLUMN B.A TO C;

重命名之后表 B 列名的列表：

SELECT COLUMN_NAME, POSITION FROM TABLE_COLUMNS WHERE SCHEMA_NAME = CURRENT_SCHEMA AND TABLE_NAME = 'B' ORDER BY POSITION;

RENAME INDEX

语法：

RENAME INDEX <old_index_name> TO <new_index_name>

语法元素：

<old_index_name> ::= <identifier>
<new_index_name> ::= <identifier>

描述：

RENAME INDEX 语句重命名索引名。

关于索引名的详情，请参阅 identifiers。

例子：

表 B 已创建，然后索引 idx 建立在表 B 的列 B：

```
CREATE TABLE B (A INT PRIMARY KEY, B INT);
CREATE INDEX idx on B(B);
```

显示表 B 的索引名列表：

```
SELECT INDEX_NAME FROM INDEXES WHERE SCHEMA_NAME = CURRENT_SCHEMA AND
TABLE_NAME='B';
```

索引 idx 重名为 new_idx：

```
RENAME INDEX idx TO new_idx;
```

重命名之后表 B 索引名的列表：

```
SELECT INDEX_NAME FROM INDEXES WHERE SCHEMA_NAME = CURRENT_SCHEMA AND
TABLE_NAME='B';
```

RENAME TABLE
语法：

RENAME TABLE <old_table_name> TO <new_table_name>

语法元素：

<old_table_name> ::= <identifier>
<new_table_name> ::= <identifier>

描述：

RENAME TABLE 语句在同一个集合下，将表名修改为 new_table_name。

关于表名的详情，请参阅 Identifiers。

例子：

在当前集合创建表 A：

```
CREATE TABLE A (A INT PRIMARY KEY, B INT);
```

显示当前集合下表名的列表：

```
SELECT TABLE_NAME FROM TABLES WHERE SCHEMA_NAME = CURRENT_SCHEMA;
```

表 A 重命名为 B：

```
RENAME TABLE A TO B;
```

显示重命名后当前集合下表名的列表：

```
SELECT TABLE_NAME FROM TABLES WHERE SCHEMA_NAME = CURRENT_SCHEMA;
```

集合 mySchema 已创建，然后创建表 mySchema.A：

```
CREATE SCHEMA mySchema;
CREATE TABLE mySchema.A (A INT PRIMARY KEY, B INT);
```

显示模式 mySchema 下表名的列表：

```
SELECT TABLE_NAME FROM TABLES WHERE SCHEMA_NAME = 'MYSCHEMA';
```

表 mySchema.A 重命名为 B：

```
RENAME TABLE mySchema.A TO B;
```

显示重命名后集合 mySchema 下表名的列表：

ALTER TABLE ALTER TYPE

语法：

```
<table_conversion_clause> ::= [ALTER TYPE] { ROW [THREADS <number_of_threads>] | CO

LUMN [THREADS <number_of_threads> [BATCH <batch_size>]] }
```

语法元素：

```
<number_of_threads> ::= <numeric_literal>
```

指定进行表转换的并行线程数。线程数目的最佳值应设置为可用的 CPU 内核的数量。

<batch_size> ::= <numeric_literal>

指定批插入的行数，默认值为最佳值 2,000,000。插入表操作在每个<batch_size>记录插入后立即提交，可以减少内存消耗。BATCH 可选项只可以在表 从行转换为列时使用。然而，大于 2,000,000 的批大小可能导致高内存消耗，因此不建议修改该值。

描述：

通过复制现有的表中的列和数据，可以从现有的表创建一个不同存储类型的新表。该命令用来把表从行转换为列或从列转换为行。如果源表是行式存储，则新建的表为列式存储。

配置参数:

用于表转换的默认线程数在 indexserver.ini 的[sql]部分定义，table_conversion_parallelism = <numeric_literal> (初始值为 8)。

例子：

对于这个例子，你需要先创建欲进行转换的表：

```
CREATE COLUMN TABLE col_to_row (col1 INT, col2 INT)
CREATE ROW TABLE row_to_col (col1 INT, col2 INT)
```

表 col_to_row 将以列式存储方式创建，表 row_to_col 则为行式存储。

现在你可以将表 col_to_row 的存储类型从列转为行：

```
ALTER TABLE col_to_row ALTER TYPE ROW
```

你也可以将表 row_to_col 的存储类型从行转为列：

```
ALTER TABLE row_to_col ALTER TYPE COLUMN
```

为了使用批量转换模式，你需要在语句结尾处添加批选项：

```
ALTER TABLE row_to_col ALTER TYPE COLUMN BATCH 10000
```

TRUNCATE TABLE
语法：

```
TRUNCATE TABLE <table_name>
```

描述：

删除表中所有记录。当从表中删除所有数据时，TRUNCATE 比 DELETE FROM 快，但是 TRUNCATE 无法回滚。要回滚删除的记录，应使用"DELETE FROM <table_name>"。

HISTORY 表可以通过执行该语句像正常表一样删除。历史表的所有部分(main, delta, history main and history delta)将被删除并且内容会丢失。

数据操纵语句：

DELETE：

语法：

DELETE [HISTORY] FROM <table_name> [WHERE <condition>]

语法元素：

<table_name> ::= [<schema_name>.]<identifier>
<schema_name> ::= <identifier>

关于 identifier 的详情，请参阅 Identifiers。

<condition> ::= <condition> OR <condition>
| <condition> AND <condition>
| NOT <condition>
| (<condition>)
| <predicate>

关于谓词的详情，请参阅 Predicates。

描述：

DELETE 语句当满足条件时，从表中删除所有记录。如果省略了 WHERE 子句，将删除表中所有记录。

DELETE HISTORY

DELETE HISTORY 将标记选中的历史表中历史记录进行删除。这表示执行完该语句后，引用已删除记录的时间旅行查询语句可能仍然可以查看这些数据。为了在物理上删除这些记录，必须执行下面的语句：

ALTER SYSTEM RECLAIM VERSION SPACE; MERGE HISTORY DELTA of <table_name>;

请注意：在某些情况中，即使执行了上述两条语句，仍无法从物理上删除。

欲检查记录是否已从物理上删除，以下语句会有帮助：

SELECT * FROM <table_name> WHERE <condition> WITH PARAMETERS ('REQUEST_FLAGS'= ('ALLCOMMITTED','HISTORYONLY'));

注意：“WITH PARAMETERS ('REQUEST_FLAGS'= ('ALLCOMMITTED','HISTORYONLY'))”子句可能只适用于验证 DELETE HISTORY 语句的执行结果。

例子：

```
CREATE TABLE T (KEY INT PRIMARY KEY, VAL INT);
INSERT INTO T VALUES (1, 1);
INSERT INTO T VALUES (2, 2);
INSERT INTO T VALUES (3, 3);
```

在下面的例子中，将删除一条记录：

```
DELETE FROM T WHERE KEY = 1;
```

EXPLAIN PLAN
语法：

EXPLAIN PLAN [SET STATEMENT_NAME = <statement_name>] FOR \ref sql_subquery

语法元素：

<statement_name> ::= string literal used to identify the name of a specific executi

on plan in the output table for a given SQL statement.

如果未指定 SET STATEMENT_NAME，则将设为 NULL。

描述：

EXPLAIN PLAN 语句用来评估 SAP HANA 数据库遵循的执行 SQL 语句的执行计划。评估的结果存在视图 EXPLAIN_PLAN_TABLE，以便稍后的用户检查。

SQL 语句必须是数据操纵语句，因此集合定义语句不能在 EXPLAIN STATEMENT 中使用。

你可以从 EXPLAIN_PLAN_TABLE 视图中得到 SQL 计划，该视图为所有用户共享。这里是从视图读取 SQL 计划的**例子：**

<tr><td>SELECT * FROM EXPLAIN_PLAN_TABLE;</td></tr>

EXPLAIN_PLAN_TABLE 视图中的列：

表 1：列名和描述

列名	描述
STATEMENT_NAME	执行 EXPLAIN_PLAN 命令时，指定为 STATEMENT NAME 的字符串。当在 EXPLAIN_PLAN_TABLE 视图中有多个计划时，用来区分计划。
OPERATOR_NAME	操作符的名字。详细描述在下面的部分中。
OPERATOR_DETAILS	操作符的详情。操作符使用的谓词和表达式显示在此。
SCHEMA—NAME	已访问表的数据集合名。
TABLE_NAME	已访问表的表名。
TABLE_TYPE	已访问表的类型。类型如下： COLUMN TABLE, ROW TABLE, MONITORING VIEW, JOIN VIEW, OLAP VIEW, CALCULATION VIEW 和 HIERARCHY VIEW.
TABLE_SIZE	已访问表中的估计记录数量。
OUTPUT_SIZE	由操作符产生的估计记录数量。
SUBTREE_COST	执行从操作符开始的子树估计成本。该值仅用于比较。
OPERATOR_ID	计划中操作符的唯一 ID，数值从 1 开始。
PARENT_OPERATOR_ID	计划双亲的 OPERATOR_ID。SQL 计划的形状是一棵树，其拓扑结构可以使用 OPERATORJD 和 PARENT_OPERATOR_ID 重建。 根操作符的 PARENT_OPERATOR_ID 显示为 NULL。
LEVEL	从根操作符起始的级别。根操作符的级别为 1，其子操作符级别为 2，等等。可以用于输出。
POSITION	双亲操作符的位置。第一个子操作符为 1，第二个子操作符为 2，等等。
HOST	执行操作符所在的主机名。
PORT	连接至主机的 TCP/IP 端口。
TIMESTAMP	执行 EXPLAIN PLAN 命令的日期和时间
CONNECTIONJD	执行 EXPLAIN PLAN 的连接 ID。
EXECUTION_ENGINE	执行操作符的引擎类型：COLUMN 或 ROW

EXPLAIN_PLAN_TABLE 视图中的 OPERATOR_NAME 列。表 2：OPERATOR_NAME 列显示的列式引擎操作符列表：

操作符名	描述
COLUMN SEARCH	列式引擎操作符起始位置。 OPERATOR_DETAILS 列出了投影列。

LIMIT	限制输出行数的操作符。
ORDER BY	对输出行排序的操作符。
HAVING	对分组和聚合顶端使用谓词进行过滤的操作符。
GROUP BY	进行分组和聚合的操作符。
DISTINCT	移除重复记录的操作符。
FILTER	使用谓词过滤的操作符。
JOIN	联接输入关系的操作符。
COLUMN TABLE	关于访问的列表的信息。
MULTI PROVIDER	生成联合多个相同分组和聚合结果的操作符。

表 3：OPERATOR_NAME 列显示的行式引擎操作符列表：

COLUMN SEARCH 为列式引擎操作符起始位置标记，ROW SEARCH 为行式引擎操作符起始位置标记。在以下的例子中，COLUMN SEARCH (ID 10)生成的中间结果被 ROW SEARCH (ID 7)使用，ROW SEARCH (ID 7) 被另一个 COLUMN SEARCH (ID 1)使用。位于 COLUMN SEARCH (ID 10) 最底层的操作符解释了 COLUMN SEARCH (ID 10) 是如何执行的。ROW SEARCH (ID 7) 和 COLUMN SEARCH (ID 10) 之间的操作符说明了 ROW SEARCH (ID 7) 如何处理由 COLUMN SEARCH (ID 10) 生成的中间结果。位于 COLUMN SEARCH (ID 1) 和 ROW SEARCH (ID 7)顶层的操作符解释了顶层 COLUMN SEARCH (ID 1) 是如何处理由 ROW SEARCH (ID 7)生成的中间结果。

表 4：操作符

OPERATOR_NAME	OPERATOR」D	PARENT_OPERATOR_ID	LEVEL	POSITION
COLUMN SEARCH	1	NULL	1	1
LIMIT	2	1	2	1
ORDER BY	3	2	3	1

GROUP BY	4	3	4	1
JOIN	5	4	5	1
COLUMN TABLE	6	5	6	1
ROW SEARCH	7	〈font color="BLUE">5	6</fo n t>	2
BTREE INDEX JOIN	8	7	7	1
BTREE INDEX JOIN	9	8	8	1
COLUMN SEARCH	10	〈font color="RED">9	9	l
FILTER	11	10	10	1
COLUMN TABLE	12	11	11	1

SQL 计划解释例子。

该语句来自语 TPC-H 基准。例子中的所有表都是行式存储。

```
DELETE FROM expl a i n_pl a n_ta bl e WHERE s ta tement_na me = 'TPC-H Q10';
EXPLAIN PLAN SET STATEMENT_NAME = 'TPC-H Q10' FOR
SELECT TOP 20
c_cus tkey,
c_na me,
SUM(l _extendedpri ce * (1 - l _di s count)) AS revenue,
c_a cctba l ,
n_na me,
c_a ddres s ,
c_phone,
c_comment
FROM
cus tomer,
orders ,
l i nei tem,
na ti on
WHERE
c_cus tkey = o_cus tkey
AND l _orderkey = o_orderkey
AND o_orderda te >= '1993-10-01'
AND o_orderda te < ADD_MONTHS('1993-10-01',3)
AND l _returnfl a g = 'R'
AND c_na ti onkey = n_na ti onkey
GROUP BY
c_cus tkey,
c_na me,
c_a cctba l ,
```

```
c_phone,
n_na me,
c_a ddres s ,
c_comment
ORDER BY
revenue DESC;
SELECT opera tor_na me, opera tor_deta i l s , ta bl e_na me
FROM expl a i n_pl a n_ta bl e
WHERE s ta tement_na me = 'TPC-H Q10';
```

以下是对上文查询语句的计划解释：

OPERATOR_NAME	OPERATOR_DETAILS	TABLE－NAME
ROW SEARCH	CUSTOMER.C_CUSTKEY, CUSTOMER.C－NAME, SUM(LINEITEM.L_EXTENDEDPRICE * (1 - LINEITEM.L_DISCOUNT)), CUSTOMER.C_ACCTBAL, NATION.N_NAME, CUSTOMER.C_ADDRESS, CUSTOMER.C_PHONE, CUSTOMER.C_COMMENT	None
LIMIT	NUM RECORDS: 20	
ORDER BY	SUM(LINEITEM.L_EXTENDEDPRICE * (1 - LINEITEM.L_DISCOUNT)) DESC	None
MERGE AGGREGATION	NUM PARTITIONS: 4	None
GROUP BY	GROUPING: NATION.N_NAME, R_CUSTOMER.C_CUSTKEY, AGGREGATION: SUM(LINEITEM.L_EXTENDEDPRICE * (1 - LINEITEM.L_DISCOUNT))	None
CPBTREE INDEX JOIN	INDEX NAME: _SYS_TREE_RS_279_#0_#P0/ INDEX CONDITION: ORDERS.O_ORDERKEY = LINEITEM.L_ORDERKEY, INDEX FILTER: 'R' = LINEITEM.L_RETURNFLAG	LINEITEM
BTREE INDEX JOIN	INDEX NAME: _SYS_TREE_RS_285_#0_#P0, INDEX CONDITION: CUSTOMER.C_NATIONKEY = NATION.N_NATIONKEY	NATION
BTREE INDEX JOIN	INDEX NAME: _SYS_TREE_RS_283_#0_#P0, INDEX CONDITION: ORDERS.O_CUSTKEY = CUSTOMER.C_CUSTKEY	CUSTOMER
TABLE SCAN	FILTER CONDITION: ORDERS.O_ORDERDATE < '1994-01-01' AND ORDERS.O_ORDERDATE >= '1993-10-01'	ORDERS

INSERT

语法:

```
INSERT INTO <table_name> [ <column_list_clause> ] { <value_list_clause> | <subquery> }
```

语法元素：

<table_name> ::= [<schema_name>.]<identifier>
<schema_name> ::= <identifier>
<column_list_clause> ::= (<column_name>, ...)
<column_name> ::= <identifier>

有关 identifier 的详情，请参阅 Identifiers。

<value_list_clause> ::= VALUES (<expression>, ...)

有关表达式的详情，请参阅 Expressions。

描述：

INSERT 语句添加一条到表中。返回记录的子查询可以用来插入表中。如果子查询没有返回任何结果，数据库将不会插入任何记录。可以使用 INSERT 语句指定列的列表。不在列表中的列将显示默认值。如果省略了列的列表，数据库插入所有记录到表中。

例子：

CREATE TABLE T (KEY INT PRIMARY KEY, VAL1 INT, VAL2 NVARCHAR(20));

在以下的例子中，你可以插入值：

INSERT INTO T VALUES (1, 1, 'The first');

KEY	VAL1	VAL2
1	1	The first

你可以将值插入到指定的列：

INSERT INTO T (KEY) VALUES (2);

KEY	VAL1	VAL2
1	1	The first
2	NULL	NULL

你也可以使用子查询：

INSERT INTO T SELECT 3, 3, 'The third' FROM DUMMY;

KEY	VAL1	VAL2
1	1	The first
2	NULL	NULL
3	3	The third

LOAD
语法：

LOAD <table_name> {DELTA | ALL | (<column_name>, ...)}

描述：

LOAD 语句明确地加载列式存储表的数据至内存中，而非第一次访问时加载。

语法元素：

<table_name> ::= <identifier>

将加载至内存中的表名。

<column_name> ::= <identifier>

将加载至内存中的列名。

DELTA
使用 DELTA，列式表的一部分将加载至内存。由于列式存储对于读操作做了优化和压缩，增量用来优化插入或更新。所有的插入被传递给一个增量。

ALL
主要的和增量的列存储表中所有数据加载到内存中。

例子：

以下的例子加载整张表 a_table 至内存中。

LOAD a_table all;

以下的例子加载列 a_column 和表 a_table 列 another_column 至内存中。

LOAD a_table (a_column,another_column);

表加载状态可以查询：

select loaded from m_cs_tables where table name = '<table_name>'

MERGE DELTA
语法：

MERGE [HISTORY] DELTA OF <table_name> [PART n] [WITH PARAMETERS (<parameter_key_value>, ...)]

语法元素：

WITH PARAMETERS (<parameter_list>):

列式存储特定的选项可以使用"WITH PARAMETERS"子句传递。

<table_name> ::= [<schema_name>.]<identifier>
<schema_name> ::= <identifier>
<parameter_list> ::= <parameter>,<parameter_list>
<parameter> ::= <parameter_name> = <parameter_setting>
<parameter_name> ::= 'SMART_MERGE' | 'MEMORY_MERGE'
<parameter_setting> ::= 'ON' | 'OFF'

当前参数：'SMART_MERGE' = 'ON' | 'OFF'。当 SMART_MERGE 为 ON，数据库执行智能合并，这代表数据库基于 Indexserver 配置中定义的合并条件来决定是否合并。

'MEMORY_MERGE' = 'ON' | 'OFF' 数据库只合并内存中表的增量部分，不会被持久化。

描述：

使用 DELTA，列式表的一部分将加载至内存。由于列式存储对于读操作做了优化和压缩，增量用来优化插入或更新。所有的插入被传递至一个增量部分。

HISTORY 可以指定合并历史表增量部分到临时表的主要历史部分。
PART-可以指定合并历史表增量部分到临时表的主要历史部分，该表已分区。

例子：

MERGE DELTA OF A;
Merges the column store table delta part to its main part.
MERGE DELTA OF A WITH PARAMETERS('SMART_MERGE' = 'ON');
Smart merges the column store table delta part to its main part.

MERGE DELTA OF A WITH PARAMETERS('SMART_MERGE' = 'ON', 'MEMORY_MERGE' = 'ON');
Smart merges the column store table delta part to its main part non-persistent, in memory only.

MERGE DELTA OF A PART 1;
Merge the delta of partition no. 1 of table with name "A" to main part of partion no. 1.

MERGE HISTORY DELTA OF A;
Merge the history delta part of table with name "A" into its history main part.

MERGE HISTORY DELTA OF A PART 1;
Merges the column store table delta part of the history of table with name "A" to its history main part.

REPLACE | UPSERT

语法：

UPSERT <table_name> [<column_list_clause>] { <value_list_clause> [WHERE <condition> | WITH PRIMARY KEY] | <subquery> }

REPLACE <table_name> [<column_list_clause>] { <value_list_clause> [WHERE <condition> | WITH PRIMARY KEY] | <subquery> }

语法元素：

<table_name> ::= [<schema_name>.]<identifier>
<schema_name> ::= <identifier>
<column_list_clause> ::= (<column_name>, ...)
<column_name> ::= <identifier>

有关标识符的详情，请参阅 Identifiers。

<value_list_clause> ::= VALUES (<expression>, ...)

有关表达式的详情，请参阅 Expressions。

<condition> ::= <condition> OR <condition>
| <condition> AND <condition>
| NOT <condition>
| (<condition>)
| <predicate>

有关谓词的详情，请参阅 Predicates。

描述：

没有子查询的 UPSERT 或者 REPLACE 语句与 UPDATE 相似。唯一的区别是当 WHERE 子句为假时，该语句像 INSERT 一样添加一条新的记录到表中。

对于表有 PRIMARY KEY 的情况，主键列必须包含在列的列表中。没有默认设定，由 NOT NULL 定义的列也必须包含在列的列表中。

有子查询的 UPSERT 或者 REPLACE 语句与 INSERT 一样，除了如果表中旧记录与主键的新记录值相同，则旧记录将被子查询返回的记录所修改。除非表有一个主键，否则就变得等同于 INSERT，因为没有使用索引来判断新记录是否与旧记录重复。

有'WITH PRIMARY KEY'的 UPSERT 或 REPLACE 语句与有子查询的语句相同。其在 PRIMARY KEY 基础上工作。

例子：

CREATE TABLE T (KEY INT PRIMARY KEY, VAL INT);

你可以插入一条新值：

UPSERT T VALUES (1, 1);

KEY	VAL
1	1

如果 WHERE 子句中的条件为假，将插入一条新值：

UPSERT T VALUES (2, 2) WHERE KEY = 2;

KEY	VAL
1	1
2	2

你可以更新列"VAL"的第一条记录

UPSERT T VALUES (1, 9) WHERE KEY = 1;

KEY	VAL
1	9
2	2

或者你可以使用"WITH PRIMARY KEY" 关键字

UPSERT T VALUES (1, 8) WITH PRIMARY KEY;

KEY	VAL
1	8
2	2

你可以使用子查询插入值：

UPSERT T SELECT KEY + 2, VAL FROM T;

KEY	VAL
1	8
2	2
3	8
4	2

SELECT
语法：

<select_statement> ::= <subquery> [<for_update> | <time_travel>]
| (<subquery>) [<for_update> | <time_travel>]
<subquery> ::= <select_clause>

<from_clause>
[<where_clause>]
[<group_by_clause>]
[<having_clause>]
[{<set_operator> <subquery>, ... }]
[<order_by_clause>]
[<limit>]

语法元素：

SELECT 子句：

SELECT 子句指定要返回给用户或外部的 select 子句一个输出，如果存在的话。

<select_clause> ::= SELECT [TOP <integer>] [ALL | DISTINCT] <select_list>
<select_list> ::= <select_item>[, ...]
<select_item> ::= [<table_name>.] <asterisk>
| <expression> [AS <column_alias>]
<table_name> ::= [<schema_name>.] <identifier>
<schema_name> ::= <identifier>
<column_name> ::= <identifier>
<column_alias> ::= <identifier>
<asterisk> ::= *

TOP n
TOP n 用来返回 SQL 语句的前 n 条记录。

DISTINCT 和 ALL

可以使用 DISTINCT 返回重复的记录，每一组选择只有一个副本。
使用 ALL 返回选择的所有记录，包括所有重复记录的拷贝。默认值为 ALL。

Select_list
select_list 允许用户定义他们想要从表中选择的列。

Asterisk
asterisk 可以从 FROM 子句中列出的表或视图中选择所有列。如果集合名和表名或者表名带有星号(*)，其用来限制结果集至指定的表。

column_alias
column_alias 可以用于简单地表示表达式。

FROM
FROM 子句中指定输入值，如表、视图、以及将在 SELECT 语句中使用的子查询。

<from_clause> ::= FROM {<table>, ... }
<table> ::= <table_name> [[AS] <table_alias>]
| <subquery> [[AS] <table_alias>]
| <joined_table>
<table_alias> ::= <identifier>
<joined_table> ::= <table> [<join_type>] JOIN <table> ON <predicate>|
| <table> CROSS JOIN <table>
| <joined_table>
<join_type> ::= INNER | { LEFT | RIGHT | FULL } [OUTER]

table alias

表别名可以用来简单地表示表或者子查询。

join_type 定义了将执行的联接类型， LEFT 表示左外联接，RIGHT 表示右外联接，FULL 表示全外联接。执行联接操作时，OUT 可能或者可能不使用。
ON <predicate>
ON 子句定义联接谓词。
CROSS JOIN
CROSS 表示执行交叉联接，交叉联接生成两表的交叉积结果。

WHERE 子句

WHERE 子句用来指定 FROM 子句输入的谓词，使用户可以检索所需的记录。

<where_clause> ::= WHERE <condition>

<condition> ::=<condition> OR <condition>
| <condition> AND <condition>
| NOT <condition>
| (<condition>)
| <predicate>

<predicate> ::= <comparison_predicate>
| <range_preciate>
| <in_predicate>
| <exist_predicate>
| <like_predicate>
| <null_predicate>

<comparison_predicate> ::= <expression> { = | != | <> | > | < | >= | <= }

[ANY | SOME | ALL] ({<expression_list> | <subquery>})
<range_predicate> ::= <expression> [NOT] BETWEEN <expression> AND <expression>
<in_predicate> ::= <expression> [NOT] IN ({ <expression_list> | <subquery> })

<exist_predicate> ::= [NOT] EXISTS (<subquery>)
<like_predicate> ::= <expression> [NOT] LIKE <expression> [ESCAPE <expression>]
<null_predicate> ::= <expression> IS [NOT] NULL
<expression_list> ::= {<expression>, ... }

GROUP BY 子句

<group_by_clause> ::=GROUP BY { <group_by_expression_list> | <grouping_set> }
<group_by_expression_list> ::= { <expression>, ... }

<grouping_set> ::= { GROUPING SETS | ROLLUP | CUBE }[BEST <integer>] [LIMIT <integer>[OFFSET <integer>]]
[WITH SUBTOTAL] [WITH BALANCE] [WITH TOTAL]
[TEXT_FILTER <filterspec> [FILL UP [SORT MATCHES TO TOP]]]
[STRUCTURED RESULT [WITH OVERVIEW] [PREFIX <string_literal>] | MULTIPLE RESULTSETS]
(<grouping_expression_list>)

<grouping_expression_list> ::= { <grouping_expression>, ... }

<grouping_expression> ::=<expression>| (<expression>, ...)
| ((<expression>, ...) <order_by_clause>)

GROUP BY 用来对基于指定列值选定的行进行分组。

GROUPING SETS
在一条语句中，生成多个特定数据分组结果。如果没有设置例如 best 和 limit 的可选项，结果将和 UNION ALL 每个指定组的聚合值相同。例如："select col1, col2, col3, count(*) from t group by grouping sets ((col1, col2), (col1, col3))"与"select col1, col2, NULL, count(*) from t group by col1, col2 union all select col1, NULL, col3,count(*) from t group by col1, col3"相同。在 grouping-sets 语句中，每个(col1, col2) 和(col1, col3)定义了分组。

ROLLUP
在一条语句中，生成多级聚合结果。例如，"rollup (col1, col2,col3)"与有额外 聚合，但没有分组的"grouping sets ((col1, col2, col3), (col1, col2), (col1))"结果相同。因此，结果集所包含的分组的数目是 ROLLUP 列表中的列加上一个最后聚合的数目，如果没有额外的选项。

CUBE
在一条语句中，生成多级聚合的结果。例如，"cube (col1, col2,col3)" 与有额外聚合，但没有分组的"grouping sets ((col1, col2, col3), (col1, col2), (col1, col3), (col2, col3), (col1), (col2),(col3))"结果相同。因此，结果集所包含分组的数目与所有可能排列在 CUBE 列表的列加上一个最后聚合的数目是相同的，如果没有附加的选项。

BEST n
返回每个以行聚合数降序排列的分组集中前 n 个分组集。n 可以是任意零，正数或负数。当 n 为零时，作用和没有 BEST 选项一样。当 n 为负数表示以升序排序。

LIMIT n1 [OFFSET n2]
返回每个分组集中第一个 N1 分组记录跳过 N2 个后的结果。

WITH SUBTOTAL
返回每个分组集中由 OFFSET 或者 LIMIT 控制的返回结果的分类汇总。除非设置了 OFFSET 和 LIMIT，
返回值将和 WITH TOTAL 相同。

WITH BALANCE
返回每个分组集中 OFFSET 或者 LIMIT 没有返回的其余结果值。

WITH TOTAL
返回每个分组集中额外的合计总值行。OFFSET 和 LIMIT 选项不能修改该值。

TEXT_FILTER <filterspec>
执行文本过滤或者用<filterspec>高亮分组列，<filterspec>为单引号字符串，语法如下：

<filterspec> ::= '[<prefix>]<element>{<subsequent>, ...}'
<prefix> ::= + | - | NOT
<element> ::= <token> | <phrase>
<token> ::= !! Unicode letters or digits
<phrase> ::= !! double-quoted string that does not contain double quotations inside
<subsequent> ::= [<prefix_subsequent>]<element>
<prefix_subsequent> ::= + | - | NOT | AND | AND NOT | OR

<filterspec>定义的过滤是由与逻辑操作符 AND, OR 和 NOT 连接的标记/词组或者短语组成。一个标记相匹配的字符串，其中包含对应不区分大小写的单词，'ab' 匹配 'ab cd' 和 'cd Ab' ，但不匹配 'abcd'。一个标记可以包含通配字符’，匹配任何字符串，’匹配任意字母。但是在词组内，’和’不是通配符。逻辑运算符 AND，OR 和 NOT 可以与标记，词组一起使用。由于 OR 是默认操作符，'ab cd' 和'ab OR cd'意义一样。注意，逻辑运算符应该大写。作为一种逻辑运算符，前缀'+' 和 '-'各自表示包含(AND) 和不包含 (AND NOT)。例如，'ab -cd' 和 'ab AND NOT cd'意义相同。如果没有 FILL UP 选项，只返回含有匹配值的分组记录。需要注意的是一个过滤器仅被运用到每个分组集的第一个分组列。

FILL UP
不仅返回匹配的分组记录，也包含不匹配的记录。text_filter 函数对于识别哪一个匹配是很有用的。参阅下面的'Related Functions’。

SORT MATCHES TO TOP
返回匹配值位于非匹配值前的分组集。该选项不能和 SUBTOTAL, BALANCE 和 TOTAL 一起使用。

STRUCTURED RESULT
结果作为临时表返回。对于每一个分组集创建一个临时表，如果设置 WITH OVERVIEW 选项，将为分组集的总览创建额外的临时表，该临时表的名字由 PREFIX 选项定义。

WITH OVERVIEW
将总览返回至单独的额外一张表中。

PREFIX 值
使用前缀命名临时表。必须以"#"开始，代表是临时表。如果省略，默认前缀为"#GN"，然后，连接该前缀值和一个非负整数，用作临时表的名称，比如"#GN0", "#GN1" 和 "#GN2"。

MULTIPLE RESULTSETS
返回多个结果集中的结果。

相关函数
grouping_id (<grouping_column1, ..., grouping_columnn>)函数返回一个整数，判断每个分组记录属于哪个分组集。text_filter (<grouping_column>) 函数与 TEXT_FILTER, FILL UP, 和 SORT MATCHES TO TOP 一起使用，显示匹配值或者 NULL。当指定了 FILL UP 选项时，未匹配值显示为 NULL。

返回格式
如果 STRUCTURED RESULT 和 MULTIPLE RESULTSETS 都没有设置，返回所有分组集 的联合，以及对于没有包含在指定分组集中的属性填充的 NULL 值。使用 STRUCTURED RESULT，额外的创建临时表，在同一会话中用"SELECT * FROM <table name>"可以查询。表名遵循的格式：

<PREFIX>0：如果定义了 WITH OVERVIEW，该表将包含总览。
<PREFIX>n：由 BEST 参数重新排序的第 n 个分组集。
使用 MULTIPLE RESULTSETS，将返回多个结果集。每个分组集的分组记录都在单个结果集中。

HAVING 子句：
HAVING 子句用于选择满足谓词的特定分组。如果省略了该子句，将选出所有分组。

<having_clause> ::= HAVING <condition>

SET OPERATORS
SET OPERATORS 使多个 SELECT 语句相结合，并只返回一个结果集。

<set_operator> ::= UNION [ALL | DISTINCT] | INTERSECT [DISTINCT] | EXCEPT [DISTINCT]

UNION ALL
选择所有 select 语句中的所有记录。重复记录将不会删除。

UNION [DISTINCT]
选择所有 SELECT 语句中的唯一记录，在不同的 SELECT 语句中删除重复记录。UNION 和 UNION DISTINCT 作用相同。

INTERSECT [DISTINCT]
选择所有 SELECT 语句中共有的唯一记录。

EXCEPT [DISTINCT]
在位于后面的 SELECT 语句删除重复记录后，返回第一个 SELECT 语句中所有唯一的记录。

ORDER BY 子句

<order_by_clause> ::= ORDER BY { <order_by_expression>, ... }
<order_by_expression> ::= <expression> [ASC | DESC]
| <position> [ASC | DESC
<position> ::= <integer>

ORDER BY 子句用于根据表达式或者位置对记录排序。位置表示选择列表的索引。对"select col1, col2 from t order by 2", 2 表示 col2 在选择列表中使用的第二个表达式。ASC 用于按升序排列记录，DESC 用于按降序排列记录。默认值为 ASC。

LIMIT

LIMIT 关键字定义输出的记录数量。

<limit> ::= LIMIT <integer> [OFFSET <integer>]

LIMIT n1 [OFFSET n2]

返回跳过 n2 条记录后的最先 n1 条记录。

FOR UPDATE

FOR UPDATE 关键字锁定记录，以便其他用户无法锁定或修改记录，直到本次事务结束。

<for_update> ::= FOR UPDATE

TIME TRAVEL

该关键字与时间旅行有关，用于语句级别时间旅行回到 commit_id 或者时间指定的快照。

<time_travel> ::= AS OF { { COMMIT ID <commit_id> } | { UTCTIMESTAMP <timestamp> }}

时间旅行只对历史列表适用。<commit_id>在每次提交后可以从 m_history_index_last_commit_id 获得，其相关的<timestamp>可以从 sys.m_transaction_history 读取。

```
create history column table x ( a int, b int ); // after turnning off auto commit
insert into x values (1,1);
commit;
select last_commit_id from m_history_index_last_commit_id where session_id = current_connection;
// e.g., 10
insert into x values (2,2);
commit;

select last_commit_id from m_history_index_last_commit_id where session_id = curren
t_connection; // e.g., 20
delete from x;
commit;
select last_commit_id from m_history_index_last_commit_id where session_id = curren
t_connection; // e.g., 30
select * from x as of commit id 30; // return nothing
select * from x as of commit id 20; // return two records (1,1) and (2,2)
select * from x as of commit id 10; // return one record (1,1)
select commit_time from sys.transaction_history where commit_id = 10; // e.g., '201
2-01-01 01:11:11'
select commit_time from sys.transaction_history where commit_id = 20; // e.g., '201
2-01-01 02:22:22'
select commit_time from sys.transaction_history where commit_id = 30; // e.g., '201
2-01-01 03:33:33'
select * from x as of utctimestamp '2012-01-02 02:00:00'; // return one record (1,1
)
select * from x as of utctimestamp '2012-01-03 03:00:00'; // return two records (1,
1) and (2,2)
select * from x as of utctimestamp '2012-01-04 04:00:00'; // return nothing
```

例子：

表 t1：

```
drop table t1;
create column table t1 ( id int primary key, customer varchar(5), year int, product
varchar(5), sales int );
insert into t1 values(1, 'C1', 2009, 'P1', 100);
insert into t1 values(2, 'C1', 2009, 'P2', 200);
insert into t1 values(3, 'C1', 2010, 'P1', 50);
insert into t1 values(4, 'C1', 2010, 'P2', 150);
insert into t1 values(5, 'C2', 2009, 'P1', 200);
insert into t1 values(6, 'C2', 2009, 'P2', 300);
insert into t1 values(7, 'C2', 2010, 'P1', 100);
insert into t1 values(8, 'C2', 2010, 'P2', 150);
```

以下的 GROUPING SETS 语句和第二个 group-by 查询相等。需要注意的是，两组在第一个查询的分组集内指定的各组在第二个查询。

```
select customer, year, product, sum(sales)
from t1
group by GROUPING SETS
(
(customer, year),
(customer, product)
);

select customer, year, NULL, sum(sales)
from t1
group by customer, year
union all
select customer, NULL, product, sum(sales)
from t1
group by customer, product;
```

ROLLUP 和 CUBE 经常使用的分组集的简明表示。下面的 ROLLUP 查询与第二个 group-by 查询相等。

```
select customer, year, sum(sales)
from t1
group by ROLLUP(customer, year);

select customer, year, sum(sales)
from t1
group by grouping sets
(
(customer, year),
(customer)
)
union all
select NULL, NULL, sum(sales)  from t1;
```

以下的 CUBE 查询与第二个 group-by 查询相等。

```
select customer, year, sum(sales)
from t1
group by CUBE(customer, year);

select customer, year, sum(sales)
from t1
group by grouping sets
(
(customer, year),
(customer),
(year)
)
union all
```

```
select NULL, NULL, sum(sales)
from t1;
```

BEST 1 指定以下查询语句只能返回最上面的 1 个 best 组。在该个例子中，对于(customer, year)组存在 4 条记录，而(product)组存在 2 条记录，因此返回之前的 4 条记录。对于 'BEST -1' 而非 'BEST 1'，返回后 2 条记录。

```
select customer, year, product, sum(sales)
from t1
group by grouping sets BEST 1
(
(customer, year),
(product)
);
```

LIMIT2 限制每组最大记录数为 2。对于(customer, year) 组，存在 4 条记录，只返回前 2 条记录；(product)组的记录条数为 2，因此返回所有结果。

```
select customer, year, product, sum(sales)
from t1
group by grouping sets LIMIT 2
(
(customer, year),
(product)
);
```

WITH SUBTOTAL 为每一组生成额外的一条记录，显示返回结果的分类汇总。这些记录的汇总对 customer, year, product 列返回 NULL，选择列表中 sum(sales)的总和。

```
select customer, year, product, sum(sales)
from t1
group by grouping sets LIMIT 2 WITH SUBTOTAL
(
(customer, year),
(product)
);
```

WITH BALNACE 为每一组生成额外的一条记录，显示未返回结果的分类汇总。

```
select customer, year, product, sum(sales)
from t1
group by grouping sets LIMIT 2 WITH BALANCE
(
(customer, year),
(product)
);
```

WITH TOTAL 为每一组生成额外的一条记录，显示所有分组记录的汇总，不考虑该分组记录是否返回。

```
select customer, year, product, sum(sales)
from t1
group by grouping sets LIMIT 2 WITH TOTAL
(
(customer, year),
(product)
)
```

TEXT_FILTER 允许用户获得有指定的<filterspec>的分组的第一列。以下查询将搜索以’2’结尾的列：对于第一个分组集为 customers， 第二个为 products。只返回三条匹配的记录。在 SELECT 列表中的 TEXT_FILTER 对于查看哪些值匹配是很有用的。

```
select customer, year, product, sum(sales), text_filter(customer), text_filter(prod
uct)
from t1
group by grouping sets TEXT_FILTER '*2'
(
(customer, year),
(product)
);
```

FILL UP 用于返回含有<filterspec>的匹配和不匹配的记录。因此，下面的查询返回 6 条记录，而先前的查询返回 3 条。

```
select customer, year, product, sum(sales), text_filter(customer), text_filter(prod
uct)
from t1
group by grouping sets TEXT_FILTER '*2' FILL UP
(
(customer, year),
(product)
);
```

SORT MATCHES TO TOP 用于提高匹配记录。对于每个分组集，将对其分组记录进行排序。

```
select customer, year, product, sum(sales), text_filter(customer), text_filter(prod
uct)
from t1
group by grouping sets TEXT_FILTER '*2' FILL UP SORT MATCHES TO TOP
(
(customer, year),
(product)
);
```

STRUCTURED RESULT 为每个分组集创建一张临时表，并且可选地，为总览表也创建一张。表"#GN1" 为 分组集(customer, year)，表"#GN2" 为分组集(product)。注意，每张表只含有一列相关列。也就是说，表"#GN1"不包含列"product"，而表"#GN2"不包含列"customer" and "year"。

```
select customer, year, product, sum(sales)
from t1
group by grouping sets STRUCTURED RESULT
(
(customer, year),
(product)
);
select * from "#GN1";
select * from "#GN2";
```

WITH OVERVIEW 为总览表创建临时表"#GN0"。

```
drop table "#G1";
drop table "#G2";
select customer, year, product, sum(sales)
from t1
group by grouping sets structured result WITH OVERVIEW
(
(customer, year),
(product)
);
select * from "#GN0";
select * from "#GN1";
select * from "#GN2";
```

用户可以通过使用 PREFIX 关键字修改临时表的名字。注意，名字必须以临时表的前缀'#'开始。

```
select customer, year, product, sum(sales)
from t1
group by grouping sets STRUCTURED RESULT WITH OVERVIEW PREFIX '#MYTAB'
(
(customer, year),
(product)
);
select * from "#MYTAB0";
select * from "#MYTAB1";
select * from "#MYTAB2";
```

当相应的会话被关闭或用户执行 drop 命令，临时表被删除。临时列表是显示在 m_temporary_tables。

```
select * from m_temporary_tables;
```

MULTIPLE RESULTSETS 返回多个结果的结果集。在 SAP HANA Studio 中，以下查询将返回三个结果集：一个为总览表，两个为分组集。

```
select customer, year, product, sum(sales)
from t1
group by grouping sets MULTIPLE RESULTSETS
(
(customer, year),
(product)
);
```

UNLOAD

语法：

```
UNLOAD <table_name>
```

语法元素：

```
<table_name> ::= <identifier>
```

从内存卸载的表的名称。

描述：

UNLOAD 语句从内存中卸载列存储表，以释放内存。表将在下次访问时重新加载。

例子：

在下面的例子中，表 a_table 将从内存中卸载。

```
UNLOAD a_table;
```

卸载表的状态可以通过以下语句查询：

```
select loaded from m_cs_tables where table name = '<table_name>'
```

UPDATE

语法

```
UPDATE <table_name> [ <alias_name> ] <set_clause> [ WHERE <condition> ]
```

语法元素：

```
<table_name> ::= [<schema_name>.]<identifier>
<schema_name> ::= <identifier>
<alias_name> ::= [AS] <identifier>
```

关于 Identifier 的详情， 请参见 Identifiers。

<set_clause> ::= SET {<column_name> = <expression>},...

关于表达式的详情，请参见 Expressions。

<condition> ::= <condition> OR <condition>
| <condition> AND <condition>
| NOT <condition>
| (<condition>)
| <predicate>

关于谓词的详情，请参见 Predicates。

描述：

UPDATE 语句修改满足条件的表中记录的值。如果 WHERE 子句中条件为真，将分配该列至表达式的结果中。如果省略了 WHERE 子句，语句将更新表中所有的记录。

例子：

```
CREATE TABLE T (KEY INT PRIMARY KEY, VAL INT);
INSERT INTO T VALUES (1, 1);
INSERT INTO T VALUES (2, 2);
```

如果 WHERE 条件中的条件为真，记录将被更新。

```
UPDATE T SET VAL = VAL + 1 WHERE KEY = 1;
```

KEY	VAL
1	2
2	2

如果省略了 WHERE 子句，将更新表中所有的记录。

```
UPDATE T SET VAL = KEY + 10;
```

KEY	VAL
1	11
2	12

系统管理语句

SET SYSTEM LICENSE

语法：

SET SYSTEM LICENSE '<license key>'

描述：

安装许可证密钥的数据库实例。许可证密钥(<license key>="">) 将从许可证密钥文件中复制黏贴。执行该命令需要系统权限 LICENSE ADMIN。

例子：

```
SET SYSTEM LICENSE '----- Begin SAP License -----
SAPSYSTEM=HD1
HARDWARE-KEY=K4150485960
INSTNO=0110008649
BEGIN=20110809
EXPIRATION=20151231
LKEY=...
SWPRODUCTNAME=SAP-HANA
SWPRODUCTLIMIT=2147483647
SYSTEM-NR=00000000031047460'
```

ALTER SYSTEM ALTER CONFIGURATION
语法：

```
ALTER CONFIGURATION (<filename>, <layer>[, <layer_name>]) SET | UNSET
<parameter_key_value_list> [ WITH RECONFIGURE]
```

语法元素：

<filename> ::= <string_literal>

行存储引擎配置的情况下，文件名是'indexserver.ini'。所使用的文件名必须是一个位于’DEFAULT’层的 ini 文件。如果选择文件的文件名在所需的层不存在，该文件将用 SET 命令创建。

<layer> ::= <string_literal>

设置配置变化的目标层。该参数可以是'SYSTEM'或'HOST'。SYSTEM 层为客户设置的推荐层。HOST 层应该一般仅可用于少量的配置，例如，daemon.ini 包含的参数。

<layer_name> ::= <string_literal>

如果上述的层设为’HOST’，layer_name 将用于设置目标 tenant 名或者目标主机名。例如，'selxeon12' 为目标 'selxeon12' 主机名。

SET
SET 命令更新键值，如果该键已存在，或者需要的话插入该键值。

UNSET
UNSET 命令删除键及其关联值。

<parameter_key_value_list> ::=
{(<section_name>,<parameter_name>) = <parameter_value>},...

指定要修改的 ini 文件的段、键和值语句如下：

<section_name> ::= <string_literal>

将要修改的参数段名：

<parameter_name> ::= <string_literal>

将要修改的参数名：

<parameter_value> ::= <string_literal>

将要修改的参数值。

WITH RECONFIGURE
当指定了 WITH RECONFIGURE，配置的修改将直接应用到 SAP HANA 数据库实例。
当未指定 WITH RECONFIGURE，新的配置将写到文件 ini 中，然而，新的值将不会应用到当前运行系统中，只在数据库下次的启动时应用。这意味 ini 文件中的内容可能和 SAP HANA 数据库使用的实际配置值存在不一致。

描述：

设置或删除 ini 文件中的配置参数。 ini 文件配置用于 DEFAULT, SYSTEM, HOST 层。

注意：DEFAULT 层配置不能使用此命令更改或删除。

以下为 ini 文件位置的例子：

DEFAULT: /usr/sap/<SYSTEMNAME>/HDB<INSTANCENUMBER>/exe/config/indexserver.ini
SYSTEM: /usr/sap/<SYSTEMNAME>/SYS/global/hdb/custom/config/indexserver.ini
HOST: /usr/sap/<SYSTEMNAME>/HDB<INSTANCENUMBER>/<HOSTNAME>/indexserver.ini

配置层的优先级：DEFAULT < SYSTEM < HOST。这表示 HOST 层具有最高优先级，跟着是 SYSTEM 层，最后是 DEFAULT 层。最高优先级的配置将应用到运行环境中。如果最高优先级的配置被删除，具有下一个最高优先级的配置将被应用。

系统和监控视图：

目前可供使用的 ini 文件在系统表 M_INIFILES 列出，并且当前配置在系统表 M_INIFILE_CONTENTS 可见。

例子：

修改系统层配置的例子如下：

ALTER SYSTEM ALTER CONFIGURATION ('filename', 'layer') SET ('section1', 'key1') = 'value1', ('section2', 'key2') = 'value2', ... [WITH RECONFIGURE];

ALTER SYSTEM ALTER CONFIGURATION ('filename', 'layer', 'layer_name') UNSET ('section1', 'key1'), ('section2'), ...[WITH RECONFIGURE];

ALTER SYSTEM ALTER SESSION SET

语法：

ALTER SYSTEM ALTER SESSION <session_id> SET <key> = <value>

语法元素：

<session_id> ::= <unsigned_integer>

应当设置变量的会话的 ID。

<key> ::= <string_literal>

会话变量的键值，最大长度为 32 个字符。

<value> ::= <string_literal>

会话变量的期望值，最大长度为 512 个字符。

描述：

使用该命令，你可以设置数据库会话的会话变量：

注意：有几个只读会话变量，你不能使用该命令修改值：APPLICATION, APPLICATIONUSER, TRACEPROFILE。

会话变量可以使用 SESSION_CONTEXT 函数获得，使用 ALTER SYSTEM ALTER SESSION UNSET 命令.取消设置。

例子：

在以下的例子中，你在会话 200006 将变量'MY_VAR' 设为 'dummy'：

ALTER SYSTEM ALTER SESSION 200006 SET 'MY_VAR'= 'dummy';

ALTER SYSTEM ALTER SESSION UNSET

语法：

ALTER SYSTEM ALTER SESSION <session_id> UNSET <key>

语法元素：

<session_id> ::= <unsigned_integer>

应当取消设置变量的会话 ID。

<key> ::= <string_literal>

会话变量的键值，最大长度为 32 个字符。

描述：

使用该命令，你可以取消设置数据库会话的会话变量。

会话可以通过 SESSION_CONTEXT 函数获得。

例子：

获得当前会话的会话变量：

```
SELECT * FROM M_SESSION_CONTEXT WHERE CONNECTION_ID = CURRENT_CONNECTION
```

从特定会话中删除会话变量：

```
ALTER SYSTEM ALTER SESSION 200001 UNSET 'MY_VAR';
```

ALTER SYSTEM CANCEL [WORK IN] SESSION

语法

```
ALTER SYSTEM CANCEL [WORK IN] SESSION <session_id>
```

语法元素：

<session_id> ::= <string_literal>

所需会话的会话 ID。

描述：

通过指定会话 ID 取消当前正在运行的语句。取消的会话将在取消后回滚，执行中的语句将返回错误代码 139(current operation cancelled by request and transaction rolled back)。

例子：

你可以使用下面的查询来获取当前的连接 ID 和它们执行的语句。

```
SELECT C.CONNECTION_ID, PS.STATEMENT_STRING
FROM M_CONNECTIONS C JOIN M_PREPARED_STATEMENTS PS
ON C.CONNECTION_ID = PS.CONNECTION_ID AND C.CURRENT_STATEMENT_ID = PS.STATEM
ENT_ID
WHERE C.CONNECTION_STATUS = 'RUNNING'
```

```
AND C.CONNECTION_TYPE = 'Remote'
```

利用上文中的查询语句获得的连接 ID，你现在可以取消一条正在运行的查询，语句如下：

```
ALTER SYSTEM CANCEL SESSION '200001'
```

ALTER SYSTEM CLEAR SQL PLAN CACHE
语法：

```
ALTER SYSTEM CLEAR SQL PLAN CACHE
```

描述：

SQL PLAN CACHE 存储之前执行的 SQL 语句生成的计划，SAP HANA 数据库使用该计划缓存加速查询语句的执行，如果同样的 SQL 语句 再次执行。计划缓存也收集关于计划准备和执行的数据。

你可以从以下的监控视图中找到更多有关 SQL 缓存计划的内容：
M_SQL_PLAN_CACHE, M_SQL_PLAN_CACHE_OVERVIEW

ALTER SYSTEM CLEAR SQL PLAN CACHE 语句删除所有当前计划缓存没有执行的 SQL 计划。该命令还可以从计划缓存中删除所有引用计数为 0 的计划，并重置所有剩余计划的统计数据。最后，该命令也重置监控视图 M_SQL_PLAN_CACHE_OVERVIEW 的内容。

例子：

```
ALTER SYSTEM CLEAR SQL PLAN CACHE
```

ALTER SYSTEM CLEAR TRACES
语法：

```
ALTER SYSTEM CLEAR TRACES (<trace_type_list>)
```

语法元素：

```
<trace_type_list> ::= <trace_type> [,...]
```

通过在逗号分隔的列表中加入多个 trace_types，您可以同时清除多个追踪。

```
<trace_type> ::= <string_literal>
```

你可以通过设置 trace_type 为以下类型之一，有选择地清除特定的追踪文件：

<trace_type>	Trace Files
ALERT	*alert_*.trc
CLIENT	localclient_*.trc
CRASHDUMP	*.crashdump.*
EMERGENCYDUMP	*.emergencydump.*
*	all *.trc files of services listed below
INDEXSERVER,NAMESERVER,...,DAEMON	open *.trc files of a single service type

描述：

你可以使用 ALTER SYSTEM CLEAR TRACES 清除追踪文件中的追踪内容。当您使用此命令所有开设了 SAP HANA 数据库的跟踪文件将被删除或清除。在分布式系统中，该命令将清除所有主机上的所有跟踪文件。

使用此命令可以减少大跟踪文件使用的磁盘空间，例如，当追踪组件设为 INFO 或 DEBUG。

你可以使用系统表 M_TRACEFILES, M_TRACEFILE_CONTENTS 各自监控追踪文件及其内容。

例子：

要清除警告的跟踪文件，使用下面的命令：

```
ALTER SYSTEM CLEAR TRACES('ALERT');
```

要清除警告和客户端跟踪文件，使用下面的命令：

```
ALTER SYSTEM CLEAR TRACES('ALERT', 'CLIENT');
```

ALTER SYSTEM DISCONNECT SESSION

语法：

```
ALTER SYSTEM DISCONNECT SESSION <session_id>
```

语法元素：

<session_id> ::= <string_literal>

要断开连接的会话 ID。

描述：

你使用 ALTER SYSTEM DISCONNECT SESSION 来断开数据库指定的会话。在断开连接之前，与会话相关联的所有正在运行的操作将被终止。

例子：

你使用如下的命令获得空闲会话的会话 ID：

```
SELECT CONNECTION_ID, IDLE_TIME
FROM M_CONNECTIONS
WHERE CONNECTION_STATUS = 'IDLE' AND CONNECTION_TYPE = 'Remote'
ORDER BY IDLE_TIME DESC
```

你使用如下命令断开会话连接：

```
ALTER SYSTEM DISCONNECT SESSION '200001'
```

ALTER SYSTEM LOGGING

语法：

```
ALTER SYSTEM LOGGING <on_off>
```

语法元素：

```
<on_off> ::= ON | OFF
```

描述：

启动或禁用日志。

日志记录被禁用后，任何日志条目将不会持久化。当完成一个保存点，只有数据区被写入数据。这可能会导致损失已提交的事务，当 indexserver 在加载中时被终止。在终止的情况下，你必须截断，并再次插入的所有数据。

启用日志记录后，你必须执行一个保存点，以确保所有的数据都保存，并且你必须执行数据备份，否则你将不能恢复这些数据。

只在初次加载时使用该命令！

你可以使用 ALTER TABLE ... ENABLE/DISABLE DELTA LOG 为单个列表完成操作。

ALTER SYSTEM RECLAIM DATAVOLUME

语法：

```
ALTER SYSTEM RECLAIM DATA VOLUME [SPACE] [<host_port>] <percentage_of_overload_size>
<shrink_mode>
```

语法元素：

```
<host_port> ::= 'host_name:port_number'
```

指定服务器在持久层应减少的大小：

<percentage_of_overload_size> ::= <int_const>

指定过载的数据量应减少的百分比。

<shrink_mode> ::= DEFRAGMENT | SPARSIFY

指定持续层减少大小的策略，默认值为 DEFRAGEMENT。请注意，SPARSIFY 尚未支持，并保留以备将来使用

描述：

该命令应在持久层中未使用的空间释放时使用。它减少数据量到过载量的 N%; 它的工作原理就像一个硬盘进行碎片整理，散落在页面的数据将被移动到数据量的前端和数据量尾端的自由空间将被截断。

如果省略了<host_port>，该语句将持久化地分配至所有服务器。

例子：

在下面的例子中，架构中的所有服务器持久层将进行碎片整理，并减少至过载尺寸的 120%。

ALTER SYSTEM RECLAIM DATAVOLUME 120 DEFRAGMENT

ALTER SYSTEM RECLAIM LOG

语法：

ALTER SYSTEM RECLAIM LOG

描述：

当数据库中已经积累了大量的日志段时，你可以使用此命令，收回磁盘空间目前未使用的日志段。

日志段的积累，可以以多种方式引起。例如，当自动日志备份不可长期操作或日志保存点被阻塞很长时间，当这样的问题发生时，你只能在修复日志积累的根本原因后，使用 ALTER SYSTEM CLAIM LOG 命令。

例子：

你回收目前未使用的日志段的磁盘空间，使用下面的命令：

ALTER SYSTEM RECLAIM LOG

ALTER SYSTEM RECLAIM VERSION SPACE

语法：

ALTER SYSTEM RECLAIM VERSION SPACE

描述：

执行 MVCC 版本垃圾回收来重用资源。

ALTER SYSTEM RECONFIGURE SERVICE
语法：

ALTER SYSTEM RECONFIGURE SERVICE (<service_name>,<host>,<port>)

语法元素：

<service_name> ::= <string_literal>

你希望重新配置的服务名称。关于可用的服务类型的列表，请参阅监控视图 M_SERVICE_TYPES。

<host> ::= <string_literal>
<port> ::= <integer>

你将重新配置服务的主机和端口号。

描述：

你可以使用 ALTER SYSTEM RECONFIGURE SERVICE 通过应用当前配置参数，重新配置指定的服务。

在使用没有 RECONFIGURE 选项的 ALTER CONFIGURATION 修改多个配置参数使用该命令。参见 **ALTER SYSTEM ALTER CONFIGURATION。**

欲重新配置特定的服务，指定<host> 和 <port>的 值， 而<service_name> 留空。

欲重新配置一种类型的所有服务，指定<service_name> 的值， 而 host> 和 <port>留空。

欲重新配置所有服务，所有参数留空。

例子：

你可以使用以下命令来重新配置 ld8520.sap.com 主机上所有使用端口号 30303 的服务：

ALTER SYSTEM RECONFIGURE SERVICE ('','ld8520.sap.com',30303)

你可以使用以下命令重新配置类型 indexserver 的所有服务：

ALTER SYSTEM RECONFIGURE SERVICE ('indexserver','',0)

参见 ALTER SYSTEM ALTER CONFIGURATION。

ALTER SYSTEM REMOVE TRACES

语法：

ALTER SYSTEM REMOVE TRACES (<host>, <trace_file_name_list>)

<trace_file_name_list> ::= <trace_file>,...

语法元素：

<host> :== <string_literal>

将要删除追踪记录的主机名。

<trace_file_name_list> ::= <trace_file> [,..]

你可以通过在逗号分隔的列表中添加多条 trace_file 记录，同时删除多条追踪记录。

<trace_file> :== see table below.

你可以将 trace_file 设置为以下类型之一：

Trace Type	<trace_file>
ALERT	*alert_*.trc
CLIENT	localclient_*.trc
CRASHDUMP	*.crashdump.*
EMERGENCYDUMP	*.emergencydump.*
*	all *.trc files of services listed below
INDEXSERVER,NAMESERVER,...,DAEMON	open *.trc files of a single service type

描述：

你可以使用该命令删除指定主机中的追踪文件，减少大追踪文件占用的硬盘空间。当某个服务的追踪文件已打开，则不能被删除。这种情况下，你可以使用 ALTER SYSTEM CLEAR TRACES 命令清除追踪文件。

例子：

你使用以下命令删除主机 lu873.sap.com 上所有 ALERT 追踪文件：

ALTER SYSTEM REMOVE TRACES ('lu873.sap.com', '*alert_*.trc');

参见 ALTER SYSTEM CLEAR TRACES。

ALTER SYSTEM RESET MONITORING VIEW
语法：

ALTER SYSTEM RESET MONITORING VIEW <view_name>

语法元素：

<view_name> ::= <identifier>

重设可重置监控视图的名字。

注意：不是所有监控视图可以使用该命令进行重置。可重设视图的名字后缀为"_RESET"，你可以通过其名字判断是否可以重置。

描述：

你可以使用此命令重置指定的监视视图的统计数据。

你可以使用此命令来定义测量的起始点。首先，你重置监控视图，然后执行一个操作。当该操作完成后，查询监控视图"_RESET"版本获得从上次重置之后收集到的统计信息。

例子：

在以下的例子中，你重置"SYS"."M_HEAP_MEMORY_RESET"监控视图：

ALTER SYSTEM RESET MONITORING VIEW "SYS"."M_HEAP_MEMORY_RESET"

ALTER SYSTEM SAVE PERFTRACE

语法：

ALTER SYSTEM SAVE PERFTRACE [INTO FILE <file_name>]

语法元素：

<file_name> ::= <string_literal>

原始性能数据保存的文件。

描述：

你可以使用命令收集. prf 文件中的原始性能数据，保存该信息至.tpt 文件。.tpt 文件保存在 SAP HANA 数据库实例的追踪文件目录中。如果你未指定文件名，则文件将保存为'perftrace.tpt'。

性能追踪数据文件(.tpt)可以从'SAP HANA Computing Studio'->Diagnosis-Files 下载，之后性能追踪可以利用 SAP HANA 实例中的 HDBAdmin 加载和分析。

监控视图：

性能文件的状态可以从 M_PERFTRACE 监控。

例子：

你可以使用如下命令将原始性能数据保存至'mytrace.tpt'文件：

ALTER SYSTEM SAVE PERFTRACE INTO FILE 'mytrace.tpt'

ALTER SYSTEM SAVEPOINT

语法：

ALTER SYSTEM SAVEPOINT

描述：

持久层管理器上执行保存点。保存点是一个数据库的完整连续镜像保存在磁盘上的时间点，该镜像可以用于重启数据库。

通常情况下，保存点定期执行，由[persistence]部分的参数 savepoint_interval_s 配置。对于特殊的（通常测试）的目的，保存点可能会被禁用。在这种情况下，你可以使用此命令来手动执行保存点。

ALTER SYSTEM START PERFTRACE

语法：

ALTER SYSTEM START PERFTRACE [<user_name>] [<application_user_name>] [PLAN_EXECUTION] [FUNCTION_PROFILER] [DURATION <duration_seconds>]

语法元素：

<user_name> ::= <identifier>

限制 perftrace 收集为指定的 SQL 用户名。

<application_user_name> ::= <identifier>

限制为指定的 SQL 用户名收集 perftrace，应用用户可以通过会话变量 APPLICATIONUSER 定义。

PLAN_EXECUTION

收集计划执行细节：

FUNCTION_PROFILER

收集函数级别细节：

<duration_seconds> ::= <numeric literal>

经过 duration_seconds 后，perftrace 自动停止。如果未指定该参数，仅停止有 ALTER SYSTEM STOP PERFTRACE 的 perftrace。

描述：

开始性能追踪。

利用'Explain Plan' 或 'Visualize Plan'，你可以在逻辑级别查看语句的执行。利用'Perfomance Trace'，语句的执行将记录在线程和函数级别。

一次只能有一个 perftrace 活动。

性能追踪文件状态可以从 M_PERFTRACE 监控。

例子：

ALTER SYSTEM START PERFTRACE sql_user app_user PLAN_EXECUTION FUNCTION_PROFILER

ALTER SYSTEM STOP PERFTRACE

语法：

ALTER SYSTEM STOP PERFTRACE

描述：

停止先前启动的性能追踪。停止后，需要利用 ALTER SYSTEM SAVE PERFTRACE 收集和保存性能追踪数据。

例子：

ALTER SYSTEM STOP PERFTRACE

ALTER SYSTEM STOP SERVICE

语法：

ALTER SYSTEM STOP SERVICE <host_port> [IMMEDIATE [WITH COREFILE]]

语法元素：

<host_port> ::= <host_name:port_number> | ('<host_name>',<port_number>)

将停止的服务的位置。

IMMEDIATE

立即停止（中止）服务，无需等待正常关机。

WITH COREFILE

写入 core 文件。

描述：

停止或终止单个或者多个服务。通常，该服务将由守护进程重新启动。

修改了不能在线更改的参数之后使用。

例子：

ALTER SYSTEM STOP SERVICE 'ld8520:30303'

UNSET SYSTEM LICENSE ALL
语法：

UNSET SYSTEM LICENSE ALL

描述：

删除所有已安装的许可证密钥。使用此命令后，系统将被立即锁定，并且需要一个新的有效许可证密钥，然后才可以继续使用。执行该命令需要有 LICENSE ADMIN 权限。

例子：

UNSET SYSTEM LICENSE ALL

会话管理语句

CONNECT
语法：

CONNECT <connect_option>

语法元素：

<connect_option> ::=
<user_name> PASSWORD <password>
| WITH SAML ASSERTION '<xml>'
<password> ::=
<letter_or_digit>...

描述：

通过指定 user_name 和密码或者指定 SAML 断言连接数据库实例。

例子：

CONNECT my_user PASSWORD myUserPass1

SET HISTORY SESSION
语法：

SET HISTORY SESSION TO <when>

语法元素：

<when>:

用户应该指定一个确切的会话旅行的时间。

<when> ::= NOW | COMMIT ID <commit_id> | UTCTIMESTAMP <utc_timestamp>
<commit_id> ::= <unsigned_integer>
<utc_timestamp> ::= <string_literal>

描述：

SET HISTORY SESSION 使当前会话查看历史记录表过去的版本。用户可以指定 COMMIT ID 中的版本或 UTCTIMESTAMP 格式，或者通过指定 NOW 回到当前版本。发布带有 COMMIT ID 或 UTCTIMESTAMP 的 SET HISTORY SESSION 之后，当前会话中看到了一个旧版本的历史记录表，而不能写进系统任何东西。如果给定了 NOW 选项，当前会话恢复到一个正常的会话，看到当前版本的历史记录表，并能写入系统。此命令只适用于历史记录表，普通表的可见性不会受到影响。

例子：

SELECT CURRENT_UTCTIMESTAMP FROM SYS.DUMMY
SELECT LAST_COMMIT_ID FROM M_HISTORY_INDEX_LAST_COMMIT_ID WHERE SESSION_ID = CURRENT_CONNECTION COMMIT
SET HISTORY SESSION TO UTCTIMESTAMP '2012-03-09 07:01:41.428'
SET HISTORY SESSION TO NOW

SET SCHEMA
语法：

SET SCHEMA <schema_name>

语法元素：

Schema_name
schema name string

描述：

你可以修改会话的当前数据集合。当前集合使用数据库对象（如表名）的名字，例如不带集合名字前缀的表名。

SET [SESSION]
语法：

SET [SESSION] <key> = <value>

语法元素：

<key> ::= <string_literal>

会话变量的键值，最大长度为 32 个字符。

<value> ::= <string_literal>

会话变量的期望值，最大长度为 512 个字符。

描述：

你可以使用该命令设置你数据库会话的会话变量，通过提供键值对。

注意：有几个只读会话变量，你不能使用该命令修改值：APPLICATION, APPLICATIONUSER, TRACEPROFILE。

会话变量可以使用 SESSION_CONTEXT 函数获得，使用 UNSET [SESSION]命令取消设置。

例子：

```
SET 'MY_VAR' = 'dummy';

SELECT SESSION_CONTEXT('MY_VAR') FROM dummy;

UNSET 'MY_VAR';
```

UNSET [SESSION]

语法：

```
UNSET [SESSION] <key>
```

语法元素：

<key> ::= <string_literal>

会话变量的键值，最大长度为 32 个字符。

描述：

你可以使用 UNSET [SESSION]取消设置当前会话的会话变量。

注意：有几个只读会话变量，你不能使用该命令修改值：APPLICATION, APPLICATIONUSER, TRACEPROFILE。

例子：

```
SET 'MY_VAR'= 'dummy';
```

```
SELECT SESSION_CONTEXT('MY_VAR') FROM dummy;

UNSET 'MY_VAR';
```

事务管理语句

COMMIT

语法：

```
COMMIT
```

描述：

该系统支持事务一致性，保证了当前作业是完全应用到系统中或者弃用。如果用户希望持久地应用当前作业至系统中，用户应使用 COMMIT 命令。如果 COMMIT 命令发出后，并成功处理，任何改变将应用到当前事务完成的系统中，改变也将对其他未来开始的作业可见。通过 COMMIT 命令已经承诺的工作，将不能恢复。分布式系统中，遵守标准的两阶段提交协议。在第一阶段，事务处理协调器将询问每一位参与者是否准备提交，并将结果发送到第二阶段的参与者。COMMIT 命令只适用于'autocommit'的禁用会话。

例子：

```
COMMIT
```

LOCK TABLE

语法：

```
LOCK TABLE <table_name> IN EXCLUSIVE MODE [NOWAIT]
```

描述：

LOCK TABLE 命令显式地尝试获取表的互斥锁。如果指定了 NO WAIT 选项，其只是试图获得表的锁。如果指定了 NOWAIT 选项不能获得锁，将返回一个错误代码，但是当前事务将回滚。

例子：

```
LOCK TABLE mytaable IN EXCLUSIVE MODE NOWAIT
```

ROLLBACK

语法：

```
ROLLBACK
```

描述：

该系统支持事务一致性，保证了当前作业是完全应用到系统中或者弃用。在事务的中间过程，可以显式恢复，因为由于 ROLLBACK 命令，事务尚未执行。发布 ROLLBACK 命令后，将完全恢复事务

系统做的任何变化，当前会话将处于闲置状态。ROLLBACK 命令只适用于'autocommit'的禁用会话。

例子：

ROLLBACK

SET TRANSACTION

语法：

SET TRANSACTION <isolation_level> | <transaction_access_mode>

语法元素：

isolation_level ::= ISOLATION LEVEL <level>

隔离级别设置数据库中的数据语句级读一致性。如果省略了 isolation_level，默认值为 READ COMMITTED。

level ::= READ COMMITTED | REPEATABLE READ | SERIALIZABLE

READ COMMITTED
READ COMMITTED 隔离级别提供事务过程中语句级别读一致性。在语句开始执行时，事务中的每条语句都能看到已提交状态的数据。这意味着在同一事务中，每个语句可能会看到执行时数据库中不同的快照，因为数据可以在事务中提交。

REPEATABLE READ/SERIALIZABLE
REPEATABLE READ/SERIALIZABLE 隔离级别提供了事务级快照隔离。事务所有语句共享数据库同样的快照。该快照包含所有已提交的事务开始的时间以及事务本身的修改。

transaction_access_mode ::= READ ONLY | READ WRITE

SQL 事务访问模式控制事务是否可以在执行期间修改数据。如果省略了 transaction_access_mode，默认值为 READ ONLY。

READ ONLY
如果设置了 READ ONLY 访问模式，则只允许只读的 SELECT 语句。如果在这种模式下尝试更新或插入操作，会抛出一个异常。

READ WRITE
如果设置了 READ WRITE 访问模式，在一个事务中的语句可以按需自由地读取或更改数据库的数据。

描述：

SAP HANA 数据库使用多版本并发控制 (MVCC) 确保读取操作的一致性。并发的读操作不阻塞并发写操作数据库中的数据的一致视图。并发的读操作不阻塞并发写数据库数据的一致视图。更新操作通过插入数据的新版本而不是覆盖已有数据执行。

指定的隔离级别确定将要使用的锁操作类型。系统同时支持语句级快照隔离和事务级快照隔离。

- 对于语句级快照隔离，使用 READ COMMITED。
- 对于事务级快照隔离，使用 REPEATABLE READ 或者 SERIALIZABLE。

在一个事务中，当记录被插入、更新或删除时，系统对事务执行中，受影响的记录设置互斥锁的持续时间，也对受影响的表设置锁。这样可以保证当表中的记录正在更新时，该表不会被删除或更改。数据库在事务结束时释放这些锁。

注意：读取操作不设置任何数据库中表或行的锁，无论使用何种隔离级别。

数据定义语言和事务隔离

数据定义语言(DDL) 语句(CREATE TABLE, DROP TABLE, CREATE VIEW, etc)总是立即对随后的 SQ 语句生效，无论使用何种隔离级别。对于这种行为的一个例子，请考虑下面的顺序：

1. 一个长期运行 SERIALIZABLE 的隔离事务在表 C 开始操作。
2. 一些 DDL 语句在事务外运行，添加一列新列至表 C。
3. 在 SERIALIZABLE 隔离事务内，只要 DDL 语句执行完毕，新生成的列可以要访问。访问发生不论使用何种隔离级别。

例子：

SET TRANSACTION READ COMMITTED;

访问控制语句

ALTER SAML PROVIDER

ALTER SAML PROVIDER <saml_provider_name> WITH SUBJECT <subject_name> ISSUER <issuer

_distinguished_name>

语法元素：

<saml_provider_name> ::=

<simple_identifier>

<subject_name> ::=

<string_literal>

<issuer_distinguished_name> ::=

<string_literal>

描述：

ALTER SAML PROVIDER 语句修改 SAP HANA 数据库已知的 SAML 提供商的属性。
<saml_provider_name> 必须是一个现有的 SAML 提供商。只有拥有系统权限 USER ADMIN 的数据库用户允许修改 SAML 提供商。

<subject_name> 以及 <issuer_distinguished_name>是 SAML 身份提供程序中证书对应的名字。

系统和监控视图：

SAML_PROVIDER：显示所有 SAML 提供商主题名和 issuer_name。

ALTER USER
语法：

ALTER USER <user_name> <alter_user_option>

语法元素：

<user_name> ::=
<identifier>
<alter_user_option> ::=
PASSWORD <password> [<user_parameter_option>]
| <user_parameter_option>
| IDENTIFIED EXTERNALLY AS <external_identity> [<user_parameter_option>]
| RESET CONNECT ATTEMPTS
| DROP CONNECT ATTEMPTS
| DISABLE PASSWORD LIFETIME
| FORCE PASSWORD CHANGE
| DEACTIVATE [USER NOW]
| ACTIVATE [USER NOW]
| DISABLE <authentication_mechanism>
| ENABLE <authentication_mechanism>
| ADD IDENTITY <provider_identity>...
| ADD IDENTITY <external_identity> FOR KERBEROS
| DROP IDENTITY <provider_info>...
| DROP IDENTITY FOR KERBEROS
| <string_literal>
<authentication_mechanism> ::= PASSWORD | KERBEROS | SAML
<provider_identity> ::=

<mapped_user_name> FOR SAML PROVIDER <saml_provider_name>
| <external_identity> FOR KERBEROS
<mapped_user_name> ::=
ANY
| <string_literal>
<saml_provider_name> ::=
<simple_identifier>
<provider_info> ::= FOR SAML PROVIDER <saml_provider_name>
<password> ::=
<letter_or_digit>...
<user_parameter_option> ::=
<set_user_parameters> [<clear_user_parameter_option>]
| <clear_user_parameter_option>
<set_user_parameters> ::=
SET PARAMETER CLIENT = <string_literal>
<clear_user_parameter_option> ::=
CLEAR PARAMETER CLIENT
| CLEAR ALL PARAMETERS
<external_identity> ::=
<simple_identifier>

描述：

ALTER USER 语句修改数据库用户。<user_name>必须指定一个现有的数据库用户。
每个用户可以为自己执行 ALTER USER。但并非所有<alter_user_option>可以由用户自己指定。对于<alter_user_option>其他用户，只有拥有系统权限 USER ADMIN 权限的用户可以执行 ALTER USER。

使用 PASSWORD 创建的用户不能修改为 EXTERNALLY，反之亦然。但他们的<password>或者<external_identity>是可以修改的。

你可以使用此命令更改用户的密码。密码的修改必须遵循当前数据库定义的规则，包括最小密码长度和定义的字符类型（大写、小写、数字、特殊字符）必须是密码的一部分。用户根据指定数据库实例定义的策略，必须定期更换密码，或者由首次连接到数据库实例的用户，自己更改密码。

你可以更改外部认证。外部用户使用外部系统需要进行身份验证，例如，Kerberos 系统。这些用户没有密码，但是有 Kerberos 实体名称。有关外部身份的详细信息，请联系您的域管理员。

<user_parameter_option>可以用来设置、修改或者清除用户参数 CLIENT。
<set_user_parameters>用来为数据库中的用户设置用户参数 CLIENT。
当使用报表时，该用户参数 CLIENT 可以用于限制用户 <user_name>访问有关特定客户端的信息。
<user_parameter_option>不能由用户自己指定。

如果在成功连接(正确的用户/密码组合)前，达到参数 MAXIMUM_INVALID_CONNECT_ATTEMPTS（参见监控视图 M_PASSWORD_POLICY）定义的错误次数，用户将在允许重新连接前，被锁定几

分钟。拥有系统权限 USER ADMIN 的用户或者用户自己，可以使用命令 ALTER USER <user_name> RESET CONNECT ATTEMPTS 可以删除已发生的无效连接尝试的信息。

拥有系统权限 USER ADMIN 的用户可以使用命令 ALTER USER <user_name> DISABLE PASSWORD LIFETIME 排除用户<user_name>的所有密码生命周期检查。这应该只为技术用户使用，而非正常的数据库用户。

拥有系统权限 USER ADMIN 的用户可以使用命令 ALTER USER <user_name> FORCE PASSWORD CHANGE 强制用户<user_name>在下次连接后立即修改密码，然后才可以正常工作。

拥有系统权限 USER ADMIN 的用户可以使用命令 ALTER USER <user_name> DEACTIVATE USER NOW 关闭/锁定用户<user_name>的账号。用户<user_name>的账号关闭/锁定之后，用户将不能连接到 SAP HANA 数据库。欲重新激活/解锁用户<user_name>，系统权限 USER ADMIN 用户使用命令 USER <user_name> ACTIVATE USER NOW，或者，在用户使用 PASSWORD 身份验证机制的情况下，使用 ALTER USER <user_name> PASSWORD <password>重设用户密码。

拥有系统权限 USER ADMIN 的用户可以使用命令 ALTER USER <user_name> ACTIVATE USER NOW 重新激活/解锁之前已经关闭的用户<user_name>账号。

配置参数：

有关密码的配置参数，可以查看监控视图 M_PASSWORD_POLICY。这些参数存储在 indexserver.ini，'password policy'部分中。相关的参数描述可以在 SAP HANA 安全指南,附录,密码策略参数中找到。

系统和监控视图:

USERS: 显示所有用户、用户的创建者、创建时间和当前状态的信息。
USER_PARAMETERS：显示定义的 user_parameters，目前只提供 CLIENT。
INVALID_CONNECT_ATTEMPTS：显示每个用户无效连接的尝试次数。
LAST_USED_PASSWORDS: 显示用户上次密码修改日期。
M_PASSWORD_POLICY：显示描述密码所允许的样式的配置参数及其生命周期。

例子：

在可能使用给定的密码连接数据库以及已有的 SAML 提供商 OUR_PROVIDER 断言之前，用户名为 NEW_USER 的用户已经创建完成。由于断言将提供数据库用户名，<mapped_user_name>设为 ANY。这由如下的语句完成:

```
CREATE USER new_user PASSWORD Password1 WITH IDENTITY ANY FOR SAML PROVIDER
OUR_PROVIDER;
```

现在，该用户将被强制修改密码，用户被禁止使用 SAML。

ALTER USER new_user FORCE PASSWORD CHANGE;

ALTER USER new_user DISABLE SAML;

假设用户已经过于频繁的尝试一个错误的密码，管理员将重置无效的连接尝试数为零。

ALTER USER new_user RESET CONNECT ATTEMPTS;

用户 new_user 应当允许使用 KERBEROS 机制进行身份验证。因此，需要定义该连接的外部身份。

ALTER USER new_user ADD IDENTITY 'testkerberosName' FOR KERBEROS;
ALTER USER new_user ENABLE KERBEROS;

另一方面，用户 new_user 将放松使用 SAML 提供商 OUR_PROVIDER 断言的可能性。

ALTER USER new_user DROP IDENTITY FOR SAML PROVIDER OUR_PROVIDER;

最后，管理员希望禁止此用户 new_user 的所有连接，因为他最近执行的可疑操作。

ALTER USER new_user DEACTIVATE;

CREATE ROLE
语法：

CREATE ROLE <role_name>

语法元素：

<role_name> ::= <identifier>

描述：

CREATE ROLE 语句创建一个新的角色。
只有拥有系统权限 ROLE ADMIN 的用户可以创建新角色。
指定的角色名称不能与现有用户或角色的名称相同。

角色是权限的一个命名集合，可以授予一个用户或角色。如果你想允许多个数据库用户执行相同的操作，你可以创建一个角色，授予该角色所需的权限，并将角色授予不同的数据库用户。

每个用户允许将权限授予一个已有的角色，但只有只有拥有系统权限 ROLE ADMIN 的用户可以将角色授予角色和用户。

SAP HANA 数据库提供了四种角色：

PUBLIC
每个数据库用户默认已被授予该角色。

该角色包括只读访问系统视图、监控视图和一些存储过程的执行权限。这些权限可以被撤销。该角色可以授予过后将被撤销的权限。

MODELING
该角色包含使用 SAP HANA Studio 信息建模器所需的权限。

CONTENT_ADMIN
该角色包含与 MODELING 角色相同的角色，但是使用扩展该角色将被允许授予其他用户这些权限。此外，它包含了与导入对象工作的元库权限。

MONITORING
该角色包含所有元数据、当前的系统状态、监控视图和服务器统计数据的只读访问。

系统和监控视图：

ROLES：显示所有角色、它们的创建者和创建时间。
GRANTED_ROLES：显示每个用户或角色被授予的角色。
GRANTED_PRIVILEGES：显示每个用户或角色被授予的权限。

例子：

创建名称为 role_for_work_on_my_schema 的角色。

```
CREATE ROLE role_for_work_on_my_schema;
```

CREATE SAML PROVIDER
语法：

```
CREATE SAML PROVIDER <saml_provider_name> WITH SUBJECT <subject_distinguished_name>

ISSUER <issuer_distinguished_name>
```

语法元素：

```
<saml_provider_name> ::=
<simple_identifier>
<subject_distinguished_name> ::=
<string_literal>
<issuer_distinguished_name> ::=
<string_literal>
```

描述：

CREATE SAML PROVIDER 语句定义 SAP HANA 数据库已知的 SAML 提供商。<saml_provider_name>必须与已有的 SAML 提供商不同。

只有拥有系统权限 USER ADMIN 的用户可以创建 SAML 提供商，每个有该权限的用户允许删除任何 SAML 提供商。

需要一个现有的 SAML 提供商，能够为用户指定 SAML 连接。<subject_distinguished_name> 和 <issuer_distinguished_name>是 SAML 提供商使用的 X.509 证书的主题和发布者的 X.500 可分辨名字。这些名字的语法可以在 ISO/IEC 9594-1 中找到。
SAML 概念的详细细节可以在 Oasis SAML 2.0 中找到。

系统和监控视图：

SAML_PROVIDERS：显示所有 SAML 提供商主题名和发布者名字。

例子：

创建一个名称为 gm_saml_provider 的 SAML 提供商，指定主题和发布者所属的公司。

```
CREATE SAML PROVIDER gm_saml_provider
WITH SUBJECT 'CN = wiki.detroit.generalmotors.corp,OU = GMNet,O = GeneralMotors,
C = EN'
ISSUER 'E = John.Do@gm.com,CN = GMNetCA,OU = GMNet,O = GeneralMotors,C = EN';
```

CREATE USER
语法：

```
CREATE USER <user_name>
[PASSWORD <password>]
[IDENTIFIED EXTERNALLY AS <external_identity>]
[WITH IDENTITY <provider_identity>...]
[<set_user_parameters>]
```

语法元素：

```
<user_name> ::=
<identifier>
<password> ::=
<letter_or_digit>...
<external_identity> ::=
<simple_identifier>
| <string_literal>
<provider_identity> ::=
<mapped_user_name> FOR SAML PROVIDER <saml_provider_name>
| <external_identity> FOR KERBEROS
<mapped_user_name> ::=
ANY
| <string_literal>
<saml_provider_name> ::=
<simple_identifier>
```

<set_user_parameters> ::=
SET PARAMETER CLIENT = <string_literal>

描述：

CREATE USER 创建一个新的数据库用户。
只有拥有系统权限 USER ADMIN 的用户可以创建另一个数据库用户。
指定的用户名必须不能与已有的用户名、角色名或集合名相同。
SAP HANA 数据库提供的用户有：SYS, SYSTEM, _SYS_REPO,_SYS_STATISTICS。

数据库中的用户可以通过不同的机制进行身份验证，内部使用密码的身份验证机制和而外部则使用 Kerberos 或 SAML 等机制验证。用户可以同时使用不止一种方式进行身份验证，但在同一时间，只有一个密码和一个外部识别有效。与之相反的是，同一时间可以有一个以上<provider_identity>为一个用户存在。至少需指定一种验证机制允许用户连接和在数据库实例上工作。

由于兼容性原因，语法 IDENTIFIED EXTERNALLY AS <external_identity>以及<external_identity> FOR KERBEROS 会继续使用。

密码必须遵循当前数据库定义的规则。密码的修改必须遵循当前数据库定义的规则，包括最小密码长度和定义的字符类型（大写、小写、数字、特殊字符）必须是密码的一部分.用户根据指定数据库实例定义的策略，必须定期更换密码。在执行 CREATE USER 命令期间提供的密码将被视为已提供，<user_name>将会修改为大写作为每个<simple_identifier>。

外部用户使用外部系统进行身份验证，例如 Kerberos 系统。这些用户没有密码，但是有 Kerberos 实体名称。有关外部身份的详细信息，请联系您的域管理员。

如果 ANY 作为映射的用户名，SAML 断言将包含断言生效的数据库用户名。<saml_provider_name>必须指定一个已有的 SAML 提供商。

<set_user_parameters>可以用于为数据库中的用户设置用户参数 CLIENT。
当使用报表时，该用户参数 CLIENT 可以用于限制用户 <user_name>访问有关特定客户端的信息。<user_parameter_option>不能由用户自己指定。

对于每个数据库用户，数据集合将以包含用户名方式创建。这是不能显式删除。用户删除时，该集合也将被删除。数据库用户拥有该集合，并当他不显式指定集合名称时，作为自己的默认集合使用。

配置参数：

与密码相关的配置参数可以在监控视图 M_PASSWORD_POLICY 查看。这些参数存储 indexserver.ini 的'password policy'部分中。相关的参数描述可以在 SAP HANA 安全指南,附录,密码策略参数中找到。

系统和监控视图：

USERS: 显示所有用户、用户的创建者、创建时间和当前状态的信息。
USER_PARAMETERS：显示定义的 user_parameters，目前只提供 CLIENT。
INVALID_CONNECT_ATTEMPTS：显示每个用户无效连接的尝试次数。
LAST_USED_PASSWORDS: 显示用户上次密码修改日期。
M_PASSWORD_POLICY：显示描述密码所允许的样式的配置参数及其生命周期。
SAML_PROVIDERS；显示已有的 SAML 提供商。
SAML_USER_MAPPING：显示每个 SAML 提供商的映射用户名。

例子：

在可能使用给定的密码连接数据库以及已有的 SAML 提供商 OUR_PROVIDER 断言之前，用户名为 NEW_USER 的用户已经创建完成。由于断言将提供数据库用户名，<mapped_user_name>设为 ANY。这由如下的语句完成:

CREATE USER new_user PASSWORD Password1 WITH IDENTITY ANY FOR SAML PROVIDER OUR_PROVIDER;

DROP ROLE
语法：

DROP ROLE <role_name>

语法元素：

<role_name> ::= <identifier>

例子：

DROP ROLE 语句删除角色。<drop_name>必须指定已经存在的角色。
只有拥有系统权限 ROLE ADMIN 的用户可以删除角色。任何有该权限的用户允许删除任意角色。
只有 SAP HANA 提供的角色可以删除：PUBLIC, CONTENT_ADMIN, MODELING and MONITORING。

如果一个角色授予用户或角色，在角色删除时将被撤销。撤销角色可能会导致一些视图无法访问或者存储过程再也不工作，如果一个视图或存储过程依赖于该角色中的任意权限，会发生这种情况。

系统和监控视图：

ROLES：显示所有角色、它们的创建者和创建时间。
GRANTED_ROLES：显示每个用户或角色被授予的角色。
GRANTED_PRIVILEGES：显示每个用户或角色被授予的权限。

例子：

创建名为 role_for_work_on_my_schema 的角色，随后立即删除。

CREATE ROLE role_for_work_on_my_schema;

DROP ROLE role_for_work_on_my_schema;

DROP SAML PROVIDER
语法：

DROP SAML PROVIDER <saml_provider_name>

语法元素：

<saml_provider_name> ::=

<simple_identifier>

描述：

DROP SAML PROVIDER 语句删除指定的 SAML 提供商。<saml_provider_name>必须是一个已有的 SAML 提供商。如果指定的 SAML 提供商正在被 SAP HANA 用户使用，则该提供商不能被删除。只有拥有系统 USER ADMIN 权限的用户可以删除 SAML 提供商。

系统和监控视图：

SAML_PROVIDERS:显示所有 SAML 提供商主题名称和 issuer_name。

DROP USER
语法：

DROP USER <user_name> [<drop_option>]

语法元素：

<user_name> ::=

<identifier>

<drop_option> ::= CASCADE | RESTRICT

Default = RESTRICT

描述：

DROP USER 语句删除数据库用户。<user_name>必须指定一个已有的数据库用户。

只有拥有系统 USER ADMIN 权限的用户可以删除用户。拥有该权限的用户可以删除任何用户。SAP HANA 数据库提供的用户不能删除：SYS, SYSTEM, _SYS_REPO,_SYS_STATISTICS。

如果显式或隐式指定了<drop_option> RESTRICT，则当用户为数据集合的所有者或以及创建了其他集合，或者该用户集合下存有非本人创建的对象时，该用户不能被删除。
如果指定了<drop_option> CASCADE，包含用户名的集合和属于该用户的集合，连同所有存在这些集合中的对象（即使是由其他用户创建）一起删除。用户拥有的对象，即使为其他集合中的一部分，将被删除。依赖于已删除对象的对象将被删除，即使已删除的用户所拥有的公共同义词。
已删除对象的权限将被撤销，授予已删除用户的权限也将被撤销。撤销权限可能会造成更多的撤销操作，如果这些权限被进一步授予。

已删除用户创建的用户和由他们创建的角色将不会被删除。已删除的用户创建的审核策略也不会被删除。

如果用户存在一个已打开的会话，仍然可以删除该用户。

系统和监控视图：

已删除用户将从以下视图删除：
USERS: 显示所有用户、用户的创建者、创建时间和当前状态的信息。
USER_PARAMETERS：显示定义的 user_parameters，目前只提供 CLIENT。
INVALID_CONNECT_ATTEMPTS：显示每个用户无效连接的尝试次数。
LAST_USED_PASSWORDS: 显示用户上次密码修改日期。
M_PASSWORD_POLICY：显示描述密码所允许的样式的配置参数及其生命周期。

对象的删除可能影响所有描述对象的系统视图，例如 TABLES, VIEWS,PROCEDURES, ...
对象的删除可能影响描述权限的视图：例如 GRANTED_PRIVILEGES 以及所有监控视图，例如 M_RS_TABLES, M_TABLE_LOCATIONS, ...

例子：

例如，使用这条语句创建名称为 NEW_USERd 的用户：

```
CREATE USER new_user PASSWORD Password1;
```

已有的用户 new_user 将被删除，连同其所有对象一起：

```
DROP USER new_user CASCADE;
```

GRANT
语法：

```
GRANT <system_privilege>,... TO <grantee> [WITH ADMIN OPTION]
| GRANT <schema_privilege>,... ON SCHEMA <schema_name> TO <grantee> [WITH GRANT OPT
ION]
| GRANT <object_privilege>,... ON <object_name> TO <grantee> [WITH GRANT OPTION]
| GRANT <role_name>,... TO <grantee> [WITH ADMIN OPTION]
| GRANT STRUCTURED PRIVILEGE <privilege_name> TO <grantee>
```

语法元素：
<system_privilege> ::=
AUDIT ADMIN | BACKUP ADMIN
| CATALOG READ | CREATE SCENARIO
| CREATE SCHEMA | CREATE STRUCTURED PRIVILEGE
| DATA ADMIN | EXPORT
| IMPORT | INIFILE ADMIN
| LICENSE ADMIN | LOG ADMIN
| MONITOR ADMIN | OPTIMIZER ADMIN
| RESOURCE ADMIN | ROLE ADMIN
| SAVEPOINT ADMIN | SCENARIO ADMIN
| SERVICE ADMIN | SESSION ADMIN
| STRUCTUREDPRIVILEGE ADMIN | TRACE ADMIN
| USER ADMIN | VERSION ADMIN
| <identifier>.<identifier>

系统权限用来限制管理任务。定义如下的系统权限：

AUDIT ADMIN

该权限控制以下审计有关命令的执行：CREATE AUDIT POLICY, DROP AUDIT POLICY and ALTER AUDIT POLICY.

BACKUP ADMIN

该权限授权 ALTER SYSTEM BACKUP 命令来定义和启动备份进程或执行恢复过程。

CATALOG READ

该权限赋予所有用户未经过滤的只读访问所有的系统和监控视图。正常情况下，这些视图的内容根据正在访问用户的权限过滤。权限 CATALOG READ 使用户有只读访问所有的系统和监控视图的内容。

CREATE SCENARIO

该权限控制计算场景和多维数据集（数据库计算）的创建。

CREATE SCHEMA

该权限控制使用 CREATE SCHEMA 命令创建数据库数据集合。每个用户都有个集合。拥有该权限，用户允许创建更多的集合。

CREATE STRUCTURED PRIVILEGE

该权限授权创建结构化权限（分析权限）。注意，只有分析权限的所有者可以进一步授予其他用户或者角色，以及撤销。

DATA ADMIN

该强大的权限授权读取系统和监控视图中的所有数据，包括在 SAP HANA 数据库中执行 DDL (Data Definition Language) 以及 DDL 命令。这表示拥有该权限的用户不能选择或者修改存储在其他用户表中的数据，但是可以修改表的定义或甚至删除该表。

EXPORT

该权限授权通过 EXPORT TABLE 命令导出数据库中的活动。注意，除了该权限，用户仍需要将要导出的源表 SELECT 权限。

IMPORT

该权限授权通过 IMPORT TABLE 命令导入数据库中的活动。注意，除了该权限，用户仍需要将要导入的目标表 SELECT 权限。

INIFILE ADMIN

该权限授权修改系统设置的不同方式。

LICENSE ADMIN

该权限授权 SET SYSTEM LICENSE 命令安装一个新的许可。

LOG ADMIN

该权限授权 ALTER SYSTEM LOGGING [ON|OFF] 命令启用或禁用 刷新日志机制。

MONITOR ADMIN

该权限授权关于 EVENT 的 ALTER SYSTEM 命令。

OPTIMIZER ADMIN

该权限授权关于 SQL PLAN CACHE 和 ALTER SYSTEM 的 ALTER SYSTEM 命令。

UPDATE STATISTICS 命令影响查询优化器的行为。

RESOURCE ADMIN

该权限授权关于资源，例如 ALTER SYSTEM RECLAIM、DATAVOLUME 和 ALTER SYSTEM RESET MONITORING VIEW 的命令，并且授权 Management Console 中的许多命令。

ROLE ADMIN

该权限授权使用 CREATE ROLE 命令创建和删除角色。同时也授权使用 GRANT 和 REVOKE 命令授予和撤销角色。

SAVEPOINT ADMIN

该权限使用 ALTER SYSTEM SAVEPOINT 命令授权保存点流程的执行。

SCENARIO ADMIN

该权限授权所有计算场景相关的活动，包括新建。

SERVICE ADMIN

该权限授权 ALTER SYSTEM [START|CANCEL|RECONFIGURE] 命令，用于管理数据库中的系统服务。

SESSION ADMIN

该权限授权会话相关的 ALTER SYSTEM 命令，停止或重新连接用户会话或者修改会话参数。

STRUCTUREDPRIVILEGE ADMIN

该权限授权结构化权限的创建、重新激活和删除。

TRACE ADMIN

该权限授权数据库追踪文件的操作 ALTER SYSTEM [CLEAR|REMOVE] TRACES 命令。

USER ADMIN

该权限授权使用 CREATE USER, ALTER USER, and DROP 命令创建和修改用户。

VERSION ADMIN

该权限授权多版本并发控制(MVCC) ALTER SYSTEM RECLAIM VERSION SPACE command 命令。

<identifier>.<identifier>
SAP HANA 数据库组件可以创建自己需要的权限。这些权限使用组件名作为系统权限的第一标识符，使用组件-权限-名字作为第二标识符。目前元库使用该特点。有关名为 REPO.<identifier>的权限，请参阅元库手册。

<schema_privilege> ::=
CREATE ANY
| DEBUG
| DELETE
| DROP
| EXECUTE
| INDEX
| INSERT
| SELECT
| TRIGGER
| UPDATE

数据集合权限用于集合和存储在该集合中对象的访问和修改。集合权限的定义如下：

CREATE ANY
该权限允许用户在数据库中创建各种对象，尤其是表、视图、序列、同义词、SQL 脚本、或者存储过程。

DELETE, DROP, EXECUTE, INDEX, INSERT, SELECT, UPDATE
指定的权限被授予每个目前和以后存储在集合中的每个对象。有关权限详细说明，请参阅下面的描述对象权限的部分，请检查以下权限适用于哪些类型的对象。

<object_privilege> ::=
ALL PRIVILEGES
| ALTER
| DEBUG
| DELETE
| DROP
| EXECUTE
| INDEX
| INSERT
| SELECT
| TRIGGER
| UPDATE
| <identifier>.<identifier>

对象权限用于限制用户访问和修改数据库对象，例如表、视图、序列或者存储过程以及诸如此类。并不是所有的这些权限适用于所有类型的数据库对象。
对于对象类型允许的权限，见下表。
对象权限的定义如下：

ALL PRIVILEGES
该权限为所有 DDL（数据定义语言）和 DML（数据操纵语言）权限的组合。该权限一方面是授予人目前有的和允许进一步授予的权限，另一方面，特定对象上可以被授予的权限。该组合为给定的授予人和对象进行动态评估。ALL PRIVILEGES 适用于表或试图。

ALTER

该 DDL 权限授权对象的 ALTER 命令。

DEBUG

该 DML 权限授权 存储过程或者计算视图的调试功能。

DELETE

该 DML 权限授权对象的 DELETE 和 TRUNCATE 命令。

DROP

该 DDL 权限授权对象的 DROP 命令。

EXECUTE

该 DML 权限授权 SQL Script 函数或者使用 CALLS 或 CALL 命令的存储过程。

INDEX

该 DDL 权限授权对象索引的创建、修改或者删除。

INSERT

该 DML 权限授权对象的 INSERT 命令。INSERT 连同 UPDATE 权限一起允许使用对该对象的 REPLACE 和 UPSERT 命令。

SELECT

该 DML 权限授予对象的 SELECT 命令或者序列的使用。

TRIGGER

该 DDL 权限授权指定表或者指定集合中表的 CREATE TRIGGER / DROP TRIGGER 命令。

UPDATE

该 DML 权限授权对象的 UPDATE 命令 INSERT 连同 UPDATE 权限一起允许使用对该对象的 REPLACE 和 UPSERT 命令。

<identifier>.<identifier>
SAP HANA 数据库组件可以创建自己需要的权限。这些权限使用组件名作为系统权限的第一标识符，使用组件-权限-名字作为第二标识符。目前元库使用该特点。有关名为 REPO.<identifier>的权限，请参阅元库手册。

Privilege	Schema	Table	View	Sequence	Function/Procedure
ALL PRIVILEGES	---	YES	YES	---	---
ALTER	YES	YES	---	---	YES
CREATE ANY	YES	---	---	---	---
DEBUG	YES	---	YES	---	YES
DELETE	YES	YES	YES	---	---
DROP	YES	YES	YES	YES	YES
EXECUTE	YES	---	---	---	YES
INDEX	YES	YES	---	---	---
INSERT	YES	YES	YES	---	---
SELECT	YES	YES	YES	YES	---
TRIGGER	YES	YES	---	---	---
UPDATE	YES	YES	YES	---	---

对视图的 DELETE, INSERT and UPDATE 操作只适用于可更新的视图，表示这些视图遵守这样的一些限制：不包含联接、 UNION，没有聚合以及进一步的一些限制。

DEBUG 只对计算视图适用，而非其他类型的视图。
这些限制适用于同义词，以及同义词代表的对象也适用。

<object_name> ::=
<table_name>
| <view_name>
| <sequence_name>
| <procedure_name>
| <synonym_name>

对象权限用于限制用户访问和修改数据库对象，例如表、视图、序列、存储过程和同义词。

<grantee> :: =
<user_name>
| <role_name>

grantee 可以是一个用户或者角色。在权限或角色授予角色的情况下，角色授予的所有用户，将有指定的权限或角色。

角色是权限的一个命名集合，可以授予一个用户或角色。
如果你想允许多个数据库用户执行相同的操作，你可以创建一个角色，授予该角色所需的权限，并将角色授予不同的数据库用户。

当授予角色给角色时，将建立一颗角色树。当将一个角色(R)授予另一个角色或者用户(G)，G 将拥有所有直接授予 R 的权限和角色。

<user_name> ::= <identifier>
<role_name> ::= <identifier>
<schema_name> ::= <identifier>
<table_name> ::= <identifier>
<view_name> ::= <identifier>
<sequence_name> ::= <identifier>
<procedure_name> ::= <identifier>
<synonym_name> ::= <identifier>
<privilege_name> ::= <identifier>

描述：

GRANT 用于授予权限和结构化权限给用户和角色，也用于授予权限给用户和其他角色。

指定的用户、角色、对象和结构化权限必须在使用 GRANT 命令前已经存在。
只有拥有权限并且允许进一步授予权限的用户才能授予权限。每个拥有 ROLE ADMIN 权限的用户允许授予角色给其他角色和用户。

用户不能授予自己权限。
SYSTEM 用户有至少一个系统权限和 PUBLIC 角色。所有其他用户也有 PUBLIC 角色。这些权限和角色不能自己撤销。

虽然 SYSTEM 用户拥有许多权限，该用户不能选择或者修改其他用户的表，如果他没有显式地授权可以这样做。
SYSTEM 用户有在自己默认集合中创建对象的权限，名字和用户本身一样。
对于由用户创建的表，他们拥有所有权限并且可以将权限授予给用户和角色。
对依赖于例如基于表的视图的其他对象，可能发生的是用户如果没有底层对象的权限则在依赖对象也没有权限。或者可能发生的是用户有权限，但是不允许进一步授权。该用户将不能授予这些权限。

WITH ADMIN OPTION 和 WITH GRANT OPTION 指定了已分配的权限可以被特定的用户进一步分配，或者被拥有指定角色的用户分配。

使用 GRANT STRUCTURED PRIVILEGE <structured_privilege_name>，一个之前定义过的分析权限（基于通用结构化权限）被分配给用户或角色。该分析权限用于限制只读访问分析视图、属性视图和计算视图特定的数据，通过过滤属性值。

系统和监控视图：

USERS: 显示所有用户、用户的创建者、创建时间和当前状态的信息。
ROLES：显示所有角色、它们的创建者和创建时间。

GRANTED_ROLES：显示每个用户或角色被授予的角色。
GRANTED_PRIVILEGES：显示每个用户或角色被授予的权限。

例子：

假设已经创建了拥有创建集合、角色和用户权限的用户，他新建了数据集合：

CREATE SCHEMA myschema;

另外，他还在该集合中新建了一张名为 work_done 的表。

CREATE TABLE myschema.work_done (t TIMESTAMP, user NVARCHAR (256), work_done VARCHAR (256);

他创建了一个新的用户 named worker，在可能使用给定的密码和名为
role_for_work_on_my_schema 的角色连接数据库

CREATE USER worker PASSWORD His_Password_1;
CREATE ROLE role_for_work_on_my_schema;

他将其集合下所有对象的 SELECT 权限授予 role_for_work_on_my_schema 角色：

GRANT SELECT ON SCHEMA myschema TO role_for_work_on_my_schema;

另外，用户将表 work_done to 的 INSERT 权限授予 role_for_work_on_my_schema 角色：

GRANT INSERT ON myschema.work_done TO role_for_work_on_my_schema;

接着，他将角色授予新的用户：

GRANT role_for_work_on_my_schema TO worker WITH GRANT OPTION;

另外，worker 用户被直接授予表删除权限。该权限的选项允许进一步授予此权限。

GRANT DELETE ON myschema.work_done TO worker;

现在，用户将创建任何类型对象的权限授予 worker 用户：

GRANT CREATE ANY ON SCHEMA myschema TO worker;

结果，worker 用户拥有集合 myschema 下所有表和视图的 SELECT 权限，表 myschema.work_done 的 INSERT 和 DELETE 权限，以及在集合 myschema 下创建对象的权限。另外，该用户允许授予表 myschema.work_done 的 DELETE 权限给其他用户和角色。

第二个例子中，用户有相应的权限，包括允许进一步授予权限、将系统权限 INIFILE ADMIN 和 TRACE ADMIN 授予已有的用户 worker。他允许 worker 进一步授予这些权限。

GRANT INIFILE ADMIN, TRACE ADMIN TO worker WITH ADMIN OPTION;

REVOKE
语法：

REVOKE <system_privilege>,... FROM <grantee>
|| REVOKE <schema_privilege>,... ON SCHEMA <schema_name> FROM <grantee>
| REVOKE <object_privilege>,... ON <object_name> FROM <grantee>
| REVOKE <role_name>,... FROM <grantee>
| REVOKE STRUCTURED PRIVILEGE <privilege_name> FROM <grantee>

语法元素：

有关语法元素的定义，参见 GRANT。

描述：

REVOKE 语句撤销指定的角色或者结构化权限或者从指定用户或角色的指定对象中撤销权限。
只有拥有授权的用户可以撤销该权限。这对于有 ROLE ADMIN 的用户和角色的撤销也一样。
SYSTEM 用户有至少一个系统权限和 PUBLIC 角色。所有其他用户也有 PUBLIC 角色。这些权限和角色不能自己撤销。

如果用户也被授予一个角色，就不可能撤销属于该角色的一些权限。这种情况下，必须撤销所有角色，并且需要用户已授予给他的权限。
如果一个角色授予用户或角色，在角色删除时将被撤销。撤销角色可能会导致一些视图无法访问或者存储过程再也不工作，如果一个视图或存储过程依赖于该角色中的任意权限，会发生这种情况。
撤销已用 WITH GRANT OPTION 或 WITH ADMIN OPTION 授权的权限将导致不仅从指定的用户中撤销，也将从所有该用户直接或间接授权给用户和角色的权限中撤销。

由于权限可以用个不同的用户授给用户或角色，用户撤销该权限并一定意味着，该用户将失去这权限。有关语法元素的详请，请参见 GRANT。

系统和监控视图：

USERS: 显示所有用户、用户的创建者、创建时间和当前状态的信息。
ROLES：显示所有角色、它们的创建者和创建时间。
GRANTED_ROLES：显示每个用户或角色被授予的角色。
GRANTED_PRIVILEGES：显示每个用户或角色被授予的权限。

例子：

假设用户已经执行如下语句：

CREATE USER worker PASSWORD His_Password_1;
CREATE ROLE role_for_work_on_my_schema;

```
CREATE TABLE myschema.work_done (t TIMESTAMP, user NVARCHAR (256), work_done VARCHAR (256);
GRANT SELECT ON SCHEMA myschema TO role_for_work_on_my_schema;
GRANT INSERT ON myschema.work_done TO role_for_work_on_my_schema;
GRANT role_for_work_on_my_schema TO worker;
GRANT TRACE ADMIN TO worker WITH ADMIN OPTION;
GRANT DELETE ON myschema.work_done TO worker WITH GRANT OPTION;
```

已授权的用户允许撤销这些权限。他从角色中撤销权限，因此，暗示着从所有已授予角色的用户撤销权限。另外，worker 用户将不再有 TRACE ADMIN 权限。撤销权限将导致撤回操作发生至 worker 用户授予该权限的所有用户。

```
REVOKE SELECT ON SCHEMA myschema FROM role_for_work_on_my_schema;
REVOKE TRACE ADMIN FROM worker;
```

数据导入导出语句

EXPORT

语法：

```
EXPORT <object_name_list> AS <export_format> INTO <path> [WITH <export_option_list>

]
```

语法元素：

WITH <export_option_list>:

可以使用 WITH 子句传入 EXPORT 选项。

```
<object_name_list> ::= <OBJECT_NAME>,... | ALL
<export_import_format> ::= BINARY | CSV
<path> ::= 'FULL_PATH'
<export_option_list> ::= <export_option> | <export_option_list> <export_option>
<export_option> ::=
REPLACE |
CATALOG ONLY |
NO DEPENDENCIES |
SCRAMBLE [BY <password>] |
THREADS <number_of_threads>
```

描述：

EXPORT 命令以指定的格式 BINARY 或者 CSV，导出表、视图、列视图、同义词、序列或者存储过程。临时表的数据和"no logging"表不能使用 EXPORT 导出表。

OBJECT_NAME
将导出对象的 SQL 名。欲导出所有集合下的所有对象，你要使用 ALL 关键字。如果你想导出指定集合下的对象，你应该使用集合名和星号，如"SYSTEM"."*"。

BINARY
表数据将以内部 BINARY 格式导出。使用这种方式导出数据比以 CSV 格式快几个数量级。只有列式表可以以二进制格式导出。行式表总是以 CSV 格式导出，即使指定了 BINARY 格式。

CSV
表数据将以 CSV 格式导出。导出的数据可以导入至其他数据库中。另外，导出的数据顺序可能被打乱。列式和行式表都可以以 CSV 格式导出。

FULL_PATH
将导出的服务器路径。

注意：当使用分布式系统，FULL_PATH 必须指向一个共享磁盘。由于安全性原因，路径可能不包含符号链接，也可能不指向数据库实例的文件夹内，除了'backup' 和 'work'子文件夹。有效路径（假设数据库实例位于/usr/sap/HDB/HDB00）的例子：

'/tmp'
'/usr/sap/HDB/HDB00/backup'
'/usr/sap/HDB/HDB00/work'

REPLACE
使用 REPLACE 选项，之前导出的数据将被删除，而保存最新导出的数据。如果未指定 REPLACE 选项，如果在指定目录下存在先前导出的数据，将抛出错误。

CATALOG ONLY
使用 CATALOG ONLY 选项，只导出数据库目录，不含有数据。

NO DEPENDENCIES
使用 NO DEPENDENCIES 选项，将不导出已导出对象的相关对象。

SCRAMBLE
以 CSV 格式导出时，使用 SCRAMBLE [BY '<password>']，可以扰乱敏感的客户数据。当未指定额外的数据库，将使用默认的扰乱密码。只能扰乱字符串数据。导入数据时，扰乱数据将以乱序方式导入，使最终用户无法读取数据，并且不可能回复原状。

THREADS
表示用于并行导出的线程数。

使用的线程数

给定 THREADS 数目指定并行导出的对象数，默认为 1。增加数字可能减少导出时间，但也会影响系统性能。

应当考虑如下：

- 对于单个表，THREADS 没有效果。
- 对于视图或者存储过程，应使用 2 个或更多的线程（最多取决于对象数）。
- 对于整个集合，考虑使用多余 10 个线程（最多取决于系统内核数）。
- 对于整个 BW / ERP 系统（ALL 关键字）的上千张表，数量大的线程是合理的（最多 256）。

系统和监控视图：

你可以使用系统视图 M_EXPORT_BINARY_STATUS 监控导出的进度。

你可以在如下语句中，使用会话 ID 从相应的视图中终止导出会话。

ALTER SYSTEM CANCEL [WORK IN] SESSION 'sessionId'

导出的详细结果存储在本地会话临时表#EXPORT_RESULT。

例子：

EXPORT "SCHEMA"."*" AS CSV INTO '/tmp' WITH REPLACE SCRAMBLE THREADS 10

IMPORT
语法：

IMPORT <object_name_list> [AS <import_format>] FROM <path> [WITH <import_option_list>]

语法元素：

WITH <import_option_list>:

可以使用 WITH 子句传入 IMPORT 选项。

<object_name_list> ::= <object_name>,... | ALL
<import_format> ::= BINARY | CSV
<path> ::= 'FULL_PATH'
<import_option_list> ::= <import_option> | <import_option_list> <import_option>
<import_option> ::=
REPLACE |
CATALOG ONLY |
NO DEPENDENCIES |
THREADS <number_of_threads>

描述：

IMPORT 命令导入表、视图、列视图、同义词、序列或者存储过程。临时表的数据和"no logging"表不能使用 IMPORT 导入。

OBJECT_NAME
将导入对象的 SQL 名。欲导入路径中的所有对象，你要使用 ALL 关键字。如果你想将对象导入至指定集合下，你应该使用集合名和星号，如"SYSTEM"."*"。

BINARY | CSV
导入过程可能忽略格式的定义，因为在导入过程中，将自动检测格式。将以导出的同样格式导入。

FULL_PATH
从该服务器路径导入。

注意：当使用分布式系统，FULL_PATH 必须指向一个共享磁盘。如果未指定 REPLACE 选项，在指定目录下存在相同名字的表，将抛出错误。

CATALOG ONLY
使用 CATALOG ONLY 选项，只导入数据库目录，不含有数据。

NO DEPENDENCIES
使用 NO DEPENDENCIES 选项，将不导入已导入对象的相关对象。

THREADS
表示用于并行导入的线程数。

使用的线程数

给定 THREADS 数目指定并行导入的对象数，默认为 1。增加数字可能减少导入时间，但也会影响系统性能。

应当考虑如下：

- 对于单个表，THREADS 没有效果。
- 对于视图或者存储过程，应使用 2 个或更多的线程（最多取决于对象数）。
- 对于整个集合，考虑使用多余 10 个线程（最多取决于系统内核数）。
- 对于整个 BW / ERP 系统（ALL 关键字）的上千张表，数量大的线程是合理的（最多 256）。

系统和监控视图：

你可以使用系统视图 M_IMPORT_BINARY_STATUS 监控导入的进度。

你可以在如下语句中，使用会话 ID 从相应的视图中终止导入会话。

ALTER SYSTEM CANCEL [WORK IN] SESSION 'sessionId'

导入的详细结果存储在本地会话临时表#IMPORT_RESULT。

IMPORT FROM
语法：

IMPORT FROM [<file_type>] <file_path> [INTO <table_name>] [WITH <import_from_option_list>]

语法元素：

WITH <import_from_option_list>:

可以使用 WITH 子句传入 IMPORT FROM 选项。

<file_path> ::= '<character>...'
<table_name> ::= [<schema_name>.]<identifier>
<import_from_option_list> ::= <import_from_option> | <import_from_option_list> <imp
ort_from_option>
<import_from_option> :: =
THREADS <number_of_threads> |
BATCH <number_of_records_of_each_commit> |
TABLE LOCK |
NO TYPE CHECK |
SKIP FIRST <number_of_rows_to_skip> ROW |
COLUMN LIST IN FIRST ROW |
COLUMN LIST (<column_name_list>) |
RECORD DELIMITED BY '<string_for_record_delimiter>' |
FIELD DELIMITED BY '<string_for_field_delimiter>' |
OPTIONALLY ENCLOSED BY '<character_for_optional_enclosure>' |
DATE FORMAT '<string_for_date_format>' |
TIME FORMAT '<string_for_time_format>' |
TIMESTAMP FORMAT '<string_for_timestamp_format>' |

描述：

IMPORT FROM 语句将外部 csv 文件的数据导入至一个已有的表中。

THREADS：表示可以用于并行导出的线程数。默认值为 1，最大值为 256。

BATCH：表示每个提交中可以插入的记录数。

THREADS 和 BATCH 可以通过启用并行加载和一次提交多条记录，实现加载的高性能。一般而言，对于列式表， 10 个并行加载线程以及 10000 条记录的提交频率是比较好的设置。

TABLE LOCK：锁住表为了更快的导入数据至列式表。如果指定了 NO TYPE CHECK，记录将在插入时，不检查每个字段的类型。

SKIP FIRST <int> ROW：跳过插入前 n 条记录。

COLUMN LIST IN FIRST ROW：表示在 CSV 文件中第一行的列。

COLUMN LIST (<column_name_list>)：表示将要插入的字段列表。

RECORD DELIMITED BY '<string>'：表示 CSV 文件中的记录分隔符。

FIELD DELIMITED BY '<string>'：表示 CSV 文件中的字段分隔符。

OPTIONALLY ENCLOSED BY '<character>'：表示字段数据的可选关闭符。

DATE FORMAT '<string>'：表示字符的日期格式。如果 CSV 文件有日期类型，将为日期类型字段使用指定的格式。

TIME FORMAT '<string>'：表示字符的时间格式。如果 CSV 文件有时间类型，将为时间类型字段使用指定的格式。

TIMESTAMP FORMAT '<string>'：表示字符的时间戳格式。如果 CSV 文件有时间戳类型，将为日期类型字段使用指定的格式。

例子：

```
IMPORT FROM CSV FILE '/data/data.csv' INTO "MYSCHEMA"."MYTABLE" WITH RECORD DELIMITED BY
'\n' FIELD DELIMITED BY ','
```

SQL 语句的限制

下面的表格列出了每条记录允许的最大限制。

Database	
数据库大小限制	由存储大小限制 RS: 1TB
锁的数量	对于记录锁没有限制，对于表限制为 16384
会话数量	**8192**
Schemas	
一个数据集合下的表数量	**131072**
标识符长度	127 字符
别名长度	128 字符
表名长度	请参见上面的"标识符长度"
列名长度	请参见上面的"标识符长度"
字符常量长度	32767 字节
二进制数中十六进制字符数	**8192**
Tables and Views	
一张表中列的数量	**1000**
一张视图列的数量	**1000**
一张列式表中的分区数	**1000**

每张表中的行数。	由存储大小限制 RS: lTB/sizeof(row), CS: 2^31 * number of partitions
行的长度	由 RS 存储大小限制 size (1TB)
非分区表的大小	由 RS 存储大小限制 size (1TB)
Indexes and Constraints	
一张表的索引数量	**1023**
每张表中主键数量	**16**
索引中列的数量	**16**
UNIQUE 约束中的列的数量	**16**
主键、索引和 UNIQUE 约束数量的总和	16384
SQL	
SQL 语句的长度	2GB
SQL 嵌套视图的深度	128
SQL 解析树的深度	255
SQL 语句或视图中联接表的数量	255
ORDER BY,GROUP,SELECT 子句中列的数量	65535

谓词中元素的数量	65535
SELECT 子句中元素的数量	65535
SQLScript	
所有存储过程的大小	由 RS 存储大小限制 size (1TB)

SQL 错误代码

下表列出 SAP HANA 显示的错误代码和描述。

1	General warning	普通警告
2	General error	普通错误
3	Fatal error	致命错误
4	Cannot allocate enough memory	无法分配足够的内存
5	Initialization error	初始化错误
6	Invalid data	无效数据
7	Feature not supported	不支持该功能
8	Invalid argument	非法参数
9	Index out of bounds	下标越界
10	Invalid username or password	无效用户名或密码
11	Invalid state	无效状态
12	Cannot open file	无法打开文件
13	Cannot create/write file	无法创建/写文件
14	Cannot allocate enough disk space	无法分配足够的磁盘空间
15	Cannot find file	找不到文件
16	Statement retry	重试语句
17	Metadata schema version incompatible between database and executable file	数据库和可执行文件之间元数据集合版本不兼容
18	Service shutting down	服务正在关闭
19	Invalid license	无效证书
128	Transaction error	事务错误
129	Transaction rolled back byan internal	内部错误，事务回滚

	error	
130	Transaction rolled back by integrity constraint violation	违反完整性约束，事务回滚
131	Transaction rolled back by lock wait timeout	等待锁定超时，事务回滚。
132	Transaction rolled back due to unavailable resource	资源不可用，事务回滚。
133	Transaction rolled back by detected deadlock	检测到死锁，事务回滚。
134	Failure in accessing checkpoint file	获取检查点文件失败
135	Failure in accessing anchor file	获取锚文件失败
136	Failure in accessing log file	获取日志文件失败
137	Failure in accessing archive file	获取档案文件失败
138	Transaction serialization failure	事务序列化失败
139	Current operation cancelled by request and transaction rolled back	由于请求和事务回滚，取消当前操作
140	Invalid write-transaction identifier	无效的写事务标识符
141	Failure in accessing invisible log file	获取隐藏记录文件失败
142	Exceed max num of concurrent transactions	超过同步事务的最大值
143	Transaction serialization failure until timeout expires	超时，事务序列化失败
144	Transaction rollback, unique constraint violated	事务回滚，违反唯一约束。
145	Transaction distribution work failure	事务分布任务失败
146	Resource busyand acquire with NOWAIT specified	资源忙碌，需指定 NOWAIT 获取
147	Inconsistency between data and log	数据与记录不一致

148	Transaction start is blocked until Master_Restart finishes	Master_Restart 未完成，事务启动仍被锁定。
149	Distributed transaction commit failure	已分配的事务提交失败
150	Statement cancelled due to old snapshot	语句被旧快照取消
256	SQL processing error	SQL 处理错误
257	SQL syntax error	SQL 语法错误
258	Insufficient privilege	权限不足
259	Invalid table name	无效的表名
260	Invalid column name	无效列名
261	Invalid index name	无效索引名
262	Invalid query name	无效查询名称
263	Invalid alias name	无效别名
264	Invalid datatype	无效数据类型
265	Expression missing	缺少表达式
266	Inconsistent datatype	数据类型不一致
267	Specified length too long for its datatype	超出指定类型的长度
268	Column ambiguously defined	模糊的列定义
269	Too many values	数值太多
270	Not enough values	数值不足
271	Duplicate alias	重复别名
272	Duplicate column name	重复列名
273	Not a single character string	没有一个单一的字符串
274	Inserted value too large for column	插入的数值对于列来说过大

275	Aggregate function not allowed	聚集功能不被允许
276	Missing aggregation or grouping	缺少聚集或分组
277	Not a GROUP BY expression	不是 GROUP BY 表达式
278	Nested group function without GROUP BY	嵌套分组函数没有使用 GROUP BY
279	Group function is nested	分组函数已嵌套。
280	ORDER BY item must be the number of a SELECT-list	GROUP BY 项必须是 SELECT 列表数
281	Outer join not allowed in operand of OR or IN	在 OR 和 IN 的操作数中，不允许有外联结
282	Two tables cannot be outer-joined to each other	两个表格之间不能互相外联结
283	A table may be outer joined to at most one other table	一张表最多只能被外联接到一张表
284	Join field does not match	联结字段不匹配
285	Invalid join condition	无效的联结条件
286	Identifier is too long	标识符名字过长
287	Cannot insert NULL or update to NULL	无法插入或更新 NULL
288	Cannot use duplicate table name	无法使用重复表名
289	Cannot use duplicate index name	无法使用重复索引名
290	Cannot use duplicate query name	无法使用重复查询名称
291	Argument identifier must be positive	参数标识符必须是正值
292	wrong number of arguments	参数数目错误
293	Argument type mismatch	参数类别不匹配
294	Cannot have more than one primary key	不能存在多于一个主键
295	Too long multi key length	过长的多键长度

296	Replicated table must have a primary key	复制的表只能有一个主键
297	Cannot update primary key field in replicated table	无法在复制表中更新主键
298	Cannot store DDL	无法储存 DDL
299	Cannot drop index used for enforcement of unique/primary key	不能删除用于增强唯一键/主键的索引
300	Argument index is out of range	参数索引不在范围内
301	Unique constraint violated	违反唯一约束
302	Invalid CHAR orVARCHAR value	无效 CHAR 或 VARCHAR 值
303	Invalid DATE, TIME orTlMESTAMP value	无效 DATE, TIME 或 IMESTAMP 值
304	Division by zero undefined	未定义除数为 0
305	Single-row query returns more than one row	单行索引返回至多行
306	Invalid cursor	无效游标
307	Numeric value out of range	数目值超出范围
308	Column name alreadyexists	列名已存在
309	Correlated subquery cannot have TOP or ORDER BY	相互关联的子查询不能使用 TOP 或 ORDER BY
310	SQL error in procedure	存储过程中的 SQL 错误
311	Cannot drop all columns in a table	无法删除表格中所有列
312	Sequence is exhausted	到达序列末尾
313	Invalid sequence	无效序列
314	Numeric overflow	数值溢出
315	Invalid synonym	无效同义词

316	wrong number of arguments in function invocation	函数调用参数的数目错误
317	P_QUERYPLANS not exists nor valid format	P_QUERYPLANS 不存在或格式无效
318	Decimal precision specifier is out of range	十进制精确性指定符超出范围
319	Decimal scale specifier is out of range	十进制小数位数超出范围
320	Cannot create index on expression with datatype LOB	在 LOB 数据类型的表达下无法创建索引
321	Invalid view name	无效的视图名称
322	Cannot use duplicate view name	无法使用重复的视图名称
323	Duplicate replication ID	重复更新 ID
324	Cannot use duplicate sequence name	无法使用重复的序列名
325	Invalid escape sequence	无效的转义序列
326	CURRVAL of given sequence is not yet defined in this session	未定义本次会话中 CURRVAL 的给定序列
327	Cannot explain plan of given statement	无法解释给定语句的计划
328	Invalid name of function or procedure	功能或过程的名称无效
329	Cannot use duplicate name of function or procedure	无法使用名字重复的函数或存储过程
330	Cannot use duplicate synonym name	无法使用重复的同义名
331	User name already exists	用户名已存在
332	Invalid user name	用户名无效
333	Column not allowed	列不被允许
334	Invalid user privilege	用户特权无效
335	Field alias name alreadyexists	字段别名已存在
336	Invalid default value	初始值无效

337	INTO clause not allowed for this SELECT statement	该 SELECT 语句中不可以出现 INTO 分句
338	Zero-length collumns are not allowed	不可以出现零长度的列
339	Invalid number	数目无效
340	Not all variables bound	所有的变量都没有界限
341	Numeric underflow	数目下溢
342	Collation conflict	校验名矛盾
343	Invalid collate name	无效的校验名
344	Parse error in data loader	数据加载中出现解析错误
345	Not a replication table	表格不可被更新
346	Invalid replication ID	无效更新 ID
347	Invalid option in monitor	监控器的选项无效
348	Invalid datetime format	无效的时间格式
349	Cannot CREATE UNIQUE INDEX	无法创建唯一索引
350	Cannot drop columns in the primary-key column list	无法删除主键中的列
351	Column is referenced in a multi-column constraint	列在多列约束中被引用
352	Cannot create unique indexon CDXtable	在 CDX 表中无法创建唯一索引
353	Update log group name alreadyexists	更新日志名称已存在
354	Invalid update log group name	无效的更新日志组名称
355	The base table of the update log table must have a primary key	更新日志的基表必须有一个主键
356	Exceed maximum number of update log group	更新日志组数值超过最大值
357	The base table already has a update	基表已有一张更新日志表

	log table	
358	Update log table can not have a update log table	更新日志表中不能包含更新日志表
359	Concatenated string is too long	连接字符串过长
360	View WITH CHECK OPTION where-cla use violation	违反 WITH CHECK OPTION 视图 where 子句
361	Data manipulation operation not legal on this view	对该视图的数据操作运算不合法
362	Invalid schema name	无效的集合名
363	Number of index columns exceeds its maximum	索引列数目超过最大值
364	Invalid partial key size	部分键大小无效
365	No matching unique or primary key for this column list	列的列表中没有匹配的唯一键或主键
366	Referenced table does not have a primary key	被引用的表中没有主键
367	Number of referencing columns must match referenced columns	被引用的列必须与引用列数目相等
368	Unique constraint not allowed on temporary table	在临时表上不允许唯一约束
369	Exceed maximum view depth limit	超过了最大视图宽度限制
370	Cannot perform DIRECT INSERT operation on table with unique indexes	在表中无法直接插入唯一索引
371	Invalid XML document	无效 XML 文件
372	Invalid XPATH	无效的 XPATH
373	Invalid XML duration value	无效 XML 持续值
374	Invalid XML function usage	无效的 XML 函数应用
375	Invalid XML index operation	无效的 XML 索引操作

376	Python stored procedure error	Python 存储过程错误
377	JIT operation error	JIT 操作错误
378	Invalid column view	无效的列视图
379	Table schema mismatch	表与模式不匹配
380	Fail to change run level	改变运行级别失败
381	Fail to restart	重启失败
382	Fail to collect all version garbage	收集所有无用数据版本失败
383	Invalid identifier	无效标识符
384	Constant string is too long	字符串常数过长
385	Could not restore session	无法恢复会话
386	Cannot use duplicate schema name	无法使用重复集合名
387	Table ambiguously defined	表格定义不清晰
388	Role already exists	角色已存在
389	Invalid role name	无效的角色名
390	Invalid user type	无效用户类别
391	Invalidated view	未验证的视图
392	Can't assign cyclic role	无法分配循环角色
393	Roles must not receive a privilege with grant option	不能使用 grant 选项授权角色
394	Error revoking role	错误的撤销角色
395	Invalid user-defined type name	无效的自定义类别名称
396	Cannot use duplicate user-defined type name	无法使用重复的自定义类别名称
397	Invalid object name	无效的对象名称
398	Cannot have more than one order by	只能存在一个 order by

399	Role tree too deep	对象树过深
400	Primary key not allowed on insert-only table	只允许插入的表中不允许主键
401	Unique constraint not allowed on insert-only table	只允许插入的表格中不允许唯一键
402	The user was already dropped before queryexecution	执行查询前，该用户已被删除
403	Internal error	内部错误
404	Invalid (non-existent) structured privilege name	无效（或不存在）结构化权限名
405	Cannot use duplicate structured privilege name	无法使用重复的结构化权限名
406	INSERT, UPDATE and UPSERT are disallowed on the generated field	生成的字段不允许 INSERT,UPDATE 以及 UPSERT
407	Invalid date format	无效的日期格式
408	Password or parameter required for user	用户必需的密码或参数
409	Multiple values for a parameter not supported	参数不支持有多个值
410	Invalid privilege namespace	无效的权限命名空间
411	Invalid table type	无效的表类型
412	Invalid password layout	无效的密码样式
413	Last n passwords can not be reused	上次用过的密码无法再次使用
414	User is forced to change password	用户被强制重设密码
415	User is deactivated	用户未激活
416	User is locked	用户被锁定
417	Can't drop without CASCADE specification	无法不使用 CASCADE 选项删除

418	Invalid view query for creation	视图查询无效创建
419	Can't drop with RESTRICT specification.	无法不使用 RESTRICT 选项删除
420	Password change currently not allowed	当前不允许修改密码
421	Cannot create full text index	无法创建全文字索引
422	Privileges must be either all SQL or all from one namespace	权限必须是 SQL 中或者来自其他命名空间
423	Live Cache error	缓存错误
424	Invalid name of package	无效的组件名
425	Duplicate package name	重复组件名
426	Number of columns mismatch	列的数目不匹配
427	Cannot reserve index ID any more	无法保存索引 ID
428	Invalid query plan ID	无效查询计划 ID
429	Integrity check failed	完整性检查失败
430	Invalidated procedure	无效的存储过程
431	User's password will expire within few days	用户的密码在几天内将会失效
432	This syntax has been deprecated and will be removed in next release	该语法已废弃，将在下次发布中移除
433	Null value found	发现 NULL
434	Invalid object ID	无效的对象 ID
435	Invalid expression	无效表达式
436	Could not set system license	无法设定系统证书
437	Only commands for license handling are allowed in current state	当前状态下只允许对证书处理的命令
438	Invalid user parameter value	无效用户参数值
439	Composite error	合成错误

440	Table type conversion error	表类型转换错误
441	This feature has been deprecated and will be removed in next release	该功能已废弃，将在下次发布中移除
442	Number of columns exceeds its maximum	列的数目超过最大值
443	Invalid calculation scenario name	无效的计算场景名称
444	Package manager error	组件管理器错误
512	Replication error	复制错误
513	Cannot execute DDL statement on replication table while replicating	无法在更新时对更新表执行 DDL 语句
514	Failure in accessing anchor file	获取锚文件失败
515	Failure in accessing log file	获取日志文件失败
516	Replication table has not conflict report table	复制表中没有冲突报告
517	Conflict report table already enabled	冲突报告已能使用
518	Conflict report table already disabled	冲突报告不能使用
576	API error	API 错误
577	Cursor type of forward is not allowed	转发的游标类型不被允许
578	Invalid statement	无效语句
579	Exceed maximum batch size	成批数据大小超过最大值
580	Server rejected the connection (protocol version mismatch)	服务器无法连接（代理类型不匹配）
581	This function can be called only in the case of single statement	只有单一语句的情形下才能访问此功能
582	This query does not have result set	此项查询没有结果集合
583	Connection does not exist	连接不存在
584	No more lob data	无更多 LOBs 数据

585	Operation is not permitted	操作不允许
586	Invalid parameter is received from server	服务器接收到无效参数
587	Result set is currently invalid	当前结果集合无效
588	Next() is not called for this result set	结果集合未调用 Next()
589	Too many parameters are set	设置过多参数
590	Some paramters are missing	一些参数丢失
591	Internal error	内部错误
592	Not supported type conversion	不支持类型转换
593	Re mote-only function	只能远程操作的函数
594	No more result row in result set	结果集合中不能出现更多的结果行
595	Specified parameter is not output parameter	指定的参数未被输出
596	LOB streaming is not permitted in auto-commit mode	在自动提交模式中不允许 LOB 流
597	Session context error	会话上下文错误
598	Failed to execute the external statement	外部语句执行失败
599	Session layer is not initialized yet	会话层仍未初始化
600	Failed routed execution	执行路由失败
601	Too many session variables are set	设置过多会话变量
602	Cannot set readonly session variable	无法设定只读会话变量
603	Invalid LOB	无效 LOB
604	Remote temp table access failure	远程临时表获取失败
605	Invalid XAjoin request	无效 XA 联结请求
606	Exceed maximum LOB size	LOB 大小超过最大值

607	Failed to cleanup resources	清理资源失败
608	Exceed maximum number of prepared statements	预备语句超过最大数量
1024	Session error	会话错误
1025	Communication error	通信错误
1026	Cannot bind a communication port	无法连接通信端口
1027	Communication initialization error	通信初始化错误
1028	I/O control error	I/O 控制错误
1029	Connection failure	连接错误
1030	Send error	发送错误
1031	Receive error	接收错误
1032	Cannot create a thread	无法创建线程
1033	Error while parsing protocol	解析协议错误
1034	Exceed maximum number of sessions	超过会话最大数量
1035	Not supported version	不支持的版本
1036	Invalid session ID	无效会话 ID
1037	Unknown hostname	未知的主机名
1280	SqlScript error	Sqlscript 错误
1281	Wrong number or types of parameters in call	调用中参数数量或类型错误
1282	Output parameter not a variable	输出参数不是变量
1283	OUT and IN OUT parameters may not have default expressions	OUT 和 IN OUT 可能不包含默认表达式
1284	Duplicate parameters are not permitted	不允许重复的参数
1285	At most one declaration is permitted in	声明部分最多允许一条声明语句

	the declaration section	
1286	Cursor must be declared by SELECT statement	游标必须由 SELECT 语句声明
1287	Identifier must be declared	必须声明标识符
1288	Expression cannot be used as an assignmenttarget	表达式不能作为赋值目标
1289	Expression cannot be used as an INTO-target of SELECT/FETCH statement	表达式不能作为 SELECT/FETCH 语句的 INTO 目标
1290	Expression is inappropriate as the left hand side of an assignment statement	表达式不当位于赋值语句的左边
1291	Expression is of wrong type	表达式类型错误
1292	Illegal EXIT statement, it must appear inside a loop	非法 EXIT 语句，必须出现在循环中。
1293	Identifier name must be an exception name	标识符必须是一个例外名
1294	An INTO clause is expected in SELECT statement	SELECT 语句中需要 INTO 子句
1295	EXPLAIN PLAN and CALL statement are not allowed	不允许 EXPLAIN PLAN 和 CALL 语句
1296	Identifier is not a cursor	标识符不是一个游标
1297	Wrong number of values in the INTO list of a FETCH statement	FETCH 语句中错误的 INTO 列表值数量
1298	Unhandled user-defined exception	未处理的用户定义的异常
1299	No data found	未找到数据
1300	Fetch returns more than requested number of rows	返回多于请求数的行
1301	Numeric or value error	数字或值错误
1302	Parallelizable function cannot have OUT or IN OUT parameter	并行函数不能有 OUT 或 IN OUT 参数

1303	User-defined exception	用户定义的异常
1304	Cursor is already opened	游标已经打开
1305	Return type is invalid	无效的返回类型
1306	Return type mismatch	返回类型不匹配
1307	Unsupported datatype is used	使用了不支持的数据类型
1308	Illegal single assignment	非法赋值
1309	Invalid use of table variable	无效的表变量使用
1310	Scalar type is not allowed	不允许标量类型
1311	Out parameter is not specified	未指定输出参数
1312	At most one output parameter is allowed	只允许最多一个参数
1313	Output parameter should be a table ora table variable	输出参数需为表或者表变量
1314	Inappropriate variable name: do not allow "" for the name of variable or parameter	不当的变量名：变量名不允许""
1315	Return result set from select stmt exist when result view is defined	定义结果视图，select 语句返回的结果集已存在
1316	Some out table varis not assigned	某个输出表变量没有赋值
1317	Function name exceedes max. limit	函数名超过最大长度
1318	Built-in function not defined	未定义内置函数
1319	Parameter must be a table name	变量名必须是一个表名
1320	Parameter must be an attribute name without a table name upfront	参数必须是最前面没有表名的属性名
1321	Parameter must be an attribute name without an alias	参数必须是一个没有别名的属性名
1322	CE_CALC not a 11 owed	不允许 CE_CALC

1323	Parameter must be a vector of columns or aggregations	参数必须是一个列的矢量或是聚集
1324	Join attribute must be available in projection list	联接属性必须在投影列表可见
1325	Parameter must be a vector of SQL identifiers	参数必须是 SQL 标识符的矢量
1326	Duplicate attribute name	重复属性名
1327	Parameter has a non supported type	参数有一个不支持的属性
1328	Attribute not found in column table	列式表中未找到属性
1329	Duplicate column name	重复列名
1330	Syntax Error for calculated Attribute	计算属性语法错误
1331	Syntax Error in filter expression	过滤表达式语法错误
1332	Parameter must be a valid column table name	参数必须是有效的列式表名
1333	Join attributes not found in variable	未在变量中找到联接属性
1334	Input parameters do not have the same table type	输入参数表类型不相同
1335	Cyclic dependency found in a runtime procedure	存储过程运行时发现循环依赖
1336	Unexpected internal exception caught in a runtime procedure	存储过程运行时捕捉到意外的异常
1337	Variable depends on an unassigned variable	变量依赖于未赋值的变量
1338	CE_CON VERS ION: customizing table missing	CE_CONVERSION:自定义表丢失
1339	Too many parameters	过多参数
1340	The depth of the nested call is too deep	嵌套调用过深
1536	Swapx error	交换错误

1537	This table has no swap space	该表没有交换空间
1538	Swap already activated	交换已被激活
1539	Swap not yet activated	交换未激活
1540	Swap space is not created	未创建交换空间
1541	Failure in unpinning a swap page	取消拼接交换页失败
1542	Failure in swap file	交换文件失败
1543	Failure in accessing swap data file	访问交换数据文件失败
1544	Failure in accessing swap log file	访问交换日志文件失败
1545	Swap buffer overflow	交换缓存溢出
1546	Swap buffer reservation failure	保存交换缓存失败
1792	Shared memory error	共享内存错误
1793	Invalid key or invalid size	无效键值或者无效大小
1794	The segment already exists	段已存在
1795	Exceed the system-wide limit on shared memory	超过共享内存的系统宽度限制
1796	No segment exists for the given key, and 1 PC_CREAT was not specified	不存在指定键的段，并且未指定 IPC_CREAT
1797	The user does not have permission to access the shared memory segment	用户不允许访问共享内存段
1798	No memory could be allocated for segment overhead	没有内存可以分配给段开销
1799	Invalid shmid	无效的 shmid
1800	Allow read access for shmid	允许 shmid 的读访问
1801	Shmid points to a removed identifier	Shmid 指向一个已删除的标识符
1802	The effective user ID of the calling process is not the creator	调用进程的有效用户 ID 是不是创建者

1803	The GID or UID value is too large to be stored in the structure	GID 或 UID 值太大，无法保存在结构中
1804	The user does not have permission to access the shared memory segment	用户不允许访问共享内存段
1805	Invalid shmid	无效的 shmid
1806	No memory could be allocated forthe descriptor or for the page tables	没有内存可以分配给描述符或者页表
1807	Unknown shared memory error	未知的共享内存错误
2048	Column store error	列式存储错误
2049	Primary key is not specified for column table	未指定列式表的主键
2050	Not supported ddl type for column table	不支持的列表 DDL 类型
2051	Not supported data type for column table	不支持的列表数据类型
2052	Not supported dml type for column table	不支持的列表 DML 类型
2053	Invalid returned value from attribute engine	无效的属性引擎返回值
2304	Python DBAPI error	Python DABPI 错误
2305	Interface failure	接口失败
2306	Programming mistake	编程错误
2307	Invalid query parameter	无效的查询参数
2308	Not supported encoding for string	不支持的字符编码
2560	Distributed metadata error	分布式元数据错误
2561	DDL redirect error	DDL 重定向错误
2562	DDL notification error	DDL 通知错误
2563	DDL invalid container ID	DDL 无效的容器 ID

2564	DDL invalid index ID	DDL 无效的索引 ID
2565	Distributed environment error	分布式环境错误
2566	Network error	网络错误
2567	Metadata update not supported in slave	不支持在从中更新元数据
2568	Metadata update of master indexserver is failed	主 indexserver 更新元数据失败
2816	SqlScript Error	SqlScript 错误
2817	SqlScript Builtin Function	SqlScript 内置函数
2818 - 2889	SqlScri pt	2818 - 2889 sqlscript
3584	Distributed SQL error	分布式 SQL 错误
3585	Expression shipping failure	传送表达式失败
3586	Operator shipping failure	传送操作符失败
3587	Invalid protocol or indexserver (stati sties server) shutdown during distributed queryexecution	分布式查询执行期间，无效协议或索引服务器关闭
3588	Sequence shipping failure	传送序列失败
3589	Remote queryexectuion failure	远程查询执行失败
3840	general auditing error	审计错误
3841	Invalid privilege	无效权限
3842	Audit trail writer is blocked	审计跟踪写入器被阻塞
3843	Audit policy with current name already exists	已存在使用当前名字的审计策略
3844	Invalid combination of audit actions	无效的审计活动组合
3845	Invalid action status for auditing	无效的审计活动状态
3846	Invalid auditing level	无效的审计级别

3847	Invalid policy name	无效策略名
4096	General error in the process of stored plan	存储计划处理中的错误
4097	Invalid operation in generating plans	无效的生成计划操作
4098	Invalid operation in execution the chosen plan	无效的执行选定计划操作
4099	Invalid operation in storing the pinned plan	无效的存储计划操作
4100	Invalid operation in loading the stored plan	无效的存储加载操作
4101	Invalid operation in deleting the chosen plan	无效的删除选定计划操作
4103	Failed to prepare for runtime reorganization	准备运行时重组失败
4104	Transaction blocked since runtime reorganization is in progress	运行时重组执行中，事务被阻塞
4105	ERR_REORG_TRANS_EXISTS_GENERAL. Cannot start reorganization due to the transactions in execution	ERR_REORG_TRANS_EXISTS_GENERAL.由于事务执行中，无法启动重组