YashanDB 开发手册

v23.2.2.0

2024年5月

深圳计算科学研究院

深圳崖山科技有限公司

SQL参考手册

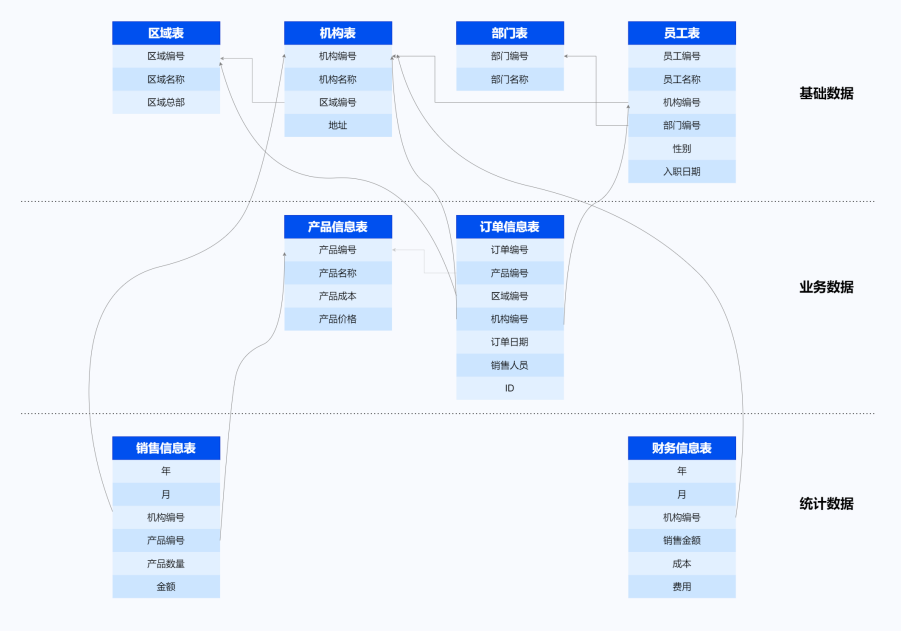
SQL（Structured Query Language）是开发者操作数据库的主要接口，本手册从数据类型、运算符、函数、SQL语句等方面全面介绍YashanDB对于SQL功能的实现。

手册中使用的示例说明

- 示例中的SQL、PL语句均在yasql工具中运行，并以yasql工具的输出结果作为样例进行展示。
- 示例中使用DBMS_OUTPUT.PUT_LINE语句向yasql工具端打印输出，为达到输出效果，需要保证在yasql工具里已运行set serveroutput on来打开控制输出的开关，详见工具手册yasql基本功能使用指导。
- 系统对浮点类型及NUMBER类型数据输出时的显示宽度默认为10，可通过SET NUMWIDTH调节显示宽度。
- 关于日期的显示格式：为尽量模拟实际业务，本手册示例库的日期格式被设置为'YYYY-MM-DD HH24:MI:SS'，如DATE_FORMAT配置参数不是按此格式设置，所看到的日期格式将与本手册不一致。
- 关于示例中使用的通用表：本手册使用sales作为示例用户（参考CREATE USER创建用户），同时在该用户下创建一套样例表（样例表来源于下面所列的一套极简化的业务模型，并依据不同架构/存储特性进行相应调整）。示例中使用的通用表即来自于这套样例表，当执行示例时，切换到sales用户下将可以查询到这些表，产品文档中附：样例表展示了这些表的创建脚本。

> **Note**：
> 为达到特定目的，示例可能需要创建其他表或者修改通用表的结构、数据，这些将不在此处体现，而是查看具体示例。

样例业务模型




```
--区域信息表：area、area1
--机构信息表：branches、branches1
--部门信息表：department
--员工信息表：employees
--产品信息表：product
--订单信息表：orders_info
--销售信息表：sales_info、sales_info_range、sales_info_list、sales_info_hash
--财务信息表：finance_info
```

样例表类型说明

YashanDB支持多种部署形态（单机/分布式/共享集群）和多种表类型（HEAP/TAC/LSC），并提供DEFAULT_TABLE_TYPE参数用于配置创建表对象时的默认表类型，同时还提供ORGANIZATION语法用于创建一个指定表类型的表对象（查看CREATE TABLE了解该语法详细说明）。

本手册示例中所使用的样例表的类型依据系统部署形态，和DEFAULT_TABLE_TYPE参数的值确定。当示例或示例所在前提中未注明语句适用于某一种部署形态或某一种表类型时，表示该示例语句可以在所有部署形态中以任一种表类型运行；否则，该示例语句应在注明的部署形态环境中，并以注明的表类型运行，不合适的部署形态环境或表类型可能导致示例语句运行异常，或产生与预期不一致的输出结果。

基本SQL元素

本章节对SQL语法中所使用的最基本和简单的要素进行说明和定义。

字面量

字面量的意思是不变的值，它以字符串的形式直接出现在SQL和PL语句中，通过声明时的格式对其进行类型区分，例如"用于识别字符型的字面量，DATE用于识别日期时间型的字面量。

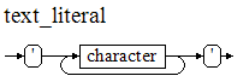
字面量几乎可以用在所有场景中，例如作为值、参数、格式、标识或输出显示。

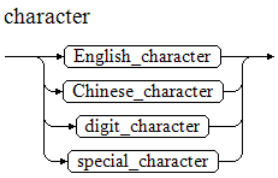
字面量与变量、常量在概念上的区别：

- 字面量是以字符串形式出现的固定值，它没有存储容器，只是一种书面上的记法。
- 变量是用来存储数据的一个容器，且容器可以被多次赋值。
- 常量是变量的一种，但它只能被初始赋值，且不能再次被赋值。

字符串字面量（Character Literal）

字符串字面量是使用单引号`''`包围的英文字母、中文汉字、数字字符和特殊字符的组合，其声明方式如下所示：





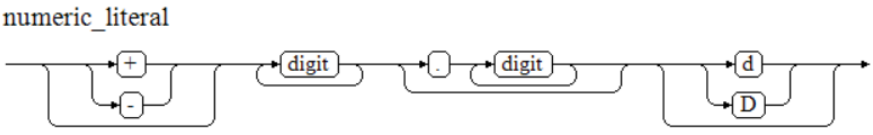
YashanDB将字符串字面量解析成与其等长的CHAR类型数据，例如'China'就是'CHAR(5)'。

示例

```
SELECT 'SHENZHEN', '3.14567', '!@#$',
TYPEOF('SHENZHEN') type1,
TYPEOF('3.14567') type2,
TYPEOF('!@#$') type3
FROM DUAL;
'SHENZHEN' '3.14567' '!@#$' TYPE1 TYPE2 TYPE3
---------- --------- ------ ----- ----- -----
SHENZHEN   3.14567   !@#$   char  char  char
```

数值字面量（Numeric Literal）

数值字面量是以数字0 ~ 9和小数点组合形成的数值，数值前可以用正号（+）或负号（-）将其变成正数负数（默认为正数）。其声明方式如下图所示：



数值字面量包含4个数据类型：INT、BIGINT、NUMBER、DOUBLE。YashanDB按如下规则解析这些类型：

- 当数值后出现'd'或'D'时，将解析为DOUBLE类型。
- 小数解析成NUMBER类型。
- 整数首先尝试将它解析为INT类型（判断是否在INT值域[-2^{31}， 2^{31} - 1]），失败则尝试解析为BIGINT类型（判断是否在BIGINT值域[-2^{63}，2^{63} - 1]），失败则解析为NUMBER类型。
- 超过NUMBER范围解析为DOUBLE类型。

例如，11d被解析为DOUBLE类型，2147483647被解析为INT类型，2147483648被解析为BIGINT类型，1.1被解析为NUMBER类型。

示例

```
SELECT 1, TYPEOF(1) FROM DUAL;
           1 TYPEOF(1)
------------ ---------
           1 integer

SELECT 2147483647, TYPEOF(2147483647) FROM DUAL;
  2147483647 TYPEOF(2147483647)
------------ -------------------
  2147483647 integer

SELECT 2147483648, TYPEOF(2147483648) FROM DUAL;
           2147483648 TYPEOF(2147483648)
--------------------- -------------------
           2147483648 bigint

SELECT 1.1, TYPEOF(1.1) FROM DUAL;
        1.1 TYPEOF(1.1)
----------- -----------
        1.1 number

SELECT 11d, TYPEOF(11d) FROM DUAL;

        11  TYPEOF(11D)
----------- -----------
   1.1E+001 double
```

科学计数法数值字面量

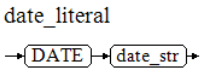
YashanDB支持将类似1.24E3格式的字面量解析为科学计数法输入的NUMBER类型数据。

示例

```
SELECT 1.24e3 FROM dual;
      1.24E3
------------
        1240
```

日期字面量（Date Literal）

日期字面量是以DATE关键字开始，直至单引号 `''` 包围的字符串右引号结束的值，采用'yyyy-mm-dd'进行格式匹配，例如 `DATE '2020-01-01'` 。其声明方式如下图所示，其中date_str表示符合DATE类型标准格式的字符串。

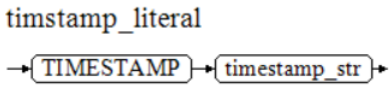


示例

```
SELECT DATE '2020-01-01', TYPEOF(DATE '2020-01-01') TYPE FROM DUAL;
DATE'2020-01-01'                TYPE
------------------------------- -----
2020-01-01 00:00:00             date
```

时间戳字面量（Timestamp Literal）

日期时间戳字面量是以TIMESTAMP开始，直至单引号 `''` 包围的字符串右引号结束的值，采用'yyyy-mm-dd hh24:mi:ss.ff'进行格式匹配，时分秒可缺省，例如 `TIMESTAMP '2020-01-01 13:08:28'` 。其声明方式如下图所示，其中timestamp_str表示符合TIMESTAMP类型标准格式的字符串。

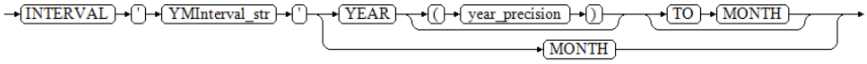


示例

```
SELECT TIMESTAMP '2020-01-01 13:08:28', TYPEOF(TIMESTAMP '2020-01-01 13:08:28') TYPE FROM DUAL;
TIMESTAMP'2020-01-01                                          TYPE
------------------------------------------------------------ -------------
2020-01-01 13:08:28.000000                                    timestamp
```

年到月字面量（Interval Year To Month Literal）

年到月字面量的声明方式如下图所示，其中YMInterval_str表示符合INTERVAL YEAR TO MONTH类型标准格式的字符串，year_precision表示年的精度（默认值为2）。



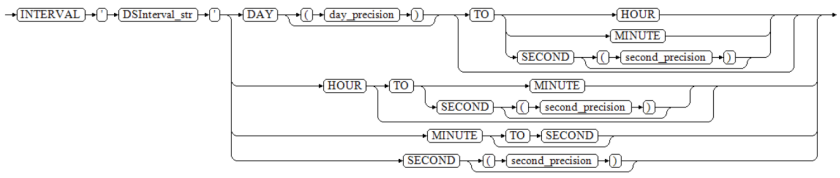
声明时可以指定年、月中的一个或多个域；当单独指定的月间隔超过11时，YashanDB将自动将其转换为年间隔。

示例

```
-- Example 1: 仅指定年间隔
SELECT INTERVAL '120' YEAR(3) YMINTERVAL,
TYPEOF(INTERVAL '120' YEAR(3)) TYPE
FROM DUAL;
YMINTERVAL      TYPE
--------------- -------------------------
+120-00         interval year to month

-- Example 2: 仅指定月间隔，超过11月将转换为年
SELECT INTERVAL '3' MONTH YMINTERVAL,
TYPEOF(INTERVAL '3' MONTH) TYPE
FROM DUAL;
YMINTERVAL      TYPE
--------------- -------------------------
+00-03          interval year to month

SELECT INTERVAL '14' MONTH YMINTERVAL,
TYPEOF(INTERVAL '14' MONTH) TYPE
FROM DUAL;
YMINTERVAL      TYPE
--------------- -------------------------
+01-02          interval year to month

-- Example 3: 指定年和月间隔，在此情况下月超过11时无法转换为年
SELECT INTERVAL '100-11' YEAR(3) TO MONTH YMINTERVAL,
TYPEOF(INTERVAL '100-11' YEAR(3) TO MONTH) TYPE
FROM DUAL;
YMINTERVAL      TYPE
--------------- -------------------------
+100-11         interval year to month

SELECT INTERVAL '100-14' YEAR(3) TO MONTH YMINTERVAL FROM DUAL;
YAS-00008 type convert error : not a valid month
```

天到秒字面量（Interval Day To Second Literal）

天到秒字面量的声明方式如下图所示，其中DSInterval_str表示符合INTERVAL DAY TO SECOND类型标准格式的字符串，day_precision表示天的精度（默认值为2），second_precision表示秒的精度（默认值为6）。



声明时可以指定天、时、分、秒中的一个或多个域；当时超过23、分和秒超过59，且没有指定上级域时，YashanDB会自动将其进位。

示例

```
-- Example 1: 仅指定天
SELECT INTERVAL '50' DAY DSINTERVAL,
TYPEOF(INTERVAL '50' DAY) TYPE
FROM DUAL;
DSINTERVAL                     TYPE
------------------------------ ------------------------
+50 00:00:00.000000            interval day to second

-- Example 2: 仅指定时，超过23时则进位为天
SELECT INTERVAL '160' HOUR INTERVAL1, INTERVAL '13' HOUR INTERVAL2 FROM DUAL;
INTERVAL1                      INTERVAL2
------------------------------ ------------------------------
+06 16:00:00.000000            +00 13:00:00.000000

-- Example 3: 仅指定分，超过59分则进位为时
SELECT INTERVAL '63' MINUTE INTERVAL1,
INTERVAL '32' MINUTE INTERVAL2
FROM DUAL;
INTERVAL1                      INTERVAL2
------------------------------ ------------------------------
+00 01:03:00.000000            +00 00:32:00.000000

-- Example 4: 仅指定秒，超过59秒则进位为分
SELECT INTERVAL '63' SECOND INTERVAL1,
INTERVAL '45' SECOND INTERVAL2,
INTERVAL '59.999999' SECOND(6)
FROM DUAL;
INTERVAL1                      INTERVAL2                      INTERVAL'59.999999'S
------------------------------ ------------------------------ ------------------------------
+00 00:01:03.000000            +00 00:00:45.000000            +00 00:00:59.999999

-- Example 5：指定多个域
SELECT INTERVAL '1000 23:30' DAY(4) TO MINUTE INTERVAL1,
INTERVAL '12:15:21.647' HOUR TO SECOND(3) INTERVAL2
FROM DUAL;
INTERVAL1                      INTERVAL2
------------------------------ ------------------------------
+1000 23:30:00.000000          +00 12:15:21.647000

-- Example 6：当存在上级域时，下级域无法进位
SELECT INTERVAL '12:63' HOUR TO MINUTE FROM DUAL;
YAS-00008 type convert error : not a valid minute
```

二进制字面量（BIT Literal）

二进制字面量是以字符b开始，直至单引号 `''` 包围的字符串右引号结束的值。其声明方式如下，其中bit_str表示符合二进制类型标准格式的字符串。

bit_literal := b'bit_str'

示例

```
SELECT b'010011101',
TYPEOF(b'010011101') TYPE
FROM DUAL;
B'010011101'                                                TYPE
```

```
------------------------------------------------------------ -----
10011101                                                     bit
```

标识符

YashanDB中包括两类标识符：

- 作为语法关键字，用于定界SQL语法树的位置；作为系统保留语法关键字将不能作为数据库对象的名称。
- 作为数据库对象的名称，用于表名、列名、别名等用途，但需满足下述命名规范。

命名规范

标识符作为名称时存在如下规则和约束：（不使用双引号时）

- 支持所有的大小写字母和数字，且大小写不敏感。
- 不能出现特殊字符'\0' ',' ' ' '+' '-' '*' '/' '|' '(' '%' ':' '?' '.' '\t' '\r' '\n' '=' '\\' '!' '>' '<' ';' '&' '^' '"' '~' '`' '{' '}' '[' ']'等。
- 必须以字母开头。
- 不能使用系统保留语法关键字作为名称（包括大小写）。

当在标识符前后使用双引号时：

- 不受上述约束限制，即名称内可以出现特殊字符，可以非字母开头，并可以使用系统保留语法关键字作为名称。
- 对大小写的处理需满足下表规则：（同时需满足双引号的通用规则）

标识符	双引号	定义名称时	引用名称时
满足约束	使用双引号	可选	全大写定义名称： * 加双引号引用时，严格区分大小写 * 不加双引号引用时，不区分大小写 非全大写定义名称： * 必须加双引号引用 * 严格区分大小写
满足约束	不使用双引号	可选	* 不区分大小写
违反约束	使用双引号	必须	* 必须加双引号引用 * 严格区分大小写
违反约束	不使用双引号	报错	报错

使用示例

```
-- 符合约束的标识符作为名称，定义时使用双引号
CREATE TABLE "CREATE01" (c1 INT);             -- 按全大写定义名称
Succeed.
INSERT INTO Create01 VALUES(1);               -- 不加双引号引用时，不区分大小写
1 row affected.
INSERT INTO "Create01" VALUES(1);             -- 加双引号引用时，严格区分大小写
[1:12]YAS-02012 table or view does not exist

DROP TABLE Create01;
Succeed.
CREATE TABLE "CREATe01" (c1 INT);             -- 按非全大写定义名称时，引用名称时必须使用双引号，且严格区分大小写
Succeed.
INSERT INTO CREATE01 VALUES(1);
[1:12]YAS-02012 table or view does not exist
INSERT INTO "CREATE01" VALUES(1);
[1:12]YAS-02012 table or view does not exist
INSERT INTO "CREATe01" VALUES(1);
1 row affected.

-- 违反约束的标识符作为名称，定义和引用都必须使用双引号
CREATE TABLE "CREATE" (c1 INT);         -- 按全大写定义名称时，引用时严格区分大小写
Succeed.
```

```
INSERT INTO CREATE VALUES(1);
[1:12]YAS-04202 missing or invalid table name
INSERT INTO "CREATE" VALUES(1);
1 row affected.
INSERT INTO "Create" VALUES(1);
[1:12]YAS-02012 table or view does not exist

DROP TABLE "CREATE";
Succeed.
CREATE TABLE "Create" (c1 INT);         -- 按非全大写定义名称时，引用时严格区分大小写
Succeed.
INSERT INTO Create VALUES(1);
[1:12]YAS-04202 missing or invalid table name
INSERT INTO "CREATE" VALUES(1);
[1:12]YAS-02012 table or view does not exist
INSERT INTO "Create" VALUES(1);
1 row affected.
```

双引号

双引号是对标识符、密码定义和使用的扩展。使用双引号时，标识符、密码定义将支持特殊字符、数字，并且区分字母大小写。

双引号的使用位置（是否可出现在SELECT，FROM和WHERE后）和使用场景规则，以及双引号内字符串的长度限制，由双引号限定的标识符和密码本身决定，与是否加双引号无关。

表名、列名、视图名、序列名、同义词、函数名、存储过程名、触发器名以及JOB名等对象名称标识符支持使用双引号。

表别名、列别名和视图别名标识符支持使用双引号。

用户密码支持除双引号外的任意字符，不支持转义双引号（例如"hhh^^\"hhh"）。

YashanDB对双引号的实现规则如下：

- 双引号内字符串长度限制不允许超过64字节，否则报错。
- 双引号成对使用。从读取左引号开始会一直检索直到读取到右引号，且保留双引号包围的字符串不做处理；右引号缺失时报错。
- DDL语句会直接将双引号内的原始字符串（保留大小写）写入元数据中。
- DML语句会将按双引号内的原始字符串跟元数据字符串进行大小写敏感比较。

下表为标识符中是否使用了双引号的不同处理规则：

字符型	使用双引号	不使用双引号
大小写	将双引号内原始字符串与元数据字符串进行大小写敏感比较	转为大写后，与元数据字符串进行大小写敏感比较
特殊字符	将双引号内原始字符串与元数据字符串对比	报错
数字	作为名称标识符处理	作为数字处理

标识符作为名称时的大小写规则以及支持的特殊字符请查阅标识符。

对象

在YashanDB中，每个用户拥有一个同名Schema，对开发人员来说，所有对数据库的访问和操作均基于位于指定的Schema下各类对象，模式对象包括：

- 表
- 访问约束
- 索引
- 分区
- 视图
- 序列
- 同义词
- 自定义数据类型
- 自定义函数
- 存储过程
- 程序包
- 定时任务
- 触发器

更多详细请查阅模式对象。

伪列

伪列的行为类似于普通列，但其数据并不存储在表中。用户可以SELECT伪列，但不能对伪列进行INSERT、UPDATE、DELETE操作。

YashanDB存在如下五种伪列：

- ROWSCN
- ROWID
- ROWNUM
- SEQUENCE
- USER

ROWSCN伪列

ROWSCN伪列表示行最后一次被修改的SCN。YashanDB的ROWSCN为页面级别，一般情况下返回的ROWSCN是行所在页面最后一次修改并提交的BLOCK SCN，若使用特殊查询条件可能会返回该行最后一次修改并提交的SCN，例如使用ROWID为查询条件时。

若事务修改了表中某行数据并提交了修改，后续查询该行的ROWSCN时，返回值仅会大于或等于修改前的ROWSCN。

该伪列仅适用于HEAP表。

示例（HEAP表）

```
SELECT ROWSCN FROM area;
```

ROWID伪列

ROWID伪列表示行所在的地址。ROWID伪列的类型为ROWID类型，使用限制等更多说明请查阅ROWID UROWID。

该伪列仅适用于HEAP表。

示例（HEAP表）

```
SELECT ROWID FROM area;
```

ROWNUM伪列

ROWNUM伪列是系统顺序分配的查询返回的行编号，返回的第一行分配为1，第二行分配为2，依此类推。该伪列可用于限制查询返回的总行数。

分布式部署中不可使用该伪列。

ROWNUM在列存表中的使用限制如下：

- ROWNUM只能与常量进行比较。
- 条件语句中不能使用小于、小于等于、大于或大于等于运算符。
- 使用等值比较时，ROWNUM所比较的对象只能为常量1。
- 不允许ROWNUM在or的左右条件中使用。
- ROWNUM在过滤条件中只能使用一次，且只能单独使用，不允许参与运算或作为函数入参。
- ROWNUM不能出现在非比较谓词和不等于比较中，例如in。
- 在外连接的连接条件中不可使用ROWNUM。

ROWNUM伪列的常用场景：

WHERE条件中做行数限制

- WHERE ROWNUM < 常数，返回常数-1行的记录。
- WHERE ROWNUM = 常数，ROWNUM从1开始累加，因此ROWNUM= 1可以返回1条记录；当ROWNUM = 任何非1值时，条件恒为FALSE，无法返回任何结果。
- WHERE ROWNUM > 常数，常数小于1，则条件恒为TRUE；常数大于等于1，则条件恒为FALSE（即查询结果恒为空）。

示例（HEAP表）

```
SELECT * FROM area WHERE ROWNUM < 2;

AREA_NO AREA_NAME                                                       DHQ
------- ------------------------------------------------------------ --------------------
01      华东                                                          Shanghai
```

ROWNUM与ORDER BY一起使用

先从表中取出符合ROWNUM条件的行然后再进行排序，其结果取决于执行计划，执行计划不同可能会导致取得的行不同。

示例（单机、共享集群部署）

```
-- 创建表STUDENTS
CREATE TABLE STUDENTS(num INT,score INT);
INSERT INTO STUDENTS VALUES(1,99);
INSERT INTO STUDENTS VALUES(1,76);
INSERT INTO STUDENTS VALUES(1,84);
INSERT INTO STUDENTS VALUES(1,63);
INSERT INTO STUDENTS VALUES(1,88);
COMMIT;

-- 查询成绩前3的学生成绩，先取出3行再进行排序，结果不一定正确
SELECT score FROM STUDENTS WHERE ROWNUM < 4 ORDER BY score DESC;

-- 查询成绩前3的学生成绩，可以使用order by + limit
SELECT score FROM STUDENTS ORDER BY score DESC LIMIT 3;

-- 查询成绩前3的学生成绩，可以使用子查询 + ROWNUM
SELECT score FROM (SELECT score FROM STUDENTS ORDER BY score DESC) WHERE ROWNUM < 4;
```

ROWNUM与JOIN一起使用

在JOIN语句中，ROWNUM与在WHERE中使用相同的规则判断。

使用ROWNUM给表中每一行附上唯一的值

示例（单机、共享集群部署）

```
UPDATE STUDENTS SET num = ROWNUM;
```

SEQUENCE伪列

SEQUENCE伪列是创建序列对象时生成的唯一序列号，可分为如下两种类型：

- CURRVAL：返回当前序列号值，即当前session上一次获取到的NEXTVAL值。
- NEXTVAL：增加序列号的值并返回序列的NEXTVAL。

分布式部署中不可使用该伪列。

获取序列号：

```
sequence.CURRVAL
sequence.NEXTVAL
```

获取指定schema的序列号：

```
schema.sequence.CURRVAL
schema.sequence.NEXTVAL
```

可以在以下场景使用CURRVAL伪列与NEXTVAL伪列：

- SELECT语句中被查询的列中不包含子查询的场景
- INSERT语句中的SELECT从句

- INSERT语句中的VALUES从句
- UPDATE语句中的SET从句

禁止使用的场景：

- DELETE、SELECT、UPDATE语句中的子查询语句
- 带有DISTINCT算子的SELECT语句
- 带有GROUP BY或ORDER BY子句的SELECT语句
- 多个SELECT语句通过UNION集合
- SELECT语句中的WHERE子句

对于某一个序列号，在使用CURRVAL伪列前，同一会话中必须先通过NEXTVAL伪列完成CURRVAL值的初始化。

示例（单机、共享集群部署）

用以下语法获取序列号：

```
CREATE SEQUENCE seq_yashan_pseudo;

SELECT seq_yashan_pseudo.NEXTVAL FROM dual;
  SEQ_YASHAN1.NEXTVAL
----------------------
                    1

SELECT seq_yashan_pseudo.CURRVAL FROM dual;
  SEQ_YASHAN1.CURRVAL
----------------------
                    1
```

USER伪列

USER伪列返回当前登录用户的用户名，返回值类型为VARCHAR(64)。

分布式部署中不可使用该伪列。

示例（单机、共享集群部署）

```
SELECT USER FROM DUAL;
```

NULL

NULL表示为空值，YashanDB中将空字符串同样视作NULL处理，NULL并不作为确定的值使用，而表示该值处于未知状态或不具有意义。

仅支持在未定义为NOT NULL或PRIMARY KEY约束限制的列中使用NULL。

SQL内置函数中的NULL

YashanDB对SQL内置函数中给定参数为NULL的处理结果请查阅内置函数。

比较条件中的NULL

如需判断是否为NULL，请使用比较运算符`IS NULL`进行判断，当操作数为NULL时，结果返回TRUE，否则返回FALSE。

NULL表示空值，即缺少数据，因此无法与其他值（包括NULL）进行比较，当使用`=` `!=`等运算符进行比较时，结果会返回FALSE。

运算中的NULL

YashanDB中算数运算及位运算中所有NULL参与运算的结果均为NULL，部分比较运算、逻辑运算及连接运算的返回结果根据给定的操作数返回不同结果，详见下表：

运算	运算符	NULL参与运算
算数运算	+	结果为NULL
	-	结果为NULL
	*	结果为NULL
	/	结果为NULL
	%	结果为NULL
比较运算	=	结果为FALSE
	!= 或 <>	结果为FALSE
	>	结果为FALSE
	>=	结果为FALSE
	<	结果为FALSE
	<=	结果为FALSE
	[NOT] IN	IN： * 左边数据为NULL：结果为FALSE * 右边集合包含NULL：如果集合中有非NULL值与左边数据相等，则结果为TRUE，否则为FALSE NOT IN： * 左边数据为NULL：结果为FALSE * 右边集合包含NULL：如果集合中含有NULL值，则返回FALSE；如果集合中不含NULL值，且所有数据与左边数据不相等，则结果为TRUE，否则为FALSE
	[NOT] LIKE	结果为FALSE
	[NOT] BETWEEN AND	结果为FALSE

运算	运算符	NULL参与运算
	IS [NOT] NULL	IS NULL： * 操作数为NULL：结果为TRUE * 操作数不为NULL：结果为FALSE IS NOT NULL： * 操作数为NULL：结果为FALSE * 操作数不为NULL：结果为TRUE
逻辑运算	AND	结果为NULL
	OR	* TRUE OR NULL：结果为TRUE * FALSE OR NULL：结果为NULL * NULL OR NULL：结果为NULL
	NOT	结果为NULL
连接运算	||	任何数据||NULL：结果为任何数据
位运算	&	结果为NULL
	|	结果为NULL
	^	结果为NULL

数据类型

数据库里所有的数据都归属于某一种数据类型（Data Type），数据类型标识了数据的存储相关属性和对该种数据的操作限制。

当用户输入一个数据时，系统将按照一定的规则识别其所属数据类型，具体可参考字面量描述。

当创建表对象时，对列字段的声明必须指定一种数据类型，依据这种数据类型，该列上的所有数据都应满足其存储属性要求，如长度、精度等，才能被成功插入到表中。YashanDB同时提供了数据类型的隐式转换功能，当用户输入数据与字段上定义的数据类型不一致时，系统将先进行数据类型转换，例如将字符型'100'转换为数值型100，此时插入的数据可以满足存储要求，操作成功；对于数据类型无法转换的，例如无法将字符型'aaa'转换为数值型数据，此时才返回插入数据失败。

当数据被用于表达式计算时，其类型也对计算的操作产生限制，符合限制条件的数据类型才能成功执行计算，例如数学类计算要求其参数均为数值型、日期类计算要求其参数均为日期时间型。但如果用户显式指定了数据转换，或者根据YashanDB的隐式转换规则可以成功执行转换，则这些跨数据类型的表达式计算是可以成功执行的。

在一些其他场景中，例如数据被用于函数的参数传递，而其类型与函数定义的参数类型不一致时，系统均会先执行数据类型的隐式转换，以保证最大的兼容性和计算能力。

基于某些数据类型在其属性和操作限制上的共同点，YashanDB的数据类型可以划分为如下几大类：

- 数值型：与数字（Numeric）相关的数据属于此类型，具体包含TINYINT、SMALLINT、INT/INTEGER、BIGINT、NUMBER、FLOAT、DOUBLE、BIT等数据类型。
- 字符型：与文本字符（Character）相关的数据属于此类型，具体包含CHAR、VARCHAR等数据类型。
- 日期时间型：与日期时间（Datetime）相关的数据属于此类型，具体包含DATE、TIME、TIMESTAMP、INTERVAL YEAR TO MONTH、INTERVAL DAY TO SECOND等数据类型。
- 布尔型：与布尔逻辑（Boolean）相关的数据属于此类型，具体为Boolean数据类型。
- 大对象型：与大二进制、大文本等大对象（Large Object）相关的数据属于此类型，具体包含CLOB、BLOB等数据类型。
- ROWID/UROWID：用于表示某条行记录的物理地址的数据属于此类型。
- RAW：与VARCHAR类似的可变长度数据类型，多被用于存储二进制对象。
- JSON：与JSON相关的可变长度数据类型，多被用于存储JSON格式的二进制对象。
- ST_GEOMETRY：与GIS相关的空间数据类型，具体包含POINT、LINESTRING、POLYGON、MULTIPOINT、MULTILINESTRING、MULTIPOLYGON等数据类型。
- XMLTYPE：与XML相关的可变长度数据类型，底层以CLOB形式存储XML格式的数据。
- BOX2D：与ST_GEOMETRY相关的空间数据类型，用于表示ST_GEOMETRY的二维边界框。

本章节将按上述列举大类分别阐述YashanDB内置的所有数据类型的定义以及它们之间的转换规则。

> **Note**：
> 以下数据类型只适用于HEAP表：
> - BIT
> - NCLOB
> - NCHAR
> - NVARCHAR
> - FLOAT（当USE_NATIVE_TYPE为FALSE时）
> - ROWID
> - XMLTYPE
> - 用户自定义类型
>
> 以下数据类型只适用于单机HEAP表：
> - ST_GEOMETRY
> - BOX2D

数值型

数值型数据包括正负整数、小数、0、正负无穷（Inf、-Inf）、以及非数（Nan），主要应用于数学运算、数字计量等场景。YashanDB支持对此类数据的加减乘除取模运算，并提供统计函数和聚集函数来满足各式场景的需求。

可以将数值型数据进一步划分为整数类型（Integer Data Type）、浮点类型（Floating-Point Data Type）、NUMBER类型（Number Data Type）、以及BIT类型（Bit Data Type）。YashanDB为不同的数值类型定义了值域、字节长度、精度等属性。

整数类型（Integer Data Type）

整数类型为定长、精确数值类型，是没有小数部分的整数，具体包括TINYINT、SMALLINT、INT、BIGINT四种数据类型，每个类型有不同的字节长度和值域，建表时需要根据数据存储所需的空间来决定使用哪一种类型。

INTEGER、PLS_INTEGER/PLS_INTEGER为INTEGER的别名，行为完全同INT。

整数类型通过十进制精度（Decimal Precision，数字0~9）存储，在值域和精度范围以内的十进制数字都可以被准确的存储。

INT类型在语法上支持定义时带精度信息，精度范围为 0 ~ 255。

当配置参数USE_NATIVE_TYPE为FALSE时，TINYINT、SMALLINT、INT、BIGINT返回值类型为NUMBER，precision为38，scale为0。

示例

```
CREATE TABLE int_test(c1 INT(255));
INSERT INTO int_test VALUES(128);
SELECT c1 res FROM int_test;
RES
------------
128
```

存储属性

类型	字节长度	值域
TINYINT	1	$[-2^7, 2^7 - 1]$
SMALLINT	2	$[-2^{15}, 2^{15} - 1]$
INT	4	$[-2^{31}, 2^{31} - 1]$
BIGINT	8	$[-2^{63}, 2^{63} - 1]$

浮点类型（Floating-Point Data Type）

浮点类型为定长、非精确数值类型，具体包括FLOAT、DOUBLE两种数据类型。在YashanDB中，浮点类型的行为与行业标准IEEE Standard 754保持一致。

BINARY_FLOAT、REAL为FLOAT的别名，行为完全同FLOAT。BINARY_DOUBLE为DOUBLE的别名，行为完全同DOUBLE。

浮点类型通过二进制精度（Binary Precision，数字0和1）存储。

- 浮点类型拥有更宽广的值域，可以表达特殊值Inf、-Inf、Nan，对小数点的位置也没有限制。
- 无法避免由于二进制精度转换至十进制精度所带来的误差，所以一些数字无法被浮点数类型准确的表达（例如0.1）。且精度根据USE_NATIVE_TYPE配置参数的值存在不同规则：
  - 当USE_NATIVE_TYPE为TRUE时，YashanDB只做语法兼容，等价于浮点类型本身（例如DOUBLE`(p)`等价于DOUBLE）。
  - 当USE_NATIVE_TYPE为FALSE时，FLOAT类型不适用于列存表，在行存表中，FLOAT类型不带精度信息时，默认为FLOAT(126)，FLOAT后面带精度信息时，精度值为设置值。

存储属性

类型	字节长度	值域	可保证准确的十进制精度
FLOAT	4	[-3.402823E38, -1.401298E-45] 0 [1.401298E-45, 3.402823E38] 数字3.402823和1.401298为四舍五入的值，非最精确值	6位
DOUBLE	8	[-1.79769313486232E308, -4.94065645841247E-324] 0 [4.94065645841247E-324, 1.79769313486232E308] 数字1.797693134862315807和4.94065645841247为四舍五入的值，非最精确值	15位

使用规则

定义格式

浮点类型有如下几种定义格式：

```
FLOAT
DOUBLE
FLOAT(m,d)
DOUBLE(m,d)
FLOAT(p)
DOUBLE(p)
```

其中，在USE_NATIVE_TYPE为TRUE时，`(m,d)`和`(p)`均为语法兼容格式，即支持DOUBLE`(m,d)`和DOUBLE`(p)`语法，但二者均等价于DOUBLE，无实际含义，但存在如下约束规则：

- m：Magnitude，0 < m <= 255
- d：Division， 0 <= d <= 30
- d <= m
- p：Precision，0 <= p <= 53
- 定义FLOAT类型时，0 <= p <= 126 ，大于53小于等于126的部分会当作53处理，如m或者p值超过23，系统将类型转换为DOUBLE类型

示例

```
--定义m超过23的FLOAT类型
DROP TABLE IF EXISTS float_tb;
CREATE TABLE float_tb(c FLOAT(24,1));

--系统将其转换为DOUBLE类型
DESC float_tb
NAME              NULL?     DATATYPE
----------------- --------- ---------------
C                           DOUBLE
```

其中，在USE_NATIVE_TYPE为FALSE时，在实现上支持FLOAT缺省或者带精度信息，存在如下规则：

- 当不带精度信息时， p = 126
- 当带精度信息时，1 <= p <= 126

示例（HEAP表）

```
DROP TABLE IF EXISTS float_tb;
CREATE TABLE float_tb(c FLOAT(2));

DESC float_tb
NAME              NULL?     DATATYPE
----------------- --------- ---------------
C                           FLOAT(2)
```

显示方式

浮点类型以科学计数法的形式显示。

特殊值

浮点类型有3个特殊值：Nan、Inf、-Inf。它们的大小排序为：Nan>Inf>正数>0>负数>-Inf。

下表列示一些特殊场景下将产生这些特殊值或0：

特殊场景	产生值
大于浮点类型值域的最大值，但被用浮点数表示	Inf
小于浮点类型值域的最小值，但被用浮点数表示	-Inf
绝对值小于值域的最小绝对值，但被用浮点数表示	0
正浮点数或Inf作为被除数与0作除法	Inf
负浮点数或-Inf作为被除数，与0作除法	-Inf
Inf、-Inf之间作除法	Nan
Nan作为被除数，与0作除法	Nan

当浮点数大小超过值域的最大绝对值时，会被表示为Inf或-Inf；当浮点数大小小于值域的最小绝对值时，会被表示成0。

精度约束

当浮点类型的实际精度超过其最大精度限制时，超过精度的部分会被舍弃，且舍弃规则无法确定为四舍五入或截断，舍弃后数字的最大精度位的结果也无法预测。

由于存储方式的差异，将其他数值型转换为浮点类型时可能产生无法避免的误差。

因此，浮点类型不适合用在对精确程度要求较高的场景，也不适合用在涉及大量等于比较条件和边界值的场景。

示例

```
SET NUMWIDTH 30;
-- Example 1: 向FLOAT插入超过最大精度数字，超过最大精度位的部分被舍弃，且舍弃方式并非四舍五入或截断
CREATE TABLE number_f (c1 FLOAT);
INSERT INTO number_f VALUES (10.567895678956789);
COMMIT;

SELECT * FROM number_f;
                            C1
-------------------------------
               1.05678959E+001


-- Example 2: 将NUMBER、整数类型转换为浮点类型会产生误差
CREATE TABLE number_bn (c1 BIGINT, c2 NUMBER(38, 60));
INSERT INTO number_bn VALUES (9223372036854775800, 0.00000000000000000000000000000000000000000000000401298);
COMMIT;

SELECT * FROM number_bn;
                  C1                             C2
--------------------- -------------------------------
  9223372036854775800  4.01298000000000000000000E-46

SELECT CAST(C1 AS FLOAT) float1,
CAST(C2 AS FLOAT) float2
FROM number_bn;
                        FLOAT1                         FLOAT2
------------------------------- -------------------------------
               9.22337204E+018                              0
```

NUMBER类型（Number Data Type）

NUMBER类型为非定长、精确的数值类型，它可以自由指定精度（P，Precision）和刻度（S，Scale）。NUMBER类型可以准确表示符合精度和刻度要求的任何数字，因此非常适合用于需要精确计算且涉及小数的场景。

DECIMAL、NUMERIC为NUMBER的别名，行为完全同NUMBER。

存储属性

类型	字节长度	值域	存储方式
NUMBER	1~22	0 绝对值 `[1E-130，1E126)`	通过十进制精度（Decimal Precision，数字0~9）存储，在值域和精度范围以内的十进制数字都可以被准确的存储。

使用规则

定义格式

NUMBER类型的定义格式为 `NUMBER(P,S)` 或 `NUMBER` ，其中P和S的含义如下：

- P：精度，表示数字的有效位数（不包含小数点、正负符号的位数），取值范围[1,38]；也可以为*，表示精度为38。
- S：刻度，表示数字从小数点到最右侧有效数字的位数，取值范围[-84,127]。刻度为正数，表示小数点在最小位数左侧（例如3.14的格式定义可以为 `NUMBER(3, 2)` ）；刻度为负数，表示小数点在最小位数右侧（例如31400的格式定义可以为 `NUMBER(3, -2)` ）。
- P-S：表示整数的最大位数。

当定义格式为NUMBER，不指定P/S时，表示为浮动精度。

精度约束

不指定P/S时为浮动精度，以实际计算产生的值为准，无精度限制。

指定P/S：

- 实际刻度超过定义的S时，系统将超过的部分四舍五入至最小刻度位。
- 实际精度超过定义的P时，系统报错。

支持科学记数法作为值输入

- 支持诸如1.24E3的格式处理。
- 异常带E的数值格式如E1，1E，1.3E2A等进行相应报错。

示例

```
-- Example 1: 将31401表示为number(3, -2)。因为最小刻度是百分位，31401会被四舍五入为31400
SELECT CAST(31401 AS NUMBER(3,-2)) FROM DUAL;
                              CAST(31401ASNUMBER(3
--------------------------------------------------
                                             31400

-- Example 2: 将31400表示为NUMBER(1, -2)。因为最大精度是1，但31400有三位有效数字，所以会有larger than specified precision allowed报错
SELECT CAST(31400 AS NUMBER(1, -2)) FROM DUAL;
[1:34]YAS-00025 value is larger than specified precision allowed for this column

-- Example 3: 支持科学记数法作为值输入，同输入全数值有一致的结果
SELECT CAST('1.24E3' AS NUMBER) FROM DUAL;
                              CAST('1.24E3'ASNUMBE
--------------------------------------------------
                                             1240
-- Example 4: 支持精度以*进行输入，精度为*时表示是38
SELECT CAST('12.38' AS NUMBER(*,0)) FROM DUAL;
```

```
CAST('12.38'ASNUMBER
--------------------
                  12

CREATE TABLE Etest7293 (NUM1 NUMBER(*,0),NUM2 NUMBER(*), NUM3 NUMBER, NUM4 NUMBER(*,23));

desc Etest7293;
NAME            NULL?     DATATYPE
--------------- --------- ---------------------
NUM1                      NUMBER(38)
NUM2                      NUMBER
NUM3                      NUMBER
NUM4                      NUMBER(38,23)
```

BIT类型（Bit Data Type）

BIT类型支持1-64位宽度（Size）的二进制位图，每BIT位只允许存放0/1值，其他值将视为非法BIT类型。

以字符串形式插入/更新BIT类型时，字母b开头的字符串以二进制数值处理，其它以十进制字符串处理。

BIT类型以二进制输出。

BIT类型的定义格式为`BIT[(Size)]`，Size：表示BIT的宽度，取值范围为`[1,64]`，省略时，系统取默认值1。

BIT类型仅适用于HEAP表。

示例（HEAP表）

```
CREATE TABLE number_nb (numbera NUMBER,numberb BIT(64));
INSERT INTO number_nb VALUES (10,10);

SELECT * FROM number_nb;
    NUMBERA NUMBERB
---------- ----------------------------------------------------------------
         10 1010
```

显示宽度调整

SET NUMWIDTH用于设置浮点类型及NUMBER类型数据输出时的显示宽度，只针对当前会话有效。

其中，SET NUMWIDTH=SET NUM，用于设置显示宽度；SHOW NUMWIDTH=SHOW NUM，用于显示设置的宽度值。

系统对浮点类型及NUMBER类型数据输出时的显示宽度默认为10。

YashanDB对浮点类型数字均采用科学计数法显示，显示格式为：

- 正数 x.yyyyE{+|-}zzz
- 负数 -x.yyyyE{+|-}zzz

其中只有yyyy显示内容长度受到width限制；指数zzz固定显示3个字节，其前面固定显示符号'+' 或 '-'。

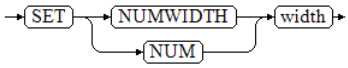
YashanDB对NUMBER类型数字的显示规则如下：

- 如果是纯小数，则小数点前的0会省略不显示。如果纯小数为0.00000xxx 其中xxx为任意数字，则使用科学计数法显示，否则使用非科学计数法显示。
- 如果是非纯小数，且给定的宽度能够完整输出整数内容，则不会使用科学计数法显示，否则使用科学计数法显示。
- NUMBER类型数字的科学计数法显示格式为：
  - 正数： x.yyyyE{+|-}zz
  - 负数：-x.yyyyE{+|-}zz

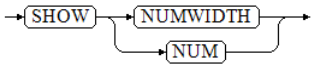
其中yyyy显示内容长度受到width限制；指数zz默认2个字节，最大3个字节，其前面固定显示符号'+' 或 '-'。

语句定义

set numwidth::=



show numwidth::=



width

该语句用于指定显示宽度值，值范围为 `[2,128]`。

当此值设置过小，导致某个数字无法完整显示出来时，系统输出按此值个数的'#'。

示例

```
--创建numbers_nfd表，包含NUMBER、FLOAT、DOUBLE三个类型的字段
CREATE TABLE numbers_nfd (cnumber NUMBER, fnumber FLOAT, dnumber DOUBLE);
INSERT INTO numbers_nfd VALUES (0.0000002334, 0.0000002334, -0.0000002334);
INSERT INTO numbers_nfd VALUES (2334.0000002334, 2334.0000002334, -2334.0000002334);
INSERT INTO numbers_nfd VALUES (233423342334.0000002334, 233423342334.0000002334, -233423342334.0000002334);
COMMIT;

--默认宽度下显示
SELECT * FROM numbers_nfd;
    CNUMBER     FNUMBER     DNUMBER
----------- ----------- -----------
 2.3340E-07  2.334E-007  -2.33E-007
       2334  2.334E+003  -2.33E+003
 2.3342E+11  2.334E+011  -2.33E+011

--设置显示宽度为20
SET NUMWIDTH 20;
SELECT * FROM numbers_nfd;
             CNUMBER               FNUMBER               DNUMBER
-------------------- -------------------- --------------------
          .0000002334       2.33400002E-007           -2.334E-007
      2334.0000002334            2.334E+003  -2.334000000233E+003
 233423342334.0000002        2.3342334E+011   -2.33423342334E+011

--设置显示宽度为8
SET NUMWIDTH 8;
SELECT * FROM numbers_nfd;
  CNUMBER   FNUMBER   DNUMBER
--------- --------- ---------
 2.33E-07  2.3E-007  ########
     2334  2.3E+003  ########
 2.33E+11  2.3E+011  ########

--设置显示宽度为5
SET NUMWIDTH 5;
SELECT * FROM numbers_nfd;
CNUMBER FNUMBER DNUMBER
------- ------- -------
      0   #####   #####
   2334   #####   #####
  #####   #####   #####
```

字符型

YashanDB中的字符型包括英文字母、中文汉字、数字字符和特殊字符等，多用于存储文本数据。此外，该类型还能通过隐式数据转换，灵活地参与到数值型、日期时间型的运算场景之中。

字符型具体可分为CHAR、VARCHAR、NCHAR和NVARCHAR四种数据类型。

存储属性

类型	字节长度
CHAR	1~8000
VARCHAR	1~32000
NCHAR	1~8000
NVARCHAR	1~32000

> **Note**：
> 在HEAP表中，长度超过8000字节的VARCHAR/NVARCHAR列会转换成LOB类型进行存储。创建表时，若指定某个列的数据类型大小超过8000字节，该列就会转换成LOB类型进行存储。
> 例如在UTF8字符集（一个字符最大可以占用4字节）下，定义了VARCHAR(2001 CHAR)列，则可能会存储8004字节的数据，此时该列会转换成LOB类型进行存储。

CHAR和VARCHAR

CHAR数据类型用于指定固定长度的字符串，VARCHAR数据类型用于指定可变长度的字符串。

CHARACTER可作为CHAR别名使用，含义与CHAR含义相同。VARCHAR2、CHARACTER VARYING可作为VARCHAR别名使用，含义与VARCHAR含义相同。

定义格式：

类型	语法格式	规则
CHAR	CHAR(Size[byte])	定长字符串，Size的范围是[1,8000]，不指定Size时，Size默认为1；当插入的字符串小于Size时会发生空格补位行为；当插入的字符串大于Size时会插入失败，报错。
VARCHAR	VARCHAR(Size[byte])	变长字符串，Size的范围是[1,32000]，当插入的字符串小于Size时不会发生空格补位行为；当插入的字符串大于Size时会插入失败，报错。
CHAR	CHAR(Size(char))	定长字符串，Size的范围是[1,8000]，不指定Size时，Size默认为1；当插入的字符串小于Size时会发生空格补位行为，补充的空格数为缺少的字符数；当插入的字符串大于Size时会插入失败，报错。
VARCHAR	VARCHAR(Size(char))	变长字符串，Size的范围是[1,32000]，当插入的字符串小于Size时不会发生空格补位行为；当插入的字符串大于Size时会插入失败，报错。

Size：宽度，表示最大长度，根据可选长度单位可分为字节长度和字符长度，行存表支持两种长度单位，列存表仅支持字节长度单位。

Size后可选单位为byte和char，分别为按照字节和按照字符，其中byte和char与Size之间可有空格也可以没有。若不指定单位，默认为byte。长度单位与NCHAR及NVARCHAR类型有所不同，请注意区分。

示例（HEAP表）

```
--定义不同单位的CHAR类型数据
CREATE TABLE chartable (c1 CHAR(30),c2 CHAR(30 byte),c3 CHAR(30 CHAR));
INSERT INTO chartable VALUES('a','abc','abcccc');
COMMIT;
SELECT * FROM chartable;
```

```
C1                               C2                               C3
-------------------------------- -------------------------------- ----------------------------------------------------------------
-------
a                                abc                              abcccc

--定义不同单位的VARCHAR类型数据
CREATE TABLE varchartable (v1 VARCHAR(30),v2 VARCHAR(30 byte),v3 VARCHAR(30 CHAR));
INSERT INTO varchartable VALUES('a','abc','abcccc');
COMMIT;
SELECT * FROM varchartable;

V1                               V2                               V3
-------------------------------- -------------------------------- ----------------------------------------------------------------
-------
a                                abc                              abcccc

--定义字符型数据时，如超出长度范围返回错误
CREATE TABLE overlimit (c1 CHAR(9000 byte),c2 CHAR(9000 CHAR));

[1:32]YAS-04204 number of column size must be between 1 and 8000
```

NCHAR和NVARCHAR

NCHAR用于指定支持UNICODE的固定长度字符串，NVARCHAR用于指定支持UNICODE的可变长度字符串，这两种字符型仅支持在配置UNICODE字符集的数据库中使用，可以对多语言数据进行存储。

NVARCHAR2可作为NVARCHAR别名使用，含义与NVARCHAR含义相同。

NCHAR和NVARCHAR类型仅适用于HEAP表。

定义格式：

类型	语法格式	规则
NCHAR	NCHAR(Size)	定长字符串，Size的范围是[1,4000]，不指定Size时，Size默认为1；当插入的字符串小于Size时会发生空格补位行为；当插入的字符串大于Size时会插入失败，报错。
NVARCHAR	NVARCHAR(Size)	变长字符串，Size的范围是[1,16000]，当插入的字符串小于Size时不会发生空格补位行为；当插入的字符串大于Size时会插入失败，报错。

Size用于表示字符个数，必须为一个整数数值，数据的存储长度为Size的2倍，单位为字符长度。

NCHAR类型存储长度上限为8000字节，NVARCHAR类型存储长度上限为32000字节，超过如上大小限制的数据不会进行存储。

示例（HEAP表）

```
--定义不同单位的NCHAR类型数据
CREATE TABLE nchartable (c1 NCHAR(30));
INSERT INTO nchartable VALUES('a');
COMMIT;
SELECT * FROM nchartable;

C1
----------------------------------------------------------------
a


--定义不同单位的NVARCHAR类型数据
CREATE TABLE nvarchartable (v1 NVARCHAR(30));
INSERT INTO nvarchartable VALUES('a');
COMMIT;
SELECT * FROM nvarchartable;

V1
----------------------------------------------------------------
```

```
a

--定义字符型数据时，如超出长度范围返回错误
CREATE TABLE overlimit_nchar (c1 NCHAR(9000));

[1:40]YAS-04204 number of column size must be between 1 and 4000

CREATE TABLE overlimit_nvarchar (c1 NVARCHAR(20000));

[1:46]YAS-04204 number of column size must be between 1 and 16000
```

字符型排序

当对字符型数据执行排序操作时：

- 默认基于字符的ASCII值排序，例如'a'小于'b'，'1'小于'2'。
- 通过NLSSORT函数，可以对UTF8编码的数据进行拼音排序。

字符型比较

当两个字符型数据执行比较运算时：

- 执行大小写敏感的字符串比较。
- 若比较符两边均为CHAR或均为NCHAR类型，YashanDB会先将较短的CHAR或NCHAR空格补位至较长CHAR的长度，再进行值比较。
- 若比较符号的任意一边为VARCHAR或NVARCHAR类型，则不进行空格补位直接比较。

示例

```
-- Example 1: 比较符号两侧都是CHAR类型时，YashanDB会先将长度补齐再比较
SELECT 1 FROM DUAL
WHERE CAST('A' AS CHAR(50)) = CAST('A' AS CHAR(30));
           1
------------
           1

-- Example 2: 比较符号任何一侧为VARCHAR类型时，YashanDB不会补齐长度直接比较
SELECT 1 FROM DUAL
WHERE CAST('A' AS CHAR(50)) > CAST('A' AS VARCHAR(50));
           1
------------
           1

SELECT 1 FROM DUAL
WHERE CAST('A' AS VARCHAR(30)) < CAST('A' AS CHAR(50));
           1
------------
           1

-- Example 3: 不指定CHAR的Size大小时，Size默认为1
SELECT CAST('A' AS CHAR) result FROM DUAL;
RESULT
------
A

SELECT CAST('ABC' AS CHAR) result FROM DUAL;
RESULT
------
A

SELECT CAST('ABC' AS CHAR(1)) result FROM DUAL;
RESULT
------
A

SELECT CAST('ABC' AS CHAR(2)) result FROM DUAL;
RESULT
```

```
------
AB
```

布尔型

布尔型数据的值只有1（TRUE）和0（FALSE），其字节长度为1，可用于指示某个二元特性的状态（例如该用户是否为VIP）。此外，该类型也是很多条件（例如比较运算、LIKE/NOT LIKE语句等）的输出类型，被广泛用于WHERE和HAVING语句中来过滤数据。

YashanDB对布尔型数据的处理规则：

1.允许对布尔型字段插入如下值（大小写不敏感）。

类型	输入值	转换值
字符型	'true'、't'、 'yes'、 'y'、 'on'、 '1'	1
字符型	'false'、'f'、 'no'、 'n'、 'off'、 '0'	0
标识符	true	1
标识符	false	0
整型数值	非0整数	1
整型数值	0	0

2.允许布尔型与整型数值互转，其中非0的整数可以转换为true，但true只能转换为整数1。

3.允许布尔型和整型数值进行比较。

示例

```
CREATE TABLE bools(c_b1 BOOLEAN, c_b2 BOOLEAN, c_b3 BOOLEAN, c_b4 BOOLEAN);

--插入其他类型转布尔
INSERT INTO bools VALUES('t', 'no', 'on', 4);
SELECT * FROM bools;
C_B1                 C_B2                 C_B3                 C_B4
-------------------- -------------------- -------------------- --------------------
true                 false                true                 true

--布尔转整型参与比较
SELECT CASE CAST(c_b1 AS INT)
WHEN 1 THEN '11111'
WHEN 2 THEN '22222'
END b
FROM bools
WHERE c_b2<4;
B
---------
11111
```

日期时间型

日期时间型是时间维度上的数据类型，多用于时间维度上的数据提取和分析。可进一步将类型划分为：

- 日期时间类型（Date&Time Data Type）：表示某个日期或时刻。具体包括日期（DATE）、时间（TIME）、时间戳（TIMESTAMP）三个数据类型。
- 间隔类型（Interval Data Type）：表示两个日期或时刻之间的间隔长度。具体包括年到月间隔（INTERVAL YEAR TO MONTH）、天到秒（INTERVAL DAY TO SECOND）两个数据类型。

日期类型（Date Data Type)

日期类型存储了与时区无关的逻辑日历信息，其信息包括年、月、日（时、分、秒）。 可通过set命令来设置date格式。

存储属性

类型	字节长度	取值范围	精度
DATE	8	0001-01-01 00:00:00 ~ 9999-12-31 23:59:59	秒

数据格式

DATE类型的默认格式为YYYY-MM-DD，也可以按类似YYYY-MM-DD [HH[24]][:MI][:SS]的标准格式进行指定，其中各字符含义为：

- YYYY：以四位数字表示的年份，取值范围[0001,9999]。
- MM：以一位或两位数字表示的月份，取值范围[1,12]。
- DD：以一位或两位数字表示的日期，取值范围[1,31]。
- HH[24]：以一位或两位数字表示的小时，24表示24小时制，取值范围[0,23]。
- MI：以一位或两位数字表示的分钟，取值范围[0,59]。
- SS：以一位或两位数字表示的秒钟，取值范围[0,59]。

示例

```
CREATE TABLE date_date(C1 DATE);
INSERT INTO date_date VALUES ('2020-01-01');
INSERT INTO date_date VALUES ('2020-1-1');
COMMIT;

SELECT C1 FROM date_date;
C1
--------------------------------
2020-01-01 00:00:00
2020-01-01 00:00:00
```

时间类型（Time Data Type）

时间类型表示一日以内的时间，包含时、分、秒和微秒。

存储属性

类型	字节长度	取值范围	精度
TIME	8	00:00:00.000000 ~ 23:59:59.999999	微秒

数据格式

按类似HH[24][:MI][:SS][.FF] 的标准格式进行指定，其中各字符含义为：

- HH[24]：以一位或两位数字表示的小时，24表示24小时制，取值范围[0,23]。
- MI：以一位或两位数字表示的分钟，取值范围[0,59]。

- SS：以一位或两位数字表示的秒钟，取值范围[0,59]。
- FF：以一位到六位数字表示的微秒，取值范围[0,999999]。

对TIME类型执行字符串转换时，上面标准格式里的所有部分可以从低位向上省略，省略的部分按0补位。

示例

```
CREATE TABLE date_time(C1 TIME);
INSERT INTO date_time VALUES ('23:59:59.999999');
INSERT INTO date_time VALUES ('2:5:5');
INSERT INTO date_time VALUES ('1:2');
COMMIT;

SELECT C1 FROM date_time ;
C1
--------------------
23:59:59.999999
02:05:05.000000
01:02:00.000000
```

时间戳类型（Timestamp Data Type）

时间戳类型是信息最为全面的日期时间类型，其信息包含了年、月、日、时、分、秒、微秒。

存储属性

类型	字节长度	取值范围	精度
TIMESTAMP	8	1-1-1 00:00:00.000000 ~ 9999-12-31 23:59:59.999999	微秒

数据格式

按类似YYYY-MM-DD [HH[24]][:MI][:SS][.FF]的标准格式进行指定，其中各字符含义为：

- YYYY：以四位数字表示的年份，取值范围[0001,9999]。
- MM：以一位或两位数字表示的月份，取值范围[1,12]。
- DD：以一位或两位数字表示的日期，取值范围[1,31]。
- HH[24]：以一位或两位数字表示的小时，24表示24小时制，取值范围[0,23]。
- MI：以一位或两位数字表示的分钟，取值范围[0,59]。
- SS：以一位或两位数字表示的秒钟，取值范围[0,59]。
- FF：以一位到六位数字表示的微秒，取值范围[0,999999]。

> **Note**：
> 微秒的特殊处理：
> - 在数据类型定义时，微秒精度可定义范围为0~9，但输出的实际精度值为6。
> - 系统可以接收0~9位数字的微秒输入，但会将其补齐或四舍五入到6位。

示例

```
CREATE TABLE date_timestamp (C1 TIMESTAMP);
INSERT INTO date_timestamp VALUES ('2020-01-01 12:30:30.123456');
INSERT INTO date_timestamp VALUES ('2020-1-1 12:30:30.123456');
INSERT INTO date_timestamp VALUES ('2020-1-1');
COMMIT;

SELECT C1 FROM date_timestamp;
C1
----------------------------------------------------------------
2020-01-01 12:30:30.123456
2020-01-01 12:30:30.123456
2020-01-01 00:00:00.000000
```

```
--可定义最大9的微秒精度，但实际存储精度为6
CREATE TABLE timestamp_test(c1 TIMESTAMP(3));
INSERT INTO timestamp_test VALUES('2012-1-1 1:1:1.123456789');
SELECT c1 res FROM timestamp_test;
RES
----------------------------------------------------------------
2012-01-01 01:01:01.123457
```

年到月间隔类型（Interval Year To Month Data Type）

年到月间隔类型表示以年月为单位的时间间隔。

存储属性

类型	字节长度	取值范围	精度
INTERVAL YEAR TO MONTH	4	-178000000-00 ~ 178000000-00	月

定义格式

年到月间隔类型的定义格式为INTERVAL YEAR [(year_precision)] TO MONTH。year_precision：表示年的精度，取值范围[0,9]，省略时取默认值2。

> **Note**：
> 00-00特殊处理：
> 在年精度最大值9以内，和月精度最大值2以内，输入任意个0，都将被视为'00'，而不受定义格式里所指定的精度限制。

数据格式

按`YYYY-MM`标准格式或`INTERVAL 'YYYY' YEAR(MM)`指定：

- YYYY：年份，有效值为-178000000到178000000。
- MM：月份，有效值为0到11。

示例

```
CREATE TABLE date_iytm (no INT, c1 INTERVAL YEAR(4) TO MONTH);
INSERT INTO date_iytm VALUES(1,'1000-10');
INSERT INTO date_iytm VALUES(2,'1000-0');
INSERT INTO date_iytm VALUES(3,'000000000-11');
INSERT INTO date_iytm VALUES(4,INTERVAL '1000' YEAR(9));
INSERT INTO date_iytm VALUES(5,INTERVAL '11' MONTH);
COMMIT;

SELECT * FROM date_iytm ORDER BY no;
          NO C1
------------ ---------------
           1 +1000-10
           2 +1000-00
           3 +00-11
           4 +1000-00
           5 +00-11
```

天到秒间隔类型（Interval Day To Second Data Type）

天到秒间隔类型表示以天、时、分、秒、微秒为单位的时间间隔。

存储属性

类型	字节长度	取值范围	精度
INTERVAL DAY TO SECOND	8	-100000000 00:00:00.000000 ~ 100000000 00:00:00.000000	微秒

定义格式

天到秒间隔类型的定义格式为`INTERVAL DAY[(day_precision)] TO SECOND[(fractional_seconds_precision)]`，其中：

- day_precision：表示天的精度，语法支持[0,9]，实际取值范围[0,8]。
- fractional_seconds_precision：表示微秒的精度，取值范围[0,9]。

`(day_precision)`省略时，系统取默认值2；`(fractional_seconds_precision)`省略时，系统取默认值6。

数据格式

该类型的标准格式为`DD HH:MI:SS[.FF]`或`INTERVAL 'DD' DAY(day_precision)`或`INTERVAL 'HH' HOUR或INTERVAL 'MI:SS.DDDDDD' MINUTE TO SECOND(fractional_seconds_precision)`，各字符含义为：

- DD：日期，取值范围[-100000000，100000000]。
- HH：小时，取值范围[0，23]。
- MI：分钟，取值范围[0，59]。
- SS：秒钟，取值范围[0，59]。
- FF：微秒，取值范围[0，999999]。

> **Note**：
> 微秒的特殊处理：
> - 在数据类型定义时，微秒精度可定义范围为0~9，但输出的实际精度值为6。
> - 系统可以接收0~9位数字的微秒输入，但会将其补齐或四舍五入到定义的精度（定义的精度大于6时按6处理）。

示例

```
CREATE TABLE date_idts(no INT, c1 INTERVAL DAY(9) TO SECOND(9));
INSERT INTO date_idts VALUES(1,'50 10:30:59.999999');
INSERT INTO date_idts VALUES(2,'50 00:00:59.999999');
INSERT INTO date_idts VALUES(3,'00 10:30:00.000000');
INSERT INTO date_idts VALUES(4,'50 00:00:00.000000');
INSERT INTO date_idts VALUES(5,INTERVAL '50' DAY(6));
INSERT INTO date_idts VALUES(6,INTERVAL '10' HOUR);
INSERT INTO date_idts VALUES(7,INTERVAL '30:59.9' MINUTE TO SECOND(6));
INSERT INTO date_idts VALUES(8,INTERVAL '30:59.9999999' MINUTE TO SECOND(6));
INSERT INTO date_idts VALUES(9,INTERVAL '30:59.9999990' MINUTE TO SECOND(6));
INSERT INTO date_idts VALUES(10,INTERVAL '30:59.9999990' MINUTE TO SECOND(3));
INSERT INTO date_idts VALUES(11,INTERVAL '30:59.999999567' MINUTE TO SECOND(8));
COMMIT;

SELECT * FROM date_idts ORDER BY no;
          NO C1
------------ --------------------------------
           1 +50 10:30:59.999999
           2 +50 00:00:59.999999
           3 +00 10:30:00.000000
           4 +50 00:00:00.000000
           5 +50 00:00:00.000000
           6 +00 10:00:00.000000
           7 +00 00:30:59.900000
           8 +00 00:31:00.000000
           9 +00 00:30:59.999999
          10 +00 00:31:00.000000
          11 +00 00:31:00.000000
```

大对象型

LOB大对象型用于在数据库中存储较大二进制或较大文本的可变长度数据。

YashanDB的LOB类型包括BLOB、CLOB和NCLOB。

存储属性

类型	字节长度
BLOB	1~4G*DB_BLOCK_SIZE
CLOB	1~4G*DB_BLOCK_SIZE
NCLOB	行存： 1~4G*DB_BLOCK_SIZE 列存： 无此类型

定义格式：

类型	格式	规则
BLOB	BLOB	变长二进制串，无需指定size
CLOB	CLOB	变长字符串，无需指定size
NCLOB	NCLOB	支持UNICODE的变长字符串，无需指定size

YashanDB对大对象类型的存储包含行内存储和行外存储两种方式：

- 当一行的LOB列的数据小于一定的字节限制时，LOB数据将存储在行内。对于HEAP表，该限制是4000字节；对于TAC/LSC表，该限制是32000字节。
- 当超过上述字节限制时，LOB数据存入单独的大对象数据空间（可为其指定表空间），行内存储的则是指向LOB数据的指针。

> **Note**：
> 1. 数据在存储时会产生一些内部元信息，行存表（HEAP表）字节限制包括这部分元数据所占空间，但列存表（TAC和LSC表）字节限制不包括元数据所占空间。
> 2. 以DB_BLOCK_SIZE = 8K为例，LOB类型数据在行存表中的最大字节长度为 `4G*8K=32T` 。

使用规则

使用限制

YashanDB对于LOB型的使用限制如下：

- 不能作为索引列
- 不能修改LOB列的数据类型
- 不能作为分区键
- 不能与其他数据类型进行四则运算和取余运算
- 不能作为比较条件
- 不能使用DISTINCT去重
- 不能用于GROUP BY分组查询

BLOB

BLOB表示二进制大对象，例如照片、视频、音频等文件，对于这类文件的存储，可采用在数据库中只存储文件地址，读取时通过链接获取文件的方式，也可采用直接将其二进制数据作为BLOB字段存储到数据库表中的方式。

由于BLOB内容为二进制数据，除与RAW类型的隐式数据转换外，该数据类型在SQL中不参与其他任何类型转换及运算，但可以通过DBMS_LOB.COMPARE实现与BLOB/RAW类型数据的比较，该高级包同时提供了对BLOB类型的一些运算方法。

列存表中，BLOB在函数需要转成字符串时做UTF-8字符集合法校验。

示例

```
--1.在sales用户下创建含有BLOB字段的customer_img表
CREATE TABLE customer_img (id INT, logo BLOB);
```

本示例运用YashanDB JDBC驱动客户端程序对customer_img插入一条包含图片的记录，该示例引用到了在YashanDB JDBC驱动使用示例程序中的getConnection方法。

```
--2.创建java文件
package jdbc0;
import java.io.FileInputStream;
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.PreparedStatement;

public class Blobexample {
    public static void main(String[] args) throws Exception {
         //创建数据库连接。
        Connection conn = Jdbcexample.getConnection("sales", "sales");

        PreparedStatement pst = null;
        try {
            //生成预处理语句。
            pst = conn.prepareStatement("INSERT INTO customer_img VALUES (?,?)");
            //添加参数。
            pst.setInt(1, 1);
            //读取图片文件，转为二进制
            FileInputStream in = new FileInputStream("./logo.png");

            pst.setBytes(2, trans2Byte(in));
            pst.executeUpdate();
            System.out.println("insert table customer_img succeed!");
            pst.close();
        } catch (SQLException e) {
            if (pst != null) {
                pst.close();
            }
            e.printStackTrace();
        }
    }

    public static byte[] trans2Byte(FileInputStream in) {
        int count = 0;
        try {
            while (count == 0) {
                count = in.available();
            }

            byte[] blob = new byte[count];
            in.read(blob);
            return blob;
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }
}
```

```
--3.在linux下编译并运行
$ javac -d . Blobexample.java
$ java jdbc0.Blobexample
```

```
Connection succeed!
insert table customer_img succeed!
```

```
--4.查询插入的数据（部分显示）
SELECT * FROM customer_img;
          ID LOGO
------------ --------------------------------------------------------------------
           1
89504E470D0A1A0A0000000D49484452000000FA0000004408060000000A82C2838000000467414D410000B18F0BFC61050000003865584966 4D4D002A000
0000800018769000400000001000000 1A000000000002A0020004
```

CLOB

CLOB表示可变长度文本，与VARCHAR类型类似，而VARCHAR类型的最大存储规格为32000字节，对于预计可能会存储超过该规格数据的字段，可将其设为CLOB类型。

- 所有CLOB类型的数据在发送到客户端时会做字符集校验，检验失败将返回错误。
- CLOB类型支持大部分字符型函数，也可以在执行数据转换后参与运算，具体见每个函数的描述。

可以通过DBMS_LOB.COMPARE实现CLOB与CLOB/CHAR/VARCHAR类型数据的比较，该高级包同时提供了对CLOB类型的一些运算方法。

示例

```
--1.在sales用户下创建含有CLOB字段的customer_intro_clob表
CREATE TABLE customer_intro_clob (id INT, intro CLOB);

--2.插入CLOB数据
INSERT INTO customer_intro_clob
VALUES (1, 'It gives me great pleasure to introduce our company.');
COMMIT;

--3.CLOB数据参与字符型运算
SELECT SUBSTRING(intro, 3,5) FROM customer_intro_clob WHERE id = 1;
SUBSTRING(INTRO,3,5)
--------------------------------------------------------------------
 give
```

NCLOB

NCLOB存储UNICODE可变长度数据，与CLOB类型功能类似，最大支持存储1~4G*DB_BLOCK_SIZE数据。

NCLOB类型仅适用于HEAP表。

示例（HEAP表）

```
-- 1.创建表NCLOB_TABLE
CREATE TABLE nclob_table (c1 NCLOB);

-- 2.插入NCLOB数据
INSERT INTO nclob_table
VALUES ('It gives me great pleasure to introduce our company.');
COMMIT;

-- 3.NCLOB数据转换
SELECT CAST(c1 AS CHAR(90)) FROM nclob_table;
CAST(C1ASCHAR(90))
--------------------------------------------------------------------
It gives me great pleasure to introduce our company.
```

RAW

YashanDB中的RAW类型是一种与VARCHAR类似的可变长度数据类型，可以表述二进制数据或者字节字符串。

在定义表时，可以使用RAW类型声明一个列字段，通常被用于存储图形、声音、文档或二进制数据数组等内容，具体解释取决于使用情况。

存储属性

类型	字节长度
RAW	1~8000

定义格式：

类型	格式	规则
RAW	RAW(Size)	变长二进制串，当插入的字符串小于Size时不会发生补0x00的行为。

Size：宽度，表示最大字节长度。

使用规则

使用限制

对于RAW类型数据的使用限制如下：

- 不能作为索引列
- 分布式部署中无法作为分区键

数据类型转换

YashanDB支持RAW和字符型的数据类型转换，包括显式转换和隐式转换。

YashanDB支持RAW和BLOB/ROWID/UROWID/JSON的数据类型转换，支持RAW往CLOB的单向数据类型转换。

显式转换

用户可使用CAST、TO_CHAR等内置函数进行RAW和字符型之间的数据类型转换。

示例

```
SELECT CAST('123' AS RAW(5)) raw_res FROM dual;
RAW_RES
-------
0123

SELECT TO_CHAR(CAST('123' AS RAW(5))) char_res FROM dual;
CHAR_RES
--------
0123
```

隐式转换

1.字符型向RAW类型隐式转换：

系统将连续输入的每个字符解析为二进制数据中四个连续位的十六进制表述，并将其连接后构建RAW类型数据。

输入的字符串中包含不属于十六进制表述范围（0-9、a-f、A-F）的字符时，系统报错。

如果输入字符串数量为奇数，系统将先补充4个都是0的连续位，然后再将字符串转换为RAW类型数据。

当插入或更新一个RAW列字段时，如果输入字符串的字节长度超过RAW列宽度的2倍，系统报错。

示例（HEAP表）

```
CREATE TABLE IF NOT EXISTS raw_table(c1 RAW(10));

--字符型向RAW类型隐式转换
INSERT INTO raw_table VALUES('1234ABCDEF');
INSERT INTO raw_table VALUES('1AB');

--不合法的十六进制表述导致隐式转换失败
INSERT INTO raw_table VALUES('G');
YAS-00223 invalid hex number

--插入字符串的字节长度超过RAW列宽度的2倍导致插入失败
INSERT INTO RAW_TABLE VALUES(LPAD(1,21,1));
YAS-04008 C1 size exceeding limit 10
```

2.RAW类型向字符型隐式转换：

系统将RAW类型数据作为以二进制输入的十六进制表述，转换为字符，其中每个字符是一个十六进制数字（0-9、a-F、a-F），表示四个连续的原始数据位。例如，将位为10101011的原始数据的一个字节做为十六进制值AB，并转换为字符串'AB'。

示例（HEAP表）

```
--对上例中的RAW类型数据执行字符串操作，系统进行隐式转换
SELECT GROUP_CONCAT(c1) FROM raw_table;
GROUP_CONCAT(C1)
----------------------------------------------------------------
1234ABCDEF,01AB
```

3.RAW类型向BLOB类型隐式转换：

因为RAW类型和BLOB类型都可用于存储二进制对象，所以系统可以将RAW数据直接转换为BLOB数据。

示例（HEAP表）

```
--新建BLOB表
CREATE TABLE IF NOT EXISTS blob_table(c1 BLOB);

--可将raw_table里的数据直接插入到blob_table中
INSERT INTO blob_table SELECT c1 FROM raw_table;
```

4.BLOB类型向RAW类型隐式转换：

在BLOB数据字节长度不超过32000字节的情况下，BLOB数据可以被转换成RAW数据。

示例（HEAP表）

```
--可将blob_table里的数据直接插入到raw_table中
INSERT INTO raw_table SELECT c1 FROM blob_table;
```

JSON

YashanDB中的JSON类型是一种可变长度数据类型，通过解析符合标准JSON格式的字符串获得的二进制数据。

> **Note**：
> JSON对象数据类型是基于UTF8字符集的。将字符串编码成JSON对象时，如果字符串的字符集不是UTF8，则先将字符串自动转换为UTF8字符集再进行JSON对象编码。

关于JSON格式的详细描述具体可以参考JSON文档。

存储属性

类型	字节长度
JSON	行存： 1-32MB 列存： 1~32MB

定义格式：

类型	格式	规则
JSON	JSON	变长二进制串，无需指定size

YashanDB的JSON类型存储方式与大对象类型的BLOB类型相同，存储包含行内存储和行外存储两种方式：

- 当一行的JSON列的数据小于一定的字节限制时，JSON数据将存储在行内。对于HEAP表，该限制是4000字节；对于TAC/LSC表，该限制是32000字节。
- 当超过上述字节限制时，JSON数据存入单独的大对象数据空间（可为其指定表空间）。

> **Note**：
> 1. 数据在存储时会产生一些内部元信息，行存表（HEAP表）字节限制包括这部分元数据所占空间，但列存表（TAC和LSC表）字节限制不包括元数据所占空间。
> 2. 以DB_BLOCK_SIZE = 8K为例，JSON类型数据在行存表中的最大字节长度为`4G*8K=32T`。

使用规则

JSON表示可变长度二进制数据，数据通过解析符合标准格式的字符串得到。可以解析为JSON对象的字符串长度上限为32MB。

解析成为JSON类型的数据长度与原有的字符串长度不一定相同，大部分情况下转换为JSON类型后的数据会比原字符串更大。

JSON类型支持与字符串的相互转换，但不能进行加减乘除、比较、拼接等运算。JSON的运算能力主要由具体的JSON相关的内置函数提供。

JSON类型的列字段不能作为分区键。

示例

```
--1.在sales用户下创建含有JSON字段的customer_intro_json表
CREATE TABLE customer_intro_json (id INT, intro JSON);

--2.通过字符串隐式转换，插入JSON数据
INSERT INTO customer_intro_json
VALUES (1, '{"name":"Jack", "city":"Beijing","school":"TsingHua University"}');
COMMIT;

--3.通过JSON构造函数，插入JSON数据
INSERT INTO customer_intro_json
VALUES (2, JSON('{"name":"Bob", "city":"Shenzhen","school":"Shenzhen University"}'));
```

```
COMMIT;

--4.使用JSON_QUERY函数对JSON数据进行查询
SELECT JSON_QUERY(intro, '$.name' WITH WRAPPER) name FROM customer_intro_json;
NAME
--------------------------------------------------------------------
["Jack"]
["Bob"]
```

XMLTYPE

YashanDB中的XMLTYPE类型是一种可变长度数据类型，用于存储XML类型数据，底层以CLOB方式进行存储，支持XMLTYPE类型进行建表、插入，更新和查询功能。

存储属性

类型	字节长度
XMLTYPE	行存：[1,4G*DB_BLOCK_SIZE]

YashanDB的XMLTYPE类型存储方式与大对象类型的CLOB类型相同，存储包含行内存储和行外存储两种方式：

- 当一行的XMLTYPE列的数据小于4000字节时，XMLTYPE数据将存储在行内。
- 当超过4000字节时，XMLTYPE数据存入单独的大对象数据空间（可为其指定表空间），行内存储的则是指向XMLTYPE数据的指针。

定义格式

类型	格式	规则
XMLTYPE	XMLTYPE	变长字符串，无需指定size

使用规则如下：

- 仅适用于HEAP表。
- 对于XMLTYPE类型数据的查询为无格式的全部内容输出。
- 对于XMLTYPE类型数据的插入为无格式的直接插入，不对XMLTYPE数据的合法性做校验。
- 不支持四则运算与大小比较。
- XMLTYPE类型与其他数据类型的转换支持情况请查阅数据类型转换。
- 支持exp/imp使用，使用规格与CLOB一致。
- 无法用于除长度、截取相关函数外的其它函数传参，CONCAT、TRIM、LTRIM、RTIMR返回VARCHAR类型，其余表现形式与CLOB一致。
- 无法用于LOB高级包。

示例（HEAP表）

```
--1.在sales用户下创建含有XMLTYPE字段的tbl_xml表
CREATE TABLE tbl_xml (co1 xmltype);
```

```
--2.给tbl_xml表增添数据
INSERT INTO tbl_xml VALUES('<employee><id>2</id><name>hahaha</name></employee>');
```

```
--3.查看tbl_xml表数据
SELECT * FROM tbl_xml;
```

```
--3.修改tbl_xml表数据
UPDATE tbl_xml SET co1='<employee><id>1</id><name>join</name></employee>';
```

ROWID UROWID

YashanDB单机、共享集群部署中表对象的默认存储方式为HEAP方式，数据按行（Row）组织，系统根据每一行数据所在物理地址信息生成一个全局唯一记录，即ROWID。ROWID用于检索表数据时的寻址，也可以用作每一行数据的唯一标识。

ROWID是一种特殊的数据类型，既能作为数据类型被用户进行列字段声明，也能作为伪列（非表结构中定义的列，但可以被当作列字段进行查询）被使用。

UROWID（Universal ROWID）即通用ROWID，在YashanDB中作为数据类型的一种，用于以二进制串形式存储行表的ROWID。

存储属性

类型	字节长度
ROWID	16
UROWID	1~8000，默认长度为4000

定义格式：

类型	格式	规则
ROWID	ROWID	见下文数据格式
UROWID	UROWID(Size)	变长二进制串，若不指定Size则默认长度为4000

Size：宽度，表示最大字节长度。

数据格式

ROWID的数据格式为`dataoid:spaceid:fileid:blockid:dir`。

在用于存储行表的ROWID时，UROWID的数据格式为`0x1+ROWID的十六进制表述`。

dataoid

data object id，行所在的Segment的ID，该值可从USER_OBJECTS等视图中的DATA_OBJECT_ID字段查询获得。

spaceid

space id，行所在的表空间的ID，该值可从V$TABLESPACE等视图中的ID字段查询获得。

fileid

file id，行所在数据文件在对应表空间中的数据文件ID，该值可从V$DATAFILE等视图中的ID字段查询获得。

blockid

block id，行所在数据块在对应文件中的块ID。

dir

dir，行在数据块上的槽位。

ROWID使用规则

ROWID可以作为伪列字段来使用，但必须被显式指定，即通过SELECT *语句无法查询到ROWID的值。

ROWID也可以作为字段的数据类型，用来存储ROWID类型数据，且支持对其进行增删改查等DML操作。

在DML操作中，YashanDB支持ROWID与UROWID/RAW/字符型的隐式数据类型转换。但将ROWID作为函数的参数，且与函数要求的参数数据类型不一致时，函数不会对其进行数据类型的隐式转换，而是提示错误返回。

对于ROWID类型数据的使用限制如下：

- 不能作为分区键
- 仅适用于HEAP表

对于ROWID伪列字段的使用限制如下：

- 不能在包含DISTINCT\GROUPBY\CONNECTBY\JOIN算子、ROWNUM伪列字段或聚合函数的子查询语句中使用ROWID伪列。

示例（HEAP表）

```
-- 查询ROWID伪列
SELECT ROWID,* FROM area WHERE area_no='01';
ROWID              AREA_NO AREA_NAME    DHQ
------------------ ------- ------------ ----------
1350:5:0:148:0     01      华东         Shanghai

SELECT LENGTH(ROWID) FROM area WHERE area_no='01';
        LENGTH(ROWID)
---------------------
                   14

SELECT ISNULL(ROWID) FROM area WHERE area_no='01';
[1:7]YAS-04401 data type STRING expected, but ROWID got

SELECT ROWID FROM (SELECT area_no FROM area GROUP BY area_no);
YAS-04827 cannot select ROWID from, or sample, a view with DISTINCT, GROUP BY, etc.

-- 创建ROWID类型的列
CREATE TABLE ROWID_BASE(ID INT);

BEGIN
FOR I IN 1..10 LOOP
  INSERT INTO ROWID_BASE VALUES(I);
END LOOP;
COMMIT;
END;
/

CREATE TABLE ROWID_TEST(C1 ROWID);
INSERT INTO ROWID_TEST SELECT ROWID FROM ROWID_BASE;

--以字符串格式插入ROWID值
INSERT INTO ROWID_TEST VALUES ('1350:5:0:148:0');
```

UROWID使用规则

YashanDB内部将UROWID等同于RAW类型，因此UROWID使用规则与RAW使用规则完全一致。

示例（HEAP表）

```
-- 将上例中ROWID_TEST表的ROWID插入到UROWID_TABLE表中的C1列，C1列类型为UROWID
CREATE TABLE UROWID_TABLE(C1 UROWID);
INSERT INTO UROWID_TABLE SELECT ROWID FROM ROWID_TEST;

--以字符串格式插入UROWID值，只要满足十六进制表述均会成功
INSERT INTO UROWID_TABLE VALUES ('1234ABCD');
--将其转为ROWID时，必须满足ROWID的数据格式要求，否则转换失败
INSERT INTO ROWID_TEST
SELECT c1 FROM UROWID_TABLE
WHERE c1 = '1234ABCD';
YAS-00008 type convert error : invalid ROWID

SELECT ISNULL(c1) FROM UROWID_TABLE
```

```
WHERE c1='01710700000000000006100000900050' AND ROWNUM=1;
ISNULL(C1)
--------------------
false
```

数据类型转换

数据类型转换多发生在操作的输入类型与规定类型不一致，或一个表达式中包含多个不同数据类型的情况中，此时，可以通过隐式转换或显式转换来整合数据类型。

隐式转换（Implicit Data Conversion）

隐式转换表示在原始数据类型和目标数据类型不一致，且用户未指定类型转换函数时，YashanDB会运用一套转换规则来尝试转换数据类型，帮助用户降低SQL语句的编写难度，但如果超出了默认的转换范围，相应的SQL语句会报错。

以下场景中YashanDB默认进行隐式转换：

- INSERT/UPDATE语句中，要插入的数据将被转换成目标列的类型。
- 算术运算时，对参与运算的数据类型进行转换。
- 比较运算中，参与运算的数据类型不一致时，进行转换。
- 当向函数输入的参数为与定义不相符的数据类型时，该参数将先被隐式转换为可以接受的数据类型。
- PL或接口程序中，Filter Condition使用绑定参数时，当输入的参数为与绑定列不相符的数据类型时，该参数将先被隐式转换为可以接受的数据类型。（绑定列为数值型时，输入参数将被执行向NUMBER类型的隐式转换）

执行隐式转换可能导致如下问题：

- FLOAT/DOUBLE类型本身是不精确的数据类型，执行与它们之间的类型可能会导致精度差异。
- 从TIMESTAMP转换为DATE，微秒会被舍弃。

下列表列示YashanDB支持的所有隐式转换：（列表示原始类型；行表示目标类型；✓表示支持转换，X表示不支持转换，--表示无需转换）

（1）

	TINYINT	SMALLINT	INT	BIGINT	NUMBER	FLOAT	DOUBLE	字符型
TINYINT	--	✓	✓	✓	✓	✓	✓	✓
SMALLINT	✓	--	✓	✓	✓	✓	✓	✓
INT	✓	✓	--	✓	✓	✓	✓	✓
BIGINT	✓	✓	✓	--	✓	✓	✓	✓
NUMBER	✓	✓	✓	✓	--	✓	✓	✓
FLOAT	✓	✓	✓	✓	✓	--	✓	✓
DOUBLE	✓	✓	✓	✓	✓	✓	--	✓
字符型	✓	✓	✓	✓	✓	✓	✓	--
DATE	X	X	X	X	X	X	X	✓
TIMESTAMP	X	X	X	X	X	X	X	✓
YM_INTERVAL	X	X	X	X	X	X	X	✓
DS_INTERVAL	X	X	X	X	X	X	X	✓
TIME	X	X	X	X	X	X	X	✓
BOOLEAN	✓	✓	✓	✓	✓	X	X	✓①
BIT	✓	✓	✓	✓	✓	X	X	✓
CLOB	X	X	X	X	X	X	X	✓
BLOB	X	X	X	X	X	X	X	✓
NCLOB	X	X	X	X	X	X	X	✓

	TINYINT	SMALLINT	INT	BIGINT	NUMBER	FLOAT	DOUBLE	字符型
XMLTYPE	X	X	X	X	X	X	X	✓②
RAW	X	X	X	X	X	X	X	✓
JSON	X	X	X	X	X	X	X	✓
ROWID	X	X	X	X	X	X	X	✓
UROWID	X	X	X	X	X	X	X	✓

（2）

	DATE	TIMESTAMP	YM_INTERVAL	DS_INTERVAL	TIME	BOOLEAN
TINYINT	X	X	X	X	X	✓
SMALLINT	X	X	X	X	X	✓
INT	X	X	X	X	X	✓
BIGINT	X	X	X	X	X	✓
NUMBER	X	X	X	X	X	X
FLOAT	X	X	X	X	X	X
DOUBLE	X	X	X	X	X	X
字符型	✓	✓	✓	✓	✓	✓①
DATE	--	✓	X	X	X	X
TIMESTAMP	✓	--	X	X	X	X
YM_INTERVAL	X	X	--	X	X	X
DS_INTERVAL	X	X	X	--	✓	X
TIME	X	X	X	✓	--	X
BOOLEAN	X	X	X	X	X	--
BIT	X	X	X	X	X	✓
CLOB	X	X	X	X	X	X
BLOB	X	X	X	X	X	X
NCLOB	X	X	X	X	X	X
XMLTYPE	X	X	X	X	X	X
RAW	X	X	X	X	X	X
JSON	X	X	X	X	X	X
ROWID	X	X	X	X	X	X
UROWID	X	X	X	X	X	X

（3）

	BIT	CLOB	BLOB	NCLOB	XMLTYPE	RAW	JSON	ROWID	UROWID
TINYINT	✓	✓	X	✓	X	X	X	X	X

	BIT	CLOB	BLOB	NCLOB	XMLTYPE	RAW	JSON	ROWID	UROWID
SMALLINT	✓	✓	X	✓	X	X	X	X	X
INT	✓	✓	X	✓	X	X	X	X	X
BIGINT	✓	✓	X	✓	X	X	X	X	X
NUMBER	✓	✓	X	✓	X	X	X	X	X
FLOAT	X	✓	X	✓	X	X	X	X	X
DOUBLE	X	✓	X	✓	X	X	X	X	X
字符型	✓	✓	✓	✓	✓②	✓	✓	✓	✓
DATE	X	X	X	X	X	X	X	X	X
TIMESTAMP	✓	X	X	X	X	X	X	X	X
YM_INTERVAL	X	X	X	X	X	X	X	X	X
DS_INTERVAL	X	X	X	X	X	X	X	X	X
TIME	X	X	X	X	X	X	X	X	X
BOOLEAN	✓	X	X	X	X	X	X	X	X
BIT	--	X	X	X	X	X	X	X	X
CLOB	X	--	X	X	✓	X	✓	X	X
BLOB	X	X	--	X	X	✓	✓	X	✓
NCLOB	X	✓	X	--	X	X	✓	X	X
XMLTYPE	X	X	X	X	--	X	X	X	X
RAW	X	✓	✓	X	X	--	✓	✓	✓
JSON	X	✓	✓	✓	X	✓	--	X	X
ROWID	X	X	X	X	X	✓	X	--	✓
UROWID	X	✓	✓	X	X	✓	X	✓	--

- ① 仅支持特定字符串转换为BOOLEAN类型，具体请查阅布尔型。
- ② XMLTYPE类型仅支持与字符型中的CHAR和VARCHAR类型进行相互转换。

显式转换（Explicit Data Conversion）

显式转换是通过类型转换函数来清楚直言地指定转换的方向。相比隐式转换，显式转换可以使SQL语句更加容易理解，输出类型的可预测性更强。

下表举例列示YashanDB内置的一些类型转换函数：

函数	功能
BIN	其他类型向BIT类型转换。
NUMTODSINTERVAL	数值型向INTERVAL DAY TO SECOND类型转换。
NUMTOYMINTERVAL	数值型向INTERVAL YEAR TO MONTH类型转换。
SCN_TO_TIMESTAMP	BIGINT类型向TIMESTAMP类型转换。
TIMESTAMP_TO_SCN	TIMESTAMP类型向BIGINT类型转换。
TO_CHAR	其他类型向VARCHAR类型转换。

函数	功能
TO_NUMBER	其他类型向NUMBER类型转换。
TO_DATE	其他类型向DATE类型转换。
TO_TIMESTAMP	其他类型向TIMESTAMP类型转换。
TO_YMINTERVAL	其他类型向INTERVAL YEAR TO MONTH类型转换。
TO_DSINTERVAL	其他类型向INTERVAL DAY TO SECOND类型转换。
JSON	字符串类型向JSON类型转换。
JSON_SERIALIZE	JSON类型向字符串类型转换。
CAST	指定向任意类型转换。

用户自定义类型

YashanDB支持通过PL语法创建一个UDT（自定义类型）对象，成功创建的UDT对象可以作为数据类型被列字段进行声明。

用户自定义类型只能在HEAP表上使用，不参与普通标量数据类型之间的转换和运算，且存在如下限制：

- 不能将某个列字段的数据类型修改为UDT，或者将UDT修改为其他类型。
- UDT字段不能作为分区列、索引列、外键列。
- 不能通过直接指定字段名的方式查询，而应该用本文所描述的方式。
- 为一个分区表定义了非分区的Nested Table列时，不允许drop/truncate该表的分区。
- 包含Nested Table列的表回收站无效，即无法闪回。

Object类型

Object为面向对象概念的抽象数据类型，包含属性和方法，YashanDB支持将一个Object表示的对象结构数据作为列字段存储在物理表中。

在定义一个未包含Nested Table类型属性的Object类型时：

- Object类型的声明方法与普通标量数据类型相同，且可通过对象初始化方法插入或更新数据。
- Object类型数据通过表别名.列名.属性名的方式查询。

在定义一个包含Nested Table类型属性的Object类型时：

- 按Nested Table类型所描述的方法进行声明和使用。

示例（HEAP表）

```
DROP TABLE IF EXISTS city;
--创建一个Object UDT，包含population和province两个属性
CREATE OR REPLACE TYPE obj_city AS OBJECT(population INT, province VARCHAR(20));
/

--创建city表并插入数据
CREATE TABLE city (id INT, info obj_city);
INSERT INTO city VALUES(1, obj_city(2000, 'guangdong'));
INSERT INTO city VALUES(2, obj_city(2500, 'guangdong'));

--更新数据
UPDATE city SET info = obj_city(2500, 'shanghai') WHERE id=2;
COMMIT;

--通过表别名.列名.属性名的方式查询数据
SELECT id, c.info.population, c.info.province FROM city c;
          ID INFO.POPULATION INFO.PROVINCE
------------ --------------- --------------------
           1            2000 guangdong
           2            2500 shanghai
```

Varray类型

Varray为一个包含有序元素的数组集合，YashanDB支持将一个Varray表示的数组结构数据作为列字段存储在物理表中。

Varray类型的声明方法与普通标量数据类型相同，且可通过对象初始化方法插入或更新数据。

Varray类型数据可通过如下方式查询：

- 数组函数，例如ARRAY_TO_STRING。
- table()方法，语法详见SELECT语句中的table_collection_clause描述。

示例（HEAP表）

```
DROP TABLE IF EXISTS city;
--创建一个Object UDT
```

```
CREATE OR REPLACE TYPE arr_type AS VARRAY(5) OF CHAR(10);
/

--创建city表并插入数据
CREATE TABLE city (id INT, info arr_type);
INSERT INTO city VALUES(1, arr_type('2000', 'guangdong'));
INSERT INTO city VALUES(2, arr_type('2500', 'guangdong'));

--更新数据
UPDATE city SET info = arr_type('2500', 'shanghai') WHERE id=2;
COMMIT;

--通过数组函数访问数据
SELECT id, ARRAY_TO_STRING(info,',') infos FROM city;
          ID INFOS
------------ ----------------------------------------------------------------
           1 2000,guangdong
           2 2500,shanghai

--通过table()方法访问数据
SELECT /*+ LEADING(c,arr) */ id, arr.*
FROM city c, TABLE(c.info) arr;
          ID COLUMN_VALUE
------------ -------------
           1 2000
           1 guangdong
           2 2500
           2 shanghai
```

Nested Table类型

Nested Table为一个与表一样包含多行元素的集合，YashanDB支持将一个Nested Table表示的表结构数据作为列字段存储在物理表中。

Nested Table的数据并不是直接存储在所声明的表中，而是需要新建一个嵌套表存储。因此，定义一个Nested Table类型的列字段，或者一个包含了Nested Table类型属性的列字段时，需要同时指定该Nested Table数据存储的嵌套表信息用于新建。

新建的嵌套表依附于列字段所在主表，不可以单独对其执行DDL/DML操作，随着主表（或主表分区）的drop/truncate而drop/truncate。

通过DBA_NESTED_TABLES/USER_NESTED_TABLES/ALL_NESTED_TABLES可查询嵌套表相关信息。

Nested Table类型的声明方法见CREATE TABLE中nested_table_clause子句描述。

可通过对象初始化方法插入或更新Nested Table数据。

对Nested Table数据的查询需使用table()方法，语法详见SELECT语句中的table_collection_clause描述。

示例（HEAP表）

```
DROP TABLE IF EXISTS city;
--创建一个Nested Table UDT
CREATE OR REPLACE TYPE user_table_type IS TABLE OF CHAR(10);
/

--创建包含嵌套表的主表
CREATE TABLE city (id INT, info user_table_type) NESTED TABLE info STORE AS nt_city;

--插入数据
INSERT INTO city VALUES (1, user_table_type('2000','2500'));
INSERT INTO city VALUES (2, user_table_type('guangdong','guangdong'));

--修改数据
UPDATE city SET info = user_table_type('guangdong','shanghai')
WHERE id = 2;
COMMIT;

--查询数据
SELECT /*+ LEADING(c) */ id, tab.*
FROM city c, TABLE(c.info) tab;
```

```
          ID COLUMN_VALUE
------------ -------------
           1 2000
           1 2500
           2 guangdong
           2 shanghai

DROP TABLE IF EXISTS provs;
--创建包含Nested Table属性的Object UDT
CREATE OR REPLACE TYPE obj_type_nested IS OBJECT(name CHAR(10), citys user_table_type);
/
--创建包含Object UDT字段的表
CREATE TABLE provs(id INT, province obj_type_nested)
NESTED TABLE province.citys STORE AS nt_province;

--插入数据
INSERT INTO provs VALUES(1,
                      obj_type_nested('guangdong', user_table_type('shenzhen','guangzhou')));

--查询数据
SELECT /*+ LEADING(p) */ p.id, p.province.name, c2.*
FROM provs p,
TABLE(p.province.citys) c2;
          ID PROVINCE.NAME COLUMN_VALUE
------------ ------------- -------------
           1 guangdong     shenzhen
           1 guangdong     guangzhou
```

ST_GEOMETRY

YashanDB中的ST_GEOMETRY类型是数据库内置的一种自定义类型，用于存储和访问符合开放地理空间信息联盟（Open Geospatial Consortium，简称OGC）制定的SFA SQL标准的几何对象。

此数据类型仅支持在单机HEAP表上使用。

存储属性

类型	字节长度
ST_GEOMETRY	4GB

定义格式

类型	格式	规则
ST_GEOMETRY	ST_GEOMETRY	自定义对象类型，无需指定size

ST_GEOEMTRY属性

属性名称	类型	长度
HEAD	RAW	40字节
GEOM	BLOB	< 4G

ST_GEOMETRY支持的子类型

- POINT

  点代表坐标系空间中的单个坐标。

  ```
  POINT (1 2)
  POINT Z (1 2 3)
  ```

- LINESTRING

  线是由连续的线段组成，且一条线段由两点定义，点之间需要使用逗号隔开。

  ```
  LINESTRING (1 2,4 5)
  ```

- POLYGON

  多边形是由外环（闭合的LINESTRING）及0条或多条内环（闭环的LINESTRING）组成，外环与内环之间需要使用逗号隔开。

  ```
  POLYGON ((1 0,1 1,2 2,1 0),(0 0,6 6,8 8,0 0))
  ```

- MULTIPOINT

  表示点的集合，点之间需要使用逗号隔开。

  ```
  MULTIPOINT ((1 1),(2 2))
  ```

- MULTILINESTRING

  表示线的集合，线之间需要使用逗号隔开。

```
MULTILINESTRING ((1 2,4 5),(2 3,5 6))
```

- MULTIPOLYGON

  表示多边形的集合，多边形之间需要使用逗号隔开。

```
MULTIPOLYGON (((1 5, 4 3, 6 6, 2 6, 1 5)), ((6 5, 8 8, 6 9, 6 5)))
```

- GEOMETRYCOLLECTION

  GEOMETRYCOLLECTION可由不同的ST_GEOMETRY子类型组成一个集合，且子类型之间需要使用逗号隔开。

```
GEOMETRYCOLLECTION (POINT (1 0),LINESTRING (1 2,4 5))
```

使用规则

使用限制

对于ST_GEOMETRY类型数据的使用限制如下：

- 不能作为分区表的分区键。
- 仅适用于单机HEAP表。
- 在ST_GEOMETRY类型上只能创建RTREE索引，不能创建其他类型索引。

使用格式

ST_GEOMETRY类型可通过符合OGC规范的交换格式（如WKT、WKB）输入，也可以将其转换成WKT、WKB等交换格式。

WKT

WKT（Well-Known Text）格式为描述空间数据的标准格式，格式样例如下：

```
POINT(2 5)
LINESTRING((1 3),(4 5))
POLYGON ((1 0,1 1,2 2,1 0),(0 0,6 6,8 8,0 0))
```

WKB

WKB（Well-Known Binary）格式为描述空间数据的二进制格式，本格式与对应的WKT格式如下：

```
WKT:POINT(5 0)
WKB:0101000000000000000000144000000000000000

WKT:LINESTRING (1 2,4 5)
WKB:0102000000020000000000000000F03F0000000000000040000000000000010400000000000001440
```

示例（单机HEAP表）

```
--1.创建包含ST_GEOMETRY类型的表position
CREATE TABLE position (id INT, pos ST_GEOMETRY);

--2.通过WKT格式插入ST_GEOEMTRY对象
INSERT INTO position
VALUES (1, ST_GEOMFROMTEXT('POINT(0 0)'));
COMMIT;

--3.通过WKB格式插入ST_GEOEMTRY对象
INSERT INTO position
```

```
VALUES (2, ST_GEOMFROMWKB('0101000000000000000000F03F000000000000F03F'));
COMMIT;

--4.以WKT格式查询ST_GEOEMTRY对象
SELECT id, ST_ASTEXT(pos) FROM position;

ID           ST_ASTEXT(POS)
------------ --------------------------------------------------------------------
           1 POINT (0.000000000000000 0.000000000000000)
           2 POINT (1.000000000000000 1.000000000000000)
```

ST_GEOMETRY的特性

坐标

每个坐标都有一个X和Y坐标值，以确定其在平面中的位置。形状由点或线段构造，其中点由单个坐标指定，线段由两个坐标指定。

坐标可以包含可选的Z和M坐标值。Z常被用来表示高度。M是一个测量值，可以表示时间或距离。如果Z或M值出现在ST_GEOMETRY值中，则必须为ST_GEOMETRY值的每个点定义它们。如果ST_GEOMETRY类型数据有Z或M坐标，则坐标维度为3D；如果它既有Z又有M，那么坐标维度就是4D。

坐标参考系

对ST_GEOMETRY类型数据的所有空间操作都使用该数据所在的坐标参考系的单位。坐标参考系由SRID编号标识。X轴和Y轴的单位由坐标参照系确定。在平面参考系统中，X和Y坐标通常代表东方和北方，而在大地测量系统中，它们代表经度和纬度。SRID=0表示一个无限的笛卡尔平面，它的轴上没有单位。

维度

维度是ST_GEOMETRY类型的属性。点类型的维度为0，线性类型的维度为1，多边形类型的维度为2。集合的维度为最大元素维度。

YashanDB目前最高仅支持到三维坐标，如输入四维坐标则会忽略第四个坐标。

外包框

包含ST_GEOMETRY坐标的二维或三维方框，它是表示ST_GEOMETRY类型数据在坐标空间中的范围以及检验两个ST_GEOMETRY类型数据是否相互作用的一种有效方法。

BOX2D

YashanDB中的BOX2D类型是数据库内置的一种自定义类型，用于存储和访问ST_GEOMETRY的二维边界框。

此数据类型仅支持在单机HEAP表上使用。

存储属性

类型	字节长度
BOX2D	32

定义格式

类型	格式	规则
BOX2D	BOX2D	自定义对象类型，无需指定size

BOX2D属性

属性名称	类型	长度
XMIN	DOUBLE	8字节
XMAX	DOUBLE	8字节
YMIN	DOUBLE	8字节
YMAX	DOUBLE	8字节

使用规则

使用限制

对于BOX2D类型数据的使用限制如下：

- 不能作为分区表的分区键。
- 仅适用于单机HEAP表。

使用格式

BOX2D类型可通过OBJECT方法或者ST_EXTENT函数生成，并通过 `表别名.列名.属性名` 的方式进行查询。

示例（单机HEAP表）

```
-- 1.创建包含BOX2D类型的表box_table
CREATE TABLE box_table (id INT, box BOX2D);

-- 2.向表中插入BOX2D对象
INSERT INTO box_table VALUES(1, BOX2D(1,2,3,4));

-- 3.查询BOX2D数据
SELECT TB.box.xmin XMIN FROM box_table TB;

XMIN
-----------
1.0E+000
```

运算符

YashanDB提供了不同类型的运算符，如算术运算符、比较运算符等。本章节将对这些运算符的含义及规则进行介绍。

操作数

操作数（operand）规定了一条SQL运算指令中进行运算的量，例如：

- 一元：只有一个操作数的运算，一元操作数的运算符一般在其前面，例如取负（-）运算。
- 二元：在运算符的左右两边各一个操作数，例如1+2。
- 三元：操作符需要三个操作数来执行运算，例如A BETWEEN B AND C。

运算优先级

当同一个SQL表达式中出现多个类型的运算符时，运算符优先级由高到低如下表所示， 同一行列出的运算符具有相同的优先级：

优先级	运算符	操作
1	+, -	正，负
2	^	异或
3	*, /, %	乘，除，取模
4	+, -, ||	加，减，拼接
5	&, |	位与，位或

可以使用圆括号()调整运算优先级。

算术运算符

YashanDB提供如下算术运算符：

运算符	操作数	含义	NULL参与运算
+	一元/二元	一元表示正数，二元表示加法	结果为NULL
-	一元/二元	一元表示负数，二元表示减法	结果为NULL
*	二元	乘法	结果为NULL
/	二元	普通除法	结果为NULL
%	二元	整数除法，返回余数	结果为NULL

除法

YashanDB中，可实现除法运算的方式有：

- 算术运算符：/、%
- 内置函数：MOD、DIV

%

取模运算，与MOD函数同义。

格式为：`n2 % n1`

含义为：将n2 整除n1后剩余的数值作为取模的结果，即余数，余数的正负符号与n2一致。

若n1为0，不报错，而是直接将n2的值作为余数结果返回。

/

普通除法运算。

/与DIV的关系：

- 对于小数（FLOAT/DOUBLE/NUMBER），/与DIV算法一致。
- 对于整数，/作普通除法运算，DIV作整除运算并返回商数。

一般情况下，在/或DIV运算里0不能作为除数，否则报错，但是在被除数是浮点数的情况下，不报错而是做如下特殊处理：

- 如果被除数是Nan，计算结果为Nan
- 如果被除数是正浮点数或Inf，计算结果为Inf
- 如果被除数是负浮点数或-Inf，计算结果为-Inf

运算优先级

从高到低的运算优先级为：+（取正）、-（取负）>*、/、%>+、-。同一优先级运算符从左向右执行。

可以使用双括号（）来调整想达到的运算优先级。

数据类型

如下数据类型可能会参与到算术运算中：

- 数值型（除BIT外）
- 字符型
- 日期时间型

类型转换

在进行二元算术运算时，YashanDB将通过隐式数据转换，将参与运算的数据类型统一到某个数据类型，统一原则为：

- 数值型加、减、乘法最小提升规则：按TINYINT->SMALLINT->INT->BIGINT->NUMBER->FLOAT->DOUBLE顺序向后统一。
- 数值型除法最小提升规则：按TINYINT/SMALLINT/INT/BIGINT->NUMBER->FLOAT->DOUBLE顺序向后统一。
- 字符型数据与数值型数据进行运算时，将会向数值型统一。
- 数值型、字符型数据与日期时间型数据进行运算时，将会向日期时间型统一。

（1）整型数值之间统一规则

行列标题格为参与算术运算的数据类型；内容单元格为统一后的数据类型；–表示不支持两种数据类型参与算术运算。

数据类型	运算符or函数	TINYINT	SMALLINT	INT	BIGINT*
TINYINT	+、-、*	SMALLINT	INT	BIGINT	BIGINT/NUMBER
	/	NUMBER	NUMBER	NUMBER	NUMBER
	%、MOD、DIV	SMALLINT	INT	BIGINT	BIGINT/NUMBER
SMALLINT	+、-、*	INT	INT	BIGINT	BIGINT/NUMBER
	/	NUMBER	NUMBER	NUMBER	NUMBER
	%、MOD、DIV	INT	INT	BIGINT	BIGINT/NUMBER
INT	+、-、*	BIGINT	BIGINT	BIGINT	BIGINT/NUMBER
	/	NUMBER	NUMBER	NUMBER	NUMBER
	%、MOD、DIV	BIGINT	BIGINT	BIGINT	BIGINT/NUMBER
BIGINT*	+、-、*	BIGINT/NUMBER	BIGINT/NUMBER	BIGINT/NUMBER	BIGINT/NUMBER
	/	NUMBER	NUMBER	NUMBER	NUMBER
	%、MOD、DIV	BIGINT/NUMBER	BIGINT/NUMBER	BIGINT/NUMBER	BIGINT/NUMBER

> **Note**：
> BIGINT类型与所有整数类型参与+、-、*、%、MOD、DIV算术运算，或者对BIGINT类型作取负（-）运算时：
> 运算结果转换为BIGINT或NUMBER由系统内部决定，YashanDB默认统一转换为BIGINT，如因业务需要转换为NUMBER，请联系我们的技术支持处理。

（2）整型数值与其他数值之间统一规则

行列标题格为参与算术运算的数据类型；内容单元格为统一后的数据类型，其中（1）表示采用上面（1）整型数值之间统一规则；–表示不支持两种数据类型参与算术运算。

数据类型	运算符or函数	整型数值	NUMBER	FLOAT	DOUBLE
整型数值	+、-、*	（1）	NUMEBR	FLOAT	DOUBLE
	/	（1）	NUMEBR	FLOAT	DOUBLE
	%、MOD、DIV	（1）	NUMEBR	FLOAT	DOUBLE
NUMBER	+、-、*	NUMEBR	NUMBER	FLOAT	DOUBLE
	/	NUMEBR	NUMEBR	FLOAT	DOUBLE
	%、MOD、DIV	NUMEBR	NUMEBR	FLOAT	DOUBLE
FLOAT	+、-、*	FLOAT	FLOAT	FLOAT	DOUBLE

数据类型	运算符or函数	整型数值	NUMBER	FLOAT	DOUBLE
	/	FLOAT	FLOAT	FLOAT	DOUBLE
	%、MOD、DIV	FLOAT	FLOAT	FLOAT	DOUBLE
DOUBLE	+、-、*	DOUBLE	DOUBLE	DOUBLE	DOUBLE
	/	DOUBLE	DOUBLE	DOUBLE	DOUBLE
	%、MOD、DIV	DOUBLE	DOUBLE	DOUBLE	DOUBLE

（3）数值型与字符型统一规则

行列标题格为参与算术运算的数据类型；内容单元格为统一后的数据类型，其中（2）表示采用上面（2）整型数值与其他数值之间统一规则；–表示不支持两种数据类型参与算术运算。

数据类型	运算符or函数	整型数值	NUMBER	FLOAT	DOUBLE	字符型
整型数值	+、-、*、/、%、MOD、DIV	（2）	（2）	（2）	（2）	NUMEBR
NUMBER	+、-、*、/、%、MOD、DIV	（2）	（2）	（2）	（2）	NUMBER
FLOAT	+、-、*、/、%、MOD、DIV	（2）	（2）	（2）	（2）	FLOAT
DOUBLE	+、-、*、/、%、MOD、DIV	（2）	（2）	（2）	（2）	DOUBLE
字符型	+、-、*、/、%、MOD、DIV	NUMBER	NUMEBR	FLOAT	DOUBLE	NUMEBR

（4）数值型、字符型与日期时间型统一规则

行列标题格为参与算术运算的数据类型；内容单元格为统一后的数据类型，其中（3）表示采用上面（3）数值型与字符型统一规则；–表示不支持两种数据类型参与算术运算；YM表示INTERVAL YEAR TO MONTH，DS表示INTERVAL DAY TO SECOND，TS表示TIMESTAMP。

数据类型	运算符	数值型	字符型	TIME	DATE	TS	YM	DS
数值型	+	（3）	（3）	--	DATE	DATE	--	--
	-	（3）	（3）	--	--	--	--	--
	*	（3）	（3）	--	--	--	YM	DS
	/	（3）	（3）	--	--	--	--	--
字符型	+、-	（3）	（3）	--	DATE	DATE	--	--
	*、/	（3）	（3）	--	--	--	YM	DS
TIME	+	--	--	--	TS	TS	--	TIME
	-	--	--	DS	--	--	--	TIME
DATE	+	DATE	DATE	TS	--	--	DATE	DATE
	-	DATE	DATE	TS	NUMBER	DS	DATE	DATE
TS	+	DATE	DATE	TS	--	--	TS	TS
	-	DATE	DATE	TS	DS	DS	TS	TS
YM	+	--	--	--	DATE	TS	YM	--
	-	--	--	--	--	--	YM	--
	*	YM	YM	--	--	--	--	
	/	YM	YM	--	--	--	--	

数据类型	运算符	数值型	字符型	TIME	DATE	TS	YM	DS
DS	+	--	--	TIME	DATE	TS	--	DS
	-	--	--	--	--	--	--	DS
	*	DS	DS	--	--	--	--	--
	/	DS	DS	--	--	--	--	--

示例

```
--字符型与数值型运算时，将字符型转换为NUMBER类型后参与计算
SELECT '44'/2 char_cal,
TYPEOF('44'/2) type_trans
FROM dual;
   CHAR_CAL TYPE_TRANS
---------- ----------
         22 number

--INT类型与FLOAT类型计算时，将INT类型转换为FLOAT类型参与计算
SELECT 2*CAST(2 AS FLOAT) int_cal,
TYPEOF(2*CAST(2 AS FLOAT)) type_trans
FROM dual;
    INT_CAL TYPE_TRANS
---------- ----------------------------------------------------------------
   4.0E+000 float
```

日期时间型算术运算

日期时间型数据可参与如下算术运算：

运算符	操作数	返回类型	运算规则
+	二元	* 时间	* 将两个间隔类型相加，返回更长的间隔类型。 * 将DATE、TIMESTAMP与TIME类型相加，返回在时间轴上更靠后的TIMESTAMP。 * 将TIME、DATE、TIMESTAMP与间隔类型相加，返回在时间轴上更靠后的TIME、DATE、TIMESTAMP。 * 将DATE、TIMESTAMP与数值相加，数值被解释为天数（小数部分表示不满一天的时间），返回在时间轴上更靠后的DATE。
-	二元	* 时间间隔 * 时间	* 将两个间隔类型相减，返回更短的间隔类型。 * 将DATE、TIMESTAMP与TIME相减，返回在时间轴上更靠前的TIMESTAMP。 * 将两个DATE相减，返回数值，表示相差的天数。 * 将DATE与TIMESTAMP相减，或TIMESTAMP与TIMESTAMP相减，返回DS_INTERVAL。 * 将DATE或TIMESTAMP与数值相减，数值被解释为天数（小数部分表示不满一天的时间），返回在时间轴上更靠前的DATE。
*	二元	* 时间间隔 * 时间	* 将间隔类型与数值相乘，返回被扩大倍数的间隔。
/	二元	* 时间间隔 * 时间	* 将间隔类型除以数值，返回被缩小倍数的间隔。

如下是时间类型和时间间隔类型支持运算的详细情况，其中YM表示INTERVAL YEAR TO MONTH，DS表示INTERVAL DAY TO SECOND，TS表示TIMESTAMP：

数据类型	NUMBER	TIME	DS	YM	DATE	TS
TIME	N/A	支持运算：- 返回类型：DS	支持运算：+- 返回类型：TIME	N/A	支持运算：+ 返回类型：TS	支持运算：+ 返回类型：TS

数据类型	NUMBER	TIME	DS	YM	DATE	TS
DS	支持运算：*/ 返回类型：DS	支持运算：+ 返回类型：TIME	支持运算：+- 返回类型：DS	N/A	支持运算：+ 返回类型：DATE	支持运算：+ 返回类型：TS
YM	支持运算：*/ 返回类型：YM	N/A	N/A	支持运算：+- 返回类型：YM	支持运算：+ 返回类型：DATE	支持运算：+ 返回类型：TS
DATE	支持运算：+- 返回类型：DATE	支持运算：+- 返回类型：TS	支持运算：+- 返回类型：DATE	支持运算：+- 返回类型：DATE	支持运算：- 返回类型：NUMBER	支持运算：- 返回类型：DS
TS	支持运算：+- 返回类型：DATE	支持运算：+- 返回类型：TS	支持运算：+- 返回类型：TS	支持运算：+- 返回类型：TS	支持运算：- 返回类型：DS	支持运算：- 返回类型：DS

示例

```
CREATE TABLE date_dd(a DATE, b DATE);
INSERT INTO date_dd VALUES('2008-12-31','2018-12-31');
COMMIT;

--运算符 '-'  返回时间间隔
SELECT b-a FROM date_dd;
        B-A
-----------
       3652

--运算符 '-'  返回时间
SELECT b-1 FROM date_dd;
B-1
--------------------------------
2018-12-30 00:00:00

SELECT SYSDATE+2 FROM DUAL;
SYSDATE+2
--------------------------------
2021-06-10 11:07:47

-- DATE和TIEMSTAMP与数值类型做加减运算，数值表示天
CREATE TABLE date_dt(c1 DATE, c2 TIMESTAMP);
INSERT INTO date_dt VALUES ('2020-03-31', '2020-03-31 12:30:59.999999');
COMMIT;

SELECT c1-2.5, c2-2.5 FROM date_dt;
C1-2.5                         C2-2.5
------------------------------ ------------------------------
2020-03-28 12:00:00            2020-03-29 00:30:59

SELECT c1+'2.5', c2+'2.5' FROM date_dt;
C1+'2.5'                       C2+'2.5'
------------------------------ ------------------------------
2020-04-02 12:00:00            2020-04-03 00:30:59

SELECT c1 FROM date_dt;
C1
--------------------------------
2020-03-31 00:00:00


-- 输出结果为TIME类型的运算
CREATE TABLE date_t(c1 TIME);
INSERT INTO date_t VALUES ('23:59:59.999999');
COMMIT;

SELECT c1+INTERVAL '5' HOUR FROM date_t;
C1+INTERVAL'5'HOUR
--------------------
04:59:59.999999
```

输出的日期如果不合法，结果会报错。

示例

```
--对上例data_dt中的字段执行下面的运算，结果为4月31日，该日期不合法
SELECT C1+INTERVAL '01-01' YEAR TO MONTH FROM date_dt;
--行存表输出
YAS-00008 type convert error : not a valid month
--列存表输出
YAS-05012 date not valid for month specified
```

比较运算符

YashanDB提供如下比较运算符：

运算符	操作数	含义	NULL参与运算
=	二元	等于	结果为FALSE
!= 或 <>	二元	不等于	结果为FALSE
>	二元	大于	结果为FALSE
>=	二元	大于等于	结果为FALSE
<	二元	小于	结果为FALSE
<=	二元	小于等于	结果为FALSE
[NOT] IN	二元	左边数据是否[不]在右边数据集合中	IN： * 左边数据为NULL：结果为FALSE * 右边集合包含NULL：如果集合中有非NULL值与左边数据相等，则结果为TRUE，否则为FALSE NOT IN： * 左边数据为NULL：结果为FALSE * 右边集合包含NULL：如果集合中含有NULL值，则返回FALSE；如果集合中不含NULL值，且所有数据与左边数据不相等，则结果为TRUE，否则为FALSE
[NOT] LIKE	二元	左边数据是否与右边数据[不]匹配	结果为FALSE
[NOT] BETWEEN AND	三元	第一个数据是否[不]在第二个数据和第三个数据组成的区间内	结果为FALSE
IS [NOT] NULL	一元	是否[不]等于NULL	IS NULL： * 操作数为NULL：结果为TRUE * 操作数不为NULL：结果为FALSE IS NOT NULL： * 操作数为NULL：结果为FALSE * 操作数不为NULL：结果为TRUE

比较运算符在SQL语法中的详细运用请参考condition条件子句描述。

运算优先级

从高到低的运算优先级为：

- =、!= 或 <>、>、>=、<、<=
- [NOT] IN、IS [NOT] NULL、[NOT] LIKE
- [NOT] BETWEEN AND

可以使用双括号（）来调整想达到的运算优先级。

数据类型

在执行=、!= 或 <>、>、>=、<、<=比较符运算时，如下数据类型可能会参与到比较运算中：

- 数值型
- 字符型
- 日期时间型
- 布尔型
- RAW

类型转换

如果比较符两边的数据类型不一致，YashanDB将通过隐式数据转换，将其中一边的数据类型向另一边统一，统一原则为：

- 数值型最小提升规则：按TINYINT->SMALLINT->INT->BIGINT->NUMBER->FLOAT->DOUBLE顺序向后统一。
- BIT型数据与BIGINT、NUMBER型数据进行运算时，将会向BIGINT/NUMBER型统一。
- 字符型数据与其他类型进行运算时，将会向其他类型统一。
- DATE型数据与TIMESTAMP型数据进行运算时，将会向TIMESTAMP型统一。
- TIME型数据与DS_INTERVAL型数据进行运算时，将会向DS_INTERVAL型统一。
- 布尔型数据与数值型数据进行运算时，将会向数值型统一。

（1）数值型与布尔型统一规则

行列标题格为参与=、!= 或 <>、>、>=、<、<=比较符运算的数据类型；内容单元格为统一后的数据类型；–表示不支持两种数据类型参与比较运算。

数据类型	TINYINT	SMALLINT	INT	BIGINT	BIT	NUMBER	FLOAT	DOUBLE	BOOLEAN
TINYINT	TINYINT	SMALLINT	INT	BIGINT	BIGINT	NUMBER	FLOAT	DOUBLE	TINYINT
SMALLINT	SMALLINT	SMALLINT	INT	BIGINT	BIGINT	NUMBER	FLOAT	DOUBLE	SMALLINT
INT	INT	INT	INT	BIGINT	BIGINT	NUMBER	FLOAT	DOUBLE	INT
BIGINT	BIGINT	BIGINT	BIGINT	BIGINT	BIGINT	NUMBER	FLOAT	DOUBLE	BIGINT
BIT	BIGINT	BIGINT	BIGINT	BIGINT	BIT	NUMBER	--	--	BIGINT
NUMBER	NUMBER	NUMBER	NUMBER	NUMBER	NUMBER	NUMBER	FLOAT	DOUBLE	--
FLOAT	FLOAT	FLOAT	FLOAT	FLOAT	--	FLOAT	FLOAT	DOUBLE	--
DOUBLE	DOUBLE	DOUBLE	DOUBLE	DOUBLE	--	DOUBLE	DOUBLE	DOUBLE	--
BOOLEAN	TINYINT	SMALLINT	INT	BIGINT	BIGINT	--	--	--	BOOLEAN

（2）数值型、布尔型与字符型统一规则

行列标题格为参与=、!= 或 <>、>、>=、<、<=比较符运算的数据类型；内容单元格为统一后的数据类型，其中（1）表示采用上面（1）数值型与布尔型统一规则；–表示不支持两种数据类型参与比较运算。

数据类型	整型数值	BIT	NUMBER	FLOAT	DOUBLE	BOOLEAN	字符型
整型数值	（1）	（1）	（1）	（1）	（1）	（1）	NUMEBR
BIT	（1）	（1）	（1）	（1）	（1）	（1）	BIT
NUMBER	（1）	（1）	（1）	（1）	（1）	（1）	NUMEBR
FLOAT	（1）	（1）	（1）	（1）	（1）	（1）	FLOAT
DOUBLE	（1）	（1）	（1）	（1）	（1）	（1）	DOUBLE
BOOLEAN	（1）	（1）	（1）	（1）	（1）	（1）	BOOLEAN
字符型	NUMEBR	BIT	NUMEBR	FLOAT	DOUBLE	BOOLEAN	字符型*

> **Note**：

字符型与字符型的比较，执行的是大小写敏感的字符串比较，详细描述请参考字符型文档。

（3）数值型、布尔型、字符型与日期时间型统一规则

行列标题格为参与=、!= 或 <>、>、>=、<、<=比较符运算的数据类型；内容单元格为统一后的数据类型，其中（2）表示采用上面（2）数值型、布尔型与字符型统一规则；–表示不支持两种数据类型参与比较运算。

数据类型	数值型	字符型	BOOLEAN	DATE	TIMESTAMP	YM_INTERVAL	DS_INTERVAL	TIME
数值型	（2）	（2）	（2）	--	--	--	--	--
字符型	（2）	（2）	（2）	DATE	TIMESTAMP	YM_INTERVAL	DS_INTERVAL	TIME
BOOLEAN	（2）	（2）	（2）	--	--	--	--	--
DATE	--	DATE	--	DATE	TIMESTAMP	--	--	--
TIMESTAMP	--	TIMESTAMP	--	TIMESTAMP	TIMESTAMP	--	--	--
YM_INTERVAL	--	YM_INTERVAL	--	--	--	YM_INTERVAL	--	--
DS_INTERVAL	--	DS_INTERVAL	--	--	--	--	DS_INTERVAL	DS_INTERVAL
TIME	--	TIME	--	--	--	--	DS_INTERVAL	TIME

示例

```
--NULL参与比较
SELECT SYSDATE FROM DUAL WHERE NULL IN (1,NULL);
SYSDATE
--------------------------------


SELECT SYSDATE FROM DUAL WHERE 1 IN (1,NULL);
SYSDATE
--------------------------------
2021-06-08 10:05:53

SELECT SYSDATE FROM DUAL WHERE 1 IN (2,NULL);
SYSDATE
--------------------------------


--字符型与数值型比较时的隐式转换
SELECT 1 FROM dual
WHERE '0'<4;
           1
------------
           1

 --布尔型与字符型比较时的隐式转换
SELECT 1 FROM dual
WHERE true LIKE 'true%';
           1
------------
           1
```

（4）RAW类型与字符型统一规则

RAW类型只可与自身或字符型进行比较，规则如下：

（行列标题格为参与=、!= 或 <>、>、>=、<、<=比较符运算的数据类型；内容单元格为统一后的数据类型。）

数据类型	RAW	字符型
RAW	RAW	RAW
字符型	RAW	字符型

（4）udt类型比较统一规则

- OBJECT类型默认只支持等于、不等于运算。已创建方法的OBJECT类型支持等于、不等于、大于、大于等于、小于、小于等于的运算，且根据其方法的结果进行比较。
- VARRAY类型不支持等于、不等于、大于、大于等于、小于、小于等于运算。
- TABLE类型只支持等于、不等于运算。
- 当UDT类型嵌套UDT类型时，目前不支持TABLE类型、未创建方法的OBJECT的比较，其他UDT类型嵌套时每一层UDT类型都需遵循比较规则。
- 支持等于、不等于运算的类型，也支持[NOT] IN运算。
- 所有UDT类型不支持[NOT] LIKE运算。
- 仅已创建方法的OBJECT类型支持[NOT] BETWEEN AND。
- 所有UDT类型支持IS [NOT] NULL运算。

逻辑运算符

逻辑运算符要求运算的数据必须为布尔型，否则不执行运算并提示错误。

YashanDB提供如下逻辑运算符：

运算符	操作数	含义	NULL参与运算
AND	二元	双值运算符，如果左右两个条件都为真，则结果为真，否则结果为假。	结果为NULL
OR	二元	双值运算符，只要左右两个条件有一个为真，则结果为真，否则结果为假。	* true OR NULL：结果为true * false OR NULL：结果为NULL * NULL OR NULL：结果为NULL
NOT	一元	单值运算符，如果原条件为真，则得到假，反之如果原条件为假，则结果为真	结果为NULL

从高到低的运算优先级为：NOT>AND>OR，可以使用双括号（）来调整想达到的运算优先级。

示例

```
--创建logics表并插入数据
CREATE TABLE logics (c1 INT, c2 CHAR(1), c3 INT);
INSERT INTO logics VALUES (1,'a',3);
COMMIT;

--WHERE语句里的逻辑计算
SELECT * FROM logics WHERE 1=1 AND 1=0;
          C1 C2              C3
------------ ----- ------------


SELECT * FROM logics WHERE 1=1 OR 1=0;
          C1 C2              C3
------------ ----- ------------
           1 a                3

SELECT * FROM logics WHERE NOT (1<>1);
          C1 C2              C3
------------ ----- ------------
           1 a                3

SELECT SYSDATE FROM dual WHERE NOT 1=2 AND 1=2;
SYSDATE
--------------------------------
```

连接运算符

YashanDB提供如下连接运算符：

运算符	操作数	含义	NULL参与运算
||	二元	字符串连接，支持两个及以上连接	任何数据||NULL：结果为任何数据

YashanDB中，可实现连接运算的方式有：

- 连接运算符：||
- 内置函数：CONCAT

数据类型

除UDT外所有数据类型都可能参与到连接运算。

连接运算符要求运算的数据为字符型，对于其他类型，YashanDB通过隐式转换，将其统一为字符型后，再进行连接运算。

示例

```
SELECT 'abc'||'nnk'||123 FROM DUAL;
'ABC'||'NNK'||123
-----------------
abcnnk123
```

位运算符

YashanDB提供如下位运算符：

运算符	操作数	含义	NULL参与运算
&	二元	按位与：1 & 1 = 1 其他为0	结果为NULL
|	二元	按位或：0 | 0 = 0 其他为1	结果为NULL
^	二元	按位异或：0 ^ 0 = 0 或 1 ^ 1 = 0 其他为1	结果为NULL

YashanDB中，可实现位运算的方式有：

- 位运算符：&、|、^
- 内置函数：BITAND BITOR BITXOR

其中，&与BITAND同义，|与BITOR同义，^与BITXOR同义。

参与运算的两个数据，按二进制展开后，由低到高每bit进行位运算，并以BIGINT类型输出运算结果。

位运算最多支持到64位。

小数参与运算：FLOOR取整后，参与位运算。

运算优先级

从高到低的运算优先级为：^>&>|。

可以使用双括号（）来调整想达到的运算优先级。

数据类型

如下数据类型可能会参与到位运算中：

- 数值型（除FLOAT、DOUBLE外）
- 字符型

在进行二元位运算时，YashanDB将通过隐式数据转换，将参与运算的数据类型全部统一到BIGINT类型，再进行位运算。

下图列示具体统一规则：（行列标题格为参与位运算的数据类型；内容单元格为统一后的数据类型；–表示不支持两种数据类型参与位运算）

数据类型	TINYINT	SMALLINT	INT	BIGINT	NUMBER	字符型	BIT
TINYINT	BIGINT	BIGINT	BIGINT	BIGINT	BIGINT	BIGINT	BIGINT
SMALLINT	BIGINT	BIGINT	BIGINT	BIGINT	BIGINT	BIGINT	BIGINT
INT	BIGINT	BIGINT	BIGINT	BIGINT	BIGINT	BIGINT	BIGINT
BIGINT	BIGINT	BIGINT	BIGINT	BIGINT	BIGINT	BIGINT	BIGINT
NUMBER	BIGINT	BIGINT	BIGINT	BIGINT	BIGINT	BIGINT	BIGINT
字符型	BIGINT	BIGINT	BIGINT	BIGINT	BIGINT	BIGINT	BIGINT
BIT	BIGINT	BIGINT	BIGINT	BIGINT	BIGINT	BIGINT	BIGINT

通用SQL语法

对于一些可能在多个SQL语句的语法定义中出现的子句，在本章节对其进行通用解释和定义，对应的SQL语句中则不再进行说明，而是直接引用该子句的名称。

RegExp

YashanDB的正则表达式实现兼容与Perl相同的语法和语义，此外也支持Unicode正则表达式规范。

> **Note**： “UTF8支持”表示：当数据库服务端字符集为UTF8时，对于除ASCII外的UTF8字符（如中文等）的支持情况。 此外，若数据库服务端采用其他编码格式，其匹配规则都遵循ASCII编码字符匹配规则。 以下规则对于ASCII编码字符均适用。

元字符：

表1-1：元字符（非出现在字符类中）

符号	描述	UTF8支持
\	通用转义字符（有多种用途）	支持
^	字符串（或行，在多行模式下）的开始	支持
$	字符串（或行，在多行模式下）的结束	支持
.	匹配除换行符以外的任何字符（默认）	支持
[	字符类开始标志	支持
]	字符类结束标志	支持
|	备用分支的启动	支持
(	子模式或控件谓词开始标志	支持
)	子模式或控件谓词结束标志	支持
*	出现0次或多次（greedy）	支持
+	出现1次或多次（possessive）	支持
?	出现0次或1次（lazy）	支持
{	重复引用符的开始标志	支持
}	重复引用符的结束标志	支持

表1-2：元字符（出现在字符类中）

符号	描述	UTF8支持
\	通用的转义字符	支持
^	该类取反，但只对第一个字符生效	支持
-	指示字符范围	支持
[	POSIX字符类开始标志（如果后面跟着POSIX语法）	支持
]	POSIX字符类结束标志	支持

匹配方式：

表2-1：单一匹配

符号	描述	UTF8支持

符号	描述	UTF8支持
.	匹配非换行符的任何字符	支持
x	单个字符x	支持
[字符类]	匹配“字符类”中的一个字符，“字符类”见后面的说明	支持
[^字符类]	匹配“字符类”外的一个字符，“字符类”见后面的说明	支持
(xxx)	匹配xxx，将内部表达式标记为子表达式	支持
\小写字母Perl标记	匹配“Perl字符类”中的一个字符，“Perl字符类”见后面的说明	支持
\大写字母Perl标记	匹配“Perl字符类”外的一个字符，“Perl字符类”见后面的说明	支持
\p{xx}	匹配“Unicode类“（普通类+脚本类）中的一个字符，“Unicode类”见后面的说明	支持
\P{xx}	匹配“Unicode类“（普通类+脚本类）外的一个字符，“Unicode类”见后面的说明	支持
\pxx	匹配“Unicode类“（仅普通类）中的一个字符，“Unicode类”见后面的说明	支持
\Pxx	匹配“Unicode类“（仅普通类）外的一个字符，“Unicode类”见后面的说明	支持

表2-2：复合匹配

符号	描述	UTF8支持
xy	匹配xy（x后面跟随y）	支持
x|y	匹配x或y（优先匹配x）	支持

表2-3：重复匹配

符号	描述	UTF8支持
x?	x，零次或一次（greedy）	支持
x?+	x，零次或一次（possessive）	支持
x??	x，零次或一次（lazy）	支持
x*	x，零次或多次（greedy）	支持
x*+	x，零次或多次（possessive）	支持
x*?	x，零次或多次（lazy）	支持
x+	x，一次或多次（greedy）	支持
x++	x，一次或多次（possessive）	支持
x+?	x，一次或多次（lazy）	支持
x{n}	x，恰好n次（greedy）	支持
x{n,m}	x，至少n次，但是不超过m次（greedy）	支持
x{n,m}+	x，至少n次，但是不超过m次（possessive）	支持
x{n,m}?	x，至少n次，但是不超过m次（lazy）	支持
x{n,}	x，至少n次（greedy）	支持
x{n,}+	x，至少n次（possessive）	支持
x{n,}?	x，至少n次（lazy）	支持

匹配位置：

表3-1：匹配位置

指定一个条件必须在一个特定的点匹配，不消耗任何字符的字符串。

符号	描述	UTF8支持
^	匹配字符串开始	--
$	匹配字符串结束	--
\b	匹配单词边界	--
\B	匹配非单词边界	--
\A	匹配字符串开始	--
\Z	匹配字符串结束，也匹配字符串结尾的换行符之前	--
\z	匹配字符串结束	--
\G	匹配字符串中的第一个匹配位置	--

转义字符：

表4-1：转义字符

转义字符	描述	UTF8支持
\a	alarm字符（hex 07）	--
\cx	控制字符	--
\d	反向引用表达式（d是介于1和9之间的阿拉伯数字），匹配出现在'('和')'之间的子表达式	--
\e	escape字符（hex 1B）	--
\f	换页字符（hex 0C）	--
\n	换行字符（hex 0A）	--
\r	回车字符（hex 0D）	--
\t	制表字符（hex 09）	--
\0dd	八进制数dd	--
\ddd	八进制数ddd（或者反向引用的序号）	--
\o{ddd..}	八进制数ddd..	--
\xhh	十六进制数hh	--
\x{hh..}	十六进制数hh..	--
\\	字符\	--
\^	字符^	--
\$	字符$	--
\.	字符.	--
*	字符*	--

转义字符	描述	UTF8支持
\+	字符+	--
\?	字符?	--
\{	字符{	--
\}	字符}	--
\(	字符(	--
\)	字符)	--
\[	字符[	--
\]	字符]	--
|	字符|	--

字符类：

表5-1：字符类汇总

一般字符类的使用方式为[...]，POSIX字符类的使用方式为[[:xxx:]]

符号	描述	UTF8支持
[...]	指定与列表中表示的任何表达式匹配的字符（字符之间是"或"的意思，例如[12]匹配1或2）	支持
[^...]	指定与列表外表示的任何表达式匹配的字符	支持
[x-y]	范围（可用于十六进制字符，包含首尾字符）	支持
[[:xxx:]]	POSIX字符类	--
[[:^xxx:]]	非POSIX字符类	--
[\字母]	Perl字符类	支持
[p{xx}]	Unicode类（普通类+脚本类）	支持
[pxx]	Unicode类（仅普通类）	支持

表5-2：POSIX字符类

POSIX字符类	描述	UTF8支持
alnum	字母数字（相当于[0-9A-Za-z]）	--
alpha	字母（相当于[A-Za-z]）	--
ascii	ASCII字符集（相当于[\x00-\x7F]）	--
blank	空白占位符（相当于[\t]）	--
cntrl	控制字符（相当于[\x00-\x1F\x7F]）	--
digit	数字（相当于[0-9]）	--
graph	图形字符（相当于[!-~]）	--
lower	小写字母（相当于[a-z]）	--
print	可打印字符（相当于[-~]，相当于[[:graph:]]）	--

POSIX字符类	描述	UTF8支持
punct	标点符号（相当于[!-/ `:-@` [-`{-~]）	--
space	空白字符（相当于[\t\n\v\f\r]）	--
upper	大写字母（相当于[A-Z]）	--
word	单词字符（相当于[0-9A-Za-z]）	--
xdigit	16进制字符（相当于[0-9A-Fa-f]）	--

表5-3：Perl字符类

Perl字符类	描述	UTF8支持
\C	一个代码单元，建议尽量避免（即使在UTF-8模式下）	--
\d	十进制数字（相当于[0-9]）	--
\D	非十进制数字	支持
\h	水平空白字符	--
\H	非水平空白字符	支持
\N	非换行符	支持
\s	空格符（相当于[\t\n\f\r]）	--
\S	非空格符	支持
\v	垂直空白字符	--
\V	非垂直空白字符	支持
\w	单词字符（相当于[0-9A-Za-z]）	--
\W	非单词字符	支持
\R	换行符	--
\X	Unicode扩展的字母簇	支持

表5-4：Unicode类（普通类）

\p和\P属性	描述	UTF8支持
C	其他（Other）	支持
Cc	控制字符（Control）	支持
Cf	格式字符（Format）	支持
Cn	未赋值字符（Unassigned）	支持
Co	私人使用区（Private use）	支持
Cs	代理区（Surrogate）	支持
L	字母字符（Letter）	支持
Ll	小写字母（Lower case letter）	支持
Lm	修饰字母（Modifier letter）	支持
Lo	其他字母（Other letter）	支持

\p和\P属性	描述	UTF8支持
Lt	首字母大写字母（Title case letter）	支持
Lu	大写字母（Upper case letter）	支持
L&	Ll、Lu或Lt	支持
M	标记（Mark）	支持
Mc	间距标记（Spacing mark）	支持
Me	嵌入标记（Enclosing mark）	支持
Mn	非间距标记（Non-spacing mark）	支持
N	数字（Number）	支持
Nd	十进制数字（Decimal number）	支持
Nl	字母数字（Letter number）	支持
No	其他数字（Other number）	支持
P	标点（Punctuation）	支持
Pc	连接标点（Connector punctuation）	支持
Pd	破折号（Dash punctuation）	支持
Pe	封闭标点，指括号（Close punctuation）	支持
Pf	最后的标点（Final punctuation）	支持
Pi	最初的标点（Initial punctuation）	支持
Po	其他标点（Other punctuation）	支持
Ps	开放的标点（Open punctuation）	支持
S	符号（Symbol）	支持
Sc	货币符号（Currency symbol）	支持
Sk	修饰符号（Modifier symbol）	支持
Sm	数学符号（Mathematical symbol）	支持
So	其他符号（Other symbol）	支持
Xan	属性L和N的并集	支持
Xps	POSIX space：属性Z或tab、NL、VT、FF、CR	支持
Xsp	Perl space：属性Z或tab、NL、VT、FF、CR	支持
Xuc	统一命名字符：可以由通用字符名称表示	支持
Xwd	Perl word：属性Xan或下划线	支持
Z	分隔符（Separator）	支持
Zl	行分隔符（Line separator）	支持
Zp	段落分隔符（Paragraph separator）	支持
Zs	空白分隔符（Space separator）	支持

表5-5：Unicode类（脚本类）

注：支持UTF8。

Adlam, Ahom, Anatolian_Hieroglyphs, Arabic, Armenian, Avestan, Balinese, Bamum, Bassa_Vah, Batak, Bengali, Bhaiksuki, Bopomofo, Brahmi, Braille, Buginese, Buhid, Canadian_Aboriginal, Carian, Caucasian_Albanian, Chakma, Cham, Cherokee, Chorasmian, Common, Coptic, Cuneiform, Cypriot, Cypro_Minoan, Cyrillic, Deseret, Devanagari, Dives_Akuru, Dogra, Duployan, Egyptian_Hieroglyphs, Elbasan, Elymaic, Ethiopic, Georgian, Glagolitic, Gothic, Grantha, Greek, Gujarati, Gunjala_Gondi, Gurmukhi, Han, Hangul, Hanifi_Rohingya, Hanunoo, Hatran, Hebrew, Hiragana, Imperial_Aramaic, Inherited, Inscriptional_Pahlavi, Inscriptional_Parthian, Javanese, Kaithi, Kannada, Katakana, Kayah_Li, Kharoshthi, Khitan_Small_Script, Khmer, Khojki, Khudawadi, Lao, Latin, Lepcha, Limbu, Linear_A, Linear_B, Lisu, Lycian, Lydian, Mahajani, Makasar, Malayalam, Mandaic, Manichaean, Marchen, Masaram_Gondi, Medefaidrin, Meetei_Mayek, Mende_Kikakui, Meroitic_Cursive, Meroitic_Hieroglyphs, Miao, Modi, Mongolian, Mro, Multani, Myanmar, Nabataean, Nandinagari, New_Tai_Lue, Newa, Nko, Nushu, Nyakeng_Puachue_Hmong, Ogham, Ol_Chiki, Old_Hungarian, Old_Italic, Old_North_Arabian, Old_Permic, Old_Persian, Old_Sogdian, Old_South_Arabian, Old_Turkic, Old_Uyghur, Oriya, Osage, Osmanya, Pahawh_Hmong, Palmyrene, Pau_Cin_Hau, Phags_Pa, Phoenician, Psalter_Pahlavi, Rejang, Runic, Samaritan, Saurashtra, Sharada, Shavian, Siddham, SignWriting, Sinhala, Sogdian, Sora_Sompeng, Soyombo, Sundanese, Syloti_Nagri, Syriac, Tagalog, Tagbanwa, Tai_Le, Tai_Tham, Tai_Viet, Takri, Tamil, Tangsa, Tangut, Telugu, Thaana, Thai, Tibetan, Tifinagh, Tirhuta, Toto, Ugaritic, Vai, Vithkuqi, Wancho, Warang_Citi, Yezidi, Yi, Zanabazar_Square.

匹配控制类：

表6-1：匹配规则控制类

这些匹配控制选项仅生效于单次匹配，匹配结束会自动取消。

这些匹配选项的优先级高于regexp_like、regexp_count、regexp_instr、regexp_substr、regexp_replace函数中match_param表达式设置的相应属性，例如：

- `SELECT regexp_like('aa','(?i)AA','c') FROM DUAL;` 返回TRUE
- `SELECT regexp_like('aa','AA','c') FROM DUAL;` 返回FALSE
- pattern中的 `(?i)` 的优先级高于match_param中的 `c`

同时，匹配控制选项生效于后面模式的其余部分，举例：`(a(?i)b)c`

以上模式匹配abc或aBc，通过这种方式，可以使选项在模式的不同部分具有不同的设置。

匹配规则控制类	描述
(?i)	大小写不敏感
(?J)	允许重复命名组
(?m)	多行模式
(?n)	禁用自动捕获
(?s)	.同时匹配换行符
(?U)	默认匹配模式改为lazy
(?x)	扩展：忽略除类之外的空白
(?xx)	同(?x)，但也忽略了类中的空格和TAB
(?-...)	取消设置
(?^)	取消imnsx选项

表6-2：匹配资源控制类

以下选项仅在模式的最开始处或使用类似语法的换行符或选项之一之后才能识别。可能会出现多个。

对于前三个，d是一个十进制数。

匹配资源控制类	描述
(*LIMIT_DEPTH=d)	将回溯限制设置为d层（匹配使用系统栈进行回溯，此设置可以限制系统栈回溯层数）
(*LIMIT_HEAP=d)	将堆大小限制设置为d*1024字节（默认20000000）

匹配资源控制类	描述
(*LIMIT_MATCH=d)	将匹配限制设置为d（默认10000000）
(*NOTEMPTY)	模式不匹配空串（举例：模式a?b?匹配主题开头的空字符串，在设置该选项后无效）
(*NOTEMPTY_ATSTART)	规则基本同上，不同的是它仅在第一个匹配位置锁定空字符串匹配
(*NO_AUTO_POSSESS)	禁用“自动占有”，这是一种优化，例如，将a+b转换为a++b，以避免回溯到永远不会成功的a+。如果希望匹配函数执行完整的未优化搜索并运行所有标注，则可以设置此选项，但它主要用于测试目的。不建议使用。
(*NO_DOTSTAR_ANCHOR)	禁用.*锚定优化，使用建议同(*NO_AUTO_POSSESS)，不建议使用。
(*NO_JIT)	禁用JIT优化（YashanDB正则表达式引擎不支持JIT模式，该设置将失效）
(*NO_START_OPT)	禁用没有开始匹配优化，使用建议同(*NO_AUTO_POSSESS)，不建议使用。
(*UTF)	为正在使用的匹配设置UTF模式
(*UCP)	这个选项有两个效果。首先，它改变了PCRE2处理\B、\b、\D、\d、\S、\s、\W、\w和一些POSIX字符类的方式。默认情况下，只能识别ASCII字符，但如果设置PCRE2_UCP，则使用Unicode属性对字符进行分类。如果设置PCRE2_UCP，匹配它影响的项目需要更长的时间。PCRE2_UCP的第二个影响是强制对代码点大于127的字符使用Unicode属性进行上/下大小写操作，即使未设置PCRE2_UTF。

表6-3：换行符控制类

这些换行符控制类将改变匹配过程中对换行符的识别规则，默认换行符为LF，即hex 0A。

以下选项仅在模式的最开始处或使用类似语法的换行符或选项之一之后才能识别。仅限一个。

newline约定的选择不会影响\n或\r转义符的解释，也不会影响\R匹配的内容。这些有自己的独立约定。

换行符控制类	描述
(*CR)	仅回车符（hex 0D）
(*LF)	仅换行符（hex 0A）（默认）
(*CRLF)	回车符+换行符（hex 0D + hex 0A）
(*ANYCRLF)	上边三项任其一
(*ANY)	任何Unicode newline序列
(*NUL)	NUL字符（二进制零）
(*BSR_ANYCRLF)	CR、LF或CRLF
(*BSR_UNICODE)	任何Unicode newline序列

组类：

表7-1：捕获组类

捕获组通常与反向引用和搭配使用，例如：

1. 模式(?<leader>king|queen)\g{leader}匹配字符串kingking。 命名组leader匹配到king，之后\g{leader}反向引用匹配到的king。
2. 模式(?<leader>king|queen)\g{leader}不匹配字符串kingqueen。

捕获组的序号按照模式中左括号的先后顺序排列。

注：在非UTF模式下，名称可能包含下划线、ASCII字母和数字；在UTF模式下，允许使用任何Unicode字母和Unicode十进制数字。 在这两种情况下，名称不得以数字开头。

捕获组类	描述

捕获组类	描述
(...)	捕获组
(?<name>...)	命名捕获组（Perl）
(?'name'...)	命名捕获组（Perl）
(?P<name>...)	命名捕获组（Python）
(?:...)	非捕获组
(?|...)	非捕获组；重置的组号，捕获每个备选方案中的组

表7-2：原子组类

原子组类	描述
(?>...)	原子非捕获组
(*atomic:...)	原子非捕获组

表7-3：注释组类

注释组类	描述
(?#....)	注释（不可嵌套）
#....	注释（仅生效于match_param设定包含"x"的情况下）

环视断言：

表8-1：前视和后视断言

前视断言可以控制匹配返回结果只包含匹配项的部分内容，其他断言类似，例如： 调用regexp_substr函数匹配字符串beijing_2022，可以使用含有前视断言的模式"beijing(?=_[1-9]*)"使其只返回beijing。

积极前视断言、消极前视断言、积极后视断言、消极后视断言使用举例：

1. 模式 `\w(?=;)` 匹配任何单词后跟一个分号，但不包含这个分号
2. 模式 `\w(?!=;)` 匹配任何单词后跟一个非分号，但不包含这个非分号
3. 模式 `(?<=;)\w` 匹配分号后跟任何单词，但不包含这个单词
4. 模式 `(?<!;)\w` 匹配分号后跟任何非单词，但不包含这个非单词

注：后视断言的每个顶级分支必须具有固定的长度。

断言	描述
(?=...)	积极前视断言
(*pla:...)	积极前视断言
(*positive_lookahead:...)	积极前视断言
(?!...)	消极前视断言
(*nla:...)	消极前视断言
(*negative_lookahead:...)	消极前视断言
(?<=...)	积极后视断言

断言	描述
(*plb:...)	积极后视断言
(*positive_lookbehind:...)	积极后视断言
(?<!...)	消极后视断言
(*nlb:...)	消极后视断言
(*negative_lookbehind:...)	消极后视断言

表8-2：非原子环视断言

断言	描述
(?*...)	非原子积极前视断言
(*napla:...)	非原子积极前视断言
(*non_atomic_positive_lookahead:...)	非原子积极前视断言
(?<*...)	非原子积极后视断言
(*naplb:...)	非原子积极后视断言
(*non_atomic_positive_lookbehind:...)	非原子积极后视断言

运行脚本：

表9-1：运行脚本类

脚本类是一系列字符，这些字符都来自相同的Unicode脚本，如拉丁语或希腊语。 使用脚本类可用于检测看起来相同但来自不同脚本的字符的欺骗攻击。 字符串"paypal.com"是一个臭名昭著的例子，字母可能是拉丁文和西里尔文的混合体。 此模式确保在空格后的非空格序列中匹配的字符是拉丁语系：

\s+(?=\p{Latin})(*sr:\S+)

运行脚本类	描述
(*script_run:...)	脚本运行，可被回溯到
(*sr:...)	脚本运行，可被回溯到
(*atomic_script_run:...)	原子脚本运行
(*asr:...)	原子脚本运行

引用：

YashanDB的正则表达仅支持在模式内部进行反向引用，不支持在regexp_replace函数的replace参数处使用。

表10-1：反向引用符

反向引用符	描述
\n	编号引用（可能不明确）
\gn	编号引用
\g{n}	编号引用
\g+n	相对编号引用（扩展）

反向引用符	描述
\g-n	相对编号引用
\g{+n}	相对编号引用（扩展）
\g{-n}	相对编号引用
\k<name>	编号引用（Perl）
\k'name'	编号引用（Perl）
\g{name}	编号引用（Perl）
\k{name}	编号引用（.NET）
(?P=name)	编号引用（Python）

表10-2：子例程引用符（可能递归）

子例程引用符	描述
(?R)	递归整个模式
(?n)	按绝对数调用子例程
(?+n)	按相对数调用子例程
(?-n)	按相对数调用子例程
(?&name)	按名称调用子例程（Perl）
(?P>name)	按名称调用子例程（Python）
\g<name>	按名称调用子例程（Oniguruma）
\g'name'	按名称调用子例程（Oniguruma）
\g<n>	按绝对数调用子例程（Oniguruma）
\g'n'	按绝对数调用子例程（Oniguruma）
\g<+n>	按相对编号调用子例程（扩展）
\g'+n'	按相对编号调用子例程（扩展）
\g<-n>	按相对编号调用子例程（扩展）
\g'-n'	按相对编号调用子例程（扩展）

条件模式：

表11-1：条件模式调用

条件模式	描述
(?(condition)yes-pattern)	condition符合即使用yes模式匹配，失败不匹配
(?(condition)yes-pattern|no-pattern)	condition符合即使用yes模式匹配，失败使用no模式匹配

表11-2：条件类

条件类	描述

条件类	描述
(?(n)	绝对参考条件
(?(+n)	相对参考条件
(?(-n)	相对参考条件
(?(<name>)	命名参考条件（Perl）
(?('name')	命名参考条件（Perl）
(?(name)	命名参考条件（已弃用）
(?(R)	总体递归条件
(?(Rn)	特定编号组递归条件
(?(R&name)	特定的命名组递归条件
(?(DEFINE)	定义组以供参考
(?(VERSION[>]=n.m)	测试版本
(?(assert)	断言条件

回溯控制：

表12-1：回溯控制符

所有回溯控制动词的格式可以是(*VERB:NAME)，对于(*MARK)，名称是必填的，对于其他名称是可选的。

注：

1. 第四项-第八项表示仅当后续匹配失败导致回溯到达它们时，才执行对应操作。 它们都会迫使匹配失败，但它们在匹配后会发生什么情况上有所不同。只有在模式未锚定的情况下，才会提前匹配点的起点。
2. 在称为子例程的组中，这些动词之一的作用仅限于子例程调用。

回溯控制符	描述
(*ACCEPT)	强制成功匹配
(*FAIL)	强制回溯；同义词(*F)
(*MARK:NAME)	设置要传回的名称；同义词(*:NAME)
(*COMMIT)	总体失败，起点不提前
(*PRUNE)	前进到下一个起始字符
(*SKIP)	前进到较早的(*MARK:NAME)的位置，如果没找到，则忽略(*SKIP)
(*SKIP:NAME)	绝对参考条件
(*THEN)	局部失败，返回到下一个替换项

标注：

表13-1：标注符

允许的字符串分隔符为“”^%$（起始和结束的分隔符相同），起始分隔符{与结束分隔符匹配}。 要在字符串中对结束分隔符进行匹配，请重复输入它。

标注	描述

标注	描述
(?C)	标注（假定编号0）
(?Cn)	标注数字数据n
(?C"text")	带字符串数据的标注

compile_clause

compile_clause::=

```
syntax::= COMPILE [DEBUG] [PACKAGE|SPECIFICATION|BODY] [compiler_parameters_clause] [REUSE SETTINGS]
```

compiler_parameters_clause::=

```
syntax::= parameter_name "=" "'" parameter_value "'"
```

compile_clause用于执行某个对象的重编译。

如果重编译的对象有任何依赖的对象失效，系统将首先重编译这些依赖的对象。

对一个对象重编译成功后，该对象将被置为有效状态。如果重编译失败，系统返回相应报错，该对象变为无效状态，且系统会同时失效依赖于该对象的其他对象。

重编译选项

debug

用于语法兼容，无实际含义。

package|specification|body

用于指定重编译的范围，可省略，则默认为PACKAGE。只能在重编译自定义高级包时指定此选项。

PACKAGE

重编译高级包的HEAD和BODY（如果存在）。

SPECIFICATION

重编译高级包的HEAD。

BODY

重编译高级包的BODY。

compiler_parameters_clause

指定重编译的参数，可指定的parameter_name及parameter_value见下文描述。

reuse settings

用于语法兼容，无实际含义。

重编译参数

详细参数见下表：

parameter_name	描述	parameter_value
PLSCOPE_SETTINGS	用于语法兼容，无实际含义	v:c {, v:c} v为IDENTIFIERS|STATEMENTS c为ALL|NONE|PLSQL|SQL|PUBLIC(for IDENTIFIERS)或ALL|NONE(for STATEMENTS)
PLSQL_CCFLAGS	用于语法兼容，无实际含义	v:c {, v:c} v为YashanDB的某个保留关键字 c为true|false
PLSQL_CODE_TYPE	用于语法兼容，无实际含义	INTERPRETED|NATIVE

parameter_name	描述	parameter_value
PLSQL_OPTIMIZE_LEVEL	用于语法兼容，无实际含义	0~3之间的一个整数
PLSQL_WARNINGS	用于语法兼容，无实际含义	v:c {, v:c} v为ENABLE|DISABLE|ERROR c为ALL|SEVERE|INFORMATIONAL|PERFORMANCE |integer|(integer,interger...)
NLS_LENGTH_SEMANTICS	用于语法兼容，无实际含义	NLS_LENGTH_SEMANTICS=BYTE|CHAR
PERMIT_92_WRAP_FORMAT	用于语法兼容，无实际含义	true|false
PLSQL_DEBUG	用于语法兼容，无实际含义	true|false

示例（单机、共享集群部署）

```
ALTER PACKAGE sales.calc_fee COMPILE PLSCOPE_SETTINGS='IDENTIFIERS:PLSQL,IDENTIFIERS:SQL,STATEMENTS:NONE';
```

condition

condition::=

```
syntax::= ["("] (([NOT] expr)|
expr ("<"|"<="|">"|">="|"="|"<>"|"!=") (((ANY|SOME|ALL) (subquery|expr_list))|expr)|
((expr [NOT] IN (expr_list|subquery))|(expr_list [NOT] IN ((expr_list) {"," (expr_list)}|subquery)))|
expr [NOT] LIKE expr|
expr [NOT] RLIKE expr|
([NOT] EXISTS) subquery|
expr IS ([NOT] NULL)|
expr [NOT] BETWEEN expr AND expr|
rownum_clause) [")"]
[(AND|OR) condition]
```

expr及expr_list定义

condition为条件子句，被应用于WHERE、CHECK等需要进行条件判断的语法场景中，condition将返回TRUE或FALSE值。

YashanDB支持符合SQL标准的各类条件指定：

- NOT
- =|>|>=|<|<=|!=|<> [ANY|SOME|ALL]
- IN|NOT IN
- LIKE|NOT LIKE [ESCAPE]
- RLIKE|NOT RLIKE
- EXISTS|NOT EXISTS
- IS NULL|IS NOT NULL
- BETWEEN...AND...
- ROWNUM
- 能产生BOOLEAN值的任何表达式

比较条件

通过比较符（=|>|>=|<|<=|!=|<>）对其左右两边的表达式进行比较，获得布尔结果。

any|some

ANY|SOME语法从比较符（=|>|>=|<|<=|!=|<>）右边的集合中选出符合比较条件的子集，然后对左右两边逐一比较，只要获得一个TRUE值，则返回TRUE；全部为FALSE值时，返回FALSE；右边集合为子查询且为空时，返回FALSE。

右边的集合可以为表达式列表，或者子查询语句；对于不指定ANY|SOME的比较，比较符的右边只能为产生单行结果的表达式。

当比较符的左右任一边出现NULL时，均返回FALSE。

示例

```
--ANY与SOME效果相同
SELECT branch_no,branch_name
FROM branches
WHERE branch_no < ANY('0303','0401');
BRANCH_NO BRANCH_NAME
--------- ----------------------------------------------------------------
0001      深圳
0101      上海
0102      南京
0103      福州
```

```
0104      厦门
0201      成都

SELECT branch_no,branch_name
FROM branches
WHERE area_no <> SOME(SELECT area_no FROM area WHERE area_no IN ('01','02'));
BRANCH_NO BRANCH_NAME
--------- ----------------------------------------------------------------
0101      上海
0102      南京
0103      福州
0104      厦门
0401      北京
0402      天津
0403      大连
0404      沈阳
0201      成都
0502      长沙
```

all

ALL语法从比较符（=|>|>=|<|<=|!=|<>）右边的集合中选出符合比较条件的子集，然后对左右两边逐一比较，只要获得一个FALSE值，则返回FALSE；全部为TRUE值时，返回TRUE；右边集合为子查询且为空时，返回TRUE。

右边的集合可以为表达式列表，或者子查询语句；对于不指定ALL的比较，比较符的右边只能为产生单行结果的表达式。

当比较符的左右任一边出现NULL时，均返回FALSE；但对于 NULL > ALL（结果为空的集合），返回TRUE。

示例

```
SELECT branch_no,branch_name
FROM branches
WHERE branch_no != ALL('0001','0303','0401','0403');
BRANCH_NO BRANCH_NAME
--------- ----------------------------------------------------------------
0101      上海
0102      南京
0103      福州
0104      厦门
0201      成都
0402      天津
0404      沈阳
0502      长沙
```

in条件

IN语法将左边集合里的值与右边集合里的值逐一做相等比较，左边的值全部命中则返回TRUE，否则返回FALSE。

右边的集合可以为表达式列表，或者子查询语句；当左边为表达式列表时，右边必须为表达式列表的集合，子查询则返回的是对应列表列项的结果集。

示例（HEAP表）

```
--行存表
SELECT branch_no,branch_name
FROM branches b
WHERE (branch_no,area_no) IN
(SELECT b.branch_no,area_no FROM area a
WHERE a.area_no IN ('01','02')
AND a.area_no=b.area_no);
BRANCH_NO BRANCH_NAME
--------- ----------------------------------------------------------------
0101      上海
0102      南京
0103      福州
0104      厦门
0201      成都
```

示例（LSC表、TAC表）

```
--列存表
SELECT branch_no,branch_name
FROM branches b
WHERE SUBSTR(branch_no,1,2) IN ('01','02');
BRANCH_NO BRANCH_NAME
--------- --------------------
0101      上海
0102      南京
0103      福州
0104      厦门
0201      成都
```

like条件

LIKE语法需要指定匹配的字符表达式char1与模式串char2，通过匹配算法将字符表达式char1与模式char2做匹配，匹配成功则返回TRUE，匹配失败则返回FALSE。其完整语法形式为：

char1 [NOT] LIKE char2 [ESCAPE esc_char].

其中，char1 代表要匹配的字符表达式，char2代表要匹配的模式，esc_char代表escape字符。

LIKE语句执行如下规则：

- char1，char2，和esc_char都可以是一个表达式，其结果可以是任何数据类型，当它们的数据类型不一致时，都将转为VARCHAR类型进行判断。
- 当char1或者char2为NULL时，返回NULL。
- char2可以包含特殊的模式匹配字符：下划线 （_） 表示与值中的一个字符完全匹配；百分比符号（%）表示可以匹配值中的零或多个字符（不包括空）。
- escape用于转义特殊的模式匹配字符，即将%或_转变为本身的字面意思。如指定了escape，则char2中的esc_char后字符必须为%或_或esc_char自身，否则返回YAS-04428或YAS-04429错误。
- esc_char后字符为esc_char自身，表示将其转变为本身的字面意思，如当esc_char为`/`时，模式`//`匹配的就是`/`这个字符，但`///`中的第三个`/`将作为转义符。
- esc_char必须是长度1的字符，也可以是运算后变成长度为1的字符。

示例

```
--完全匹配
SELECT branch_no,branch_name
FROM branches
WHERE branch_no||'11/1' LIKE '_1__11_1';
BRANCH_NO BRANCH_NAME
--------- ----------------------------------------------------------------
0101      上海
0102      南京
0103      福州
0104      厦门

--转义后无数据满足条件
SELECT branch_no,branch_name
FROM branches
WHERE branch_no||'11/1' LIKE '_1__11/_1' ESCAPE '/';
BRANCH_NO BRANCH_NAME
--------- ----------------------------------------------------------------


--改为本身字符查找，且去掉其后的_通配符后可以查找到数据
SELECT branch_no,branch_name
FROM branches
WHERE branch_no||'11/1' LIKE '_1__11//1' ESCAPE '/';
BRANCH_NO BRANCH_NAME
--------- ----------------------------------------------------------------
```

```
0101      上海
0102      南京
0103      福州
0104      厦门
```

rlike条件

RLIKE语法实现功能与REGEXP_LIKE函数相同，通过正则表达式匹配算法将字符表达式expr与正则表达式regexp做匹配，匹配成功返回TRUE，匹配失败返回FALSE。其完整语法形式为：

expr [NOT] RLIKE regexp.

RLIKE语句执行如下规则：

- expr为要匹配的字符表达式，须为字符型，或可转换为字符型的其他类型。
- regexp为要匹配的RegExp，须为字符型，或可转换为字符型的其他类型，长度不超过512字节。
- 当expr或者regexp为NULL时，返回NULL。

示例

```
SELECT branch_no,branch_name
FROM branches
WHERE branch_no RLIKE '01[0-9]+';
BRANCH_NO BRANCH_NAME
--------- ----------------------------------------------------------------
0101      上海
0102      南京
0103      福州
0104      厦门
```

between and条件

BETWEEN AND语法确定一个表达式的值是否在其他两个表达式定义的间隔内。在语句中出现的三个表达式都必须是数值型、字符型或日期时间型表达式，当三个表达式的数据类型不一致时，会先进行数据类型的转换，转换失败时返回错误。

示例

```
SELECT branch_no,branch_name
FROM branches
WHERE branch_no NOT BETWEEN '01' AND '05';
BRANCH_NO BRANCH_NAME
--------- ----------------------------------------------------------------
0001      深圳
0501      武汉
0502      长沙
```

rownum_clause

ROWNUM是系统顺序分配的查询返回的行编号，返回的第一行分配的是1，第二行是2，依此类推。该语句通过ROWNUM伪列限制查询返回的总行数，具体使用规则ROWNUM伪列。

示例（单机、共享集群部署）

```
SELECT * FROM branches WHERE ROWNUM != 3; --返回ROWNUM为1,2共2条记录
SELECT * FROM branches WHERE ROWNUM != 1; --无记录返回
SELECT * FROM branches WHERE ROWNUM >= 1; --返回全部记录
SELECT * FROM branches WHERE ROWNUM > 1; --无记录返回
SELECT * FROM branches WHERE ROWNUM < 5; --返回ROWNUM为1,2,3,4共4条记录
SELECT * FROM branches WHERE ROWNUM <= 5; --返回ROWNUM为1,2,3,4,5共5条记录
SELECT * FROM branches WHERE ROWNUM = 1; --返回ROWNUM为1共1条记录
SELECT * FROM branches WHERE ROWNUM = 2; --无记录返回
SELECT * FROM branches WHERE ROWNUM BETWEEN 1 AND 8;  --返回ROWNUM为1,2..8共8条记录
SELECT * FROM branches WHERE ROWNUM BETWEEN 2 AND 8;  --无记录返回
```

```
--将ROWNUM用于子查询中，获得前5名记录
SELECT branch_no,branch_name FROM
(SELECT ROWNUM AS rn, a.* FROM branches a ORDER BY branch_no DESC)
WHERE rn<=5;
BRANCH_NO BRANCH_NAME
--------- ----------------------------------------------------------------
0104      厦门
0103      福州
0102      南京
0101      上海
0001      深圳
```

constraint

constraint::=

```
syntax::= inline_constraint|out_of_line_constraint
```

inline_constraint**::=**

```
syntax::= [CONSTRAINT constraint_name] (UNIQUE|(PRIMARY KEY)|references_clause|CHECK condition|([NOT] NULL))
[constraint_state]
```

references_clause**::=**

```
syntax::= REFERENCES [schema "."] ref_table_name ( ["(" ((column_name) {"," (column_name)}) ")"] [ON DELETE (RESTRICT|NO
ACTION|CASCADE|SET NULL)] [ON UPDATE RESTRICT])
```

constraint_state**::=**

```
syntax::= (NOT DEFERRABLE|INITIALLY IMMEDIATE|(RELY|NORELY)|using_index_clause|(ENABLE|DISABLE)|(VALIDATE|NOVALIDATE))
{" " (NOT DEFERRABLE|INITIALLY IMMEDIATE|(RELY|NORELY)|using_index_clause|(ENABLE|DISABLE)|(VALIDATE|NOVALIDATE))}
```

using_index_clause**::=**

```
syntax::= USING INDEX [(["schema."] index_name | "(" create_index_clause ")" | index_attr_clause )]
```

out_of_line_constraint**::=**

```
syntax::= [CONSTRAINT constraint_name] ((UNIQUE "(" ((column_name) {"," (column_name)}) ")")|(PRIMARY KEY "(" ((column_name)
{"," (column_name)}) ")")|(FOREIGN KEY "(" ((column_name) {"," (column_name)}) ")" references_clause)|CHECK condition)
[constraint_state]
```

constraint用于对表中的数据进行指定规则的约束定义，只有符合约束定义的数据才能被生成，否则系统将会提示违反约束错误。

约束基于列字段定义，被应用于CREATE TABLE、ALTER TABLE等与列字段属性定义相关的语法中。

根据在语句中位置不同，YashanDB提供了行内（inline_constraint）和行外（out_of_line_constraint）两种定义约束的方法，但NOT NULL约束项只能作为行内约束被定义。

inline_constraint

行内约束，即在定义列字段属性的同时为其定义约束项。

可通过CONSTRAINT constraint_name指定约束项的名称，省略则由系统生成默认名称。

可以在行内定义的约束项有：

- UNIQUE：适用于所有表
- PRIMARY KEY：适用于所有表
- FOREIGN KEY：适用于HEAP表
- CHECK：适用于HEAP表、TAC表
- NOT NULL：适用于所有表

示例（HEAP表）

```
--创建父表area_parent
CREATE TABLE area_parent
(area_no CHAR(2) NOT NULL PRIMARY KEY,
area_name VARCHAR2(60) CHECK (area_name IS NOT NULL),
DHQ VARCHAR2(20) DEFAULT 'ShenZhen' NOT NULL);

--创建子表branches_child,在area_no上建立到area_parent表的area_no字段上的外键约束，并定义当area_parent表的记录删除时，branches_child表受外键约束影响的行的area_no字段被置为空值
CREATE TABLE branches_child
(branch_no CHAR(4) PRIMARY KEY,
branch_name VARCHAR2(200) NOT NULL,
area_no CHAR(2) CONSTRAINT c_branches_child1 REFERENCES area_parent(area_no) ON DELETE SET NULL,
address VARCHAR2(200));
```

示例（TAC表）

```
DROP TABLE IF EXISTS area_tac;
CREATE TABLE area_tac
(area_no CHAR(2) NOT NULL PRIMARY KEY,
 area_name VARCHAR2(60) CHECK (area_name IS NOT NULL),
 DHQ VARCHAR2(20) DEFAULT 'ShenZhen' NOT NULL);
```

示例（LSC表）

```
CREATE TABLE area_lsc
(area_no CHAR(2) NOT NULL,
 area_name VARCHAR2(60),
 DHQ VARCHAR2(20) DEFAULT 'ShenZhen' NOT NULL);
```

out_of_line_constraint

行外约束，即不在某一个列字段的属性定义中，而是独立定义。

可通过CONSTRAINT constraint_name指定约束项的名称，省略则由系统生成默认名称。

可以在行外定义的约束项有：

- UNIQUE：适用于所有表
- PRIMARY KEY：适用于所有表
- FOREIGN KEY：适用于HEAP表
- CHECK：适用于HEAP表、TAC表

示例（HEAP表）

```
--创建子表branches_child1,并创建与行内约束中相同含义的外键约束
CREATE TABLE branches_child1
(branch_no CHAR(4) PRIMARY KEY,
branch_name VARCHAR2(200) NOT NULL,
area_no CHAR(2),
address VARCHAR2(200),
CONSTRAINT c_branches_child2 FOREIGN KEY (area_no) REFERENCES area_parent(area_no) ON DELETE SET NULL);
```

示例（TAC表）

```
CREATE TABLE area_tac1
(area_no CHAR(2) NOT NULL,
 area_name VARCHAR2(60),
 DHQ VARCHAR2(20) DEFAULT 'ShenZhen' NOT NULL,
 CONSTRAINT c_area_tac1 PRIMARY KEY(area_no),
 CONSTRAINT c_area_tac2 CHECK (area_name IS NOT NULL));
```

约束项

not null

为某个列字段定义非空约束项，表示要求该列字段上的值不能为空。

NULL或NOT NULL仅支持在行内约束中定义。

HEAP表不为空时不能新建NOT NULL约束的列。

NULL并不定义约束，只是显式允许列包含null。

如果既不指定NOT NULL也不指定NULL，则默认值为 NULL。

unique

为某个列字段或某些列字段组合定义唯一约束项，表示要求该列字段或列字段组合上的值在当前表中无重复。

如下为添加唯一约束项的限制：

- 当该列字段或列字段组合上已存在重复的数据时，无法成功添加此约束项。
- 唯一约束项添加成功后，系统将会自动创建一个基于唯一项字段的唯一索引；若该列字段或列字段组合上已存在索引（包括唯一索引和非唯一索引），则不会创建。
- 分布式部署中建Sharded表时，如果同时有多个唯一约束键，比如主键或唯一键，则限制所有的唯一约束键必须至少有一个公共字段 （默认的分区键取唯一约束键的公共子集中的第一个恰当的数据类型的字段），如不满足则返回错误。
- 分布式部署中在Sharded表中增加唯一性约束（包括主键约束和唯一约束），也必须包括分区键。

primary key

为某个列字段或某些列字段组合定义主键约束项，表示该列字段或列字段组合将作为表的主键。

由于行内约束是针对某一个列字段定义，因此在行内约束中只能定义单列主键，多列组合主键在行外约束中定义。

如下为添加主键约束项的限制：

- 主键拥有非空性特征，因此当该列字段或列字段组合中的任一列字段上存在空数据时，无法成功添加此约束项。
- 主键拥有唯一性特性，因此当该列字段或列字段组合上存在重复的数据时，无法成功添加此约束项。
- 一个表上只能有一个主键，因此当该列字段或列字段组合所在表已存在主键时，无法成功添加此约束项。
- 主键添加成功后，系统将会自动创建一个基于主键字段的唯一索引；若该列字段或列字段组合上已存在索引（包括唯一索引和非唯一索引），则不会创建。

check

CHECK约束用于对要插入表的数据进行条件检查，只有条件返回为true时，数据才能被成功插入表中。

当对一个已存在数据的表添加新的CHECK约束项时，系统将对现有数据也执行CHECK条件检查（在指定了NOVALIDATE时不检查），当存在未通过检查的数据时，无法成功添加此约束项。

在行内约束中，CHECK的条件内容只能指定当前列字段，条件包含多列的CHECK在行外约束中定义。

CHECK的条件可以包含的内容描述请参考SQL通用语法condition。

foreign key

为一个表（子表）的列字段或列字段组合建立到另一个表（父表）的外键约束，表示子表中该列字段或列字段组合上的值必须在父表对应的列上存在，如子表上的值为NULL，则认为在父表上存在对应值，符合外键约束。

由于行内约束是针对某一个列字段定义，因此在行内约束中只能定义单列外键，多列组合外键在行外约束中定义。

不能为临时表建立FOREIGN KEY，也不能为其他表建立关联到临时表的FOREIGN KEY。

references_clause

在定义外键约束时，REFERENCES指定了外键指向的父表，及父表中与子表外键列一一对应的列字段。若仅标识父表名而省略列名，则外键会自动引用父表的主键并且外键列与主键列需要一一对应。

以下为添加外键约束项的限制：

- 外键约束要求子表中外键列上的数据在父表对应列上必须存在，在对一个已存在数据的子表添加新的外键约束项时，系统将对子表中的现有数据执行此项检查（在指定了NOVALIDATE时不检查），当存在未通过检查的数据时，无法成功添加此约束项。
- 外键约束要求父表对应列上的数据满足唯一性约束（该对应列上已定义PRIMARY KEY或UNIQUE约束项），否则无法成功添加此约束项。
- 当子表外键列到父表对应列的外键约束已经存在时，无法成功添加此约束项。

on delete

在建立外键约束后，当对父表对应列上的某个值进行删除操作时，系统将检查该值在子表外键列上是否存在，如存在，该父表上的数据不允许删除。而在约束定义里指定了ON DELETE后，系统将允许前述对父表的操作，并同时根据不同选项执行如下操作：

- 指定ON DELETE RESTRICT（默认也是RESTRICT）创建外键后，对子表中存在数据的父表数据执行删除将报错。
- 指定ON DELETE CASCADE创建外键后，对子表中存在数据的父表数据执行删除将成功，且对子表中对应的数据也进行删除。
- 指定ON DELETE SET NULL创建外键后，对子表中存在数据的父表数据执行删除将成功，且对子表中对应的数据置空。

> **Note**：
> 子表需建立外键的列字段上存在NOT NULL约束项时，不能指定ON DELETE SET NULL选项创建外键。

此外，系统还提供如下仅用于语法兼容的选项：

- ON DELETE NO ACTION（行为与RESTRICT一致）

on update

在建立外键约束后，当对父表对应列上的某个值进行更新操作时，系统将检查该值在子表外键列上是否存在，如存在，该父表上的数据不允许更新。而在约束定义里指定了ON UPDATE后，系统将允许前述对父表的操作，并同时根据不同选项执行如下操作：

- 指定ON UPDATE RESTRICT（默认也是RESTRICT）创建外键后，对子表中存在数据的父表数据执行更新将报错。
- 指定ON UPDATE CASCA DE创建外键后，对子表中存在数据的父表数据执行更新将成功，且对子表中对应的数据也进行更新。
- 指定ON UPDATE SET NULL创建外键后，对子表中存在数据的父表数据执行更新将成功，且对子表中对应的数据置空。

> **Note**：
> 子表需建立外键的列字段上存在NOT NULL约束项时，不能指定ON UPDATE SET NULL选项创建外键。

约束项属性

YashanDB提供一系列选项用于定义某个约束项的属性，其中如下选项只用于语法兼容，并无实际含义：

- NOT DEFERRABLE
- INITIALLY IMMEDIATE
- RELY|NORELY

using_index_clause

该语句只作为PRIMARY KEY或UNIQUE约束项的属性选项使用，用于指定约束项对应的索引或索引属性，对其他约束项使用本语句将报错。

YashanDB中，创建PRIMARY KEY或UNIQUE约束项时，系统将自动创建一个对应的唯一索引（如该唯一索引已存在则不会创建）。使用本语句将系统自动的操作修改为人工指定，所实现效果略有差异，具体见下面描述。

using index

该语句仅用于已有约束项为disable的情况，将该约束项置为enable，并自动创建索引，索引名称等同约束名称。

using index [schema.]index_name

为约束项指定一个已有的索引，该索引必须满足以下条件：

- 必须为一个唯一索引。

- 索引基于的表和列与约束项必须一致。
- 不能为倒序索引。

通过此种方式指定的索引，与自动创建的索引的差别为：如后续对约束项进行删除操作，自动创建的索引会随之删除，而此种方式指定的索引并不会被删除。

示例（HEAP表、TAC表）

```
--创建表area_using_index
CREATE TABLE area_using_index
(area_no CHAR(2) NOT NULL,
 area_name VARCHAR2(60),
 DHQ VARCHAR2(20) DEFAULT 'ShenZhen' NOT NULL);

--创建唯一索引idx_area_using_index_1
CREATE UNIQUE INDEX idx_area_using_index_1 ON area_using_index(area_no);

--为表area_using_index创建主键并指定索引
ALTER TABLE area_using_index ADD PRIMARY KEY(area_no) USING INDEX idx_area_using_index_1;
```

using index index_attr_clause

使用本语句表示在创建约束项时，仍由系统自动创建唯一索引，但对该索引的相关属性进行指定。

在不使用using_index_clause且不存在唯一索引时，系统在创建主键时会自动创建一个唯一索引，该索引所有属性均采用默认值，例如索引的表空间将使用和表所在的表空间，使用本语句则可以避免单独构建索引的繁琐，同时也可以满足将索引和表进行物理隔离等实际需求。

需注意的是，使用本语句的前提是约束项列上不存在唯一索引，否则系统并不是直接采用该索引，而是返回失败。

index_attr_clause

指定索引属性的语句，其语法与CREATE INDEX中的index_attr_clause一致。

示例（HEAP表，单机TAC表）

```
--创建表area_index_attr，同时指定索引的表空间
CREATE TABLE area_index_attr
(area_no CHAR(2) PRIMARY KEY USING INDEX TABLESPACE yashan,
 area_name VARCHAR2(60),
 DHQ VARCHAR2(20) DEFAULT 'ShenZhen' NOT NULL);
```

using index create_index_clause

使用本语句等同于由系统自动创建唯一索引，但与系统自动创建采用默认值不同的是，本语句可以指定索引的名称、属性等各信息。

create_index_clause

创建索引的语句，除无法创建倒序索引之外，其余语法与CREATE INDEX一致。

需注意的是，当约束项列上已存在唯一索引时，create_index_clause中的index_name必须与已存在索引同名，且index_attr_clause所指定属性必须与已存在索引属性相同，否则返回失败。

示例（HEAP表，单机TAC表）

```
--创建表area_create_index
CREATE TABLE area_create_index
(area_no CHAR(2) NOT NULL,
 area_name VARCHAR2(60),
 DHQ VARCHAR2(20) DEFAULT 'ShenZhen' NOT NULL);

--为表area_create_index创建主键并创建索引
ALTER TABLE area_create_index
ADD PRIMARY KEY(area_no)
USING INDEX (CREATE INDEX idx_area_create_index_1 ON area_create_index(area_no) TABLESPACE yashan);

--创建倒序索引会返回错误
ALTER TABLE area_create_index DROP PRIMARY KEY;
```

```
ALTER TABLE area_create_index
ADD PRIMARY KEY(area_no)
USING INDEX (CREATE INDEX idx_area_create_index_2 ON area_create_index(area_no DESC) TABLESPACE yashan);

YAS-02210 specified index cannot be used to enforce the constraint
```

validate|novalidate

指定在定义某个约束项时，是否对表中现有的数据进行该项约束检查，VALIDATE表示检查，NOVALIDATE表示不检查。可省略，则默认为VALIDATE。

对于UNIQUE/PRIMARY KEY/NOT NULL约束项，系统将始终默认为VALIDATE，执行约束检查，即使指定了NOVALIDATE。

示例（HEAP表、TAC表）

```
--area表现有数据
SELECT * FROM area;
AREA_NO AREA_NAME                          DHQ
------- ---------------------------------- ------------
01      华东                               Shanghai
02      华西                               Chengdu
03      华南                               Guangzhou
04      华北                               Beijing
05      华中                               Wuhan

--为表添加CHECK约束
ALTER TABLE area ADD CHECK (LENGTH(area_name)>10);
YAS-02255 cannot validate constraint - check constraint violated

--为表添加NOVALIDATE的CHECK约束
ALTER TABLE area ADD CHECK (LENGTH(area_name)>10) NOVALIDATE;
```

enable|disable

指定在定义某个约束项时，是否启用约束，ENABLE表示启用，DISABLE表示停用。可省略，则默认为ENABLE。

在定义UNIQUE/PRIMARY KEY约束项并指定DISABLE时，系统不会创建对应的索引。

示例（HEAP表、TAC表）

```
--area表现有数据
SELECT * FROM area;
AREA_NO AREA_NAME                          DHQ
------- ---------------------------------- ------------
01      华东                               Shanghai
02      华西                               Chengdu
03      华南                               Guangzhou
04      华北                               Beijing
05      华中                               Wuhan

--为表添加CHECK约束停用状态
ALTER TABLE area ADD CONSTRAINT ck CHECK (LENGTH(DHQ)>10) DISABLE;

--插入违反本约束的值，但需遵循上例中的CHECK约束
INSERT INTO area VALUES ('06','华北华北华北华北华北1','tianjing');

--启用CHECK约束
ALTER TABLE area MODIFY CONSTRAINT ck ENABLE;
YAS-02255 cannot validate constraint - check constraint violated
```

dblink

dblink::=

```
syntax::= "@" dblink_name
```

该语法用于YashanDB对远端数据库的表操作，远端数据库包含同构数据库和异构数据库。

dblink_name指在CREATE DATABASE LINK时所创建的远端数据库名称。

使用说明

1.通过dblink操作Oracle的表时：

- 需首先安装libaio库和下载Oracle Instant Client安装包，详细操作见异构数据库链接配置。
- 远端表字段的数据类型需在下表所列范围内：
  - smallint
  - int/integer
  - float
  - binary_float
  - binary_double
  - number/decimal
  - date
  - timestamp
  - timestamp with time zone
  - timestamp with local time zone
  - interval year to month
  - interval day to second
  - char
  - varchar
  - nchar
  - nvarchar
  - raw
  - varchar2
  - nvarchar2

2.通过dblink操作YashanDB的表时，远端表字段的数据类型需在下表所列范围内：

- bool
- tinyint
- smallint
- integer
- bigint
- float/binary_float
- double/binary_double
- number
- date
- time
- timestamp
- interval year to month
- interval day to second
- char
- varchar
- raw
- bit
- rowid

3.通过dblink执行多表连接时：

- 本地表和远端表连接：

本地表	远端表	支持情况
行存	行存	支持
行存	列存	支持
列存	行存	不支持
列存	列存	不支持

- 远端表和远端表连接：

远端表	远端表	支持情况
行存	行存	支持
行存	列存	支持
列存	列存	支持

查询远端表

对远端表进行SELECT操作时，单次查询支持的远端表数量最大是32个。

示例（单机、共享集群部署）

```
--创建远端表
conn sys/sys
CREATE TABLE table_test(c1 INT);

--创建dblink，并访问远端表
conn sales/sales
CREATE DATABASE link link_test CONNECT TO sys IDENTIFIED BY sys USING '192.168.1.2:1688';
SELECT * FROM table_test@link_test;

--更改远端表的元数据
conn sys/sys
ALTER TABLE table_test ADD COLUMN c2;

conn sales/sales

SELECT * FROM table_test@link_test;
C1          C2
------------ ------------
```

插入远端表

对远端表进行INSERT操作时，有如下限制：

- 不允许多表insert。
- 不允许指定分区insert。
- 不允许执行insert duplicate update语句。
- 不允许执行insert return语句。

示例（单机、共享集群部署）

```
--创建远端表
conn sys/sys
CREATE TABLE table_test(c1 INT, c2 INT);
```

```
--创建dblink，并访问远端表
conn sales/sales
CREATE DATABASE link link_test CONNECT TO sys IDENTIFIED BY sys USING '192.168.1.2:1688';
SELECT * FROM table_test@link_test;

--插入数据
INSERT INTO table_test@link_test VALUES(1,2);
```

更新远端表

对远端表进行UPDATE操作时，有如下限制：

- filter与更新本地数据库的对象的filter相比：
  - 不能使用聚集函数。
  - 不能使用窗口函数。
  - 不能使用子查询。
  - 不能使用序列。
  - 不能使用自定义函数（包括UPDATE SET语句）。
- 不允许多表update。
- 不允许指定分区update。

示例（单机、共享集群部署）

```
--创建远端表
conn sys/sys
CREATE TABLE table_test(c1 INT, c2 INT);

--创建dblink
conn sales/sales
CREATE DATABASE link link_test CONNECT TO sys IDENTIFIED BY sys USING '192.168.1.2:1688';

--更改远端表的元数据
conn sys/sys
UPDATE table_test@linktest SET c1=1,c2=2;
```

删除远端表

对远端表进行DELETE操作时，有如下限制：

- filter与删除本地数据库的对象的filter相比：
  - 不能使用聚集函数。
  - 不能使用窗口函数。
  - 不能使用子查询。
  - 不能使用序列。
  - 不能使用自定义函数。
- 不允许多表delete。
- 不允许指定分区delete。

示例（单机、共享集群部署）

```
conn sys/sys
CREATE TABLE table_test(c1 INT);

--创建dblink
conn sales/sales
CREATE DATABASE link link_test CONNECT TO sys IDENTIFIED BY sys USING '192.168.1.2:1688';

--删除远端表数据
DELETE FROM table_test@link_test WHERE c1=1;
```

expr

expr::=

```
syntax::= column|literal|function|NULL|ROWNUM|oper_expr
```

oper_expr::=

```
syntax::= expr ("+"|"-"|"*"|"/"|"%"|"||") expr
```

expr_list::=

```
syntax::= ((expr) {"," (expr)})|("(" ((expr) {"," (expr)}) ")")
```

expr代表了一个最基本的表达式，它可以如下列示项中的某一项，或者为多项组成的运算式：

- 列字段
- 变量、常量、字面量
- 函数
- NULL
- 伪列（其中ROWID和ROWSCN只适用于HEAP表）

在YashanDB中，以下一些均为合法有效的表达式：

```
area.area_no
'34.44'
SYSDATE
NULL
TO_NUMBER('11')%2
```

几乎所有的SQL语法描述中都会使用到expr，例如，其可以参与到SQL的查询列、函数参数、WHERE条件、表达式列表（expr_list）等各种语法场景中。

external_table

external_table_clause::=

```
syntax::=  "(" [TYPE access_driver_type] [external_table_data_props_clause] ")" [REJECT LIMIT (integer|UNLIMITED)]
```

external_table_data_props_clause::=

```
syntax::=  [DEFAULT DIRECTORY directory_name] [ACCESS PARAMETERS "(" [record_format_info_clause] [field_definitions_clause]
")"] [LOCATION "(" [directory_name ":"] "'" location_specifier "'" ")"]
```

record_format_info_clause::=

```
syntax::=  RECORDS ( (DELIMITED BY NEWLINE) | (output_files_clause)) {"" ((DELIMITED BY NEWLINE) | (output_files_clause))}
```

output_files_clause::=

```
syntax::=  ( (NOBADFILE | BADFILE file_pos_clause) | (NOLOGFILE | LOGFILE file_pos_clause) ) {"" ( (NOBADFILE | BADFILE
file_pos_clause) | (NOLOGFILE | LOGFILE file_pos_clause) )}
```

file_pos_clause::=

```
syntax::=  [directory_name ":"]  "'" file_name "'"
```

field_definitions_clause::=

```
syntax::=  FIELDS ( (TERMINATED BY "'"char"'") | (OPTIONALLY ENCLOSED BY "'"char"'") ) {"" ( (TERMINATED BY "'"char"'") |
(OPTIONALLY ENCLOSED BY "'"char"'") )}
```

external_table_clause用于在创建一个外部表时，指定其对应的外部数据信息。

本语句不会对外部数据是否存在及正确进行校验，即只要语法正确均可创建成功。查询一个外部表时，系统根据本语句所定义的外部数据信息进行检索，不满足条件时则会报错。

type

通过TYPE关键字指定外部表驱动名，可省略。access_driver_type的值可为YASDB_LOADER或ORACLE_LOADER，二者等价。

external_table_data_props_clause

指定外部表中数据的属性，可省略。

default directory

指定外部表对应数据文件所在的缺省目录，可省略。当后续语句中未指定directory_name时，将以本目录作为数据或日志文件所在的默认目录。

directory_name必须为YashanDB中已创建的一个目录对象，创建方式参考CREATE DIRECTORY。

access parameters

指定在查询外部表时，读取外部数据文件的参数，包括源文件名称、列格式以及日志记录。

record_format_info_clause

指定数据行格式，可省略。

delimited by newline

指定读取外部数据文件时，数据记录之间的分隔符是换行符。

output_files_clause

指定数据库在读取外部数据时的日志信息，若省略该子句，系统将以默认名称创建日志文件进行记录。

nobadfile | badfile

指定BADFILE时，查询外部数据过程中遇到的错误数据将记录在file_pos_clause指定的文件中。

指定NOBADFILE时，系统不会记录查询外部数据时的错误数据。

nologfile | logfile

指定LOGFILE时，查询外部数据的操作过程将记录在file_pos_clause指定的文件中。

指定NOLOGFILE时，系统不会记录查询外部数据时的操作信息。

file_pos_clause

指定上述badfile或logfile所对应的物理文件，格式为directory_name:file_name。

directory_name必须为YashanDB中已创建的一个目录对象，省略时默认为DEFAULT DIRECTORY所指定目录对象。

file_name为一个操作系统可接受的文件名称，对于不存在的文件将由系统在记录日志时创建，对于已存在的文件，系统将对badfile进行覆盖填写，对logfile进行追加填写。

对于file_name，数据库会对部分符号进行替换：

- `%%` 将被替换为 `%` ；
- `%a` 、 `%p` 将被替换为当前线程ID。

field_definitions_clause

指定外部数据列格式，可省略。

terminated by

指定外部数据文件的字段之间的分隔符，分隔符的类型只能为字符，应与外部数据文件中实际使用的分隔符保持一致，若省略则默认为','。

optionally enclosed by

如果外部数据文件中使用了封闭符号（只能使用""封闭符），需通过本语句进行指定，且只能指定为""。

location

指定外部数据文件所在的路径，格式为(directory_name:location_specifier)。

directory_name必须为YashanDB中已创建的一个目录对象，省略时默认为DEFAULT DIRECTORY所指定目录对象。

location_specifier为一个已存在的CSV文件的名称。

reject limit

指定在查询外部数据时，可容忍的最多错误数据行数，达到容忍值时系统将终止查询操作。可省略，则默认为0行。UNLIMITED表示容许所有的数据错误。

示例（单机部署）

```
--在/data/area.csv中存在如下数据：
8|load|801|loadbranch|8
9|load|901|loadbranch|9

--在数据库中创建目录对象和外部表
CREATE DIRECTORY dir_ext AS '/data';
CREATE TABLE area_external(
 area_no CHAR(2),
 area_name VARCHAR2(60),
 DHQ VARCHAR2(20) DEFAULT 'ShenZhen')
ORGANIZATION EXTERNAL(
 TYPE YASDB_LOADER
```

```
 DEFAULT DIRECTORY dir_ext
 ACCESS PARAMETERS(
  RECORDS BADFILE 'area_bad.log' LOGFILE 'area_log.log'
  FIELDS TERMINATED BY ',')
 LOCATION ('area.csv')
)
REJECT LIMIT 1;

--查询表数据，由于文件内数据格式与表定义格式不匹配，将无法查询到数据且达到容错行数退出
SELECT * FROM area_external;
YAS-02687 reject limit reached
```

hint

hint::=

```
syntax::= "/*+" (hint_name [hint_intersperse]) {(hint_name [hint_intersperse])} "*/"
```

SQL在查询优化阶段，优化器根据内定规则确定执行计划；作为辅助手段， 开发者可以通过在语句中书写hint，指示优化器改变执行路径，以便快速的查询数据。

优化器可以根据系统统计信息动态调整执行计划，使用hint则对执行计划进行了唯一指定，所以应该被谨慎使用，只有在已获取到足够的统计信息，确认改变执行路径是对性能更优的方式，才建议给语句增加hint。当语句涉及的表结构、业务场景或数据量等信息发生变化时，应通过查看执行计划等手段，重新审视该hint是否合适。

hint有时候会失效，如在hint中指定了非法的顺序（OUTER JOIN则不允许指定顺序），或者指定了非法的JOIN类型等。正常情况下，优化器会优先选择有效的hint，即使该hint的代价更高，但是对于前述非法的hint，优化器会不选择该hint，并在执行计划中输出hint unused。

hint_name

hint提示项的名称，可以同时指定多项hint_name作为组合提示。

如下标识符将被作为YashanDB认可的hint提示项名称：

FULL INDEX NO_INDEX INDEX_FFS

PARALLEL

LEADING

NO_USE_HASH NO_USE_MERGE NO_USE_NL USE_HASH USE_MERGE USE_NL

SELECTIVITY

BULKLOAD

MAX_WORKERS_PER_EXEC

hint_intersperse

对于某些提示，需要提供更多的信息来结合指定，例如指定使用索引提示时的索引名称。

YashanDB中使用hint的规则如下：

- 只有跟在SELECT、UPDATE、INSERT、MERGE或DELETE等关键字后面，且以”/*+ ” 作为开始（ “+“ 号后面需有空格 ），“*/” 作为结束的语句块 ，才会被解析成hint。
- 一个关键字后跟多组”/*+ */”时，只有第一组被解析成hint，其他忽略。
- 一个hint里某项提示出现如下情况时，优化器忽略该项提示：
  - 拼写错误或语法错误
  - 不能识别的提示项名称
  - 与其他提示项相互冲突
  - hint_intersperse里包含了查询块
  - 在SQL语句中为表对象定义了别名，而提示中未使用别名

示例（HEAP表、TAC表）

```
--创建视图
CREATE OR REPLACE VIEW b AS
SELECT * FROM branches WHERE branch_no = '0101';

--hint_intersperse里包含了查询块,该项提示被忽略
EXPLAIN SELECT /*+ FULL(b) */ *
FROM area, b
WHERE area.area_no = '04'
```

```
AND area.area_no = b.area_no;

PLAN_DESCRIPTION
------------------------------------------------------------
SQL hash value: 3312245078
Optimizer: ADOPT_C

+----+--------------------------------+---------------------+------------+----------+-------------+-------------------------
-------+
| Id | Operation type                 | Name                | Owner      | Rows     | Cost(%CPU)  | Partition info
|
+----+--------------------------------+---------------------+------------+----------+-------------+-------------------------
-------+
|  0 | SELECT STATEMENT               |                     |            |          |             |
|
|* 1 |  MERGE JOIN                    |                     |            |      102|      296( 0)|
|
|  2 |   MERGE SORT                   |                     |            |          |             |
|
|  3 |    TABLE ACCESS BY INDEX ROWID | AREA                | SALES      |          |             |
|
|* 4 |     INDEX UNIQUE SCAN          | SYS_C_18            | SALES      |         1|      148( 0)|
|
|  5 |   MERGE SORT                   |                     |            |          |             |
|
|* 6 |    TABLE ACCESS BY INDEX ROWID | BRANCHES            | SALES      |          |             |
|
|* 7 |     INDEX UNIQUE SCAN          | SYS_C_20            | SALES      |         1|      148( 0)|
|
+----+--------------------------------+---------------------+------------+----------+-------------+-------------------------
-------+

Operation Information (identified by operation id):
---------------------------------------------------

   1 - Predicate : access( "AREA"."AREA_NO" = "BRANCHES"."AREA_NO" )
                   filter("AREA"."AREA_NO" = "BRANCHES"."AREA_NO")
   4 - Predicate : access("AREA"."AREA_NO" = '04')
   6 - Predicate : filter('04' = "BRANCHES"."AREA_NO")
   7 - Predicate : access("BRANCHES"."BRANCH_NO" = '0101')

Hint Information :
---------------------------------------------------

FULL(b) / unresolved

--在branches表上新增一个索引
CREATE INDEX idx_branches_1 ON branches(area_no);

--提示项冲突
EXPLAIN SELECT /*+ FULL(b) INDEX(b IDX_BRANCHES_1) */ *
FROM branches b
WHERE b.branch_no='0401' AND b.area_no='04';

PLAN_DESCRIPTION
------------------------------------------------------------------
SQL hash value: 3081809351
Optimizer: ADOPT_C

+----+--------------------------------+---------------------+------------+----------+-------------+-------------------------
-------+
| Id | Operation type                 | Name                | Owner      | Rows     | Cost(%CPU)  | Partition info
|
+----+--------------------------------+---------------------+------------+----------+-------------+-------------------------
-------+
|  0 | SELECT STATEMENT               |                     |            |          |             |
|
|* 1 |  TABLE ACCESS BY INDEX ROWID   | BRANCHES            | SALES      |          |             |
|
|* 2 |   INDEX UNIQUE SCAN            | SYS_C_20            | SALES      |         1|      148( 0)|
```

```
|
+----+-------------------------------+---------------------+------------+----------+-------------+-------------------------
-------+

Operation Information (identified by operation id):
---------------------------------------------------

   1 - Predicate : filter("B"."AREA_NO" = '04')
   2 - Predicate : access("B"."BRANCH_NO" = '0401')

Hint Information :
---------------------------------------------------

FULL(b) / conflict with another
INDEX(b idx_branches_1) / conflict with another
```

full提示项

使用FULL提示项时，需同时使用hint_intersperse指定表名，表示对该表执行全表扫描。

示例

```
SELECT /*+ FULL(a) */ * FROM area a WHERE a.area_no='02';
```

index提示项

使用INDEX提示项时，需同时使用hint_intersperse指定表名 [索引名]，表示对该表执行指定的索引扫描。

当hint_intersperse中包含多个索引时，优化器从中选择代价最低的索引；当hint_intersperse中未指定任何索引时，优化器从该表上所有的索引中选择代价最低的索引。

示例（HEAP表、TAC表）

```
--在branches表上新增一个索引
CREATE INDEX idx_branches_2 ON branches(area_no,branch_no);

--提示多个索引，或者不指定索引，优化器将选择代价最低的索引执行扫描
SELECT /*+ INDEX(b) */ *
FROM branches b
WHERE b.branch_no LIKE '0%';
```

index_ffs提示项

使用INDEX_FFS提示项时，需同时使用hint_intersperse指定表名 索引名，表示对该表执行指定的索引快速全表扫描。

只有当所有查询列都位于指定的索引列之中，即通过扫描索引就可以得到所有的查询列而不用回表时，优化器才会接受提示选择此索引。

示例

```
--按上面示例，branches上已存在idx_branches_2索引
SELECT /*+ INDEX_FFS(b idx_branches_2) */ b.area_no,b.branch_no
FROM branches b
WHERE b.area_no='04' AND b.branch_no='0401';
```

no_index提示项

使用NO_INDEX提示项时，需同时使用hint_intersperse指定表名 [索引名]，表示不使用指定的索引对该表执行索引扫描。

当hint_intersperse中包含多个索引时，这些索引都将不会被优化器使用，但如果该表上存在其他索引，则优化器可能会选择那些索引（由优化器规则决定）；当hint_intersperse中未指定任何索引时，则优化器将不会选择该表上的任何索引。

示例

```
SELECT /*+ NO_INDEX(b) */ *
FROM branches b
WHERE b.branch_no LIKE '0%';
```

parallel提示项

使用PARALLEL提示项时，需同时使用hint_intersperse指定表名和并行度 [表名,并行度]，表示对指定表使用并行查询。

通过hint所指定的并行度最大为255，超过255时会退化为255。

无法跨select指定并行，此种场景请使用数据库参数DEGREE_OF_PARALLEL来指定。

对并行查询的指定只作为一个参考项，如果优化器评估后认为并行查询不是最优计划，则不会选择生成并行计划。

示例

```
SELECT /*+ PARALLEL(area,4)*/ area_no FROM area;
SELECT /*+ PARALLEL(a,3) PARALLEL(a,4)*/ a.area_no,b.branch_no
FROM area a,branches b
WHERE a.area_no = b.area_no;

--无效提示
SELECT /*+PARALLEL(branches,4)*/ area_no FROM area
UNION
SELECT area_no FROM branches;
```

leading提示项

使用LEADING提示项时，需同时使用hint_intersperse指定表名 [表名]，表示在多表关联的查询中，提示优化器按照指定顺序访问表。

hint_intersperse中只支持以表名或别名指定顺序，大小写不敏感。

LEADING提示的表访问顺序为建议顺序，非强制，当遇到无法调整的JOIN类型（OUTER）时，则不会改变顺序。

当hint_intersperse中指定表数量超过参与JOIN的实际表数量时，优化器不接受此项提示。

当hint_intersperse中出现无效的表时，优化器不接受在该表之后指定的顺序，但仍接受该表之前所指定的顺序。

hint_intersperse中指定多张表时，表示指定的是表连接顺序的前缀，例如LEADING(b a)，((b a) c)符合指定的顺序，(c (b a))则不符合。

示例

```
SELECT /*+ LEADING(b e)*/ *
FROM branches b, department d, employees e
WHERE b.branch_no = e.branch
AND d.deparment_no = e.department
AND b.branch_no = '0401'
AND d.deparment_no = '000';
```

no_use_hash提示项

使用NO_USE_HASH提示项时，需同时使用hint_intersperse指定表名 [表名]，表示在指定的这些表与其他表进行关联查询时，提示优化器不进行HASH JOIN 。

示例

```
SELECT /*+ NO_USE_HASH(d) */ *
FROM branches b, department d, employees e
WHERE b.branch_no = e.branch
AND d.deparment_no = e.department;
```

no_use_merge提示项

使用NO_USE_MERGE提示项时，需同时使用hint_intersperse指定表名 [表名]，表示在指定的这些表与其他表进行关联查询时，提示优化器不进行MERGE JOIN。

示例

```
SELECT /*+ NO_USE_MERGE(d e) */ *
FROM branches b, department d, employees e
WHERE b.branch_no = e.branch
AND d.deparment_no = e.department;
```

no_use_nl提示项

使用NO_USE_NL提示项时，需同时使用hint_intersperse指定表名 [表名]，表示在指定的这些表与其他表进行关联查询时，提示优化器不进行NEST LOOPS。

示例

```
SELECT /*+ NO_USE_NL(a) */ *
FROM area a, branches b
WHERE a.area_no = b.area_no
AND a.area_no='04'
AND b.branch_no='0401';
```

use_hash提示项

使用USE_HASH提示项时，需同时使用hint_intersperse指定表名 [表名]，表示在指定的这些表与其他表进行关联查询时，提示优化器进行HASH JOIN 。

示例

```
SELECT /*+ USE_HASH(d) */ *
FROM branches b, department d, employees e
WHERE b.branch_no = e.branch
AND d.deparment_no = e.department;
```

use_merge提示项

使用USE_MERGE提示项时，需同时使用hint_intersperse指定表名 [表名]，表示在指定的这些表与其他表进行关联查询时，提示优化器进行MERGE JOIN。

示例

```
SELECT /*+ USE_MERGE(d e) */ *
FROM branches b, department d, employees e
WHERE b.branch_no = e.branch
AND d.deparment_no = e.department;
```

use_nl提示项

使用USE_NL提示项时，需同时使用hint_intersperse指定表名 [表名]，表示在指定的这些表与其他表进行关联查询时，提示优化器进行NEST LOOPS。

示例

```
SELECT /*+ USE_NL(a) */ *
FROM area a, branches b
WHERE a.area_no = b.area_no
```

```
AND a.area_no='04'
AND b.branch_no='0401';
```

selectivity提示项

使用SELECTIVITY提示项时，语法与其他hint不同，SELECTIVITY提示项只可跟在filter后，表示指定这些filter的选择率，其后需跟一个0到1之间的浮点数。

AND、OR和BETWEEN的选择率无法通过hint的方式指定，只能通过计算得到。

示例

```
SELECT area_no
FROM area
WHERE area_no > 1 SELECTIVITY 0.8;
```

bulkload提示项

使用BULKLOAD提示项时，无需指定表名，仅LSC表能够使用该提示项，表示对LSC表进行批量插入，可以减少redo的产生，提高插入速度，但是对事务可见性有一定的影响，详见下文约束项。

示例（LSC表）

```
INSERT /*+ BULKLOAD */ INTO area  VALUES ('06','华东','Shanghai');
```

使用BULKLOAD提示项对表进行插入优化时，在事务提交前，保证其他事务对该插入的数据不可见，不保证本插入事务对新插入数据的可见性。事务提交后，保证该插入的数据对本会话和其他事务可见。

使用BULKLOAD提示项的相关约束如下：

- BULKLOAD提示项仅适用于单表插入。
- 开启自动提交时，无法使用BULKLOAD提示项。
- BULKLOAD提示项与SAVEPOINT功能不能同时使用。
- BULKLOAD提示项不能在INSERT ON DUPLICATE UPDATE语句中使用。
- 同一个事务内，使用BULKLOAD提示项进行插入后，不允许再对该表进行除插入（但不能为INSERT ON DUPLICATE UPDATE）、查询外的其他DML操作。

使用BULKLOAD提示项后，事务失败的回滚机制如下：

- 同一个事务内，若匿名块内的BULKLOAD提示项插入语句执行成功但其后续其他语句执行失败，会使整个事务回滚并抛出相应的报错信息。
- 同一个事务内，执行BULKLOAD提示项插入语句后再执行匿名块，若匿名块执行失败，会使整个事务回滚并抛出相应的报错信息。
- 同一个事务内，执行BULKLOAD提示项插入语句后，若因语法错误、校验错误或部分资源分配失败而导致非匿名块语句执行失败，会使该失败语句回滚；若因其他原因导致某条语句执行失败，会使整个事务回滚并抛出相应的报错信息。

示例（LSC表）

```
--开启自动提交后，使用BULKLOAD提示项进行插入，提示错误
SET AUTOCOMMIT ON;
INSERT /*+ BULKLOAD */ INTO area  VALUES ('06','华东','Shanghai');
YAS-03728 cannot execute bulkload if autocommit is on

SET AUTOCOMMIT OFF;
--事务内，area表使用BULKLOAD提示项进行插入后，再对department表使用BULKLOAD提示项进行插入，提示报错
INSERT /*+ BULKLOAD */ INTO area  VALUES ('06','华东','Shanghai');
INSERT /*+ BULKLOAD */ INTO department VALUES ('002','财务部');
YAS-03728 cannot execute bulkload if bulkloading table has changed within the transaction

--由上条语句对department表使用BULKLOAD提示项进行插入提示报错后，只会回滚当前语句，执行commit再查area表可以查到('06','华东','Shanghai')这条插入
COMMIT;
SELECT * FROM area;
AREA_NO AREA_NAME        DHQ
------- ---------------- -------------
```

```
01      华东          Shanghai
02      华西          Chengdu
03      华南          Guangzhou
04      华北          Beijing
05      华中          Wuhan
06      华东          Shanghai
```

建议在导入期间无需进行查询和其他DML操作的场景中，使用BULKLOAD提示项进行插入，例如增量批量导入。

deduplicate提示项

使用DEDUPLICATE提示项时，无需指定表名，该提示项仅适用于LSC表，且只能与BULKLOAD提示项同时使用。表示对有唯一约束、主键约束的LSC表进行批量插入时，若插入数据与已有数据冲突，则进行去重处理，将冲突的数据对应的行替换成新插入的行。

示例（LSC表）

```
--未使用DEDUPLICATE提示项时，插入重复数据报错违反唯一约束
INSERT /*+ BULKLOAD */ INTO area  VALUES ('05','华南','Shenzhen');
YAS-02030 unique constraint violated
--使用DEDUPLICATE提示项后，插入重复数据成功，查询表的AREA_NO='05'的数据已经被替换成新插入的数据
INSERT /*+ BULKLOAD DEDUPLICATE */ INTO area  VALUES ('05','华南','Shenzhen');
SELECT * FROM area WHERE AREA_NO='05';
AREA_NO AREA_NAME        DHQ
------- ---------------- -------------
05      华南          Shenzhen
```

max_workers_per_exec提示项

使用MAX_WORKERS_PER_EXEC提示项时，无需指定表名，该提示项仅适用于分布式数据库的TAC表和LSC表的查询和子查询语句，指定可以同时执行的stage的数量。

示例（分布式LSC表、TAC表）

```
SELECT /*+ max_workers_per_exec(2) */*
FROM orders_info o, sales_info s,finance_info f
WHERE o.salesperson = s.salsperson
AND s.branch = f.branch;
```

json

JSON是一种独立于编程语言的数据交换格式，采用文本格式来表述数据，对于数据库系统，支持JSON格式可以使得应用和数据库之间传递数据变得更简洁、易懂和有结构逻辑。

YashanDB支持使用标准的JSON格式来定义和计算json数据，包括其两种结构形式和六种数据类型；还支持使用JSON扩展格式，扩展支持数据库中的其它数据类型。

关于JSON的标准语法描述详见其官方网站ECMA-404 The JSON Data Interchange Standard 。

JSON的结构形式

键值对形式

这种结构的json数据是一组无序的"键(Key):值(Value)"的集合，该集合通过`{`开始，以右括号`}`结束，多个"键(Key):值(Value)"之间使用`,`分割。如下所示：

```
{
    "key1": "abc",
    "key2": 123,
    "key3": true,
    "key4": null
}
```

数组形式

这种结构的json数据是一组有序的值（Value）的集合，该集合通过`[`开始，以`]`结束，Value之间使用逗号`,`分割。如下所示：

```
[
    "ghi",
    789
]
```

多结构嵌套混合

一个json数据可以由多个结构形式嵌套混合组成，以满足各种业务数据组织要求。如下所示：

```
{
    "key1": "abc",
    "key2": 123,
    "key3": true,
    "key4": null,
    "key5": [
        "def",
        456,
        false,
        null,
        [
            "ghi",
            789
        ],
        {
            "key": "value"
        }
    ],
    "key6": {
        "key": "value"
    }
}
```

JSON的数据类型

根据标准JSON格式定义，json数据里的Value可以为如下类型：

- 对象（Object）
- 数组（Array）
- 字符串（String）
- 数字（Number）
- 布尔值（True/False）
- 空值（Null）

其中：

1.对象（Object）和数组（Array）中的值（Value）可以为字符串（String）、数字（Number）、布尔值（True/False）、空值（Null）或者嵌套的对象（Object）和数组（Array）。

2.字符串（String）是由零个或者更多个Unicode字符组成的序列，前后使用双引号`"`来标识，且支持使用反斜杠`\`进行转义。

3.布尔值和空值必须采用全小写输入，即true/false/null。

JSON扩展格式

在JSON扩展格式里，json数据里的Value还可以为如下类型：

- Tinyint
- Smallint
- Integer
- Bigint
- Float
- Double
- Number
- Binary
- Timestamp
- Date
- Time

JSON扩展格式解析

目前支持的所有扩展格式可以用一个仅包含一个键值对（key-value pair）来表示，格式为：`{"$key": value}`。

其中：

1. Tinyint、Smallint、Integer、Bigint、Float、Double与Number：
   - 这些类型的值需要为一个字符串，或不带引号的数值。两种情况都需要符合此类型的解析要求。
     - 对于整数类型（Tinyint、Smallint、Integer、Bigint），值需要为一个仅包含数字与正负号的字符串，不允许出现小数点或幂。
     - 对于浮点类型（Float、Double），值允许为"nan"或带正负号的"inf"（不分大小写）或者一个数值字符串。
   - 这些类型与YashanDB数据库里同名的数据类型等价，包括其类型长度与取值范围。详见数据类型规格。
   - 这些类型的键分别为：
     - Tinyint的键为`$numberByte`。
     - Smallint的键为`$numberShort`。
     - Integer的键为`$numberInteger`。
     - Bigint的键为`$numberLong`。
     - Float的键为`$numberFloat`。
     - Double的键为`$numberDouble`。
     - Number的键为`$numberDecimal`。
2. Binary：
   - 这个类型的键为`$binary`。
   - 这个类型的值可以是一个经过base64编码的字符串（String），也可以是有且仅有如下两个键的一个对象：
     - `base64`对应的值为一个经过base64编码的字符串；
     - `subType`对应的值为一个Tinyint数字，用于语法兼容，无实际意义。
   - 上述经过base64编码的字符串可以省略掉末尾的填充符`=`。
   - 若这个字符串长度不合法（4的倍数+1），最后一个字符将被忽略。
3. Timestamp、Date与Time：

- Timestamp：
  - 键为 `$yashanTimestamp`。
  - 值为满足ISO-8601时间戳标准要求的字符串（不支持时区）：`yyyy-mm-ddThh24:mi:ss.ff`。
    - 根据标准，字母 `T` 或者一个空格都可以匹配 `T`。
    - 根据标准，小数点 `.` 或者逗号 `,` 都可以匹配 `.`。
  - 其他日期格式详见日期时间型。
- Date：
  - 键为 `$yashanDate`。
  - 值为满足ISO-8601日期格式要求的字符串（不支持时区）：`yyyy-mm-ddThh24:mi:ss.ff`。
    - 由于Date精度为秒，毫秒数据（`.ff`，最高支持9位小数）将被忽略。
    - 其他格式与Timestamp相同。
- Time:
  - 键为 `$yashanTime`。
  - 值为满足ISO-8601时间格式要求的字符串：`Thh24:mi:ss.ff`。

在将一个对象与支持的扩展格式进行匹配时，仅当匹配完全成功时，此对象才会被解析成其对应的扩展格式数据，否则它仍然会被解析成一个对象。

常见的匹配错误包括：

- 在匹配整数类型时（Tinyint、Smallint、Integer、Bigint）：
  - 输入值包含不合法字符。
  - 输入值超出此类型可以表达的范围。
- 在匹配Number类型时，输入值为无限或非数字（NaN）；注意Number类型不支持此类值，但是Float和Double类型支持。
- 在匹配Binary类型时，输入的base64编码字符串不符合其编码要求，例如 `{"$binary": "«»"}`。
- 输入的字符串不满足解析要求的其他情况。

当处于扩展模式时，JSON函数还会进一步处理未显式指示类型的数值数据：

1. 若一个数值是整数，尝试将其用更小的数据格式表示，按Tinyint < Smallint < Integer < Bigint依次尝试。对于每个数据类型，若此数字在其能够表达的数值区间内，则将这个数值的数据类型解析成此数据类型。
2. 若一个数值是小数，将其解析为Double类型。
3. 在其他情况下，此数值仍将会以Number类型保存。

示例

```
SELECT JSON('{"$numberByte": "-128"}' EXTENDED) byte,
       JSON('{"$numberByte": "-129"}' EXTENDED) obj FROM DUAL;

BYTE                                                             OBJ
---------------------------------------------------------------- ----------------------------------------------------------------
----
-128                                                             {"$numberByte":"-129"}

SELECT JSON_QUERY(JSON('1'), '$.type()' WITH ARRAY WRAPPER) std,
       JSON_QUERY(JSON('1' EXTENDED), '$.type()' WITH ARRAY WRAPPER) ext FROM DUAL ;

STD                                                              EXT
---------------------------------------------------------------- ----------------------------------------------------------------
----
["number"]                                                       ["tinyint"]

SELECT JSON_QUERY(JSON('0.1'), '$.type()' WITH ARRAY WRAPPER) std,
       JSON_QUERY(JSON('0.1' EXTENDED), '$.type()' WITH ARRAY WRAPPER) ext FROM DUAL ;

STD                                                              EXT
---------------------------------------------------------------- ----------------------------------------------------------------
----
["number"]                                                       ["double"]
```

JSON扩展格式输出

JSON数据仅在通过使用 `JSON_SERIALIZE()` 函数的 `EXTENDED` 关键字时使用扩展模式进行输出。

其他情况下（例如客户端输出，或者调用不包含 `EXTENDED` 关键字的 `JSON_SERIALIZE()` 函数等等）都会使用标准模式进行输出。

在标准格式时，若JSON数据包含扩展类型，根据以下规则输出：

1. 当数据为Tinyint，Smallint，Integer，Bigint，Number时，直接输出其数值，输出规则与YashanDB中同名的类型的输出规则相同。
2. 当数据为Float与Double时：
   - 若此数值为有限数值，直接输出其数值，其中Float精度为9位有效数字，Double精度为17位有效数字。
   - 若此数值不为有限数值（NaN以及正负无穷），输出首字母大写的对应字符串：`"Nan"`，`"Inf"`，`"-Inf"`。
3. 当数据为Binary类型时，输出结果为一个代表此二进制数据的十六进制字符串。
4. 当数据为时间类型（Timestamp，Date，Time）时，输出结果为一个代表此时间类型的，符合ISO-8601格式的字符串。
   - Timestamp类型使用 `"yyyy-mm-ddThh24:mi:ss.ff"` 格式输出，其中 `T` 固定为字母T，精确度为毫秒，且小数点以及毫秒仅在不为0时输出。
   - Date类型使用 `"yyyy-mm-ddThh24:mi:ss"` 格式输出，其中 `T` 固定为字母T，精确度为秒。
   - Time类型使用 `"hh24:mi:ss.ff"` 格式输出，精确度为毫秒，且小数点以及毫秒仅在不为0时输出。

在扩展格式时，根据以下规则输出。为方便叙述，当说明输出是一个拥有唯一一个键值对的对象时，描述为输出“键”格式的对象，输出值为此键对应的值，例如 `{"键": 值}`。

1. 当数据为小整数类型（Tinyint，Smallint，Integer）类型时，直接输出其数值，输出规则与标准模式相同。
2. 当数据为Bigint或Number类型，且为不超过 `(-2^53, +2^53)` 区间的整数时，直接输出其数值。
3. 当数据为Bigint或Number类型，且为超过上述区间但不超过Bigint区间的整数时，输出 `"$numberLong"` 格式对象，值为其数值的字符串。
4. 当数据为Number类型且不满足上述条件时，输出 `"$numberDecimal"` 格式对象，值为其数值的字符串。
5. 当数据为Float类型时，输出 `"$numberFloat"` 格式对象：
   - 若为有限数值，值为其数值的字符串。
   - 若不为有限数值，值为其在标准格式下输出的字符串。
6. 当数据为Double类型时：
   - 若为有限数值，直接输出其数值，输出规则与标准模式相同。
   - 若不为有限数值，输出 `"$numberDouble"` 格式对象，值为其在标准格式下输出的字符串。
7. 当数据为Binary类型时，输出 `"$binary"` 格式对象，值为其二进制数据对应的base64编码。
8. 当数据为时间（Timestamp，Date，Time）类型时，输出扩展格式对象，值为其在标准格式下输出的字符串。
   - Timestamp类型的键为 `$yashanTimestamp`。
   - Date类型的键为 `$yashanDate`。
   - Time类型的键为 `$yashanTime`。

路径表达式

YashanDB提供了一些内置函数实现对一个json数据内部的搜索查找功能（例如检索某个Key是否存在于某个json数据中），这需要结合路径表达式（Path Expression）一起使用。

路径表达式有点类似于XML数据的XQuery或XPath表达式，其语法定义如下：

json_path**::=**

```
syntax::= "$" [nonfunction_steps] [function_step]
```

nonfunction_steps**::=**

```
syntax::= [object_step] [array_step] [descendent_step]
```

object_step**::=**

```
syntax::= "." ("*"|key)
```

array_step**::=**

```
syntax::= "[" ("*"|((array_index|(array_n "to" array_m)) {"," (array_index|(array_n "to" array_m))})) "]"
```

descendent_step**::=**

```
syntax::= ".." key
```

function_step::=

```
syntax::= "." item_method "()"
```

item_method::=

```
syntax::= "count"|"size"|"type"
```

路径表达式使用绝对路径，以符号`$`开始，其含义用于表述检索json数据时的匹配路径。`$`后面跟着零个或者多个nonfunction_steps，以及一个可选的function_step，即支持多步骤匹配；不跟任何内容时，表示与json数据完全一致的匹配路径。

一个路径表达式会选择零个或者多个匹配的值。一般情况下，路径表达式会依次尝试匹配路径表达式中的每一个步骤。如果当前步骤匹配失败，则不会尝试匹配后续步骤，并且路径表达式匹配失败。如果每一个步骤都匹配成功，则路径表达式匹配成功。

其中路径表达式中的key在无双引号的情况下只允许字母数字，不允许任何其余字符出现，需要使用特殊字符时需要按标准json string语法在字符串的前后加上双引号。

示例

```
SELECT JSON_QUERY(JSON('{"1":1,"2":2,"3":3}'), '$') res FROM dual;
RES
----------------------------------------------------------------
{"1":1,"2":2,"3":3}
```

nonfunction_steps

非函数类匹配步骤，可以为object_step、array_step或者descendent_step中的某一个。

object_step

针对键值对结构的json数据的匹配步骤，以`.`开始，后面跟着一个Key或者一个通配符`*`。

Key如果不使用双引号标识时，必须以大写或者小写字母a-z开头，且只能包含字母或者十进制数字（0-9），否则必须使用双引号进行标识。

Key可以为空，如果为空，则必须写成`""`的形式。

使用object_step匹配时，系统将按指定的Key检索json数据，并返回Key对应的Value。指定通配符时，返回所有的Value。

示例

```
SELECT JSON_QUERY(JSON('{"1":1,"2":2,"3":3}'), '$.*' WITH WRAPPER) res FROM dual;
RES
----------------------------------------------------------------
[1,2,3]

SELECT JSON_QUERY(JSON('{"1":1,"2":2,"3":3}'), '$.""') res FROM dual;
RES
----------------------------------------------------------------

SELECT JSON_QUERY(JSON('{"1":1,"2":2,"3":3}'), '$."1"') res FROM dual;
RES
----------------------------------------------------------------
1
```

array_step

针对数组结构的json数据的匹配步骤，以`[`开始，以`]`结束，不允许出现空的[]，[]中间的内容形式可以为：

- 通配符`*`：匹配所有Value。
- 数组索引位置array_index：按Value在数组中的位置匹配，可以同时指定多个位置匹配，以`,`分隔。数组位置为从0开始的整数，即数组的第一个元素的索引为0。也可以是`last - N`的形式，表示从数组尾部开始查找，例如last - 1表示索引数组中倒数第二个元素。单独的`last`等价于`last - 0`，

表示索引数组中最后一个元素。

- 数组索引范围array_n "to" array_m：其形式为 `N to M` ，N和M都是位置索引，按Value在数组中的位置范围进行匹配，等价于显式地指定从N到M的所有索引位置，例如[2 to 4]等价于[2, 3, 4]。其中N和M的大小顺序没有特殊要求，例如，表示第三到第六个元素的索引范围可以是[2 to 5]，也可以是[5 to 2]。需要注意的是 `[N to N]` 等价于 `[N]` 而不是 `[N, N]` 。

使用array_step匹配时，上述三种形式必须至少出现一个，但是通配符只能独立出现。

当数组索引位置和数组索引范围可以出现多个，当多个索引位置存在重叠时，重叠区域中的元素会被多次匹配到。

如果指定的索引位置或者索引范围超过了数组本身的范围，则系统将认为没有匹配到数据，而不会报错。

示例

```
SELECT JSON_QUERY(JSON('[1,2,3,4,5,6,7,8,9]'), '$[0 to 2, 5 to 3, last to last - 2, 1, 1]' WITH WRAPPER) res FROM dual;
RES
--------------------------------------------------------------
[1,2,3,4,5,6,7,8,9,2,2]
```

descendent_step

针对键值对结构的json数据的递归匹配步骤，以两个连续的句号 `..` 开始，后面跟着一个Key。它基于当前步骤之前匹配到的数据，进行递归查找，返回所有匹配到的值。

示例

```
SELECT JSON_QUERY(JSON('{ "a" : { "b" : { "d" : 1 },
                                  "c" : [ 2, { "d" : 3 } ],
                                  "d" : 4 },
                          "d" : 5 }'), '$.a..d' WITH WRAPPER) res FROM dual;
RES
--------------------------------------------------------------
[4,1,3]
```

function_step

函数类匹配步骤，function_step在路径表达式中不是必须出现的，如果出现，它只能是路径表达式的最后一步。function_step以 `.` 开始，后面跟着函数方法名称，及左右括号 `()` ，括号之间可以有空格。

item_method

函数方法名称，函数方法应用于当前步骤之前匹配到的数据，对其进行数据转换，包括如下几种：

- count：返回当前步骤之前匹配到的数据的数量。
- size：如果当前步骤之前匹配到的数据为数组，返回数组的元素数量，如果不是数组，返回NULL。
- type：返回当前步骤之前匹配到的数据的类型。

示例

```
SELECT JSON_QUERY(JSON('{ "a" : { "b" : { "d" : 1 },
                                  "c" : [ 2, { "d" : 3 } ],
                                  "d" : 4 },
                          "d" : 5 }'), '$.a..d.count()' WITH WRAPPER) res FROM dual;
RES
--------------------------------------------------------------
[3]
```

宽松匹配

在严格匹配的模式下，根据路径表达式对json数据进行检索时，路径表达式中的某个匹配步骤只能应用于特定的数据类型，具体要求为：

- object_step只能对Object类型数据进行查找，应用于其他类型均认为匹配失败。
- array_step只能对Array类型数据进行查找，应用于其他类型均认为匹配失败。

为了提供更灵活的查询，YashanDB支持宽松匹配模式（系统默认为此种模式），具体规则为：

- 如果当前为array_step匹配，但json数据为非Array类型，则系统自动将json数据封装为长度为1的数组，然后进行匹配。

  示例

```
SELECT JSON_QUERY(JSON('1'), '$[0]') res FROM dual;
RES
--------------------------------------------------------------------
1
```

- 如果当前为非array_step匹配，如object_step、function_step等，但json数据为Array类型，则系统对Array中的每个元素应用当前的匹配路径。

  示例

```
SELECT JSON_QUERY(JSON('[{"a": 1}, {"a": 2}, {"a": 3}]'), '$.a' WITH WRAPPER) res FROM dual;
RES
--------------------------------------------------------------------
[1,2,3]
```

size_clause

size_clause::=

```
syntax::= integer [B|K|M|G|T|P|E]
```

size_clause用于定义存储容量单位，该语句在一些DDL语句中描述物理存储属性时可能被使用。YashanDB支持使用如下单位值进行存储容量的定义：

- B：bytes
- K：kilobytes
- M：megabytes
- G：gigabytes
- T：terabytes
- P：petabytes
- E：exabytes

storage_clause

storage_clause::=

```
syntax::= STORAGE "("
    (
        (
            INITIAL size_clause | NEXT size_clause | MINEXTENTS integer | (MAXEXTENTS (integer | UNLIMITED)) | maxsize_clause
| PCTINCREASE integer | FREELISTS integer |
            FREELIST GROUPS integer | BUFFER_POOL (KEEP | RECYCLE | DEFAULT) | FLASH_CACHE (KEEP|NONE|DEFAULT) |
CELL_FLASH_CACHE (KEEP | NONE| DEFAULT)
        )
        {" "
            (
                INITIAL size_clause | NEXT size_clause | MINEXTENTS integer | (MAXEXTENTS (integer | UNLIMITED)) |
maxsize_clause | PCTINCREASE integer | FREELISTS integer |
                FREELIST GROUPS integer | BUFFER_POOL (KEEP | RECYCLE | DEFAULT) | FLASH_CACHE (KEEP|NONE|DEFAULT) |
CELL_FLASH_CACHE (KEEP | NONE| DEFAULT)
            )}
    )
")"
```

size_clause/maxsize_clause

storage_clause用于对segment的属性进行描述，在创建表或索引等对象时可使用此语句，但在YashanDB中本语句无实际含义，只用作语法兼容。

示例（HEAP表，单机TAC表）

```
--创建分区表，为表和分区都指定storage属性
CREATE TABLE part_storage(a INT, b VARCHAR(4000))
PARTITION BY RANGE(a, b)
(
    PARTITION p1 VALUES LESS THAN(1, 'a') STORAGE(INITIAL 0 MAXSIZE 1M NEXT 0),
    PARTITION p2 VALUES LESS THAN(10, 'c') STORAGE(MINEXTENTS 1 MAXEXTENTS 10 PCTINCREASE 0),
    PARTITION p3 VALUES LESS THAN(MAXVALUE, MAXVALUE) STORAGE(INITIAL 0 MAXSIZE 1M NEXT 0 MINEXTENTS 1 MAXEXTENTS 10
PCTINCREASE 0 FREELIST GROUPS 20 BUFFER_POOL RECYCLE FLASH_CACHE KEEP CELL_FLASH_CACHE DEFAULT)
)
STORAGE(INITIAL 63K MAXSIZE 10M NEXT 12k MINEXTENTS 1 MAXEXTENTS 10 PCTINCREASE 0 FREELISTS 10);

--创建索引，为索引指定storage属性
CREATE INDEX idx_part_storage_2
ON part_storage(b)
STORAGE(MINEXTENTS 1 MAXEXTENTS 10 PCTINCREASE 0);
```

SQL语句

YashanDB定义了符合ANSI SQL标准的Structured Query Language（SQL）语句，可以将这些语句进行如下的通用分类：

DDL（Data Definition Language）

数据定义语言，用于操作数据库中的对象和对象属性，这种对象还包括数据库本身。YashanDB提供如下DDL语句：

ALTER DATABASE

ALTER DATABASE LINK

ALTER FUNCTION

ALTER INDEX

ALTER PACKAGE

ALTER PROCEDURE

ALTER PROFILE

ALTER SEQUENCE

ALTER SESSION

ALTER SYSTEM

ALTER TABLE

ALTER TABLESPACE

ALTER TABLESPACE SET

ALTER TRIGGER

ALTER TYPE

ALTER USER

ALTER MATERIALIZED VIEW

BACKUP DATABASE

BACKUP ARCHIVELOG

BUILD DATABASE

COMMENT

CREATE ACCESS CONSTRAINT

CREATE DATABASE

CREATE DATABASE LINK

CREATE FUNCTION

CREATE INDEX

CREATE LIBRARY

CREATE MATERIALIZED VIEW

CREATE PACKAGE

CREATE PROCEDURE

CREATE PROFILE

CREATE ROLE

CREATE SEQUENCE

CREATE SYNONYM

CREATE TABLE

CREATE TABLE AS

CRETAE TABLESPACE

CRETAE TABLESPACE SET

CREATE TRIGGER

CREATE TYPE BODY

CREATE TYPE

CREATE USER

CREATE VIEW

CREATE DIRECTORY

DROP ACCESS CONSTRAINT

DROP DATABASE

DROP DATABASE LINK

DROP FUNCTION

DROP INDEX

DROP LIBRARY

DROP MATERIALIZED VIEW

DROP PACKAGE

DROP PROCEDURE

DROP PROFILE

DROP ROLE

DROP SEQUENCE

DROP SYNONYM

DROP TABLE

DROP TABLESPACE

DROP TABLESPACE SET

DROP TRIGGER

DROP TYPE BODY

DROP TYPE

DROP USER

DROP VIEW

DROP DIRECTORY

RECOVER DATABASE

RESTORE DATABASE

RESTORE ARCHIVELOG

SHUTDOWN

TRUNCATE TABLE

DML（Data Manipulation Language）

数据操控语言，用于操作数据库中对象包含的数据，即记录。YashanDB提供如下DML语句：

SELECT

INSERT

UPDATE

DELETE

MERGE

DCL（Data Control Language）

数据控制语言，用于操作数据库中对象的权限，和对事务的控制。YashanDB提供如下DCL语句：

COMMIT

GRANT

RELEASE SAVEPOINT

REVOKE

ROLLBACK

SAVEPOINT

SET TRANSACTION

其他功能性SQL

标准和加强功能

CALL/EXEC

FLASHBACK

LOAD DATA

LOCK TABLE

PURGE

SHUTDOWN

审计相关

ALTER AUDIT POLICY

AUDIT POLICY

CREATE AUDIT POLICY

DROP AUDIT POLICY

NOAUDIT POLICY

性能相关

ALTER AUDIT POLICY

通用描述

ALTER AUDIT POLICY用于修改一个审计策略，对审计项目、审计执行条件、审计条件执行频率等进行增删操作。

只有拥有AUDIT_ADMIN审计管理员角色，或者拥有AUDIT SYSTEM系统权限的用户，才能修改一个审计策略。

对审计策略的修改立即生效，系统对数据库操作的审计项目相应发生改变。

语句定义

alter_audit_policy::=

```
syntax::= ALTER AUDIT POLICY policy_name (ADD [privilege_audit_clause] [action_audit_clause] [role_audit_clause]
[toplevel_clause] | DROP [privilege_audit_clause] [action_audit_clause] [role_audit_clause] [toplevel_clause] | CONDITION
condtion_clause)
```

privilege_audit_clause
action_audit_clause
role_audit_clause
toplevel_clause

condition_clause::=

```
syntax::= DROP | "'" audit_condition "'" EVALUATE PER (STATEMENT  | SESSION | INSTANCE)
```

policy_name

要修改的审计策略的名称。

add

为审计策略增加审计项目，审计项目的语法同CREATE AUDIT POLICY中定义。

drop

为审计策略删除审计项目，审计项目的语法同CREATE AUDIT POLICY中定义。

审计策略应该最少包含一个审计项目，违反此规则的DROP操作将会失败。

condition

为审计策略删除或增加执行判断条件。

DROP

删除在审计策略上定义的执行判断条件。

audit_condition

在审计策略上定义一个执行判断条件，语法同CREATE AUDIT POLICY中定义。

示例

```
-- 修改审计策略，增加对select any table权限的审计
ALTER AUDIT POLICY up2 ADD PRIVILEGES SELECT ANY TABLE;

-- 修改审计策略，增加对create table类型语句审计，删除对表area的delete、insert操作、对表branches所有操作的审计，删除审计条件
ALTER AUDIT POLICY up3
  ADD ACTIONS CREATE TABLE
  DROP ACTIONS DELETE ON sales.area, INSERT ON sales.area, ALL ON sales.branches
  CONDITION DROP;
```

```
-- 修改审计策略，删除对不审计内部语句的限制
ALTER AUDIT POLICY up4
  DROP ONLY TOPLEVEL;
```

ALTER DATABASE LINK

通用描述

ALTER DATABASE LINK语句用于修改一个数据库链接对象的信息。当前登录的本地用户可以修改数据库链接的远程用户名和用户密码。

分布式部署中不能执行本语句。

根据数据库系统的结构，支持两种数据库链接：

- 同构数据库链接：YashanDB与YashanDB的数据库链接
- 异构数据库链接：YashanDB与Oracle的数据库链接，需要进行异构数据库链接配置

语句定义

alter database link**::=**

```
syntax::= ALTER [PUBLIC] DATABASE LINK dblink_name [CONNECT TO username IDENTIFIED BY pwd_clause]
```

pwd_clause**::=**

```
syntax::= plaintext_password | VALUES ['""'] ciphertext_password ['""']
```

public

该语句修改一个公有数据库链接，对所有数据库用户可见。

修改公有数据库链接的用户须拥有ALTER PUBLIC DATABASE LINK系统权限，修改非公有数据库链接用户须拥有ALTER DATABASE LINK系统权限，否则返回错误。

示例（单机部署、共享集群部署）

```
ALTER PUBLIC DATABASE LINK dblink_yashan;
```

dblink_name

该语句用于指定创建的数据库链接的名称，不可省略，且需符合YashanDB的对象命名规范。

username

该语句用于指定访问远端数据库的用户名。

plaintext_password

该语句用于指定访问远端数据库的用户明文密码。

ciphertext_password

该语句用于指定访问远端数据库的用户密文密码。

示例（单机部署、共享集群部署）

```
-- YashanDB与YashanDB的数据库链接
ALTER PUBLIC DATABASE LINK dblink_yashan CONNECT TO REGRESS IDENTIFIED BY REGRESS;

-- YashanDB与Oracle的数据库链接
ALTER PUBLIC DATABASE LINK dblink_oracle CONNECT TO C##REGRESS IDENTIFIED BY REGRESS;
```

ALTER DATABASE

通用描述

ALTER DATABASE用于修改数据库的相关属性。 分布式部署中，仅支持如下语法：

- startup_clauses
- convert filename [including archivelog]
- archivelog/noarchivelog
- set standby database to
- convert to physical standby
- switchover
- failover
- exit upgrade

推荐使用yasboot运维工具管理分布式集群，具体见yasboot命令介绍章节描述。

语句定义

alter database::=

```
syntax::= ALTER DATABASE
(startup_clauses|database_file_clauses|logfile_clauses|standby_database_clauses|upgrade_clauses|repair_database_clauses|delet
e_archivelog_clauses|double_write_file_clauses|supplemental_log_clauses)
```

startup_clauses**::=**

```
syntax::= MOUNT|OPEN [READWRITE|RESETLOGS|UPGRADE]
```

database_file_clauses**::=**

```
syntax::= DATAFILE filename ( (AUTOEXTEND (OFF|ON [NEXT size_clauses] [MAXSIZE (UNLIMITED|size_clause)])) | (RESIZE
size_clause) | (OFFLINE [DROP]))

syntax::= TEMPFILE filename ( (AUTOEXTEND (OFF|ON [NEXT size_clauses] [MAXSIZE (UNLIMITED|size_clause)])) | (RESIZE
size_clause))

syntax::= CONVERT FILENAME [INCLUDING ARCHIVELOG]
```

logfile_clauses**::=**

```
syntax::= (ARCHIVELOG|NOARCHIVELOG)|
    (SET STANDBY DATABASE TO MAXIMIZE (PERFORMANCE|PROTECTION|AVAILABILITY) [FORCE][TIMEOUT integer])|
    (ADD LOGFILE "(" (filename SIZE size_caluse [BLOCKSIZE size_clauses] [PARALLEL parallel]) {"," (filename SIZE size_caluse
[BLOCKSIZE size_clauses] [PARALLEL parallel])}")")|
    (DROP LOGFILE filename)
```

standby_database_clauses**::=**

```
syntax::= CONVERT TO PHYSICAL STANDBY|SWITCHOVER|FAILOVER|(RECOVER MANAGED STANDBY DATABASE (([UNTIL SCN integer][DISCONNECT
FROM SESSION])|CANCEL))|(REGISTER [OR REPLACE] ARCHIVELOG {"," filename})
```

upgrade_clauses**::=**

```
syntax::= EXIT UPGRADE
```

repair_database_clauses**::=**

```
syntax::= CONVERT TO NORMAL
```

delete_archivelog_clauses**::=**

```
syntax::= DELETE ARCHIVELOG (ALL|UNTIL ((SEQUENCE integer [THREAD integer])|TIME date|SCN integer))[FORCE]
```

double_write_file_clauses**::=**

```
syntax::= DOUBLE_WRITE RESIZE FILE size_clauses
```

supplemental_log_clauses**::=**

```
syntax::= (ADD|DROP) SUPPLEMENTAL LOG (DATA ["(" (ALL|PRIMARY KEY) {"," (ALL|PRIMARY KEY)}")" COLUMNS] | TABLE TYPE "("
(HEAP|TAC|LSC) {"," ( HEAP|TAC|LSC)}")")
```

startup_clauses

该语句用于MOUNT和OPEN数据库，以便用户访问。

MOUNT是数据库挂载物理文件后的状态，OPEN是数据库打开后的状态，详细数据库启动介绍请参考实例启停章节。

YashanDB的OPEN方式分为：

- READWRITE：以读写模式打开数据库，单机部署、共享集群部署和分布式部署下默认的数据库打开方式，禁止备库使用该方式打开数据库。
- RESETLOGS：打开数据库时重置redo时间线，并丢弃在恢复期间未应用的redo信息，从而确保永远不会应用这些信息。
- UPGRADE：仅当数据库需要升级时才会使用该方式打开数据库。

示例（单机、共享集群部署）

```
ALTER DATABASE MOUNT;

ALTER DATABASE OPEN;

ALTER DATABASE OPEN UPGRADE;
```

database_file_clauses

该语句用于对数据库的数据文件进行自动扩展的开关控制、大小指定等。此操作需要数据库处于OPEN状态。 在设置数据文件自动扩展或RESIZE数据文件时，对于TEMP表空间和SWAP表空间的数据文件使用tempfile选项，其它表空间的数据文件使用datafile选项。

autoextend off

关闭某个数据文件的自动扩展，同时该数据文件的NEXT_SIZE和MAX_SIZE被设置为0。

示例（单机、共享集群部署）

```
ALTER DATABASE DATAFILE '/home/yasdb/YASDB_DATA/dbfiles/users' AUTOEXTEND OFF;
ALTER DATABASE TEMPFILE '/home/yasdb/YASDB_DATA/dbfiles/swap' AUTOEXTEND OFF;
```

autoextend on

开启某个数据文件的自动扩展，同时NEXT用于指定自动扩展下一空间的大小，以Bytes为单位，未指定则取默认值8K个BLOCK大小；MAXSIZE用于指定自动扩展的最大空间，UNLIMITED为无限制，未指定则默认为64M个BLOCK大小。

示例（单机、共享集群部署）

```
ALTER DATABASE DATAFILE '/home/yasdb/YASDB_DATA/dbfiles/users' AUTOEXTEND ON NEXT 8M MAXSIZE 64M;
ALTER DATABASE TEMPFILE '/home/yasdb/YASDB_DATA/dbfiles/swap' AUTOEXTEND ON NEXT 8M MAXSIZE 64M;
```

resize

重新指定某个数据文件的大小。

创建表空间下的数据文件时可以通过设定数据文件自动扩展来满足用户对于更大数据文件的需求，但如果存储空间并不充足，或者数据文件按预期指定过大而实际使用很小，或者数据文件中存在大量被删除的临时数据而空间并没有被回收，这些情况下就需要使用resize的方式对数据文件进行缩小，释放磁盘空间。

使用resize对数据文件进行扩大，一次设置文件为自己所需大小 ，也可以避免频繁的申请资源。

resize后的数据文件大小必须在128到64MB个BLOCK之间。

resize操作不一定会成功，例如在扩大文件时可能会存在磁盘空间不足，或者在缩小文件时，当前数据文件的有效数据大小已超过指定的值。

注意：UNDO表空间的数据文件只能进行扩大，不能进行缩小。

示例（单机、共享集群部署）

```
ALTER DATABASE DATAFILE '/home/yasdb/YASDB_DATA/dbfiles/users' RESIZE 1048576;
ALTER DATABASE TEMPFILE '/home/yasdb/YASDB_DATA/dbfiles/temp' RESIZE 1048576;
```

offline [drop]

改变某个数据文件至offline状态，可以在数据库处于MOUNT或者OPEN时执行该操作。 执行此操作后该数据文件所属表空间也会被offline，表空间内的所有数据文件的状态都是recover。

执行offline的场景：

- 存在数据文件损坏或丢失导致无法打开数据库，此类场景只能在数据库处于MOUNT时执行offline。
- 表空间的数据文件在有数据时无法被删除，但不再需要使用该数据文件。
- 需要数据隔离，临时将某个数据文件offline。

执行offline的说明：

- 尽量在数据库处于OPEN状态时执行此操作，除非存在数据文件损坏或丢失导致无法打开数据库。
- 该数据文件后续可能会被online且仍属于数据库，因此仍需要对其进行备份，但如果该数据文件名称被修改，会由于备份时找不到数据文件而导致备份失败。
- 发现数据库为abnormal状态时，查询V$DIAG_INCIDENT排查是否因文件损坏导致abnormal，如果是则可以将该数据文件offline。
- 在数据库处于MOUNT状态时执行此操作的额外说明：
  - 某些情况下（例如open过程中回放了创建该文件所属表空间的redo日志），数据库会修复已被此操作offline的数据文件和其所属表空间，并将它们置为online。
  - 文件被offline，然后打开数据库后，查询V$DATAFILE将无法正常显示该文件的创建时间，除非其被重新online。
  - 在备库上执行该操作，然后打开数据库后，备库将变成need repair状态。尽量在主库上执行该操作，这样可以通过redo日志同步使备库offline相同的文件。

执行offline的限制：

- 每次只能offline一个数据文件。
- 内置表空间的数据文件不能被offline。
- 数据文件被offline后不可读写，且不能对该文件进行resize或设置自动扩展开关等操作。
- 主库在mount和open状态均可执行该操作，备库只能在mount状态下执行。

drop

在数据库未开启归档模式时，执行数据文件offline必须指定此选项。在开启归档模式的情况下，此选项被忽略，即OFFLINE DROP = OFFLINE。

示例（单机、共享集群部署）

```
ALTER DATABASE DATAFILE '/home/yasdb/YASDB_DATA/dbfiles/file' OFFLINE;

--非归档模式下只能用如下语句offline
ALTER DATABASE DATAFILE '/home/yasdb/YASDB_DATA/dbfiles/file' OFFLINE DROP;
```

convert filename [including archivelog]

当数据库整库迁移到其它目录后，该语法可用于将数据库记录在控制文件中的路径进行转换，从而使数据库继续正常启动、运行。该语句仅在实例处于NOMOUNT阶段时使用。

该语句的具体使用步骤如下：

1. 更新启动路径：修改配置参数文件中的控制文件路径参数CONTROL_FILES为当前数据库启动路径，或删除该参数的已有配置从而使用默认配置启动。
2. （可选）如需更新归档日志存储路径：修改配置参数文件中的归档日志路径参数ARCHIVE_LOCAL_DEST为当前数据库指定归档路径，或删除该参数的已有配置从而使用默认配置存储归档日志。
3. 配置数据文件路径转换参数DB_FILE_NAME_CONVERT，将旧的文件路径转换为新的文件路径。同时，需确认数据库是否需要开启双写，如需开启双写，请确保数据文件转换路径参数可以对双写文件路径生效。
4. 配置在线日志文件的路径转换参数REDO_FILE_NAME_CONVERT，将旧的文件路径转换为新的文件路径。
5. 配置DATABUCKET的路径转换参数DB_BUCKET_NAME_CONVERT，将旧的文件路径转换为新的文件路径。
6. 启动数据库至NOMOUNT状态，并根据是否需要转换归档日志路径从而决定是否显式指定子句INCLUDING ARCHIVELOG。
7. 启动数据库至OPEN状态。

including archivelog

当需要转换归档日志路径时指定该子句。

示例（单机、共享集群部署）

```
ALTER DATABASE CONVERT FILENAME;

包含归档日志的路径转换使用如下语句
ALTER DATABASE CONVERT FILENAME INCLUDING ARCHIVELOG;
```

logfile_clauses

该语句用于对数据库的redo日志进行归档模式设置、备库保护模式设置、增加、删除等操作。

archivelog/noarchivelog

启用或停止数据库的日志归档模式。

单机部署中，此操作需要数据库实例处于MOUNT状态。

共享集群部署中，此操作需要当前所在实例处于MOUNT状态，且其他实例处于NOMOUNT状态。

当数据库处于主备复制模式（单机主备部署或[主备共享集群部署](../../../高可用/主备集群部署/00主备集群部署.md）时，无法从归档模式切换为非归档模式。

示例（单机、共享集群部署）

```
ALTER DATABASE ARCHIVELOG;

ALTER DATABASE NOARCHIVELOG;
```

set standby database to

指定备库的保护模式，缺省值为MAXIMIZE PERFORMANCE。其中，设置为MAXIMIZE PROTECTION的前提主库的日志已经同步到备库，否则会报错。FORCE表示强制设置，忽略报错。TIMEOUT表示在设置最大保护模式时，等待备库同步的时间，超过该时间将报错，单位为秒，可省略，默认为10s。

- MAXIMIZE PERFORMANCE：主库事务提交不需要等待备库收到日志，保证了主数据库的可用性及性能，但是主库宕机后，可能丢失数据。
- MAXIMIZE PROTECTION：备库的数据保护优先于主库的可用性，主库的日志在同步备库上落盘之后，事务才能提交，在同步备库故障的情况下，主库会在一段时间后变为只读模式。注：如果COMMIT_WAIT参数设为NOWAIT，主库事务提交不会等待备库收到日志
- MAXIMIZE AVAILABILITY：同步备库正常时，主库的日志在同步备库上落盘之后，事务才能提交，同步备库故障时，事务提交也不会阻塞，保证数据库可用。注：如果COMMIT_WAIT参数设为NOWAIT，主库事务提交不会等待备库收到日志

示例（单机、共享集群部署）

```
ALTER DATABASE SET STANDBY DATABASE TO MAXIMIZE PERFORMANCE;

ALTER DATABASE SET STANDBY DATABASE TO MAXIMIZE PROTECTION;

ALTER DATABASE SET STANDBY DATABASE TO MAXIMIZE AVAILABILITY;

ALTER DATABASE SET STANDBY DATABASE TO MAXIMIZE PROTECTION FORCE;

ALTER DATABASE SET STANDBY DATABASE TO MAXIMIZE PROTECTION TIMEOUT 100;
```

add logfile

为数据库增加新的redo日志，同时增加多个文件以','隔开。此操作需要数据库处于OPEN状态。

SIZE指新增redo日志文件大小。BLOCKSIZE指redo日志文件的块大小，默认为4096，可手动指定为512的整数。redo日志实际创建的文件大小是BLOCKSIZE的整数倍，若指定大小不是整数倍，则实际创建大小向上取整。 PARALLEL可以指定创建redo日志文件的并行度，取值范围为1到8。不指定时系统根据文件大小自适应并行度，例如文件不超过1G时的并行度为1，文件超过128G时的并行度为8，文件大小在1G到128G之间时的并行度为4。

注：redo日志文件大小的最小值受DB_BLOCK_SIZE，MAX_SESSIONS和REDO_BUFFER_SIZE三个参数的影响（最小值参考公式：DB_BLOCK_SIZE * MAX_SESSIONS * 8 + REDO_BUFFER_SIZE / 2）。

示例（单机、共享集群部署）

```
ALTER DATABASE ADD LOGFILE ('/home/yasdb/YASDB_DATA/dbfiles/redo5' SIZE 72355840,'/home/yasdb/YASDB_DATA/dbfiles/redo6' SIZE
72355840);
ALTER DATABASE ADD LOGFILE '/home/yasdb/YASDB_DATA/dbfiles/redo5' SIZE 72355840 BLOCKSIZE 512;
ALTER DATABASE ADD LOGFILE '/home/yasdb/YASDB_DATA/dbfiles/redo6' SIZE 72355840 PARALLEL 4;
```

drop logfile

删除一个已存在的redo日志，对于正在使用中的redo日志则不被允许删除。此操作需要数据库处于OPEN状态。

示例（单机、共享集群部署）

```
ALTER DATABASE DROP LOGFILE '/home/yasdb/YASDB_DATA/dbfiles/redo5';
```

standby_database_clauses

该语句用于执行主备库之间的切换，此操作需要数据库处于MOUNT状态。主备库详细操作描述请参考高可用手册。

convert to physical standby

从主数据库切换为备数据库。

示例（单机、共享集群部署）

```
ALTER DATABASE CONVERT TO PHYSICAL STANDBY;
```

switchover

从备库切换回主数据库。

示例（单机、共享集群部署）

```
ALTER DATABASE SWITCHOVER;
```

failover

当主库出现故障不能恢复时，将备库强制切换为主数据库。

示例（单机、共享集群部署）

```
ALTER DATABASE FAILOVER;
```

recover managed standby database cancel

在备库并且为Open状态下执行的SQL语句，该语句的作用为停止当前回放操作。 如果有前台回放正在进行，则中断该线程使其退出。

示例（单机、共享集群部署）

```
ALTER DATABASE RECOVER MANAGED STANDBY DATABASE CANCEL;
```

recover managed standby database

在备库并且为Open状态下执行的SQL语句，该语句的作用为启动备库回放，如需退出，需要执行取消回放SQL语句。

示例（单机、共享集群部署）

```
ALTER DATABASE RECOVER MANAGED STANDBY DATABASE;
```

until scn integer

在备库并且为Open状态下执行的SQL语句，该语句的作用为启动备库回放，回放到指定到SCN时主动退出。 此时如需退出则需要执行取消回放SQL语句进行中断操作。

示例（单机、共享集群部署）

```
ALTER DATABASE RECOVER MANAGED STANDBY DATABASE UNTIL SCN 123123123;
```

disconnect from session

在备库并且为Open状态下执行的SQL语句，该语句的作用为在后台执行回放，当前会话可执行其他业务。

示例（单机、共享集群部署）

```
ALTER DATABASE RECOVER MANAGED STANDBY DATABASE DISCONNECT FROM SESSION;
ALTER DATABASE RECOVER MANAGED STANDBY DATABASE UNTIL SCN 123123123 DISCONNECT FROM SESSION;
```

register archivelog

该语句用于手动注册归档。该SQL的功能约束有：

- RESTORE DATABASE后且数据库未open，可以用该SQL手动注册归档。
- 数据库恢复或创建完整后，此操作的对象必须是备库，并且配置参数SANDBOX_STANDBY为TRUE。
- 指定的归档的路径可以为绝对路径，也可以为文件名，使用文件名时默认路径为归档路径（配置参数ARCHIVE_LOCAL_DEST）。

示例（单机、共享集群部署）

```
ALTER DATABASE REGISTER ARCHIVELOG '/home/yashan/archive/arch_0_1.ARC';
ALTER DATABASE REGISTER ARCHIVELOG '/home/yashan/archive/arch_0_1.ARC', '/home/yashan/archive/arch_0_2.ARC';
ALTER DATABASE REGISTER ARCHIVELOG 'arch_0_1.ARC';
ALTER DATABASE REGISTER ARCHIVELOG 'arch_0_1.ARC', 'arch_0_2.ARC';

-- 结合RESTORE DATABASE使用
RESTORE DATABASE FROM 'BAK1';
ALTER DATABASE REGISTER ARCHIVELOG 'arch_0_1.ARC', 'arch_0_2.ARC';
RECOVER DATABASE;
ALTER DATABASE OPEN;
```

or replace

如果发现归档已经注册，就替换原有注册的归档，该操作比较危险，需要谨慎使用。

示例（单机、共享集群部署）

```
ALTER DATABASE REGISTER OR REPLACE ARCHIVELOG '/home/yashan/archive/arch_0_1.ARC';
ALTER DATABASE REGISTER OR REPLACE ARCHIVELOG '/home/yashan/archive/arch_0_1.ARC', '/home/yashan/archive/arch_0_2.ARC';
ALTER DATABASE REGISTER OR REPLACE ARCHIVELOG 'arch_0_1.ARC';
ALTER DATABASE REGISTER OR REPLACE ARCHIVELOG 'arch_0_1.ARC', 'arch_0_2.ARC';
```

upgrade_clauses

该语句用于数据库的版本升级。

exit upgrade

当数据库升级完成之后，可以直接退出升级模式进入正常OPEN模式，无需重启。

示例（单机、共享集群部署）

```
ALTER DATABASE EXIT UPGRADE;
```

repair_database_clauses

该语句用于数据库处于ABNORMAL状态时，DBA介入修复。

convert to normal

当数据库出现故障时，数据库被设为只读，数据库为故障状态，DBA修复之后，可以通过本语句将数据库手动切换为正常模式。

> **Note**：
> 当数据库因为资源错误陷入异常状态时，无法使用此语句将数据库状态置为正常。

示例（单机、共享集群部署）

```
ALTER DATABASE CONVERT TO NORMAL;
```

delete_archivelog_clauses

该语句用于对数据库的归档文件进行手动清理，释放磁盘空间。

手动清理归档日志的筛选条件由ARCH_CLEAN_IGNORE_MODE参数决定，清理条件的具体描述见归档管理章节。

delete archivelog

all

清理掉满足清理条件的所有归档。

until sequence integer [thread integer]

清理指定序列号之前满足清理条件的归档。若不指定实例，则默认清理实例1的归档。

until time date

清理指定时间之前生成的并且满足清理条件的归档。

until scn integer

清理指定SCN之前生成的并且满足清理条件的归档。（同V$ARCHIVED_LOG中的NEXT_CHANGE#作比较）。

force

不考虑清理条件，强制清理归档。

示例（单机、共享集群部署）

```
ALTER DATABASE DELETE ARCHIVELOG ALL;

ALTER DATABASE DELETE ARCHIVELOG UNTIL SEQUENCE 5;

ALTER DATABASE DELETE ARCHIVELOG UNTIL TIME TO_DATE('2022-06-01 18:00:00', 'yyyy-mm-dd hh24:mi:ss');
```

```
--强制归档清理
ALTER DATABASE DELETE ARCHIVELOG ALL FORCE;

ALTER DATABASE DELETE ARCHIVELOG UNTIL SEQUENCE 5 FORCE;

ALTER DATABASE DELETE ARCHIVELOG UNTIL TIME TO_DATE('2022-06-01 18:00:00', 'yyyy-mm-dd hh24:mi:ss') FORCE;
```

double_write_file_clauses

该语句用于重新指定双写文件的大小。

示例（单机、共享集群部署）

```
ALTER DATABASE DOUBLE_WRITE RESIZE FILE 32M;
```

supplemental_log_clauses

该语句用于配置数据库级别的附加日志，仅适用于单机部署。

开启附加日志后，数据库将在redo里额外记录一些数据，这些数据包括DDL的原始SQL文本，update，delete时用于定位行位置的索引信息等。 结合附加日志，可以通过redo解析还原出对应的DDL，DML语句，通常用于异构数据库同步。

数据库级别的附加日志对选定类型的所有用户表生效，可以通过动态视图V$DATABASE查看数据库级别的附加日志生效状态。表级附加日志请参考ALTER TABLE章节。

supplemental log data

SUPPLEMENTAL LOG DATA表示最小附加日志，这种模式下redo里会额外记录DDL文本和DML的rowid，性能影响最小。 当开启ALL或PRIMARY KEY模式的附加日志时，最小附加日志隐式开启。且在关闭其他模式的附加日志前，不能关闭最小附加日志。 ALL模式的优先级大于PRIMARY KEY模式，当数据库级附加日志模式和表级附加日志模式中有一个为ALL时，该表使用ALL模式。

数据库级附加日志记录FUNCTION，PACKAGE，PROCEDURE，SEQUENCE，TRIGGER，SYNONYM，LIBRARY，TABLE，INDEX，TYPE，VIEW，MATERIALIZED VIEW这些对象的DDL，并且不修改元数据的DDL不会记录。

all

ALL模式下，redo里会额外记录DDL文本和DML的rowid，还会在update和delete时记录原行中的所有列（除了LOB，超过32K的varchar和char等）。 这种模式适用于没有主键的表，由于redo里额外记录了整行数据，性能影响较大。

primary key

PRIMARY KEY模式下，redo里会额外记录DDL文本和DML的rowid，还会在update和delete时尝试记录主键列。 如果表没有主键，但有非空的唯一索引，则会记录该索引列。 如果表既没有主键，也没有非空的唯一索引，则会记录原行中的所有列（除了LOB，超过32K的varchar和char等）。 这种模式下，优先记录主键，redo占用较少，性能影响较小。

示例（单机部署）

```
--开启最小附加日志
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA;

--开启PRIMARY KEY模式的附加日志
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA (PRIMARY KEY) COLUMNS;

--开启ALL模式的附加日志
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA (ALL) COLUMNS;

--关闭ALL模式和PRIMARY KEY模式的附加日志
ALTER DATABASE DROP SUPPLEMENTAL LOG DATA (ALL, PRIMARY KEY) COLUMNS;

--关闭最小附加日志
ALTER DATABASE DROP SUPPLEMENTAL LOG DATA;
```

supplemental log table type

设置数据库级别的DML附加日志对哪些类型的用户表生效。默认为空，在开启附加日志后，请同时设置需要生效的表类型。

LSC表的热数据转冷数据后，所涉及行的rowid可能会发生改变，不建议对LSC表使用rowid作为行的定位信息。 LSC表在导入或者通过create table as select创建时，不会产生insert的redo，所以不会记录insert的逻辑日志。

示例（单机部署）

```
--设置数据库级附加日志对HEAP和TAC类型的表生效
ALTER DATABASE ADD SUPPLEMENTAL LOG TABLE TYPE (HEAP, TAC);

--设置数据库级附加日志对LSC表不生效
ALTER DATABASE DROP SUPPLEMENTAL LOG TABLE TYPE (LSC);
```

Note：

- 修改数据库级附加日志状态，会对所有用户表加锁并失效数据字典，会对数据库产生一定的性能影响。请保证用户表锁不被长时间占有，以免操作超时。
- 如果修改数据库级附加日志因为表锁超时失败，建议排查表锁超时问题，并重新执行数据库级附加日志修改直到返回成功。

ALTER FUNCTION

通用描述

ALTER FUNCTION语句用于显式的重编译一个自定义函数。

对于在SYS schema内的自定义函数，需要由SYS用户执行ALTER FUNCTION语句。

对于其他schema内的自定义函数，需要由其所属用户或拥有ALTER ANY PROCEDURE权限的用户执行ALTER FUNCTION语句。

自定义函数含义及限制请参考自定义函数章节。

语句定义

alter_function::=

```
syntax::= ALTER FUNCTION [schema "."] function_name (function_compile_clause|(EDITIONABLE|NONEDITIONABLE))
```

schema

包含自定义函数的模式名称，省略则默认为当前登录用户的模式。

function_name

需要显式重编译的自定义函数名称。

function_compile_clause

指定重编译选项，详见compile_clause描述。

editionable | noneditionable

用于语法兼容，无实际含义。

示例（单机、共享集群部署）

```
ALTER FUNCTION sales.ya_func COMPILE;
```

ALTER INDEX

通用描述

ALTER INDEX用于修改索引的相关信息。

一个SQL命令行可以指定多项修改操作，以 `,` 分开；但对于存在分区索引的表，该表上的索引操作需单项执行，即不能放在同一个命令行中，且对于分区索引，其修改操作必须按指定分区进行，而不能针对整体索引操作。

语句定义

alter index::=

```
syntax::= ALTER INDEX [schema"."] index_name (INITRANS integer|VISIBLE|INVISIBLE|UNUSABLE|COALESCE|NOPARALLEL|PARALLEL
integer|NOLOGGING|LOGGING |modify_partition|modify_subpartition|rebuild_clause|RENAME TO new_name)
```

modify_partition**::=**

```
syntax::= MODIFY PARTITION partition_name (INITRANS integer|UNUSABLE|COALESCE)
```

modify_subpartition**::=**

```
syntax::= MODIFY SUBPARTITION subpartition_name (UNUSABLE|COALESCE)
```

rebuild_clause**::=**

```
syntax::= REBUILD [(PARTITION partition_name)|(SUBPARTITION subpartition_name)|(NOREVERSE|REVERSE)] [(TABLESPACE
tablespace_name|INITRANS integer|PCTFREE integer|ONLINE|(NOCOMPRESS|COMPRESS (integer))|(LOGGING|NOLOGGING)|
(NOPARALLEL|PARALLEL integer)) {" " (TABLESPACE tablespace_name|INITRANS integer|PCTFREE integer|ONLINE|(NOCOMPRESS|COMPRESS
(integer))|(LOGGING|NOLOGGING)|(NOPARALLEL|PARALLEL integer))}]
```

initrans

该语句用于修改索引的数据块（Data Block）里事务（Transaction）表的初始大小设置，修改成功后该索引后续增加的数据块将按照此值设置事务表初始大小。

示例（HEAP表、TAC表）

```
ALTER INDEX idx_sales_info_1 INITRANS 4;
```

visible|invisible

该语句用于设置索引对优化器（Optimizer）是否可见，即在执行SQL查询时优化器是否会考虑使用该索引。

示例（HEAP表、TAC表）

```
ALTER INDEX idx_sales_info_1 INVISIBLE;

ALTER INDEX idx_sales_info_1 VISIBLE;
```

unusable

该语句用于设置索引为不可用状态，之后对表操作不会触发该索引更新，因此如需要恢复该索引可用，则必须进行索引重建（REBUILD）。

示例（HEAP表、TAC表）

```
ALTER INDEX idx_sales_info_1 UNUSABLE;
```

coalesce

该语句用于重组索引，RTREE索引无法重组。

对于分区索引，只可对实体分区进行重组。

当BTree索引（尤其是单调递增索引）使用一段时间后，整个BTree上可能会出现大量的空页和稀疏页，造成空间浪费，为解决这一问题，YashanDB提供索引coalesce功能，包括对空页的回收及稀疏页的合并，但此功能并不会使BTree的树高度变低。

重组操作会对索引产生结构性变更，且会对索引基表加表级共享锁，因此建议在没有业务或者业务量很低时执行本操作。

示例（HEAP表、TAC表）

```
ALTER INDEX idx_finance_info_1 COALESCE;
```

noparallel|parallel

该语句用于语法兼容，无实际含义。

nocompress|compress

该语句用于语法兼容，无实际含义。

logging|nologging

该语句用于语法兼容，无实际含义。

modify_partition

该语句用于对索引的分区进行INITRANS/UNUSABLE/COALESCE等修改操作，对于分区索引，必须通过本语句进行指定的分区索引修改。

示例（HEAP表、TAC表）

```
--通过系统视图查看分区索引信息
SELECT index_name,partition_name,status
FROM USER_IND_PARTITIONS
WHERE index_name='IDX_SALES_INFO_1';
INDEX_NAME             PARTITION_NAME         STATUS
---------------------- ---------------------- ---------
IDX_SALES_INFO_1       IP_SALES_INFO_1        N/A
IDX_SALES_INFO_1       IP_SALES_INFO_2        N/A
IDX_SALES_INFO_1       IP_SALES_INFO_3        N/A

--对指定分区索引进行修改
ALTER INDEX idx_sales_info_1 MODIFY PARTITION ip_sales_info_1 UNUSABLE;
ALTER INDEX idx_sales_info_1 MODIFY SUBPARTITION isp_sales_info_11 COALESCE;
```

modify_subpartition

该语句用于对索引的二级分区进行INITRANS/UNUSABLE/COALESCE等修改操作，对于组合分区索引，必须通过本语句进行指定的二级分区索引修改。

示例（单机/共享集群部署）

```
--对指定分区索引进行修改
ALTER INDEX idx_sales_info_1 MODIFY SUBPARTITION isp_sales_info_11 UNUSABLE;
ALTER INDEX idx_sales_info_1 MODIFY SUBPARTITION isp_sales_info_21 COALESCE;
```

rebuild_clause

该语句用于对索引或者索引分区进行重建操作。

当索引使用了一段时间后，可能存在空间膨胀或占用空间过多的问题，此时可以通过rebuild功能进行重建，生成新的紧凑的BTree。此外，通过指定TABLESPACE参数，重建索引也可以实现索引表空间搬迁。

对于被设置UNUSABLE的索引，重建后该索引将恢复为有效状态。

对于分区索引的REBUILD操作，需在每个分区上单独执行。

本操作过程中将会在索引基表上加排他锁，但在指定了ONLINE时，本操作不会阻塞并发的DML操作。在线（ONLINE）重建索引的使用限制同在线（ONLINE）创建索引。

tablespace

重建索引或索引分区到指定的表空间。

重建分布表上的索引或索引分区时，不能指定表空间。

initrans

为重建的索引或索引分区指定INITRANS值。

pctfree

为重建的索引或索引分区指定PCTFREE值。

noparallel|parallel

设置重建索引的并行度，NOPARALLEL表示不并行。

integer

并行度值，取值范围[1,服务器CPU核数*2]，可省略，省略则默认按CPU核数的并行度并发创建索引。

示例（HEAP表，单机TAC表）

```
ALTER INDEX idx_finance_info_1 REBUILD PARALLEL 2;
```

nologging|logging

该语句用于语法兼容，无实际含义。

online

该语句用于指定重建索引过程中是否允许并发DML操作，省略则默认不允许，使用限制同CREATE INDEX。

示例（单机HEAP表、单机TAC表）

```
ALTER INDEX idx_finance_info_1 REBUILD TABLESPACE yashan INITRANS 3 PCTFREE 10 ONLINE;

ALTER INDEX idx_sales_info_1 REBUILD PARTITION isp_sales_info_11;
```

noreverse|reverse

重建索引为反向或非反向，使用限制同CREATE INDEX。

示例（HEAP表，单机TAC表）

```
ALTER INDEX idx_finance_info_1 REBUILD REVERSE;

ALTER INDEX idx_finance_info_1 REBUILD NOREVERSE;
```

rename to new_name

该语句用于指定修改索引的名称。

指定的新名称不能为空且必须符合YashanDB的对象命名规范。

示例

```
ALTER INDEX idx_sales_info_1 RENAME TO idx_sales_info_2;
```

ALTER MATERIALIZED VIEW

通用描述

ALTER MATERIALIZED VIEW语句用于更改一个已存在的物化视图的相关属性。

语句定义

alter materialized view::=

```
syntax::= ALTER MATERIALIZED VIEW [schema"."] materialized_view_name (alter_mv_refresh_properties |
alter_query_rewrite_clause )
```

alter_mv_refresh_properties::=

```
syntax::= REFRESH ([COMPLETE | FORCE]
| [ON DEMAND | ON COMMIT]
| [((START WITH date) | NEXT date)]){" " ([COMPLETE | FORCE]
| [ON DEMAND | ON COMMIT]
| [((START WITH date) | NEXT date)])}
| (NEVER REFRESH)
```

alter_query_rewrite_clause

```
syntax::= (ENABLE | DISABLE) QUERY REWRITE
```

materialized_view_name

该语句用于指定已存在的物化视图的名称。

alter_mv_refresh_properties

该语句用于更改物化视图刷新相关的属性，包括刷新类型、刷新模式、定时刷新等，多项间使用空格进行分隔。

complete|force

该语句用于指定物化视图的刷新类型，包括：

- COMPLETE：全量刷新
- FORCE：若条件满足，优先使用快速刷新；若快速刷新不可用，则使用全量刷新

刷新类型最多能够指定一项。

on demand|on commit

该语句用于指定物化视图的刷新模式，包括：

- ON DEMAND: 手动刷新，即DBMS_MVIEW高级包刷新
- ON COMMIT: 自动刷新，事务提交时刷新相关联的物化视图

刷新模式最多能够指定一项。

start with date|next date

该语句用于指定物化视图的定时刷新选项，包括：

- START WITH date：指定第一次刷新时间
- NEXT date：指定刷新时间间隔

定时刷新选项可以指定一项或指定两项。

下一次刷新时间计算后必须是未来某个时间。

never refresh

该语句用于指定不刷新。

指定后保护物化视图不被任何自动刷新、高级包机制刷新，并忽略任何发出的刷新语句、高级包执行。

示例（单机HEAP表）

```
ALTER MATERIALIZED VIEW mv_refresh
REFRESH
COMPLETE
ON DEMAND
NEXT SYSDATE + 1/(24*60);
```

alter_query_rewrite_clause

该语句用于更改该物化视图的查询重写选项。

可通过查询 `QUERY_REWRITE_ENABLED` 参数查看当前物化视图查询重写功能的具体情况。

查询重写功能包含如下限制：

- 物化视图不允许包含可变的内置函数（如时间相关函数）、UDF和UDP等。
- 物化视图不允许包含表函数。
- 物化视图不允许包含伪列。
- 当物化视图基于的表定义修改，导致物化视图失效，此时该物化视图不可进行查询重写。
  - 选中的基表列字段数据类型更改
  - 选中的基表列字段删除
  - 选中的基表列字段名称更改

(enable|diasble) query rewrite

具体含义如下：

- ENABLE QUERY REWRITE: 允许物化视图用于查询重写
- DISABLE QUERY REWRITE: 不允许物化视图用于查询重写

示例（单机HEAP表）

```
ALTER MATERIALIZED VIEW mv_refresh ENABLE QUERY REWRITE;
```

ALTER OUTLINE

通用描述

ALTER OUTLINE用于修改一个存储纲要，包括重建，更改名称，更改归属类别，设置为有效，设置为无效等操作。

用户必须拥有ALTER ANY OUTLINE权限才能修改一个存储纲要。

语句定义

alter_outline::=

```
syntax::= ALTER [PUBLIC] OUTLINE outline_name (REBUILD|RENAME TO new_outline_name|CHANGE CATEGORY TO category_name|(ENABLE |
DISABLE))
```

public

公有模式，默认值。

outline_name

将要修改的OUTLINE的名称。

rebuild

OUTLINE进行重建，即重新生成对应的hint信息。

rename to new_outline_name

对OUTLINE重命名。重命名的OUTLINE名称不能带模式名指定，且需符合YashanDB的对象命名规范。

change category to category_name

更改OUTLINE对应的类别。类别名称可以是已经存在的或者新指定的，不能带模式名指定，且需符合YashanDB的对象命名规范。

enable

允许使用该OUTLINE。

disable

不允许使用该OUTLINE。

示例

```
-- 重建OUTLINE
ALTER OUTLINE ol_a REBUILD;

-- 更改OUTLINE的归属类别，系统自动创建不存在的类别
ALTER OUTLINE ol_a CHANGE CATEGORY TO ctgy_new;

-- 禁止使用OUTLINE
ALTER OUTLINE ol_a DISABLE;

-- 取消OUTLINE的禁用状态
ALTER OUTLINE ol_a ENABLE;
```

ALTER PACKAGE

通用描述

ALTER PACKAGE语句用于显式的重编译一个自定义高级包的HEAD、BODY或整个自定义高级包。

对于在SYS schema内的自定义高级包，需要由SYS用户执行ALTER PACKAGE语句。

对于其他schema内的自定义高级包，需要由其所属用户或拥有ALTER ANY PROCEDURE权限的用户执行ALTER PACKAGE语句。

自定义高级包含义及限制请参考自定义高级包章节。

语句定义

alter_package::=

```
syntax::= ALTER PACKAGE [schema "."] package_name (package_compile_clause|(EDITIONABLE|NONEDITIONABLE))
```

schema

包含自定义高级包的schema名称，省略则默认为当前登录用户的schema。

package_name

需要显式重编译的自定义高级包名称。

package_compile_clause

指定重编译选项，详见compile_clause描述。

editionable | noneditionable

用于语法兼容，无实际含义。

示例（单机、共享集群部署）

```
ALTER PACKAGE calc_fee COMPILE;

ALTER PACKAGE calc_fee COMPILE PACKAGE;

ALTER PACKAGE calc_fee COMPILE SPECIFICATION;

ALTER PACKAGE calc_fee COMPILE BODY;
```

ALTER PROCEDURE

通用描述

ALTER PROCEDURE语句用于显式的重编译一个存储过程。

对于在SYS schema内的存储过程，需要由SYS用户执行ALTER PROCEDURE语句。

对于其他schema内的存储过程，需要由其所属用户或拥有ALTER ANY PROCEDURE权限的用户执行ALTER PROCEDURE语句。

存储过程含义及限制请参考存储过程章节。

语句定义

alter procedure::=

```
syntax::= ALTER PROCEDURE [schema "."] procedure_name (procedure_compile_clause|(EDITIONABLE|NONEDITIONABLE))
```

schema

包含存储过程的模式名称，省略则默认为当前登录用户的模式。

procedure_name

需要显式重编译的存储过程名称。

procedure_compile_clause

指定重编译选项，详见compile_clause描述。

editionable | noneditionable

用于语法兼容，无实际含义。

示例（单机、共享集群部署）

```
ALTER PROCEDURE sales.ya_proc COMPILE;
```

ALTER PROFILE

通用描述

ALTER PROFILE用于修改一个profile的内容。

执行本语句需注意如下事项：

- 用户必须拥有ALTER PROFILE权限才能修改一个profile。
- 对于已与用户关联的profile，修改profile后，其对用户的资源限制即时生效。

对于YashanDB默认的profile（名称为DEFAULT），也可使用本语句对其内容进行修改。

分布式部署中用户无法执行本语句。

语句定义

alter profile::=

```
syntax::= ALTER PROFILE profile_name LIMIT  (resource_parameters | password_parameters) {" " (resource_parameters |
password_parameters)}
```

password_parameters::=

```
syntax::= (password_parameter_name password_parameter_value)
  {" " (password_parameter_name password_parameter_value)}
```

resource_parameters::=

```
syntax::= (resource_parameter_name resource_parameter_value)
  {" " (resource_parameter_name resource_parameter_value)}
```

profile_name

已存在的一个profile的名称。

password_parameters

修改密码策略相关的资源限制，同时修改多个资源限制以空格分隔。

password_parameter_name

为安全手册密码策略中所定义的密码策略参数，指定其他参数将报错。

password_parameter_value

指定密码策略参数所对应的资源限制值，语法及规则同CREATE PROFILE中的password_parameter_value。

resource_parameters

参考SQL语句CREATE PROFILE中的resource_parameters。

resource_parameter_name

参考SQL语句CREATE PROFILE中的resource_parameter_name。

resource_parameter_value

参考SQL语句CREATE PROFILE中的resource_parameter_value。

不同参数可设置的值

参考SQL语句CREATE PROFILE中的不同参数可设置的值。

示例（单机、共享集群部署）

```
ALTER PROFILE prof_1 LIMIT FAILED_LOGIN_ATTEMPTS 1 PASSWORD_LIFE_TIME 180;

ALTER PROFILE prof_2 LIMIT FAILED_LOGIN_ATTEMPTS DEFAULT PASSWORD_LIFE_TIME 180;

ALTER PROFILE prof_3 LIMIT FAILED_LOGIN_ATTEMPTS DEFAULT PASSWORD_LIFE_TIME UNLIMITED;

ALTER PROFILE prof_4 LIMIT
FAILED_LOGIN_ATTEMPTS 1
PASSWORD_LIFE_TIME 180
PASSWORD_REUSE_TIME UNLIMITED
PASSWORD_REUSE_MAX UNLIMITED
PASSWORD_LOCK_TIME 1
PASSWORD_GRACE_TIME 0;
```

ALTER SEQUENCE

通用描述

ALTER SEQUENCE用于修改序列号生成器的各项参数。

分布式部署中用户无法执行本语句。

语句定义

alter sequence::=

```
syntax::= ALTER SEQUENCE [schema "."] sequence
((INCREMENT BY integer) | (MAXVALUE integer|NOMAXVALUE) | (MINVALUE integer|NOMINVALUE) | (CYCLE|NOCYCLE) | (ORDER|NOORDER) |
(CACHE integer|NOCACHE))
{" " ((INCREMENT BY integer) | (MAXVALUE integer|NOMAXVALUE) | (MINVALUE integer|NOMINVALUE) | (CYCLE|NOCYCLE) |
(ORDER|NOORDER) | (CACHE integer|NOCACHE))}
```

increment by

该语句用于修改序列号生成器的增量值，规则同CREATE SEQUENCE，修改后该序列器的开始值将不可回溯。

maxvalue|nomaxvalue

该语句用于修改序列号生成器的最大值，规则同CREATE SEQUENCE。

minvalue|nominvalue

该语句用于修改序列号生成器的最小值，规则同CREATE SEQUENCE。

cycle|nocycle

该语句用于修改序列号生成器的循环开关设置，规则同CREATE SEQUENCE。

order|noorder

该语句用于修改序列号生成器的ORDER属性，规则同CREATE SEQUENCE。

cache|nocache

该语句用于修改序列号预分配个数，规则同CREATE SEQUENCE。

ALTER SESSION

通用描述

ALTER SESSION用于动态地改变所在会话的属性，或一些会话级的系统配置。这些改变只在当前连接的会话中有效，会话断开后将恢复为初始值。

语句定义

alter session::=

```
syntax::= ALTER SESSION set_clause
```

set_clause::=

```
syntax::= SET parameter_name "=" parameter_value
```

set_clause

该语句用于指定会话属性或系统配置参数的值。

其中，parameter_name可以为配置参数里的会话级参数，也可以为会话的属性参数。

示例（单机、共享集群部署）

```
--会话级参数date_format
SHOW PARAMETER date_format;
NAME                 VALUE
-------------------- -------
DATE_FORMAT          yyyy-mm-dd

ALTER SESSION SET date_format='yyyy-mm-dd hh24:mi:ss';

SHOW PARAMETER date_format;
NAME                 VALUE
-------------------- -------
DATE_FORMAT          yyyy-mm-dd hh24:mi:ss

--会话级参数ISOLATION_LEVEL
-- 设置会话的默认隔离级别为SERIALIZABLE。
ALTER SESSION SET ISOLATION_LEVEL = SERIALIZABLE;

-- 设置会话的默认隔离级别为READ COMMITTED。
ALTER SESSION SET ISOLATION_LEVEL = READ COMMITTED;
```

会话属性参数

current_schema

该参数为所在会话的当前schema属性。

schema是数据库对象的集合（即所有数据库对象被真实表述为schema.对象名），当用户访问某个数据库对象时，如未显式指定schema，则默认指定为当前schema。

当连接一个会话时，初始的current_schema为当前所登录用户对应的schema，通过ALTER SESSION语句可以动态地改变这个值。改变之后，所有在此会话中执行的SQL语句的默认schema均为修改后的current_schema。

> **Note**：
> - 权限校验基于用户，切换当前schema不影响当前所登录用户权限。
> - 当前所登录用户需拥有ALTER SESSION权限。

示例

```
--系统中存在sales1、sales2两个用户
--以sales1用户登录开启会话
conn sales1/1%2;

--创建area表，默认schema为sales1
DROP TABLE IF EXISTS area;
CREATE TABLE area
(area_no CHAR(2) NOT NULL,
 area_name VARCHAR2(60),
 DHQ VARCHAR2(20) DEFAULT 'ShenZhen' NOT NULL);
INSERT INTO area VALUES ('01','华东','Shanghai');
INSERT INTO area VALUES ('02','华西','Chengdu');
COMMIT;

SELECT * FROM area;
AREA_NO AREA_NAME                 DHQ
------- ------------------------- -------------------
01      华东                      Shanghai
02      华西                      Chengdu

--修改current_schema
ALTER SESSION SET current_schema=sales2;

--默认schema被修改为sales2
SELECT * FROM area;
[1:15]YAS-02012 table or view does not exist
```

ALTER SYSTEM

通用描述

ALTER SYSTEM用于动态地改变所在数据库实例（Instance）的属性，立即生效（但对于ALTER SYSTEM SET PARAMETER可设定为重启生效）且对所有用户生效。

语句定义

alter system::=

```
syntax::= ALTER SYSTEM (set_parameter_clause
|SWITCH LOGFILE
|CHECKPOINT
|FLUSH BUFFER_CACHE
|FLUSH SHARED_POOL
|EXTEND BUFFER_CACHE SIZE size_clause
|ARCHIVE LOG CURRENT
|kill_session_clause
|cancel_sql_clause
|IGNORE STANDBY MISMATCHED REDO
|dump_clause
|FLUSH GTS
|CLEAN RESIDUAL TABLESPACE)
```

set_parameter_clause**::=**

```
syntax::= SET parameter_name "=" parameter_value
[SCOPE "=" (spfile|memory|both)]
[(TYPE "=" (CN|DN|MN|ALL)) | (NODE "=" node_id)]
```

kill_session_clause**::=**

```
syntax::= KILL SESSION "'" session_id "," session_serial "'"
```

cancel_sql_clause**::=**

```
syntax::= CANCEL SQL "'" session_id "," session_serial ["," sql_id] "'"
```

dump_clause**::=**

```
syntax::= DUMP (PRIVATE LOG
|LOGFILE file_name
|DATAFILE file_id [BLOCK block_id|MINBLOCK block_id MAXBLOCK block_id]
|SESSION sid BACKTRACE)
```

set_parameter_clause

该语句用于修改数据库的配置参数。 ALTER SYSTEM SET修改配置参数，会打印修改信息的描述到RUN LOG中。 分布式部署中，仍更建议使用YCM图形化界面或者yasboot命令行方式来修改配置参数。

如果parameter_name为YashanDB规定的只读参数，执行此语句时将会提示YAS-02138错误。

YashanDB规定了如下几类配置参数：

- 只读参数，不可被修改。
- 允许被修改，且修改后可以立即生效的配置参数。
- 允许被修改，但修改后需要重启实例才会生效的配置参数。

> **Note**：
> 在分布式集群中，执行本语句默认修改当前节点。

scope

SCOPE用于设定对配置参数修改后的生效方式，默认为BOTH。

- spfile：将参数值写入磁盘，需重启才能生效
- memory：将参数值写入内存，立即生效，但重启后失效
- both：将参数值同时写入内存和磁盘，立即生效，重启后也生效

如果VALUE被赋予为YashanDB规定的不可立即生效参数，则必须同时指定SCOPE为SPFILE值，否则将会提示YAS-06001错误。

示例

```
SHOW PARAMETER data_buffer_size;
NAME                 VALUE
-------------------- -------
DATA_BUFFER_SIZE     64M

ALTER SYSTEM SET data_buffer_size=128M scope=spfile;
SHOW PARAMETER data_buffer_size;
NAME                 VALUE
-------------------- -------
DATA_BUFFER_SIZE     64M

SHOW PARAMETER checkpoint_timeout;
NAME                 VALUE
-------------------- -------
CHECKPOINT_TIMEOUT   300


ALTER SYSTEM SET checkpoint_timeout=400;
SHOW PARAMETER checkpoint_timeout;
NAME                 VALUE
-------------------- -------
CHECKPOINT_TIMEOUT   400
```

type

本语句只在分布式部署中使用，用于指定节点类型进行配置。

- CN：修改所有CN节点配置参数
- DN：修改所有DN节点配置参数
- MN：修改所有MN节点配置参数
- ALL：修改所有节点配置参数，包括所有CN、DN和MN

如果不指定TYPE，默认修改本节点配置参数。

> **Note**：
> 1.不允许在NOMOUNT/MOUNT模式下指定TYPE，默认修改本节点配置参数。
> 2.对于未处于OPEN状态的节点，即使处于TYPE指定范围内，也无法跨节点修改其配置参数。
> 3.上述描述的是连接到某一CN节点执行时的规则，如直连到其他节点，则不允许指定TYPE，默认修改本节点配置参数。

示例（分布式部署）

```
--修改所有CN节点上的RUN_LOG_FILE_COUNT
ALTER SYSTEM SET RUN_LOG_FILE_COUNT=200 SCOPE=BOTH TYPE = CN;

--修改所有DN节点上的RUN_LOG_FILE_COUNT
ALTER SYSTEM SET RUN_LOG_FILE_COUNT=200 SCOPE=BOTH TYPE = DN;

--修改所有MN节点上的RUN_LOG_FILE_COUNT
ALTER SYSTEM SET RUN_LOG_FILE_COUNT=200 SCOPE=BOTH TYPE = MN;
```

```
--修改所有CN、DN、MN节点上的RUN_LOG_FILE_COUNT
ALTER SYSTEM SET RUN_LOG_FILE_COUNT=200 SCOPE=BOTH TYPE = ALL;
```

node

本语句只在分布式部署中使用，用于指定节点进行配置，node_id = g-n。

- g：节点所在组号ID
- n：节点ID，'*'表示组内所有节点

如果不指定NODE，默认修改本节点配置参数。

> **Note**：
> 1.不允许在NOMOUNT/MOUNT模式下指定NODE，默认修改本节点配置参数。
> 2.对于未处于OPEN状态的节点，即使处于NODE指定范围内，也无法跨节点修改其配置参数。
> 3.上述描述的是连接到某一CN节点执行时的规则，如直连到其他节点，则不允许指定NODE，默认修改本节点配置参数。

示例（分布式部署）

```
--修改MN节点1-1上的RUN_LOG_FILE_COUNT
ALTER SYSTEM SET RUN_LOG_FILE_COUNT=200 SCOPE=BOTH NODE = 1-1;

--修改所有CN节点上的RUN_LOG_FILE_COUNT
ALTER SYSTEM SET RUN_LOG_FILE_COUNT=200 SCOPE=BOTH NODE = 2-*;
```

switch logfile

该语句用于对数据库触发一次当前redo日志文件的强制切换，该操作不是进行归档的触发条件。

示例

```
ALTER SYSTEM SWITCH LOGFILE;
```

checkpoint

该语句用于对数据库触发一次全量的CHECKPOINT，以使内存中的脏数据写入到磁盘中。

示例

```
ALTER SYSTEM CHECKPOINT;
```

flush buffer_cache

该语句用于对数据库触发一次失效Buffer Cache中所有的Blocks，此操作会导致所有缓存数据被清除，其后的SQL查询将执行物理读取磁盘数据。分布式部署中不可使用本语句。

示例（单机、共享集群部署）

```
ALTER SYSTEM FLUSH BUFFER_CACHE;
```

flush shared_pool

该语句用于对数据库触发一次回收失效的pool内存资源，此操作会导致长期未使用的SQL语句缓存被清空，其后的相同的SQL语句使用将重新进行编译解析。分布式部署中不可使用本语句。

示例（单机、共享集群部署）

```
ALTER SYSTEM FLUSH SHARED_POOL;
```

extend buffer_cache size

该语句用于临时扩展数据库的Data Buffer大小，重启后失效。分布式部署中不可使用本语句。

size_clause

对Data Buffer指定一个整型值，单位可以为B/K/M/G/T/P/E。每次运行该语句所设定SIZE的限额受如下两个值制约：

- 所在服务器的物理内存
- 2^24*DB_BLOCK_SIZE*DATA_BUFFER_PARTS

超过限定值将会提示YAS-00101错误。

示例（单机、共享集群部署）

```
ALTER SYSTEM EXTEND BUFFER_CACHE SIZE 4G;
```

archive log current

该语句用于对数据库触发一次当前redo日志文件的切换和归档。此操作需要数据库的归档模式需处于开启状态，否则将提示YAS-02079错误。

示例

```
ALTER SYSTEM ARCHIVE LOG CURRENT;
```

kill_session_clause

该语句用于终止一个指定的会话，并回滚（Rollback）此会话中未提交事务，释放此会话产生的锁（Lock），之后在此会话中运行SQL语句将提示连接错误信息。

执行此命令需提供会话的SID和SERIAL#。

示例（单机/共享集群部署）

```
--在sales用户下执行如下语句
SELECT * FROM area WHERE area_no='01' FOR UPDATE;

--查看锁及会话信息
SELECT * FROM V$LOCK;
    SID             ID1        ID2 LMODE     REQUEST
-------- --------------- ---------- --------- ---------
      21            1328            TS
      21     21474844928          0 ROW

SELECT sid,serial# FROM V$SESSION WHERE sid=21;
     SID      SERIAL#
-------- ------------
      21            2


--下面操作需要拥有管理员权限
--通过SID和SERIAL#参数终止指定会话
ALTER SYSTEM KILL SESSION '21,2';
```

分布式跨CN会话杀除

在配置了多CN的分布式环境中，可使用本语句进行跨CN的会话终止，GLOBAL_SESSION_ID为分布式部署中的全局会话ID（可通过USERENV函数或查询DV$SESSION视图获取）。

示例（在一个配置了2-1和2-2两个CN节点的分布式环境中）

```
--连接2-1，建立一个会话，查看此会话的全局会话ID
--使用USERENV函数查询时使用GSID参数
SELECT USERENV('GSID') FROM dual;
USERENV('GSID')
```

```
--------------
        131091

--获取SERIAL#
SELECT global_session_id, serial FROM DV$SESSION WHERE global_session_id IN (131091);
GLOBAL_SESSION_ID      SERIAL#
----------------- ------------
           131091            5

--连接2-2，不要关闭2-1的会话，在2-2上杀除2-1的会话
ALTER SYSTEM KILL SESSION '131091,5';
```

cancel_sql_clause

该语句用于终止一个正在运行中的SQL操作，执行此命令需要提供该SQL操作所在会话的SID和SERIAL#。

指定SQL_ID时表示终止该SQL_ID对应的SQL操作，不指定时则表示终止该会话下当前正在运行的SQL操作。

在分布式下，支持跨CN终止SQL操作，需提供会话的GSID和SERIAL#。

示例（单机、共享集群部署）：会话1——在sales用户下执行如下语句：

```
SELECT * FROM area WHERE area_no='01' FOR UPDATE;
```

示例（单机、共享集群部署）：会话2——在sales用户下执行如下语句：

```
--开启新会话2，以sales用户登录，添加等锁SQL，使其处于执行状态中
UPDATE area SET DHQ='Shanghai3' WHERE area_no='01';
```

示例（单机、共享集群部署）：会话3——在DBA权限用户下执行如下语句：

```
--开启新会话3，以DBA权限用户登录
SELECT * FROM V$LOCK WHERE request='ROW';
     SID            ID1       ID2 LMODE     REQUEST
-------- -------------- --------- --------- ---------
      21     4447535124                     ROW

--查询SQL_ID
SELECT sid,serial#,sql_id FROM V$SESSION WHERE sid=21;
    SID      SERIAL#               SQL_ID
-------- ------------ ---------------------
      21            1            3829447373

--终止指定的SQL
ALTER SYSTEM CANCEL SQL '21,1,3829447373';

--终止正在运行的SQL
ALTER SYSTEM CANCEL SQL '21,1';
```

在分布式环境中，上述会话1和会话2在cn2-1上执行，会话3操作在cn2-2上执行，即可实现跨CN终止SQL。

ignore standby mismatched redo

该语句用于忽略备库与主库的不匹配的redo日志，使备库从redo日志分歧点继续接收主库日志。此操作前提是备库状态处于REDO MISMATCH，处于该状态的备库无法接收主库日志。分布式部署中不可使用本语句。

示例（单机、共享集群部署）

```
ALTER SYSTEM IGNORE STANDBY MISMATCHED REDO;
```

dump_clause

该语句用于将YashanDB内部的信息转储（dump）到跟踪（trace）文件中，供用户进行分析和判断故障。

dump

用户在dump操作中指定某类内部信息，当前会话中的该类信息就被转储到trace文件中。

每一次dump操作在trace文件中产生一段信息，一个会话中可以多次执行dump操作。

trace

trace文件在会话首次执行dump时创建，文件存放在{YASDB_DATA}/diag/trace中，文件名称为{dbname}*yas*{sid}.trc。

trace文件被创建后，后续该会话中所有被dump的信息都被写入该文件。

当会话ID被复用时，sid对应的trace文件也将被复用，不会创建新的trace文件。

private log

将当前会话的私有日志信息dump到trace文件中。执行此类信息的dump要求数据库处于mount或open状态。

logfile

将某个日志文件的页面信息dump到trace文件中。执行此类信息的dump要求数据库处于mount或open状态。

本操作会对日志文件加锁，多个会话同时dump同一个日志文件时会形成锁等待。

当dump的logfile中的某个group是关于加密表空间上的页面记录时，这些group会受到保护，不会解析。

file_name

日志名称，为V$LOGFILE视图中存在的某个日志名称。

datafile

将某个数据文件的页面信息dump到trace文件中。执行此类信息的dump要求数据库处于mount或open状态。

本操作会对数据文件加锁，多个会话同时dump同一个数据文件时会形成锁等待。

当dump加密表空间的数据文件某些页面或整个数据文件时，这些页面会受到保护，不会解析。

file_id

指定数据文件的ID，该值为数据文件的全局ID，即在V$DATAFILE视图中的ID字段值。

block_id

指定数据文件内的页面ID：

- 不指定BLOCK：dump所有页面。
- BLOCK block_id：指定一个页面。
- MINBLOCK block_id [MAXBLOCK block_id]：指定一批页面。必须同时指定MINBLOCK和MAXBLOCK，否则会报错。
- 指定一批页面时，MAXBLOCK小于MINBLOCK或MAXBLOCK超出文件大小均会报错。

session backtrace

将指定会话的当前堆栈信息dump到trace文件中。执行此类信息的dump要求数据库处于mount或open状态。

本操作不允许并发，同一时间只能有一个会话执行此类dump。

示例

```
ALTER SYSTEM DUMP PRIVATE LOG;

ALTER SYSTEM DUMP LOGFILE 'redo1';

ALTER SYSTEM DUMP DATAFILE 6;

ALTER SYSTEM DUMP DATAFILE 6 BLOCK 0;

ALTER SYSTEM DUMP DATAFILE 6 MINBLOCK 128 MAXBLOCK 137;
```

```
ALTER SYSTEM DUMP SESSION 23 BACKTRACE;
```

flush gts

该语句用于强制同步GTS服务的SCN到所有CN节点。单机部署中不可使用本语句。分布式部署，只有CN节点才能使用本语句。

示例（分布式部署）

```
ALTER SYSTEM FLUSH GTS;
```

clean residual tablespace

该语句用于手动清理迁移chunk后残留的表空间，只能在分布式部署中使用。

示例（分布式部署）

```
ALTER SYSTEM CLEAN RESIDUAL TABLESPACE;
```

ALTER TABLE

通用描述

ALTER TABLE用于修改数据库里的表的结构和定义，以及对表进行相关管理操作，包括：

- 表更名
- 修改列
- 修改分区
- 修改约束
- 开启和关闭行迁移（row movment）
- 开启和关闭附加日志（supplemental logging）
- 开启和关闭redo日志（nologging）
- 空间收缩（shrink space）
- LSC表的后台数据转换选项开关（data transformer）
- LSC表的可变数据生命周期（MCOL time to live）
- LSC表开启或关闭MCOL排序 （MCOL order by）
- LSC表的强制转换（force xfmr）

其中，依据LSC表的存储特性，只能对其分区相关属性（但不包括分区索引）进行修改操作（包括增加分区和删除分区等）。

不能对AC对象以及AC对象的源表执行ALTER TABLE操作。

语句定义

alter table::=

```
syntax::= ALTER TABLE [schema "."] table_name
(alter_table_properties
|alter_column_clause
|alter_table_partition
|alter_table_constraint)
```

alter_table_properties**::=**

```
syntax::= (RENAME TO new_table_name)
|row_movement_clause
|supplemental_table_logging
|shrink_space_clause
|lsc_properties_clause
|logging_clause
|readonly_clause
```

row_movement_clause**::=**

```
syntax::= ((ENABLE)|(DISABLE)) ROW MOVEMENT
```

supplemental_table_logging**::=**

```
syntax::= add_supplemental_logging_clause
|drop_supplemental_logging_clause
```

add_supplemental_logging_clause**::=**

```
syntax::= ADD SUPPLEMENTAL LOG DATA "("(PRIMARY KEY|UNIQUE|ALL)")" COLUMNS
```

drop_supplemental_logging_clause**::=**

```
syntax::= DROP SUPPLEMENTAL LOG DATA
```

shrink_space_clause::=

```
syntax::= SHRINK SPACE [COMPACT] [CASCADE]
```

lsc_properties_clause::=

```
syntax::= enable_xfmr_clause|mcol_ttl_clause|mcol_order_by_clause|force_xfmr_clause
```

enable_xfmr_clause::=

```
syntax::= (ENABLE|DISABLE) ((TRANSFORM | COMPACT | BUILD AC){" "(TRANSFORM | COMPACT| BUILD AC)})
```

mcol_ttl_clause::=

```
syntax::= ALTER MCOL TTL timestamp
```

mcol_order_by_clause::=

```
syntax::= (ENABLE|DISABLE) MCOL order by
```

force_xfmr_clause::=

```
syntax::= ALTER SLICE ALL (STABLE | COMPACT | CLEAN) [ASYNC]
```

logging_clause::=

```
syntax::= LOGGING [ASYNC]|NOLOGGING
```

readonly_clause::=

```
syntax::= READONLY | READWRITE
```

alter_column_clause::=

```
syntax::= add_column_clause
|drop_column_clause
|rename_column_clause
|modify_column_clause
```

add_column_clause::=

```
syntax::= ADD [COLUMN] "(" (column_definition) {"," (column_definition)} ")"
[lob_clauses]
```

column_definition::=

```
syntax::= column dataType [(DEFAULT default_expr | inline_constraint)
{" " (DEFAULT default_expr | inline_constraint)}]
```

inline_constraint定义

lob_clauses::=

```
syntax::= (lob_clause) {" " (lob_clause)}
```

lob_clause::=

```
syntax::= LOB "(" (column) {"," (column)} ")" STORE AS [BASICFILE|SECUREFILE]
"("(TABLESPACE space_name|(ENABLE|DISABLE) STORAGE IN ROW)")"
```

drop_column_clause::=

```
syntax::= DROP drop_column
```

drop_column::=

```
syntax::= [COLUMN] "(" (column_name) {"," (column_name)} ")"
```

rename_column_clause::=

```
syntax::= RENAME [COLUMN] old_name TO new_name
```

modify_column_clause::=

```
syntax::= MODIFY (column_name (dataType|DEFAULT default_expr|inline_constraint)
{" " (dataType|DEFAULT default_expr|inline_constraint)})
{"," (column_name (dataType|DEFAULT default_expr|inline_constraint)
{" " (dataType|DEFAULT default_expr|inline_constraint)})}
```

alter_table_partition::=

```
syntax::= add_table_partition
|drop_table_partition
|drop_table_subpartition
|truncate_table_partiton
|truncate_table_subpartition
|set_partition_clause
|modify_partition_clause
|split_table_partition
```

add_table_partition::=

```
syntax::= ADD (((PARTITION [partname] add_range_partition_clause)
{"," (PARTITION [partname] add_range_partition_clause)})|
    ((PARTITION [partname] add_list_partition_clause)
    {"," (PARTITION [partname] add_list_partition_clause)})|
    (PARTITION [partname] add_hash_partition_clause))
```

add_range_partition_clause::=

```
syntax::= range_values_clause [table_partition_description]
[("(" (
 ((range_subpartition_desc) {"," (range_subpartition_desc)})
 | ((list_subpartition_desc) {"," (list_subpartition_desc)})
 | (individual_hash_subparts { "," individual_hash_subparts})) ")")]
```

range_values_clause::=

```
syntax::= VALUES LESS THAN "(" (literal|MAXVALUE) {"," (literal|MAXVALUE)} ")"
```

table_partition_description::=

```
syntax::= (TABLESPACE tablespace|PCTFREE integer|PCTUSED integer|INITRANS integer|MAXTRANS integer)
```

add_list_partition_clause::=

```
syntax::= list_values_clause [table_partition_description]
[("(" (
 ((range_subpartition_desc) {"," (range_subpartition_desc)})
 | ((list_subpartition_desc) {"," (list_subpartition_desc)})
 | (individual_hash_subparts { "," individual_hash_subparts})) ")")]
```

list_values_clause::=

```
syntax::= VALUES "(" (DEFAULT|list_values) ")"
```

list_values::=

```
syntax::= ((literal|NULL) {"," (literal|NULL)})
|(("(" ((literal|NULL) {"," (literal|NULL)})")") {"," ("(" ((literal|NULL) {"," (literal|NULL)})")")})
```

add_hash_partition_clause::=

```
syntax::= [partition_storage_clause]
[("(" (
 ((range_subpartition_desc) {"," (range_subpartition_desc)})
 | ((list_subpartition_desc) {"," (list_subpartition_desc)})
 | (individual_hash_subparts { "," individual_hash_subparts})) ")")]
```

partition_storage_clause::=

```
syntax::= TABLESPACE tablespace
```

drop_table_partition::=

```
syntax::= DROP PARTITION ((partname) {"," (partname)}) [update_index_clause]
```

update_index_clause::=

```
syntax::= [UPDATE | INVALIDATE] GLOBAL INDEXES
```

drop_table_subpartition::=

```
syntax::= DROP SUBPARTITION ((subpartname) {"," (subpartname)}) [update_index_clause]
```

truncate_table_partition::=

```
syntax::= TRUNCATE PARTITION ((partname) {"," (partname)}) [truncate_part_clause]
```

truncate_part_clause::=

```
syntax::= ((DROP|REUSE) STORAGE)|CASCADE|PURGE
```

truncate_table_subpartition::=

```
syntax::= TRUNCATE SUBPARTITION ((subpartname) {"," (subpartname)}) [truncate_part_clause]
```

set_partition_clause**::=**

```
syntax::= SET ((INTERVAL "(" [expr] ")" )
| (STORE IN "(" ((tablespace) {"," (tablespace)}) ")"))
```

modify_partition_clause**::=**

```
syntax::= MODIFY (PARTITION | SUBPARTITION) partname (shrink_space_clause | add_subpartition_clause)
```

split_table_partition**::=**

```
syntax::= SPLIT PARTITION partname
    (AT "(" literal ")" [INTO "(" range_partition_desc "," range_partition_desc ")"] |
     VALUES "(" list_values ")" [INTO "(" list_partition_desc "," list_partition_desc ")"] |
     INTO "(" ((range_partition_desc) {"," (range_partition_desc)} | (list_partition_desc) {"," (list_partition_desc)}) ","
PARTITION [partname] [table_partition_description] ")")
    [UPDATE [GLOBAL] INDEXES]
```

add_subpartition_clause**::=**

```
syntax::= ADD
 (((range_subpartition_desc) {"," (range_subpartition_desc)})
 | ((list_subpartition_desc) {"," (list_subpartition_desc)})
 | (individual_hash_subparts))
```

alter_table_constraint**::=**

```
syntax::= add_constraint
|drop_constraint
|modify_constraint
|enable_disable_constraint
```

add_constraint**::=**

```
syntax::= ADD "(" (out_of_line_constraint) {"," (out_of_line_constraint)} ")"
```

out_of_line_constraint定义

drop_constarint**::=**

```
syntax::= DROP
((PRIMARY KEY)
    | (UNIQUE "("  ((column_name) {"," (column_name)}) ")")
    | (CONSTRAINT constraint_name))
[CASCADE]
[(KEEP | DROP) INDEX]
```

modify_constraint**::=**

```
syntax::= MODIFY
((PRIMARY KEY)
    | (UNIQUE "(" ((column_name) {"," (column_name)}) ")")
    | (CONSTRAINT constraint_name))
(ENABLE|DISABLE) [VALIDATE|NOVALIDATE]
[CASCADE]
```

enable_disable_constraint::=

```
syntax::= (ENABLE|DISABLE) [VALIDATE|NOVALIDATE]
((PRIMARY KEY)
    | (UNIQUE "(" ((column_name) {"," (column_name)}) ")")
    | (CONSTRAINT constraint_name))
[using_index_clause]
[CASCADE]
[(KEEP|DROP) INDEX]
```

alter_table_properties

该语句用于修改表的一系列属性。

rename to

该语句用于修改表名，指定的新名称不能为空且必须符合YashanDB的对象命名规范。

示例

```
ALTER TABLE area RENAME TO area_new;
ALTER TABLE area_new RENAME TO area;
```

row_movement_clause

该语句用于开启或关闭表的行迁移功能，参考CREATE TABLE中相应功能描述。

示例（单机HEAP表、单机TAC表）

```
-- 创建一张DISABLE ROW MOVEMENT的分区表
DROP TABLE IF EXISTS orders_info_rowmove;
CREATE TABLE orders_info_rowmove (order_no CHAR(14) NOT NULL,
order_desc VARCHAR2(100),
area CHAR(2),
branch CHAR(4),
order_date DATE DEFAULT SYSDATE NOT NULL,
salesperson CHAR(10),
id NUMBER)
PARTITION BY RANGE (id)
(PARTITION p1 VALUES LESS THAN (800),
PARTITION p2 VALUES LESS THAN (1800),
PARTITION p3 VALUES LESS THAN (2800));

-- 插入数据
INSERT INTO orders_info_rowmove VALUES ('20010102020001','product 001','02','0201',SYSDATE-400,'0001',300);
INSERT INTO orders_info_rowmove VALUES ('20010102020001','product 001','02','0201',SYSDATE-400,'0001',1300);
INSERT INTO orders_info_rowmove VALUES ('20010102020001','product 001','02','0201',SYSDATE-400,'0001',2300);
COMMIT;

-- 查询p1分区的记录
SELECT * FROM orders_info_rowmove PARTITION(p1);
ORDER_NO          ORDER_DESC   AREA  BRANCH ORDER_DATE           SALESPERSON      ID
----------------- ------------ ----- ------ -------------------- ------------- -----
20010102020001    product 001  02    0201   2020-12-06 22:55:32  0001            300

-- 更新分区列值，未开启行迁移时本语句报错
UPDATE orders_info_rowmove SET id=500 WHERE id=1300;
YAS-02209 ROW MOVEMENT is not enabled

-- 执行开启
ALTER TABLE orders_info_rowmove ENABLE ROW MOVEMENT;
UPDATE orders_info_rowmove SET id=500 WHERE id=1300;

-- 重新查询p1分区的记录，之前在p2分区的数据被移动到了此分区中
SELECT * FROM orders_info_rowmove PARTITION(p1);
```

```
ORDER_NO          ORDER_DESC   AREA  BRANCH ORDER_DATE           SALESPERSON      ID
----------------- ------------ ----- ------ -------------------- ------------- -----
20010102020001    product 001  02    0201   2020-12-06 22:55:32  0001            300
20010102020001    product 001  02    0201   2020-12-06 22:55:32  0001            500

-- 跨分区更新数据结束后，建议关闭行迁移
ALTER TABLE orders_info_rowmove DISABLE ROW MOVEMENT;
```

示例（分布式TAC表）

```
-- 创建一张DISABLE ROW MOVEMENT的TAC表
DROP TABLE IF EXISTS orders_info_rowmove;
CREATE TABLE orders_info_rowmove (order_no CHAR(14) NOT NULL,
order_desc VARCHAR2(100),
area CHAR(2),
branch CHAR(4),
order_date DATE DEFAULT SYSDATE NOT NULL,
salesperson CHAR(10),
id NUMBER)
PARTITION BY HASH (area)
SUBPARTITION BY RANGE (id)
SUBPARTITION TEMPLATE(
SUBPARTITION sp1 VALUES LESS THAN (800),
SUBPARTITION sp2 VALUES LESS THAN (1800),
SUBPARTITION sp3 VALUES LESS THAN (2800))
(PARTITION p1,
PARTITION p2,
PARTITION p3,
PARTITION p4,
PARTITION p5,
PARTITION p6,
PARTITION p7,
PARTITION p8,
PARTITION p9,
PARTITION p10,
PARTITION p11,
PARTITION p12,
PARTITION p13,
PARTITION p14,
PARTITION p15,
PARTITION p16,
PARTITION p17,
PARTITION p18,
PARTITION p19,
PARTITION p20,
PARTITION p21);

-- 插入数据
INSERT INTO orders_info_rowmove VALUES ('20010102020001','product 001','02','0201',SYSDATE-400,'0001',300);
INSERT INTO orders_info_rowmove VALUES ('20010102020001','product 001','02','0201',SYSDATE-400,'0001',1300);
INSERT INTO orders_info_rowmove VALUES ('20010102020001','product 001','02','0201',SYSDATE-400,'0001',2300);
COMMIT;

-- 查询p1中sp1子分区的记录(由于分布式一级分区为HASH分区，实际结果可能与本例中结果不同)
SELECT * FROM orders_info_rowmove SUBPARTITION(p1_sp1);
ORDER_NO          ORDER_DESC                                                      AREA  BRANCH ORDER_DATE
SALESPERSON            ID
----------------- ---------------------------------------------------------------- ----- ------ ----------------------------
--- ------------- -----------
20010102020001    product 001                                                     02    0201   2022-08-21
0001                  300

-- 更新分区列值，未开启行迁移时本语句报错
UPDATE orders_info_rowmove SET id=500 WHERE id=1300;
YAS-02209 ROW MOVEMENT is not enabled

-- 执行开启
ALTER TABLE orders_info_rowmove ENABLE ROW MOVEMENT;
UPDATE orders_info_rowmove SET id=500 WHERE id=1300;
```

```
-- 重新查询p1中sp1子分区的记录，之前在sp2子分区的数据被移动到了此分区中
SELECT * FROM orders_info_rowmove SUBPARTITION(p1_sp1);
ORDER_NO          ORDER_DESC                                                      AREA  BRANCH ORDER_DATE
SALESPERSON            ID
----------------- ------------------------------------------------------------------------ ----- ------ ------------------------------
--- ------------- -----------
20010102020001    product 001                                                     02    0201   2022-08-21
0001                   300
20010102020001    product 001                                                     02    0201   2022-08-21
0001                   500

-- 跨分区更新数据结束后，建议关闭行迁移
ALTER TABLE orders_info_rowmove DISABLE ROW MOVEMENT;
```

supplemental_table_logging

该语句用于开启或关闭表级附加日志属性。表级附加日志只对当前的表生效，数据库级附加日志请参考ALTER DATABASE章节。

分布式和共享集群部署中用户无法执行此操作。

add_supplemental_logging_clause

该语句用于为该表开启附加日志属性，开启附加日志后，对该表执行DML或DDL类型语句，将会在redo日志里记录额外的信息，以便逻辑日志解析工具能从redo日志里解析出DML和DDL语句。尤其是UPDATE和DELETE语句，会额外记录目标行的主键，唯一键或整行（除LOB列）数据到redo日志里，因此对数据库性能有影响。

附加日志有三种类型：

- PRIMARY KEY：在UPDATE，DELETE的redo日志中，仅记录该行的主键列的值。
- UNIQUE：在UPDATE，DELETE的redo日志中，仅记录该行的非空的唯一索引列的值。
- ALL：在UPDATE，DELETE的redo日志中，记录该行的所有列（LOB型，32K以上varchar和char等除外）的值。

附加日志约束项：

- 不能对已开启附加日志的表重复执行ADD SUPPLEMENTAL LOG操作。
- 不能对附加日志类型为PRIMARY KEY或UNIQUE的表，删除附加日志依赖的索引列。
- 不能对没有主键的表，开启PRIMARY KEY类型的附加日志。
- 不能对没有非空唯一索引列的表，开启UNIQUE类型的附加日志。
- 不能对开启附加日志的表，执行长度超过32K的DDL语句。
- 不能对使用了字典编码的TAC表开启附加日志。

通过DBA_LOG_GROUPS、ALL_LOG_GROUPS、USER_LOG_GROUPS视图可以查看当前已开启的附加日志信息。 示例（单机部署）

```
ALTER TABLE area ADD SUPPLEMENTAL LOG DATA (PRIMARY KEY) COLUMNS;
ALTER TABLE branches ADD SUPPLEMENTAL LOG DATA (ALL) COLUMNS;
```

drop_supplemental_logging_clause

该语句用于关闭表上的附加日志属性。

示例（单机部署）

```
ALTER TABLE area DROP SUPPLEMENTAL LOG DATA;
```

shrink_space_clause

该语句用于对表或分区进行收缩，使得数据存储更加紧凑，降低segment的高水位线，释放空闲出来的连续的extent。该语句只适用于单机HEAP表。

需注意的是，由于收缩数据涉及到物理位置变更，执行本操作前必须开启行迁移，参考row_movement_clause。

COMPACT

指定此关键字表示只将数据存储紧凑，降低水位线，并不释放空出来的extent。

CASCADE

指定此关键字表示在收缩表的同时，对其上索引也进行收缩，语法兼容，无实际意义。

示例（单机HEAP表）

```
ALTER TABLE orders_info ENABLE ROW MOVEMENT;
ALTER TABLE orders_info SHRINK SPACE;
ALTER TABLE orders_info SHRINK SPACE COMPACT;
ALTER TABLE orders_info SHRINK SPACE COMPACT CASCADE;
```

lsc_properties_clause

该语句只用于修改LSC表的相关属性，对其他类型表执行这些修改将会报错。

enable_xfmr_clause

该语句用于控制LSC表的后台数据转换能力。

目前的YashanDB后台提供给LSC表如下数据转换能力：

- TRANSFORM：LSC的可变数据依据此开关转换为稳态数据存储，可变数据缓冲区的数据被转换，转移到稳态数据内。关闭此开关即不再允许自动转换。需要注意，如果一直不转换，数据将一直存放在可变数据区，无法对数据存储格式进行优化，不建议长时间关闭。
- COMPACT：LSC的稳态数据依据此开关将数据格式进行自动优化（包括排序和合并）。关闭此开关即不再允许自动优化。需要注意，稳态数据的优化有利于查询性能提升，但是优化期间会产生额外的资源消耗，可以根据业务要求适当关闭。
- BUILD AC：LSC的稳态数据依据此开关进行AC的数据文件生成。关闭此开关即不再允许自动生成AC的数据文件，AC的能力可根据业务自行关闭开启。

此子句允许同时开启或关闭多种能力，用户也可通过配置DATA_TRANSFORMER_ENABLED参数整体开启或关闭后台数据转换能力。

> **Note**：
> 1.TRANSFORM和COMPACT涉及物理空间变动， 转换任务执行后将在一段时间内保留转换前的数据，用于满足长查询的需要；当达到保留的最大时间时对这些数据进行清理 。用户在确保业务不受影响的情况下，可参照force_xfmr_clause执行立即清理。
> 2.系统中不存在AC对象时，打开BUILD AC开关不会创建AC数据文件。

示例（LSC表）

```
ALTER SYSTEM SET DATA_TRANSFORMER_ENABLED = TRUE SCOPE=SPFILE;
ALTER TABLE orders_info ENABLE TRANSFORM;
ALTER TABLE orders_info ENABLE TRANSFORM BUILD AC COMPACT;
ALTER TABLE orders_info DISABLE TRANSFORM;
ALTER TABLE orders_info DISABLE COMPACT;
```

mcol_ttl_clause

该语句用于修改LSC表的可变生命周期，其含义请参考CREATE TABLE中对应语句描述。

该语句不适用于分布式部署。

示例（单机LSC表）

```
--将finance_info的可变生命周期由1个月修改为10天
ALTER TABLE finance_info ALTER MCOL TTL '10' DAY;

--area表创建时未设置可变生命周期，将其设为10年
ALTER TABLE area ALTER MCOL TTL '10' YEAR(9);
```

mcol_order_by_clause

该语句用于开启或关闭可变数据（MCOL）排序，其含义请参考CREATE TABLE中table_sort_clause语句描述。

示例（LSC表）

```
ALTER TABLE orders_info ENABLE MCOL ORDER BY;
ALTER TABLE orders_info DISABLE MCOL ORDER BY;
```

force_xfmr_clause

该语句用于LSC表强制转换，其中ASYNC标记表示异步转换，省略情况下默认进行同步转换。

强制转换有如下三种模式：

- STABLE：将LSC表的可变数据强制转换成稳态数据并生成此表下所有AC数据。
- COMPACT：将LSC表的稳态数据强制进行合并。
- CLEAN：将LSC的所有可删除数据（转换任务执行完成后达到延期清理条件的数据）强制进行删除。

示例（LSC表）

```
ALTER TABLE sales_info ALTER SLICE ALL STABLE;
ALTER TABLE sales_info ALTER SLICE ALL COMPACT;
ALTER TABLE sales_info ALTER SLICE ALL CLEAN;
```

logging_clause

该语句用于修改表的logging属性，logging属性用于指定表记录日志的方式。

该语句不适用于分布式部署。

logging

该语句用于将表转为logging属性，即对该表的所有操作记录日志。

- 若对已是logging属性的表执行该语句，直接返回成功。
- 若对nologging属性的表执行该语句，系统将执行一次全量checkpoint，将数据写盘并修改flushback，最后修改表的logging属性。

logging async

该语句用于异步将表转为logging属性。启动新线程完成表模式转化的操作从而不阻塞主线程工作。

注意当客户端返回成功时并不保证转换一定成功，只代表启动线程成功。后续转换仍有可能失败。转换结果通过运行日志记载。

在异步转换表的过程中存在如下约束限制：

- 不能对转换中的表做任意DML操作。
- 不能对转换中的表做除了DROP以外的任意DDL操作。

nologging

该语句用于将表转为nologging属性。若对已是nologging属性的表执行该语句，直接返回成功。若对logging属性的表执行该语句，会将其转为nologging属性。建议只在数据迁移场景打开此属性。

该语句不适用于共享集群部署。

执行该语句存在如下约束限制：

- 不能将临时表设置为nologging属性。
- 主备环境中，不能将表设置为nologging属性。

nologging属性的表存在如下特征：

- nologging表对DML操作仅支持插入数据和导入数据，且插入数据时只会记录必要的redo和undo（例如segment相关的redo），其他redo会被忽略。
- nologging表不能执行并发操作。
- nologging表不能执行回滚操作。
- nologging表不能在线创建索引（CREATE INDEX ONLINE）和在线重建索引（ALTER INDEX REBUILD ONLINE）。
- 数据库重启时会将nologging表标记为corrupted，阻止对其进行除drop和truncate外的任何操作。
- 如果一个事务失败，该事务内所有执行过插入数据操作的nologging表都会被标记为corrupted。
- nologging属性的LSC表使用bulkload模式导入数据时，不受nologging属性影响（性能无变化，失败也不会被标记为corrupted）。
- nologging属性的TAC表不允许修改列和删除主键约束。

示例（单机部署）

```
-- 执行如下语句开启logging
ALTER TABLE area LOGGING;

-- 执行如下语法关闭nologging，但主备环境中无法关闭
ALTER TABLE area NOLOGGING;
YAS-02328 table nologging is not allowed when standby exists
```

readonly_clause

无实际含义。

alter_column_clause

该语句用于指定对表的列字段的操作。

add_column_clause

该语句用于为表增加列字段，同时增加多项时在`()`中以`,`分隔。COLUMN关键字可省略。

column_definition

为表增加一个新的列字段，并对增加的列字段进行数据类型（DataType）、缺省值（DEFAULT）、行内约束（inline_constraint）等定义。

对非空表增加定义了NOT NULL约束的列字段时，必须同时为其指定缺省值，否则将提示错误。

datatype

指定列字段的数据类型。查看YashanDB的数据类型描述。

其中，数据类型指定为LOB/JSON类型时，可通过lob_clauses子句指定其存储属性。

YashanDB不允许将新增的列字段指定为Nested Table UDT类型。

default

为增加的列字段指定一个符合该列字段数据类型的缺省值，default_expr可以为字面量、运算式、函数等表达式。

当指定了缺省值时，系统将对表中现有行的该列字段填充该缺省值。

对于LSC表的新增列字段，该列为LOB型时不能指定缺省值；为其他类型时，缺省值不能定义为Sequence伪列。

示例（单机LSC表）

```
--创建LSC表lsc_forb_def
CREATE TABLE lsc_forb_def(c1 INT,c2 INT);

--LSC表中新增指定了默认值的lob列会报错
ALTER TABLE lsc_forb_def ADD(c3 CLOB DEFAULT 'default');
YAS-00004 feature "add lob column with default expr on LSC table" has not been implemented yet

--创建序列seq1
CREATE SEQUENCE seq1;

--LSC表中新增列的默认值为Sequence伪列时会报错
ALTER TABLE lsc_forb_def ADD(c3 INT DEFAULT seq1.NEXTVAL);
YAS-00004 feature "add column with default sequence on LSC table" has not been implemented yet

ALTER TABLE lsc_forb_def ADD(c3 INT DEFAULT seq1.currval);
YAS-00004 feature "add column with default sequence on LSC table" has not been implemented yet
```

inline_constraint

该语句用于在增加列字段时同时定义表的行内约束项。关于约束项的详细描述请参考通用SQL语法constraint。

示例

```
-- 为area表新增address非空字段
ALTER TABLE area ADD (address VARCHAR(30) NOT NULL DEFAULT 'no address');
```

```
SELECT area_no,address FROM area;
AREA_NO ADDRESS
------- --------------------------------
01      no address
02      no address
03      no address
04      no address
05      no address
```

lob_clause

该语句为增加的LOB/JSON字段指定存储属性，语法同CREATE TABLE中lob_clause子句描述。

drop_column_clause

该语句用于删除表中已有的列字段或约束项。

drop_column

删除指定的列字段，指定多个列字段时用`,`分隔。COLUMN关键字可省略。

该语句遵循如下规则：

- 不允许删除表中的所有列字段。
- 如果删除的列字段已被定义为主键组合中的一项，则不允许删除此列，但可以删除整个主键对应的列组合。
- 如果删除的列字段已被定义为索引对象中的一项，则删除此列字段后，其对应的索引对象也被删除。
- 分布式部署中，不允许删除分区键所在的列字段。
- LSC表中，不允许删除被定义为排序键的列字段。
- 如果数据库后台正在执行回滚，则不允许删除TAC表/含有热数据的LSC表中使用变长存储的列。

示例（单机、共享集群部署）

```
-- 创建product_pri表
CREATE TABLE product_pri AS SELECT * FROM product;

-- 定义组合主键product_no+product_name
ALTER TABLE product_pri ADD PRIMARY KEY(product_no,product_name);

-- 创建索引
CREATE UNIQUE INDEX idx_product_pri_1 ON product_pri(cost,price);

-- 删除部分主键字段，返回错误
ALTER TABLE product_pri DROP COLUMN product_no;
YAS-02132 cannot drop column referenced in a multi-column constraint

-- 删除全部主键字段，成功
ALTER TABLE product_pri DROP COLUMN(product_no,product_name);

-- 删除部分索引字段，成功
ALTER TABLE product_pri DROP COLUMN cost;

-- 删除表中最后的一个字段，返回错误
ALTER TABLE product_pri DROP COLUMN price;
YAS-02054 cannot drop all columns in a table
```

示例（分布式TAC表）

```
-- 创建area_part表
CREATE TABLE area_part AS SELECT * FROM area;

-- 删除分区键字段，返回错误
ALTER TABLE area_part DROP COLUMN area_no;
YAS-02142 cannot drop partitioning column
```

```
-- 删除非分区键字段，成功
ALTER TABLE area_part DROP COLUMN area_name;
```

示例（LSC表）

```
--创建area_orderkey表，并指定排序键为area_no
CREATE TABLE area_orderkey AS SELECT * FROM area ORDER BY area_no;

--删除排序键所在字段，返回错误
ALTER TABLE area_orderkey DROP COLUMN area_no;
YAS-03726 cannot drop order key column
```

rename_column_clause

该语句用于对列字段进行重命名，指定的新名称不允许为空且必须符合YashanDB的对象命名规范。

示例（HEAP表、TAC表）

```
ALTER TABLE branches RENAME branch_no TO branchno;
ALTER TABLE branches RENAME branchno TO branch_no;
```

modify_column_clause

该语句用于对列字段的数据类型、默认值、约束等属性进行修改，同时修改多个列字段时需在`()`中以`,`分隔。

YashanDB支持同时对列字段修改多个属性，但需遵循与CREATE TABLE相同的顺序要求，否则系统提示错误；当同时修改多个约束项时，对这些约束项无顺序要求。

datatype

修改列字段的数据类型为YashanDB认可的数据类型。

LSC表不允许修改列字段的数据类型，HEAP表和TAC表列字段数据类型修改规则如下：

- **外键约束所在列**：不允许修改子表和父表中相应列的数据类型。
- **索引所在列**：
  - 若为空表（即无数据），允许将其修改为除LOB/JSON/UDT/XMLTYPE外的数据类型。
  - 若为非空表，则不允许修改其数据类型。
- **使用字典编码的列（仅适用于TAC表）**：仅允许将其修改为字符型，但仍遵循字符型列的修改规则。
- **变长存储的字符型列（仅适用于TAC表）**：不允许在数据库进行回滚时，修改其数据类型。
- **分区键、函数索引或AC所在列**：不允许修改其数据类型。
- **原数据类型为LOB/JSON/UDT/XMLTYPE的其他列**：不允许修改其数据类型。
- **原数据类型为字符型的其他空列**：允许将其修改为除LOB/JSON/UDT/XMLTYPE外的其他数据类型。
- **原数据类型为字符型的其他非空列**：允许修改为不同字符型，但仍需遵项以下规则：
  - 不允许CHAR/VARCHAR与NCHAR/NVARCHAR交叉修改。
  - 当按长度属性从大向小修改时，必须保证该列现有数据的长度均未超过目标数据类型的上限，否则无法修改成功并提示相应错误。
  - HEAP表不允许跨存储方式修改数据类型，例如HEAP表不允许将列数据类型从VARCHAR(3200)（采用普通字符串存储）修改为VARCHAR(8004)（会自动转换为LOB型存储），而TAC表允许此操作。存储方式介绍请查阅字符型。
- **原数据类型为其他数据类型的其他空列**：允许将其修改为除LOB/JSON/UDT/XMLTYPE外的其他数据类型。
- **原数据类型为其他数据类型的其他非空列**：
  - 不允许跨大类修改数据类型，例如不允许将数值型修改为字符型。

- 同一数据大类中，除字符型外其他类型只允许按值域、精度、长度等属性从小向大修改数据类型，例如不允许将INT修改为指定了精度的NUMBER，不允许将DATE修改为TIME。

default

修改列字段的缺省值，default_expr可以为字面量、运算式、函数等表达式。

示例（HEAP表、TAC表）

```
ALTER TABLE area MODIFY area_no DEFAULT 3;
```

inline_constraint

修改列字段的约束项，遵循通用constraint里的行内约束所定义语法要求。

modfiy null/not null

修改NOT NULL约束项：

- 不允许对未定义NOT NULL约束项的列字段执行modify NULL。
- 不允许对已定义NOT NULL约束项的列字段执行modify NOT NULL。
- 不允许对已存在NULL数据的列字段执行modify NOT NULL。
- 不允许对已定义ON DELETE SET NULL或ON UPDATE SET NULL外键的列字段执行modify NOT NULL。

modify unique

修改UNIQUE约束项：

- 不允许对已定义UNIQUE约束项的列字段执行modify UNIQUE。
- 不允许对已存在重复数据的列字段执行modify UNIQUE。
- 不允许对多个列字段同时执行modify UNIQUE。

modify primary key

修改主键约束项：

- 不允许对已有主键的表执行任意列字段的modify PRIMARY KEY。
- 不允许对已定义UNIQUE约束项的列字段执行modify PRIMARY KEY。
- 不允许对不符合创建主键要求的列字段执行modify PRIMARY KEY，创建主键的要求请参考constraint描述。

modify check

修改CHECK约束项：

- 如新的CHECK与表中现有其他的CHECK冲突，无法执行修改。
- 如表中现有的数据不符合CHECK，无法执行修改。

modfiy foreign key

修改外键约束项：

- 不允许对不符合创建外键要求的列字段执行modify FOREIGN KEY，创建外键的要求请参考constraint描述。

示例（HEAP表、TAC表）

```
--修改branches非空表的branch_no非空列字段数据类型，只能修改为字符型
ALTER TABLE branches MODIFY branch_no VARCHAR(5);

--修改branches表的branch_no和branch_name为UNIQUE，无法同时修改多个列字段为UNIQUE，返回错误
ALTER TABLE branches MODIFY (branch_no,branch_name) UNIQUE;
YAS-04297 invalid ALTER TABLE option

--修改branches表的address列字段的缺省值
ALTER TABLE branches MODIFY address DEFAULT 'no address';

--修改area表的area_name列字段为非空
ALTER TABLE area MODIFY area_name NOT NULL;
```

alter_table_partition

该语句用于指定对表的分区的操作。

如在表上已建立了分区索引，当执行新增分区和删除分区的操作时，系统也会同时新增索引分区和删除索引分区。

分布式部署中不允许对一级分区执行本语句。

add_table_partition

该语句用于对表增加一个新的分区（Partition）。

不同类型的表可增加的分区类型不同：

- HEAP表支持增加范围（Range）、列表（List）和哈希（Hash）类型的分区。
- TAC表支持增加范围（Range）、列表（List）和哈希（Hash）类型的分区。
- LSC表支持增加范围（Range）和列表（List）类型的分区。

add_range_partition_clause

增加范围类型的分区，只能在当前最大分区界值之上建立分区，如表的最大分区界值被设为MAXVALUE，则不允许增加分区。

定义了INTERVAL的范围分区表，其分区由系统自动维护，不可以用本语句为其增加分区。

table_partition_description

可以为新增的分区指定所属表空间（缺省为表所属表空间），及PCTFREE/PCTUSED/INITRANS/MAXTRANS等属性。

可以为新增的分区指定range、list或者hash子分区，子分区的类型需要和表的定义一致。

如建表时未指定该表为组合分区表，不允许通过本语句为新增分区指定子分区。

示例（单机、共享集群部署）

```
ALTER TABLE sales_info_range ADD PARTITION p_sales_info_range_4 VALUES LESS THAN('2038');

--创建一张range-list组合分区表range_list_table
CREATE TABLE range_list_table(a INT, b INT)
PARTITION BY RANGE(a)
SUBPARTITION BY LIST(b)
(PARTITION p1 VALUES LESS than(1) (SUBPARTITION sp1 VALUES(10), SUBPARTITION sp2 VALUES(20)),
PARTITION p2 VALUES LESS than(2) (SUBPARTITION sp3 VALUES(10), SUBPARTITION sp4 VALUES(20)));

--为其添加一个range分区并指定其list子分区
ALTER TABLE range_list_table ADD PARTITION p3 VALUES LESS THAN(3)(SUBPARTITION sp5 VALUES(10),SUBPARTITION sp6 VALUES(20));
```

add_list_partition_clause

增加列表类型的分区，如表已在列表值里定义了DEFAULT，则不允许增加分区。

分区列表值可以指定为通用表达式（expr）。

table_partition_description

可以为新增的分区指定所属表空间（缺省为表所属表空间），及PCTFREE/PCTUSED/INITRANS/MAXTRANS等属性。

可以为新增的分区指定range、list或者hash子分区，子分区的类型需要和表的定义一致。

如建表时未指定该表为组合分区表，不允许通过本语句为新增分区指定子分区。

当分区列数量为一并且新增分区包含多个值时，每个值需要用括号包起来，否则报错。

示例（单机、共享集群部署）

```
ALTER TABLE sales_info_list ADD PARTITION p_sales_info_list_3 VALUES ('2022');
ALTER TABLE sales_info_list ADD PARTITION p_sales_info_list_4 VALUES (TO_CHAR(2023));

--创建一张list-list组合表list_list_table
CREATE TABLE list_list_table(a INT, b INT)
PARTITION BY LIST(a)
SUBPARTITION BY LIST(b)
```

```
(PARTITION p1 VALUES(1) (SUBPARTITION sp1 VALUES(1), SUBPARTITION sp2 VALUES(2)),
PARTITION p2 VALUES(2) (SUBPARTITION sp3 VALUES(1), SUBPARTITION sp4 VALUES(2)));

--为其添加一个list分区并指定其list子分区
ALTER TABLE list_list_table ADD PARTITION p3 VALUES (3)(SUBPARTITION sp5 VALUES(1), SUBPARTITION sp6 VALUES(2));
```

add_hash_partition_clause

增加哈希类型的分区。本语句只适用于HEAP/TAC表。

新增分区的PCTFREE/PCTUSED/INITRANS/MAXTRANS等属性从表上继承，不允许指定。

可以为新增的分区指定range、list或者hash子分区，子分区的类型需要和表的定义一致。

如建表时未指定该表为组合分区表，不允许通过本语句为新增分区指定子分区。

partition_storage_clause

可以为新增的分区指定所属表空间（缺省为表所属表空间）。

示例（HEAP表、单机TAC表）

```
ALTER TABLE sales_info_hash ADD PARTITION p_sales_info_hash_3;
ALTER TABLE sales_info_hash ADD PARTITION p_sales_info_hash_4;

--创建一张hash-list组合分区表hash_list_table
CREATE TABLE hash_list_table(c1 INT, c2 VARCHAR(10))
PARTITION BY HASH(c1)
SUBPARTITION BY LIST(c2)
(
PARTITION p1(SUBPARTITION sp1 VALUES('a')),
PARTITION p2 (SUBPARTITION sp3 VALUES('d'), SUBPARTITION sp4 VALUES(DEFAULT))
);
--为其添加一个hash分区并指定其list子分区
ALTER TABLE hash_list_table ADD PARTITION p3(SUBPARTITION sp5 VALUES ('f'));
```

drop_table_partition

该语句用于删除表的分区，同时删除分区里的数据。指定多个分区用`,`分隔。

该语句指定的分区中存有数据时，会同步失效表上的全局索引。分布式部署模式下，不管指定的分区是否存有数据，都会失效表上的全局索引。

该语句存在如下约束限制：

- 删除分区时，不允许将表上的所有分区都删除。
- 不允许删除哈希类型分区。
- 在表的分区被删除时，对应的索引分区也会被删除。
- 对于范围类型分区，删除分区后其分区界值将会向相邻大一级分区合并，这样符合此被删除分区界值范围的新增表数据将进入相邻分区中，如不存在相邻大一级分区则数据无法再插入成功。

示例（单机、共享集群部署）

```
--sales_info_range为一张范围分区表
SELECT * FROM sales_info_range;
YEAR  MONTH BRANCH PRODUCT      QUANTITY      AMOUNT SALSPERSON
----- ----- ------ --------- ----------- ----------- -------------
2001  01    0201   11001              30         500 0201010011
2000  12    0102   11001              20         300
2015  11    0101   11001              20         300
2015  03    0102   11001              20         300
2021  10    0101   11001              20         300
2021  05    0101   11001              40         600

--分区及数据被删除
ALTER TABLE sales_info_range DROP PARTITION p_sales_info_range_2;
SELECT * FROM sales_info_range;
YEAR  MONTH BRANCH PRODUCT      QUANTITY      AMOUNT SALSPERSON
```

```
----- ----- ------ --------- ----------- ----------- -------------
2001  01    0201   11001              30         500 0201010011
2000  12    0102   11001              20         300
2021  10    0101   11001              20         300
2021  05    0101   11001              40         600

INSERT INTO sales_info_range VALUES ('2015','03','0101','11001',20,300,'');
SELECT * FROM sales_info_range PARTITION(p_sales_info_range_3);
YEAR  MONTH BRANCH PRODUCT      QUANTITY      AMOUNT SALSPERSON
----- ----- ------ --------- ----------- ----------- -------------
2021  10    0101   11001              20         300
2021  05    0101   11001              40         600
2015  03    0101   11001              20         300
```

update_index_clause

指定分区被删除后，对表上的全局索引（Global Index）的处理，默认为INVALIDATE。

- INVALIDATE GLOBAL INDEXES：将全局索引全部失效，不可用。
- UPDAET GLOBAL INDEXES：将全局索引进行更新，仍然可用。

示例（单机、共享集群部署）

```
ALTER TABLE sales_info_list DROP PARTITION p_sales_info_list_4 UPDATE GLOBAL INDEXES;
```

drop_table_subpartition

该语句用于删除表的子分区，同时删除子分区里的数据。指定多个子分区用`,`分隔。

本语句存在如下约束限制：

- 删除子分区时，不允许将分区下的所有子分区都删除，也不允许跨分区删除子分区。
- 不允许删除哈希类型子分区。
- 在表的子分区被删除时，对应的索引子分区也会被删除。

示例

```
--获得sales_info表的子分区名称
SELECT partition_name,subpartition_name
FROM DBA_TAB_SUBPARTITIONS
WHERE table_name='SALES_INFO';
--以下输出以单机为例
PARTITION_NAME              SUBPARTITION_NAME
--------------------------- ---------------------------------------
P_SALES_INFO_1              P_SALES_INFO_1_SP_SALES_INFO_1
P_SALES_INFO_1              P_SALES_INFO_1_SP_SALES_INFO_2
P_SALES_INFO_1              P_SALES_INFO_1_SP_SALES_INFO_3
P_SALES_INFO_2              P_SALES_INFO_2_SP_SALES_INFO_1
P_SALES_INFO_2              P_SALES_INFO_2_SP_SALES_INFO_2
P_SALES_INFO_2              P_SALES_INFO_2_SP_SALES_INFO_3
P_SALES_INFO_3              P_SALES_INFO_3_SP_SALES_INFO_1
P_SALES_INFO_3              P_SALES_INFO_3_SP_SALES_INFO_2
P_SALES_INFO_3              P_SALES_INFO_3_SP_SALES_INFO_3

--选择其中一个子分区删除
ALTER TABLE sales_info DROP SUBPARTITION P_SALES_INFO_1_SP_SALES_INFO_1;
```

update_index_clause

同drop_table_partition语句中描述一致。

truncate_table_partition

该语句用于删除分区里的所有数据，指定删除多个分区数据用`,`分隔。

该语句同时删除分区对应的本地索引数据（Local Index）。

该语句指定的分区中存有数据时，会同步失效表上的全局索引。分布式部署模式下，不管指定的分区是否存有数据，都会失效表上的全局索引。

truncate_part_clause

该语句用于指定删除分区数据的方式。

drop|reuse storage

指定删除分区数据后释放|保留其存储空间。

cascade

如某张分区表为被子表定义了外键约束的父表，且要删除的分区数据在子表对应列字段值中已经存在，则指定CASCADE将同时删除子表中对应的数据行。本语句只作用于HEAP表。

purge

当回收站开启时，被删除的数据默认将进入回收站，指定本关键字则表示数据被彻底删除，不进入回收站。本语句只作用于HEAP表。

示例（单机、共享集群部署）

```
--获得sales_info表的分区名称
SELECT partition_name
FROM DBA_TAB_PARTITIONS
WHERE table_name='SALES_INFO';
--以下输出以单机为例
PARTITION_NAME
------------------------------------------------------------------
P_SALES_INFO_1
P_SALES_INFO_2
P_SALES_INFO_3

ALTER TABLE sales_info TRUNCATE PARTITION P_SALES_INFO_1 DROP STORAGE;
```

truncate_table_subpartition

该语句用于删除子分区里的所有数据，指定删除多个子分区数据用`,`分隔。

该语句同时删除子分区对应的本地索引数据（Local Index）。

truncate_part_clause

同truncate_table_partition语句中描述一致。

示例（单机、共享集群部署）

```
ALTER TABLE sales_info TRUNCATE SUBPARTITION P_SALES_INFO_3_SP_SALES_INFO_1 DROP STORAGE;
```

set_partition_clause

该语句只作用于HEAP表，且只针对范围分区表，用于指定INTERVAL和非INTERVAL分区类型之间的相互转换。转换分区类型不会影响表中原有的数据，新增加的数据则按转换后的分区类型特点存储。

分布式部署中用户无法执行此操作。

transfer range to range-interval

将RANGE分区表转换为INTERVAL分区时，要求分区键的数据类型为number或date，语法如下：

```
ALTER TABLE table_name SET INTERVAL(expr);
```

建立INTERVAL分区的要求和规则与CREATE TABLE里的interval_clause中描述一致，不符合时无法成功转换。

示例（HEAP表）

```
--创建示例表sales_info_range1，与样例表中sales_info_range区别仅在于分区键year的数据类型为int
CREATE TABLE sales_info_range1
(year INT NOT NULL,
```

```
month CHAR(2) NOT NULL,
branch CHAR(4),
product CHAR(5),
quantity NUMBER DEFAULT 0 NOT NULL,
amount NUMBER(10,2) DEFAULT 0 NOT NULL,
salsperson CHAR(10))
PARTITION BY RANGE(year)
(PARTITION p_sales_info_range_1 VALUES LESS THAN(2011),
PARTITION p_sales_info_range_2 VALUES LESS THAN(2021),
PARTITION p_sales_info_range_3 VALUES LESS THAN(2031));
--将sales_info_range1表的分区类型转换为INTERVAL类型
ALTER TABLE sales_info_range1 SET INTERVAL(2);

--对于超过现有分区界值的数据，系统将创建新分区且类型为INTERVAL
INSERT INTO sales_info_range1 VALUES ('2031','05','0101','11001',40,600,'');
COMMIT;
SELECT partition_name,tablespace_name,high_value
FROM USER_TAB_PARTITIONS
WHERE table_name='SALES_INFO_RANGE1'
ORDER BY partition_name;
PARTITION_NAME          TABLESPACE_NAME  HIGH_VALUE
----------------------- ---------------- -----------
P_SALES_INFO_RANGE_1    USERS            2011
P_SALES_INFO_RANGE_2    USERS            2021
P_SALES_INFO_RANGE_3    USERS            2031
SYS_P21                 USERS            2033
```

modify range-interval

修改INTERVAL分区的属性。

修改interval值

调整INTERVAL为新的值，之后插入的数据将按新值创建分区。

本功能语法为：

```
ALTER TABLE table_name SET INTERVAL(expr);
```

相关要求和规则与上述transfer range to range-interval一致。

修改分区表空间

指定INTERVAL分区的表空间，指定多个以`,`分隔，之后所有新创建的分区将循环使用这些表空间。

本功能语法为：

```
ALTER TABLE table_name SET STORE IN(tablespace_name, ...);
```

只能对INTERVAL类型的范围分区表执行此操作，否则系统报错。

示例（HEAP表）

```
--为sales_info_range1表新建两个表空间
CREATE TABLESPACE sales_tb1;
CREATE TABLESPACE sales_tb2;

--对上例中的INTERVAL分区表修改INTERVAL和表空间
ALTER TABLE sales_info_range1 SET INTERVAL(3);
ALTER TABLE sales_info_range1 SET STORE IN(sales_tb1,sales_tb2);
INSERT INTO sales_info_range1 VALUES ('2033','05','0101','11001',40,600,'');
INSERT INTO sales_info_range1 VALUES ('2043','05','0101','11001',40,600,'');
COMMIT;
SELECT partition_name,tablespace_name,high_value
FROM USER_TAB_PARTITIONS
WHERE table_name='SALES_INFO_RANGE1'
ORDER BY partition_name;
PARTITION_NAME          TABLESPACE_NAME      HIGH_VALUE
----------------------- -------------------- -----------
```

```
P_SALES_INFO_RANGE_1    USERS           2011
P_SALES_INFO_RANGE_2    USERS           2021
P_SALES_INFO_RANGE_3    USERS           2031
SYS_P21                 USERS           2033
SYS_P22                 SALES_TB1       2036
SYS_P23                 SALES_TB2       2045
```

transfer range-interval to range

将INTERVAL范围分区转换为非INTERVAL的范围分区，语法如下：

```
ALTER TABLE table_name SET INTERVAL();
```

示例（HEAP表）

```
--将上例中的INTERVAL分区表修改为非INTERVAL的范围分区表
ALTER TABLE sales_info_range1 SET INTERVAL();
```

modify_partition_clause

该语句用于对表分区进行操作，对于一级分区其中包含收缩操作和添加二级分区的操作，对于二级分区只有收缩操作。

shrink_space_clause

收缩操作与表的收缩一致，请参考shrink_space_clause。

示例（单机HEAP表）

```
ALTER TABLE orders_info ENABLE ROW MOVEMENT;
ALTER TABLE sales_info ENABLE ROW MOVEMENT;

ALTER TABLE orders_info MODIFY PARTITION p_orders_info_1 SHRINK SPACE COMPACT CASCADE;
ALTER TABLE sales_info MODIFY PARTITION P_SALES_INFO_1 ADD SUBPARTITION P_SALES_INFO_1_SP_SALES_INFO_4;
ALTER TABLE sales_info MODIFY SUBPARTITION P_SALES_INFO_3_SP_SALES_INFO_1 SHRINK SPACE COMPACT CASCADE;
```

split_table_partition

该语句能将一个范围或者列表分区重新划分为多个分区。当一个分区过于臃肿影响到了查询、备份等性能时，可以考虑SPLIT PARTITION。

只能对HEAP表和单机TAC表执行本语句。

该语句须遵循如下限制：

- 不可对组合分区表执行SPLIT PARTITION。
- UPDATE GLOBAL INDEXES为语法兼容，SPLIT PARTITION后会失效全局索引，请SPLIT后重建（rebuild_clause）全局索引。
- 若指定UPDATE INDEXES，则SPLIT PARTITION后不会失效LOCAL索引，会失效全局索引。
- 未指定分区名的表分区、索引分区、LOB分区会按照SYS_Pn的格式自动生成分区名。

split at (literal) into

该语句根据指定的字面量将一个范围分区划分为两个范围分区，仅适用于范围分区。

literal

划分值，该值须位于被划分的分区范围内，否则返回错误。

range_partition_desc

该语句用于指定划分后两个范围分区的名称，其中一个分区名可保留原分区名称，另一分区名不可与已有的分区名称重复。

新范围分区上下限分别为`[原范围分区下限，literal]`和`[literal，原范围分区上限]`。

示例（HEAP表，单机TAC表）

```
-- split range分区
DROP TABLE IF EXISTS split_range_part;
CREATE TABLE split_range_part(c1 INT)
```

```
PARTITION BY RANGE(c1)
(PARTITION p1 VALUES LESS than(100),
PARTITION p2 VALUES LESS than(200),
PARTITION p3 VALUES LESS than(300),
PARTITION p4 VALUES LESS than(MAXVALUE));
-- 将p4 split出p4和p5两个分区，新p4的分区边界值为350，p5的分区边界值为maxvalue
ALTER TABLE split_range_part split PARTITION p4 at(350) INTO (PARTITION p4, PARTITION p5);
```

split values (list_values) into

该语句根据指定的列表内容将一个列表分区划分为两个列表分区，仅适用于list分区。

list_values

划分的列表内容，其中的值须包含在被划分的列表分区内，否则返回错误。

list_partition_desc

该语句用于指定划分后两个列表分区的名称，其中一个分区名可保留原分区名称，另一分区名不可与已有的分区名称重复。

划分后的首个列表分区包含list_values中所有值，其余值保存在第二个列表分区中。

示例（HEAP表，单机TAC表）

```
-- split list分区
DROP TABLE IF EXISTS split_list_part;
CREATE TABLE split_list_part(c1 INT)
PARTITION BY LIST(c1)
(PARTITION p1 VALUES (100, 150, 170),
PARTITION p2 VALUES (200, 250, 280),
PARTITION p3 VALUES (300, 400),
PARTITION p4 VALUES (DEFAULT));
-- 将p4 split出p4和p5两个分区，新p4分区仅包含值350，p5分区包含default中除350之外所有值
ALTER TABLE split_list_part split PARTITION p4 VALUES(350) INTO (PARTITION p4, PARTITION p5);
```

split into (range/list_partition_desc)

该语句可将范围分区或列表分区根据指定内容划分为多个分区，适用于范围分区和列表分区。

range_partition_desc|list_partition_desc

该语句用于指定划分后的分区具体信息，包括分区名称、类型和分区值，新分区类型须与原分区类型保持一致，多个分区间使用`,`进行分隔。

划分后的其中一个分区名可保留原分区名称，其余分区名不可与已有的分区名称重复。

partition partname table_partition_description

该语句用于指定包含所有剩余值的分区名称及信息，名称不可与已有分区的名称重复。

示例（HEAP表，单机TAC表）

```
-- split list分区
DROP TABLE IF EXISTS split_list_part;
CREATE TABLE split_list_part(c1 INT)
PARTITION BY LIST(c1)
(PARTITION p1 VALUES (100, 150, 170),
PARTITION p2 VALUES (200, 250, 280),
PARTITION p3 VALUES (300, 400),
PARTITION p4 VALUES (DEFAULT));
-- 使用split into语法
ALTER TABLE split_list_part split PARTITION p4 INTO (PARTITION p4 VALUES(350), PARTITION p5);

-- split range分区
DROP TABLE IF EXISTS split_range_part;
CREATE TABLE split_range_part(c1 INT)
PARTITION BY RANGE(c1)
(PARTITION p1 VALUES LESS than(100),
PARTITION p2 VALUES LESS than(200),
PARTITION p3 VALUES LESS than(300),
PARTITION p4 VALUES LESS than(MAXVALUE));
-- 使用split into语法
ALTER TABLE split_range_part split PARTITION p4 INTO (PARTITION p4 VALUES LESS than(350), PARTITION p5);
```

add_subpartition_clause

该语句用于在组合分区表中给指定的一级分区添加二级分区。

该语句中range_subpartition_desc、list_subpartition_desc、individual_hash_subparts的描述请参考CREATE TABLE。

示例（HEAP表，单机TAC表）

```
CREATE TABLE composite_table(c1 INT, c2 INT)
PARTITION BY RANGE(c1)
SUBPARTITION BY HASH(c2)
(
PARTITION p1 VALUES LESS than(1) (SUBPARTITION sp1, SUBPARTITION sp2),
PARTITION p2 VALUES LESS than(2) (SUBPARTITION sp3, SUBPARTITION sp4)
);

ALTER TABLE composite_table MODIFY PARTITION p1 ADD SUBPARTITION p1_subp1;

--表创建时未指定为组合分区表，执行本语句会返回错误
CREATE TABLE partition_table(c1 INT,c2 INT)
PARTITION BY RANGE(c1)
INTERVAL (10)
(PARTITION par1 VALUES LESS THAN(100));

ALTER TABLE partition_table MODIFY PARTITION par1 ADD SUBPARTITION par1_subp1;
YAS-02374 table is not partitioned by composite partition method
```

alter_table_constraint

该语句用于指定对表上的约束的操作包括对添加约束、删除约束、修改约束和停用或启用约束。

add_constraint

该语句用于添加约束，同时增加多项在`()`中以`,`分隔。

out_of_line_constraint

约束项，关于约束项的详细描述请参考通用SQL语法constraint。

drop_constraint

该语句用于删除约束项。

如某个约束项在其创建时未指定名称，可以从系统提供视图（如USER_CONSTRAINTS）中查询到其默认名称后再执行删除操作。

删除UNIQUE/PRIMARY KEY约束项会自动删除对应的索引。

示例（HEAP表）

```
ALTER TABLE branches DROP CONSTRAINT c_branches_1;

--存在外键时无法删除主键约束项
ALTER TABLE department DROP PRIMARY KEY;
YAS-02188 this unique/primary key is referenced by some foreign keys
```

modify_constraint

该语句用于修改约束项的属性。

约束项名称

指定用于修改属性的约束项，可按如下三种方式指定（指定的约束项不存在时报错）：

- 对于主键约束项，以PRIMARY KEY表示即可。

- 对于唯一约束项，以UNIQUE表示即可，但需同时指定该约束项对应的列。
- 以CONSTRAINT 约束项名称表示，从系统视图（例如USER_CONSTRAINTS）可获得表上所有的约束项名称。

enable|disable

启用或停用指定的约束项（含义见constraint中的ENABLE|DISABLE描述）。

对于存在子表外键关联的约束项，指定DISABLE将无法成功停用，除非同时指定CASCADE。

在指定ENABLE启用约束项时，如表中现有数据无法满足约束规则，则启用失败，除非同时指定NOVALIDATE（但对于主键/唯一/NOT NULL约束项无效）。

validate|novalidate

启用或停用约束检查（含义见constraint中的VALIDATE|NOVALIDATE描述）。

本关键字可省略，则对ENABLE操作默认为VALIDATE，对DISABLE操作默认为NOVALIDATE。

cascade

指定CASCADE表示在DISABLE一个存在子表外键关联的约束项时，同时对子表的外键约束项执行DISABLE。

需注意的是，在重新ENABLE父表的该约束项时，即使指定CASCADE也不会对子表的外键约束项执行ENABLE，用户需手工操作。

本关键字可省略，则在DISABLE一个存在子表外键关联的约束项时默认不会对子表的外键约束项执行DISABLE，而是进行错误提示。

示例

```
--停用area表的主键约束，同时停用子表上的外键约束
ALTER TABLE area MODIFY PRIMARY KEY DISABLE CASCADE;

--在branches表上创建不启用的唯一约束，之后启用
ALTER TABLE branches ADD UNIQUE(branch_no, area_no) DISABLE;
ALTER TABLE branches MODIFY UNIQUE(branch_no, area_no) ENABLE;
```

示例（HEAP表）

```
--停用branches表的area_no外键约束
ALTER TABLE branches ADD CONSTRAINT c_branches_1
FOREIGN KEY (area_no) REFERENCES area(area_no) ON DELETE SET NULL;
ALTER TABLE branches MODIFY CONSTRAINT c_branches_1 DISABLE;
--修改area_no为在area表中不存在的值
UPDATE branches SET area_no='99' WHERE area_no='01';
COMMIT;
--启用branches表的area_no外键约束，但不启用约束检查，则可以启用成功
ALTER TABLE branches MODIFY CONSTRAINT c_branches_1 ENABLE NOVALIDATE;
```

enable_disable_constraint

该语句用于启用或者停用表上的某个约束项，拥有与modify_constraint子句除MODIFY外相同的关键字和含义，但顺序不相同。

示例（HEAP表）

```
--modify_constraint中的如下例句：
ALTER TABLE area MODIFY PRIMARY KEY DISABLE CASCADE;
ALTER TABLE branches MODIFY UNIQUE(branch_no, area_no) ENABLE;
ALTER TABLE branches MODIFY CONSTRAINT SYS_C_17 ENABLE NOVALIDATE;

--在enable_disable_constraint中可以如下表示：
ALTER TABLE area DISABLE PRIMARY KEY CASCADE;
ALTER TABLE branches ENABLE UNIQUE(branch_no, area_no);
ALTER TABLE branches ENABLE NOVALIDATE CONSTRAINT c_branches_1;
```

同时，本语句还提供如下与索引相关的语法选项（针对主键/唯一约束项）：

- using_index_clause
- (KEEP|DROP) INDEX

默认情况下，停用主键/唯一约束项会自动删除对应的索引，启用主键/唯一约束项会自动复用或者创建索引，上述两个选项用于对此情况进行人工干预。

using_index_clause

在启用主键/唯一约束项时，使用本语句可对系统自动复用或创建的索引进行人工指定，详细语法定义和描述见constraint中的using_index_clause。

(keep|drop) index

在停用主键/唯一约束项时，使用KEEP INDEX可指定不删除对应的索引，使用DROP INDEX则同默认情况，即删除对应的索引。

示例

```
--停用主键但保留对应索引
ALTER TABLE area DISABLE PRIMARY KEY CASCADE KEEP INDEX;

--停用主键同时删除对应索引
ALTER TABLE area DISABLE PRIMARY KEY CASCADE DROP INDEX;
```

ALTER TABLESPACE SET

通用描述

ALTER TABLESPACE SET语句用于更改分布式部署中一个已存在的表空间集的相关属性。

表空间集是YashanDB分布式部署中的一个逻辑存储单位，用于存储分布表及与分布表相关数据信息，物理上对应了各DN节点上的数据文件。

目前只能指定修改非内存映射表空间集。

> **Note**：
> 本文出现的所有CHUNK_NUM，均表示当前分布式部署环境中的Chunk总数量，该值可由建库参数USERS_DATASPACE_SCALE_OUT_FACTOR * 当前DN组个数计算得到结果，其中，建库参数USERS_DATASPACE_SCALE_OUT_FACTOR在安装过程中配置且后续不可修改，可咨询数据库管理员获得该参数的值。

语句定义

alter tablespace set::=

```
syntax::= ALTER TABLESPACE SET tablespace_set_name (MAXSIZE size_clause | NEXT size_clause | RESIZE size_clause|
databucket_clause)
```

databucket_clause**::=**

```
syntax::= add_databucket_clause | alter_databucket_clause | drop_databucket_clause
```

add_databucket_clause**::=**

```
syntax::= ADD DATABUCKET ((bucket_clause) {"," (bucket_clause)})
```

bucket_clause**::=**

```
syntax::= "'bucket_name'" [s3_bucket_clause] [MAXSIZE size_clause]
```

s3_bucket_clause**::=**

```
syntax::= S3 "(" URL "'url'" ["," REGION "'region'"] "," ACCESS KEY "'access_key'" "," SECRET KEY "'secret_key'" ")"
```

alter_databucket_clause**::=**

```
syntax::= ALTER DATABUCKET "'bucket_name'" (READONLY | READWRITE)
```

drop_databucket_clause**::=**

```
syntax::= DROP DATABUCKET "'bucket_name'"
```

tablespace_set_name

表空间集的名称。

maxsize size_clause

该语句用于修改表空间集的最大可扩展空间，不能小于原有的最大可扩展空间。

size_clause的取值范围为[128 * DB_BLOCK_SIZE * CHUNK_NUM, 4G * DB_BLOCK_SIZE * CHUNK_NUM]，当DB_BLOCK_SIZE参数为默认的8K值时，该范围为[1M * CHUNK_NUM, 32T * CHUNK_NUM]。

增加表空间集的最大可扩展空间会相应地增加所有DN组上的数据文件的最大可扩展空间：

1. 所有数据文件的最大可扩展空间之和构成了表空间集的最大可扩展空间。
2. 当增加表空间集的最大可扩展空间时，系统将新的最大空间均分给每个DN组，每个DN组先将已有的文件的最大值扩展到最大，然后继续建立新的文件。
3. 新建立的数据文件初始大小默认为1M ，除最后一个文件外的其它文件的最大可扩展空间为最大值。
4. 表空间集在每个DN组上最多有一个文件的最大可扩展空间不是最大值 。

示例（分布式部署）

```
--查看当前USERS表空间集最大的max_size (TSS_2390为系统默认的users表空间标识符)
SELECT SUM(max_size) FROM DV$DATAFILE WHERE name LIKE '%/TSS_2390%';

SUM(MAX_SIZE)
-------------
   2.3090E+13

--将USERS表空间集的maxsize扩大至22T
ALTER TABLESPACE SET users MAXSIZE 22T;
```

next size_clause

该语句用于修改表空间集内部的数据文件每次自动扩展的大小。

size_clause的取值范围为[512 * DB_BLOCK_SIZE, 32768 * DB_BLOCK_SIZE]，当DB_BLOCK_SIZE参数为默认的8K值时，该范围为[4M,256M]。

示例（分布式部署）

```
--接上例
ALTER TABLESPACE SET users NEXT 100M;

--查询修改后的数据文件信息
SELECT group_id||'_'||group_node_id dn_node,
SPLIT(name,'/',-1) filename,
TS#,BYTES,RELATIVE_FNO,AUTO_EXTEND,NEXT_SIZE,MAX_SIZE
FROM DV$DATAFILE
WHERE TS#=7
ORDER BY 3,1,2;
DN_NODE   FILENAME                    TS#      BYTES RELATIVE_FNO AUTO_EXTEND   NEXT_SIZE       MAX_SIZE
--------- -------------------------- ----- ---------- ------------ ----------- ----------- --------------
3-1       TSS_1800_CHUNK_0_FILE_0        7   68157440            0 ON            104857600   549755813888
3-1       TSS_1800_CHUNK_0_FILE_1        7    1048576            1 ON            104857600   549755813888
3-1       TSS_1800_CHUNK_0_FILE_2        7    1048576            2 ON            104857600   549755813888
3-1       TSS_1800_CHUNK_0_FILE_3        7    1048576            3 ON            104857600   549755813888
4-1       TSS_1800_CHUNK_1_FILE_0        7    1048576            0 ON            104857600   549755813888
4-1       TSS_1800_CHUNK_1_FILE_1        7    1048576            1 ON            104857600   549755813888
4-1       TSS_1800_CHUNK_1_FILE_2        7    1048576            2 ON            104857600   549755813888
4-1       TSS_1800_CHUNK_1_FILE_3        7    1048576            3 ON            104857600   549755813888
5-1       TSS_1800_CHUNK_2_FILE_0        7    1048576            0 ON            104857600   549755813888
5-1       TSS_1800_CHUNK_2_FILE_1        7    1048576            1 ON            104857600   549755813888
5-1       TSS_1800_CHUNK_2_FILE_2        7    1048576            2 ON            104857600   549755813888
5-1       TSS_1800_CHUNK_2_FILE_3        7    1048576            3 ON            104857600   549755813888
```

resize size_clause

该语句用于修改表空间集数据文件的大小。

size_clause的取值范围为[128 * DB_BLOCK_SIZE * CHUNK_NUM, MAXSIZE]，MAXSIZE表示当前表空间集的最大可扩展空间，当DB_BLOCK_SIZE参数为默认的8K时，最小值为1M * CHUNK_NUM。

当用户设置的size小于当前实际大小时，将不会进行修改。

示例（分布式部署）

```
--接上例
ALTER TABLESPACE SET users RESIZE 300M;

--查询修改后的数据文件信息
SELECT group_id||'_'||group_node_id dn_node,
SPLIT(name,'/',-1) filename,
TS#,BYTES,RELATIVE_FNO,AUTO_EXTEND,NEXT_SIZE,MAX_SIZE
FROM DV$DATAFILE
WHERE TS#=7
ORDER BY 3,1,2;
DN_NODE   FILENAME                    TS#      BYTES RELATIVE_FNO AUTO_EXTEND   NEXT_SIZE       MAX_SIZE
--------- -------------------------- ----- ---------- ------------ ----------- ----------- --------------
3-1       TSS_1800_CHUNK_0_FILE_0       7  104857600            0 ON          104857600   549755813888
3-1       TSS_1800_CHUNK_0_FILE_1       7    1048576            1 ON          104857600   549755813888
3-1       TSS_1800_CHUNK_0_FILE_2       7    1048576            2 ON          104857600   549755813888
3-1       TSS_1800_CHUNK_0_FILE_3       7    1048576            3 ON          104857600   549755813888
4-1       TSS_1800_CHUNK_1_FILE_0       7  104857600            0 ON          104857600   549755813888
4-1       TSS_1800_CHUNK_1_FILE_1       7    1048576            1 ON          104857600   549755813888
4-1       TSS_1800_CHUNK_1_FILE_2       7    1048576            2 ON          104857600   549755813888
4-1       TSS_1800_CHUNK_1_FILE_3       7    1048576            3 ON          104857600   549755813888
5-1       TSS_1800_CHUNK_2_FILE_0       7  104857600            0 ON          104857600   549755813888
5-1       TSS_1800_CHUNK_2_FILE_1       7    1048576            1 ON          104857600   549755813888
5-1       TSS_1800_CHUNK_2_FILE_2       7    1048576            2 ON          104857600   549755813888
5-1       TSS_1800_CHUNK_2_FILE_3       7    1048576            3 ON          104857600   549755813888
```

databucket_clause

该语句用于修改表空间集中的databucket（数据桶）信息。

add_databucket_clause

该语句用于增加databucket，可同时挂载多个，以`,`分隔。对于新建databucket的描述与约束与CREATE TABLESPACE SET章节中databucket_clause语句相同。

bucket_clause

同CREATE TABLESPACE SET章节中bucket_clause语句描述。

s3_bucket_clause

同CREATE TABLESPACE SET章节中s3_bucket_clause语句描述。

示例（分布式部署）

```
ALTER TABLESPACE SET tbs_tb ADD DATABUCKET '?/lscfile3' MAXSIZE 1G;
```

alter_databucket_clause

该语句用于修改databucket的读写属性，创建的databucket均默认为可读写属性。

bucket_name

对于不同的Databucket类型，bucket_name具有不同的含义：

- 对于本地存储的bucket，bucket_name为bucket在本地文件系统的相对路径。
- 对于S3 bucket，bucket_name为逻辑名称，用于在数据库内查询具体的bucket信息。

readonly|readwrite

readonly表示只读，readwrite表示可读写。

注意databucket修改为只读后不支持任何形式的写入操作，但用户执行的DML语句（非Bulkload操作）不受影响，执行Bulkload操作时由于需要生成SCOL数据，若表空间下没有可写入的databucket则会报错。

示例（分布式部署）

```
ALTER TABLESPACE SET tbs_tb ALTER DATABUCKET '?/lscfile3' READONLY;
```

drop_databucket_clause

该语句用于删除databucket，每次只允许指定一个databucket删除。

bucket_name

同alter_databucket_clause语句描述。

示例（分布式部署）

```
ALTER TABLESPACE SET tbs_tb DROP DATABUCKET '?/lscfile3';
```

ALTER TABLESPACE

通用描述

ALTER TABLESPACE语句用于更改一个已存在的表空间的相关属性。

语句定义

alter tablespace::=

```
syntax::= ALTER TABLESPACE tablespace_name (datafile_clause|
databucket_clause|
shrink_clause|
offline_clause|
ONLINE|
rename_clause)
```

datafile_clause**::=**

```
syntax::= ADD (DATAFILE|TEMPFILE) [(file_specification) {"," (file_specification)}]
| DROP (DATAFILE|TEMPFILE) file_name
```

file_specification**::=**

```
syntax::= "'file_name'" SIZE size_clause [AUTOEXTEND (OFF|(ON [NEXT size_clause] [MAXSIZE (UNLIMITED|size_clause)]))]
[PARALLEL parallel]
```

databucket_clause**::=**

```
syntax::= add_databucket_clause | alter_databucket_clause | drop_databucket_clause
```

add_databucket_clause**::=**

```
syntax::= ADD DATABUCKET ((bucket_clause) {"," (bucket_clause)})
```

bucket_clause**::=**

```
syntax::= "'bucket_name'" [s3_bucket_clause] MAXSIZE size_clause
```

s3_bucket_clause**::=**

```
syntax::= S3 "(" URL "'url'" ["," REGION "'region'"] "," ACCESS KEY "'access_key'" "," SECRET KEY "'secret_key'" ")"
```

alter_databucket_clause**::=**

```
syntax::= ALTER DATABUCKET "'bucket_name'" (READONLY | READWRITE)
```

drop_databucket_clause**::=**

```
syntax::= DROP DATABUCKET "'bucket_name'"
```

shrink_clause**::=**

```
syntax::= SHRINK SPACE [KEEP size_clause]
```

offline_clause::=

```
syntax::= OFFLINE [NORMAL|TEMPORARY|IMMEDIATE]
```

rename_clause::=

```
syntax::= RENAME TO space_name
```

datafile_clause

该语句用于修改表空间的数据文件。

分布式部署中，仅允许对通过CREATE TABLESPACE语句创建的表空间新增/删除数据文件，若修改表空间数据文件时出现节点故障，恢复措施见用户表空间管理章节描述。

add (datafile|tempfile)

为表空间增加一个或多个数据文件（临时文件需使用TEMPFILE关键字）。对于本地临时表空间以及本地SWAP表空间而言，添加一个文件等于添加一组文件，每一组文件数量的个数等于集群实例个数。

file_specification的描述请参考CREATE TABLESPACE。

当不指定file_specification时，系统按如下规则自动创建一个数据文件：

- 文件名称由表空间名称以及数据文件在表空间内的序号组合生成，如：tablespace_name1，tablespace_name2...，且统一转换为大写。
- 文件的默认大小为8192个BLOCK，文件路径为系统默认的数据文件路径。
- 对于非MEMORY MAPPED表空间，默认文件开启自动扩展，next为8192个块，maxsize为64MB个块。
- 如果没有显式的规定extent分配方式，extent的默认分配方式为系统自动分配。

drop (datafile|tempfile)

删除表空间的某个数据文件（临时文件需使用TEMPFILE关键字）。对于本地临时表空间以及本地SWAP表空间而言，删除一个文件等于删除一组文件，每一组文件数量的个数等于集群实例个数。

> **Note**：
> 不允许删除创建该表空间时指定的第一个数据文件以及非ONLINE的数据文件。
> 不允许删除UNDO表空间的数据文件。
> MEMORY MAPPED表空间的数据文件删除后该文件内部ID以及全局ID在该表空间被删除并重启数据库之前都不允许被复用。
> 当增加数据文件导致文件的表空间内部ID到达表空间限制（63）时，删除已有文件也无法向该表空间添加数据文件。
> 如果删除数据文件中途数据库掉电，重启后磁盘可能会有残留文件，此时需要手动删除。

示例（单机、共享集群部署）

```
ALTER TABLESPACE SYSTEM ADD DATAFILE 'sys1' SIZE 4M AUTOEXTEND ON NEXT 4M MAXSIZE 1G;

ALTER TABLESPACE yashan ADD DATAFILE 'yashan_add1' SIZE 4M AUTOEXTEND ON NEXT 4M MAXSIZE 1G;

ALTER TABLESPACE yashan ADD DATAFILE 'yashan_add2' SIZE 4M AUTOEXTEND ON NEXT 4M MAXSIZE 1G PARALLEL 4;

ALTER TABLESPACE yashan ADD DATAFILE;

ALTER TABLESPACE SYSTEM DROP DATAFILE 'sys1';

ALTER TABLESPACE yashan DROP DATAFILE 'yashan_add1';
```

databucket_clause

该语句用于修改表空间的databucket（数据桶）信息。

add_databucket_clause

为表空间增加databucket，可同时挂载多个，以`,`分隔。对于新建databucket的描述与约束与CREATE TABLESPACE章节中databucket_clause语句相同。

bucket_clause
同CREATE TABLESPACE章节中bucket_clause语句描述。

s3_bucket_clause
同CREATE TABLESPACE章节中s3_bucket_clause语句描述。

示例（单机、分布式部署）

```
ALTER TABLESPACE lsc_tb ADD DATABUCKET '?/lscfile_add3' MAXSIZE 1G;
```

alter_databucket_clause
该语句用于修改databucket的读写属性，创建的databucket均默认为可读写属性。

分布式部署中用户无法执行此操作。

bucket_name
对于不同的databucket类型，bucket_name具有不同的含义：

- 对于本地存储的bucket，bucket_name为bucket在本地文件系统的绝对路径或相对路径（分布式部署中不允许指定绝对路径）。
- 对于S3 bucket，bucket_name为逻辑名称，用于在数据库内查询具体的bucket信息。

readonly|readwrite
readonly表示只读，readwrite表示可读写。

注意databucket修改为只读后不支持任何形式的写入操作，但用户执行的DML语句（非Bulkload操作）不受影响，执行Bulkload操作时由于需要生成SCOL数据，若表空间下没有可写入的databucket则会报错。

示例（单机部署）

```
ALTER TABLESPACE lsc_tb ALTER DATABUCKET '?/lscfile_add3' READONLY;
```

drop_databucket_clause
删除表空间的某个databucket，每次只允许指定一个databucket删除。

分布式部署中用户无法执行此操作。

bucket_name
同alter_databucket_clause语句描述。

示例（单机部署）

```
ALTER TABLESPACE lsc_tb DROP DATABUCKET '?/lscfile_add3';
```

shrink_clause
该语句用于根据各个数据文件的空间使用情况回收空闲空间，收缩表空间大小。

共享集群部署中不允许执行shrink_clause。

> **Note**：
> - 不能对UNDO表空间执行该语句。
> - 在主库上对SWAP和TEMP表空间进行shrink，不会同步到备库。

keep size_clause

将表空间收缩至指定大小，如表空间实际使用已超过该大小值，则收缩失败并返回错误。

不指定KEEP时，系统将最大化收缩表空间。

示例（单机、分布式部署）

```
ALTER TABLESPACE SYSTEM SHRINK SPACE KEEP 100M;

ALTER TABLESPACE SYSTEM SHRINK SPACE;
```

offline_clause

改变表空间至脱机状态。本操作一般应用于如下业务场景：

- 数据库的部分数据不再开放访问，则可以将其所在表空间offline
- 需要对表空间里的数据文件进行重命名或重分配时，将表空间offline，则其下所有的数据文件均会被offline

当某个表空间被offline后，该表空间下的所有数据文件将不可读写也不可修改，且不能对该表空间执行DDL操作，包括增删数据文件等操作。

SYSTEM/SYSAUX/UNDO/SWAP/TEMPORARY等内置表空间不允许offline。

共享集群部署中不允许执行offline_clause。

> **Note**：
> 被offline的表空间及其下数据文件仍为数据库的一部分，因此不能创建与它们同名的表空间或数据文件。

normal| temporary | immediate

offline选项，可省略，默认为NORMAL。

OFFLINE NORMAL：当表空间所有的数据文件均没有脱机时，才会将表空间及数据文件offline，否则报错。

OFFLINE TEMPORARY：允许表空间中存在已经脱机的数据文件，只对online的数据文件执行offline。

OFFLINE IMMDIATE：对非脱机的数据文件立即执行offline，可能存在归档或redo日志没有完全应用于数据文件，处于不一致状态，所以该操作offline的数据文件无法直接online。当数据库未开启归档模式时，不允许指定此选项执行offline。

示例（单机、分布式部署）

```
ALTER TABLESPACE yashan OFFLINE;

ALTER TABLESPACE yashan1 OFFLINE NORMAL;

ALTER TABLESPACE yashan2 OFFLINE TEMPORARY;

ALTER TABLESPACE yashan3 OFFLINE IMMEDIATE;
```

online

改变脱机表空间至在线状态，使其正常可用。

本操作将表空间及其下的所有数据文件online，如果存在无法online的数据文件，则执行失败并返回错误。

共享集群部署中不允许执行online。

示例（单机、分布式部署）

```
ALTER TABLESPACE yashan ONLINE;
```

rename_clause

该语句用于表空间或数据文件重命名，限制如下：

- 不允许将某个表空间重命名为当前已存在的表空间名称。

- 不允许使用该语句对内置表空间进行重命名。
- 不允许使用该语句对OFFLINE的表空间进行重命名。
- 不允许使用该语句对正在使用的SWAP表空间进行重命名，如需对此进行重命名，先修改DEFAULT_SWAP_TABLESPACE配置参数，再进行重命名操作。

示例

```
ALTER TABLESPACE yashan1 RENAME TO yashanDb;
```

ALTER TRIGGER

通用描述

ALTER TRIGGER语句用于更改一个已存在的触发器的相关属性或显式的重编译一个触发器。

对于在SYS schema内的触发器，需要由SYS用户执行ALTER TRIGGER语句。

对于其他schema内的触发器，需要由其所属用户或拥有ALTER ANY TRIGGER权限的用户执行ALTER TRIGGER语句。

触发器含义及限制请参考触发器章节。

语句定义

alter_trigger::=

```
syntax::= ALTER TRIGGER [schema "."] trigger_name
  (trigger_compile_clause
  | (ENABLE | DISABLE)
  | RENAME TO new_name
  | (EDITIONABLE | NONEDITIONABLE))
```

schema

包含触发器的模式名称，省略则默认为当前登录用户的模式。

trigger_name

需要更改或重编译的触发器的名称。

trigger_compile_clause

指定重编译选项，详见compile_clause描述。

enable|disable

启用或禁用触发器。

rename to new_name

修改触发器的名称。

修改触发器名称时，指定的新名称不能为空且必须符合YashanDB的对象命名规范。

editionable | noneditionable

用于语法兼容，无实际含义。

示例（单机、共享集群部署）

```
-- 显式重编译sales模式的tri触发器
ALTER TRIGGER sales.tri COMPILE;

-- 禁用sales模式的tri触发器
ALTER TRIGGER sales.tri DISABLE;

-- 启用sales模式的tri触发器
ALTER TRIGGER sales.tri ENABLE;

-- 重命名sales模式的tri触发器
ALTER TRIGGER sales.tri RENAME TO newTri;
ALTER TRIGGER sales.newTri RENAME TO tri;
```

ALTER TYPE

通用描述

ALTER TYPE语句用于更改一个已存在的自定义类型（UDT）的相关属性。

自定义类型含义及限制请参考自定义类型章节。

语句定义

alter_type::=

```
syntax::=  ALTER TYPE  [ schema "." ]  type_name
( EDITIONABLE | NONEDITIONABLE )
| type_compile_clause
```

type_compile_clause**::=**

```
syntax::=  COMPILE [DEBUG] [(SPECIFICATION | BODY)] [(compiler_parameters_clause) {(compiler_parameters_clause)}] [REUSE
SETTINGS]
```

schema

包含UDT的模式名称，省略则默认为当前登录用户的模式。

type_name

要更改的UDT的名称。

editionable | noneditionable

用于语法兼容，无实际含义。

type_compile_clause

指定重编译选项，详见compile_clause描述。

示例（单机、共享集群部署）

```
-- 显式重编译udt_object类型，如果存在对应类型主体，会同时重编译。
ALTER TYPE udt_object COMPILE;

-- 显式重编译udt_object类型。（不会重编译类型主体）
ALTER TYPE udt_object COMPILE SPECIFICATION;

-- 显式重编译udt_object类型的类型主体。（不会重编译类型）
ALTER TYPE udt_object COMPILE BODY;

-- 显式重编译udt_varray类型。
ALTER TYPE udt_varray COMPILE;
```

ALTER USER

通用描述

ALTER USER用于修改用户属性，包括密码、默认表空间等。YashanDB的用户管理体系请参考产品安全手册用户管理。

语句定义

alter user::=

```
syntax::= ALTER USER user_name (IDENTIFIED BY [VALUES] password
|DEFAULT TABLESPACE tablespace
|PASSWORD EXPIRE
|ACCOUNT (LOCK|UNLOCK)
|PROFILE profilename)
{" " (IDENTIFIED BY [VALUES] password
|DEFAULT TABLESPACE tablespace
|PASSWORD EXPIRE
|ACCOUNT (LOCK|UNLOCK)
|PROFILE profilename)}
```

user_name

已存在的用户名。

values

该语句用于指定按密文修改用户的密码，若省略则按明文修改密码。

YashanDB的密码策略为加密传输和加密存储，即服务端存储的是密文密码。在某些特定场景（例如数据库迁移），可通过直接指定密文来修改密码，该操作不影响用户前端输入明文密码登录。

password

为用户指定的新密码，新密码需遵循与CREATE USER中一致的password要求。

tablespace

为用户指定默认表空间。

account (lock|unlock)

锁定/解锁用户，锁定后该用户将不可登录。

password expire

使用户密码失效，用户密码失效后，用户无法登录，需要重新设置密码后方可登录。

profile profilename

为用户指定新的profile，profile见CREATE PROFILE中描述。

示例

```
ALTER USER sales1 IDENTIFIED BY "23%ad1";

ALTER USER sales2 DEFAULT TABLESPACE users;

ALTER USER sales3 PASSWORD EXPIRE;

ALTER USER sales3 ACCOUNT UNLOCK;

ALTER USER sales3 PASSWORD EXPIRE ACCOUNT LOCK;
```

```
ALTER USER sales3 PROFILE DEFAULT;
```

ANALYZE DATABASE

通用描述

ANALYZE DATABASE用于收集数据库的统计信息，与之对应的是DBMS_STATS高级包的GATHER_DATABASE_STATS程序。

语句定义

analyze database::=

```
syntax::= ANALYZE DATABASE [(OPTIONS options|ESTIMATE_PERCENT estimate_value|PARALLEL_DEGREE parallel_value|METHOD_OPTION
method_clause|GRANULARITY "'" graularity_value "'"|INDEX_CASCADE index_value|GATHER_SYS gather_sys_value)
{" " (OPTIONS options|ESTIMATE_PERCENT estimate_value|PARALLEL_DEGREE parallel_value|METHOD_OPTION method_clause|GRANULARITY
"'" graularity_value "'"|INDEX_CASCADE index_value|GATHER_SYS gather_sys_value)}]
```

options

数据库统计信息收集选项，可省略，则为GATHER AUTO，等同于指定DBMS_STATS中高级包GATHER_DATABASE_STATS的参数options。

estimate_percent

指定统计的采样率，可省略，等同于指定DBMS_STATS中高级包GATHER_DATABASE_STATS的参数estimate_percent。

parallel_degree

指定并行度，对于大表，增大并行度可以提升统计信息收集的效率，可省略，则parallel_value默认为1，等同于指定DBMS_STATS中高级包GATHER_DATABASE_STATS的参数degree。

method_option

指定列统计信息选项，可省略，等同于指定DBMS_STATS中高级包GATHER_DATABASE_STATS的参数method_option。

granularity

指定分区统计粒度，可省略，等同于指定DBMS_STATS中高级包GATHER_DATABASE_STATS的参数granularity。

index_cascade

指定是否收集索引统计信息（true/false），可省略，则index_value默认为false，等同于指定DBMS_STATS中高级包GATHER_DATABASE_STATS的参数cascade。

gather_sys

指定是否收集系统表的统计信息（true/false），可省略，则默认为false，等同于指定DBMS_STATS中高级包GATHER_DATABASE_STATS的参数gather_sys。

示例（分布式部署）

```
ANALYZE DATABASE OPTIONS 'GATHER AUTO' ESTIMATE_PERCENT 1 PARALLEL_DEGREE 2 METHOD_OPTION 'FOR ALL COLUMNS SIZE AUTO'
GRANULARITY 'AUTO' INDEX_CASCADE TRUE GATHER_SYS FALSE;
```

ANALYZE SCHEMA

通用描述

ANALYZE SCHEMA用于收集指定用户下所有对象（表、AC、列、索引）的统计信息，与之对应的是DBMS_STATS高级包的GATHER_SCHEMA_STATS程序。

语句定义

analyze schema::=

```
syntax::= ANALYZE SCHEMA OWNER [(ESTIMATE_PERCENT estimate_value|BLOCK_SAMPLE block_value|METHOD_OPTION
method_clause|PARALLEL_DEGREE parallel_value|GRANULARITY "'" graularity_value "'"|INDEX_CASCADE index_value)
{" " (ESTIMATE_PERCENT estimate_value|BLOCK_SAMPLE block_value|METHOD_OPTION method_clause|PARALLEL_DEGREE
parallel_value|GRANULARITY "'" graularity_value "'"|INDEX_CASCADE index_value)}]
```

estimate_percent

指定统计的采样率，可省略，等同于指定DBMS_STATS中高级包GATHER_SCHEMA_STATS的参数estimate_percent。

block_sample

指定是否采用块级采样，预留字段，恒为TRUE，可省略，等同于指定DBMS_STATS中高级包GATHER_SCHEMA_STATS的参数block_sample。

method_option

指定列统计信息选项，可省略，等同于指定DBMS_STATS中高级包GATHER_SCHEMA_STATS的参数method_option。

parallel_degree

指定并行度，对于大表，增大并行度可以提升统计信息收集的效率，可省略，则parallel_value默认为1，等同于指定DBMS_STATS中高级包GATHER_SCHEMA_STATS的参数degree。

granularity

指定分区统计粒度，可省略，等同于指定DBMS_STATS中高级包GATHER_SCHEMA_STATS的参数granularity。

index_cascade

指定是否收集索引统计信息（true/false），可省略，则index_value默认为false，等同于指定DBMS_STATS中高级包GATHER_SCHEMA_STATS的参数cascade。

示例（分布式部署）

```
ANALYZE SCHEMA SALES ESTIMATE_PERCENT 1 BLOCK_SAMPLE TRUE METHOD_OPTION 'FOR ALL COLUMNS SIZE AUTO' PARALLEL_DEGREE 1
GRANULARITY 'ALL' INDEX_CASCADE TRUE;
```

ANALYZE TABLE

通用描述

ANALYZE TABLE用于收集分布式部署中的表、AC或分区的统计信息，与之对应的是DBMS_STATS高级包的GATHER_TABLE_STATS程序。

对于组合分区表，不可指定收集某个分区/子分区，仅收集其GLOBAL级统计信息。

语句定义

analyze table::=

```
syntax::= ANALYZE TABLE table_name [(PARTITION partition_value|ESTIMATE_PERCENT estimate_value|BLOCK_SAMPLE
block_value|METHOD_OPTION method_clause|PARALLEL_DEGREE parallel_value|GRANULARITY "'" graularity_value "'"|INDEX_CASCADE
index_value)
{" " (PARTITION partition_value|ESTIMATE_PERCENT estimate_value|BLOCK_SAMPLE block_value|METHOD_OPTION
method_clause|PARALLEL_DEGREE parallel_value|GRANULARITY "'" graularity_value "'"|INDEX_CASCADE index_value)}]
```

method_clause::=

```
syntax::= "'" (FOR ALL COLUMNS [size_clause]
| FOR COLUMNS "(" (column_name) {"," (column_name)} ")" [size_clause]
) "'"
```

size_clause::=

```
syntax::=  SIZE (integer|AUTO)
```

partition

按指定的分区名收集统计信息，可省略，则partition_value默认为NULL，表示不按分区，收集整张表信息，等同于在DBMS_STATS中高级包GATHER_TABLE_STATS的参数partname。

estimate_percent

指定统计的采样率，可省略，等同于指定DBMS_STATS中高级包GATHER_TABLE_STATS的参数estimate_percent。

block_sample

指定是否采用块级采样，预留字段，恒为TRUE，可省略，等同于指定DBMS_STATS中高级包GATHER_TABLE_STATS的参数block_sample。

method_option

指定列统计信息选项，可省略，等同于指定DBMS_STATS中高级包GATHER_TABLE_STATS的参数method_option。

size_clause

指定直方图信息，可省略，则默认为SIZE AUTO。

- integer：直方图的bucket数量，范围为[1,2048]。
- AUTO： 由系统决定是否生成直方图。

parallel_degree

指定并行度，对于大表，增大并行度可以提升统计信息收集的效率，可省略，则parallel_value默认为1，等同于指定DBMS_STATS中高级包GATHER_TABLE_STATS的参数degree。

granularity

指定分区统计粒度，可省略，等同于指定DBMS_STATS中高级包GATHER_TABLE_STATS的参数granularity。

index_cascade

指定是否收集索引统计信息（true/false），可省略，则index_value默认为false，等同于指定DBMS_STATS中高级包GATHER_TABLE_STATS的参数cascade。

示例（分布式部署）

```
ANALYZE TABLE branches PARTITION NULL ESTIMATE_PERCENT 0.5 BLOCK_SAMPLE true
INDEX_CASCADE true METHOD_OPTION 'FOR COLUMNS (branch_no, area_no) SIZE AUTO'
GRANULARITY 'AUTO' PARALLEL_DEGREE 1;
```

AUDIT POLICY

通用描述

AUDIT POLICY对已创建的审计策略进行使能，只有使能的审计策略才会在数据库操作符合审计策略的审计项时触发审计。

只有拥有AUDIT_ADMIN审计管理员角色，或者拥有AUDIT SYSTEM系统权限的用户，才能执行某个审计策略的使能操作。

按照AUDIT POLICY定义规则使能后的审计策略，在配置参数UNIFIED_AUDITING为true的情况下，系统将对数据库的操作进行审计，用户可通过结果UNIFIED_AUDIT_TRAIL视图查看相应审计记录。

语句定义

audit_policy::=

```
syntax::= AUDIT POLICY policy_name [((BY | EXCEPT) (user_name) {"," (user_name)})] [WHENEVER [NOT] SUCCESSFUL]
```

policy_name

要使能的审计策略名称。

by|except

指定要审计的操作用户或者排除不需要审计的操作用户，同时指定或排除多个用户以`,`分隔。本语句可省略，则默认审计所有的操作用户。

whenever [not] successful

指定在数据库操作执行成功|失败时才进行审计记录。不使用本语句则表示针对满足审计项的数据操作，无论其执行成功或失败都将被审计。

示例

```
-- 使能审计策略，策略对应用户sales，在策略对应的审计项执行成功时进行审计
AUDIT POLICY up1 BY sales WHENEVER SUCCESSFUL;

-- 使能审计策略，排除用户sales,在策略对应的审计项执行成功或失败都进行审计
AUDIT POLICY up2 EXCEPT sales;
```

BACKUP ARCHIVELOG

通用描述

BACKUP ARCHIVELOG用于执行对当前数据库归档日志文件的备份。

在`DBA_ARCHIVE_BACKUPSET`视图中查询生成的备份集信息。

在`V$BACKUP_PROGRESS`视图中查询备份的过程信息。

只能在数据库状态为OPEN时备份数据库，且归档模式必须为开启状态（使用ALTER DATABASE语句调整归档模式）。

关于备份恢复的详细操作描述请参考备份恢复。

该语句仅适用于单机和共享集群部署。

语句定义

backup archivelog::=

```
syntax::= BACKUP ARCHIVELOG archivelogRangeSpecifier [backupCommonSpecifier]
```

archive_log_range_specifier**::=**

```
syntax::= ((
(FROM SCN | SCN BETWEEN interger AND | UNTIL SCN ) interger |
(FROM SEQUENCE | SEQUENCE BETWEEN interger AND | UNTIL SEQUENCE ) interger [THREAD interger] |
(FROM TIME | TIME BETWEEN date_string AND | UNTIL TIME ) date_string | ALL))
```

backup_common_specifier**::=**

```
syntax::= ((COMPRESSION [ALGORITHM (ZSTD|LZ4) [LOW|MEDIUM|HIGH]])|
(ENCRYPTION [AES128|AES192|AES256|SM4] IDENTIFIED BY password)|
(FORMAT backup_path)|(TAG tag_name)|
PARALLELISM integer|
SECTION SIZE size_clause)
{" " ((COMPRESSION [ALGORITHM (ZSTD|LZ4) [LOW|MEDIUM|HIGH]])|
(ENCRYPTION [AES128|AES192|AES256|SM4] IDENTIFIED BY password)|
(FORMAT backup_path)|(TAG tag_name)|
PARALLELISM integer|
SECTION SIZE size_clause)}
```

archivelograngespecifier

用于指定需备份的归档日志范围。

若范围区间内有指定的归档日志文件不存在（例如文件已被删除、某一序列号对应的归档日志文件暂未生成等）现象，备份操作会失败并提示错误码YAS-02544。

scn

用于指定需备份的归档日志SCN范围，SCN为64位无符号整数。

通过V$ARCHIVED_LOG视图可以获取当前已存在的归档日志文件中第一条日志的SCN（FIRST_CHANGE#字段）、最后一条日志的SCN（NEXT_CHANGE#字段）以及其他详细信息。

示例（单机、共享集群部署）

```
BACKUP ARCHIVELOG SCN BETWEEN 505411989407080448 AND 505412089999663104 FORMAT 'SCNRANGE' TAG 'SCNRANGE' COMPRESSION;
```

sequence

用于指定需备份的归档日志序列号范围，序列号为32位无符号整数。

通过V$ARCHIVED_LOG视图可以获取当前已存在的归档日志文件的序列号（SEQUENCE#字段）以及其他详细信息。

示例（单机、共享集群部署）

```
BACKUP ARCHIVELOG SEQUENCE BETWEEN 2 AND 10 FORMAT 'ASNRANGE' TAG 'ASNRANGE' PARALLELISM 8 SECTION SIZE 128M;
```

time

用于指定需备份的归档日志的时间范围，最早默认不早于当前环境的安装时间，最晚可以设置为未来时间但实际备份的内容截止于最大归档文件序列号所对应文件中的最后一条日志的SCN所对应的时间。

示例（单机、共享集群部署）

```
BACKUP ARCHIVELOG TIME BETWEEN TO_DATE('2023-10-28 08:14:01','yyyy-mm-dd hh24:mi:ss') AND TO_DATE('2023-11-28
11:14:01','yyyy-mm-dd hh24:mi:ss') FORMAT 'TIMERANGE' TAG 'TIMERANGE' COMPRESSION;
```

from

在归档备份语句中，与SCN、SEQUENCE或TIME配合使用指定归档日志备份的起点。

指定FROM指令备份归档日志时，备份的终点为当前数据库已存在的最大归档文件序列号所对应文件中的最后一条日志，通过V$ARCHIVED_LOG视图可以获取当前已存在的归档日志文件相关信息。

若指定的起点大于（或晚于）终点，则提示错误码YAS-02541，没有符合条件的归档文件。

示例（单机、共享集群部署）

```
BACKUP ARCHIVELOG FROM SCN 505412089999663104 FORMAT 'FROMSCN' TAG 'FROMSCN' COMPRESSION;

BACKUP ARCHIVELOG FROM SEQUENCE 10 FORMAT 'FROMASN' TAG 'FROMASN' PARALLELISM 8 SECTION SIZE 128M;

BACKUP ARCHIVELOG FROM TIME TO_DATE('2023-11-28 11:14:01','yyyy-mm-dd hh24:mi:ss')  FORMAT 'FROMTIME' TAG 'FROMTIME'
COMPRESSION;
```

between … and …

在归档备份语句中，与SCN、SEQUENCE或TIME配合使用指定归档日志备份的区间。

指定BETWEEN …（起点） AND …（终点）指令备份归档日志时，备份的起点和终点均为指定值，通过V$ARCHIVED_LOG视图可以获取当前已存在的归档日志文件相关信息并按需选定起点和终点。

若指定的起点大于（或晚于）终点，则提示错误码YAS-02541，没有符合条件的归档文件。

until

在归档备份语句中，与SCN、SEQUENCE或TIME配合使用指定为归档日志备份的终点。

指定UNTIL指令备份归档日志时，备份的起点为当前数据库已存在的最小归档文件序列号所对应文件中的第一条日志，通过V$ARCHIVED_LOG视图可以获取当前已存在的归档日志文件相关信息。

若指定的终点小于（或早于）起点，则提示错误码YAS-02541，没有符合条件的归档文件。

示例（单机、共享集群部署）

```
BACKUP ARCHIVELOG UNTIL SCN  505412089999663104 FORMAT 'UNTILSCN' TAG 'UNTILSCN' COMPRESSION;

BACKUP ARCHIVELOG UNTIL SEQUENCE 10 FORMAT 'UNTILASN' TAG 'UNTILASN' PARALLELISM 8 SECTION SIZE 128M;

BACKUP ARCHIVELOG UNTIL TIME TO_DATE('2023-11-28 11:14:01','yyyy-mm-dd hh24:mi:ss')  FORMAT 'UNTILTIME' TAG 'UNTILTIME'
COMPRESSION;
```

backupcommonspecifier

用于指定备份操作的压缩、加密、并行度等通用配置。

compression

该语句用于压缩备份集，COMPRESSION后的选项都省略时，表示按ZSTD算法和LOW级别压缩备份数据。

增量备份的每个备份集可采用不同的压缩算法及级别，不影响对其的恢复。

压缩算法

通过ALGORITHM关键字指定，YashanDB提供如下两种压缩算法供用户选择：

- ZSTD：可提供更高的压缩率。
- LZ4：可提供更高的压缩速率。

压缩级别

在指定压缩算法时可同时指定压缩级别，可省略，则缺省采用LOW级别。

压缩级别包含HIGH、MEDIUM、LOW三个选项，其中HIGH表示最高的压缩率（最低的压缩速度），LOW表示最低的压缩率（最高的压缩速度）。

示例（单机、共享集群部署）

```
BACKUP ARCHIVELOG ALL COMPRESSION;
BACKUP ARCHIVELOG SEQUENCE BETWEEN 2 AND 10 COMPRESSION ALGORITHM lz4 HIGH;
```

encryption

该语句用于加密备份集。

增量备份的每个备份集需要保证统一的都加密或者都不加密，且每个备份集的密钥必须保持一致。

加密算法

YashanDB提供如下四种加密算法供选择，不指定时默认采用AES128加密算法：

- AES128
- AES192
- AES256
- SM4

IDENTIFIED BY

指定加密备份时必须输入的密钥，该密钥输入的约束规则同用户口令规则，参考CREATE USER中password描述。

示例（单机、共享集群部署）

```
BACKUP ARCHIVELOG ALL ENCRYPTION IDENTIFIED BY 12345;
```

format

该语句用于指定生成的备份集的文件名称，省略则默认为`$YASDB_DATA/backup/bak_{date}`。

当文件名称中未指定路径时，默认创建在数据库的$YASDB_DATA/backup文件夹下。

共享集群部署中，通过指定为YFS路径，可将数据备份至共享存储上；指定为普通磁盘路径时，则将数据备份至实例所在服务器上。

tag

该语句用于指定备份集的别名，该别名最大长度为64（包含结束符'\0'）。

parallelism

该语句用于指定多线程备份的并行度，该值范围`[1, 8]`，省略时默认为2。

section size

该语句用于指定文件分片大小，超过该值的文件会被拆分为多个小文件执行备份，取值范围为[128M,32T]，省略时默认为系统自动计算的最优值。若指定默认文件大小不是1M整数倍，则会向下取整按照1M对齐。

BACKUP DATABASE

通用描述

BACKUP DATABASE用于执行对当前数据库的备份，即将数据库文件拷贝一个副本，以备份集的形式持久化。

备份包括全量备份和增量备份。

- 全量备份：对某一时间点上的所有数据进行完全复制，不依赖之前的备份集。一个全量备份集可以恢复出所有数据。
- 增量备份：首次执行基线备份（level0），之后每次只需备份增量数据（level1），备份效率高，节省磁盘空间。恢复时需要从基线备份集开始依次恢复所有增量备份。

在`DBA_BACKUP_SET`视图中查询生成的备份集信息。

在`V$BACKUP_PROGRESS`视图中查询备份的过程信息。

只能在数据库状态为OPEN时备份数据库，且归档模式必须为开启状态（使用ALTER DATABASE语句调整归档模式）。

关于备份恢复的详细操作描述请参考备份恢复。

分布式部署中，用户可以执行此语句，但只能对所在节点进行备份，因此不建议使用。如需对分布式集群进行备份应使用yasrman工具，具体见yasrman章节描述。

语句定义

backupCommonSpecifier::=

```
backupCommonSpecifier::= ((COMPRESSION [ALGORITHM (ZSTD|LZ4) [LOW|MEDIUM|HIGH]])|
(ENCRYPTION [AES128|AES192|AES256|SM4] IDENTIFIED BY password)|
(FORMAT backup_path)|(TAG tag_name)|
PARALLELISM integer|
SECTION SIZE size_clause)
{" " ((COMPRESSION [ALGORITHM (ZSTD|LZ4) [LOW|MEDIUM|HIGH]])|
(ENCRYPTION [AES128|AES192|AES256|SM4] IDENTIFIED BY password)|
(FORMAT backup_path)|(TAG tag_name)|
PARALLELISM integer|
SECTION SIZE size_clause)}
```

backup database::=

```
syntax::= BACKUP DATABASE (((FULL|(INCREMENTAL LEVEL integer [CUMULATIVE|INDEPEND|BASE ON tag_name])) [backupCommonSpecifier]
[FORCE])
|(DELETE BACKUPSET [IF EXISTS] (TAG tag_name|PATH backup_path)) | CANCEL)
```

full

该语句用于指定备份方式为数据库全量备份，全量备份是默认的备份方式。

示例（单机、共享集群部署）

```
BACKUP DATABASE FULL;

--通过视图查看备份信息
SELECT RECID#,TYPE,INCREMENT_LEVEL,INCREMENT_ID#,PATH,TAG,TRUNC_LSN
FROM DBA_BACKUP_SET
ORDER BY START_TIME;
  RECID# TYPE    INCREMENT_LEVEL INCREMENT_ID# PATH                                                                TAG
TRUNC_LSN
-------- ------- --------------- ------------- ------------------------------------------------------------------------ -------------
----------
       1 FULL                  0             0 /data/yashan/yasdb_data/db-1-1/backup/bak_2023121319584275
bak_2023121319584275         187034
```

incremental level

该语句用于指定备份方式为增量备份。

LEVEL后只能指定0或者1，且必须指定，其中0即等同于全量备份，1表示增量备份。第一次进行增量备份时，必须指定LEVEL 0。

independ

仅支持单机部署。

该语句在指定备份集为增量备份时，指定INDEPEND属性可生成独立增量备份链路的备份集，在DBA_BACKUP_SET视图中DEFAULT_BASE字段显示为FALSE，且该字段仅可在增量备份指定为LEVEL 0时可用。

base on tag_name

仅支持单机部署。

该语句与INDEPEND字段使用目的一致，但BASE ON字段仅在增量备份指定为LEVEL 1时可用，且tag_name指定备份集必须为独立链路的备份集，即在DBA_BACKUP_SET视图中DEFAULT_BASE字段显示为FALSE的备份集。

cumulative

对于LEVEL 1增量备份，显式指定CUMULATIVE时表示增量备份方式为累积增量备份，否则为普通增量备份。

- 普通增量备份：普通增量备份的基线（LSN）是上一次增量备份，可以是level 0的，也可以是level 1的。从此种方式的增量备份集进行数据库恢复时，除此增量备份集外还需要至少一个其他的增量备份集。
- 累积增量备份：累积增量备份的基线（LSN）是最近一次level 0的增量备份。从此种方式的增量备份集进行数据库恢复时，除此增量备份集外只需要一个其他的增量备份集。

> **Note**：
> 当备份指定CUMULATIVE后，不可同时指定备份属性为INDEPEND或者使用BASE ON语法。

示例（单机、共享集群部署）

```
BACKUP DATABASE INCREMENTAL LEVEL 0;
SELECT RECID#,TYPE,INCREMENT_LEVEL,INCREMENT_ID#,PATH,TAG,TRUNC_LSN
FROM DBA_BACKUP_SET
ORDER BY START_TIME;
  RECID# TYPE          INCREMENT_LEVEL INCREMENT_ID# PATH                                                                TAG
TRUNC_LSN
-------- ------------- --------------- ------------- ------------------------------------------------------------------------ -------
---------------- ----------
       1 FULL                        0             0 /data/yashan/yasdb_data/db-1-1/backup/bak_2023121319584275
bak_2023121319584275       187034
       2 INCREMENTAL                 0             0 /data/yashan/yasdb_data/db-1-1/backup/bak_2023121320010894
bak_2023121320010894       187038

BACKUP DATABASE INCREMENTAL LEVEL 1 CUMULATIVE;
SELECT RECID#,TYPE,INCREMENT_LEVEL,INCREMENT_ID#,PATH,TAG,TRUNC_LSN
FROM DBA_BACKUP_SET
ORDER BY START_TIME;
  RECID# TYPE          INCREMENT_LEVEL INCREMENT_ID# PATH                                                                TAG
TRUNC_LSN
-------- ------------- --------------- ------------- ------------------------------------------------------------------------ -------
---------------- ----------
       1 FULL                        0             0 /data/yashan/yasdb_data/db-1-1/backup/bak_2023121319584275
bak_2023121319584275       187034
       2 INCREMENTAL                 0             0 /data/yashan/yasdb_data/db-1-1/backup/bak_2023121320010894
bak_2023121320010894       187038
       3 INCREMENTAL                 1             1 /data/yashan/yasdb_data/db-1-1/backup/bak_2023121320030305
bak_2023121320030305       187042
```

compression

该语句用于压缩备份集，COMPRESSION后的选项都省略时，表示按ZSTD算法和LOW级别压缩备份数据。

增量备份的每个备份集可采用不同的压缩算法及级别，不影响对其的恢复。

压缩算法

通过ALGORITHM关键字指定，YashanDB提供如下两种压缩算法供用户选择：

- ZSTD
- LZ4
- 其中：ZSTD算法提供更高的压缩率，LZ4算法提供更高的压缩速率。

压缩级别

在指定压缩算法时可同时指定压缩级别，可省略，则缺省采用LOW级别。

压缩级别包含HIGH、MEDIUM、LOW三个选项，其中HIGH表示最高的压缩率（最低的压缩速度），LOW表示最低的压缩率（最高的压缩速度）。

示例（单机、共享集群部署）

```
BACKUP DATABASE INCREMENTAL LEVEL 1 CUMULATIVE COMPRESSION;
BACKUP DATABASE INCREMENTAL LEVEL 1 CUMULATIVE COMPRESSION ALGORITHM lz4 HIGH;
```

encryption

该语句用于加密备份集。

增量备份的每个备份集需要保证统一的都加密或者都不加密，且每个备份集的密钥必须保持一致。

加密算法

YashanDB提供如下四种加密算法供选择，不指定时默认采用AES128加密算法：

- AES128
- AES192
- AES256
- SM4

IDENTIFIED BY

指定加密备份时必须输入的密钥，该密钥输入的约束规则同用户口令规则，参考CREATE USER中password描述。

示例（单机、共享集群部署）

```
BACKUP DATABASE INCREMENTAL LEVEL 0 ENCRYPTION IDENTIFIED BY 12345;

--增量备份必须都加密
BACKUP DATABASE INCREMENTAL LEVEL 1;
YAS-00607 encryption algorithm error, the baseline backup set is encrypted

--增量备份的密钥必须相同
BACKUP DATABASE INCREMENTAL LEVEL 1 ENCRYPTION SM4 IDENTIFIED BY 123;
YAS-04261 invalid password: the backup password entered is inconsistent with the baseline backup password

--增量备份的加密算法可以不同
BACKUP DATABASE INCREMENTAL LEVEL 1 ENCRYPTION SM4 IDENTIFIED BY 12345;
```

format

该语句用于指定生成的备份集的文件名称，省略则默认为`$YASDB_DATA/backup/bak_{date}`。

当文件名称中未指定路径时，默认创建在数据库的$YASDB_DATA/backup文件夹下。

共享集群部署中，通过指定为YFS路径，可将数据备份至共享存储上；指定为普通磁盘路径时，则将数据备份至实例所在服务器上。

tag

该语句用于指定备份集的别名，该别名最大长度为64（包含结束符'\0'）。

parallelism

该语句用于指定多线程备份的并行度，该值范围`[1, 8]`，省略时默认为2。

section size

该语句用于指定文件分片大小，超过该值的文件会被拆分为多个小文件执行备份，范围`[128M, 32T]`，省略时默认为系统自动计算的最优值。若指定默认文件大小不是1M整数倍，则会向下取整按照1M对齐。

示例（单机、共享集群部署）

```
BACKUP DATABASE FULL FORMAT 'backup' TAG 'backup' PARALLELISM 8 SECTION SIZE 128M;
```

force

该语句用于指定是否强制备份。

强制备份被应用于对只读数据库的备份，例如主备形态中的备库备份，强制备份不会记录系统表，也不需要备库连接到主库。

只能在执行全量备份时指定FORCE。

示例（单机、共享集群部署）

```
BACKUP DATABASE FULL FORMAT 'backup1' TAG 'backup1' PARALLELISM 8 SECTION SIZE 128M FORCE;
```

cancel

该语句用于取消当前的backup，restore或build任务。

示例（单机、共享集群部署）

```
BACKUP DATABASE CANCEL;
```

delete backupset

该语句用于删除备份集，以及DBA_BACKUP_SET里的备份集记录。

本语句只能在单机部署中执行。

示例（单机、共享集群部署）

```
BACKUP DATABASE DELETE BACKUPSET TAG 'backup';
BACKUP DATABASE DELETE BACKUPSET PATH '/data/yashan/yasdb_data/db-1-1/backup/bak_2023121319584275';
```

BUILD DATABASE

通用描述

BUILD DATABASE用于在主备高可用部署环境中，在备库上创建standby数据库，有关主备高可用架构的详细操作描述请参考高可用。

在存在级联备的环境中，级联备库的上级备库对应为下文中的主库，级联备库对应为下文中的备库，具有与主备库相同的语法、功能及约束。

主库端BUILD命令仅SYS超级用户或拥有SYSDBA、SYSBACKUP权限的用户才可执行。

该功能不适用于分布式部署。

对备库成功执行本语句的前提是：

- 主库数据库已OPEN，并处于归档模式。
- 备库已安装数据库软件，且启动实例到NOMOUNT状态（增量BUILD中的备库须为OPEN状态）。
- 主备库上网络及权限畅通，相关配置参数值已正确设置。
- 满足CREATE DATABASE语句的所有约束条件。

本语句涵盖了YashanDB需要进行BUILD操作的各种场景：

- **全量BUILD**：用于新建或重建备库的初始化
  - 由备库分别对自身发起BUILD（BUILD DATABASE）：适用于单机、共享集群部署
  - 由主库向备库发起BUILD（BUILD DATABASE TO STANDBY指定单个standby_name或BUILD DATABASE TO REMOTE指定单个IP）：适用于单机、共享集群部署
  - 由主库向多备库同时发起BUILD（BUILD DATABASE TO STANDBY或BUILD DATABASE TO REMOTE）：适用于单机部署
  - 由备库向多备库同时发起BUILD（BUILD DATABASE TO REMOTE）：适用于单机部署
- **增量BUILD**：用于异常备库的快速修复
  - 由主库向存在gap的备库发起BUILD（BUILD DATABASE INCREMENTAL TO STANDBY或BUILD DATABASE INCREMENTAL TO REMOTE）：适用于单机部署
  - 由备库向存在gap的备库发起BUILD（BUILD DATABASE INCREMENTAL TO REMOTE）：适用于单机部署
  - 由主库向need repair的备库发起BUILD（BUILD DATABSE REPAIR STANDBY）：适用于单机部署

本语句执行过程中，可以通过V$BACKUP_PROGRESS视图查询某个BUILD操作的进度。若BUILD中断或失败，需注意：

- 由于执行本语句时，系统会将主库（或者某个备库）的控制文件、数据文件、redo日志文件和归档日志文件在线传输至备库的对应目录，过程中若出现错误将报错并中断操作，已传输的文件不会自动清除，再次运行本语句前，需在备库上手动清除脏数据。
- 对于同时发起的多备库并行BUILD，当某一个备库BUILD操作失败时，所有的备库BUILD操作都将失败。

语句定义

build database::=

```
syntax::= BUILD DATABASE [((INCREMENTAL TO ((STANDBY standby_name) | (REMOTE standby_address))) | (REPAIR STANDBY
standby_name) | (TO ((STANDBY "(" (((standby_name) {"," (standby_name)})| "*")")") | (REMOTE "(" (standby_address) {","
(standby_address)}")"))))) [DISCONNECT FROM SESSION]]
```

build database

该语句后面无任何选项时，表示本操作在备库上执行，将主库服务器文件在线传输到备库服务器。

示例（单机、共享集群部署）

```
--在某个备库上执行
BUILD DATABASE;
```

build database to standby

本操作一般在主库上执行，用于指定多个不同备库，将主库服务器文件同时在线传输到这些备库服务器。

在共享集群部署中执行本操作时，只能指定1个备集群。

standby_name

指定备库的DB_UNIQUE_NAME（通过V$ARCHIVE_DEST视图获得该值），在一个HA集群中，DB_UNIQUE_NAME唯一标识了一台备库服务器。

多个standby_name之间以 `,` 分隔，且不可以重复。

*****

指定与该主库连接的所有的备库。

示例（单机部署）

```
-- 指定对应的备库DB_UNIQUE_NAME执行BUILD。
BUILD DATABASE TO STANDBY (standby1_name, standby2_name);

-- 指定 * 符号即为该主库的所有备库执行BUILD。
BUILD DATABASE TO STANDBY (*);
```

示例（共享集群部署）

```
-- 指定对应的备集群DB_UNIQUE_NAME执行BUILD。
BUILD DATABASE TO STANDBY (standby1_name);
```

build database to remote

本操作可以在主库或备库上执行，用于指定多个不同IP，将本机服务器文件同时在线传输到这些服务器。

当某个备库出现异常时，为避免影响主库的业务，可以在一个正常同步备库上执行本语句，通过指定IP地址在线传输服务器文件到异常备库进行修复。

在共享集群部署中执行本操作时，只能指定1个备集群。

standby_address

指定备库的IP地址，需为`address:port`格式的一个有效地址，例如`127.0.0.1:1689`。

多个standby_name之间以`,`指定，且不可以重复。

YashanDB支持指定一个未与主库建立复制链路的IP地址，此时主库仍可以连接该备库且执行BUILD操作（由于未建立链路，该备库只是创建standby库，而不会加入HA集群），但需遵循该主库所处HA集群环境的一切限制，例如指定的IP个数不能超过备库最大限制数量。

示例（单机部署）

```
BUILD DATABASE TO REMOTE ('127.0.0.1:2901', '127.0.0.1:2902');
```

示例（共享集群部署）

```
BUILD DATABASE TO REMOTE ('127.0.0.1:2901');
```

build database incremental to standby

当备库与主库间存在较大gap时，在主库上通过standby_name指定备库执行本语句可以快速修复异常备库，缩小主备之间的差距。

本操作每次只能指定一个备库，且指定的备库必须是一个正常open状态的备库（通过V$REPLICATION_STATUS视图查看备库的状态）。

> **Caution：**
> - 本操作会自动重启备库，清理控制文件，且BUILD过程中备库处于非open状态，谨慎使用。
> - 如操作过程中途失败，只能对备库进行全量BUILD修复。

示例（单机部署）

```
BUILD DATABASE INCREMENTAL TO STANDBY standby1_name;
```

build database incremental to remote

当备库与主库间存在较大gap时，在主库上通过standby_address指定备库执行本语句可以快速修复异常备库，缩小主备之间的差距。

为了避免影响主库的业务，也可以在某个正常的同步备库或者主备差距较小的备库上执行本语句。

本操作每次只能指定一个IP地址（通过standby_address指定），且指定的备库必须是一个正常open状态的备库（通过V$REPLICATION_STATUS视图查看备库的状态）。

> **Caution：**
> - 本操作会自动重启备库，清理控制文件，且BUILD过程中备库处于非open状态，谨慎使用。
> - 如操作过程中途失败，只能对备库进行全量BUILD修复。

示例（单机部署）

```
BUILD DATABASE INCREMENTAL TO REMOTE '127.0.0.1:1689';
```

build databse repair standby

当某个备库为need repair状态（通过V$REPLICATION_STATUS视图查看备库的状态）时，在主库上通过standby_name指定备库执行本语句可以快速修复异常备库。

本操作每次只能指定一个备库，且指定的备库必须是某些异常类型（通过V$REPLICATION_STATUS视图查看备库的异常类型）的need repair状态，例如主库归档被清理等，其他的异常只能通过全量BUILD修复，更详细的描述请查看高可用手册备库异常修复。

示例（单机部署）

```
BUILD DATABASE REPAIR STANDBY standby1_name;
```

disconnect from session

本选项表示不需要等待BUILD操作完成之后再返回succeed，此时BUILD操作由后台线程完成，用户的会话线程不会阻塞。不指定本选项表示用户会话等待BUILD操作完成之后才返回succeed。

示例（单机部署）

```
BUILD DATABASE TO STANDBY (standby1_name, standby2_name) DISCONNECT FROM SESSION;
```

示例（共享集群部署）

```
--在主集群上执行
BUILD DATABASE TO STANDBY (standby1_name) DISCONNECT FROM SESSION;
```

COMMENT

通用描述

COMMENT用于为表|视图或表|视图的列字段添加注释。可以在USER_TAB_COMMENTS/USER_COL_COMMENTS等系统视图中查询添加的注释信息。

不能对已创建的COMMENT进行删除/修改操作，每执行一次COMMENT会覆盖上一次的注释信息，如想删除注释信息，请将COMMENT的内容指定为''。

语句定义

comment::=

```
syntax::= COMMENT ON (TABLE [schema "."] (table|view)|COLUMN [schema "."] (table|view) "." column) IS string
```

table table|view

该语句用于指定要添加注释信息的表|视图名称。

示例

```
COMMENT ON TABLE area IS '区域表';
SELECT * FROM USER_TAB_COMMENTS WHERE table_name='AREA';
TABLE_NAME        TABLE_TYPE COMMENTS
----------------- ---------- -------------
AREA              TABLE      区域表

--清除注释
COMMENT ON TABLE area IS '';
SELECT * FROM USER_TAB_COMMENTS WHERE table_name='AREA';
TABLE_NAME        TABLE_TYPE COMMENTS
----------------- ---------- -------------
AREA              TABLE
```

column table|view

该语句用于指定要添加注释信息的列字段的名称。

示例

```
COMMENT ON COLUMN area.DHQ IS '区域总部';
SELECT * FROM USER_COL_COMMENTS WHERE table_name='AREA' AND column_name='DHQ';
TABLE_NAME        COLUMN_NAME       COMMENTS
----------------- ----------------- --------------
AREA              DHQ               区域总部

--清除注释
COMMENT ON COLUMN area.DHQ IS '';
SELECT * FROM USER_COL_COMMENTS WHERE table_name='AREA' AND column_name='DHQ';
TABLE_NAME        COLUMN_NAME       COMMENTS
----------------- ----------------- --------------
AREA              DHQ
```

COMMIT

通用描述

COMMIT用于提交一个事务。

单机部署中，COMMIT等同于COMMIT WORK WRITE WAIT IMMEDIATE。

分布式部署中，COMMIT等同于COMMIT WORK WRITE WAIT，且除此之外的其他选项均不适用于分布式。

执行COMMIT操作前，用户在事务过程做的任何修改只有自己能看到，其他用户无法看到，并可以通过回滚（Rollback）恢复修改之前的数据。

执行COMMIT操作时，系统对用户所做的修改进行数据持久化（redo日志写入磁盘，触发磁盘I/O），同时清除事务所在会话所有的SAVEPOINT，释放本事务持有的所有锁，本次事务结束。

执行COMMIT操作后，所有用户都能看到修改后的数据，且不能通过回滚恢复修改之前的数据。

系统在如下时间点自动调用COMMIT语句：

- DDL操作开始前。
- DDL操作成功结束后。
- 会话正常退出登录时。

语句定义

commit::=

```
syntax::= COMMIT [WORK] (
  ([WRITE [IMMEDIATE|BATCH] [WAIT|NOWAIT]]))
```

work

该语句用于和标准SQL的语法兼容，无实际意义。

write

该语句用于指定redo日志写入磁盘的方式，默认为WRITE WAIT IMMEDIATE。

immediate|batch

指定redo日志何时写入磁盘，IMMEDIATE表示立即写入磁盘，BATCH表示将系统中需要执行WRITE的redo数据累积到一定数量后再一起写入磁盘。

通过V$REDOSTAT视图的BATCH_COMMIT_DELAY字段可查看当前系统在BATCH模式下redo数据累积刷盘所延迟的时间。

wait|nowait

指定是否进行写操作等待，WAIT表示直到redo日志成功写入磁盘后，COMMIT才会进行成功返回，NOWAIT表示COMMIT直接进行成功返回，此时不保证redo日志成功写入磁盘。

> **Note**：
> 当配置参数COMMIT_WAIT被设置为WAIT_FORCE时，即使指定了NOWAIT，系统仍强制执行等待。

示例

```
COMMIT WRITE IMMEDIATE NOWAIT;
```

CREATE ACCESS CONSTRAINT

通用描述

CREATE ACCESS CONSTRAINT用于在指定的表上创建一个AC对象。只能基于LSC表执行本语句来创建一个AC对象，且该LSC表不可以为组合分区表或INTERVAL类型的分区表。

创建AC后，系统将同时在后台生成AC数据，AC数据可被用于：

- 执行源表查询时，在满足AC条件的情况下，系统将使用AC进行有界计算，加速查询。
- 用户可以像查询表一样直接查询AC，查询结果为当前后台已生成的AC数据。

当源表为一个分区表时，所创建的AC也将分区，且其分区界值、数量、分区表空间、存储属性等与源表一致。

语句定义

create access constraint::=

```
syntax::= CREATE ACCESS CONSTRAINT [schema "."] ac_name FROM [schema "."] table_name ac_model_clause
```

ac_model_clause**::=**

```
syntax::= ON xy_clause [n_clause] [where_clause] [aggr_clause][order_clause][ac_attr_clause]
```

xy_clause**::=**

```
syntax::= ( column_expr {( "," column_expr)} TO | ONLY ) column_expr  {( "," column_expr)}
```

column_expr**::=**

```
syntax::= (column_name|column_oper_expr) [[AS] alias]
```

column_oper_expr**::=**

```
syntax::= (column_name|literal) ("+"|"-"|"*"|"/"|"%") (column_name|literal)
```

n_clause**::=**

```
syntax::= BOUND literal
```

where_clause**::=**

```
syntax::= WHERE filter_clause [(AND|OR) filter_clause]
```

filter_clause**::=**

```
syntax::= ((column_name|literal) ("<"|"<="|">"|">="|"="|"!=") (column_name|literal))
| (column_name IS [NOT] NULL)
| (column_name IN "(" (literal) {"," (literal)} ")")
```

aggr_clause**::=**

```
syntax::= INCLUDE aggr_func [[AS] alias] { "," aggr_func [[AS] alias]}
```

order_clause**::=**

```
syntax::= [NO] ORDER
```

ac_attr_clause**::=**

```
syntax::= TABLESPACE (tablespace_name|DEFAULT)
```

ac_name

该语句用于指定创建的AC的名称，不可省略，且需符合YashanDB的对象命名规范。

table_name

该语句用于指定要创建AC的源表。创建AC的操作过程中会对该表加排他锁。

ac_model_clause

用于建立AC数据模型的语句。

xy_clause

该语句用于指定AC里的x值和y值，可通过TO关键字建立x to y的模型，也可以通过ONLY关键字建立只包含y值的模型。相同的x值和y值组合上只能创建一次AC对象，除非该对象被删除。

x/y值均由一个或多个基于列字段的表达式组成，以`,`分隔。

column_expr

该语句用于指定x/y值基于列字段的表达式，表达式可以为：

- 要创建AC的表的列字段。
- 列字段与常数，或者列字段与列字段的四则运算式。

表达式中必须至少包含一个列字段，但所有包含在表达式中的列字段均不能为LOB大对象型。

可以为每个表达式指定一个别名，若未指定，则使用默认命名。若别名与其他别名或者默认别名重复则报错。

示例（LSC表）

```
CREATE ACCESS CONSTRAINT ac_area FROM area
ON area_no TO area_name,dhq;
```

n_clause

该语句用于指定AC的边界大小，BOUND为系统预留关键字，无实际含义，可省略，则默认为int64最大值。

如果指定本语句，需满足如下约束规则：

- BOUND的值必须为正整数，上限为int64最大值，即9223372036854775807。
- 只可以指定为一个常数。

where_clause

该语句用于设置AC数据的过滤条件，可省略，则表示获取所有数据。

filter_clause

可用于AC模型数据过滤的条件包括：

- =，!=，>，>=，<，<=：此类条件操作数必须为列字段与列字段，或者列字段与常数。
- IS NULL，IS NOT NULL：必须对列字段进行此类条件判断。
- IN：须对列字段进行此类条件判断，且IN后的集合只能包含常数。

示例（LSC表）

```
DROP ACCESS CONSTRAINT ac_area;
CREATE ACCESS CONSTRAINT ac_area FROM area
ON area_name TO area_no,dhq
WHERE dhq IN ('Shanghai','Chengdu')
AND area_name IS NOT NULL;
```

aggr_clause

该语句用于指定AC列的聚集信息，可省略，则表示不对列进行聚集。

可同时指定多个聚集信息，以`,`分隔，但每个聚集信息只能针对一个列字段，可执行聚集的函数包括：

- SUM
- MAX
- MIN
- COUNT

上述聚集函数不可以进行嵌套，且不可以在单个AC中对相同列进行相同的聚集操作。

示例（LSC表）

```
DROP ACCESS CONSTRAINT ac_area;
CREATE ACCESS CONSTRAINT ac_area FROM area
ON area_name TO area_no,dhq
WHERE dhq IN ('Shanghai','Chengdu')
AND area_name IS NOT NULL
INCLUDE COUNT(dhq),MAX(area_no);
```

order_clause

该语句用于指定AC的数据是否排序，省略则默认为ORDER。

ac_attr_clause

该语句用于指定AC的表空间，省略则默认为源表的表空间。分布式部署中用户无法为在分布表上创建的AC指定表空间，使用表所在的表空间作为AC的表空间。

- tablespace_name：指定到一个存在的表空间。
- DEFAULT：指定到缺省表空间，即源表的表空间。

在LSC表上创建的AC对象，其表空间必须指定为一个databucket表空间。

示例（单机LSC表）

```
DROP ACCESS CONSTRAINT ac_area;
CREATE ACCESS CONSTRAINT ac_area FROM area
ON area_name TO area_no,dhq
NO ORDER
TABLESPACE DEFAULT;
```

CREATE AUDIT POLICY

通用描述

CREATE AUDIT POLICY用于创建一个审计策略，通过指定不同的审计项对数据库操作进行审计。

只有拥有AUDIT_ADMIN审计管理员角色，或者拥有AUDIT SYSTEM系统权限的用户，才能创建一个审计策略。

一个成功创建的审计策略，还必须满足如下两个前提条件才会触发系统开始审计操作：

- 配置参数UNIFIED_AUDITING值为true，即审计开关为打开状态。
- 通过AUDIT POLICY语句将审计策略使能。

审计策略被成功创建后，可通过AUDIT_UNIFIED_POLICIES视图查看其定义信息。

YashanDB对用户提供如下方面的审计管理：

- 系统权限审计
- 系统行为和对象行为的操作审计
- 角色审计

对于审计管理更具体的描述请查看安全手册审计。

语句定义

create audit policy::=

```
syntax::= CREATE AUDIT POLICY policy_name [privilege_audit_clause] [action_audit_clause] [role_audit_clause] [when_clause]
[toplevel_clause]
```

privilege_audit_clause**::=**

```
syntax::= PRIVILEGES (system_privileges_clause) {"," (system_privileges_clause)}
```

action_audit_clause**::=**

```
syntax::= ACTIONS (system_action_clause|object_action_clause) {"," (system_action_clause|object_action_clause)}
```

system_action_clause**::=**

```
syntax::= system_action|ALL
```

object_action_clause**::=**

```
syntax::= (object_action|ALL) ON [schema "."] object_name
```

role_audit_clause**::=**

```
syntax::= ROLES (role_name) {"," (role_name)}
```

when_clause**::=**

```
syntax::= WHEN "'" audit_condition "'" EVALUATE PER (STATEMENT|SESSION|INSTANCE)
```

toplevel_clause**::=**

```
syntax::= ONLY TOPLEVEL
```

policy_name

审计策略的名称，审计策略名称不可重复，不可省略，不可带模式名，且需符合YashanDB的对象命名规范。

privilege_audit_clause

该语句用于定义审计策略包含的系统权限的审计项，可同时定义多个系统权限审计项，用`,`分隔。

system_privileges_clause

指定具体的系统权限名称，具体项目可在安全手册系统权限管理中查看。

对指定的系统权限启用审计策略后，所有使用到该权限的操作将被审计。

示例

```
-- 创建审计策略，定义对涉及select any table, delete any table权限的语句进行审计
CREATE AUDIT POLICY up1
PRIVILEGES SELECT ANY TABLE, DELETE ANY TABLE;
```

action_audit_clause

该语句用于定义审计策略包含的系统行为或对象行为审计项，可同时定义多个行为审计项，用`,`分隔，其中审计项可分为如下两种类型：

- 全局的审计项。
- 针对具体对象的审计项。

system_action_clause

指定全局审计项。

system_action
全局审计项目，具体项目可以通过AUDITABLE_SYSTEM_ACTIONS视图查看。

all
当全局审计项指定为ALL时，表示在AUDITABLE_SYSTEM_ACTIONS中列出的所有项目都将纳入审计策略。

object_action_clause

指定针对具体对象的审计项。

object_action
针对对象的审计项目，具体项目可以通过AUDITABLE_OBJECT_ACTIONS视图查看。

schema
模式名称，可省略，则使用当前登录用户的模式。

object_name
对象名称，YashanDB支持对表、视图等所有对象进行操作审计。

示例

```
-- 创建审计策略，定义针对drop table类型语句、对表area的delete、insert、update操作、在branches所有操作进行审计
CREATE AUDIT POLICY up2
  ACTIONS DROP TABLE,
    DELETE ON sales.area,
    INSERT ON sales.area,
    UPDATE ON sales.area,
    ALL ON sales.branches;
```

role_audit_clause

该语句用于定义审计策略包含的角色的审计项，可同时定义多个角色审计项，用`,`分隔。

role_name

角色名称，只能为NORMAL ROLE，具体项目可在DBA_ROLES视图中查看。

对指定的角色启用审计策略后，被直接赋予到该角色的所有系统权限都将会被审计，即所有使用到这些权限的操作都将被审计。

示例

```
-- 创建角色
CREATE ROLE role_a;
-- 给角色赋予系统权限
GRANT CREATE SESSION TO role_a;
--创建角色审计策略，使能后该策略生效，使能范围内的会话连接将被审计
CREATE AUDIT POLICY audit_role_a ROLES role_a;
```

when_clause

该语句用于定义是否执行审计策略的判断条件，若条件结果为true则执行审计策略，否则不执行。

audit_condition

条件表达式，为简单的条件判断，条件结果值是true或者false。具体可为如下内容：

- 数学函数：BITAND, CEIL, FLOOR, POWER
- 字符函数：CONCAT, LOWER, UPPER
- 长度计算函数：INSTR, LENGTH
- 环境信息相关函数：SYS_CONTEXT
- 比较运算符：=, !=, <>, <, >, <=, >=
- 逻辑运算符：AND, OR
- [NOT] BETWEEN
- [NOT] IN

在审计条件表达式中使用SYS_CONTEXT不适用于分布式部署。

evaluate per (statement|session|instance)

表示条件判断执行频率，分别对应语句级、SESSION级、实例级（一个实例运行周期里只进行一次条件判断）。

示例（单机、共享集群部署）

```
-- 创建审计策略，针对表area的select查询，在满足给定的条件的情况下进行审计
CREATE AUDIT POLICY up_SYS_CONTEXT
  ACTIONS SELECT ON sales.area
  WHEN 'SYS_CONTEXT(''USERENV'', ''OS_USER'') = ''SALES'''
  EVALUATE PER SESSION;
```

示例

```
-- 创建审计策略，针对表area的select查询，在满足给定的条件的情况下进行审计
CREATE AUDIT POLICY up3
  ACTIONS SELECT ON sales.area
  WHEN '2>1'
  EVALUATE PER SESSION;
```

toplevel_clause

该语句用于定义是否对匿名块、存储过程、函数等过程体内部的SQL语句进行审计。指定ONLY TOPLEVEL则不对内部语句进行审计。

示例

```
-- 创建审计策略，针对执行时涉及SELECT ANY TABLE权限的语句进行审计但不针对匿名块、存储过程、函数的内部语句进行审计。
CREATE AUDIT POLICY up4
  PRIVILEGES SELECT ANY TABLE
  ONLY TOPLEVEL;
```

CREATE DATABASE LINK

通用描述

CREATE DATABASE LINK语句用于创建一个数据库链接对象。本地当前用户通过数据库链接可以访问远程数据库中某用户的物理表数据。

本语句不适用于分布式部署。

根据数据库系统的结构，支持两种数据库链接：

- 同构数据库链接：YashanDB与YashanDB的数据库链接
- 异构数据库链接：YashanDB与Oracle的数据库链接，需要进行异构数据库链接配置

语句定义

create database link**::=**

```
syntax::= CREATE [PUBLIC] DATABASE LINK dblink_name
[connect_clause] [USING connect_string]
```

connect_clause**::=**

```
syntax::= CONNECT TO username IDENTIFIED BY pwd_clause
```

pwd_clause**::=**

```
syntax::= plaintext_password | VALUES ['"'] ciphertext_password ['"']
```

connect_string**::=**

```
syntax::= [database_type ":"] url ["/" db_name]
```

database_type**::=**

```
syntax::= [YASHAN | ORACLE | MYSQL | ODBC]
```

public

该语句建立一个公有数据库链接，对所有数据库用户可见。

创建公有数据库链接的用户须拥有CREATE PUBLIC DATABASE LINK系统权限，创建非公有数据库链接用户须拥有CREATE DATABASE LINK系统权限，否则返回错误。

示例（单机部署、共享集群部署）

```
CREATE PUBLIC DATABASE LINK dblink_yashan;
```

dblink_name

该语句用于指定创建的数据库链接的名称，不可省略，且需符合YashanDB的对象命名规范。

username

该语句用于指定访问远端数据库的用户名。

plaintext_password

该语句用于指定访问远端数据库的用户明文密码。

ciphertext_password

该语句用于指定访问远端数据库的用户密文密码。

url

该语句用于指定连接远程数据库使用的IP信息，例如192.168.1.2:1688。

db_name

该语句用于指定远程数据库实例名称。

database_type

该语句用于指定远程数据库系统或标准接口标签名，支持识别Yashan和Oracle。不指定标签名时默认为Yashan。

示例（单机部署、共享集群部署）

```
-- YashanDB与YashanDB的数据库链接
CREATE DATABASE LINK dblink_yashan CONNECT TO REGRESS IDENTIFIED BY REGRESS USING '192.168.1.2:1688';

-- YashanDB与Oracle的数据库链接
CREATE DATABASE LINK dblink_oracle CONNECT TO C##REGRESS IDENTIFIED BY REGRESS USING 'oracle:192.168.1.2:1521/orcl';
```

CREATE DATABASE

通用描述

CREATE DATABASE用于创建一个新的数据库，该语句只能在一个处于NOMOUNT状态的空实例（Instance）中执行。系统通过本语句成功创建新数据库后，同时将其启动至OPEN状态。

- 单机部署中，使用CREATE DATABASE语句创建一个新的单实例数据库，数据库文件路径仅支持指定为本地文件系统路径。
- 分布式部署中，通过yasboot安装过程中即在所有的分布式节点上依据配置文件中指定参数创建数据库，后续不再允许在某个节点上执行本语句（除非用于故障恢复），否则将导致分布式集群出现节点不一致情况。
- 共享集群部署中，必须使用CREATE CLUSTER DATABASE创建一个新的多实例数据库，且redo、undo、swap等所有数据文件须指定为YFS路径，否则返回错误。

创建数据库的过程中会在指定或默认位置创建物理文件，如过程中出现任何异常导致创建数据库失败时，这些已创建的文件将残留，用户需进行手动清理，未清理且使用相同文件再次CREATE DATABASE时将失败。

语句定义

create database::=

```
syntax::= create_single_instance_database|create_multiple_instance_database
```

create_single_instance_database**::=**

```
syntax::= CREATE DATABASE database_name [(attribute_clause) {" " (attribute_clause)}]
```

attribute_clause**::=**

```
syntax::=  (USER SYS IDENTIFIED BY password|
CONTROLFILES "("  ((controlfile) {", " (controlfile)}) ")"|
MAXDATAFILES integer|
MAXDATABUCKETS integer|
MAXINSTANCES integer|
CHARACTER SET charset|
NATIONAL CHARACTER SET charset|
UNDO_SEGMENTS integer|
MAXLOGFILES  integer|
MAXLOGHISTORY integer|
(ARCHIVELOG|NOARCHIVELOG)|
DOUBLE_WRITE DATAFILE  dbwfile SIZE size_clause |
logfiles_clause|
tablespace_clause)
{" " (USER SYS IDENTIFIED BY password|
CONTROLFILES "("  ((controlfile) {", " (controlfile)}) ")"|
MAXDATAFILES integer|
MAXDATABUCKETS integer|
MAXINSTANCES integer|
CHARACTER SET charset|
NATIONAL CHARACTER SET charset|
UNDO_SEGMENTS integer|
MAXLOGFILES  integer|
MAXLOGHISTORY integer|
(ARCHIVELOG|NOARCHIVELOG)|
DOUBLE_WRITE DATAFILE  dbwfile SIZE size_clause |
logfiles_clause|
tablespace_clause)}
```

logfiles_clause**::=**

```
syntax::= "(" (logfile SIZE size_clause [BLOCKSIZE size_clause]){"," (logfile SIZE size_clause [BLOCKSIZE size_clause])} ")"
```

tablespace_clause::=

```
syntax::= ((SYSTEM|SYSAUX|UNDO|SWAP|TEMPORARY|DEFAULT) TABLESPACE (TEMPFILE|DATAFILE) datafiles_clause [extent_clause]
[MEMORY MAPPED] [databucket_clause])
{" " ((SYSTEM|SYSAUX|UNDO|SWAP|TEMPORARY|DEFAULT) TABLESPACE (TEMPFILE|DATAFILE) datafiles_clause [extent_clause] [MEMORY
MAPPED] [databucket_clause])}
```

datafiles_clause::=

```
syntax::= (datafile SIZE size_clause [AUTOEXTEND (ON|OFF)] [NEXT size_clause] [MAXSIZE (UNLIMITED|size_clause)])
{"," (datafile SIZE size_clause [AUTOEXTEND (ON|OFF)] [NEXT size_clause] [MAXSIZE (UNLIMITED|size_clause)])}
```

extent_clause::=

```
syntax::= EXTENT (AUTOALLOCATE|UNIFORM SIZE size_clause)
```

databucket_clause::=

```
syntax::= DATABUCKET (bucketurl) {"," (bucketurl)}
```

create_multiple_instance_database::=

```
syntax::= CREATE CLUSTER DATABASE database_name (INSTANCES integer | ([(attribute_clause) {" " (attribute_clause)}]
(instance_clause) {" " (instance_clause)}) )
```

instance_clause::=

```
syntax::= INSTANCE "(" (logfiles_clause|UNDO TABLESPACE DATAFILE datafiles_clause)")"
```

create_single_instance_database

该语句用于创建一个单实例的数据库。

database_name

指定要创建的数据库的名称，不可省略，且需符合YashanDB的对象命名规范。

attribute_clause

该语句用来指定创建数据库的配置。

user sys identified by

该语句用于指定SYS用户的口令，可省略，则数据库创建后SYS用户的口令仍为初始值，并可通过ALTER USER语句并更口令。

controlfiles

该语句用于指定创建数据库时，同时创建的控制文件，多个控制文件用`,`分隔，可省略，则系统根据CONTROL_FILES参数定义的默认值在磁盘上生成控制文件。YashanDB支持指定最多8个控制文件。

maxdatafiles

该语句用于指定数据库能打开的数据文件的最大个数，该值决定了控制文件里Datafile Section所占的初始空间，可省略，则取系统默认值4096。YashanDB支持指定的最大值为16384。

maxdatabuckets

该语句用于指定数据库能挂载的databucket的最大个数，可省略，则取系统默认值256。YashanDB支持指定的最大值为4096。

maxinstances

该语句用于指定能同时挂载（Mount）该数据库的最大实例个数，可省略，则取系统默认值8。YashanDB支持指定的最大值为64。

character set
该语句用于指定数据库存储数据到文件所使用的字符集，可省略，则取系统默认值UTF8。YashanDB支持指定的字符集有ASCII、GBK、UTF8、ISO8859-1、GB18030。

如需使用TAC表和LSC表，数据库的字符集必须设置为UTF-8。

national character set
该语句用于指定数据库使用的国家字符集，可省略，则取系统默认值UTF16。YashanDB支持指定的国家字符集仅有UTF16。

undo_segments
该语句用于指定数据库可自动扩展回滚段的最大个数，可省略，则为默认值64。YashanDB支持指定的最大值为256。

maxlogfiles
该语句用于指定redo日志文件的元数据预留个数，该值决定了控制文件里Redofile Section所占的初始空间，可省略，则取系统默认值256，YashanDB支持指定的最大值为4096。该值不限制实例可创建的redo日志文件个数，当创建的redo日志个数超过该值时，控制文件可能会扩展。

maxloghistory
该语句用于指定归档日志文件的元数据预留个数，该值决定了控制文件里Archfile Section所占的初始空间，可省略，则取系统默认值64000。YashanDB支持指定的最大值为1000000。该值不限制实例可创建的归档日志文件个数，当创建的归档日志个数超过该值时，控制文件可能会扩展。

archivelog|noarchivelog
该语句用于指定数据库是否开启归档模式，可省略，则取系统默认值NOARCHIVELOG。

double_write datafile dbwfile size size_clause
该语句用于指定创建数据库时，同时创建的双写文件的路径和大小，可省略，则在系统默认的数据文件路径下创建一个名称为dwf，大小为8192个BLOCK的文件。

当指定创建双写文件时文件过小会出现建库失败，双写文件的可用最小值受配置参数DBWR_BUFFER_SIZE和DBWR_COUNT影响，最小值近似公式：4 * DBWR_BUFFER_SIZE * DBWR_COUNT + 4M。

logfile logfiles_clause
该语句用于指定创建数据库时，同时创建的redo日志文件的路径和大小，可省略，则在系统默认的数据文件路径下创建三个名称为redo1/redo2/redo3，大小为128M，BLOCKSIZE为4K的文件。

logfiles_clause
logfile：日志文件的名称，未包含路径时，取系统默认的数据文件路径。

SIZE：日志文件的大小。

BLOCKSIZE：日志文件的页面大小。默认是4K，最小512字节，最大32K。其值一般设置为操作系统BLOCK的大小，小于系统BLOCK大小的话，会有性能影响。一般不建议修改。

tablespace_clause
该语句用于指定创建数据库时，同时创建SYSTEM/SYSAUX/UNDO/SWAP/TEMPORARY表空间，可省略，则系统按下面各项给出的缺省值创建这些表空间。

tempfile|datafile
表空间对应的数据文件类型，其中SWAP/TEMPORARY表空间的数据文件类型缺省为TEMPFILE，其他表空间缺省为DATAFILE。

datafiles_clause
表空间对应的数据文件名称，多个数据文件用`,`分隔。缺省在系统默认的数据文件路径下创建名称与表空间名称相同（DEFAULT表空间对应的数据文件名默认为users），大小为8192个BLOCK，开启自动扩展，每次扩展8192个BLOCK，最大可扩展到64MB个BLOCK（UNDO表空间为8MB个BLOCK）的数据文件。

extent_clause
表空间对象申请extent时的空间分配方式，AUTOALLOCATE分配方式根据对象当前大小由系统自动分配extent空间，UNIFORM分配方式每次为对象分配的extent空间为固定值。 UNDO/SWAP表空间不允许指定extent分配方式，只能使用UNIFORM分配方式，默认为UNIFORM SIZE 1个BLOCK。 TEMP表空间若不指定分配方式，默认为UNIFORM SIZE 8个BLOCK，其它表空间默认为AUTOALLOCATE（系统自动分配）。

memory mapped
指定是否启用数据常驻内存模式，默认不启用。

databucket_clause
当为表空间指定了DATABUCKET时，系统将按指定路径信息创建用于对象存储的目录。对于DEFAULT表空间，如未指定DATABUCKET，系统将在默认路径（local_fs）下创建users文件夹用于对象存储。

示例（单机部署）

```
CREATE DATABASE yashan
CHARACTER SET utf8
NATIONAL CHARACTER SET utf16
ARCHIVELOG
LOGFILE ('/home/yasdb/YASDB_DATA/dbfiles/redo1' size 2G BLOCKSIZE 512,
'/home/yasdb/YASDB_DATA/dbfiles/redo2' size 2G BLOCKSIZE 512,
'/home/yasdb/YASDB_DATA/dbfiles/redo3' size 2G BLOCKSIZE 512,
'/home/yasdb/YASDB_DATA/dbfiles/redo4' size 2G BLOCKSIZE 512)
UNDO TABLESPACE DATAFILE '?/dbfiles/undo' size 2G autoextend ON NEXT 256M MAXSIZE 64G
SWAP TABLESPACE TEMPFILE '?/dbfiles/swap' size 2G autoextend ON NEXT 256M MAXSIZE 64G
SYSTEM TABLESPACE DATAFILE 'system' size 128M MEMORY MAPPED
DEFAULT TABLESPACE DATAFILE '?/dbfiles/users' size 5G;
```

create_multiple_instance_database

该语句用于在共享集群部署下创建多实例数据库。

database_name

指定要创建的数据库的名称，不可省略，且需符合YashanDB的对象命名规范。

instances interger

该语句用于指定共享集群部署下多实例数据库的实例个数，最大值为4，可省略。若省略，则必须指定instance_clause。

attribute_clause

指定除redo文件和undo文件之外的数据库配置，这些属性定义与创建单实例数据库时的属性定义一致，见create_single_instance_database中的attribute_clause。

> **Note**：
> 共享集群部署中redo、undo、swap、default等所有数据文件路径需指定为YFS路径，否则返回错误。

instance_clause

该语句用于指定共享集群部署下多实例数据库每个实例相关undo、redo日志文件和数据文件配置，根据所需实例个数多次指定本语句，最多支持指定4次（即创建4个实例），语句之间使用空格间隔。

示例（共享集群部署）

```
--通过INSTANCES关键字创建2个实例的数据库，数据库文件均为默认属性
CREATE cluster DATABASE yashan instances 2;
--通过instance_clause语句创建2个实例的数据库，且自定义数据库文件属性
CREATE cluster DATABASE yashan
SYSTEM TABLESPACE DATAFILE 'system' size 64M
SYSAUX TABLESPACE DATAFILE 'sysaux' size 64M
DEFAULT TABLESPACE DATAFILE 'users' size 64M
TEMPORARY TABLESPACE TEMPFILE 'temp' size 64M
SWAP TABLESPACE TEMPFILE 'swap' size 64M
instance(
LOGFILE('logfile11' size 32M, 'logfile12' size 32M, 'logfile13' size 32M)
UNDO TABLESPACE DATAFILE 'undo1' size 64M
)
```

```
instance (
LOGFILE('logfile21' size 32M, 'logfile22' size 32M, 'logfile23' size 32M)
UNDO TABLESPACE DATAFILE 'undo2' size 64M
);
--指定路径的配置
CREATE cluster DATABASE yashan
instance (
LOGFILE('+DG_0/redo1' size 128M BLOCKSIZE 512,
'+DG_0/redo2' size 128M BLOCKSIZE 512)
UNDO TABLESPACE DATAFILE '+DG_0/undo1' size 128M autoextend ON)
SWAP TABLESPACE TEMPFILE '+DG_0/swap' size 128M autoextend ON
SYSAUX TABLESPACE DATAFILE '+DG_0/sysaux' size 128M autoextend ON
SYSTEM TABLESPACE DATAFILE '+DG_0/system' size 128M autoextend ON
TEMPORARY TABLESPACE TEMPFILE '+DG_0/temp' size 128M autoextend ON
DEFAULT TABLESPACE DATAFILE '+DG_0/users' size 128M autoextend ON;
```

CREATE DIRECTORY

通用描述

CREATE DIRECTORY用于新建数据目录对象。具有create any directory权限的所有用户均可执行本语句创建数据目录对象，但目录对象始终为sys用户所有。

该语句仅适用于单机部署。

语句定义

create directory::=

```
syntax::= CREATE [OR REPLACE] DIRECTORY directory_name AS "path_name"
```

directory_name

该语句用于指定创建的目录的名称，不可省略，且需符合YashanDB的对象命名规范。

path_name

该语句用于指定创建的目录的地址。

该语句需满足如下规则：

- 长度限制为4000。
- 其中不可以包含父目录(..)。
- 创建目录时不检查目录是否真实存在。

示例（单机部署）

```
CREATE DIRECTORY dir AS '/data/yashan';
```

CREATE FUNCTION

通用描述

CREATE FUNCTION用于创建一个自定义函数（UDF）。

自定义函数为PL对象，对其的创建需采用过程体语言实现，且需满足YashanDB所实现PL（包括变量、参数、语句等）的一切规范和约束，具体描述请查看PL参考手册。

关于CREATE FUNCTION的语法描述详见PL参考手册的自定义函数。

CREATE INDEX

通用描述

CREATE INDEX用于在指定的表上创建一个索引对象。在YashanDB中可创建多种类型的索引，例如：

- 唯一索引
- 列式索引
- 函数索引
- 反向索引
- 分区索引
- RTREE索引

语句定义

create index::=

```
syntax::= CREATE [UNIQUE|COLUMNAR|RTREE] INDEX [schema"."] index_name ON table_name "(" index_expr [DESC|ASC]{ "," index_expr
[DESC|ASC]} ")" [index_attr_clause]
```

index_expr**::=**

```
syntax::= column_name|column_expression
```

index_attr_clause**::=**

```
syntax::= (TABLESPACE (tablespace_name|DEFAULT)|INITRANS integer|PCTFREE integer|storage_clause|(VISIBLE|INVISIBLE)|
(USABLE|UNUSABLE)|local_index_clause|GLOBAL|ONLINE|(NOPARALLEL|PARALLEL [integer])|(NOCOMPRESS|COMPRESS [integer])|
(LOGGING|NOLOGGING)|(NOREVERSE|REVERSE)|readonly_clause|inmemory_clause)
{" " (TABLESPACE (tablespace_name|DEFAULT)|INITRANS integer|PCTFREE integer|storage_clause|(VISIBLE|INVISIBLE)|
(USABLE|UNUSABLE)|local_index_clause|GLOBAL|ONLINE|(NOPARALLEL|PARALLEL [integer])|(NOCOMPRESS|COMPRESS [integer])|
(LOGGING|NOLOGGING)|(NOREVERSE|REVERSE)|readonly_clause|inmemory_clause)}
```

storage_clause定义

local_index_clause**::=**

```
syntax::= LOCAL [store_in_clause|index_partition_clause|index_comp_partition_clause]
```

readonly_clause**::=**

```
syntax::= READONLY | READWRITE
```

inmemory_clause**::=**

```
syntax::= INMEMORY | NO INMEMORY
```

store_in_clause**::=**

```
syntax::= STORE IN "(" ((tablespace) {"," (tablespace)}) ")"
```

index_partition_clause**::=**

```
syntax::= "(" ((PARTITION [partition] [index_partition_attr_clause])
{"," (PARTITION [partition] [index_partition_attr_clause])}) ")"
```

index_partition_attr_clause::=

```
syntax::= (TABLESPACE ( tablespace_name|DEFAULT)|INITRANS integer|PCTFREE integer|USABLE|UNUSABLE)
{" " (TABLESPACE ( tablespace_name|DEFAULT)|INITRANS integer|PCTFREE integer|USABLE|UNUSABLE)}
```

index_comp_partition_clause::=

```
syntax::=[store_in_clause] "(" PARTITION [partition] [index_partition_attr_clause] [USABL] [index_subpartition_clause] {","
PARTITION [partition] [index_partition_attr_clause] [index_subpartition_clause]} ")"
```

index_subpartition_clause::=

```
syntax::= STORE IN "(" (tablespace) {"," (tablespace)} ")" | "(" (SUBPARTITION [subpartition] [TABLESPACE tablespace]
[USABLE|UNUSABLE]) {"," (SUBPARTITION [subpartition] [TABLESPACE tablespace] [USABLE|UNUSABLE])} ")"
```

unique

该语句用于指定创建的索引为唯一索引，即索引列字段上的数值唯一。如表中该列字段上的值存在重复，则创建唯一索引失败。

LSC表必须指定该关键字。

分布式部署中，分区列必须是唯一索引关键列的子集，否则报错。

示例

```
--在表orders_info中的id,area列创建唯一索引idx_orders_info_1
CREATE UNIQUE INDEX idx_orders_info_1 ON orders_info (id, area);
```

columnar

该语句用于指定创建的索引为列式索引，即索引的数据内容采用列存方式存储，索引本身无序。

列式索引存在如下限制：

- 当指定为列式索引时，不能同时指定为函数索引或反向索引，且不能在线（ONLINE）创建和重建（rebuild_clause）。
- 只能在单机HEAP表中创建列式索引。

示例（单机HEAP表）

```
--在表orders_info中的product_no列创建列式索引colidx_orders_info_1
CREATE COLUMNAR INDEX colidx_orders_info_1 ON orders_info (product_no);
```

rtree

该语句用于指定创建RTREE索引。

RTREE索引存在如下限制：

- 当指定为RTREE索引时，不能同时指定为函数索引或反向索引，且不能在线（ONLINE）创建和重建（rebuild_clause）。
- 每次执行该语句时，所指定的列中仅允许存在1个ST_Geometry类型（其他类型不受此限制）。如需在同一张表中为多个ST_Geometry类型列创建RTREE索引，需重复执行该语句并指定不同列。
- RTREE索引仅适用于单机部署HEAP表。
- 不能为临时表创建RTREE索引。

使用RTREE索引时，不能将事务隔离级别设为SERIALIZABLE。

示例（单机HEAP表）

```
--在表geom_test中的shp列创建RTREE索引idx_rtree_shp，其中shp列的数据类型为ST_Geometry
CREATE TABLE geom_test(id INT, geom_type VARCHAR(32), shp st_geometry);
```

```
CREATE RTREE INDEX idx_rtree_shp ON geom_test (shp);
```

index_name

该语句用于指定创建的索引的名称，不可省略，且需符合YashanDB的对象命名规范。

table_name

该语句用于指定要创建索引的表。创建索引的操作过程中会对该表加排他锁，但在指定了ONLINE时，本操作不会阻塞对该表的并发DML事务。

index_expr

指定索引所基于的列或列表达式，多项组合指定时以`,`分隔。

所有的索引列或列表达式里不能包含LOB大对象型的列字段。

column_name

将指定名称的列字段作为索引列。

column_expression

将指定的表达式作为索引列，含有表达式的索引列的索引称为函数索引。

YashanDB支持以任何通用表达式作为索引列来建立函数索引，但存在如下约束规则：

- column_expression计算出来的结果须满足系统对索引列的所有约束限制，例如不能为LOB型，长度不能超过索引键值长度规格等。
- column_expression中包含内置函数时，该函数必须是一个可以返回确定和不变结果的非聚集函数，例如不能为SYSDATE、USERENV、SUM等函数。
- column_expression中存在自动填充行为时，不能为非固定自动填充行为的表达式，例如在日期转换时发生自动填充年月的行为，由于系统需使用当前年月这类非固定值填充，导致表达式输出的结果不能确定，所以无法建立函数索引；而对于时分秒系统固定按0填充，或者对于CHAR类型系统固定按空格补位，表达式输出结果确定，则可以建立函数索引。
- 在创建分区索引时，不能对分区键使用函数表达式。
- column_expression中不能包含用户自定义函数。
- 当指定为RTREE索引时，不能同时指定为函数索引。
- 不能为LSC表创建函数索引。
- 当指定为函数索引时，不能指定NOPARALLEL|PARALLEL语句。

示例（HEAP表，单机TAC表）

```
--函数索引可以为任意表达式
CREATE UNIQUE INDEX idx_orders_info_2 ON orders_info (TO_CHAR(id)||order_no);
CREATE INDEX idx_orders_info_3 ON orders_info (CAST(id AS CHAR(14)),order_no);

--字符串向DATE转换时，未指定format将导致系统获取不能保证恒定的默认格式，所以无法返回一个不变的结果
CREATE INDEX idx_orders_info_4 ON orders_info (order_date-TO_DATE('2001-01-01'));
[1:69]YAS-04399 only pure functions can be indexed
```

desc|asc

指定索引列的同时指定排序方式，可省略，则系统默认按升序排序。

示例（HEAP表、TAC表）

```
--指定索引列的升降序
CREATE INDEX idx_orders_info_4 ON orders_info (order_date DESC, TO_NUMBER(order_no) ASC);
```

index_attr_clause

该语句用于指定索引的各项属性。

tablespace

该语句用于指定索引的表空间，省略则默认为当前用户所在的表空间。用户无法为建立在分布表上的索引和索引分区指定表空间，这类索引和索引分区所在的表空间将自动和表保持一致。

- tablespace_name：指定到一个存在的表空间。
- DEFAULT：指定到缺省表空间，即当前用户所在的表空间。

在临时表上创建的索引，其表空间只能为系统默认的temporary表空间，不可指定。

initrans/pctfree/storage_clause

该语句用于指定索引的存储属性，省略则INITRANS/PCTFREE默认为2/8，其他存储属性请参考通用SQL语法storage描述。

visible|invisible

该语句用于指定创建的索引是否能被优化器（Optimizer）使用，省略则默认为VISIBLE。

usable|unusable

该语句用于指定创建的索引是否可用，省略则默认为USABLE。

global

该语句用于显式指定创建的索引是全局索引。若不指定该关键字，YashanDB也将默认创建全局索引。分布表指定GLOBAL关键字将会报错。

online

该语句用于指定创建索引过程中是否允许并发DML操作，省略则默认不允许。

在线（Online）创建索引存在如下限制：

- 只允许在单机部署中在线创建索引。
- 不允许为LSC表在线创建索引。
- 不允许在线创建列式索引和RTREE索引。

noparallel|parallel

该语句用于设置创建索引的并行度，NOPARALLEL表示不并行。

LSC表不允许并行创建索引，即integer不得大于1。

integer

并行度值，取值范围[1,服务器CPU核数*2]，可省略，省略则默认按CPU核数的并行度并发创建索引。

示例（单机HEAP表、TAC表）

```
--在线并行创建索引
DROP INDEX idx_orders_info_1;
CREATE INDEX idx_orders_info_1 ON orders_info (id) ONLINE PARALLEL 2;
```

nocompress|compress

该语句用于语法兼容，无实际含义。

logging|nologging

该语句用于指定创建索引的logging属性，可省略，省略则默认为logging。

只能在nologging表上创建nologging索引。

示例（HEAP表、TAC表）

```
CREATE TABLE logging_table(c1 INT);

CREATE INDEX logging_index ON logging_table(c1);

-- 只能在nologging表上创建nologging索引
```

```
CREATE INDEX nologging_index ON logging_table(c1) NOLOGGING;
YAS-02367 create nologging index is not allowed for logging table
```

noreverse|reverse

该语句用于指定创建的索引是否为反向索引。反向索引在存入数据时将索引列字节进行反转，以离散数据，减少叶块争用。省略时默认为NOREVERSE。

不能为LSC表创建反向索引。

示例（HEAP表、TAC表）

```
DROP INDEX idx_orders_info_1;
CREATE INDEX idx_orders_info_1 ON orders_info (id) REVERSE;

DROP INDEX idx_orders_info_1;
CREATE INDEX idx_orders_info_1 ON orders_info (id) NOREVERSE;
```

readonly_clause

语法兼容，无实际含义。

inmemory_clause

语法兼容，无实际含义。

local_index_clause

该语句用于创建分区索引（Local Partitioned Index）。只能在分区表上建立分区索引，且索引的分区数量、分区界值必须与表的分区相同。

创建唯一索引，要保证一级分区键和二级分区键的并集需要是索引键的子集。

如LOCAL后面的语句都省略，即按如下规则创建索引分区：

- 分区数量与表分区数量一致
- 分区界值与表分区界值一致
- 表空间与在INDEX上指定的值一致
- INITRANS/PCTFREE等存储属性与在INDEX上指定的值一致
- USABLE/UNUSABLE开关设置为USABLE

示例（HEAP表、单机TAC表）

```
--创建本地分区索引
CREATE INDEX idx_sales_info_2 ON sales_info (year,month,branch)
TABLESPACE yashan
INITRANS 3
UNUSABLE
LOCAL;
```

store_in_partition_clause

该语句仅被用于指定在哈希分区表上的索引分区的表空间，指定多个表分区用`,`分隔，且数量需与索引分区的数量一致。

示例（HEAP表、单机TAC表）

```
--指定hash分区索引的分区表空间
CREATE INDEX idx_sales_info_hash_1 ON sales_info_hash (year,month,branch)
TABLESPACE yashan
INITRANS 3
UNUSABLE
LOCAL STORE IN (yashan,yashan1);
```

index_partition_clause

指定索引分区的名称和其他属性，同时指定多个分区用`,`分隔，且数量需与索引分区的数量一致。

partition_name
指定索引分区的名称，省略则由系统生成的默认名称。

index_partition_attr_clause
指定索引分区的如下属性：

- 表空间：省略则默认为在INDEX上指定的表空间。其中DEFAULT值等于在INDEX上指定的表空间名称。
- INITRANS/PCTFREE：省略则默认为在INDEX上指定的存储属性值。
- USABLE/UNUSABLE：省略则为USABLE。

示例（HEAP表、单机TAC表）

```
--指定索引分区属性
CREATE INDEX idx_sales_info_range_1 ON sales_info_range (year,month,branch)
TABLESPACE yashan
INITRANS 3
UNUSABLE
LOCAL (PARTITION p1 TABLESPACE DEFAULT INITRANS 2,PARTITION p2,PARTITION p3);
```

index_comp_partition_clause
指定索引组合分区的名称和其他属性，同时指定多个分区用`,`分隔，且数量需与表分区的数量一致。

store in
指定二级分区模板信息，只适用于hash分区。

index_subpartition_clause
指定索引二级分区的名称和其他属性，同时指定多个二级分区用`,`分隔，且数量需与表的对应二级分区的数量一致。

示例（分布式部署）

```
--创建默认名称的二级分区索引
CREATE UNIQUE INDEX idx_orders_info_sub ON orders_info(area, id) LOCAL;
```

示例（单机/共享集群部署、HEAP/TAC表）

```
--创建指定名称的二级分区索引
CREATE INDEX idx_sales_info_sub ON sales_info(product) LOCAL
(PARTITION ip_sales_info_1 UNUSABLE(SUBPARTITION isp_sales_info_11, SUBPARTITION isp_sales_info_12, SUBPARTITION
isp_sales_info_13),
PARTITION ip_sales_info_2 USABLE (SUBPARTITION isp_sales_info_21, SUBPARTITION isp_sales_info_22, SUBPARTITION
isp_sales_info_23),
PARTITION ip_sales_info_3 USABLE (SUBPARTITION isp_sales_info_31, SUBPARTITION isp_sales_info_32, SUBPARTITION
isp_sales_info_33));
```

CREATE LIBRARY

通用描述

CREATE LIBRARY用于创建一个自定义库。

关于CREATE LIBRARY的语法详细描述见PL参考手册的自定义库。

CREATE MATERIALIZED VIEW

通用描述

CREATE MATERIALIZED VIEW语句用于创建一个物化视图，该物化视图包含了子查询的结果。区别于普通视图，物化视图会占用物理存储空间。

物化视图只能基于单机HEAP表创建。

语句定义

create materialized view::=

```
syntax::= CREATE MATERIALIZED VIEW [SCHEMA "."] materialized_view_name
["("((column_name) {"," (column_name)})")"]
[TABLESPACE tablespace_name]
[BUILD (IMMEDIATE | DEFERRED)]
[create_mv_refresh_clause]
[query_rewrite_clause]
AS subquery
```

create_mv_refresh_clause::=

```
syntax::= REFRESH ([COMPLETE | FORCE]
| [ON DEMAND | ON COMMIT]
| [((START WITH date) | NEXT date)]){" "([COMPLETE | FORCE]
| [ON DEMAND | ON COMMIT]
| [((START WITH date) | NEXT date)])}
| (NEVER REFRESH)
```

query_rewrite_clause

```
syntax::= (ENABLE | DISABLE) QUERY REWRITE
```

materialized_name

该语句用于指定创建的物化视图名称，不可省略，且需符合YashanDB的对象命名规范。

如物化视图名称里出现了`.`符号，表示该符号之前为用户名、之后为物化视图名，此时必须拥有在指定用户名下创建该对象的权限。

column_name

该语句用于指定物化视图列字段的别名，可以省略，不指定数据类型，且需符合YashanDB的对象命名规范。

tablespace

该语句用于指定所创建的物化视图对应的持久化数据存储在哪个表空间上。

示例（单机HEAP表）

```
CREATE MATERIALIZED VIEW mv1 TABLESPACE USERS AS SELECT * FROM area;
```

build (immediate|deferred)

该语句用于指定物化视图是否在创建时同步刷新数据，可省略，省略则默认为BUILD IMMEDIATE。

IMMEDIATE表示物化视图创建时刷新数据，该选项会导致物化视图创建过程耗费时间较长。

DEFERRED表示物化视图创建时仅创建基础定义，不刷新数据，后续依据其他手段进行刷新。

示例（单机HEAP表）

```
CREATE MATERIALIZED VIEW mv2 BUILD IMMEDIATE AS SELECT * FROM area;
```

create_mv_refresh_clause

该语句用于指定物化视图刷新相关的属性，包括刷新类型、刷新模式、定时刷新等，多项间使用空格进行分隔。

complete|force

该语句用于指定物化视图的刷新类型，包括：

- COMPLETE：全量刷新
- FORCE：若条件满足，优先使用快速刷新；若快速刷新不可用，则使用全量刷新

刷新类型最多能够指定一项，省略则默认为FORCE。

on demand|on commit

该语句用于指定物化视图的刷新模式，包括：

- ON DEMAND: 手动刷新，即DBMS_MVIEW高级包刷新
- ON COMMIT: 自动刷新，事务提交时刷新相关联的物化视图

刷新模式最多能够指定一项，省略则默认为ON DEMAND。

start with date|next date

该语句用于指定物化视图的定时刷新选项，包括：

- START WITH date：指定第一次刷新时间
- NEXT date：指定刷新时间间隔

定时刷新选项可以指定一项或指定两项，也可省略表示不进行定时刷新。

下一次刷新时间计算后必须是未来某个时间。

never refresh

该语句用于指定不刷新。

指定后保护物化视图不被任何自动刷新、高级包机制刷新，并忽略任何发出的刷新语句、高级包执行。

示例（单机HEAP表）

```
CREATE MATERIALIZED VIEW mv_refresh(mya,myb) BUILD IMMEDIATE
REFRESH
COMPLETE
ON DEMAND
NEXT SYSDATE + 10/(2460)
AS SELECT area_no,area_name FROM area;
```

query_rewrite_clause

该语句用于指定该物化视图是否可以用于查询重写功能，可省略，省略则默认不可用于查询重写。

可通过查询 `QUERY_REWRITE_ENABLED` 参数查看当前物化视图查询重写功能的具体情况。

查询重写功能包含如下限制：

- 物化视图不允许包含可变的内置函数（如时间相关函数）、UDF和UDP等。
- 物化视图不允许包含表函数。
- 物化视图不允许包含伪列。
- 当物化视图基于的表定义修改，导致物化视图失效，此时该物化视图不可进行查询重写。
  - 选中的基表列字段数据类型更改
  - 选中的基表列字段删除
  - 选中的基表列字段名称更改

(enable|diasble) query rewrite

具体含义如下：

- ENABLE QUERY REWRITE: 允许物化视图用于查询重写
- DISABLE QUERY REWRITE: 不允许物化视图用于查询重写

subquery

该语句用户指定创建物化视图的子查询语句，子查询规则同SELECT语句中描述。 其中，子查询的列项不能指定为某个序列号。

如创建时指定了物化视图的列字段，则子查询中的列字段数量需与物化视图列字段数量相同，否则返回错误。

如在子查询中包含了对某个表的`SELECT *`操作时：

- 创建的物化视图将该表的所有列字段作为自己的列字段。
- 如该表增加了新的列字段，则本物化视图不会体现该变化，仍只包含原来的列字段项。
- 如该表删除了某个列字段，则本物化视图对象失效，无法使用，直到该列字段被重新增加。
- 如该表更名了某个列字段，则本物化视图对象失效，无法使用，直到该列字段被改回原来的名称。

示例（单机HEAP表）

```
CREATE MATERIALIZED VIEW mv_subqu AS SELECT * FROM area;

CREATE MATERIALIZED VIEW mv_subquery(mya,myb) AS SELECT b.branch_name, a.area_name
FROM branches b, area a
WHERE a.area_no=b.area_no AND b.branch_no LIKE '01%';

--列字段数量不匹配时返回错误
CREATE MATERIALIZED VIEW mv_colmismatch(c1,c2,c3) AS SELECT area_no,area_name FROM area;
[1:26]YAS-04304 the count of column is mismatched
```

CREATE OUTLINE

通用描述

CREATE OUTLINE用于创建一个存储纲要，保存在指定SQL上的指定hint信息，当某个SQL语句存在OUTLINE信息时，优化器在非例外的情况下将会依据这个OUTLINE生成执行计划。这将使固定的SQL语句的执行计划变得稳定，不会由于运行环境、统计信息波动而导致优化器生成用户不期望的执行计划。

启用存储纲要的目的是为了对优化器的自动处理进行人工干预，详细信息可参考性能调优SQL调优原理与规则。需说明的是，在大多数情况下，用户应该更相信优化器所作的判断，采纳优化器自动生成的执行计划，除非能准确地预见到在某段时间OUTLINE是更好的选择，则可以手动地打开控制开关，使用OUTLINE。

用户必须拥有CREATE ANY OUTLINE权限才能创建一个存储纲要。

语句定义

create_outline::=

```
syntax::= CREATE [OR REPLACE] [PUBLIC] OUTLINE [outline_name] [FROM [PUBLIC] source_outline] [FOR CATEGORY category_name] [ON
statement]
```

or replace

替换已有的OUTLINE。

public

公有模式，默认值。

outline_name

将要创建的OUTLINE的名称，不能与已有OUTLINE名称重复，不能带模式名指定，且需符合YashanDB的对象命名规范。 若省略，系统将根据当前时间自动生成一个OUTLINE名称。

from source_outline

使用本语句表示从一个已有的OUTLINE进行复制创建。

对于此种方式创建的OUTLINE，必须为其指定一个与源OUTLINE不同的类别，且不能同时使用ON statement。

for category category_name

指定OUTLINE归属的类别，省略此语句时，系统将默认归属到DEFAULT类别。

category_name为类别的名称，可以为已存在的一个类别名，或者为新指定的一个类别名。category_name不能带模式名指定，且需符合YashanDB的对象命名规范。

对于同一个SQL语句，不能将其重复指定到同一个类别。

on statement

指定OUTLINE对应的SQL语句。SQL语句必须为如下类型中的一种且需遵循相应的语法要求：

- SELECT语句
- DELETE语句
- UPDATE语句
- INSERT...SELECT语句

对于违反语法要求的SQL语句，OUTLINE将创建失败且提示错误信息。

示例

```
-- 对给定的SQL创建OUTLINE。
CREATE OUTLINE ol_a FOR CATEGORY ctgy_ab ON
SELECT /*+ FULL(a) */ a.area_name
FROM area a
WHERE a.area_no LIKE '01';

-- 对给定的源outline 进行复制，且归属到默认的DEFAULT类别
CREATE OUTLINE ol_a1 FROM ol_a;
```

CREATE PACKAGE

通用描述

CREATE PACKAGE用于创建一个高级包（UDP）。高级包是多个PL对象的集合，依据一定业务逻辑将这些对象封装在一个高级包里，便于对内的管理和对外的接口简洁。

高级包也是一个可持久化的PL对象，由如下两部分组成：

- PACKAGE HEAD：通过CREATE PACKAGE语法创建，包含集合中有哪些PL对象的声明。
- PACKAGE BODY：通过CREATE PACKAGE BODY语法创建，包含在HEAD中声明的每个PL对象对应的过程体定义。

对高级包的创建需采用过程体语言实现，且需满足YashanDB所实现PL（包括变量、参数、语句等）的一切规范和约束，具体描述请查看PL参考手册。

关于CREATE PACKAGE的语法描述详见PL参考手册的自定义高级包。

CREATE PROCEDURE

通用描述

CREATE PROCEDURE用于创建一个存储过程。

存储过程为PL对象，对其的创建需采用过程体语言实现，且需满足YashanDB所实现PL（包括变量、参数、语句等）的一切规范和约束，具体描述请查看PL参考手册。

关于CREATE PROCEDURE的语法描述详见PL参考手册的存储过程。

CREATE PROFILE

通用描述

CREATE PROFILE用于创建一个profile，profile为系统对用户的一组资源限制的集合。在YashanDB中，profile主要用于定义密码策略相关限制（详见安全手册密码策略描述）。

执行本语句需注意如下事项：

- 用户必须拥有CREATE PROFILE权限才能创建一个profile。
- profile本身并不包含用户信息，需使用CREATE USER或ALTER USER语句关联user与profile。

通过DBA_PROFILES视图可查询系统中所有的profile，及其内容。

YashanDB已预置一个默认的profile（名称为DEFAULT，内容为安全手册密码策略中所描述资源限制），在使用缺省语法创建新用户时，该用户都将被关联到默认的profile，受其指定的资源限制。

分布式部署中用户无法执行本语句。

语句定义

create profile::=

```
syntax::= CREATE PROFILE profile_name LIMIT (resource_parameters | password_parameters) {" " (resource_parameters |
password_parameters)}
```

password_parameters::=

```
syntax::= (password_parameter_name password_parameter_value)
  {" " (password_parameter_name password_parameter_value)}
```

resource_parameters::=

```
syntax::= (resource_parameter_name resource_parameter_value)
  {" " (resource_parameter_name resource_parameter_value)}
```

profile_name

profile名称，不可省略，且需符合YashanDB的对象命名规范。

password_parameters

定义密码策略相关的资源限制，包括密码策略参数和参数值，同时定义多个参数以空格分隔。

当某个密码策略参数未在本语句中指定时，系统按DEFAULT值（见下文DEFAULT描述）将其生成到profile中。

password_parameter_name

为安全手册密码策略中所定义的密码策略参数，指定其他参数将报错。

password_parameter_value

指定密码策略参数所对应的资源限制值。

所有的参数均可设置为UNLIMITED或DEFAULT，除此之外的值依据不同参数存在不同规则。

resource_parameters

定义系统相关的资源限制，包括资源限制参数和参数值，同时定义多个参数以空格分隔。

当某个资源策略参数未在本语句中指定时，系统按DEFAULT值（见下文DEFAULT描述）将其生成到profile中。

resource_parameter_name

目前包括SESSIONS_PER_USER，IDLE_TIME。

sessions_per_user
用户最多并行会话连接数。

idle_time
用户最长空闲无操作时间，指用户登录后，连续长时间无操作达到阈值，系统将断开该会话，该会话被断开后，再执行操作将报错。 检测并断开长时间空闲会话由后台线程每个10s进行一次，因此用户最长连续空闲的时间至多可比该用户的对应阈值多10s。 该资源配置值单位为分钟。

resource_parameter_value
指定资源参数所对应的资源限制值。

所有的参数均可设置为UNLIMITED或DEFAULT，除此之外的值依据不同参数存在不同规则。

UNLIMITED

指定为UNLIMITED表示不对该参数对应资源进行限制。

DEFAULT

如果要省略profile中对应资源的限制，则可指定为DEFAULT，表示与该profile关联的用户将受到默认profile中对应资源的限制。

不同参数可设置的值

见下表：

策略参数	参数值规则
FAILED_LOGIN_ATTEMPTS	最大连续错误登录次数，取值需为整数，取值范围(0,2147483647)。
PASSWORD_LIFE_TIME	密码生命周期，单位为天，取值需为整数，取值范围(0,24855]。
PASSWORD_REUSE_TIME	密码不能被复用的时间，单位为天，取值需为整数，取值范围(0,24855]。
PASSWORD_REUSE_MAX	密码被复用需要经历的变更次数，取值需为整数，取值范围(0,2147483647)。
PASSWORD_LOCK_TIME	达到最大连续错误登录次数需要锁定对应用户的时间，单位为天，取值需为整数，取值范围(0,24855]。
PASSWORD_GRACE_TIME	密码过期宽限期，单位为天，取值需为整数，取值范围[0,24855]。
SESSIONS_PER_USER	用户最大并发会话数，取值需为整数，取值范围[1,2147483646]。
IDLE_TIME	用户最长连续空闲时间，单位为分钟，取值需为整数，取值范围[1,2147483646]。

> **Note**：
> - 上述参数值支持数据类型隐式转换和表达式运算，结果满足规则即可。
> - 输入空字符串，或者表达式结果为NULL时，视为0值处理。

示例（单机、共享集群部署）

```
CREATE PROFILE prof_1 LIMIT FAILED_LOGIN_ATTEMPTS 1 PASSWORD_LIFE_TIME 180;

CREATE PROFILE prof_2 LIMIT FAILED_LOGIN_ATTEMPTS DEFAULT PASSWORD_LIFE_TIME 30*2.5;

CREATE PROFILE prof_3 LIMIT FAILED_LOGIN_ATTEMPTS DEFAULT PASSWORD_LIFE_TIME UNLIMITED;

CREATE PROFILE prof_4 LIMIT
FAILED_LOGIN_ATTEMPTS 1
PASSWORD_LIFE_TIME 180
PASSWORD_REUSE_TIME UNLIMITED
PASSWORD_REUSE_MAX UNLIMITED
PASSWORD_LOCK_TIME 1
PASSWORD_GRACE_TIME 0;
```

CREATE ROLE

通用描述

CREATE ROLE用于创建一个角色。角色管理属于权限体系的一部分，YashanDB的角色管理体系请参考产品安全手册角色管理。

YashanDB包含如下两种类型的角色：

- 系统内置角色：如DBA角色和PUBLIC角色。
- 普通角色：通过本语句创建的普通角色。

语句定义

create role::=

```
syntax::= CREATE ROLE role
```

role

该语句用于指定要创建的角色的名称，不可省略，且需符合YashanDB的对象命名规范。

示例

```
CREATE ROLE rolename;
```

CREATE SEQUENCE

通用描述

CREATE SEQUENCE用于创建一个序列对象，包括升序序列号生成器（生成的序列号为正数）和降序序列号生成器（生成的序列号为负数）。

序列号生成器的数据类型只能为整数。

对于升序序列号生成器，生成的序列号不能小于1；对于降序序列号生成器，生成的序列号不能大于-1。

分布式部署中用户无法执行本语句。

语句定义

create sequence::=

```
syntax::= CREATE SEQUENCE [schema "."] sequence
[(((INCREMENT BY|START WITH) integer)|(MAXVALUE integer|NOMAXVALUE)|(MINVALUE integer|NOMINVALUE)|(CYCLE|NOCYCLE)|
(ORDER|NOORDER)|(CACHE integer|NOCACHE))
{" " (((INCREMENT BY|START WITH) integer)|(MAXVALUE integer|NOMAXVALUE)|(MINVALUE integer|NOMINVALUE)|(CYCLE|NOCYCLE)|
(ORDER|NOORDER)|(CACHE integer|NOCACHE))}]
```

sequence

该语句用于指定要创建的序列号生成器的名称，不可省略，且需符合YashanDB的对象命名规范。

如该语句后面的语句全部省略，则会生成一个从1开始，每次增长1，无最大数限制（但满足操作系统最大整数限制）的升序序列号生成器。

序列号分为CURRVAL和NEXTVAL两种类型，具体描述见伪列章节描述。

示例（单机、共享集群部署）

```
CREATE SEQUENCE seq_yashan1;

SELECT seq_yashan1.NEXTVAL FROM DUAL;
  SEQ_YASHAN1.NEXTVAL
----------------------
                    1
```

increment by

该语句用于指定序列号生成器的增量（Interval），即下一序列号值=当前序列号值+增量。该值不能为0，省略则默认为1。

本值为正数即表示创建的是一个升序序列号生成器，为负数则表示创建的是一个降序序列号生成器。

start with

该语句用于指定序列号生成器的起始值，该值需介于MINVALUE和MAXVALUE之间。对于升序序列号生成器，本值不能小于1；对于降序序列号生成器，本值不能大于-1。省略则默认为1（升序序列号生成器）或-1（降序序列号生成器）。

maxvalue|nomaxvalue

该语句用于指定序列号生成器的最大值，对于升序序列号生成器，本值不能小于1；对于降序序列号生成器，本值不能大于-1。

对于升序序列号生成器，NOMAXVALUE表示为操作系统的最大正整数数值；对于降序序列号生成器，NOMAXVALUE表示为-1。

本语句省略时默认为NOMAXVALUE。

minvalue|nominvalue

该语句用于指定序列号生成器的最小值，对于升序序列号生成器，本值不能小于1；对于降序序列号生成器，本值不能大于-1。

对于升序序列号生成器，NOMINVALUE表示为1；对于降序序列号生成器，NOMAXVALUE表示为操作系统的最大负整数数值。

本语句省略时默认为NOMINVALUE。

cycle|nocycle

该语句用于指定当生成的序列号到达MAXVALUE（升序序列号生成器）或MINVALUE（降序序列号生成器）后，是否开启序列号循环。

- CYCLE：对于升序序列号生成器，当要生成的序列号大于MAXVALUE时，将该序列号置为MINVALUE；对于降序序列号生成器，当要生成的序列号小于MINVALUE时，将该序列号置为MAXVALUE。
- NOCYCLE：当要生成的序列号大于MAXVALUE（升序序列号生成器）或小于MINVALUE（降序序列号生成器）时，不产生值并返回错误信息。

本语句省略时默认为NOCYCLE。

示例（单机、共享集群部署）

```
--创建一个最大值为20的升序序列号生成器
CREATE SEQUENCE seq_yashan3 INCREMENT BY 10 MAXVALUE 20 CYCLE NOCACHE;

--当要生成的序列号超过20时，重新从最小值1开始
SELECT seq_yashan3.NEXTVAL FROM DUAL;
  SEQ_YASHAN3.NEXTVAL
----------------------
                    1
SELECT seq_yashan3.NEXTVAL FROM DUAL;
  SEQ_YASHAN3.NEXTVAL
----------------------
                   11
SELECT seq_yashan3.NEXTVAL FROM DUAL;
  SEQ_YASHAN3.NEXTVAL
----------------------
                    1

--创建一个含自增列的表
CREATE TABLE area_seq(area_no INT DEFAULT seq_yashan3.NEXTVAL, area_name VARCHAR2(60));
```

order|noorder

ORDER/NOORDER用于指定是否保证序列号按请求的先后顺序生成。

在单机部署中，YashanDB已保证了序列号按照请求的先后顺序生成，ORDER/NOORDER对序列号的表现没有影响。

在共享集群部署中，如果指定了ORDER，序列号将在所有实例上保持有序；如果指定了NOORDER，且预分配序列号个数大于1，那么序列号将在每个实例内保持有序。

示例（共享集群部署）

在两实例共享集群部署中，实例1执行：

```
-- 创建一个预分配5个序列号的有序序列
CREATE SEQUENCE seq_yashan4_order CACHE 5 ORDER;

-- 创建一个预分配5个序列号的无序序列
CREATE SEQUENCE seq_yashan4_noorder CACHE 5 NOORDER;

-- 在实例1获取这两个序列的序列号
SELECT seq_yashan4_order.NEXTVAL FROM DUAL;
SEQ_YASHAN4_ORDER.NE
--------------------
                   1
SELECT seq_yashan4_noorder.NEXTVAL FROM DUAL;
SEQ_YASHAN4_NOORDER.
--------------------
                   1
```

实例2执行：

```
-- 在实例2获取这两个实例的序列号
SELECT seq_yashan4_order.NEXTVAL FROM DUAL;
SEQ_YASHAN4_ORDER.NE
--------------------
                   2
SELECT seq_yashan4_noorder.NEXTVAL FROM DUAL;
SEQ_YASHAN4_NOORDER.
--------------------
                   6
```

cache|nocache

该语句用于指定是否在内存中对序列号进行预分配，省略则默认为在内存中预分配20个序列号。

- NOCACHE：不在内存中进行序列号预分配。
- CACHE：指定预分配序列号的个数。当同时指定NOCYCLE时，本值可为任意一个非0正整数；当同时指定CYCLE时，本值的大小不能超过序列号生成器一个循环内可产生的序列号个数，即小于`(MAXVALUE-MINVALE)/ABS(INCREMENT BY) +1`。

示例（单机、共享集群部署）

```
 --由于((11-1)/10)+1=2,而cache值需小于此值，返回错误
CREATE SEQUENCE seq_yashan5 INCREMENT BY 10 MAXVALUE 11 CYCLE CACHE 2;
YAS-02096 sequence param CACHE error, number of CACHE must be less than one cycle
```

在共享集群部署中，预分配是实例级别的，各实例按先后顺序预分配一段序列号，序列号在实例内有序，在集群全局不保证有序。

如果指定为NOCACHE，则序列不进行预分配，序列号在集群全局上有序。

示例（共享集群部署）

- 预分配序列号场景：

在两实例共享集群部署中，实例1执行：

```
CREATE SEQUENCE seq_yashan6 CACHE 5;
-- 实例1预分配了序列号1、2、3、4、5
SELECT seq_yashan6.NEXTVAL FROM DUAL;
SEQ_YASHAN6.NEXTVAL
-------------------
                  1
```

实例2执行：

```
-- 实例2在实例1预分配之后，预分配了序列号6、7、8、9、10
SELECT seq_yashan6.NEXTVAL FROM DUAL;
SEQ_YASHAN6.NEXTVAL
-------------------
                  6
```

- 不进行预分配序列号场景：

在两实例共享集群部署中，实例1执行：

```
CREATE SEQUENCE seq_yashan7 NOCACHE;
-- 实例1获取序列号1
SELECT seq_yashan7.NEXTVAL FROM DUAL;
SEQ_YASHAN7.NEXTVAL
-------------------
                  1
```

实例2执行：

```
-- 实例2在实例1获取序列号之后，获取到2
SELECT seq_yashan7.NEXTVAL FROM DUAL;
SEQ_YASHAN7.NEXTVAL
-------------------
                  2
```

CREATE SQLMAP

通用描述

CREATE SQLMAP用于创建一个SQL映射，通过将A语句映射为B语句，实现数据库接收A语句请求而实际执行B语句的能力。此功能可作为SQL调优的辅助手段，在某些优化器没有选取到最优执行计划的特殊场景中，通过映射表将特定SQL替换为等价SQL，以使数据库按最优计划执行。

执行本语句需注意如下事项：

- 用户必须拥有DBA权限才能创建一个SQL映射。
- 要映射的SQL语句类型必须是DML语句。
- 系统参数SQL_MAP（默认值`FALSE`）为true时映射才会生效，为false时本语句可以执行成功，但映射并不会生效。
- 本语句不会校验原始语句和映射语句的正确性，输入错误的语句将导致执行时报错或输出非预期的结果。

语句定义

create_sqlmap::=

```
syntax::= CREATE SQLMAP map_name "(" user_name "," "'sql'" "," "'map_sql'"  ")"
```

map_name

将要创建的SQL映射的名称，该名称全局唯一，不能与已有SQL映射名称重复，不能带模式名指定，且需符合YashanDB的对象命名规范。

user_name

指定SQL映射对应的用户名，可为如下值：

- ALL：表示该映射对所有用户都生效。
- CURRENT_USER：表示该映射对当前的用户生效。
- 数据库中存在的用户名，对于使用双引号命名创建的用户，需为其实际大小写名称，否则不区分大小写。

sql

原始语句，如果指定的语句已经被创建过映射，不能再次创建。

原始语句长度不能超过32000字节。

map_sql

映射语句，必须与原始语句是相同的SQL语句类型，例如都为SELECT语句。

映射语句长度不能超过32000字节。

示例

```
--创建一个SQL映射，将查询所有机构的语句进行优化，改为获取区域有效的机构，且只获取有用的字段
CREATE SQLMAP map_branch (sales,'SELECT * FROM branches','select branch_no,branch_name from branches where area_no in (select
area_no from area)');

--sales用户执行SELECT * FROM branches时的效果
SELECT * FROM branches;
BRANCH_NO BRANCH_NAME
--------- ----------------------------
0101      上海
0102      南京
0103      福州
0104      厦门
0401      北京
0402      天津
0403      大连
0404      沈阳
0201      成都
0502      长沙
```

```
--不能为原始语句创建多个映射
CREATE SQLMAP map_branch (ALL,'SELECT * FROM branches','select branch_no,branch_name from branches where area_no in (select area_no from area)');
[1:31]YAS-04398 duplicate sql
```

CREATE SYNONYM

通用描述

CREATE SYNONYM用于为一个数据库对象创建一个同义词对象。用户可以在SQL语句中使用同义词来代替原始对象。

分布式部署中用户无法执行本语句。

语句定义

CREATE SYNONYM::=

```
syntax::= CREATE [OR REPLACE] [EDITIONABLE|NONEDITIONABLE] [PUBLIC] SYNONYM [schema "."] synonym FOR [schema "."] object
```

or replace

该语句用于指定当要创建的同义词已经存在时，将其进行重建。

public

该语句用于指定同义词为公共同义词，即所有用户都能访问该同义词，此时不能再为同义词指定schema属主。

本语句可省略，默认为私有同义词，则执行schema语句描述规则。

editionable | noneditionable

用于语法兼容，无实际含义。

schema

该语句用于指定同义词的所有者，即用户名。可省略，则默认所有者为当前用户。

synonym

该语句用于指定要创建的同义词的名称，不可省略，且需符合YashanDB的对象命名规范。

object

该语句用于指定要创建同义词的对象名称。此时，不会检查该对象是否存在或有效，只有在使用该同义词时才会提示相应错误。

示例（单机、共享集群部署）

```
--创建本用户对象的公共同义词
CREATE PUBLIC SYNONYM sy_area1 FOR area;

--创建其他用户对象的私有同义词，即使area表不存在，也可以创建成功
CREATE SYNONYM sy_area2 FOR sales.area;
```

CREATE TABLE AS

通用描述

CREATE TABLE AS语句根据一个子查询结果创建一张新的表对象，包括列字段及其类型，同时插入查询结果包含的数据集。

创建的表的列字段与子查询的列项具有相同的名称和类型，但列字段名称可以通过指定语句重置，类型则不可改变。同时还可以指定表的如下属性：

- 列约束
- 存储参数
- 分区（不能为临时表创建分区）
- 表空间

语句定义

create table as::=

```
syntax::= CREATE [(GLOBAL|PRIVATE) TEMPORARY] TABLE [schema "."] table_name [column_clause] [table_properties] AS subquery
```

column_clause::=

```
syntax::= "(" (out_of_line_constraint|column_name [DEFAULT default_expr] [inline_constraint])      {","
(out_of_line_constraint|column_name [DEFAULT default_expr] [inline_constraint])} ")"
```

global|private temporary

该语句用于指定创建的表为临时表，语法同CREATE TABLE中描述。

table_name

该语句用于指定创建的表的名称，语法同CREATE TABLE中描述。

out_of_line_constraint

该语句用于定义表的行外约束项，语法同CREATE TABLE中描述。

column_name

该语句用于重置表的列字段名称，不定义此项时，默认按子查询结果的所有列项及类型创建表的各项列字段及类型；定义此项时，则必须与子查询结果的列项一一对应，且名称符合YashanDB的对象命名规范。

default default_expr

定义列字段的缺省值，可省略，default_expr可以为字面量|运算式|函数等表达式。

inline_constraint

为列字段定义行内约束项，语法同CREATE TABLE中描述。

table_properties

该语句用于指定表的存储、分区、表空间等属性，语法同CREATE TABLE中描述。

subquery

分布式部署中，不允许子查询语句包含分布式系统视图，否则返回错误。

语法同SELECT中描述。

示例

```
--1、复制表的所有列及数据到新表，包括列字段名称、列字段类型和表数据，不包括约束项、存储参数、分区、表空间
CREATE TABLE branches_copy_all AS SELECT * FROM branches;
--添加false条件可以只复制表结构而不复制数据
CREATE TABLE branches_copy_no_data AS SELECT * FROM branches WHERE 1=2;

--2、复制表的部分列，重置列字段名称，并定义行内约束，此时列字段必须与子查询列项一一对应
CREATE TABLE branches_copy_column(branch_no1 PRIMARY KEY,branch_name) AS SELECT branch_no,branch_name bname FROM branches;

--3、复制表，并定义行外约束和表空间
CREATE TABLE branches_copy_add(FOREIGN KEY (area) REFERENCES area(area_no)) TABLESPACE yashan
AS SELECT branch_no,branch_name,area_no area FROM branches;
```

CREATE TABLE

通用描述

CREATE TABLE语句用于创建一个表对象，表类型可分为HEAP表、TAC表和LSC表。分布式部署中，本语句还可以指定表对象的分布类型：分布表（Sharded Table）或复制表（Duplicated Table），默认为分布表。

YashanDB支持通过配置DEFAULT_TABLE_TYPE参数（HEAP|TAC|LSC）指定创建表对象时的默认表类型，且该参数允许在线切换。同时，支持通过ORGANIZATION语法在创建表对象时指定其表类型。表对象创建成功后无法再修改表类型。

类型	存储方式	存储结构	适用部署形态
HEAP Table	行存	段页式结构	• 单机部署 • 共享集群部署
TAC (Transaction Analytics Columnar) Table	列存	段页式结构	• 单机部署 • 分布式部署
LSC (Large-scale Storage Columnar) Table	列存	列存结构	• 单机部署 • 分布式部署

创建LSC表前，需确保其所在表空间已挂载bucket（数据桶）。通过CREATE USER语句创建的普通用户默认所属表空间会默认挂载bucket，因此普通用户可直接创建LSC表，而系统用户则因产品架构而异：

- 单机部署中，由于SYSTEM表空间默认未挂载bucket，系统用户（如SYS）无法直接创建LSC表，需先执行ALTER TABLESPACE语句为SYSTEM表空间挂载bucket再创建LSC表。
- 分布式部署中，由于系统用户创建的表对象默认在users表空间中且users表空间默认已挂载bucket，无需额外操作即可直接创建LSC表。

> **Note**：
> 在实际生产环境中，建议在创建业务表前先为其合理规划表空间和用户，相关语句请查阅CREATE TABLESPACE和CREATE USER。

语句定义

create table::=

```
syntax::= CREATE [(GLOBAL|PRIVATE) TEMPORARY|SHARDED|DUPLICATED] TABLE [IF NOT EXISTS] [schema "."] table_name
"("  relation_properties ")"
[table_properties]
[lsc_table_properties]
[row_movement_clause]
```

relation_properties**::=**

```
syntax::= (column_definition|out_of_line_constraint)
{"," (column_definition|out_of_line_constraint)}
```

out_of_line_constraint定义

column_definition**::=**

```
syntax::= column dataType [(DEFAULT default_expr|codec_expr|inline_constraint)
{" " (DEFAULT default_expr|codec_expr|inline_constraint)}]
```

codec_expr**::=**

```
syntax::= [compression_clause] [encoding_clause]
```

compression_clause**::=**

```
syntax::= COMPRESSION (compression_type (HIGH|MEDIUM|LOW) | UNCOMPRESSED)
```

compression_type**::=**

```
syntax::= (LZ4|ZSTD)
```

encoding_clause**::=**

```
syntax::= ENCODING (PLAIN|RLE|DICTIONARY"("(RLE|(PLAIN ["," CARDINALITY]))")"|"BYTE-PACKED")
```

inline_constraint定义

table_properties**::=**

```
syntax::= (organization_clause
|temp_table_attr_clause
|physical_attribute_clause
|table_partition_clause
|lob_clauses
|logging_clause
|parallel_clause
|cache_clause|readonly_clause|inmemory_clause
|table_compression
|nested_table_clauses)
{" " (organization_clause
|temp_table_attr_clause
|physical_attribute_clause
|table_partition_clause
|lob_clauses
|logging_clause
|parallel_clause
|cache_clause|readonly_clause|inmemory_clause
|table_compression
|nested_table_clauses)}
```

organization_clause**::=**

```
syntax::= ORGANIZATION (HEAP|TAC|LSC|EXTERNAL external_table_clause)
```

external_table_clause定义

temp_table_attr_clause**::=**

```
syntax::= ON COMMIT (((DROP|PRESERVE) DEFINITION)|((DELETE|PRESERVE) ROWS))
```

physical_attribute_clause**::=**

```
syntax::= ((TABLESPACE tablespace|TABLESPACE SET tablespace_set)|PCTFREE integer|PCTUSED integer|INITRANS integer|MAXTRANS
integer|storage_clause|deferred_segment_creation)
```

storage_clause定义

deferred_segment_creation**::=**

```
syntax::= SEGMENT CREATION (IMMEDIATE | DEFERRED)
```

table_partition_clause**::=**

```
syntax::= (range_partitions
|list_partitions
|hash_partitions
|composite_range_partitions
|composite_list_partitions
|composite_hash_partitions
|consistent_hash_partitions
|consistent_hash_with_subpartitions)
```

range_partitions**::=**

```
syntax::= PARTITION BY RANGE "(" ((column) {"," (column)}) ")" [interval_clause]
"(" (PARTITION [partname]  range_values_clause  [table_partition_description | lob_clauses])
{"," (PARTITION [partname]  range_values_clause  [table_partition_description | lob_clauses])} ")"
```

interval_clause**::=**

```
syntax::= INTERVAL "(" expr ")" [STORE IN "(" ((tablespace) {"," (tablespace)}) ")"]
```

range_values_clause**::=**

```
syntax::= VALUES LESS THAN "(" (literal|MAXVALUE) {"," (literal|MAXVALUE)} ")"
```

table_partition_description**::=**

```
syntax::= (TABLESPACE tablespace|PCTFREE integer|PCTUSED integer|INITRANS integer|MAXTRANS integer|deferred_segment_creation)
```

list_partitions**::=**

```
syntax::= PARTITION BY LIST "(" (column) {"," (column)} ")" "(" (PARTITION [partname]
list_values_clause  [table_partition_description | lob_clauses])
{"," (PARTITION [partname]  list_values_clause  [table_partition_description | lob_clauses])} ")"
```

list_values_clause**::=**

```
syntax::= VALUES "(" (DEFAULT|list_values) ")"
```

list_values**::=**

```
syntax::= ((literal|NULL) {"," (literal|NULL)})
|(("(" ((literal|NULL) {"," (literal|NULL)})")") {"," ("(" ((literal|NULL) {"," (literal|NULL)})")")})
```

hash_partitions**::=**

```
syntax::= PARTITION BY HASH "(" ((column) {"," (column)}) ")"
(individual_partition_clause | hash_partitions_by_quantity)
```

individual_partition_clause**::=**

```
syntax::= "(" ( PARTITION [partname]  [partition_storage_clause | lob_clauses] )
{"," ( PARTITION [partname]  [partition_storage_clause | lob_clauses] )}")"
```

partition_storage_clause**::=**

```
syntax::= TABLESPACE tablespace
```

hash_partitions_by_quantity::=

```
syntax::= PARTITIONS hash_partition_quantity [STORE IN "(" ((tablespace) {"," (tablespace)}) ")"]
[OVERFLOW STORE IN "(" ((tablespace) {"," (tablespace)}) ")"]
```

composite_range_partitions::=

```
syntax::= PARTITION BY RANGE "(" ((column) {"," (column)}) ")"
(subpartition_by_range|subpartition_by_list|subpartition_by_hash)
"(" ((range_partition_desc) {"," (range_partition_desc)}) ")"
```

subpartition_by_range::=

```
syntax::= SUBPARTITION BY RANGE "(" ((column) {"," (column)}) ")" [subpartition_template]
```

subpartition_template::=

```
syntax::= SUBPARTITION TEMPLATE (("(" (
((range_subpartition_desc) {"," (range_subpartition_desc)})
| ((list_subpartition_desc) {"," (list_subpartition_desc)})
| (individual_hash_subparts { "," individual_hash_subparts})) ")"))
```

range_subpartition_desc::=

```
syntax::= (SUBPARTITION [subpartname] range_values_clause [TABLESPACE tablespace | lob_clauses] [deferred_segment_creation])
```

list_subpartition_desc::=

```
syntax::= (SUBPARTITION [subpartname] list_values_clause [TABLESPACE tablespace | lob_clauses] [deferred_segment_creation])
```

individual_hash_subparts::=

```
syntax::= SUBPARTITION [subpartname] [TABLESPACE tablespace | lob_clauses] [deferred_segment_creation]
```

subpartition_by_list::=

```
syntax::= SUBPARTITION BY LIST "(" ((column) {"," (column)}) ")" [subpartition_template]
```

subpartition_by_hash::=

```
syntax::= SUBPARTITION BY HASH "(" ((column) {"," (column)}) ")"
[(subpartition_template | hash_subparts_by_quantity)]
```

hash_subparts_by_quantity::=

```
syntax::= SUBPARTITIONS integer [STORE IN "(" ((tablespace) {"," (tablespace)}) ")"]
```

range_partition_desc::=

```
syntax::= PARTITION [partname]  range_values_clause  [table_partition_description]
[hash_subparts_by_quantity | ("(" (
((range_subpartition_desc) {"," (range_subpartition_desc)})
| ((list_subpartition_desc) {"," (list_subpartition_desc)})
| ((individual_hash_subparts) {"," (individual_hash_subparts)})) ")")]
```

composite_list_partitions::=

```
syntax::= PARTITION BY LIST "(" ((column) {"," (column)}) ")"
(subpartition_by_range | subpartition_by_list | subpartition_by_hash)
"(" ((list_partition_desc) {"," (list_partition_desc)}) ")"
```

list_partition_desc::=

```
syntax::= PARTITION [partname]  list_values_clause [table_partition_description]
[hash_subparts_by_quantity | ("(" (
((range_subpartition_desc) {"," (range_subpartition_desc)})
| ((list_subpartition_desc) {"," (list_subpartition_desc)})
| ((individual_hash_subparts) {"," (individual_hash_subparts)})) ")")]
```

composite_hash_partitions::=

```
syntax::= PARTITION BY HASH "(" ((column) {"," (column)}) ")"
(subpartition_by_range | subpartition_by_list | subpartition_by_hash)
(("(" ((hash_partition_desc) {"," (hash_partition_desc)}) ")") | hash_partitions_by_quantity)
```

hash_partition_desc::=

```
syntax::= PARTITION [partname] [table_partition_description]
["(" (
((range_subpartition_desc) {"," (range_subpartition_desc)})
| ((list_subpartition_desc) {"," (list_subpartition_desc)})
| ((individual_hash_subparts) {"," (individual_hash_subparts)})) ")"]
```

consistent_hash_partitions::=

```
syntax::= PARTITION BY [CONSISTENT] HASH "(" ((column) {"," (column)}) ")"
(individual_partition_clause | hash_partitions_by_quantity)
```

consistent_hash_with_subpartitions::=

```
syntax::= PARTITION BY [CONSISTENT] HASH "(" ((column) {"," (column)}) ")"
(subpartition_by_range|subpartition_by_list|subpartition_by_hash)
(("(" ((hash_partition_desc) {"," (hash_partition_desc)}) ")") | hash_partitions_by_quantity)
```

lob_clauses::=

```
syntax::= (lob_clause) {" " (lob_clause)}
```

lob_clause::=

```
syntax::= LOB "("(column) {"," (column)} ")" STORE AS [BASICFILE|SECUREFILE]
"("
    (
    TABLESPACE space_name
    |(ENABLE|DISABLE) STORAGE IN ROW
    | CHUNK integer
    | (CACHE | NOCACHE | CACHE READS) [LOGGING | NOLOGGING]
    | ((COMPRESS [LOW|MEDIUM|HIGH])|NOCOMPRESS)
    | (DEDUPLICATE|KEEP_DUPLICATES)
    )
{
    " "
    (
    TABLESPACE space_name
    |(ENABLE|DISABLE) STORAGE IN ROW
```

```
    | CHUNK integer
    | (CACHE | NOCACHE | CACHE READS) [LOGGING | NOLOGGING]
    | ((COMPRESS [LOW|MEDIUM|HIGH])|NOCOMPRESS)
    | (DEDUPLICATE|KEEP_DUPLICATES)
    )}
")"
```

logging_clause::=

```
syntax::= LOGGING | NOLOGGING
```

parallel_clause::=

```
syntax::= NOPARALLEL | PARALLEL [integer]
```

cache_clause::=

```
syntax::= CACHE | NOCACHE
```

readonly_clause::=

```
syntax::= READONLY | READWRITE
```

inmemory_clause::=

```
syntax::= INMEMORY | NO INMEMORY
```

table_compression::=

```
syntax::= COMPRESS | NOCOMPRESS
```

nested_table_clauses::=

```
syntax::= (nested_table_clause) {" " (nested_table_clause)}
```

nested_table_clause::=

```
syntax::= NESTED TABLE (nested_item|COLUMN_VALUE) [LOCAL|GLOBAL] STORE AS
storage_table ["(" nested_table_clause ")"] ["(" TABLESPACE tablespace_name ")"]
```

lsc_table_properties::=

```
syntax::= [compression_clause] [table_sort_clause] [mcol_ttl_clause] [transformer_clause] storage_table
```

table_sort_clause::=

```
syntax::= ORDER BY "("(column_name){"," (column_name)}")" [NULLS (FIRST|LAST)] [ASC|DESC] [SCOL]
```

mcol_ttl_clause::=

```
syntax::= MCOL TTL timestamp
```

transformer_clause::=

```
syntax::= (ENABLE|DISABLE) ((TRANSFORM|COMPACT|BUILD AC) {(TRANSFORM|COMPACT|BUILD AC)})
```

row_movement_clause**::=**

```
syntax::= (ENABLE|DISABLE) ROW MOVEMENT
```

global temporary

该语句只作用于HEAP表和单机TAC表，用于指定创建的表为全局临时表。全局临时表对数据库所有会话（Session）可见，但表中数据在各会话间隔离，即每个会话只能看到在本会话中插入的数据。

全局临时表被创建后将一直存在（除非被DROP），ON COMMIT DELETE | PRESERVE ROWS则定义了其数据在事务提交后是否保留，具体在temp_table_attr_clause中说明。

示例（HEAP表和单机TAC表）

```
CREATE GLOBAL TEMPORARY TABLE globaltemtable(c1 INT,c2 INT);
```

private temporary

该语句只作用于HEAP表和单机TAC表，用于指定创建的表为私有临时表。在某个会话中创建的私有临时表，其表结构及数据只对本会话可见。

私有临时表的数据在当前会话中一直保留，且不会在事务提交时被删除，ON COMMIT DROP | PRESERVE DEFINITION则定义了事务提交后是否DROP该表，具体在temp_table_attr_clause中说明。

私有临时表的名称必须以`YAS$PTT_`开头，并且不适用于创建约束。而其他类型的表名称则不允许以`YAS$PTT_`开头。

示例（HEAP表和单机TAC表）

```
CREATE PRIVATE TEMPORARY TABLE YAS$PTT_privatetemtable(c1 INT,c2 INT);

--如不以YAS$PTT_开头会返回错误
CREATE PRIVATE TEMPORARY TABLE privatetemtable(c1 INT,c2 INT);
YAS-02170 create a private temporary table with a name not matching 'YAS$PTT_' prefix
```

sharded

该语句只在分布式部署中使用，表示创建一张分布表，分布表在每个数据节点上依据Chunk将数据分片存储。

YashanDB将分布表的每一个分区（Partition）作为一个Chunk，插入表中的数据将按照指定的规则被分配到各个Chunk中，可通过consistent_hash_partitions语句定义这种规则，不指定规则时，将对表按如下规则进行哈希分区来分片存储数据：

- 如果表上定义了主键列，则分区键列为主键列。
- 如果表上未定义主键列，但存在唯一索引，则分区键为所有唯一索引的公共子集中首个恰当数据类型的字段。
- 否则，分区键为表的首个恰当数据类型的列字段。

SHARDED为YashanDB分布式数据库的默认建表方式，即CREATE TABLE=CREATE SHARDED TABLE。

示例（分布式部署）

```
--创建默认的分布表并插入记录
CREATE SHARDED TABLE area_shard
(area_no CHAR(2) NOT NULL,
area_name VARCHAR2(60),
DHQ VARCHAR2(20) DEFAULT 'ShenZhen' NOT NULL);

--查询表属性，sharded=Y表示为分布表
SELECT table_name,table_type,partitioned,SHARDED,DUPLICATED
FROM USER_TABLES
WHERE table_name='AREA_SHARD';
TABLE_NAME                TABLE_TYPE PARTITIONED SHARDED DUPLICATED
```

```
------------------------- ---------- ----------- ------- ----------
AREA_SHARD                LSC        N           Y       N

--查询分布表信息
SELECT table_name,dist_type,dist_key_count FROM USER_DIST_TABLES
WHERE table_name='AREA_SHARD';
TABLE_NAME               DIST_TYPE     DIST_KEY_COUNT
------------------------ ------------- --------------
AREA_SHARD               HASH                      1

--查询默认的分区键
SELECT name,column_name,column_position
FROM USER_DIST_KEY_COLUMNS
WHERE name='AREA_SHARD';
NAME               COLUMN_NAME       COLUMN_POSITION
------------------ ----------------- ---------------
AREA_SHARD         AREA_NO                         0

--查询表分区数量，即Chunk数量
SELECT COUNT(1)
FROM USER_TAB_PARTITIONS
WHERE table_name='AREA_SHARD';
             COUNT(1)
---------------------
                    6
```

duplicated

该语句只在分布式部署中使用，表示创建一张复制表，插入表中的数据将被完全复制到各个数据节点中。

示例（分布式部署）

```
--创建复制表
CREATE DUPLICATED TABLE area_dupli
(area_no CHAR(2) NOT NULL,
area_name VARCHAR2(60),
DHQ VARCHAR2(20) DEFAULT 'ShenZhen' NOT NULL);

--查询表属性，duplicated=Y表示为复制表
SELECT table_name,table_type,partitioned,SHARDED,DUPLICATED
FROM USER_TABLES
WHERE table_name='AREA_DUPLI';
TABLE_NAME                TABLE_TYPE PARTITIONED SHARDED DUPLICATED
------------------------- ---------- ----------- ------- ----------
AREA_DUPLI                LSC        N           N       Y
```

if not exists

该语句用于在CREATE表之前，先判断该表是否存在，省略则不会判断。若不省略，此时如果要创建的表已存在，系统将不会报错，并且使用旧表，不会创建新的表。

示例

```
--创建已存在的表area时，不会报错，且新表未创建成功
CREATE TABLE IF NOT EXISTS area(c1 INT, c2 INT);

SELECT * FROM area WHERE area_no='01';
AREA_NO AREA_NAME        DHQ
------- ---------------- ----------
01      华东             Shanghai
```

table_name

该语句用于指定创建的表的名称，不可省略，且需符合YashanDB的对象命名规范。

如表名称里出现了`.`符号，表示该符号之前为用户名、之后为表名，此时必须拥有在指定用户名下创建表对象的权限。

relation_properties

该语句用于指定创建的表的结构，包括列字段（Column）和约束（Constraint）。

column_definition

该语句用于定义表的列字段项。

column

列字段的名称，不可省略，且需符合YashanDB的对象命名规范。

datatype

指定列字段的数据类型。查看YashanDB的数据类型描述。

其中，数据类型指定为LOB/JSON类型时，可通过lob_clauses子句指定其存储属性。

数据类型指定为Nested Table UDT类型时，必须通过nested_table_clauses子句创建嵌套表，且不能为临时表的列字段指定Nested Table UDT类型。

default default_expr

列字段的缺省值。default_expr可以为字面量、运算式、函数等表达式。

codec_expr

对于LSC表的列字段，使用本语句定义列字段的编码压缩属性，系统按此设置存储该列上的数据。 对于单机TAC表的列字段，使用本语句定义列字段的字典编码属性，系统按此设置存储该列上的数据。

compression_clause

该语句用于对于LSC表定义压缩属性。

- 压缩算法：可指定为LZ4或ZSTD压缩算法，或指定为UNCOMPRESSED不对数据进行压缩。
- 压缩级别：指定压缩算法为LZ4或ZSTD后，可指定按LOW、MEDIUM、HIGH三个级别进行压缩。若不指定级别，会根据配置项COMPRESSION_LEVEL默认指定。

对于某一个列上的数据，基于表列级别定义的压缩属性与系统默认的压缩属性之间的优先关系如下：

列级压缩属性 > 表级压缩属性 > 系统参数COMPRESSION与COMPRESSION_LEVEL的相关配置。

建议您根据数据库的使用场景，选择合适的压缩算法。LZ4压缩算法的优点是：解压与压缩速度快，并且有着较好的压缩效果，ZSTD压缩算法相对于LZ4压缩算法，解压和压缩速度较慢，但是压缩效果更好。

- 如果您不能确定表的使用场景，推荐您使用系统默认的LZ4 LOW进行压缩，大多数场景下，LZ4可以用较少的CPU资源，减少IO需要时间。
- 对于有高查询性能要求的场景，建议您使用LZ4进行压缩，压缩等级需要根据您对数据导入和冷热数据转换的性能要求来决定，低压缩等级时数据导入和冷热转换速度更快。
- 对于归档数据，且查询性能要求较为宽松，建议您使用ZSTD进行压缩，等级可根据您对导入速度的要求进行调整。

YashanDB支持按表级别或列级别定义压缩属性，并提供如下配置参数用于设置系统默认的压缩属性。

compression_type

该子句用于指定压缩算法及压缩机别。

- COMPRESSION：是否进行数据压缩。
- COMPRESSION_LEVEL：数据压缩的级别。

encoding_clause

对于LSC表的列定义编码属性，指定按如下一种方式进行编码：

- PLAIN编码：即不编码，按原始方式存储数据。
- RLE编码：游程长度编码。
- DICTIONARY(PLAIN)：字典编码后的数据进行PLAIN编码。
- DICTIONARY(RLE)：字典编码后的数据进行RLE编码。

- BYTE-PACKED：根据数据字节长度编码。

不指定编码时，默认采用自适应编码，即内部自动根据数据类型与数据特征，自适应地探测合适的编码类型，对同一列在不同数据块中支持使用不同的算法来进行编码。

对于某一个列上的数据，是否应该编码和采用何种编码方式，与该列的数据类型和数据特点相关，如下为YashanDB对不同数据类型可适配的编码方式（✓表示适配，X表示不适配）：

数据类型	PLAIN编码	RLE编码	字典编码	BYTE-PACKED编码
TINYINT/SMALLINT/大对象型	✓	X	X	X
INT/BIGINT/FLOAT/DOUBLE/日期时间型	✓	✓	✓	X
字符型	✓	X	✓	X
NUMBER型	✓	X	X	✓

其中，BOOLEAN类型数据由系统自动进行布尔编码存储，无需为其指定编码方式。长度大的CHAR字段采用字典编码对内存占用比较大，对于超过128字节的CHAR类型，请尽量不要选择字典编码。

示例（LSC表）

```
--创建LSC表，压缩指定列且对其字典编码存储
DROP TABLE IF EXISTS finance_info;
CREATE TABLE finance_info
(year CHAR(4) NOT NULL COMPRESSION lz4 HIGH ENCODING DICTIONARY(RLE),
month CHAR(2) NOT NULL,
branch CHAR(4),
revenue_total NUMBER(10,2),
cost_total NUMBER(10,2),
fee_total NUMBER(10,2)
);
```

对于TAC表的列定义字典编码属性，指定按DICTIONARY(PLAIN)方式进行编码，该属性仅支持数据类型为字符型的列使用。 同时TAC表的列使用该属性支持指定字典值上限，规格为[1,65535]，缺省默认字典值上限为65535。

示例（单机TAC表）

```
--创建TAC表，对指定列字典编码存储且指定字典上限为100
DROP TABLE IF EXISTS finance_info;
CREATE TABLE finance_info
(year CHAR(4) NOT NULL ENCODING DICTIONARY(PLAIN, 100),
month CHAR(2) NOT NULL,
branch CHAR(4),
revenue_total NUMBER(10,2),
cost_total NUMBER(10,2),
fee_total NUMBER(10,2)
);
```

inline_constraint

该语句用于定义表的行内约束项。关于约束项的详细描述请参考通用SQL语法constraint。

其中，不能为临时表建立FOREIGN KEY约束，也不能为其他表建立到临时表的FOREIGN KEY约束。

out_of_line_constraint

该语句用于定义表的行外约束项。关于约束项的详细描述请参考通用SQL语法constraint。

其中，不能为临时表建立FOREIGN KEY约束，也不能为其他表建立到临时表的FOREIGN KEY约束。

table_properties

该语句用于指定创建的表的各项属性，多项间以空格进行分隔。

organization_clause

该语句指定表的数据存储方式（表类型），可省略，则系统按DEFAULT_TABLE_TYPE参数的值创建相应类型的表。YashanDB针对不同表类型的数据存储方式参考产品描述存储引擎。

heap

创建HEAP行存表。

tac

创建TAC列存表。

lsc

创建LSC列存表，此种表类型要求默认或指定表空间为databucket表空间，否则创建失败。

external

创建外部表，外部表为一种表结构存储在数据库内，而数据存储在数据库外的特殊表类型。由于外部表的数据未存储在数据库内，因此基于行的所有DDL操作，如索引、行迁移、闪回等，均不适用于外部表。

外部表对应数据只能被查询，不可增删改。

对于外部表而言，关于segment的属性设置没有意义也不会实际生效。

external_table_clause

external_table_clause外部表所对应的外部数据结构定义，具体描述参考通用语法EXTERNAL_TABLE。

本语句可省略，若省略则所创建的外部表不指定相应的外部数据结构，该表将无法正常查询。

示例（单机、分布式部署）

```
CREATE TABLE area_tac
(area_no CHAR(2) NOT NULL,
area_name VARCHAR2(60),
DHQ VARCHAR2(20) DEFAULT 'ShenZhen' NOT NULL)
ORGANIZATION TAC;
```

temp_table_attr_clause

该语句只针对全局临时表和私有临时表，且DELETE|PRESERVE ROWS只可用于全局临时表，而DROP|PRESERVE DEFINITION只可用于私有临时表。

若省略该语句，默认值分别为DELETE ROWS和DROP DEFINITION。

on commit delete|preserve rows

指定事务提交后在当前会话中插入的全局临时表的数据是否保留，DELETE为删除，PRESERVE为保留。

示例（HEAP表、单机TAC表）

```
--创建全局临时表T_orders_info，并指定PRESERVE ROWS
CREATE GLOBAL TEMPORARY TABLE T_orders_info (
order_no CHAR(14) NOT NULL,
order_desc VARCHAR2(100),
area CHAR(2),
branch CHAR(4),
order_date DATE DEFAULT SYSDATE NOT NULL,
salesperson CHAR(10))
ON COMMIT PRESERVE ROWS;

--插入数据并提交
INSERT INTO T_orders_info VALUES ('20010102020001','product 001','02','0202',SYSDATE-400,'0001');
COMMIT;

--查询结果，由于指定了PRESERVE ROWS，则仍可以查询到数据（且只为本会话插入的数据），如为DELETE ROWS则无法查询到数据
SELECT * FROM T_orders_info;
```

```
ORDER_NO          ORDER_DESC     AREA  BRANCH ORDER_DATE             SALESPERSON
----------------- -------------- ----- ------ ---------------------- -------------
20010102020001    product 001    02    0202   2020-12-06 22:55:32    0001
```

on commit drop|preserve definition

指定当前会话中创建的私有临时表是否在事务提交后被删除，DROP表示删除，PRESERVE表示保留。

示例（单机HEAP表、TAC表）

```
--创建私有临时表PTT_orders_info，并指定DROP DEFINITION
CREATE PRIVATE TEMPORARY TABLE YAS$PTT_orders_info (
order_no CHAR(14),
order_desc VARCHAR2(100),
area CHAR(2),
branch CHAR(4),
order_date DATE DEFAULT SYSDATE,
salesperson CHAR(10))
ON COMMIT DROP DEFINITION;

--插入数据并提交
INSERT INTO YAS$PTT_orders_info VALUES ('20010102020001','product 001','02','0202',SYSDATE-400,'0001');
COMMIT;

--查询结果，由于指定了DROP DEFINITION，事务提交后PTT_orders_info表被删除
SELECT * FROM YAS$PTT_orders_info;
[1:15]YAS-02012 table or view does not exist
```

physical_attribute_clause

该语句用于指定创建的表的物理存储属性或创建表时是否立即创建segment。

tablespace tablespace_name

该语句用于指定表所在的表空间。

对于临时表，只能指定一个temporary类型的表空间，省略则默认为表所属用户所在的表空间（临时表的表空间默认为数据库创建时生成的temporary表空间）。

对于LSC表，为其所指定的表空间必须拥有bucket属性，详见CREATE TABLESPACE描述。其中，系统的缺省表空间（DEFAULT表空间）已默认拥有bucket属性，可以作为LSC表的表空间。

分布式部署中，创建分布表时不能使用本语句指定表空间。

tablespace set tablespace_set_name

指定分布表所在的表空间集。省略时则默认为内置的USERS表空间集。

pctfree/pctused/initrans/maxtrans

指定表的PCTFREE/PCTUSED/INITRANS/MAXTRANS属性，省略则默认为8/NULL/2/255。

其中：

- PCTFREE：表示数据块为数据库对象进行UPDATE保留的空间百分比，当可用空间低于该百分比时无法进行INSERT，只能进行UPDATE。
- PCTUSED：表示数据块为数据库对象保留的最小已用空间百分比，当数据所占空间低于该百分比时可进行INSERT。
- INITRANS：表示每个数据块中初始并发事务项的数量。
- MAXTRANS：表示每个数据块中并发事务项数量的最大值。

storage_clause

指定为表分配的初始空间和最大空间大小，由于YashanDB采取延迟分配的策略（有数据才分配空间），不建议指定此参数。

deferred_segment_creation

segment creation deferred|immediate

用于指定创建表对象时segment的创建方式，该语法不适用于临时表。

- IMMEDIATE：表示立即创建。

- DEFERRED：表示延迟创建，默认为此方式。

示例（单机、共享集群部署）

```
CREATE TABLE employee_info (
name CHAR(10) NOT NULL,
age INT,
id INT NOT NULL
)TABLESPACE users PCTFREE 50 PCTUSED 20 INITRANS 3 MAXTRANS 254 SEGMENT CREATION DEFERRED;
```

table_partition_clause

该语句用于为表创建分区（Partition）或组合分区，可通过USER_TAB_PARTITIONS和USER_TAB_SUBPARTITIONS视图查看所有分区和子分区信息。

创建组合分区时，一级分区不能指定为INTERVAL类型的范围分区（range_partitions）。

该语句包含如下限制：

- 不允许为临时表创建分区。
- 不允许将LOB列/JSON列指定为分区键和子分区键。

range_partitions

创建范围分区，分区列为多项时以`,`分隔。

如果指定了INTERVAL，那么分区列的数量必须为1，并且只能是数值或者时间类型。

interval_clause

创建INTERVAL类型的范围分区。

INTERVAL(expr)

范围分区根据分区界值将每个分区定义了下限值（即为前一分区的上限值）和上限值（分区界值），最后的一个分区其上限值即为最大的分区界值。指定INTERVAL表示，当要插入的数据超过最大分区界值时，系统将根据如下规则创建一个新分区，并将这行数据插入到新分区中：

- 新分区由系统自动命名。
- 新分区的上限值=最大分区界值+INTERVAL值，之后新分区的上限值将成为最大分区界值。
- 如果最大分区界值已被定义为MAXVALUE，则无法创建新分区，该数据无法插入表。
- expr为表达式，表达式的结果类型需与分区列字段的数据类型一致。

此时，如表上已建立了分区索引（Local Partitioned Index），则系统在创建新分区的同时，也会创建相应的索引分区。

STORE IN

为INTERVAL分区指定表空间，指定多个表空间以`,`分隔，则系统在创建新分区时，按照此处定义的顺序，循环使用作为新分区的表空间。未指定本语句时分区的表空间默认为表所在的表空间。

示例（单机、共享集群部署）

```
--创建范围分区表orders_info,并指定INTERVAL值
DROP TABLE IF EXISTS orders_info;
CREATE TABLE orders_info (order_no CHAR(14) NOT NULL,
order_desc VARCHAR2(100),
area CHAR(2),
branch CHAR(4),
order_date DATE DEFAULT SYSDATE NOT NULL,
salesperson CHAR(10),
id NUMBER)
TABLESPACE yashan
PARTITION BY RANGE (id)
INTERVAL (1000)
(PARTITION p_orders_info_1 VALUES LESS THAN (800));

--插入三条数据均成功，如未指定INTERVAL值，则只有第一条数据可以插入成功
INSERT INTO orders_info VALUES ('20010102020001','product 001','02','0201',SYSDATE-400,'0001',300);
INSERT INTO orders_info VALUES ('20010102020001','product 001','02','0201',SYSDATE-400,'0001',2000);
```

```
INSERT INTO orders_info VALUES ('20010102020001','product 001','02','0201',SYSDATE-400,'0001',20000);
COMMIT;

--查询该表上的分区，除p_orders_info_1 分区外，系统自动创建了另外两个分区
SELECT TABLE_NAME t_name,TABLESPACE_NAME ts_name,PARTITION_NAME par_name,HIGH_VALUE par_max FROM USER_TAB_PARTITIONS WHERE
TABLE_NAME='ORDERS_INFO';
T_NAME             TS_NAME   PAR_NAME         PAR_MAX
---------------- ---------- ---------------- ---------
ORDERS_INFO       YASHAN     SYS_P22          20800
ORDERS_INFO       YASHAN     SYS_P21          2800
ORDERS_INFO       YASHAN     P_ORDERS_INFO_1  800

--创建分区键为时间类型的INTERVAL范围分区表
DROP TABLE IF EXISTS orders_info;
CREATE TABLE orders_info (order_no CHAR(14) NOT NULL,
order_desc VARCHAR2(100),
area CHAR(2),
branch CHAR(4),
order_date DATE DEFAULT SYSDATE NOT NULL,
salesperson CHAR(10),
id NUMBER)
PARTITION BY RANGE (order_date)
INTERVAL (INTERVAL '3' month)
(PARTITION p_firstquarter VALUES LESS THAN (DATE '2000-01-01'));
```

range_values_clause
定义范围分区的上限值。

literal

分区列为多项时，则此处的上限值用`,`分隔并与分区列一一对应。

MAXVALUE

MAXVALUE表示无上限，即超过前一分区的上限的所有数据均进入此分区中，包括NULL。如所有分区列都定义了以MAXVALUE作为上限值的分区，不允许再继续定义MAXVALUE分区。

在定义了MAXVALUE的分区后，不能通过ADD PARTITION语句增加新的分区，仅能通过SPLIT PARTITION语句将已存在分区划分成新的分区。

示例（单机、共享集群部署）

```
DROP TABLE IF EXISTS orders_maxvalue;
CREATE TABLE orders_maxvalue(
order_no CHAR(14) NOT NULL,
order_desc VARCHAR2(100),
area CHAR(2),
branch CHAR(4),
order_date DATE DEFAULT SYSDATE NOT NULL,
salesperson CHAR(10),
id NUMBER)
PARTITION BY RANGE(id)
(PARTITION p_orders_max_1 VALUES LESS THAN (800),
 PARTITION p_orders_max_2 VALUES LESS THAN (1000),
 PARTITION p_orders_max_3 VALUES LESS THAN (1500),
 PARTITION p_orders_max_4 VALUES LESS THAN (MAXVALUE)
);

--创建分区列为多项的范围分区表
DROP TABLE IF EXISTS orders_multikey;
CREATE TABLE orders_multikey(
order_no CHAR(14) NOT NULL,
order_desc VARCHAR2(100),
area CHAR(2),
branch CHAR(4),
order_date DATE DEFAULT SYSDATE NOT NULL,
salesperson CHAR(10),
id NUMBER)
PARTITION BY RANGE(id,order_date)
```

```
(PARTITION p_orders_max_1 VALUES LESS THAN (800,'2010-01-01'),
 PARTITION p_orders_max_2 VALUES LESS THAN (1000,'2010-05-01'),
 PARTITION p_orders_max_3 VALUES LESS THAN (1500,'2010-10-01')
);

--所有分区列均定义MAXVALUE分区后继续定义MAXVALUE分区，此时返回错误
DROP TABLE IF EXISTS orders_multimax;
CREATE TABLE orders_multimax(
order_no CHAR(14) NOT NULL,
order_desc VARCHAR2(100),
area CHAR(2),
branch CHAR(4),
order_date DATE DEFAULT SYSDATE NOT NULL,
salesperson CHAR(10),
id NUMBER)
PARTITION BY RANGE(id,order_date)
(PARTITION p_orders_max_1 VALUES LESS THAN (800,'2010-01-01'),
 PARTITION p_orders_max_2 VALUES LESS THAN (1000,'2010-05-01'),
 PARTITION p_orders_max_3 VALUES LESS THAN (1500,'2010-10-01'),
 PARTITION p_orders_max_4 VALUES LESS THAN (MAXVALUE,'2010-10-01'),
 PARTITION p_orders_max_5 VALUES LESS THAN (MAXVALUE,MAXVALUE),
 PARTITION p_orders_max_6 VALUES LESS THAN (MAXVALUE,MAXVALUE)
);

YAS-04240 partition bound is too high
```

table_partition_description
定义范围分区的存储属性。

TABLESPACE tablespace_name

为分区指定一个表空间，省略则默认为表所在的表空间。

PCTFREE/PCTUSED/INITRANS/MAXTRANS

指定PCTFREE/PCTUSED/INITRANS/MAXTRANS属性，省略则默认为8/NULL/2/255。

各属性定义与physical_attribute_clause语句中的描述一致。

deferred_segment_creation

SEGMENT CREATION DEFERRED|IMMEDIATE

与physical_attribute_clause语句中SEGMENT CREATION的描述一致。

示例（单机部署）

```
--单机部署中创建定义了存储属性的范围分区
DROP TABLE IF EXISTS sales_info_range;
CREATE TABLE sales_info_range
(year CHAR(4) NOT NULL,
 month CHAR(2) NOT NULL,
 branch CHAR(4),
 product CHAR(5),
 quantity NUMBER DEFAULT 0 NOT NULL,
 amount NUMBER(10,2) DEFAULT 0 NOT NULL,
 salsperson CHAR(10))
PARTITION BY RANGE(year)
(PARTITION p_sales_info_range_1 VALUES LESS THAN('2011') PCTFREE 3 SEGMENT CREATION DEFERRED,
 PARTITION p_sales_info_range_2 VALUES LESS THAN('2021'),
 PARTITION p_sales_info_range_3 VALUES LESS THAN('2031'));
```

list_partitions
创建列表分区，分区列为多项时以`,`分隔。

list_values_clause

定义列表分区的列表内容。

DEFAULT

DEFAULT表示将在已定义列表值之外的所有数据均进入此分区中，如不指定DEFAULT分区，插入已定义列表值之外的所有数据会报错。如指定，须作为最后一个分区指定，且DEFAULT分区只允许指定一次。

在定义了DEFAULT的分区后，不能通过ADD PARTITION语句增加新的分区，仅能通过SPLIT PARTITION语句将已存在分区划分成新的分区。

list_values

分区列为多项时，须用`,`分隔并与分区列一一对应，该列表内容须指定为字面量或NULL。

table_partition_description
同范围分区描述一致。

示例（单机、共享集群部署）

```
--创建列表分区的销售信息表
DROP TABLE IF EXISTS sales_info;
CREATE TABLE sales_info
(year CHAR(4) NOT NULL,
month CHAR(2) NOT NULL,
branch CHAR(4) NOT NULL,
product CHAR(10) NOT NULL,
quantity NUMBER DEFAULT 0 NOT NULL,
amount NUMBER(10,2) DEFAULT 0 NOT NULL,
salsperson CHAR(10))
PARTITION BY LIST(year,month)
(PARTITION p_sales_info1_1 VALUES (('2018','01'),('2018','02')),
PARTITION p_sales_info1_2 VALUES ('2020','01'),
PARTITION p_sales_info1_3 VALUES (DEFAULT));
```

hash_partitions

创建哈希分区，分区列为多项时以`,`分隔，哈希分区有如下两种创建方式：

- 逐个指定哈希分区。
- 使用hash分数批量指定哈希分区。

individual_partition_clause
本语句用于逐个指定哈希分区，对每个分区分别定义其名称和存储属性，分区之间用`,`分隔。

partition_storage_clause

定义哈希分区的存储属性。

TABLESPACE tablespace_name

为分区指定一个表空间，省略则默认为表所在的表空间。此操作不适用于分布式部署。

示例（单机部署）

```
DROP TABLE IF EXISTS sales_info_hash;
CREATE TABLE sales_info_hash
(year CHAR(4) NOT NULL,
 month CHAR(2) NOT NULL,
 branch CHAR(4),
 product CHAR(5),
 quantity NUMBER DEFAULT 0 NOT NULL,
 amount NUMBER(10,2) DEFAULT 0 NOT NULL,
 salsperson CHAR(10))
PARTITION BY HASH(year)
(PARTITION p_sales_info_hash_1,
 PARTITION p_sales_info_hash_2);

-- 为分区指定表空间
```

```
DROP TABLE IF EXISTS sales_info_hashpar;
CREATE TABLE sales_info_hashpar
(year CHAR(4) NOT NULL,
 month CHAR(2) NOT NULL,
 branch CHAR(4))
PARTITION BY HASH(year)
(PARTITION p_sales_info_hash_1 TABLESPACE users,
 PARTITION p_sales_info_hash_2);
```

hash_partitions_by_quantity
本语句用于按hash分数批量指定哈希分区，所有分区名称由系统生成。

hash_partition_quantity

表示hash分区的数量。

STORE IN

STORE IN子句描述哈希分区所属的表空间，表空间可指定多个，且数量不需要与分区的数量一致， 系统将按本语句中表空间指定的顺序进行循环归属。

不指定本语句时，系统默认使用表所在的表空间作为所有分区的表空间。

示例（单机部署）

```
--单机部署中通过hash分数创建哈希分区表，并指定STORE IN语句
DROP TABLE IF EXISTS sales_info_hash;
CREATE TABLE sales_info_hash
(year CHAR(4) NOT NULL,
 month CHAR(2) NOT NULL,
 branch CHAR(4),
 product CHAR(5),
 quantity NUMBER DEFAULT 0 NOT NULL,
 amount NUMBER(10,2) DEFAULT 0 NOT NULL,
 salsperson CHAR(10))
PARTITION BY HASH(year)
PARTITIONS 4
STORE IN (yashan1,yashan2);
```

composite_range_partitions
定义范围分区的组合分区，每个范围分区下可以定义范围、列表、哈希三种子分区的一种。

subpartition_by_range
定义范围子分区的分区列或子分区模板。

range_partition_desc
创建范围分区及其子分区。

range_subpartition_desc
具体定义一个范围子分区表空间和segment是否立即创建的属性。

range_values_clause

定义范围分区的上限值。

literal

分区列为多项时，则此处的上限值用`,`分隔并与分区列一一对应。

MAXVALUE

MAXVALUE表示无上限，即超过前一分区的上限的所有数据均进入此分区中，包括NULL。如所有分区列都定义了以MAXVALUE作为上限值的分区，不允许再继续定义MAXVALUE分区。在定义了MAXVALUE的分区后，不能再增加新的分区。

deferred_segment_creation

SEGMENT CREATION IMMEDIATE|DEFERRED

与physical_attribute_clause语句中SEGMENT CREATION的描述一致。

示例（单机、共享集群部署）

```
-- range-hash组合分区
CREATE TABLE rh_composite(a INT, b INT)
PARTITION BY RANGE(a)
SUBPARTITION BY HASH(b)
(
PARTITION p1 VALUES LESS than(1) (SUBPARTITION sp1, SUBPARTITION sp2),
PARTITION p2 VALUES LESS than(2) (SUBPARTITION sp3, SUBPARTITION sp4)
);
-- range-list组合分区
CREATE TABLE rl_composite(a INT, b INT)
PARTITION BY RANGE(a)
SUBPARTITION BY LIST(b)
(
PARTITION p1 VALUES LESS than(1) (SUBPARTITION sp1 VALUES(10), SUBPARTITION sp2 VALUES(20)),
PARTITION p2 VALUES LESS than(2) (SUBPARTITION sp3 VALUES(10), SUBPARTITION sp4 VALUES(20))
);
-- range-range组合分区
CREATE TABLE rr_composite(a INT, b INT)
PARTITION BY RANGE(a)
SUBPARTITION BY RANGE(b)
(
PARTITION p1 VALUES LESS than(1) (SUBPARTITION sp1 VALUES LESS than(10), SUBPARTITION sp2 VALUES LESS than(20)),
PARTITION p2 VALUES LESS than(2) (SUBPARTITION sp3 VALUES LESS than(10), SUBPARTITION sp4 VALUES LESS than(20))
);
```

composite_list_partitions

定义列表分区的组合分区，每个列表分区下可以定义范围、列表、哈希三种子分区的一种。

subpartition_by_list

定义列表子分区的分区列或子分区模板。

list_partition_desc

创建列表分区及其子分区。

list_subpartition_desc

具体定义一个列表子分区的表空间和segment是否立即创建的属性。

list_values_clause

定义列表分区的列表内容。

DEFAULT

DEFAULT表示将在已定义列表值之外的所有数据均进入此分区中，如不指定DEFAULT分区，插入已定义列表值之外的所有数据会报错。如指定，须作为最后一个分区指定，且DEFAULT分区只允许指定一次。在定义了DEFAULT的分区后，不能再增加新的分区。

list_values

分区列为多项时，须用`,`分隔并与分区列一一对应，该列表内容须指定为字面量或NULL。

deferred_segment_creation

用于指定创建表对象时，segment的创建方式是立即创建或延迟创建，该语法不适用于临时表。

- IMMEDIATE：表示立即创建。
- DEFERRED：表示延迟创建。若省略，默认为延迟创建。

示例（单机、共享集群部署）

```
-- list-hash组合分区
CREATE TABLE lh_composite(a INT, b VARCHAR(10))
PARTITION BY LIST(a)
SUBPARTITION BY HASH(b)
subpartitions 8
(PARTITION p1 VALUES(10), PARTITION p2 VALUES(DEFAULT));
-- list-list组合分区
CREATE TABLE ll_composite(a INT, b INT)
PARTITION BY LIST(a)
SUBPARTITION BY LIST(b)
(
PARTITION p1 VALUES(1) (SUBPARTITION sp1 VALUES(1), SUBPARTITION sp2 VALUES(2)),
PARTITION p2 VALUES(2) (SUBPARTITION sp3 VALUES(1), SUBPARTITION sp4 VALUES(2))
);
-- list-range组合分区
CREATE TABLE lr_composite(a INT, b INT)
PARTITION BY LIST(a)
SUBPARTITION BY RANGE(b)
(
PARTITION p1 VALUES(1) (SUBPARTITION sp1 VALUES LESS than(10), SUBPARTITION sp2 VALUES LESS than(20)),
PARTITION p2 VALUES(2) (SUBPARTITION sp3 VALUES LESS than(10), SUBPARTITION sp4 VALUES LESS than(20))
);
```

composite_hash_partitions

定义哈希分区的组合分区，每个哈希分区下可以定义范围、列表、哈希三种子分区的一种。

subpartition_by_hash

定义哈希子分区的分区列或子分区模板，也可以指定子分区分区数量。

hash_partition_desc

创建哈希分区及其子分区。

individual_hash_subparts

定义一个哈希子的表空间和segment是否立即创建的属性。

deferred_segment_creation

用于指定创建表对象时segment的创建方式，该语法不适用于临时表。

- IMMEDIATE：表示立即创建。
- DEFERRED：表示延迟创建。若省略，默认为延迟创建。

hash_subparts_by_quantity

使用hash分数批量定义哈希子分区。

该语句不适用于分布式部署。

示例（单机、共享集群部署）

```
-- hash-hash组合分区
CREATE TABLE hh_composite(a INT, b VARCHAR(10))
PARTITION BY HASH(a)
SUBPARTITION BY HASH(b)
subpartitions 8
(PARTITION p1, PARTITION p2);
-- hash-list组合分区
CREATE TABLE hl_composite(c1 INT, c2 VARCHAR(10))
PARTITION BY HASH(c1)
SUBPARTITION BY LIST(c2)
(
PARTITION p1(SUBPARTITION sp1 VALUES('a')),
PARTITION p2 (SUBPARTITION sp3 VALUES('d'), SUBPARTITION sp4 VALUES(DEFAULT))
);
```

```
-- hash-range组合分区
CREATE TABLE hr_composite(a INT, b INT, c INT,d INT)
PARTITION BY HASH(a)
SUBPARTITION BY RANGE(b,c,d)
(
PARTITION p1 (SUBPARTITION sp1 VALUES LESS than(10,20,30), SUBPARTITION sp2 VALUES LESS than(20,20,30)),
PARTITION p2
);
```

subpartition_template

定义子分区模板，支持RANGE、LIST及HASH三种类型。

执行本语句可为分区批量创建相同定义的子分区，避免冗杂操作。当组合分区表的某个分区没有定义子分区，则使用子分区模板为默认子分区定义。

执行本语句时须为子分区定义名称，否则返回错误。通过子分区模板创建的子分区名为`分区名_子分区名`，可通过DBA_TAB_SUBPARTITIONS视图查看所有子分区信息。

对于分布表，分区数等同于Chunk数。

示例（单机、共享集群部署）

```
-- 定义range子分区模板
CREATE TABLE hr_composite_template(a INT, b VARCHAR(10))
PARTITION BY HASH(a)
SUBPARTITION BY RANGE(b)
SUBPARTITION template (SUBPARTITION sub1 VALUES LESS than ('a') , SUBPARTITION sub2 VALUES LESS than (MAXVALUE))
(PARTITION p1,PARTITION p2);
-- 定义list子分区模板
CREATE TABLE list_composite(a INT, b VARCHAR(10))
PARTITION BY HASH(a) SUBPARTITION BY LIST(b)
SUBPARTITION template(SUBPARTITION sp1 VALUES('a'), SUBPARTITION sp2 VALUES(DEFAULT))
partitions 8;
-- 定义hash子分区模板
CREATE TABLE hash_composite(a INT, b VARCHAR(10))
PARTITION BY HASH(a)
SUBPARTITION BY HASH(b)
SUBPARTITION template (SUBPARTITION sp1 TABLESPACE yashan1, SUBPARTITION sp2 TABLESPACE yashan2)
(PARTITION p1, PARTITION p2);
--仅部分定义了子分区，其余通过子分区模板自行创建子分区
CREATE TABLE hl_partcomposite(c1 INT,c2 INT)
PARTITION BY HASH(c1)
SUBPARTITION BY LIST(c2)
SUBPARTITION template(SUBPARTITION sp9 VALUES(2))
(PARTITION p1 (SUBPARTITION sp11 VALUES(1)),
 PARTITION p2,
 PARTITION p3);
```

consistent_hash_partitions

该语句仅可用于为分布表创建分区，即表示建立分布表的数据分片（Chunk）规则，且分布表仅可使用该语句创建分区。

consistent

本关键字可省略，表示为分布表所创建分区必须指定为一致性哈希分区。

column

指定分区键对应的列，包含多个列时以`,`分隔。分布表的数据分片规则即为按此分区键对应的列或列组合进行哈希分配。

YashanDB对分区键存在如下约束要求：

- 列需符合类型要求，不能为LOB、JSON、RAW、ST_GEOMETRY、ROWID及UROWID类型。
- 如果表上定义了主键列，则分区键必须为主键列的其中一项或多项。
- 如果表上未定义主键列，则分区键可以为任意项。

individual_partition_clause

本语句用于逐个指定哈希分区，语法定义与单机部署中的individual_partition_clause类似，但不可以指定表空间属性，即在定义分布表的分区时，分区只能使用表所在的表空间。

使用本方法逐个指定分区时，需注意指定的分区数量必须与当前分布式部署环境中的Chunk总数量完全一致，否则返回错误。

hash_partitions_by_quantity
本语句用于按hash分数批量指定哈希分区，语法定义与单机部署中的hash_partitions_by_quantity类似，但不可以指定表空间属性，即在定义分布表的分区时，分区只能使用表所在的表空间。

hash_partition_quantity

表示hash分区的数量，可以指定为AUTO关键字或者任意数值。

如指定为AUTO关键字，系统将依据Chunk总数量自动生成hash分区，且hash分区的数量与Chunk总数量一致。

如指定为数值，当该数值不等于当前分布式部署环境中的Chunk总数量时，系统将依据Chunk总数量自动生成hash分区，且hash分区的数量与Chunk总数量一致。

> **Note**：
> Chunk总数量可由建库参数USERS_DATASPACE_SCALE_OUT_FACTOR * 当前DN组个数计算得到结果，其中，建库参数USERS_DATASPACE_SCALE_OUT_FACTOR在安装过程中配置且后续不可修改，可咨询数据库管理员获得该参数的值。

示例（分布式部署）

```
-- 重新创建销售信息表，且要求按年和月的组合哈希分配数据到各节点中(以Chunk总数量为21为例)
DROP TABLE IF EXISTS sales_info;
CREATE TABLE sales_info
(year CHAR(4) NOT NULL,
 month CHAR(2) NOT NULL,
 branch CHAR(4),
 product CHAR(5),
 quantity NUMBER DEFAULT 0 NOT NULL,
 amount NUMBER(10,2) DEFAULT 0 NOT NULL,
 salsperson CHAR(10))
    PARTITION BY CONSISTENT HASH (year,month)
(PARTITION p1, PARTITION p2, PARTITION p3, PARTITION p4, PARTITION p5, PARTITION p6, PARTITION p7,
PARTITION p8, PARTITION p9, PARTITION p10, PARTITION p11, PARTITION p12, PARTITION p13, PARTITION p14, PARTITION p15,
PARTITION p16, PARTITION p17, PARTITION p18, PARTITION p19, PARTITION p20, PARTITION p21);

-- 通过AUTO关键字指定hash分区数量
DROP TABLE IF EXISTS sales_info;
CREATE TABLE sales_info
(year CHAR(4) NOT NULL,
 month CHAR(2) NOT NULL,
 branch CHAR(4),
 product CHAR(5),
 quantity NUMBER DEFAULT 0 NOT NULL,
 amount NUMBER(10,2) DEFAULT 0 NOT NULL,
 salsperson CHAR(10))
PARTITION BY CONSISTENT HASH (year,month)
PARTITIONS AUTO;

-- 通过数值指定hash分区数量
DROP TABLE IF EXISTS sales_info;
CREATE TABLE sales_info
(year CHAR(4) NOT NULL,
 month CHAR(2) NOT NULL,
 branch CHAR(4),
 product CHAR(5),
 quantity NUMBER DEFAULT 0 NOT NULL,
 amount NUMBER(10,2) DEFAULT 0 NOT NULL,
 salsperson CHAR(10))
PARTITION BY CONSISTENT HASH (year,month)
PARTITIONS 7;

INSERT INTO sales_info VALUES ('2001','01','0201','11001',30,500,'0201010011');
INSERT INTO sales_info VALUES ('2015','11','0101','11001',20,300,'');
INSERT INTO sales_info VALUES ('2021','10','0101','11001',20,300,'');
INSERT INTO sales_info VALUES ('2000','12','0102','11001',20,300,'');
INSERT INTO sales_info VALUES ('2015','03','0102','11001',20,300,'');
```

```
INSERT INTO sales_info VALUES ('2021','05','0101','11001',40,600,'');
COMMIT;

-- 按分区检索sales_info数据
SELECT year,month,product,amount FROM sales_info PARTITION (SYS_P1);
YEAR  MONTH PRODUCT        AMOUNT
----- ----- --------- -----------
2000  12    11001             300
2015  03    11001             300
```

consistent_hash_with_subpartitions

该语句用于为分布表创建哈希分区的组合分区，且分布表仅可使用该语句创建组合分区，仅适用于分布式部署。

本语句语法同composite_hash_partitions描述，但不能为分区指定表空间属性。

示例（分布式部署）

```
--重建上例中的sales_info表，为其中一个分区创建range子分区
DROP TABLE IF EXISTS sales_info;
CREATE TABLE sales_info
(year CHAR(4) NOT NULL,
 month CHAR(2) NOT NULL,
 branch CHAR(4),
 product CHAR(5),
 quantity NUMBER DEFAULT 0 NOT NULL,
 amount NUMBER(10,2) DEFAULT 0 NOT NULL,
 salsperson CHAR(10))
PARTITION BY CONSISTENT HASH (year,month)
SUBPARTITION BY RANGE(quantity)
(
PARTITION p1 (SUBPARTITION sp1 VALUES LESS than(30), SUBPARTITION sp2 VALUES LESS than(60)),
PARTITION p2, PARTITION p3, PARTITION p4, PARTITION p5, PARTITION p6, PARTITION p7, PARTITION p8,
PARTITION p9, PARTITION p10, PARTITION p11, PARTITION p12, PARTITION p13, PARTITION p14, PARTITION p15, PARTITION p16,
PARTITION p17, PARTITION p18, PARTITION p19, PARTITION p20, PARTITION p21
);

INSERT INTO sales_info VALUES ('2001','01','0201','11001',10,500,'0201010011');
INSERT INTO sales_info VALUES ('2015','11','0101','11001',20,300,'');
INSERT INTO sales_info VALUES ('2021','10','0101','11001',30,300,'');
INSERT INTO sales_info VALUES ('2000','12','0102','11001',40,300,'');
INSERT INTO sales_info VALUES ('2015','03','0102','11001',50,300,'');
INSERT INTO sales_info VALUES ('2021','05','0101','11001',60,600,'');
COMMIT;

--按组合分区检索sales_info数据
SELECT year,month,product,quantity,amount
FROM sales_info SUBPARTITION (sp2);
YEAR  MONTH PRODUCT      QUANTITY      AMOUNT
----- ----- --------- ----------- -----------
2000  12    11001              40         300
2015  03    11001              50         300
```

lob_clause

该语句用于指定LOB/JSON类型列字段的存储属性，同时指定多列用`,`分隔。

- KEEP_DUPLICATES/DEDUPLICATE, CHUNK integer, NOCACHE/CACHE, LOGGING/NOLOGGING, NOCOMPRE/COMPRESS(LOW|MEDIUM|HIGH) 等LOB参数仅为语法兼容， tablespace space_name参数在分区LOB以及二级分区LOB的LOB子句中仅为语法兼容。
- chunk integer中integer取值范围为（0，32K]。
- hash分区与二级分区的LOB子句仅能指定TABLESPACE space_name参数。

basicfile|securefile

该语句用于语法兼容，无实际含义。

tablespace

指定LOB/JSON数据进行行外存储时的表空间，不指定该属性时LOB/JSON数据存储在表所在表空间中。

disable storage in row
指定对于LOB/JSON列字段，无论其大小，都执行行外存储，即为其创建新的存储空间，而不是与行中的其他列字段数据存储在一起。

enable storage in row
指定将LOB/JSON列字段的数据存放在行内空间，即与行中的其他列字段数据存储在一起。但当该数据大小超过一定的行内限制时，即使指定了ENABLE STORAGE IN ROW，仍进行行外存储。对于HEAP表，该限制是4000字节；对于TAC/LSC表，该限制是32000字节。

> **Note**：
> 数据在存储时会产生一些内部元信息，行存表字节限制包括这部分元数据所占空间，列存表则不包括。

示例（单机、共享集群部署）

```
--创建一张员工信息表，其中introduce为超长文本的LOB型字段
CREATE TABLE employees1(area CHAR(2),
branch CHAR(4),
employee_info JSON,
introduce CLOB)
LOB(introduce,employee_info) STORE AS (TABLESPACE yashan ENABLE STORAGE IN ROW);

--插入两行数据，其中第一行数据里的introduce因为数据长度小于3988字节，采用的是行内存储，而第二行数据则采用的是行外存储，json字段长度小于3988字节，采用的是行内存储。
INSERT INTO employees1 VALUES ('04','0401',
JSON('{"employee_no":"0401010008","employee_name":"Sam1","sex":"1","entire_date":"SYSDATE-3"}'),
'深圳计算科学研究院是深圳市人民政府2018年11月批准建设的十大基础研究机构之一');
INSERT INTO employees1 VALUES ('04','0401',
JSON('{"employee_no":"0401010008","employee_name":"Sam2","sex":"1","entire_date":"SYSDATE-3"}'),
LPAD('深圳',4000,'深圳'));
COMMIT;

SELECT JSON_FORMAT(employee_info), LENGTHB(introduce) FROM employees1;
JSON_FORMAT(EMPLOYEE                                                    LENGTHB(INTRODUCE)
------------------------------------------------------------------- --------------------
{"sex":"1","employee_no":"0401010008","entire_date":"SYSDATE-3","employee_name":"Sam1"}                  108
{"sex":"1","employee_no":"0401010008","entire_date":"SYSDATE-3","employee_name":"Sam2"}                12000
```

logging_clause

用于指定表的logging属性，默认值为logging，logging描述请参考ALTER TABLE的logging_clause部分。

nologging不适用于共享集群部署。

parallel_clause

用于指定后续查询创建的表对象时，默认的并行度，语法兼容。

cache_clause

语法兼容，无实际含义。

readonly_clause

语法兼容，无实际含义。

inmemory_clause

语法兼容，无实际含义。

table_compression

语法兼容，无实际含义。

nested_table_clause

该语句用于定义和创建嵌套表，YashanDB中的嵌套表被用于实现与含有Nested Table类型的UDT（用户自定义类型）列的存储，详见数据类型中关于用户自定义类型的描述，本语句不适用于共享集群部署。

nested_item

指定嵌套表对应的列字段，或列字段属性（该属性为Nested Table类型）。

column_value

当出现多层嵌套表（列字段为一个行成员包含Nested Table类型的Nested Table集合）时，对于里层的嵌套表使用COLUMN_VALUE关键字来表示nested_item。

local|global

当主表为分区表时，使用LOCAL关键字指定嵌套表也为与主表一一对应的分区，使用GLOBAL关键字指定嵌套表不分区，省略关键字时默认为LOCAL。

storage_table

指定嵌套表的名称，不可省略，且需符合YashanDB的对象命名规范。

对于多层嵌套表，在名称后接(nested_table_clause)定义里层的嵌套表信息。

tablespace

为嵌套表指定独立的表空间，省略时默认嵌套表存储在与主表相同的表空间。

示例（单机部署）

```
DROP TABLE IF EXISTS area_intro;
DROP TABLE IF EXISTS province_intro;
CREATE OR REPLACE TYPE tb_type AS TABLE OF CHAR(10);
/
CREATE OR REPLACE TYPE obj_type AS OBJECT(city_id INT, branch_list tb_type);
/

--创建单层嵌套表
CREATE TABLE province_intro (id INT, citys tb_type, city_intro obj_type)
NESTED TABLE citys STORE AS nt_province_citys TABLESPACE yashan
NESTED TABLE city_intro.branch_list STORE AS nt_province_branch;

--创建多层嵌套表
CREATE OR REPLACE TYPE tb_type_city AS TABLE OF tb_type;
/
CREATE TABLE area_intro(id INT, provinces tb_type_city)
NESTED TABLE provinces STORE AS nt_area_provinces(
    NESTED TABLE COLUMN_VALUE STORE AS nt_nt_province_citys);
```

lsc_table_properties

该语句用于定义LSC表的专有属性，这些属性只可以在创建LSC表时指定，否则报错。

compression_clause

定义LSC表的压缩属性，同列字段定义的压缩属性描述，系统按此设置对所有列数据进行压缩存储。

table_sort_clause

本语句用于定义LSC表的排序键，其中排序列可以是表中的任意列（可组合），不指定本语句时，默认以第一列作为排序键，且默认值为NULLS FIRST ASC。

YashanDB在存储LSC表数据时，会自适应的按照一定的排序粒度对存储进行排序。通过本语句所指定的排序键仅作为加速属性使用，即当对表执行查询、更新或删除等操作时，如过滤条件中包含已定义的排序键，系统将在存储层进行排序加速计算，从而获得更优的查询性能。

column_name

本语句用于指定表的排序列，排序列不允许为如下类型：

- CLOB
- BLOB
- NCLOB
- BIT
- ROWID
- CURSOR
- JSON
- UDT

nulls (first|last)
本语句用于指定空值的排列位置，NULLS FIRST表示空值排列在最前，NULLS LAST表示空值排列在最后。当未指定此语句时，对升序排列缺省为NULLS LAST，对降序排列缺省为NULLS FIRST。

asc|desc
本语句用于指定排序方式，ASC表示升序，DESC表示降序，省略则默认为升序。

scol
默认LSC表可变数据（MCOL）也会使用排序键，如可变数据（MCOL）不需要点查加速，在创建时可指定仅针对稳态数据（SCOL）使用排序键，也可以通过ALTER TABLE语句开启或关闭可变数据（MCOL）的排序键。

> **Note**：
> 可变数据的排序键会降低导入性能，生产环境中的存量数据建议先导入然后再打开可变数据的排序键，或直接通过Bulkload模式导入成稳态数据。

示例（LSC表）

```
--创建LSC表，压缩且按指定列排序存储
DROP TABLE IF EXISTS finance_info;
CREATE TABLE finance_info
(year CHAR(4) NOT NULL,
month CHAR(2) NOT NULL,
branch CHAR(4),
revenue_total NUMBER(10,2),
cost_total NUMBER(10,2),
fee_total NUMBER(10,2)
)
COMPRESSION lz4 HIGH
ORDER BY (year,month,branch);
```

mcol_ttl_clause

该语句用于定义LSC表的可变数据最大生命周期。

在YashanDB中，对LSC表插入的数据将先作为可变数据（MCOL）存储，满足一定条件后触发后台线程将可变数据转为稳态数据（SCOL，不可变数据）进入数据桶（bucket，对象存储目录）。

此值保证用户插入数据在可变数据区内至少保留1/2 MCOL TTL时间，最多则保留MCOL TTL时间。

指定了可变数据最大生命周期后，系统将把该值和数据量一起作为触发转换线程的条件，让用户可以根据自身业务情况设定更合理的转换时间。

timestamp

生命周期数值，该值须遵循INTERVAL YEAR TO MONTH或INTERVAL DAY TO SECOND数据类型的格式表述，例如`'1' YEAR(9)`、`'30:59.9' MINUTE TO SECOND(6)`，具体请参考日期时间型中这两种类型的描述。

示例（LSC表）

```
--创建可变数据生命周期为1个月的LSC表，即在满1个月时，MCOL数据将被转换为SCOL数据
DROP TABLE IF EXISTS finance_info;
CREATE TABLE finance_info
(year CHAR(4) NOT NULL,
month CHAR(2) NOT NULL,
branch CHAR(4),
revenue_total NUMBER(10,2),
cost_total NUMBER(10,2),
```

```
fee_total NUMBER(10,2)
)
COMPRESSION lz4 HIGH
ORDER BY (year,month,branch)
MCOL TTL '1' MONTH;
```

transformer_clause

LSC表默认拥有转换、合并和生成AC数据的后台数据转换能力。

在某些情况下，如果希望建表时就设置某些后台数据转换能力，可以使用此语法进行关闭或打开。建表完成后也可再使用ALTER TABLE语句对后台数据转换能力进行开启或关闭。

示例（LSC表）

```
--创建一个关闭合并能力的LSC表
DROP TABLE IF EXISTS lsc_compact_disable;
CREATE TABLE lsc_compact_disable(x INT) ORGANIZATION LSC DISABLE compact;
```

row_movement_clause

该语句用于指定创建的表中的行数据是否可以物理移动，HEAP表和TAC表省略则默认为DISABLE ROW MOVEMENT，LSC表省略则默认为ENABLE ROW MOVEMENT。

在YashanDB中，默认行的Rowid是不变化的，但在某些特殊场景，例如对分区表的分区键值进行更新操作，导致其所在行需要从当前分区迁移到另一个分区中，行的Rowid将发生变更，这种变动在表ENABLE ROW MOVEMENT时是被允许的，且YashanDB在此情况下会保证事务的强一致性，避免出现漏更新（lost update）。

开启row movement后，并发事务中可能会触发语句重启（restart statement，回滚之前的操作并重新执行DML事务），影响并发性能 ，因此在正常情况下不建议开启此开关，在可预知会发生行迁移的业务场景中（例如闪回数据），可通过ALTER TABLE语句临时打开，并在结束后关闭。

分布式部署中仅有分区表跨分区更新或LSC表中稳态数据更新时需要ENABLE ROW MOVEMENT，跨分区更新场景中仅DN节点内进行跨分区更新，无法更新分区键。

示例（单机、共享集群部署）

```
--创建一张ENABLE ROW MOVEMENT的分区表
DROP TABLE IF EXISTS orders_info_rowmove;
CREATE TABLE orders_info_rowmove (order_no CHAR(14) NOT NULL,
order_desc VARCHAR2(100),
area CHAR(2),
branch CHAR(4),
order_date DATE DEFAULT SYSDATE NOT NULL,
salesperson CHAR(10),
id NUMBER)
PARTITION BY RANGE (id )
(PARTITION p1 VALUES LESS THAN (800),
PARTITION p2 VALUES LESS THAN (1800),
PARTITION p3 VALUES LESS THAN (2800))
ENABLE ROW MOVEMENT;
```

CREATE TABLESPACE SET

通用描述

CREATE TABLESPACE SET语句用于在分布式部署模式下创建一个新的表空间集。

表空间集是YashanDB分布式部署中的一个逻辑存储单位，用于存储分布表及与分布表相关数据信息，物理上对应了各DN节点上的数据文件。

> **Note**：
> 本文出现的所有CHUNK_NUM，均表示当前分布式部署环境中的Chunk总数量，该值可由建库参数USERS_DATASPACE_SCALE_OUT_FACTOR * 当前DN组个数计算得到结果，其中，建库参数USERS_DATASPACE_SCALE_OUT_FACTOR在安装过程中配置且后续不可修改，可咨询数据库管理员获得该参数的值。

语句定义

create tablespace set::=

```
syntax::= CREATE TABLESPACE SET tablespace_set_name (([databucket_clause] ON dataspace_name (MAXSIZE size_clause [NEXT
size_clause] [SIZE size_clause]))|(ON dataspace_name MEMORY MAPPED SIZE size_clause ))
```

databucket_clause**::=**

```
syntax::= DATABUCKET ((bucket_clause) {"," (bucket_clause)})
```

bucket_clause**::=**

```
syntax::= "'bucket_name'" [MAXSIZE size_clause]
```

tablespace_set_name

该语句用于指定要创建的表空间集的名称，不可省略，且需符合YashanDB的对象命名规范。

databucket_clause

指定对象存储目录的路径信息，指定多个路径用`,`分隔，每一个路径表示创建一个bucket（数据桶），可省略，省略则挂载至系统默认生成的bucket中，如不省略则不会默认生成bucket。

可通过查询DV$DATABUCKET视图查看当前系统中所有创建的bucket信息。

YashanDB中单个表空间集下允许挂载DataBucket数量的最大值为64，数据库中允许挂载Databucket数量的最大值取决于建库参数，可查看CREATE DATABASE章节中maxdatabuckets语句描述。

指定本语句后，不能再同时指定MEMORY MAPPED关键字。

bucket_clause

该语句用于指定bucket信息，如路径，大小等。

YashanDB支持为表空间集创建本地存储bucket，本地存储bucket将数据存储至本地路径中。

bucket_name

-对于本地存储的bucket，bucket_name为bucket在本地文件系统的相对路径，且系统将对指定的路径进行以下有效性判断：

- 允许指定一个不存在的目录，执行本语句将创建该目录。
- 允许指定一个已存在的目录，但该目录不能为已被其他bucket使用的路径。
- 系统对指定的目录必须拥有写权限。
- 分布式部署中不允许指定绝对路径。
- 对于S3 bucket，bucket_name为逻辑名称，用于在数据库内查询具体的bucket信息。

新创建的bucket名称不可与系统内置bucket重复，否则返回错误，可通过查看`DV$DATABUCKET`视图查看当前系统中所有的bucket信息。

示例（分布式部署）

```
--如下语句将新建tbs_tb表空间集，同时分别于dn的$YASDB_DATA/local_fs和dbfiles目录下新建lsc_bucket1/lsc_bucket2这两个自定义的数据桶和chunk数个名称为TSS_OID_CHUNK_CHUNKID_FILE_ID的默认数据文件；
--且分别于mn/cn的$YASDB_DATA/local_fs和dbfiles目录下默认新建名称为TSS_OID_ROOT_DATABUCKET的根数据桶和名称为TSS_OID_ROOT_DATAFILE的根数据文件
CREATE TABLESPACE SET tbs_tb DATABUCKET '?/local_fs/lsc_bucket1','?/local_fs/lsc_bucket2' ON USERS MAXSIZE 8G;
```

s3_bucket_clause

该语句用于指定S3 bucket的访问信息。

> **Note**：
> 该功能默认无法使用，如需使用请联系我们的技术支持处理。

url
用于指定访问地址，不可省略，最大值为1023字节。

region
用于指定S3 bucket的物理所属区域，可省略，省略则默认使用对象存储服务的默认配置，最大值为127字节。

access key
用于标识客户端身份，不可省略，最大值为127字节。

secret key
访问密钥，不可省略，最大值为127字节。

maxsize size_clause

用于指定bucket的大小，可省略，省略则默认为UNLIMITED。

示例（分布式部署）

```
--如下语句将新建s3_tablespace表空间并挂载S3 bucket
CREATE TABLESPACE SET s3_tablespace DATABUCKET 's3_bucket3' S3(URL '192.168.0.1:8000/s3bucket',ACCESS key 'ak',secret key 'sk') ON USERS MAXSIZE 8G;
```

dataspace_name

该语句用于指定表空间集所属的数据空间，不可省略，目前仅支持的dataspace为users。

memory mapped

当指定MEMORY MAPPED关键字时，表示所创建表空间集的文件的所有页面都将映射在内存中，YashanDB将此种类型的表空间集命名为内存映射表空间集。

内存映射表空间集支持在数据库启动时采用预加载方式，将表空间集中的数据全部加载到内存中，提高访问性能。

内存映射表空间集创建后无法修改属性。

size size_clause

该语句用于指定内存映射表空间集预分配的大小，不可省略。

size_clause的取值范围为[128 * DB_BLOCK_SIZE * CHUNK_NUM, 4GB * DB_BLOCK_SIZE * CHUNK_NUM]。

示例（分布式部署）

```
CREATE TABLESPACE SET mm_tss
ON USERS
MEMORY MAPPED SIZE 8G;
```

maxsize size_clause

该语句用于指定表空间集的最大可扩展空间。该参数对于非内存映射表空间集不可省略，且不能小于预分配的空间大小。该参数不用于内存映射表空间集。

size_clause的取值范围为[128 * DB_BLOCK_SIZE * CHUNK_NUM, 4G * DB_BLOCK_SIZE * CHUNK_NUM]，当DB_BLOCK_SIZE参数为默认的8K值时，该范围为[1M * CHUNK_NUM, 32T * CHUNK_NUM]。

next size_clause

该语句用于指定表空间集包含的数据文件每次自动扩展的大小。

size_clause默认为8192 * DB_BLOCK_SIZE，用户设置时取值范围为[512 * DB_BLOCK_SIZE, 32768 * DB_BLOCK_SIZE]，当DB_BLOCK_SIZE参数为默认的8K值时，该范围为[4M,256M]。

size size_clause

该语句用于指定表空间集预分配的大小，可省略，省略则默认按照数据文件及bucket大小进行预分配。

size_clause的取值范围为[128 * DB_BLOCK_SIZE * CHUNK_NUM, MAXSIZE]，MAXSIZE表示当前表空间集的最大可扩展空间，当DB_BLOCK_SIZE参数为默认的8K时，最小值为1M * CHUNK_NUM。

示例（分布式部署）

```
CREATE TABLESPACE SET tss1
ON USERS
MAXSIZE 8G SIZE 7G;
```

CREATE TABLESPACE

通用描述

CREATE TABLESPACE用于在数据库中创建一个新的表空间。

SYSTEM/SYSAUX/UNDO表空间在CREATE DATABASE时自动创建，分布式部署中的SWAP表空间和TEMP临时表空间也在CREATE DATABASE时自动创建，不能用CREATE TABLESPACE语句创建。

单机/分布式部署中，可同时指定挂载到该表空间的数据文件（不指定时系统自动创建一个数据文件并挂载）和数据桶（bucket，对象存储目录，包含slice文件，不指定时系统不会自动创建）。在创建LSC表所属表空间时，必须同时进行数据桶挂载，否则该表空间下将无法创建LSC表，除非通过ALTER TABLESPACE语句为该表空间进行bucket挂载。

分布式部署中创建新表空间出现节点故障时，恢复措施见用户表空间管理章节描述。

语句定义

create tablespace::=

```
syntax::= CREATE (permanent_tablespace_clause | temporary_tablespace_clause | swap_tablespace_clause)
```

permanent_tablespace_clause**::=**

```
syntax::= TABLESPACE tablespace_name [datafile_clause] [extent_management_clause] [MEMORY MAPPED] [databucket_clause]
[encryption_clause] [compress_clause]
```

temporary_tablespace_clause**::=**

```
syntax::= (TEMPORARY TABLESPACE | LOCAL TEMPORARY TABLESPACE FOR (ALL | LEAF)) tablespace_name [datafile_clause]
[extent_management_clause]
```

swap_tablespace_clause**::=**

```
syntax::= (SWAP TABLESPACE | LOCAL SWAP TABLESPACE) tablespace_name [datafile_clause]
```

datafile_clause**::=**

```
syntax::= ((DATAFILE | TEMPFILE) (file_specification) {"," (file_specification)})
```

file_specification**::=**

```
syntax::= "file_name" SIZE size_clause [AUTOEXTEND (OFF|(ON [NEXT size_clause] [MAXSIZE (UNLIMITED|size_clause)]))] [PARALLEL
parallel]
```

extent_management_clause**::=**

```
syntax::= EXTENT (AUTOALLOCATE|UNIFORM SIZE size_clause)
```

databucket_clause**::=**

```
syntax::= DATABUCKET ((bucket_clause) {"," (bucket_clause)})
```

bucket_clause**::=**

```
syntax::= "bucket_name" [s3_bucket_clause] [MAXSIZE size_clause]
```

s3_bucket_clause::=

```
syntax::= S3 "(" URL "'url'" ["," REGION "region"] "," ACCESS KEY "access_key" "," SECRET KEY "secret_key" ")"
```

encryption_clause::=

```
syntax::= ENCRYPTION ENCRYPT
```

compress_clause::=

```
syntax::= COMPRESS [LZ4|ZSTD]
```

permanent_tablespace_clause

该语句用于在数据库中创建一个新的表空间，表空间中的对象存储在数据文件或者数据桶中。

temporary_tablespace_clause

该语句用于指定创建临时表空间，其中包含的数据仅在用户会话期间持续，临时表空间中的对象存储在临时文件中。 该语句有两种创建临时表空间的方式，包括创建TEMP临时表空间以及创建本地临时表空间。其中有如下规则：

- 在单机部署中，只能创建TEMP临时表空间，不能创建本地临时表空间。
- 在共享集群部署中，TEMP临时表空间的临时文件仅支持YFS路径，而本地临时表空间的临时文件支持YFS与本地磁盘路径，但不允许同一个表空间同时存在YFS路径与本地磁盘路径。

swap_tablespace_clause

该语句用于指定创建SWAP表空间，其中包含的数据仅在用户会话期间持续，SWAP表空间中的对象存储在临时文件中。 该语句有两种创建SWAP表空间的方式，包括创建SWAP表空间以及创建本地SWAP表空间。其中有如下规则：

- 在单机部署中，只能创建SWAP表空间，不能创建本地SWAP表空间。
- 在共享集群部署中，SWAP表空间的临时文件仅支持YFS路径，而本地SWAP表空间的临时文件支持YFS与本地磁盘路径，但不允许同一个表空间同时存在YFS路径与本地磁盘路径。

tablespace_name

该语句用于指定要创建的表空间的名称，不可省略，且需符合YashanDB的对象命名规范。

分布式部署需注意创建的表空间的名称不可与已有的表空间集中分配的表空间名称相同，否则会返回错误。

示例

```
--使用所有的默认选项创建一个表空间
CREATE TABLESPACE yashan;
```

datafile_clause

该语句用于指定创建表空间对应的数据文件，可省略，则系统按如下规则自动创建一个数据文件：

- 文件名称由表空间名称以及数据文件在表空间内的序号组合生成，如：tablespace_name1，tablespace_name2...，且统一转换为大写。
- 文件的默认大小为8192个BLOCK，文件路径为系统默认的数据文件路径。
- 对于非MEMORY MAPPED表空间，默认文件开启自动扩展，next为8192个块，maxsize为64MB个块。
- 如果没有显式的规定extent分配方式，extent的默认分配方式为系统自动分配。
- 只能使用TEMPFILE关键字创建临时表空间或SWAP表空间。
- 对于本地临时表空间以及本地SWAP表空间而言，添加一个文件等于添加一组文件，每一组文件数量的个数等于集群实例个数。

file_specification

指定多个数据文件用`，`分隔，可省略，则系统将在默认的数据文件路径下创建一个以表空间名称（转换为大写）命名，大小为8192个块，开启自动扩展，每次扩展8192个块，最大可扩展到64MB个块的数据文件。 可以通过PARALELL指定创建数据文件的并行度，取值范围为1到8。 如果未指定并行度数据库会根据文件大小自适应并行度，文件不超过1G时并行度为1，文件超过128G并行度为8，文件大小在1G到128G之间时并行度为4。

file_name
指定数据文件的名称，如名称里未包含路径，则取系统默认的数据文件路径。

分布式部署中数据文件路径仅允许使用相对路径。

共享集群中的数据文件路径必须为YFS路径。

共享集群中创建数据文件不指定完整YFS路径时默认存储在+DG0/dbfiles中。

size
指定数据文件的大小，遵循通用size_clause定义。

单个数据文件最少具有128个块用于存储元信息。

UNDO表空间内的单个数据文件最多具有8MB个块，非UNDO表空间内的单个数据文件最多具有64MB个块。

数据文件的大小等于数据库块大小乘以数据文件块个数，例如当数据库块大小为8K时非UNDO表空间的单个数据文件最小值为1M，最大值为512G。

autoextend on|autoextend off
对创建的数据文件打开|关闭自动扩展，该语句省略时默认为关闭自动扩展。当打开自动扩展时：

- NEXT size_clause：数据文件每次自动扩展时的大小由此值指定，默认为8192个块。
- MAXSIZE UNLIMITED/size_clause：数据文件可扩展到的最大数由此值决定，默认为64MB个块。UNLIMITED表示不限制最大值。

不能对MMS表空间指定AUTOEXTEND ON。

parallel
当创建一个较大的数据文件时，可以通过本语句指定并行度， 提高创建大文件的速度 。不指定本语句时，系统根据文件大小自动选取并行度，例如文件不超过1G时的并行度为1，文件超过128G时的并行度为8，文件大小在1G到128G之间时的并行度为4。

parallel的值应该介于1~8之间。

示例（单机/分布式部署）

```
--普通路径
CREATE TABLESPACE yashan1 DATAFILE 'yashan1' SIZE 4M AUTOEXTEND ON NEXT 4M MAXSIZE 1G PARALLEL 2;
```

示例（共享集群部署）

```
--YFS路径
CREATE TABLESPACE yashan1 DATAFILE '+DG0/yashan1' SIZE 4M AUTOEXTEND ON NEXT 4M MAXSIZE 1G PARALLEL 2;
```

extent_management_clause

该语句用于指定表空间里的对象申请extent时的空间分配方式，AUTOALLOCATE分配方式根据对象当前大小由系统自动分配extent空间，UNIFORM分配方式每次为对象分配的extent空间为固定值。

- 创建表空间后不能修改extent分配方式。
- 创建临时表空间若不指定extent分配方式，默认使用UNIFORM分配方式，大小为8个BLOCK。
- 创建SWAP表空间不可指定extent分配方式，默认使用UNIFORM分配方式，大小为8个BLOCK。
- 创建非临时表空间若不指定extent分配方式，默认使用AUTOALLOCATE分配方式。
- 使用UNIFORM分配方式的表空间内的每个数据文件大小都必须大于UNIFORM SIZE。

示例

```
CREATE TABLESPACE yashan2 DATAFILE 'yashan2' SIZE 4M AUTOEXTEND ON NEXT 4M MAXSIZE 1G EXTENT UNIFORM SIZE 64K;
```

memory mapped

当指定MEMORY MAPPED关键字时，表示所创建表空间的文件的所有页面都将映射在内存中 （AIM：all in memory），YashanDB将此种类型的表空间命名为内存映射表空间（MMS：Memory mapped space）。共享集群部署下不允许创建MMS。

MMS表空间支持在数据库启动时采用预加载方式，将表空间数据全部加载到内存中，提高访问性能，具体的预加载策略通过MMS_DATA_LOADERS参数配置。

通过V$TABLESPACE视图的MEMORY_MAPPED字段可查询某个表空间是否为MMS表空间。

分布式部署中，指定了MEMORY MAPPED关键字后，不能再同时指定databucket_clause。

示例（单机、分布式部署）

```
CREATE TABLESPACE yashan3 DATAFILE 'yashan3' SIZE 4M EXTENT UNIFORM SIZE 64K MEMORY MAPPED;
CREATE TABLESPACE yashan4 DATAFILE 'yashan4' SIZE 4M AUTOEXTEND ON NEXT 4M MAXSIZE 1G PARALLEL 6 EXTENT UNIFORM SIZE 64K;
```

databucket_clause

指定对象存储目录的路径信息，指定多个路径用`,`分隔，每一个路径表示创建一个bucket（数据桶），可省略，省略则默认创建的表空间不挂载bucket。

可通过查询V$DATABUCKET视图查看当前系统中所有创建的bucket信息。

YashanDB中单个表空间下允许挂载DataBucket数量的最大值为64，数据库中允许挂载Databucket数量的最大值取决于建库参数，可查看CREATE DATABASE章节中maxdatabuckets语句描述。

共享集群部署中不允许指定databucket_clause。

bucket_clause

该语句用于指定bucket信息，如路径，大小等。

YashanDB支持创建本地存储bucket及S3（Simple Storage Service） bucket，本地存储bucket将数据存储至本地路径中，S3 bucket为面向互联网的云存储。

分布式部署中不允许创建S3 bucket。

bucket_name

对于不同的bucket类型，bucket_name具有不同的含义：

- 对于本地存储的bucket，bucket_name为bucket在本地文件系统的绝对路径或相对路径（分布式部署中不允许指定绝对路径），且系统将对指定的路径进行以下有效性判断：
  - 允许指定一个不存在的目录，执行本语句将创建该目录。
  - 允许指定一个已存在的目录，但该目录不能为已被其他bucket使用的路径。
  - 系统对指定的目录必须拥有写权限。
- 对于S3 bucket，bucket_name为逻辑名称，必须由字母、数字或下划线组成，用于在数据库内查询具体的bucket信息。

示例（单机、分布式部署）

```
--如下语句将新建lsc_tb表空间，同时于$YASDB_DATA/local_fs目录下新建lscfile1/lscfile2两个bucket，同时也将默认新建一个名称为LSC_TB0的数据文件
CREATE TABLESPACE lsc_tb DATABUCKET '?/local_fs/lscfile1','?/local_fs/lscfile2';
```

s3_bucket_clause

该语句用于指定S3 bucket的访问信息。

> **Note**：
> 该功能默认无法使用，如需使用请联系我们的技术支持处理。

url

用于指定访问地址，不可省略，最大值为1023字节。

region

用于指定S3 bucket的物理所属区域，可省略，省略则默认使用对象存储服务的默认配置，最大值为127字节。

access key
用于标识客户端身份，不可省略，最大值为127字节。

secret key
访问密钥，不可省略，最大值为127字节。

maxsize size_clause
用于指定bucket的大小，可省略，省略则默认为UNLIMITED。

示例（单机部署）

```
--如下语句将新建s3_tablespace表空间并挂载S3 bucket
CREATE TABLESPACE s3_tablespace DATABUCKET 's3_bucket3' S3(URL '192.168.0.1:8000/s3bucket',ACCESS key 'ak',secret key 'sk')
MAXSIZE 8G;
```

encryption_clause

该语句用于指定表空间加密属性，以ENCRYPTION开头。本语句可省略，表示创建的表空间为非加密表空间。 指定为ENCRYPT时，表示创建的表空间为加密表空间，即该表空间对应存储介质上的数据将被加密。YashanDB对加密表空间的约束规则如下：

- 不能将内置表空间指定为加密表空间。
- 不能将临时属性表空间指定为加密表空间。
- 对于加密表空间的表对象，在其上创建的索引和AC也必须位于某个加密表空间。
- 加密属性在创建表空间时指定，后续不可更改。

对于分区表，系统并不限制各分区所在的表空间加密属性是否一致，在更新操作引发的数据分区移动时，该数据将依据新分区的加密属性按明文或密文存储。

示例

```
-- 如下语句将新建表空间，并指定为加密表空间
CREATE TABLESPACE yashan5 ENCRYPTION ENCRYPT;
```

compress_clause

该语句用于指定表空间压缩属性，以COMPRESS开头，压缩算法仅支持LZ4和ZSTD，不指定压缩算法默认是LZ4。本语句可省略，表示创建的表空间为非压缩表空间。 指定为COMPRESS时，表示创建的表空间为压缩表空间，即该表空间对应存储介质上的数据将被压缩。YashanDB对压缩表空间的约束规则如下：

- 不能将内置表空间指定为压缩表空间。
- 压缩属性在创建表空间时指定，后续不可更改。
- 如果表空间的数据文件所在文件系统不支持punch hole或者其页面大小不是1024、2048或4096，则无法创建压缩表空间。
- 部分文件系统不支持压缩表空间，常见支持的文件系统有：XFS、ext4、Btrfs、tmpfs(5)、gfs2(5)等。

共享集群部署中，创建本地临时表空间或者本地swap表空间本地磁盘文件时可以指定compress_clause，其他创建表空间操作不允许指定compress_clause。

示例（单机、分布式部署）

```
-- 如下语句将新建yashan1表空间，并指定为压缩表空间
CREATE TABLESPACE yashan6 COMPRESS;
CREATE TABLESPACE yashan7 COMPRESS lz4;
CREATE TABLESPACE yashan8 COMPRESS zstd;
```

CREATE TRIGGER

通用描述

CREATE TRIGGER用于创建或替换一个数据库触发器，系统会在指定的条件发生时自动运行相关触发器。

用户需要拥有CREATE TRIGGER系统权限，才能在自己模式下，对自己模式中的对象创建触发器。

用户需要拥有CREATE ANY TRIGGER系统权限，才能在任何模式下，对任何模式中的对象创建触发器。

触发器为PL对象，对其的创建需采用过程体语言实现，且需满足YashanDB所实现PL（包括变量、参数、语句等）的一切规范和约束，具体描述请查看PL参考手册。

关于CREATE TRIGGER的语法描述详见PL参考手册的触发器。

CREATE TYPE BODY

通用描述

CREATE TYPE BODY用于创建一个用户自定义类型的主体，是对Object类型的UDT在CREATE TYPE中声明的方法的具体实现。

类型主体中的方法实现和类型中的方法声明必须是一一对应的。

对类型主体的创建需采用过程体语言实现，且需满足YashanDB所实现PL（包括变量、参数、语句等）的一切规范和约束，具体描述请查看PL参考手册。

关于CREATE TYPE BODY的语法描述详见PL参考手册的自定义类型。

CREATE TYPE

通用描述

CREATE TYPE用于创建或替换一个数据库自定义类型（UDT），相对于数据库内置的数据类型而言，UDT是由用户自定义的数据类型。

在YashanDB中，用户可以定义如下类型的UDT：

- Object：抽象数据类型（Abstract Data Type），该类UDT由属性和方法组成，创建此类UDT必须同时使用CREATE TYPE BODY语句构建方法的具体实现。
- Varray：数组类型，该类UDT为一组相同数据类型的集合。

用户需要拥有CREATE TYPE系统权限，才能在自己的模式下创建自定义类型。

用户需要拥有CREATE ANY TYPE系统权限，才能在任何模式下创建自定义类型。

对自定义类型的创建需采用过程体语言实现，且需满足YashanDB所实现PL（包括变量、参数、语句等）的一切规范和约束，具体描述请查看PL参考手册。

关于CREATE TYPE的语法描述详见PL参考手册的自定义类型。

CREATE USER

通用描述

CREATE USER用于新建一个数据库用户。系统默认创建的是普通用户，YashanDB的用户管理体系请参考产品安全手册用户管理。

语句定义

create user::=

```
syntax::= CREATE USER user_name
[
IDENTIFIED BY [VALUES] password
| DEFAULT TABLESPACE tablespace
| PROFILE profilename
| ACCOUNT (LOCK|UNLOCK)
]
{" "
[
IDENTIFIED BY [VALUES] password
| DEFAULT TABLESPACE tablespace
| TEMPORARY TABLESPACE tablespace
| LOCAL TEMPORARY TABLESPACE tablespace
| PROFILE profilename
| ACCOUNT (LOCK|UNLOCK)
]}
```

user_name

该语句用于指定创建的用户的名称，不可省略，且需符合YashanDB的对象命名规范。

values

该语句用于指定按密文创建用户的密码，可省略，则按明文创建密码。

YashanDB的密码策略为加密传输和加密存储，即服务端存储的是密文密码。在某些特定场景（例如数据库迁移），可通过直接指定密文来创建密码，该操作不影响用户前端输入明文密码登录。

identified by password

该语句用于指定创建的用户的密码，可省略，省略则默认无密码。

该语句需满足如下规则：

- 不可指定为NULL或''。
- 可以为数字、字母和特殊符号的组合。
- 密码字符不可以为双引号。
- 当包含 `%` ，`\` ，`;` ，`@` ，`/` 等特殊符号时，密码字符串前后需加双引号。
- 区分大小写。
- 密码字符串长度不超过64。
- 遵循安全手册中所制定的密码策略要求。

> **Note**：
> 无密码的用户无法连接至YashanDB中，创建新用户时建议执行本语句为该用户指定密码。

当指定VALUES时，password内容为密文，不遵循上述规则，但需符合如下基本格式要求，否则创建用户失败且提示YAS-04289错误：

- 必须以 `S:` 开头。
- 整个长度必须是86字节。
- `S:` 后只能是数字和大写的字母。

@和/的特殊处理

YashanDB允许在密码中包含 `@` 和 `/` 等特殊符号，在输入密码进行登录时，包含 `@` 或 `/` 的密码字符串前后也必须使用双引号标识。

此外需注意的是，在Linux OS命令行模式中，双引号为特殊含义字符，对其的使用需用\进行转义或由单引号包裹。

示例

```
--创建sales1和sales2两个用户，其中sales2用户密码包含了@和/
CREATE USER sales1 IDENTIFIED BY "1%2";
CREATE USER sales2 IDENTIFIED BY "1@2/";

--需至少授予用户CREATE SESSION权限，以使用户能连接数据库
GRANT CREATE SESSION TO sales1;
GRANT CREATE SESSION TO sales2;

--在SQL客户端连接时应输入的密码格式
conn sales1/1%2
conn sales2/"1@2/"

--在命令行连接时应输入的密码格式
$ yasql sales1/1%2
$ yasql sales2/\"1@2/\"
```

default tablespace

该语句用于指定创建的用户的默认表空间，可省略，则缺省为数据库创建时的DEFAULT表空间。

示例

```
CREATE USER sales3 IDENTIFIED BY 123 DEFAULT TABLESPACE users;
```

temporary tablespace

该语句用于指定创建的用户的默认TEMP表空间，可省略，则缺省为数据库创建时的TEMP表空间。

分布式部署中无法使用此功能。

示例（单机、共享集群部署）

```
CREATE USER sales3 IDENTIFIED BY 123 TEMPORARY TABLESPACE temp;
```

local temporary tablespace

该语句用于指定创建的用户的默认本地TEMP表空间，可省略，缺省值为空，此时用户创建临时表默认使用TEMP表空间。

如同时指定本地TEMP表空间与TEMP表空间，优先使用本地TEMP表空间。

分布式部署/单机部署中无法使用此功能。

示例（共享集群部署）

```
-- 创建本地临时表空间local_temp
CREATE LOCAL TEMPORARY TABLESPACE FOR ALL local_temp TEMPFILE '?/file1' size 2M;
CREATE USER sales3 IDENTIFIED BY 123 LOCAL TEMPORARY TABLESPACE local_temp;
```

account (lock|unlock)

该语句用于指定在创建用户时是否锁定该用户，若锁定则该用户将不可登录。

本语句可省略，若省略则默认为不锁定。

示例

```
CREATE USER sale4 IDENTIFIED BY 123 ACCOUNT LOCK;
```

profile

该语句用于为创建的用户指定对应的profile，profile见CREATE PROFILE中描述。

本语句可省略，则缺省对应系统默认的profile。

示例

```
CREATE USER sales5 IDENTIFIED BY 123 PROFILE DEFAULT;
```

CREATE VIEW

通用描述

CREATE VIEW用于创建一个视图对象。创建的视图可以被SELECT，但不能被INSERT/UPDAET/DELETE。

本语句可能包含对某个表的DML操作，则需要拥有对该表的相应DML权限才能成功创建视图，且同时遵循相应DML的一切约束。

语句定义

create view::=

```
syntax::= CREATE [OR REPLACE] [FORCE|NO FORCE] [EDITIONABLE|NONEDITIONABLE] VIEW [schema "."] view_name ["(" (alias) {","
(alias)} ")"] AS subquery [subquery_restriction_clause]
```

subquery_restriction_clause::=

```
syntax::= WITH (READ ONLY | CHECK OPTION) [CONSTRAINT constraint]
```

or replace

该语句用于指定当要创建的视图已经存在时，将其进行重建。

force|no force

指定FORCE表示强行创建视图。默认为NO FORCE，表示条件不满足时返回报错，而不是强行创建视图。

当视图依赖的子查询内对象不存在，或者视图的owner对子查询内依赖的对象无访问权限时，指定FORCE可以强行创建视图，但实际使用该视图可能会报错。

使用FORCE的好处是，在所依赖对象被创建，并且视图owner有依赖对象的访问权限后，无需重新创建即可以成功查询该视图。

示例

```
CREATE FORCE VIEW v_table_not_exsits AS SELECT * FROM table_not_exsits;
```

editionable | noneditionable

用于语法兼容，无实际含义。

view_name

该语句用于指定要创建的视图的名称，不可省略，且需符合YashanDB的对象命名规范。

alias

该语句用于指定视图列字段的别名，可省略，且需符合YashanDB的对象命名规范，不可用于更改列字段的数据类型。

as subquery

该语句用于指定创建视图的子查询语句，子查询规则同SELECT语句中描述。其中，子查询的列项不能指定为某个序列号。

如在子查询中包含了对某个表的SELECT *操作时：

- 创建的视图将该表的所有列字段作为自己的列字段。
- 如该表增加了新的列字段，则本视图不会体现该变化，仍只包含原来的列字段项。

示例

```
CREATE VIEW v_area AS SELECT * FROM area;
```

视图失效

当视图用到的子对象发生以下变化时，视图失效，其status变为invalid。

- 基表删除了某个列字段，则本视图对象失效，重新查询视图时：
  - 若删除非视图依赖的列，查询视图触发视图重新变成有效。
  - 若删除视图依赖的列，查询视图，视图仍失效，视图的列信息变成unknown。此时需重新添加删除的列，再次查询视图。
- 基表更名了某个列字段，则本视图对象失效，重新查询视图时：
  - 若更名非视图依赖的列，查询视图触发视图重新变成有效。
  - 若更名视图依赖的列，查询视图，视图仍失效，视图的列信息变成unknown。此时需重新将该列改回原来的名称，再次查询视图。
- 基表修改了某个列字段（数据类型，属性），则本视图对象失效，重新查询视图时：
  - 若修改不影响视图，视图重新编译成功，查询视图触发视图变成有效。
  - 若修改会影响视图，导致编译失败，视图仍失效，视图的列信息变成unknown。此时需将该列还原，再次查询视图。
- 删除或更名视图依赖的对象，也会导致视图失效。

视图元数据

当视图依赖的对象发生变更（如上所述），会触发视图的元数据刷新。

当视图依赖的对象是同义词或dblink时，连接对象的变更无法失效视图，也就无法刷新视图的元数据。

老版本升级到当前版本之后，视图是valid状态，但是视图的元数据可能不正确（基表发生过结构变更），视图的元数据刷新不了。此时需重建视图或对基表执行DDL，触发视图刷新元数据。

若视图失效，备库查询会报错，需要在主库上重新编译视图刷新状态从而在备库查询。

with read only

指定该语句表示所创建的视图只读，即不能更改视图中的数据，YashanDB默认即为此模式。

示例

```
CREATE VIEW v_area_readonly AS SELECT * FROM area WITH READ ONLY;
```

with check option

该语句用于语法兼容。

constraint constraint

该语句用于语法兼容。

DELETE

通用描述

DELETE用于执行对数据库的表的删除记录操作，包括HEAP表、TAC表和LSC表。

在DELETE的事务（Transaction）被提交（Commit）之前，其他会话仍可以查询到这些被删除的数据。在YashanDB里，可以通过打开自动提交（SET AUTOCOMMIT ON）的开关，这样其他会话将不会查询到这些数据。

对于被子表定义了外键约束的父表，如果要被删除的外键列项数据值已在子表中存在，则无法删除该父表的此行数据（除非该外键约束被定义了ON DELETE），详细描述请参考通用SQL语法constraint的FOREIGN KEY。

语句定义

delete::=

```
syntax::= DELETE [hint] [FROM] table_reference  [WHERE condition]
```

table_reference::=

```
syntax::= [schema "." ] table [dblink][partition_extension_clause] [t_alias]
```

partition_extension_clause::=

```
syntax::= PARTITION ("(" partition ")"|FOR "(" (partition_key_value) {","(partition_key_value)} ")")|SUBPARTITION ("(" 
subpartition ")"|FOR "(" (subpartition_key_value) {","(subpartition_key_value)} ")")
```

hint

该语句用于提出给定的方案到优化器（Optimizer），使其按照此方案生成语句的执行计划。查看hint说明。

table_reference

该语句用于指定要删除记录的对象，及表（包括本地数据库的表和远端表)。

dblink

使用该语句表示要删除的是远端表。详细说明参见dblink。

partition_extension_clause

与INSERT语句中partition_extension_clause的描述一致。

未指定本语句时，对分区表，由系统根据分区键字段的值判断要删除记录所在的表分区。

指定本语句时，要删除的记录须存在于指定的分区中，否则无法删除成功。

t_alias

定义别名。

where condition

该语句用于指定condition，按此条件过滤出的记录行被执行删除操作。可省略，则表示删除表的所有行，但该表及表上索引所占的数据空间仍然被保留。

示例1

```
DELETE FROM area WHERE area_no='08';
```

示例2（单机/共享集群部署）

```
--orders_info表的p_orders_info_1分区中包含如下四条记录
SELECT * FROM orders_info PARTITION (p_orders_info_1);
ORDER_NO          PRODUCT_NO AREA  BRANCH ORDER_DATE                      SALESPERSON            ID
----------------- ---------- ----- ------ ------------------------------- ------------- -----------
20010102020001    11001      02    0201   2022-05-01 22:55:32             0201010011            300
20210102020002    11001      02    0201   2022-05-01 22:55:32             0201008003            400
20210102020002    11002      02    0201   2022-05-01 22:55:32             0201010011            200
20210102020002    10001      02    0201   2022-05-01 22:55:32             0201008003            100

--删除指定分区的指定记录
DELETE FROM orders_info PARTITION(p_orders_info_1) WHERE ID=300;

--result
SELECT * FROM orders_info PARTITION (p_orders_info_1);
ORDER_NO          PRODUCT_NO AREA  BRANCH ORDER_DATE                      SALESPERSON            ID
----------------- ---------- ----- ------ ------------------------------- ------------- -----------
20210102020002    11001      02    0201   2022-05-01 22:55:32             0201008003            400
20210102020002    11002      02    0201   2022-05-01 22:55:32             0201010011            200
20210102020002    10001      02    0201   2022-05-01 22:55:32             0201008003            100
```

示例2（HEAP表）

```
--branches为HEAP表，其area_no字段上存在CONSTRAINT c_branches_1 REFERENCES area(area_no) ON DELETE SET NULL外键约束定义

--查看area_no为'04'对应的子表记录
SELECT * FROM branches WHERE area_no='04';
BRANCH_NO BRANCH_NAME             AREA_NO ADDRESS
--------- ----------------------- ------- -----------------
0401      北京                    04
0402      天津                    04
0403      大连                    04      大连市
0404      沈阳                    04

--对父表area执行删除
DELETE FROM area WHERE area_no='04';
COMMIT;

--子表相应字段值被置为NULL
SELECT * FROM branches WHERE area_no='04';
BRANCH_NO BRANCH_NAME                                                    AREA_NO ADDRESS
--------- ------------------------------------------------------------- ------- ----------------------------------------
----------------------
```

DROP ACCESS CONSTRAINT

通用描述

DROP ACCESS CONSTRAINT用于删除一个AC对象，此时该AC所占的数据空间也全部被释放。

语句定义

drop access constraint::=

```
syntax::= DROP ACCESS CONSTRAINT ac_name
```

ac_name

指定要删除的AC的名称。

示例（LSC表）

```
DROP ACCESS CONSTRAINT ac_area;
```

DROP AUDIT POLICY

通用描述

DROP AUDIT POLICY用于删除一个审计策略。

只有拥有AUDIT_ADMIN审计管理员角色，或者拥有AUDIT SYSTEM系统权限的用户，才能删除一个审计策略。

对于已被使能的审计策略，不允许直接删除，而应该先执行NOAUDIT POLICY之后再进行删除操作。

语句定义

drop_audit_policy::=

```
syntax::= DROP AUDIT POLICY policy_name
```

policy_name

将要删除的审计策略的名称。

示例

```
DROP AUDIT POLICY up1;
DROP AUDIT POLICY up2;
DROP AUDIT POLICY up3;
DROP AUDIT POLICY up4;
DROP AUDIT POLICY audit_role_a;
```

DROP DATABASE LINK

通用描述

DROP DATABASE LINK语句用于删除一个数据库链接对象。

本语句适用于单机部署和共享集群部署。

根据数据库系统的结构，支持两种数据库链接：

- 同构数据库链接：YashanDB与YashanDB的数据库链接
- 异构数据库链接：YashanDB与Oracle的数据库链接，需要进行异构数据库链接配置

语句定义

drop database link**::=**

```
syntax::= DROP [PUBLIC] DATABASE LINK dblink_name
```

public

该语句删除一个公有数据库链接，对所有数据库用户可见。

删除公有数据库链接的用户须拥有DROP PUBLIC DATABASE LINK系统权限，删除非公有数据库链接用户须拥有DROP DATABASE LINK系统权限，否则返回错误。

示例（单机部署、共享集群部署）

```
DROP PUBLIC DATABASE LINK dblink_yashan;
```

dblink_name

该语句用于指定创建的数据库链接的名称，不可省略，且需符合YashanDB的对象命名规范。

示例（单机部署、共享集群部署）

```
-- YashanDB与YashanDB的数据库链接
DROP DATABASE LINK dblink_yashan;

-- YashanDB与Oracle的数据库链接
DROP DATABASE LINK dblink_oracle;
```

DROP DATABASE

通用描述

DROP DATABASE用于删除当前数据库，删除数据库所包含的如下持久化文件：

- 控制文件
- 数据文件
- Slice文件
- 日志文件

> **Note**：
> 本语句只能由SYS用户在数据库处于NOMOUNT状态时执行，且要求数据库的控制文件完整（控制文件组成员均存在且正确）。
> 共享集群部署模式下执行该语句需保证执行节点角色为MASTER ROLE，且为NOMOUNT状态。
> 执行完DROP DATABASE语句之后数据库会自行重启。

本语句不适用于分布式部署。

语句定义

drop database::=

```
syntax::= DROP DATABASE [INCLUDING ARCHIVELOG]
```

including archivelog

该语句决定是否删除数据库的归档日志文件，缺省为不删除。

示例（单机、共享集群部署）

```
--启动实例到NOMOUNT
$ yasboot cluster restart -c yashandb -m nomount

--删除数据库
SQL> DROP DATABASE;

--完成后将自动重启实例到NOMOUNT
Starting instance nomount
Instance started
conn sys/password
```

DROP DIRECTORY

通用描述

DROP DIRECTORY用于删除数据目录对象，执行本语句的用户需具有drop any directory权限。

该语句仅适用于单机部署。

语句定义

drop directory::=

```
syntax::= DROP DIRECTORY directory_name
```

directory_name

该语句用于指定待删除的目录名称，不可省略。

示例（单机部署）

```
DROP DIRECTORY dir;
```

DROP FUNCTION

通用描述

DROP FUNCTION用于删除一个已有的自定义函数。

普通用户只能删除自己创建的自定义函数。

当要删除的函数被其他过程体引用时，删除本函数会导致该过程体运行时报编译错误。

语句定义

drop function::=

```
syntax::= DROP FUNCTION [IF EXISTS] [schema "."] function_name
```

if exists

该语句用于指定在DROP 自定义函数之前，先判断该自定义函数是否存在，省略则不会判断，此时如果要删除的自定义函数不存在，系统将提示错误。

示例（单机、共享集群部署）

```
DROP FUNCTION IF EXISTS ya_func;
```

DROP INDEX

通用描述

DROP INDEX用于删除一个索引对象，此时该索引所占的数据空间也全部被释放。

当分区索引被删除时，其所有的索引分区也被删除。

语句定义

drop index::=

```
syntax::= DROP INDEX [schema"."] index_name
```

index_name

指定要删除的索引的名称。

示例（HEAP表、TAC表）

```
DROP INDEX idx_sales_info_1;
```

DROP LIBRARY

通用描述

DROP LIBRARY用于删除一个已有的自定义库。

普通用户只能删除自己创建的自定义库。删除其他用户的自定义库，需要有DROP ANY LIBRARY权限。

当要删除的自定义库被外置UDF引用时，删除本自定义库会导致该外置UDF运行时提示错误。

语句定义

drop library::=

```
syntax::= DROP LIBRARY [IF EXISTS] [schema "."] library_name
```

if exists

该语句用于指定在DROP自定义库之前，先判断该自定义库是否存在，省略则不会判断，此时如果要删除的自定义库不存在，系统将提示错误。

示例（单机、共享集群部署）

```
DROP LIBRARY IF EXISTS ya_lib;
```

DROP MATERIALIZED VIEW

通用描述

DROP MATERIALIZED VIEW用于删除一个物化视图对象。

分布式、集群部署中用户无法执行本语句。

语句定义

drop materialized view::=

```
syntax::= DROP MATERIALIZED VIEW [schema"."] materialized_view_name
```

materialized_name

指定要删除的物化视图的名称。

示例（单机HEAP表）

```
DROP MATERIALIZED VIEW mv_refresh;
```

DROP OUTLINE

通用描述

DROP OUTLINE用于删除一个存储纲要。

用户必须拥有DROP ANY OUTLINE权限才能删除一个存储纲要。

语句定义

drop_outline::=

```
syntax::= DROP [(PUBLIC)] OUTLINE outline_name
```

public

公有模式，默认值。

outline_name

将要删除的OUTLINE的名称。

示例

```
-- 删除指定的OUTLINE
DROP OUTLINE ol_a;
```

DROP PACKAGE

通用描述

DROP PACKAGE用于删除一个已有的自定义高级包。

普通用户只能删除自己创建的高级包。

当要删除的高级包的变量或子对象被其他过程体引用时，删除本高级包会导致该过程体运行时报编译错误。

语句定义

drop package::=

```
syntax::= DROP PACKAGE [BODY] [IF EXISTS] [schema "."] package_name
```

body

用于指定是否只删除UDP的BODY。当不指定BODY时，将同时删除PACKAGE HEAD和PACKAGE BODY。

if exists

该语句用于指定在删除UDP之前，先判断该UDP是否存在，省略则不会判断，此时如果要删除的UDP不存在，系统将提示错误。

示例（单机、共享集群部署）

```
DROP PACKAGE IF EXISTS calc_fee;
```

DROP PROCEDURE

通用描述

DROP PROCEDURE用于删除一个已有的存储过程。

普通用户只能删除自己创建的存储过程。

当要删除的存储过程被其他过程体引用时，删除本存储过程会导致该过程体运行时报编译错误。

语句定义

drop procedure::=

```
syntax::= DROP PROCEDURE [IF EXISTS] [schema "."] procedure_name
```

if exists

该语句用于指定在DROP 存储过程之前，先判断该存储过程是否存在，省略则不会判断，此时如果要删除的存储过程不存在，系统将提示错误。

示例（单机、共享集群部署）

```
DROP PROCEDURE IF EXISTS ya_proc;
```

DROP PROFILE

通用描述

DROP PROFILE 用于删除一个profile。

执行本语句需注意如下事项：

- 用户必须拥有DROP PROFILE权限才能删除一个profile。
- 系统默认的profile（名称为DEFAULT）不能被删除。
- 对于已与用户关联的profile，必须使用CASCADE关键字才能删除。

分布式部署中用户无法执行本语句。

语句定义

drop profile::=

```
syntax::= DROP PROFILE profile_name [CASCADE]
```

profile_name

已存在的一个profile的名称。

cascade

指定CASCADE表示删除profile的同时，也删除其与用户的关联关系。

示例（单机、共享集群部署）

```
DROP PROFILE prof_1;

DROP PROFILE prof_2 CASCADE;
```

DROP ROLE

通用描述

DROP ROLE用于删除一个存在的角色。角色被删除时，系统自动对已授权该角色的用户和角色进行角色收回。

系统内置角色不允许删除。

语句定义

drop user::=

```
syntax::= DROP ROLE role
```

role

该语句用于指定要删除的角色的名称。

示例

```
DROP ROLE rolename;
```

DROP SEQUENCE

通用描述

DROP SEQUENCE用于删除一个已存在的序列号生成器。

当序列号生成器已被其他数据库对象引用时，删除这个序列号生成器可能会引起一些错误，或者让对象失效。例如：

- 如在某张表的列字段上引用序列号生成器定义了DEFAULT值，则在该序列号生成器被删除后，对该表使用DEFAULT生成列字段的值时，提示YAS-02012错误。
- 如在某张视图的定义中指定了序列号生成器作为一个列字段，则在该序列号生成器被删除后，该视图失效。

分布式部署中用户无法执行本语句。

语句定义

drop sequence::=

```
syntax::= DROP SEQUENCE [schema "."] sequence_name
```

示例（单机、共享集群部署）

```
DROP SEQUENCE seq_yashan1;
```

DROP SQLMAP

通用描述

DROP SQLMAP用于删除一个SQL映射。

用户必须拥有DBA权限才能删除一个SQL映射。

语句定义

drop sqlmap::=

```
syntax::= DROP SQLMAP sqlmap_name
```

sqlmap_name

该语句用于指定要删除的SQL映射的名称。

示例

```
DROP SQLMAP map_branch;
```

DROP SYNONYM

通用描述

DROP SYNONYM用于删除一个同义词。

分布式部署中用户无法执行本语句。

语句定义

drop synonym::=

```
syntax::= DROP [PUBLIC] SYNONYM synonym_name
```

public

删除一个公共同义词时，此语句不可省略，必须指定。

synonym_name

该语句用于指定要删除的同义词的名称。

示例（单机、共享集群部署）

```
--删除公共同义词
DROP PUBLIC SYNONYM sy_area1;

--删除本用户下的私有同义词
DROP SYNONYM sy_area2;
```

DROP TABLE

通用描述

DROP TABLE语句用于删除表对象（包括建立其上面的索引、约束、触发器等对象），同时该表的数据被删除，该表及索引所占数据空间被全部释放。

DROP 操作删除了表对象的所有相关定义，如需再次使用该表，只能按创建新表的方式进行重建。

DROP 操作不能回滚（Rollback），也不能通过flashback_query_clause获得操作之前的数据。如果指定了PURGE语句，也无法从回收站恢复。

当需要DROP被子表定义了外键约束的父表时，需要指定CASCADE CONSTRAINTS。

当需要DROP已建立了AC的源表时，需先DROP该表上全部的AC后，才可以DROP该表。

语句定义

drop table::=

```
syntax::= DROP TABLE [IF EXISTS] [schema "."] table_name [CASCADE CONSTRAINTS] [PURGE]
```

if exists

该语句用于指定在DROP表时忽略表不存在的错误。

示例

```
--DROP不存在的表area_drop
DROP TABLE IF EXISTS area_drop;
```

cascade constraints

该语句只作用于HEAP表，用于指定当表被子表定义了外键约束时，先删除子表上的外键约束项，再DROP该表。

示例（HEAP表）

```
--area的子表branches表上定义的外键约束被删除
DROP TABLE area CASCADE CONSTRAINTS;
```

purge

该语句用于指定彻底删除表，而不是进入回收站（ReclycleBin）。分布式部署中无法使用本语句。

YashanDB的配置参数里定义了是否使用回收站的开关，即RECYCLYBIN参数。当RECYCLEBIN设置为ON时，执行DROP TABLE时系统会将表对象及数据放入回收站（表名称被修改为BIN开头的系统生成字符串），用于误操作之后的恢复。

如果显式指定了PURGE语句，则DROP的表对象及数据不会进入回收站，而是被彻底删除，无法恢复。

示例（单机、共享集群部署）

```
DROP TABLE finance_info PURGE;
```

DROP TABLESPACE SET

通用描述

DROP TABLESPACE SET语句用于删除一个表空间集。

不能删除内置表空间集USERS/USERS_AIM。

语句定义

drop tablespace set::=

```
syntax::= DROP TABLESPACE SET tablespace_set_name INCLUDING CONTENTS
```

tablespace_set_name

该语句用于指定要删除的表空间集的名称。

including contents

该语句用于指定将表空间集及其包含的所有对象都删除。

表空间集挂载的bucket中数据文件尚未完成归档清理时，该bucket无法删除，此时删除该表空间集会返回错误。

示例（分布式部署）

```
DROP TABLESPACE SET tbs_tb INCLUDING CONTENTS;
DROP TABLESPACE SET mm_tss INCLUDING CONTENTS;
DROP TABLESPACE SET tss1 INCLUDING CONTENTS;
```

DROP TABLESPACE

通用描述

DROP TABLESPACE用于删除一个表空间。

该语句有如下约束限制：

- 不能删除一个已被某个用户指定的DEFAULT TABLESPACE，必须先通过ALTER USER语句将其变更。
- 不能删除一个已被某个实例指定的DEFAULT SWAP TABLESPACE，必须先通过ALTER SYSTEM SET DEFAULT_SWAP_TABLESPACE语句将其变更。
- 不能删除一个已创建了任何数据库对象的表空间，除非指定了INCLUDING CONTENTS语句。
- 如果表空间中的分区表或分区索引存在至少一个分区在存储参数里指定了其他的表空间，那么即使指定了INCLUDING CONTENTS语句，该表空间也不能被删除。
- 如果表空间中存在被其他表空间对象定义的外键约束，那么即使指定了INCLUDING CONTENTS语句，该表空间也不能被删除，除非同时指定了CASCADE CONSTRAINT。
- 不能删除SYSTEM/SYSAUX/UNDO/SWAP/TEMPORARY等内置表空间。

分布式部署中删除表空间出现节点故障时，恢复措施见用户表空间管理章节描述。

集群部署中删除表空间出现实例故障时，可能会产生没有数据文件的残留表空间，残留表空间状态是OFFLINE，无法正常使用，只能执行删除语句将该表空间删除。

语句定义

drop tablespace::=

```
syntax::= DROP TABLESPACE [IF EXISTS] tablespace_name [INCLUDING CONTENTS [(AND|KEEP) DATAFILES] [CASCADE CONSTRAINTS]]
```

if exists

该语句用于指定在DROP表空间之前，先判断该表空间是否存在。省略则不会判断，此时如果要删除的表空间不存在，系统将提示错误。

tablespace_name

该语句用于指定要删除的表空间名称。

including contents

该语句用于指定将表空间及其包含的所有对象都删除。

and|keep datafiles

指定在删除表空间时，其对应的数据文件是被删除，还是被保留，缺省为保留。

cascade constraints

当存在其他表空间里的对象定义了到本表空间里对象的外键约束时，先将这些外键约束项删除，再删除本表空间。

示例

```
DROP TABLESPACE yashan1;

DROP TABLESPACE IF EXISTS yashan2;

DROP TABLESPACE yashan3
INCLUDING CONTENTS
KEEP DATAFILES
CASCADE CONSTRAINTS;
```

DROP TRIGGER

通用描述

DROP TRIGGER语句用于删除一个已存在的触发器。

语句定义

DROP TRIGGER::=

```
syntax::=  DROP TRIGGER [IF EXISTS] [ schema "." ]  trigger_name
```

if exists

该语句用于指定在DROP触发器之前，先判断该触发器是否存在。省略则不会判断，此时如果要删除的触发器不存在，系统将提示错误。

schema

包含触发器的模式名称，省略时默认为当前登录用户的模式。

trigger_name

将要删除的触发器的名称。

示例（单机、共享集群部署）

```
-- 删除sales模式的tri触发器
DROP TRIGGER sales.tri;
```

DROP TYPE BODY

通用描述

DROP TYPE BODY语句用于删除一个已存在的类型主体。

语句定义

DROP TYPE BODY::=

```
syntax::=  DROP TYPE BODY [IF EXISTS] [ schema "." ]  type_name
```

if exists

该语句用于指定在删除类型主体之前，先判断该类型主体是否存在。省略则不会判断，此时如果要删除的类型主体不存在，系统将提示错误。

schema

包含类型主体的模式名称，省略时默认为当前登录用户的模式。

type_name

将要类型主体的名称。

示例（单机、共享集群部署）

```
-- 删除当前模式的udt_object类型主体，不删除已存在的udt_object类型。
DROP TYPE BODY udt_object;
```

DROP TYPE

通用描述

DROP TYPE语句用于删除一个已存在的自定义类型（UDT），同时会删除该类型存在的类型主体。

语句定义

DROP TYPE::=

```
syntax::=  DROP TYPE [IF EXISTS] [ schema "." ]  type_name [FORCE]
```

if exists

该语句用于指定在删除UDT之前，先判断该UDT是否存在。省略则不会判断，此时如果要删除的UDT不存在，系统将提示错误。

schema

包含UDT的模式名称，省略时默认为当前登录用户的模式。

type_name

将要删除的UDT的名称。

force

当类型被其它对象依赖时，FORCE选项可以强制删除该类型及类型主体。

示例（单机、共享集群部署）

```
-- 删除当前模式的udt_object自定义类型，会同时删除udt_object类型主体（如果存在）。
DROP TYPE udt_object;
```

DROP USER

通用描述

DROP USER用于在删除一个已存在的数据库用户。

不能删除一个处于会话连接中的用户。

不能删除一个已创建了任何数据库对象的用户（除非指定CASCADE）。

语句定义

drop user::=

```
syntax::= DROP USER user_name [CASCADE]
```

user_name

该语句用于指定要删除的用户的名称。

cascade

该语句用于指定删除用户的同时，删除该用户下所有的对象，包括：

- 表：表及该表上的所有的索引、约束都将被删除；如果该表上存在被其他用户的某个表（子表）的外键引用，则子表上的外键将被同步删除。
- 视图、同义词、PL对象：都将被删除；如果存在另一个用户的视图、同义词、PL对象依赖本用户的对象，则另一个用户的视图、同义词、PL对象不会删除，但被置为失效状态。
- 索引：将被删除；如果在本用户下创建过另一个用户下某个表的索引，该索引也会被删除。
- SEQUENCE等其他所有对象：都将被删除。

示例

```
DROP USER sales1 CASCADE;
```

DROP VIEW

通用描述

DROP VIEW用于删除一个视图，包括处于失效状态中的视图。

语句定义

drop view::=

```
syntax::= DROP VIEW [IF EXISTS] view_name
```

if exists

该语句用于指定在DROP视图之前，先判断该视图是否存在，若视图不存在则返回成功。省略则不会判断，此时如果要删除的视图不存在，系统将提示错误。

view_name

该语句用于指定要删除的视图的名称。

示例

```
DROP VIEW IF EXISTS v_area;
```

EXPLAIN

通用描述

EXPLAIN用于展示一条SQL语句的执行计划（Execution Plan）。

当SQL语句的执行计划已经在在plan cache中存在时，直接重读该计划并输出；否则，由优化器进行试算生成计划到plan cache后再读取并输出。

只能对DML类的SQL语句展示执行计划，同时需满足执行该SQL语句的各项约束要求。

语句定义

explain::=

```
syntax::= EXPLAIN [PLAN FOR] sql_statement
```

plan for

用于兼容，可省略。EXPLAIN PLAN FOR=EXPLAIN。

sql_statement

要展示执行计划的SQL语句，长度不超过2M。

示例

```
--单机部署中的SQL语句
EXPLAIN
WITH q_area(ano,aname) AS
(SELECT area_no ano,area_name aname FROM area WHERE area_no IN ('01','04'))
SELECT bno,bname,aname FROM (
SELECT a.branch_no bno,a.branch_name bname,b.aname aname FROM branches a,q_area b WHERE a.area_no=b.ano AND a.area_no='01'
UNION
SELECT a.branch_no bno,a.branch_name bname,b.aname aname FROM branches a,q_area b WHERE a.area_no=b.ano AND a.area_no='04'
);

--分布式部署中的SQL语句
EXPLAIN
SELECT *
FROM (SELECT * FROM area WHERE area_no IN ('01','02')
      UNION
      SELECT * FROM area WHERE area_no IN ('01','03')
     );
```

输出定义

输出1.单机HEAP表输出的执行计划为：

```
PLAN_DESCRIPTION
------------------------------------------------------------------
SQL hash value: 3782799764
Optimizer: ADOPT_C

+----+--------------------------------+----------------------+------------+----------+-------------+-------------------------
-------+
| Id | Operation type                 | Name                 | Owner      | Rows     | Cost(%CPU)  | Partition info
|
+----+--------------------------------+----------------------+------------+----------+-------------+-------------------------
-------+
|  0 | SELECT STATEMENT               |                      |            |          |             |
|
|  1 |  VIEW                          |                      |            |      1000|     1652( 0)|
|
```

```
|  2 |   HASH DISTINCT                |                      |            |      1000|     1652( 0)|
|
|  3 |    VIEW                        |                      |            |   2512000|     1576( 0)|
|
|  4 |     UNION ALL                  |                      |            |   2512000|     1446( 0)|
|
|  5 |      NESTED LOOPS INNER        |                      |            |   1256000|      687( 0)|
|
|* 6 |       TABLE ACCESS FULL        | BRANCHES             | SALES      |      4000|      445( 0)|
|
|* 7 |       VIEW                     |                      |            |       314|      151( 0)|
|
|  8 |        TABLE ACCESS BY INDEX ROWID| AREA                | SALES      |          |             |
|
|* 9 |         INDEX RANGE SCAN       | SYS_C_13             | SALES      |      7840|      151( 0)|
|
| 10 |      NESTED LOOPS INNER        |                      |            |   1256000|      687( 0)|
|
|*11 |       TABLE ACCESS FULL        | BRANCHES             | SALES      |      4000|      445( 0)|
|
|*12 |       VIEW                     |                      |            |       314|      151( 0)|
|
| 13 |        TABLE ACCESS BY INDEX ROWID| AREA                | SALES      |          |             |
|
|*14 |         INDEX RANGE SCAN       | SYS_C_13             | SALES      |      7840|      151( 0)|
|
+----+--------------------------------+----------------------+------------+----------+-------------+-------------------------
-------+

Operation Information (identified by operation id):
---------------------------------------------------

   6 - Predicate : filter("A"."AREA_NO" = '01')
   7 - Predicate : filter("B"."ANO" = '01')
   9 - Predicate : access("AREA"."AREA_NO" IN ('01', '04'))
  11 - Predicate : filter("A"."AREA_NO" = '04')
  12 - Predicate : filter("B"."ANO" = '04')
  14 - Predicate : access("AREA"."AREA_NO" IN ('01', '04'))
```

输出2.单机TAC表输出的执行计划为：

```
PLAN_DESCRIPTION
---------------------------------------------------------------------
SQL hash value: 3782799764
Optimizer: ADOPT_C

+----+--------------------------------+----------------------+------------+----------+-------------+-------------------------
-------+
| Id | Operation type                 | Name                 | Owner      | Rows     | Cost(%CPU)  | Partition info
|
+----+--------------------------------+----------------------+------------+----------+-------------+-------------------------
-------+
|  0 | SELECT STATEMENT               |                      |            |          |             |
|
|  1 |  COL TO ROW                    |                      |            |          |             |
|
|  2 |   RESULT                       |                      |            |      1000|     1652( 0)|
|
|  3 |    HASH GROUP                  |                      |            |      1000|     1652( 0)|
|
|  4 |     RESULT                     |                      |            |   2512000|     1576( 0)|
|
|  5 |      UNION ALL                 |                      |            |   2512000|     1446( 0)|
|
|  6 |       NESTED LOOPS INNER       |                      |            |   1256000|      687( 0)|
|
|* 7 |        TABLE ACCESS FULL       | BRANCHES             | SALES      |      4000|      445( 0)|
|
|* 8 |        RESULT                  |                      |            |       314|      151( 0)|
```

```
|
|  9 |         TABLE ACCESS BY INDEX ROWID| AREA                | SALES      |          |             |
|
|*10 |          INDEX RANGE SCAN      | SYS_C_58            | SALES      |      7840|      151( 0)|
|
| 11 |       NESTED LOOPS INNER       |                     |            |   1256000|      687( 0)|
|
|*12 |        TABLE ACCESS FULL       | BRANCHES            | SALES      |      4000|      445( 0)|
|
|*13 |        RESULT                  |                     |            |       314|      151( 0)|
|
| 14 |         TABLE ACCESS BY INDEX ROWID| AREA                | SALES      |          |             |
|
|*15 |          INDEX RANGE SCAN      | SYS_C_58            | SALES      |      7840|      151( 0)|
|
+----+--------------------------------+---------------------+------------+----------+-------------+-------------------------
-------+

Operation Information (identified by operation id):
---------------------------------------------------

   1 - Projection: RemoteTable[1][CHAR, 4], RemoteTable[1][VARCHAR, 200], RemoteTable[1][VARCHAR, 60]
   2 - Projection: Tuple[0, 0][CHAR, 4], Tuple[0, 1][VARCHAR, 200], Tuple[0, 2][VARCHAR, 60]
   3 - Projection: Tuple[0, 0][CHAR, 4], Tuple[0, 1][VARCHAR, 200], Tuple[0, 2][VARCHAR, 60]
   4 - Projection: Tuple[0, 0][CHAR, 4], Tuple[0, 1][VARCHAR, 200], Tuple[0, 2][VARCHAR, 60]
   5 - Projection: Tuple[0, 0][CHAR, 4], Tuple[0, 1][VARCHAR, 200], Tuple[0, 2][VARCHAR, 60]
   6 - Projection: Tuple[0, 0][CHAR, 4], Tuple[0, 1][VARCHAR, 200], Tuple[1, 0][VARCHAR, 60]
   7 - Projection: Tuple[0, 0][CHAR, 4], Tuple[0, 1][VARCHAR, 200]
       Predicate : access("A"."AREA_NO" = '01')
   8 - Projection: Tuple[0, 1][VARCHAR, 60]
       Predicate : filter(Tuple[0, 0] = '01')
  10 - Projection: Tuple[0, 0][CHAR, 2], Tuple[0, 1][VARCHAR, 60]
       Predicate : access("AREA"."AREA_NO" IN ('01', '04'))
  11 - Projection: Tuple[0, 0][CHAR, 4], Tuple[0, 1][VARCHAR, 200], Tuple[1, 0][VARCHAR, 60]
  12 - Projection: Tuple[0, 0][CHAR, 4], Tuple[0, 1][VARCHAR, 200]
       Predicate : access("A"."AREA_NO" = '04')
  13 - Projection: Tuple[0, 1][VARCHAR, 60]
       Predicate : filter(Tuple[0, 0] = '04')
  15 - Projection: Tuple[0, 0][CHAR, 2], Tuple[0, 1][VARCHAR, 60]
       Predicate : access("AREA"."AREA_NO" IN ('01', '04'))
```

输出3.单机LSC表输出的执行计划为：

```
PLAN_DESCRIPTION
------------------------------------------------------------------
SQL hash value: 3782799764
Optimizer: ADOPT_C

+----+--------------------------------+---------------------+------------+----------+-------------+-------------------------
-------+
| Id | Operation type                 | Name                | Owner      | Rows     | Cost(%CPU)  | Partition info
|
+----+--------------------------------+---------------------+------------+----------+-------------+-------------------------
-------+
|  0 | SELECT STATEMENT               |                     |            |          |             |
|
|  1 |  COL TO ROW                    |                     |            |          |             |
|
|  2 |   RESULT                       |                     |            |      1000|     2252( 0)|
|
|  3 |    HASH GROUP                  |                     |            |      1000|     2252( 0)|
|
|  4 |     RESULT                     |                     |            |   2512000|     2176( 0)|
|
|  5 |      UNION ALL                 |                     |            |   2512000|     2046( 0)|
|
|  6 |       NESTED LOOPS INNER       |                     |            |   1256000|      987( 0)|
|
|* 7 |        TABLE ACCESS FULL       | BRANCHES            | SALES      |      4000|      445( 0)|
```

```
|
|* 8 |        RESULT                |                      |            |       314|      451( 0)|
|
|* 9 |         TABLE ACCESS FULL    | AREA                 | SALES      |      7840|      451( 0)|
|
| 10 |       NESTED LOOPS INNER     |                      |            |   1256000|      987( 0)|
|
|*11 |        TABLE ACCESS FULL     | BRANCHES             | SALES      |      4000|      445( 0)|
|
|*12 |        RESULT                |                      |            |       314|      451( 0)|
|
|*13 |         TABLE ACCESS FULL    | AREA                 | SALES      |      7840|      451( 0)|
|
+----+------------------------------+----------------------+------------+----------+-------------+-------------------------
-------+

Operation Information (identified by operation id):
---------------------------------------------------

   1 - Projection: RemoteTable[1][CHAR, 4], RemoteTable[1][VARCHAR, 200], RemoteTable[1][VARCHAR, 60]
   2 - Projection: Tuple[0, 0][CHAR, 4], Tuple[0, 1][VARCHAR, 200], Tuple[0, 2][VARCHAR, 60]
   3 - Projection: Tuple[0, 0][CHAR, 4], Tuple[0, 1][VARCHAR, 200], Tuple[0, 2][VARCHAR, 60]
   4 - Projection: Tuple[0, 0][CHAR, 4], Tuple[0, 1][VARCHAR, 200], Tuple[0, 2][VARCHAR, 60]
   5 - Projection: Tuple[0, 0][CHAR, 4], Tuple[0, 1][VARCHAR, 200], Tuple[0, 2][VARCHAR, 60]
   6 - Projection: Tuple[0, 0][CHAR, 4], Tuple[0, 1][VARCHAR, 200], Tuple[1, 0][VARCHAR, 60]
   7 - Projection: Tuple[0, 0][CHAR, 4], Tuple[0, 1][VARCHAR, 200]
       Predicate : access("A"."AREA_NO" = '01')
   8 - Projection: Tuple[0, 1][VARCHAR, 60]
       Predicate : filter(Tuple[0, 0] = '01')
   9 - Projection: Tuple[0, 0][CHAR, 2], Tuple[0, 1][VARCHAR, 60]
       Predicate : access("AREA"."AREA_NO" IN ('01', '04'))
  10 - Projection: Tuple[0, 0][CHAR, 4], Tuple[0, 1][VARCHAR, 200], Tuple[1, 0][VARCHAR, 60]
  11 - Projection: Tuple[0, 0][CHAR, 4], Tuple[0, 1][VARCHAR, 200]
       Predicate : access("A"."AREA_NO" = '04')
  12 - Projection: Tuple[0, 1][VARCHAR, 60]
       Predicate : filter(Tuple[0, 0] = '04')
  13 - Projection: Tuple[0, 0][CHAR, 2], Tuple[0, 1][VARCHAR, 60]
       Predicate : access("AREA"."AREA_NO" IN ('01', '04'))
```

输出4.分布式TAC/LSC表输出的执行计划为：

```
PLAN_DESCRIPTION
--------------------------------------------------------------------
SQL hash value: 3958538062
Optimizer: ADOPT_C

+----+------------------------------+----------------------+------------+----------+-------------+-------------------------
-------+
| Id | Operation type               | Name                 | Owner      | Rows     | Cost(%CPU)  | Partition info
|
+----+------------------------------+----------------------+------------+----------+-------------+-------------------------
-------+
|  0 | SELECT STATEMENT             |                      |            |          |             |
|
|  1 |  DISTRIBUTED COORDINATOR     |                      |            |          |             |
|
|  2 |   COL TO ROW                 |                      |            |          |             |
|
|  3 |    RESULT                    |                      |            |      1000|      959( 0)|
|
|  4 |     HASH DISTINCT            |                      |            |      1000|      959( 0)|
|
|  5 |      RESULT                  |                      |            |     15680|      958( 0)|
|
|  6 |       UNION ALL              |                      |            |     15680|      957( 0)|
|
|  7 |        PX N2I REMOTE         | QUEUE_0              |            |      7840|      478( 0)|
|
|* 8 |         TABLE ACCESS FULL    | AREA                 | SALES      |      7840|      451( 0)|
```

```
|
|  9 |        PX N2I REMOTE           | QUEUE_1              |            |      7840|      478( 0)|
|
|*10 |         TABLE ACCESS FULL      | AREA                 | SALES      |      7840|      451( 0)|
|
+----+--------------------------------+----------------------+------------+----------+-------------+-------------------------
-------+

Operation Information (identified by operation id):
---------------------------------------------------

   2 - Projection: RemoteTable[1][CHAR, 2]
   3 - Projection: Tuple[0, 0][CHAR, 2]
   4 - Projection: Tuple[0, 0][CHAR, 2]
   5 - Projection: Tuple[0, 0][CHAR, 2]
   6 - Projection: Tuple[0, 0][CHAR, 2]
   7 - Projection: Tuple[0, 0][CHAR, 2]
       PX RemoteInfo: (RANDOM SENDER -> RANDOM RECEIVER : 3->1 [3][4][5]->[2])
   8 - Projection: Tuple[0, 0][CHAR, 2]
       Predicate : access("AREA"."AREA_NO" IN ('01', '02'))
   9 - Projection: Tuple[0, 0][CHAR, 2]
       PX RemoteInfo: (RANDOM SENDER -> RANDOM RECEIVER : 3->1 [3][4][5]->[2])
  10 - Projection: Tuple[0, 0][CHAR, 2]
       Predicate : access("AREA"."AREA_NO" IN ('01', '03'))
```

sql hash value

SQL语句的哈希号，即V$SQL视图里的hash_value字段值。

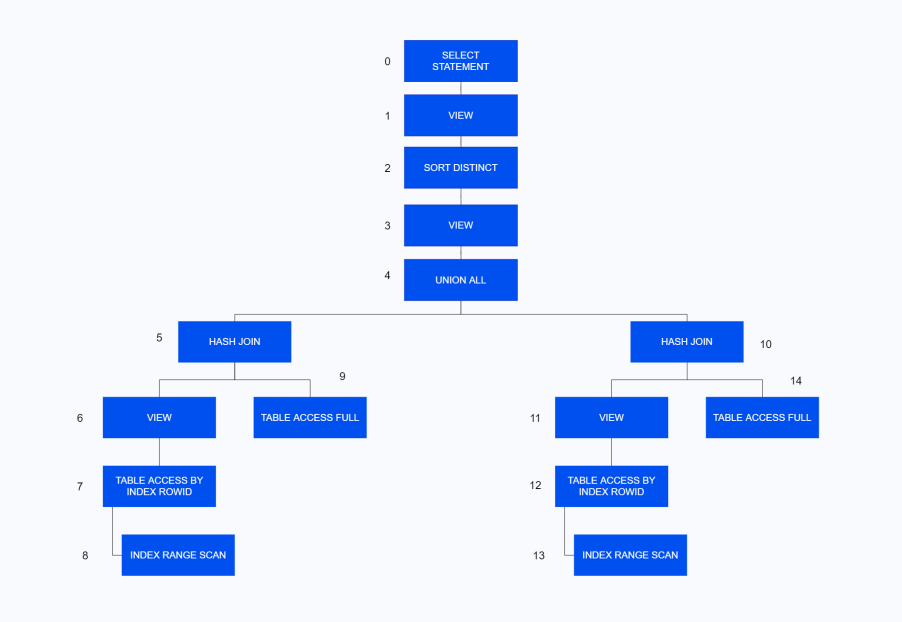
optimizer

YashanDB使用的优化器名称， ADOPT_C表示CBO。

id

执行步骤号，按二叉树结构从底部向上逆序执行。

如下为一个执行二叉树示例：



operation type

执行算子，定义了执行计划每一步的操作。

如上例输出1和2中的INDEX RANGE SCAN表示执行索引范围扫描，检索到ROWID，TABLE ACCESS BY INDEX ROWID则根据这个ROWID读取行数据，然后继续向上直到0步的算子。

上例输出3和4中的TABLE ACCESS FULL表示执行全表扫描，检索到记录RESULT后，对两个RESULT进行UNION ALL算法，然后继续向上直到0步的算子。

其中，PX算子可分为BROADCAST、HASH、RANDOM等类型，表示数据在节点间的重分布方式。在每个PX算子后面会列出节点发送情况，N2I表示多个节点将数据发往一个节点，具体发送和接收节点可在投影信息中查看。

name

算子操作的数据库对象名称，如扫描表、索引时对应的表名、索引名等。

对于PX算子，Name列示的是该算子在通讯过程中的唯一标识号。

owner

操作对象的属主。如上例输出1中的area、branches表，及SYS_C_16索引，其Owner为SALES用户。

上例输出2中的area、branches表，其Owner为SALES_LSC用户。

上例输出3中的area表，其Owner为SALES用户。

rows

算子扫描的记录行数。

cost(%cpu)

估算执行此算子产生的CPU代价。

partition info

对于分区表，此列展示算子扫描到的分区。

示例1（单机HEAP表）

```
--创建sales表为一张范围分区表，且在year字段上创建local索引。
CREATE TABLE sales
(year CHAR(4) NOT NULL,
month CHAR(2) NOT NULL,
branch CHAR(4),
product CHAR(5),
quantity NUMBER DEFAULT 0 NOT NULL,
amount NUMBER(10,2) DEFAULT 0 NOT NULL,
salsperson CHAR(10))
PARTITION BY RANGE(year)
(PARTITION p_sales_1 VALUES LESS THAN ('2001'),
PARTITION p_sales_2 VALUES LESS THAN ('2011'),
PARTITION p_sales_3 VALUES LESS THAN (2022));

CREATE INDEX idx_sales ON sales(year) LOCAL;

INSERT INTO sales VALUES ('2001','01','0201','11001',30,500,'0201010011');
INSERT INTO sales VALUES ('2015','11','0101','11001',20,300,'');
INSERT INTO sales VALUES ('2021','10','0101','11001',20,300,'');
INSERT INTO sales VALUES ('2000','12','0102','11001',20,300,'');
INSERT INTO sales VALUES ('2015','03','0102','11001',20,300,'');
INSERT INTO sales VALUES ('2020','05','0101','11001',40,600,'');
COMMIT;

--查看执行计划
EXPLAIN
SELECT a.area_name,
b.branch_name,
s.product,
s.amount
FROM branches b,
area a,
(SELECT branch,
product,SUM(amount) amount
FROM sales
```

```
WHERE year>'2005'
GROUP BY branch,product) s
WHERE s.branch=b.branch_no
AND b.area_no=a.area_no;
```

上例的输出结果（[1,2]表示扫描第1到第2个分区）为：

```
--以HEAP表为例，TAC表将会多出一个COL TO ROW算子
PLAN_DESCRIPTION
----------------------------------------------------------------
SQL hash value: 3362858118
Optimizer: ADOPT_C

+----+--------------------------------+---------------------+------------+----------+-------------+-------------------------
-------+
| Id | Operation type                 | Name                | Owner      | Rows     | Cost(%CPU)  | Partition info
|
+----+--------------------------------+---------------------+------------+----------+-------------+-------------------------
-------+
|  0 | SELECT STATEMENT               |                     |            |          |             |
|
|  1 |  NESTED LOOPS INNER            |                     |            |      1000|      461( 0)|
|
|  2 |   NESTED LOOPS INNER           |                     |            |      1000|      311( 0)|
|
|  3 |    VIEW                        |                     |            |      1000|      161( 0)|
|
|* 4 |     HASH GROUP                 |                     |            |      1000|      161( 0)|
|
|  5 |      PART SCAN ITERATOR        |                     |            |     33000|      160( 0)| [1,2]
|
|  6 |       TABLE ACCESS BY INDEX ROWID| SALES               | SALES      |          |             |
|
|* 7 |        INDEX RANGE SCAN        | IDX_SALES           | SALES      |     33000|      160( 0)|
|
|  8 |    TABLE ACCESS BY INDEX ROWID | BRANCHES            | SALES      |          |             |
|
|* 9 |     INDEX UNIQUE SCAN          | SYS_C_125           | SALES      |         1|      149( 0)|
|
| 10 |   TABLE ACCESS BY INDEX ROWID  | AREA                | SALES      |          |             |
|
|*11 |    INDEX UNIQUE SCAN           | SYS_C_123           | SALES      |         1|      149( 0)|
|
+----+--------------------------------+---------------------+------------+----------+-------------+-------------------------
-------+

Operation Information (identified by operation id):
---------------------------------------------------

   4 - Predicate : group expressions("SALES"."BRANCH", "SALES"."PRODUCT")
   7 - Predicate : access("SALES"."YEAR" > '2005')
   9 - Predicate : access("B"."BRANCH_NO" = "S"."BRANCH")
  11 - Predicate : access("A"."AREA_NO" = "B"."AREA_NO")
```

示例2（分布式LSC表）

```
--sales表为一张范围分区表
CREATE DUPLICATED TABLE sales
(year CHAR(4) NOT NULL,
month CHAR(2) NOT NULL,
branch CHAR(4),
product CHAR(5),
quantity NUMBER DEFAULT 0 NOT NULL,
amount NUMBER(10,2) DEFAULT 0 NOT NULL,
salsperson CHAR(10))
PARTITION BY RANGE(year)
(PARTITION p_sales_1 VALUES LESS THAN ('2001'),
```

```
PARTITION p_sales_2 VALUES LESS THAN ('2011'),
PARTITION p_sales_3 VALUES LESS THAN (2022));

INSERT INTO sales VALUES ('2001','01','0201','11001',30,500,'0201010011');
INSERT INTO sales VALUES ('2015','11','0101','11001',20,300,'');
INSERT INTO sales VALUES ('2021','10','0101','11001',20,300,'');
INSERT INTO sales VALUES ('2000','12','0102','11001',20,300,'');
INSERT INTO sales VALUES ('2015','03','0102','11001',20,300,'');
INSERT INTO sales VALUES ('2020','05','0101','11001',40,600,'');
COMMIT;


--查看如下语句执行计划，[1,2]表示扫描到第1到第2个分区
EXPLAIN
SELECT a.area_name,
b.branch_name,
s.product,
s.amount
FROM branches b,
area a,
(SELECT branch,
product,SUM(amount) amount
FROM sales
WHERE year>'2005'
GROUP BY branch,product) s
WHERE s.branch=b.branch_no
AND b.area_no=a.area_no;

PLAN_DESCRIPTION
------------------------------------------------------------------
SQL hash value: 3362858118
Optimizer: ADOPT_C

+----+--------------------------------+---------------------+------------+----------+-------------+-------------------------
-------+
| Id | Operation type                 | Name                | Owner      | Rows     | Cost(%CPU)  | Partition info
|
+----+--------------------------------+---------------------+------------+----------+-------------+-------------------------
-------+
|  0 | SELECT STATEMENT               |                     |            |          |             |
|
|  1 |  DISTRIBUTED COORDINATOR       |                     |            |          |             |
|
|  2 |   COL TO ROW                   |                     |            |      1000|      734( 0)|
|
|  3 |    PX N2I REMOTE               | QUEUE_0             |            |      1000|      734( 0)|
|
|* 4 |     HASH JOIN INNER            |                     |            |      1000|      730( 0)|
|
|  5 |      JOIN FILTER USE           |                     |            |    100000|      271( 0)|
|
|  6 |       PART SCAN ALL            |                     |            |    100000|      271( 0)| [0,20]
|
|* 7 |        TABLE ACCESS FULL       | AREA                | SYS        |    100000|      271( 0)|
|
|* 8 |      JOIN FILTER CREATE        |                     |            |      1000|      450( 0)|
|
|* 9 |       PX N2N REMOTE            | QUEUE_1             |            |      1000|      450( 0)|
|
|*10 |        HASH JOIN INNER         |                     |            |      1000|      446( 0)|
|
| 11 |         PART SCAN ALL          |                     |            |    100000|      272( 0)| [0,20]
|
| 12 |          TABLE ACCESS FULL     | BRANCHES            | SYS        |    100000|      272( 0)|
|
| 13 |         RESULT                 |                     |            |      1000|      169( 0)|
|
| 14 |          HASH GROUP            |                     |            |      1000|      169( 0)|
|
|*15 |           PX I2N REMOTE        | QUEUE_2             |            |     33000|      166( 0)|
```

```
|
| 16 |            PART SCAN ITERATOR  |                      |            |     33000|      153( 0)| [1,2]
|
|*17 |             TABLE ACCESS FULL  | SALES                | SYS        |     33000|      153( 0)|
|
+----+--------------------------------+----------------------+------------+----------+-------------+-------------------------
-------+

Operation Information (identified by operation id):
---------------------------------------------------

   2 - Projection: RemoteTable[3][VARCHAR, 60], RemoteTable[3][VARCHAR, 200], RemoteTable[3][CHAR, 5], RemoteTable[3][NUMBER]
   3 - Projection: Tuple[0, 0][VARCHAR, 60], Tuple[0, 1][VARCHAR, 200], Tuple[0, 2][CHAR, 5], Tuple[0, 3][NUMBER]
       PX RemoteInfo: (RANDOM SENDER -> RANDOM RECEIVER : 3->1 [3][4][5]->[2])
   4 - Projection: Tuple[0, 1][VARCHAR, 60], Tuple[1, 1][VARCHAR, 200], Tuple[1, 2][CHAR, 5], Tuple[1, 3][NUMBER]
       Predicate : access(Tuple[0, 0] = Tuple[1, 0])
   5 - Projection: Tuple[0, 0][CHAR, 2], Tuple[0, 1][VARCHAR, 60]
   6 - Projection: Tuple[0, 0][CHAR, 2], Tuple[0, 1][VARCHAR, 60]
   7 - Projection: Tuple[0, 0][CHAR, 2], Tuple[0, 1][VARCHAR, 60]
       Predicate : RUNTIME FILTER(RUNTIME USE(0): "A"."AREA_NO")
   8 - Projection: Tuple[0, 0][CHAR, 2], Tuple[0, 1][VARCHAR, 200], Tuple[0, 2][CHAR, 5], Tuple[0, 3][NUMBER]
       Predicate : RUNTIME FILTER(RUNTIME CREATE(0): Tuple[0, 0])
   9 - Projection: Tuple[0, 0][CHAR, 2], Tuple[0, 1][VARCHAR, 200], Tuple[0, 2][CHAR, 5], Tuple[0, 3][NUMBER]
       PX RemoteInfo: (HASH SENDER -> RANDOM RECEIVER : 3->3 [3][4][5]->[3][4][5])
       Predicate : access(Tuple[0, 0])
  10 - Projection: Tuple[0, 1][CHAR, 2], Tuple[0, 2][VARCHAR, 200], Tuple[1, 1][CHAR, 5], Tuple[1, 2][NUMBER]
       Predicate : access(Tuple[0, 0] = Tuple[1, 0])
  11 - Projection: Tuple[0, 0][CHAR, 4], Tuple[0, 1][CHAR, 2], Tuple[0, 2][VARCHAR, 200]
  12 - Projection: Tuple[0, 0][CHAR, 4], Tuple[0, 2][CHAR, 2], Tuple[0, 1][VARCHAR, 200]
  13 - Projection: Tuple[0, 1][CHAR, 4], Tuple[0, 2][CHAR, 5], Tuple[0, 0][NUMBER]
  14 - Projection: SUM(Tuple[0, 0])[NUMBER], Tuple[0, 1][CHAR, 4], Tuple[0, 2][CHAR, 5]
       Group Expression: (Tuple[0, 1], Tuple[0, 2])
  15 - Projection: Tuple[0, 0][NUMBER, (10, 2)], Tuple[0, 1][CHAR, 4], Tuple[0, 2][CHAR, 5]
       PX RemoteInfo: (HASH SENDER -> RANDOM RECEIVER : 1->3 [3]->[3][4][5])
       Predicate : access(Tuple[0, 1])
  16 - Projection: Tuple[0, 0][NUMBER, (10, 2)], Tuple[0, 1][CHAR, 4], Tuple[0, 2][CHAR, 5]
  17 - Projection: Tuple[0, 5][NUMBER, (10, 2)], Tuple[0, 2][CHAR, 4], Tuple[0, 3][CHAR, 5]
       Predicate : access("SALES"."YEAR" > '2005')
```

predicate

谓词信息，包括：

- access：限定谓词，用于减少结果集的大小，如索引限定谓词、HashJoin限定谓词。
- filter：过滤谓词，对结果集中的每条记录进行过滤运算。
- projection：投影信息，对下层算子的引用。其中Tuple[sourceId, attrId]表示下层算子的第sourceId个数据源的第attrId个投影，两者的序号都是从0开始。

FLASHBACK

通用描述

FLASHBACK用于实现对表的历史数据闪回。本语句只作用HEAP表。

FLASHBACK的使用遵守如下规则：

- FLASHBACK在以下表类型中不允许使用：
  - TABLES THAT ARE PART OF A CLUSTER
  - MATERIALIZED VIEWS
  - ADVANCED QUEUING (AQ) TABLES
  - STATIC DATA DICTIONARY TABLES
  - SYSTEM TABLES
  - REMOTE TABLES
  - OBJECT TABLES
  - NESTED TABLES
  - INDIVIDUAL TABLE PARTITIONS OR SUBPARTITIONS
  - TEMPORARY TABLE
- FLASHBACK不允许使用于在以下闪回的SCN至当前期间内，发生过以下改变了表结构的DDL：
  - UPDATE TABLE、 MOVE TABLE、 TRUNCATE TABLE
  - 表增加约束
  - MODIFY COLUMN、 DROP COLUMN
  - ADD 、DROP、MERGE、SPLIT、COALESCE、TRUNCATE分区或子分区
- FLASHBACK不允许在事务中进行。
- FLASHBACK的表不允许与其他表存在依赖关系。
- FLASHBACK不维护ROWID。
- FLASHBACK仅对当前时刻存在的索引进行维护，即使SCN点该索引还未创建；SCN点已存在的索引在当前时刻不存在时，不会恢复索引。
- FLASHBACK不允许会违反约束的数据。
- FLACKBACK适用于多表的场景下必须全部成功或全部失败。
- FLACKBACK不还原统计信息。

语句定义

flashback::=

```
syntax::= FLASHBACK TABLE table_name (TO (SCN scn|Timestamp timestamp| BEFORE (DROP [RENAME TO new_name]|TRUNCATE)) |
(PARITITION|SUBPARTITION) part_name TO BEFORE TRUNCATE)
```

to scn|timestamp

该语句用于将表的数据闪回至指定的SCN|TIMESTAMP点，此功能无需开启回收站。

通过此功能闪回的数据可能会发生rowid的变化，因此执行前需要开启表的ROW MOVEMENT开关（参考ALTER TABLE中描述语法），否则将返回错误。

当需要闪回的表存在外键约束的情况下，需要在执行前先删除其他表中的外键约束，否则将返回错误。

不支持闪回二级分区。

示例（HEAP表）

```
--开启finance_info的ROW MOVEMENT开关
ALTER TABLE finance_info ENABLE ROW MOVEMENT;

--finance_info表中存在的一条记录
SELECT * FROM finance_info WHERE year='2021' AND month='02';
YEAR  MONTH BRANCH REVENUE_TOTAL  COST_TOTAL   FEE_TOTAL
----- ----- ------ ------------- ----------- -----------
2021  02    0101          37778       33000        6000
```

```
--获取当前时间戳
SELECT SYSTIMESTAMP res FROM dual;
RES
---------------------------------------------
2023-12-17 14:10:28.736908

--删除此条记录并提交
DELETE FROM finance_info WHERE year='2021' AND month='02';
COMMIT;
SELECT * FROM finance_info WHERE year='2021' AND month='02';
YEAR  MONTH BRANCH REVENUE_TOTAL  COST_TOTAL   FEE_TOTAL
----- ----- ------ ------------- ----------- -----------


--利用FLASHBACK闪回历史数据（通过时间戳闪回）
FLASHBACK TABLE finance_info TO TIMESTAMP  TIMESTAMP('2023-12-17 14:10:28.736908');
SELECT * FROM finance_info WHERE year='2021' AND month='02';
YEAR  MONTH BRANCH REVENUE_TOTAL  COST_TOTAL   FEE_TOTAL
----- ----- ------ ------------- ----------- -----------
2021  02    0101           37778       33000        6000

--获取最后一次修改时SCN（通过ROWSCN获取）
SELECT rowscn FROM finance_info WHERE year='2021' AND month='02';
               ROWSCN
---------------------
   408883147271815168

--再次删除记录
DELETE FROM finance_info WHERE year='2021' AND month='02';
COMMIT;
SELECT * FROM finance_info WHERE year='2021' AND month='02';
YEAR  MONTH BRANCH REVENUE_TOTAL  COST_TOTAL   FEE_TOTAL
----- ----- ------ ------------- ----------- -----------


--利用FLASHBACK闪回历史数据(通过SCN闪回)
FLASHBACK TABLE finance_info TO SCN 408883147271815168;
SELECT * FROM finance_info WHERE year='2021' AND month='02';
YEAR  MONTH BRANCH REVENUE_TOTAL  COST_TOTAL   FEE_TOTAL
----- ----- ------ ------------- ----------- -----------
2021  02    0101           37778       33000        6000
```

before

该语句用于将被删除的表闪回至删表之前的状态，或者将被truncate的表的数据闪回，此功能需要开启回收站（修改配置参数RECYCLEBIN_ENABLED为ON）。

drop

该语句用于将被删除的表闪回，包括表结构和数据。

实际情况中，被删除的表名称有可能已经被重建，或被其他对象占用，此时需指定RENAME TO关键字用于为恢复的表重命名，否则将报YAS-02013错误。

示例（HEAP表）

```
ALTER SYSTEM SET RECYCLEBIN_ENABLED=ON;

DROP TABLE finance_info;

SELECT * FROM finance_info;
[1:15]YAS-02012 table or view does not exist

FLASHBACK TABLE finance_info TO BEFORE DROP;
--当finance_info名称已被使用时,需对其重命名
--FLASHBACK TABLE finance_info TO BEFORE DROP RENAME TO finance_info_new;

SELECT * FROM finance_info;
```

```
YEAR  MONTH BRANCH REVENUE_TOTAL  COST_TOTAL   FEE_TOTAL
----- ----- ------ -------------- ----------- -----------
2001  01    0201            2888        2000         300
2021  01    0201           28888       24000        3000
2021  01    0101           38888       34000        4000
2021  02    0101           37778       33000        6000
```

truncate

该语句用于将表truncate的数据闪回。若有多次truncate只能闪回最后一次truncate的数据。

示例（HEAP表）

```
ALTER SYSTEM SET RECYCLEBIN_ENABLED=ON;

TRUNCATE TABLE sales_info_range;

SELECT * FROM sales_info_range;
YEAR  MONTH BRANCH PRODUCT      QUANTITY      AMOUNT SALSPERSON
----- ----- ------ --------- ----------- ----------- -------------

FLASHBACK TABLE sales_info_range TO BEFORE TRUNCATE;

SELECT * FROM sales_info_range;
YEAR  MONTH BRANCH PRODUCT      QUANTITY      AMOUNT SALSPERSON
----- ----- ------ --------- ----------- ----------- -------------
2001  01    0201   11001              30         500 0201010011
2000  12    0102   11001              20         300
2015  11    0101   11001              20         300
2015  03    0102   11001              20         300
2021  10    0101   11001              20         300
2021  05    0101   11001              40         600
```

partition|subpartition

对于某个分区或子分区的truncate，本语句支持对该分区或子分区的数据闪回，此功能需要开启回收站（修改配置参数RECYCLEBIN_ENABLED为ON）。

实际情况中，某个分区或子分区被truncate后可能会新插入数据，此时执行闪回将导致这部分数据被truncate到回收站中。同时若有存在该表被外键依赖的全局索引主键或唯一键则会报错。

对于分区表truncate表无法闪回其分区，同理truncate一级分区也无法单独闪回去其二级分区。空分区在truncate时也会进入回收站可以被闪回。

只支持分区和二级分区的truncate的闪回，不支持分区drop闪回。

示例（HEAP表）

```
ALTER SYSTEM SET RECYCLEBIN_ENABLED=ON;

ALTER TABLE sales_info TRUNCATE PARTITION p_sales_info_1;
ALTER TABLE sales_info TRUNCATE SUBPARTITION  P_SALES_INFO_2_SP_SALES_INFO_1;

SELECT * FROM sales_info PARTITION (p_sales_info_1);
YEAR  MONTH BRANCH PRODUCT      QUANTITY      AMOUNT SALSPERSON
----- ----- ------ --------- ----------- ----------- -------------

INSERT INTO sales_info VALUES ('2021','06','0402','11001',40,600,'');
COMMIT;

FLASHBACK TABLE sales_info PARTITION p_sales_info_1 TO BEFORE TRUNCATE;
FLASHBACK TABLE sales_info SUBPARTITION P_SALES_INFO_2_SP_SALES_INFO_1 TO BEFORE TRUNCATE;
--闪回后上面新增的数据被放入回收站，p_sales_info_list_2分区中只存在回收站上一次的数据
SELECT * FROM sales_info PARTITION (p_sales_info_1);
YEAR  MONTH BRANCH PRODUCT      QUANTITY      AMOUNT SALSPERSON
----- ----- ------ --------- ----------- ----------- -------------
2021  10    0402   11001              20         300
```

GRANT

通用描述

GRANT用于对某一用户或角色授予权限，权限内容包括：

- 系统级权限SYSTEM PRIVILEGE
- 对象级权限OBJECT PRIVILEGE
- 角色ROLE

授予给用户上的权限立即生效。

授予给角色上的权限立即生效，如该角色已被授权给某个用户，则该用户的权限也立即生效。

授予给用户或角色的角色在用户登录时生效，即给用户A授予角色B，授权前已登录的用户A不拥有该角色，A用户再次登录时角色B生效。

YashanDB的权限体系管理具体描述请查看产品安全手册权限管理。

语句定义

grant::=

```
syntax::= grant_object_privilege|grant_system_privilege|grant_role_privilege
```

grant_object_privilege::=

```
syntax::= GRANT (object_privilege|ALL PRIVILEGES) ON [schema "."] table_name TO ((user_name [WITH GRANT OPTION])|role)
```

grant_system_privilege::=

```
syntax::= GRANT (system_privilege|ALL PRIVILEGES) TO ((user_name [WITH ADMIN OPTION])|role)
```

grant_role_privilege::=

```
syntax::= GRANT role TO ((user_name [WITH ADMIN OPTION])|role)
```

grant_object_privilege

该语句用于对用户或角色授予对象级权限，如指定ALL PRIVILEGES表示授予所有对象级权限。

某个对象的对象级权限应该由其所属用户来授予，此外还有这几种用户可以授予：

- 若打开三权分立功能，可以通过拥有SECURITY_ADMIN安全管理员角色的用户授予。
- 若关闭三权分立功能，可以通过拥有SECURITY_ADMIN安全管理员角色或拥有DBA角色的用户授予。
- 可以通过被授权该权限时指定了WITH GRANT OPTION语句的普通用户授予。
- 可以通过拥有GRANT ANY OBJECT PRIVILEGE系统级权限的普通用户授予。

object_privilege

对象级权限名称，所有可授权的对象级权限见OBJECT PRIVILEGE中描述。

table_name|user_name|role

对象|用户|角色名称。

with grant option

将对象级权限授予某一个用户时，指定本语句表示该用户可以再将拥有的对象级权限授予其他的用户或角色。

示例

```
--在用户sales下将表area的SELECT权限授予sales1用户
GRANT SELECT ON area TO sales1 WITH GRANT OPTION;

--此时sales1将拥有表area的SELECT权限，并可以再将此权限授予其他用户
conn sales1/1%2
GRANT SELECT ON sales.area TO sales2;
```

grant_system_privilege

该语句用于对用户或角色授予系统级权限，如指定ALL PRIVILEGES表示授予所有系统级权限。此外还有这几种用户可以授予：

- 若打开三权分立功能，可以通过拥有SECURITY_ADMIN安全管理员角色的用户授予。
- 若关闭三权分立功能，可以通过拥有SECURITY_ADMIN安全管理员角色或拥有DBA角色的用户授予。
- 可以通过被授权该权限时指定了WITH ADMIN OPTION语句的普通用户授予。
- 可以通过拥有GRANT ANY PRIVILEGE系统级权限的普通用户授予。

system_privilege

系统级权限名称，所有可授权的系统级权限见SYSTEM PRIVILEGE中描述。

user_name|role

用户|角色名称。

with admin option

将系统级权限授予某一个用户时，指定本语句表示该用户拥有了对此权限的管理权限，即可以将该系统级权限再授予给其他用户或角色。

示例

```
--将ALTER SYSTEM权限授予sales1用户
GRANT ALTER SYSTEM TO sales1 WITH ADMIN OPTION;

--此时sales1将拥有ALTER SYSTEM权限，并可以再将此权限授予其他用户
GRANT ALTER SYSTEM TO sales2;
```

grant_role_privilege

该语句用于对用户或角色授予角色权限。此外还有这几种用户可以授予：

- 若打开三权分立功能，可以通过拥有SECURITY_ADMIN安全管理员角色的用户授予。
- 若关闭三权分立功能，可以通过拥有SECURITY_ADMIN安全管理员角色或拥有DBA角色的用户授予。
- 可以通过拥有GRANT ANY ROLE的系统级权限的普通用户授予。

该语句不适用于分布式部署。

user_name|role

用户|角色名称。

with admin option

将角色授予某一个用户时，指定本语句表示该用户拥有了对此角色的管理权限，即可以将该角色再授予给其他用户。

> **Note**：
> 授权SYSDBA和SYSOPER角色时，指定WITH ADMIN OPTION无效。

示例（单机部署）

```
--在sales用户下创建角色ya_rol1并授予其area表的SELECT权限
CREATE ROLE ya_rol1;
GRANT SELECT ON area TO ya_rol1;

--将ya_rol1角色权限授予user1用户，则在user1重新连接后，即可以拥有sales.area的SELECT权限。此操作需要sales用户拥有GRANT ANY ROLE权限。
GRANT ya_rol1 TO sales1;
```

```
--在sales用户下创建角色ya_rol2并授予其CREATE TABLE的权限，将ya_rol2授予ya_rol1, 由于ya_rol1已经授予sales1用户，则sales1重新连接后，可拥有ya_rol2角色
CREATE ROLE ya_rol2;
GRANT CREATE TABLE TO ya_rol2;
GRANT ya_rol2 TO ya_rol1;
```

INSERT

通用描述

INSERT用于执行对数据库表的插入数据操作。

单机部署中，YashanDB支持对某一张表插入数据，也支持同时对多张表插入数据。

在INSERT的事务（Transaction）被提交（Commit）之前，其他会话无法查询到这些新插入的数据。在YashanDB里，可以通过打开自动提交（SET AUTOCOMMIT ON）的开关，此时其他会话将可以及时查询到这些数据。

语句定义

insert::=

```
syntax::= INSERT [hint] (single_table_insert|multi_table_insert)
```

single_table_insert**::=**

```
syntax::= single_insert_into_clause (insert_values_clause|subquery) [on_duplicate_clause|returning_clause]
```

single_insert_into_clause**::=**

```
syntax::= INTO table_reference [t_alias] [("(" column_name ")") {"," ("(" column_name ")")}]
```

table_reference**::=**

```
syntax::= [schema "." ] table_name [dblink] [partition_extension_clause]
```

partition_extension_clause**::=**

```
syntax::= PARTITION ("(" partition ")"|FOR "(" (partition_key_value) {","(partition_key_value)} ")")|SUBPARTITION ("("
subpartition ")"|FOR "(" (subpartition_key_value) {","(subpartition_key_value)} ")")
```

insert_values_clause**::=**

```
syntax::= VALUES ("(" (expr|DEFAULT|udt_expr) {","(expr|DEFAULT|udt_expr)} ")")
{"," ("(" (expr|DEFAULT|udt_expr) {","(expr|DEFAULT|udt_expr)} ")")}
```

on_duplicate_clause**::=**

```
syntax::= ON DUPLICATE KEY UPDATE (set_clause) {"," (set_clause)}
```

set_clause**::=**

```
syntax::= column_name "=" (DEFAULT|expr|udt_expr)
```

returning_clause**::=**

```
syntax::= (RETURN|RETURNING) (expr) {"," (expr)} INTO (variable) {"," (variable)}
```

multi_table_insert**::=**

```
syntax::= ALL (multi_insert_into_clause [insert_values_clause])
{" " (multi_insert_into_clause [insert_values_clause])} subquery
```

multi_insert_into_clause**::=**

```
syntax::= INTO table_reference ["(" (column_name) {"," (column_name)} ")"]
```

hint

该语句用于提出给定的方案到优化器（Optimizer），使其按照此方案生成语句的执行计划。查看hint说明。

single_table_insert

对单张表执行INSERT操作的语句。

single_insert_into_clause

该语句用于指定要插入数据的表（包括本地数据库的表或远端表）及列字段，可为表定义一个别名。

column_name

列字段，指定多个时以`,`分隔，也可全部省略，表示按表的列字段定义顺序插入所有列项。而在指定列字段时，对于非空且没有DEFAULT值的列字段必须指定。

在列字段上定义的所有约束，对插入的数据也会起到约束作用，如违反约束则插入数据失败并提示错误。

示例

```
--area表的DHQ列字段为非空项，提示错误
INSERT INTO area VALUES ('01','0000','');
YAS-04006 cannot insert NULL value to column DHQ

--area表的DHQ列字段非空且定义了DEFAULT值，插入成功
INSERT INTO area (area_no) VALUES ('07');

--branches表的BRANCH_NAME列字段非空且无DEFAULT值，则必须指定该列插入，否则提示错误
INSERT INTO branches (branch_no) VALUES ('0202');
YAS-04006 cannot insert NULL value to column BRANCH_NAME
```

table_reference

该语句用于指定要插入数据的表（包括本地数据库的表或远端表）或表分区名称。

对于分区表，如未显式指定分区对象，由系统根据分区项字段的值判断要插入的表分区。而在显式指定时：

- 当分区表类型为RANGE时，新插入记录的分区项字段值请注意不要超出其所在分区的界值，否则会插入失败并提示错误。
- 当分区表类型为LIST时，新插入记录的分区项字段值请注意数据值在其所在分区的列表范围内，否则会插入失败并提示错误。

示例（单机、共享集群部署）

```
--sales_info_range为一张范围分区表，已有p_sales_info_range_1（VALUES LESS THAN('2011')）、p_sales_info_range_2（VALUES LESS
THAN('2021')）和p_sales_info_range_3（VALUES LESS THAN('2031')）三个分区

--未指定分区时，按分区项判断界值，插入数据成功
INSERT INTO sales_info_range VALUES ('2012','01','0103','11002',5,100,'');

--插入超过分区最大值的数据失败
INSERT INTO sales_info_range VALUES ('2040','01','0103','11002',5,100,'');
YAS-02115 partition key does not map to any partition

--'2012'与p1分区的界值冲突，插入数据失败
INSERT INTO sales_info_range PARTITION (p_sales_info_range_1) VALUES ('2012','01','0103','11002',5,100,'');
YAS-02120 the partition number is invalid or out-of-range
```

partition_extension_clause
该语句用于指定分区或子分区。

指定分区或子分区的方式均有如下两种：

- 根据名称
- 根据提供的键值

PARTITION (partition)

该语句用于根据名称直接获取分区对象。

示例（单机部署）

```
INSERT INTO sales_info_range PARTITION (p_sales_info_range_3) VALUES ('2029','01','0103','11002',5,100,'');
```

PARTITION FOR (partition_key_value)

该语句用于根据提供的键值（Key Value）获取分区对象。此时会将键值与表分区界值比较，计算得到分区对象，键值可为界值范围内的任意值。键值的个数须与分区列个数对应，多项用`,`分隔。

示例（单机部署）

```
INSERT INTO sales_info_range PARTITION FOR('2015') VALUES ('2012','01','0103','11002',5,100,'');
```

SUBPARTITION (subpartition)

该语句用于根据名称直接获取子分区对象。

分布式部署中，由于一级分区为hash分配，只有在用户知晓数据会落入某个分区（例如可以通过该分区中已存在的对应分区键数据进行判断）时，才可使用本方式指定分区插入数据，否则可能出现数据插入失败情况。

示例（单机/共享集群部署）

```
--获得sales_info表的子分区名称
SELECT partition_name,subpartition_name
FROM DBA_TAB_SUBPARTITIONS
WHERE table_name='SALES_INFO';
--以下输出以单机为例
PARTITION_NAME               SUBPARTITION_NAME
---------------------------- ---------------------------------------
P_SALES_INFO_1               P_SALES_INFO_1_SP_SALES_INFO_1
P_SALES_INFO_1               P_SALES_INFO_1_SP_SALES_INFO_2
P_SALES_INFO_1               P_SALES_INFO_1_SP_SALES_INFO_3
P_SALES_INFO_2               P_SALES_INFO_2_SP_SALES_INFO_1
P_SALES_INFO_2               P_SALES_INFO_2_SP_SALES_INFO_2
P_SALES_INFO_2               P_SALES_INFO_2_SP_SALES_INFO_3
P_SALES_INFO_3               P_SALES_INFO_3_SP_SALES_INFO_1
P_SALES_INFO_3               P_SALES_INFO_3_SP_SALES_INFO_2
P_SALES_INFO_3               P_SALES_INFO_3_SP_SALES_INFO_3

--选择其中一个子分区指定插入数据
INSERT INTO sales_info SUBPARTITION(P_SALES_INFO_1_SP_SALES_INFO_1)
VALUES ('2002','01','0402','11002',5,100,'');
```

SUBPARTITION FOR (subpartition_key_value)

该语句用于根据提供的键值（Key Value）获取子分区对象。此时会将键值与表分区界值比较，计算得到子分区对象，键值可为界值范围内的任意值。键值的个数须与分区列和子分区列个数对应，多项用`,`分隔。

示例

```
INSERT INTO sales_info SUBPARTITION FOR('0103','2018')
VALUES ('2012','01','0103','11002',5,100,'');
```

subquery

该语句用于指定要插入的数据从一个子查询中获得。插入对象的列项需与子查询的select_list列项按顺序一一对应。

查看SELECT文档中的subquery部分可获得子查询相关详细信息。

示例

```
INSERT INTO sales_info SELECT '2021','01',a.branch_no,'11002',100,1000,'' FROM branches a WHERE area_no ='02';
```

insert_values_clause

该语句用于按single_insert_into_clause中定义的列字段一一指定对应的数据值。

YashanDB支持按如下语句对列字段赋值：

- expr表达式
- DEFAULT
- 对于UDT列字段，通过对象初始化方法赋值，详见用户自定义类型中描述。

其中，当使用DEFAULT对列字段赋值时，如对应的列上已定义了DEFAULT值，则插入的数据为该DEFAULT值，否则为NULL，基于这个规则，如对应的列存在非空约束，则插入数据失败。

当表达式的结果与列字段定义的数据类型不一致时，系统会先进行数据类型转换，转换失败时则返回错误。

VALUES关键字后可以跟多组值，代表一条SQL语句可以插入多组值，这组值的上限为32768个，超过时将返回YAS-04816错误。

示例

```
-area表上的DHQ字段已定义了DEFAULT值'ShenZhen',则按此值插入数据成功
INSERT INTO area VALUES ('08',33005,DEFAULT);

--branches表上的BRANCH_NAME字段非空且无DEFAULT值，则按NULL值插入数据失败
INSERT INTO BRANCHES VALUES ('0202',DEFAULT,'','02');
YAS-04006 cannot insert NULL value to column BRANCH_NAME

--area表的area_no字段插入两组值，插入成功
INSERT INTO area (area_no) VALUES ('20'),('21');

ROLLBACK;
```

on_duplicate_clause

该语句用于指定当插入的数据导致表的唯一约束（包括唯一索引）冲突时，不报错，而是对冲突行使用set_clause执行更新。

当表中存在多个唯一约束时，系统将只针对第一个约束项（按约束项的声明顺序确定）所产生的冲突行进行本操作处理。

需注意的是，在分布式部署中不能对分布表一级分区键执行本语句操作，否则返回失败。

set_clause

指定更新的字段和值，指定值的规则和要求与insert_values_clause一致，不满足要求时更新失败，系统提示错误。

示例1（单机/共享集群部署）

```
--为area表的DHQ字段增加唯一约束，且在area_no上已存在主键约束
SELECT * FROM area;
AREA_NO AREA_NAME        DHQ
------- ---------------- -------------
01      华东             Shanghai
02      华西             Chengdu
```

```
03      华南            Guangzhou
04      华北            Beijing
05      华中            Wuhan

ALTER TABLE area ADD CONSTRAINT c_area_unique UNIQUE (DHQ);

--插入的数据导致DHQ唯一约束冲突时，通过set_clause将冲突值修改，更新冲突行
INSERT INTO area VALUES('08',33005,'Shanghai') ON DUPLICATE KEY UPDATE DHQ = 'XiAn';
SELECT * FROM area;
AREA_NO AREA_NAME        DHQ
------- ---------------- -------------
01      华东            XiAn
02      华西            Chengdu
03      华南            Guangzhou
04      华北            Beijing
05      华中            Wuhan

--area表的area_no和DHQ列均存在唯一约束，同时冲突时set_clause只作用于第一项约束的记录
INSERT INTO area VALUES('03',33006,'Chengdu') ON DUPLICATE KEY UPDATE DHQ = 'Shenzhen';
SELECT * FROM area;
AREA_NO AREA_NAME        DHQ
------- ---------------- -------------
01      华东            XiAn
02      华西            Chengdu
03      华南            Shenzhen
04      华北            Beijing
05      华中            Wuhan
```

示例2（分布式部署）

```
--创建area0表，并为area表的DHQ字段增加唯一约束
CREATE DUPLICATED TABLE area0
(area_no CHAR(2) NOT NULL,
 area_name VARCHAR2(60),
 DHQ VARCHAR2(20) DEFAULT 'ShenZhen' NOT NULL UNIQUE);
INSERT INTO area0 VALUES ('01','华东','Shanghai');
INSERT INTO area0 VALUES ('02','华西','Chengdu');
INSERT INTO area0 VALUES ('03','华南','Guangzhou');
INSERT INTO area0 VALUES ('04','华北','Beijing');
INSERT INTO area0 VALUES ('05','华中','Wuhan');

--插入的数据导致DHQ唯一约束冲突时，通过set_clause将冲突值修改，更新冲突行
INSERT INTO area0 VALUES('08',33005,'Shanghai') ON DUPLICATE KEY UPDATE DHQ = 'XiAn';

SELECT * FROM area0;
AREA_NO AREA_NAME                    DHQ
------- ---------------------------- ---------------
01      华东                         XiAn
02      华西                         Chengdu
03      华南                         Guangzhou
04      华北                         Beijing
05      华中                         Wuhan
```

returning_clause

本语句表示在插入数据后返回一个结果集，并将结果集赋值给指定的变量。本语句只可与insert_values_clause结合使用。

expr

指定结果列，多列以`，`分隔，每一列为一个通用表达式，但不能为伪列、聚集函数和窗口函数。

variable

与expr一一对应的赋值变量，根据运用的语句不同，variable可以为一个已声明的变量，或者为一个绑定参数。

本语句可以在过程体或者驱动客户端程序中使用，PL手册的DML Statement和EXECUTE Statement中分别提供了该语句作为静态SQL和动态SQL在过程体中的使用示例。

multi_table_insert

本语句适用于单机部署中的行存表，对一张或多张表同时进行INSERT操作，多张表的操作以空格分隔。

对所有表均可以指定按值插入（insert_values_clause），或是按子查询插入（subquery）。但是整个语句中最多只能有一个子查询，且subquery语句必须位于整个语句最后面。

该语句要求最后必须有一个子查询，当所有表都使用insert_values_clause时，可以在最后放入SELECT * FROM DUAL语句作为subquery。

若子查询返回结果集为空，则所有表都不执行插入。

multi_insert_into_clause

除不可以为待插入的表指定别名外，该语句与single_insert_into_clause保持一致。

多表插入时，会按照表的出现次数申请对应数量的cursor，可申请的最多cursor个数为CURSOR_POOL_SIZE / ((2 * (DB_BLOCK_SIZE + 6000)) + 2920)。

示例（单机HEAP表）

```
--语句后面必须有subquery
INSERT ALL INTO area VALUES('00','unknown','unknown')
           INTO branches VALUES('0002','南山','00','----')
           INTO branches VALUES('0003','福田','00','----')
           SELECT * FROM DUAL;

--subquery必须放在语句最后面
ROLLBACK;
INSERT ALL INTO area VALUES('00','unknown','unknown')
           INTO branches
           INTO branches VALUES('0003','福田','00','----')
           SELECT '0002','南山','00',b.address FROM branches b WHERE b.branch_no='0101';

--当subquery查询结果为空时，所有数据均不会插入
ROLLBACK;
INSERT ALL INTO area VALUES('00','unknown','unknown')
           INTO branches
           INTO branches VALUES('0003','福田','00','----')
           SELECT '0002','南山','00',b.address FROM branches b WHERE b.branch_no='0109';
```

LOAD DATA

通用描述

LOAD DATA用于将CSV文件中的数据加载到数据库的物理表中。

本语句为SQL级别的数据加载（SQL LOADER），等同于对目标表执行DDL操作，因此需满足目标表上已定义的一切约束限制；且也会对目标表加行锁，需确保在执行数据加载前要操作的行上无未提交事务，否则会由于等待行锁而导致加载任务被挂起。

本语句在执行初始阶段会执行一次事务提交，以避免在同一事务中与执行本语句之前的其它语句产生事务等待而导致事务挂起。

分布式部署中用户无法执行本语句。

导入数据的事务提交

在加载过程中，SQL LOADER的事务提交以存放CSV数据的缓存区为单位。当存放CSV数据的缓存区被存满之后，执行一次当前事务的提交。

CSV数据缓存区以一条完整的数据行开始，以一条完整的数据行结束，缓存区中的数据长度视具体情况而定，但不超过2M字节。

若缓存区中存在一行数据插入失败，则整个缓存区数据插入失败，回滚当前事务。

设置nologging导入

如果是在数据迁移场景通过本语句执行数据导入，通过在options参数中指定nologging为true，可以实现nologging导入。

只建议在数据迁移场景设置nologging属性，且需要关注如下事项：

- 主备环境不可启用nologging，需要在完成数据导入后再建备库。
- 使用nologging导入后，执行一次全量checkpoint可以保证数据持久性，否则宕机后可能导致数据丢失。
- 当表的属性为nologging，同时发生宕机时，重启后该表将被置为corrupted，只能通过truncate表进行恢复，且需要重新执行上述导入过程以恢复数据。
- 如果一个事务失败，该事务内所有执行过插入数据操作的nologging表都会被标记为corrupted。
- nologging属性的LSC表使用bulkload模式导入数据时，不受nologging属性影响（性能无变化，失败也不会被标记为corrupted）。
- 当表为nologging时，不能对其执行update和delete操作，导入操作完成后需及时将其修改为logging。
- 设置表状态为nologging属于DDL操作，会加表锁。如果并发执行DML语句，可能导致DML语句失败。
- 当表为nologging，或使用nologging方式导入时，在违反约束条件时可能出现容错失败的现象，具体受违反约束的数据在所有数据中的位置影响。

导入环境准备

用户实际配置值应大于该值，建议导入结束后，依据在线业务的要求重新调整这些值。

配置参数项	建议操作
LARGE_POOL_SIZE	调整为参数允许范围内的最大值，之后导入过程如出现no free blocks in large pool错误，表示该值仍不能满足导入资源要求，此时需： 1.调整导入表，减少导入heap分区表的分区总个数。 2.降低导入时的degree_of_parallelism参数后重新导入。
WORK_AREA_STACK_SIZE	reader线程数 = degree_of_parallelism / (decoder_thread_times + 1)，向上取整。 decoder线程数 = degree_of_parallelism - reader线程数。 SIZE = 512KB + 126KB * 表个数 * decoder线程数。
WORK_AREA_POOL_SIZE	MIN_SIZE = 线程数量 * 表数量 * 分区数量 * 256B + reader线程数 * decoder线程数 * 1KB。

语句定义

load data::=

```
syntax::= LOAD DATA [option_clause]
(input_file_clause [bad_file_clause] [discard_file_clause])
{" " (input_file_clause [bad_file_clause] [discard_file_clause])}
table_clause
```

option_clause::=

```
syntax::= OPTIONS "(" [PARAMETER_NAME "=" PARAMETER_VALUE] ")"
```

input_file_clause::=

```
syntax::= INFILE input_file_path
[FIELDS file_type]
[WITH EMBEDDED|WITHOUT EMBEDDED]
[FIELDS TERMINATED BY fields_terminated_char]
[OPTIONALLY ENCLOSED BY enclosed_by_char]
```

bad_file_clause::=

```
syntax::= BADFILE bad_file_path
```

discard_file_clause::=

```
syntax::= [DISCARDFILE discard_file_path] [(DISCARDS|DISCARDMAX) discard_number]
```

table_clause::=

```
syntax::= [load_type] (INTO TABLE ([schema "."] table_name) [when_clause] [TRAILING NULLCOLS] column_clause)
{" " (INTO TABLE ([schema "."] table_name) [when_clause] [TRAILING NULLCOLS] column_clause)} [directory_clause]
```

load_type::=

```
syntax::= INSERT|APPEND
```

when_clause::=

```
syntax::= WHEN load_condition_clause {AND load_condition_clause}
```

load_condition_clause::=

```
syntax::= ["("] (table_column_name|"(" csv_column_order ")") compare_oper "'column_value'" [")"]
```

compare_oper::=

```
syntax::= "="|"!="|"<>"
```

column_clause::=

```
syntax::= "(" (normal_column_clause|filler_column_clause|lob_column_clause|lls_column_clause)
{"," (normal_column_clause|filler_column_clause|lob_column_clause|lls_column_clause)} ")"
```

normal_column_clause::=

```
syntax::= table_column_name [COLUMN "(" csv_column_order ")"] [nullif_clause]
```

nullif_clause::=

```
syntax::= NULLIF load_condition_clause {AND load_condition_clause}
```

filler_column_clause**::=**

```
syntax::= filler_column_name FILLER [COLUMN "(" csv_column_order ")"]
```

lob_column_clause**::=**

```
syntax::= table_column_name LOBFILE "(" filler_column_name ")" [terminated_by_eof] [nullif_clause]
```

lls_column_clause**::=**

```
syntax::= table_column_name [COLUMN "(" csv_column_order ")"] [nullif_clause] LLS
```

directory_clause**::=**

```
syntax::= directory split_file_path
```

option_clause

该语句让用户指定加载操作的相关选项，可省略，即不由用户指定任何选项，等同于OPTIONS()。

COMMIT_ROWS

决定事务的提交频率，假设其值为n，解码数据文件中n条数据，提交一次事务。

parameter_value须为一个范围在[1,4294967295]之间的整数，默认值为4096。

DECODER_THREAD_TIMES

决定线程的分配比，在yasldr客户端导入时意为READER线程与SENDER线程的比值，在LOAD DATA导入时意为READER线程和DECODER线程的比值，两种线程的总和不超过DEGREE_OF_PARALLELISM。

parameter_value须为一个范围在[1,255]之间的整数，默认值为7。

DEGREE_OF_PARALLELISM

指定数据加载任务的并行度，服务端模式受参数MAX_PARALLEL_WORKERS限制，若degree_number大于该参数，以该参数为准。

parameter_value须为一个范围在[2,256]之间的整数，默认值为8，该数值代表了用户期望的并行度值，系统会结合用户期望值、硬件环境、数据文件大小等计算实际的并行度值，并使加载任务按实际值并发运行。

ENABLE_BULK

指定HEAP表和LSC表的导入模式。

parameter_value须为TRUE或FALSE，默认值为FALSE。

当其值为TRUE时，HEAP表使用bcp模式导入，LSC表使用bulkload模式导入。

ENABLE_DEDUP

指定LSC表的冲突处理方式。

该选项必须与ENABLE_BULK选项同时使用，parameter_value须为TRUE或FALSE，默认值为FALSE。

当其值为TRUE时，LSC表对导入过程中产生唯一约束冲突的数据会进行去重处理。

ERRORS

指定数据导入的容错数据条数上限，当数据错误条数达到上限时，终止数据导入。多个文件导入时，错误数据条数累加计算。

parameter_value须为一个范围在[0,4294967295]之间的整数，默认值为50。

NOLOGGING

当其值为TRUE，对于语句中声明的状态为LOGGING的表，会将其转为NOLOGGING进行导入，导入结束后，将恢复表的LOGGING状态；当其值为FALSE，不对表的LOGGING状态做操作。

若表的初始状态为LOGGING，导入完成后需恢复表的LOGGING状态，机制为异步，在后台进行表的LOGGING状态转换，该行为可能失败，转换结果通过运行日志记载，详情参见LOGGING ASYNC动作。

parameter_value需为TRUE或FALSE，默认值为FALSE。

NULL_LOB_FORMAT_SIZE

值为1，表示特定的LOB型NULL值表示，该模式下会组织成4B全F，只在LLS导入适配下使用；值为0，表示正常的LOB型NULL值表示，该模式下会组织成8B全F，是一种通用的LOB型NULL表示方式。

parameter_value只有0或1，其他值无效，默认值为0。

RUN_LEVEL

指定程序执行的任务类型，可指定SPLIT或SPLIT_TO_PART模式。

RUN_LEVEL选项仅在使用yasldr工具进行数据导入时实现，无法直接通过LOAD DATA语句指定该选项，具体操作可参考yasldr使用指导章节描述。

parameter_value须为SPLIT或SPLIT_TO_PART，不指定RUN_LEVEL情况下默认不进行文件拆分操作。

LOB_PATH_RELATIVE

指定进行文件拆分时，outline lob导入形式的LOB文件路径类型。

parameter_value须为TRUE或FALSE，默认值为FALSE。

- 如果指定为TRUE，则进行文件拆分时，仅支持LOB文件指定为相对路径。若指定为绝对路径，执行语句会报错。
- 如果指定为FALSE，则进行文件拆分时，LOB文件可以指定为相对路径或绝对路径。若指定为绝对路径，yasboot不会将数据拷贝到远端节点。

LOG

指定生成日志文件，记录执行数据加载任务的过程中过程信息，包括数据导入失败原因，导入成功条数等信息，与input_file_clause语句指定的导入文件相关。无论是否指定本参数，均会生成日志文件，省略则默认在首个导入文件目录下生成与该文件同名，后缀为log的日志文件。

parameter_value支持相对路径及绝对路径，仅支持指定到文件名。

> **Note**:
> YashanDB数据导入生成日志文件的相关参数有SILENT、STATS和LOG，三种参数的关系如下：
> - SILENT为TRUE时，STATS和LOG参数失效，此时不会生成导入日志、错误数据文件和过滤数据文件。
> - SILENT为FALSE时：
>   - STATS为TRUE，日志文件内包含统计信息。
>   - STATS为FALSE，日志文件内不包含统计信息，仅包含过程信息。

示例（单机、共享集群部署）

```
--自行创建area_log.csv文件，文件位于/data目录下，文件内容为:
1|load|101|loadbranch|1
2|load|201|loadbranch|2

LOAD DATA OPTIONS(DEGREE_OF_PARALLELISM=2,LOG='/data/area_log.log')
INFILE '/data/area_log.csv' fields terminated BY '|'
INTO TABLE area(area_no,area_name);

--日志文件内容
YashanDB Loader Release 23.1.0.200 x86_64 ff1fe7a
Production on 2023-08-24 04:34:29.409

Table AREA:
column_name                                                      infile_column
AREA_NO                                                          1
AREA_NAME                                                        2
DHQ                                                              NULL
```

```
Table AREA:
  2 Rows successfully loaded.
  0 rows not loaded due to data errors.
  0 Rows not loaded because all WHEN clauses were failed.
  0 Rows not loaded because all fields were null.

Total logical records read: 2
Total logical records rejected: 0
Total logical records discarded: 0

Elapsed time was: 00:00:00.07
```

SILENT

指定导入过程中是否生成记录文件，包括日志文件、错误数据文件及过滤数据文件。当其值为TRUE，表示不生成记录文件；值为FALSE，表示生成记录文件。

parameter_value须为TRUE或FALSE，默认值为FALSE。

SINGLE_PART

指定导入文件中的数据是否属于同一个分区，如果属于同一分区，将简化分区计算提升性能，指定为TRUE时，请确保导入文件中的数据属于相同分区。

parameter_value须为TRUE或FALSE，默认值为FALSE。

本参数仅支持于yasldr工具中指定。

STATS

指定是否打印导入过程中的统计信息。值为TRUE时，会将导入过程中的统计信息打印输出到日志中；值为FALSE时，不会打印统计信息。打印输出信息的日志会保存至导入数据文件所在的路径中，且与该数据文件同名，文件格式为log格式。

parameter_value须为TRUE或FALSE，默认值为FALSE。

具体打印的统计信息如下：

- 导入对象
- READER、DECODER线程的数量、时间和内存
- 存储的时间及内存
- 总时间及内存

示例（单机、共享集群部署）

```
--自行创建area_stats.csv文件，文件位于/data目录下，文件内容为:
1|load|101|loadbranch|1
2|load|201|loadbranch|2

LOAD DATA OPTIONS(DEGREE_OF_PARALLELISM=2,STATS=TRUE)
INFILE '/data/area_stats.csv' fields terminated BY '|'
INTO TABLE area(area_no,area_name);

--日志内容节选
YashanDB Loader Release 23.1.0.100 x86_64 1a34179
Production on 2023-07-31 20:18:59.849

Table AREA:
column_name                                                      infile_column
AREA_NO                                                          1
AREA_NAME                                                        2
DHQ                                                              NULL

In this Task, degree of parallelism is 2.
The number of reader is 1.
The number of decoder is 1.
```

TRIM

表示对CSV数据文件中空格的处理情况。LDRTRIM表示修剪数据列起始的空格、制表符，NOTRIM表示不对数据列起始的空格、制表符做处理。对于有包围符的数据，包围符内的字符均视为数据，不受该参数约束。

parameter_value须为LDRTRIM或NOTRIM，默认值为LDRTRIM。

示例（单机、共享集群部署）

```
--自行创建area.csv文件，文件位于/data目录下，文件内容为:
8|load|801|loadbranch|8
9|load|901|loadbranch|9

LOAD DATA OPTIONS(DEGREE_OF_PARALLELISM=2,TRIM=NOTRIM)
INFILE '/data/area.csv' fields terminated BY '|'
INTO TABLE area(area_no,area_name);
```

input_file_clause

该语句用于指定导入文件的相关信息，可同时指定多个导入文件，以空格分隔。

一个数据加载命令最多可指定250个文件，且同一文件不可重复指定。

导入文件的数据内容格式为：

列1数据 分隔符 列2数据 分隔符 ...

其中，每个文件内数据的最大列数为4096，且多个文件的同一位置列代表相同含义，单行数据长度上限为63KB。

数据加载对文件内数据的读取规则为：

- 每一行代表要导入的一条记录。
- 数据列与目标表列字段的映射关系由column_clause语句决定。
- 当存在未被映射到的数据列时，该数据列丢弃。
- 当一个数据列以双引号开头时，该双引号将被识别为包围符，此时要求：
  1. 数据列必须以双引号结尾。
  2. 对于中间出现的普通双引号，必须在其前面再加上一个双引号进行转义。
- 当数据列未以双引号开头时，后续所有的双引号都被认为是普通双引号，无需转义。

示例（单机、共享集群部署）

```
--创建含有JSON字段的customer_intro表
CREATE TABLE customer_intro (id INT, intro JSON);

--在/data目录下创建customer.csv文件，包含如下数据，其中的json数据包含双引号时需转义
1,"[755,false,null,""city"",{""no"":1},[12]]"
2,""""""
3,"{""city"":""shenzhen"",""no"":2}"
4,""shenzhen,2""

--执行加载语句
LOAD DATA
INFILE '/data/customer.csv'
INTO TABLE customer_intro (id,intro);

SELECT * FROM customer_intro;

          ID INTRO
------------ ----------------------------------------------------------------
           1 [755,false,null,"city",{"no":1},[12]]
           2 ""
           3 {"no":2,"city":"shenzhen"}
           4 "shenzhen,2"
```

input_file_path

导入文件的路径及名称，支持指定绝对和相对路径。

指定的导入文件必须存在，且系统用户必须拥有对其的读取权限，否则加载任务报错退出。

可以指定一个空数据文件，不影响加载任务的继续运行。

fields

指定导入文件的类型，可省略，则默认为CSV文件。

file_type的值只可以为CSV，否则加载任务报错退出。

with embedded|without embedded

指定换行符是否可被包围符包围作为数据，不指定本语句将默认为WITHOUT EMBEDDED，即换行符不可作为数据。

fields terminated by

指定导入数据列的分隔符，不指定时默认将逗号作为分隔符。

fields_terminated_char支持三种表示方法：

1. 单字节字符表示（逗号、竖杠号、分号、水平制表符）。
2. 十六进制表示（0x01、0x02、0x03、0x04、0x05、0x06、0x07、0x08、0x0B、0x0C、0x0D、0x0E、0x0F、0x10、0x11、0x12、0x13、0x14、0x15、0x16、0x17、0x18、0x19、0x1A、0x1B、0x1C、0x1D、0x1E、0x1F）。
3. 整数表示（1、2、3、4、5、6、7、8、11、12、13、14、15、16、17、18、19、20、21、22、23、24、25、26、27、28、29、30、31）。

示例（单机、共享集群部署）

```
--创建表loadterm
CREATE TABLE loadterm(c1 VARCHAR(10),c2 VARCHAR(10),c3 VARCHAR(10),c4 VARCHAR(10), c5 VARCHAR(10));

--自行创建loadterm.csv，文件位于/data目录下，文件内容为：
8|load|801|loadbranch|8
9|load|901|loadbranch|9

LOAD DATA OPTIONS(DEGREE_OF_PARALLELISM=2)
INFILE '/data/loadterm.csv' FIELDS csv
WITH EMBEDDED FIELDS TERMINATED BY '|'
OPTIONALLY ENCLOSED BY '"'
INSERT INTO TABLE loadterm(c1,c2,c3,c4,c5);

--更改loadterm.csv文件内容为：
"1""2""3""4""5"
"6""7""8""9""10"

LOAD DATA OPTIONS(DEGREE_OF_PARALLELISM=2)
INFILE '/data/loadterm.csv' FIELDS csv
WITH EMBEDDED FIELDS TERMINATED BY 0x01
OPTIONALLY ENCLOSED BY '"'
INSERT INTO TABLE loadterm(c1,c2,c3,c4,c5);

--更改loadterm.csv文件内容为：
"1" "2" "3" "4" "5"
"6" "7" "8" "9" "10"

LOAD DATA OPTIONS(DEGREE_OF_PARALLELISM=2)
INFILE '/data/loadterm.csv' FIELDS csv
WITH EMBEDDED FIELDS TERMINATED BY 9
OPTIONALLY ENCLOSED BY '"'
INSERT INTO TABLE loadterm(c1,c2,c3,c4,c5);
```

optionally enclosed by

指定导入数据的包围符，只可以指定为双引号，不指定时系统默认为按双引号包围。

示例（单机、共享集群部署）

```
LOAD DATA OPTIONS()
INFILE '/data/area.csv' FIELDS csv
```

```
WITH EMBEDDED FIELDS TERMINATED BY '|'
OPTIONALLY ENCLOSED BY '"'
INSERT INTO TABLE area(area_no,area_name);
```

bad_file_clause

该语句用于指定导入文件的存放错误数据的地址，与input_file_clause语句指定的导入文件相关，其中写入错误数据条数受ERRORS参数限制，无论是否指定该语句，只要导入错误数据就会生成错误数据文件，省略则默认在导入文件目录下生成与导入文件同名，后缀为bad的数据文件，存在同名文件时会覆盖该文件。

支持相对路径及绝对路径，支持指定到目录及文件名，若指定到目录，则在指定目录下生成与导入文件同名，后缀为bad的数据文件，存在同名文件时会覆盖该文件。

错误数据不会被导入至数据库中，下述数据会被写入错误数据文件中：

- 类型转换失败的数据。
- 违反约束的数据。
- 不符合CSV格式的数据。
- 未命中分区的数据。

示例（单机、共享集群部署）

```
--自行创建badfile.csv和badfile.csv1文件，文件位于/datav目录中，文件内容均为:
"1","2"
"nihao"
"hello"

--创建表bad_load表
CREATE TABLE bad_load(c1 INT,c2 INT);

--执行导入命令
LOAD DATA OPTIONS(DEGREE_OF_PARALLELISM=3)
INFILE '/data/badfile.csv' FIELDS TERMINATED BY ','
BADFILE '/data/badfile.bad'
INSERT INTO TABLE bad_load(c1,c2);

--由于"nihao"和"hello"无法转换成int类型数据，如上两种数据会被记录在错误数据文件中，于终端中执行如下命令查看badfile.bad数据文件内容
$ cat badfile.bad
"nihao"
"hello"

--导入多个数据文件
LOAD DATA OPTIONS(DEGREE_OF_PARALLELISM=3)
INFILE '/data/badfile.csv' FIELDS TERMINATED BY ','
BADFILE '/data/badfile.bad' INFILE '/data/badfile.csv1' FIELDS TERMINATED BY ','
BADFILE '/data/badfile1.bad'
INSERT INTO TABLE bad_load(c1,c2);
```

discard_file_clause

该语句用于指定导入文件的存放过滤数据的地址，与input_file_clause语句指定的导入文件相关，可省略，省略则默认不创建过滤数据文件。如已指定了本语句，但导入数据不满足对应条件，仍不生成过滤数据文件。

支持相对路径及绝对路径，支持指定到目录及文件名，若指定到目录，则在指定目录下生成与导入文件同名，后缀为dsc的数据文件。

过滤数据不会被导入至数据库中，下述数据会被写入过滤数据文件中：

- 不满足when子句的语句。
- 整行映射均为NULL的数据。

discards|discardmax

用于指定数据导入的过滤上限，当数据被过滤的数量达到上限时，终止数据导入，可省略，省略则默认为UINT64_MAX。

如未指定discard_file_clause语句但指定了DISCARDS|DISCARDMAX关键字，将在导入文件目录下生成与导入文件同名，后缀为dsc的数据文件，存在同名文件时会覆盖该文件。

示例（单机、共享集群部署）

```
--自行创建discardfile.csv和discardfile1.csv，文件位于/data目录中，文件内容均为:
"3","4"
""
""

--创建表discard_load
CREATE TABLE discard_load(c1 INT,c2 INT);

--执行导入命令
LOAD DATA OPTIONS(DEGREE_OF_PARALLELISM=3)
INFILE '/data/discardfile.csv' FIELDS TERMINATED BY ','
DISCARDFILE '/data/discardfile.dsc'
INSERT INTO TABLE discard_load(c1,c2);

--由于导入文件中的第二行数据均映射为NULL，该行数据会被记录在过滤数据文件中，于终端中执行如下命令查看discardfile.dsc数据文件内容
$ cat discardfile dsc
""
""

--导入多个数据文件
LOAD DATA OPTIONS(DEGREE_OF_PARALLELISM=3)
INFILE '/data/discardfile.csv' FIELDS TERMINATED BY ','
DISCARDFILE '/data/discardfile.dsc'
INFILE '/data/discardfile1.csv' FIELDS TERMINATED BY ','
DISCARDFILE '/data/discardfile1.dsc'
INSERT INTO TABLE discard_load(c1,c2);
```

table_clause

该语句用于指定导入目标表的相关信息，可同时指定多个导入目标表，以空格分隔。

指定多个导入目标表时，这些表必须具有相同的表类型，例如均为HEAP表。

一个数据加载命令最多可指定16个目标表，且同一目标表不可重复指定。

数据加载对导入多个目标表的主要规则为：

- 由于格式错误、映射不匹配等原因导致某一个目标表无法执行导入时，所有的目标表都将无法导入。
- 在所有目标表均可以执行导入时，多个目标表的导入事务独立，即某一个表导入失败并不会导致所有表的导入失败。
- 导入存在外键关联的两个目标表时，应该将父表放在前面。
- 当未明确以COLUMN(csv_column_order)指定目标表列与导入文件数据列的映射关系时，多个目标表按顺序与导入文件的数据列一一映射。

示例（单机、共享集群部署）

```
--area表与branches表为两张HEAP表，其中area.area_no为branches.area_no的父键
--自行创建area.csv文件，文件位于/data目录下，文件内容为：
12|load|1201|loadbranch|12
13|load|1301|loadbranch|13

--数据加载命令为：
LOAD DATA
INFILE '/data/area.csv' FIELDS TERMINATED BY '|'
INTO TABLE area (area_no,area_name)
INTO TABLE branches (branch_no,branch_name,area_no);
```

load_type

数据加载的执行类型，可省略，则默认为INSERT。

其中，指定为INSERT和APPEND均表示执行对目标表（包括空表和非空表）的插入数据操作。

table_name

目标表名称，可带模式指定，需保证当前用户对该表拥有所执行操作的权限。

when_clause

对导入的数据指定筛选条件，可指定多个以AND连接的筛选条件。只有满足筛选条件的数据才能被导入。

load_condition_clause

筛选条件的内容。

compare_oper

比较符，可以为=、!=、<>。

table_column_name|csv_column_order

可以按目标表列字段或导入文件数据列进行筛选，且允许在多个条件中指定重复的列。

table_column_name需为目标表列字段名称，csv_column_order需为导入文件数据列的列序号。

column_value

以''包围的一个字符串，用于进行列值比较，当为空串（''）时：

- =''的比较结果恒为false
- !=''和<>''的比较结果恒为true

示例（单机、共享集群部署）

```
LOAD DATA
INFILE '/data/area.csv' FIELDS TERMINATED BY '|'
INTO TABLE area WHEN area_no='12' (area_no,area_name)
INTO TABLE branches WHEN (3)='1201' (branch_no,branch_name,area_no);
```

trailing nullcols

当某个目标表的某个列在导入文件中未映射到数据，但该目标表有其他列能映射到数据（即导入文件中数据列少于目标表的列数）时，指定TRAILING NULLCOLS可以对缺少映射数据的列补NULL值（如果该列上存在一些不允许NULL值的约束定义，补NULL值仍会导致错误），否则该数据会导入失败。

当某个目标表的所有列在导入文件中均未映射到数据（即导入文件中不存在目标表数据）时，指定TRAILING NULLCOLS可以忽略对该表的导入（不影响其他目标表的导入），否则该数据会导入失败。

上述两类错误（指定TRAILING NULLCOLS补NULL值报错和未指定TRAILING NULLCOLS加载任务报错）会影响同一个数据加载命令里的所有目标表，即全部不能导入。

由于存在容错机制，当失败的数据条数未达到容错参数ERRORS指定的容错数据条数上限，语句的执行结果会返回Success。具体的导入情况可查看文件导入的日志文件。

当导入文件中的列数多于目标表的列数时，多余的数据列会被自动忽略且不会抛出错误，这种行为与trailing nullcols配置参数无关。

示例（单机、共享集群部署）

```
--自行创建文件area1.csv和area2.csv，文件位于/data目录下
--area1.csv文件内容为：
13|load
--area2.csv文件内容为：
14|load|1401|loadbranch

--不指定TRAILING NULLCOLS，数据导入失败，且所有目标表都不能导入该行数据
LOAD DATA
INFILE '/data/area1.csv' FIELDS TERMINATED BY '|'
INTO TABLE area (area_no,area_name)
INTO TABLE branches (branch_no,branch_name,area_no);
```

```
--日志（/data/area1.log）内容节选
infile /data/area1.csv offset 1 is rejected by table BRANCH because table column mismatch csv column.

Table AREA:
  0 Rows successfully loaded.
  0 rows not loaded due to data errors.
  0 Rows not loaded because all WHEN clauses were failed.
  0 Rows not loaded because all fields were null.

Table BRANCHES:
  0 Rows successfully loaded.
  1 rows not loaded due to data errors.
  0 Rows not loaded because all WHEN clauses were failed.
  0 Rows not loaded because all fields were null.


--指定TRAILING NULLCOLS加载
--导入area1.csv时将只导入area表数据
LOAD DATA
INFILE '/data/area1.csv' FIELDS TERMINATED BY '|'
INTO TABLE area (area_no,area_name)
INTO TABLE branches TRAILING NULLCOLS (branch_no,branch_name,area_no);

--日志节选
Table AREA:
  1 Rows successfully loaded.
  0 rows not loaded due to data errors.
  0 Rows not loaded because all WHEN clauses were failed.
  0 Rows not loaded because all fields were null.

Table BRANCHES:
  0 Rows successfully loaded.
  0 rows not loaded due to data errors.
  0 Rows not loaded because all WHEN clauses were failed.
  1 Rows not loaded because all fields were null.

--导入area2.csv时将导入area表和branches表数据，branches.area_no补NULL
LOAD DATA
INFILE '/data/area2.csv' FIELDS TERMINATED BY '|'
INTO TABLE area (area_no,area_name)
INTO TABLE branches TRAILING NULLCOLS (branch_no,branch_name,area_no);

--日志节选
Table AREA:
  1 Rows successfully loaded.
  0 rows not loaded due to data errors.
  0 Rows not loaded because all WHEN clauses were failed.
  0 Rows not loaded because all fields were null.

Table BRANCHES:
  1 Rows successfully loaded.
  0 rows not loaded due to data errors.
  0 Rows not loaded because all WHEN clauses were failed.
  0 Rows not loaded because all fields were null.
```

column_clause

目标表的列字段，指定多个列字段以`,`分隔。

对于目标表中存在但未在数据加载中指定的列字段，系统自动补NULL值，因此对非空的列字段必须指定，否则数据加载无法成功。

normal_column_clause

该语句将导入模式指定为普通模式，将导入文件对应列的全部内容用于导入。

鉴于CSV格式文件存在数据大小限制，如需导入较大的数据文件，建议使用LLS模式进行部分导入或者LOBFILE模式进行全导入 。

table_column_clause

目标表已存在的列字段名称。

column(csv_column_order)
对目标表列字段指定到导入文件数据列的映射，可省略，此时处理规则为：

- 当第一个目标表的第一个列字段省略时，默认映射到导入文件的第一个数据列。
- 当非第一个目标表的第一个列字段省略时，默认映射到前一个目标表在导入文件中最后一个映射数据列的下一个数据列，该数据列不存在时按trailing nullcols语句规则处理。
- 当非第一个列字段省略时，默认映射到前一个列字段在导入文件中映射数据列的下一个数据列，该数据列不存在时按trailing nullcols语句规则处理。

映射未省略的处理规则为：

- 允许多个目标表列字段映射到导入文件的同一个数据列。
- 允许映射到一个不存在的数据列，此时按NULL值导入。

示例（单机、共享集群部署）

```
--自行准备area3.csv文件，文件位于/data目录下，文件内容为：
15|16|17|18|19

--执行加载语句
LOAD DATA
INFILE '/data/area3.csv' FIELDS TERMINATED BY '|'
INTO TABLE area (area_no,area_name)
INTO TABLE branches TRAILING NULLCOLS (branch_no,branch_name,area_no COLUMN(1));

SELECT * FROM area WHERE area_no='15';
AREA_NO AREA_NAME       DHQ
------- --------------- --------------
15      16              ShenZhen
SELECT * FROM branches WHERE area_no='15';
BRANCH_NO BRANCH_NAME        AREA_NO ADDRESS
--------- ------------------ ------- --------------------
17        18                 15
```

nullif_clause
本语句用于对列指定是否以NULL值导入的条件，可指定多个以AND连接的条件。当条件结果为true时，数据加载将对该列以NULL值，而非导入文件中映射的数据导入。

load_condition_clause

同when_clause中的load_condition_clause。

示例（单机、共享集群部署）

```
--自行穿件area4.csv文件，文件位于/data目录下，文件内容为：
16|load|1601|loadbranch|16

--执行加载语句，branches.area_no将为NULL而非16
LOAD DATA
INFILE '/data/area4.csv' FIELDS TERMINATED BY '|'
INTO TABLE area (area_no,area_name)
INTO TABLE branches (branch_no,branch_name,area_no NULLIF (2)='load');

SELECT * FROM branches WHERE branch_no='1601';
BRANCH_NO BRANCH_NAME        AREA_NO ADDRESS
--------- ------------------ ------- --------------------
1601      loadbranch
```

filler_column_clause
该语句用于构造伪列，该伪列用于映射CSV文件中的LOB或XMLTYPE文件地址数据，可被lob_column_clause语句引用。

filler_column_name

伪列名称，建议与目标表中已存在列字段不同，否则可能导致导入错误。

csv_column_order
同normal_column_clause中的csv_column_order。

nullif_clause
同normal_column_clause中的nullif_clause。

lob_column_clause

该语句将导入模式指定为LOBFILE模式，该模式下通过引用FILLER伪列指向的LOB或XMLTYPE数据文件，将整个文件导入至目标表指定的列中。

table_column_name
目标表中已存在的列字段名称。

filler_column_name
filler_column_clause中指定的伪列名称。

terminated_by_eof
仅用于语法兼容，无实际含义。

nullif_clause
同normal_column_clause中的nullif_clause。

> **Note**：
> 允许存在未被引用的FILLER伪列，但建议配套使用filler_colomn_clause和lob_column_clause语句。

示例（单机、共享集群部署）

```
--自行创建load_lls.csv文件，文件位于data目录下，内容为：
LOB_FILE.dat.1.2/
LOB_FILE.dat.2.3/

----在数据库中创建表sqlldr_lob
CREATE TABLE sqlldr_lob(c1 INT,c2 CLOB);

LOAD DATA INFILE '/data/load_lls.csv'
INTO TABLE sqlldr_lob(c1,file1 filler,c2 lobfile(file1) terminated BY eof);
```

lls_column_clause

该语句将导入模式指定为LLS（Lob Location Specifier）模式，通过指定LLS关键词选择该导入模式。该模式选取数据文件的部分内容进行导入，且可指定从任意位置和任意长度开始导入。

指定了LLS关键字的目标表列字段映射的导入文件数据列格式需为`filename.ext.nnn.mmm/`：

- filename.ext 为包含LOB或XMLTYPE数据的文件名称。
- nnn 是文件中LOB数据的字节的偏移，仅允许为整数，且偏移量不允许超过数据文件大小且不允许为负数。
  - 当输入偏移量的值为正数时，实际偏移量 = 输入值 - 1。
  - 值为0时，实际偏移量 = 0。
  - 值为负数时返回错误。
- mmm 是字节中的LOB或XMLTYPE的长度，仅允许为整数。值为-1时表示null，为0时表示导入一个空LOB或空XMLTYPE，不允许将其指定为小于-1的值。
- 正斜杠（/）为终止字符，格式中必须包含该字符，否则报错。

table_column_clause
目标表已存在的列字段名称。

csv_column_order
同normal_column_clause中的csv_column_order。

nullif_clause
同normal_column_clause中的nullif_clause。

示例（单机、共享集群部署）

```
--同上例中的文件和表，另外在/data下创建LOB_FILE文件，内容为：
abcdefg
--导入文件列数据为 LOB_FILE.dat.1.2/，意为从LOB_FILE.dat文件的偏移为0的位置为起始，截取长度为2的数据内容进行导入，正斜杠表示终止。
--导入文件列数据为 LOB_FILE.dat.2.3/，意为从LOB_FILE.dat文件的偏移为1的位置为起始，截取长度为3的数据内容进行导入，正斜杠表示终止。

--导入命令为：
LOAD DATA OPTIONS (degree_of_parallelism=3)
INFILE '/data/load_lls.csv' INTO TABLE sqlldr_lob(c2 LLS);
```

directory_clause

本语句用于指定对于run_level为split时拆分后的文件存放的目录，由于SPLIT模式仅在yasldr工具中实现，该语句的具体使用见yasldr使用指导章节描述。

不指定时默认以infile的第一个文件所在的目录作为输出目录。

LOCK TABLE

通用描述

LOCK TABLE用于对表、视图或物化视图以指定模式加表锁，执行该操作的用户需对操作对象具备相应权限：

- 操作对象属于当前用户。
- 操作对象不属于当前用户：
  - 操作对象属于sys用户，当前用户需可访问视图，对表具备INSERT、DELETE或UPDATE对象权限中的任意一种。
  - 操作对象属于非sys用户，当前用户需可访问视图，对表具备除READ外的任何对象权限。

LOCK TABLE不适用于分布式部署。

语句定义

lock table::=

```
syntax::= LOCK TABLE ([ schema"." ](table | view)){","([ schema"." ](table | view))} IN lockmode MODE[NOWAIT|(WAIT
integer)]";"
```

lockmode

该字段用于指定表锁类型。

SHARE

共享表锁允许其他事务对该表加共享锁，但不允许其他事务对该表进行DDL操作。

EXCLUSIVE

排他表锁仅允许其他事务对表进行查询，禁止其他事务对该表进行DML操作或对表加任何锁。

waitmode

该字段用于指定加锁操作是否进行等待以及等待超时的判断条件，等待期间不再具备当前会话的控制权（即无法输入新语句）。

若不指定WAIT或NOWAIT，数据库会无限期地等待直至可对表进行加锁，然后还原会话控制权。

NOWAIT

指定NOWAIT模式，数据库会在无法加锁时立即返回错误信息。

WAIT

指定WAIT模式，数据库会以integer值作为可等待的超时上限，若等待达到时间上限后仍无法加锁，返回错误信息。

integer需指定为整数值，取值范围[0,2147483647]，单位为秒。

示例（单机、共享集群部署）

```
-- 对表进行加锁
LOCK TABLE area,department IN SHARE MODE NOWAIT;

-- 对视图进行加锁
LOCK TABLE v_area IN EXCLUSIVE MODE WAIT 100;
```

MERGE

通用描述

MERGE语句整合了多步的SELECT/UPDATE/INSERT/DELETE操作，使用一次该语句即可实现根据条件（Condition）把源（Source）对象记录整合到目标（Target）对象的功能。

本语句适用于单机HEAP/TAC表和共享集群部署。

语句中的merge_update_clause和merge_insert_clause至少要定义一个。

MERGE语句是对SELECT/UPDATE/INSERT/DELETE的操作整合，也受到在其语句中出现的SELECT/UPDATE/INSERT/DELETE语句相同的条件约束。

语句定义

merge::=

```
syntax::= MERGE [hint] INTO target_table_clause USING source_table_clause ON "(" condition ")" (merge_update_clause |
merge_insert_clause) {" " (merge_update_clause | merge_insert_clause)}
```

target_table_clause**::=**

```
syntax::= [schema "."] table [partition_extension_clause] [t_alias]
```

partition_extension_clause**::=**

```
syntax::= partition (("(" partition ")")|(for "(" (partition_key_value) {"," (partition_key_value)} ")"))
```

source_table_clause**::=**

```
syntax::= ([schema "." ] ((table [partition_extension_clause])|view)|"(" subquery ")") [t_alias]
```

subquery查看SELECT中描述

merge_update_clause**::=**

```
syntax::= WHEN MATCHED THEN UPDATE SET (column "=" (expr|DEFAULT)) {"," (column "=" (expr|DEFAULT))} [where_clause] [DELETE
where_clause]
```

merge_insert_clause**::=**

```
syntax::= WHEN NOT MATCHED THEN INSERT ["(" (column) {"," (column)}")"] VALUES "(" (expr|DEFAULT) {"," (expr|DEFAULT)} ")"
[where_clause]
```

hint

该语句用于提出给定的方案到优化器（Optimizer ），使其按照此方案生成语句的执行计划。查看hint说明。

target_table_clause

该语句用于指定MERGE操作的目标，即表或表分区。

table

目标表的名称。

partition_extension_clause

与INSERT语句中的描述一致。

t_alias

定义别名。

source_table_clause

该语句用于指定MERGE操作的源，可以为表|表分区、视图或子查询的结果集。

view

视图的名称。

subquery

与SELECT语句中的描述一致。

condition

该语句为YashanDB通用条件语句，用于指定MERGE选择下一步操作的条件，对目标表里的每行记录执行此条件判断，为真（MATCHED）时，将执行merge_update_clause；为假（NOT MATCHED）时，将执行merge_insert_clause。

condition可以为任何能产生Boolean结果的表达式，例如：

- TRUE，Boolean结果为TRUE。
- 1=2，Boolean结果为FALSE。
- target.column=source.column，当Target某行按column字段在Source中能匹配到行记录时，Boolean结果为TRUE，否则为FALSE。

merge_update_clause

该语句用于在condition结果为真时，对目标表指定的列字段执行UPDATE语句，指定多列用 `,` 分隔。 存在UPDATE子句时，会让MERGE语句中触发的INSERT、DELETE触发器中 `UPDATING ('*column*')` 条件谓词的结果在更新的列匹配时返回TRUE。

对列字段赋值的数据可以为：

- expr表达式
- DEFAULT：与UPDATE语句中的DEFAULT使用方法一致。

where_clause

对于目标表中根据上面ON condition进入到UPDATE语句的行记录，执行一次条件判断（语法同condition），满足条件的行记录才执行数据的更新。

delete where_clause

对于目标表中已被执行UPDATE语句的行记录，执行一次条件判断（语法同condition），满足条件的记录将从目标表中删除。 DELETE时会直接执行DELETE操作而不是先执行UPDATE再DELETE，该行为会影响UPDATE/DELETE事件触发器的表现。

示例（单机HEAP/TAC表、共享集群部署）

```
--employees为一张员工信息表，包含如下五条数据
SELECT * FROM employees;
BRANCH DEPARTMENT EMPLOYEE_NO   EMPLOYEE_NAME SEX   ENTRY_DATE
------ ---------- ------------- ------------- ----- --------------------------------
0101   000        0101000001    Mask          1     2020-09-08
0101   000        0101000002    John          1     2017-12-13
0201   010        0201010011    Anna          0     2022-08-09
0201   008        0201008003    Jack          1     2021-07-05
0101   008        0201008004    Jim           1     2022-11-17

--创建与employees同构的employees_merge表
CREATE TABLE employees_merge AS SELECT * FROM employees WHERE 1=2;
```

```
INSERT INTO employees_merge VALUES ('0101','008','0201008003','Jim','0',DATE '2021-11-17');
INSERT INTO employees_merge VALUES ('0101','000','0101000002','John','1',DATE '2021-11-17');
COMMIT;

--将eemployees_merge表中的数据merge

--由于0201008003号员工在employees中存在两行记录，需利用此处的department与branch组合条件过滤成一条，否则更新失败
MERGE INTO employees_merge b
USING (SELECT * FROM employees) a
ON (a.employee_no=b.employee_no)
WHEN MATCHED THEN UPDATE SET b.sex=a.sex,b.entry_date=a.entry_date
WHERE a.department='008' AND a.branch='0201';

--employees_merge表merge后数据被更新如下
SELECT * FROM employees_merge;
BRANCH DEPARTMENT EMPLOYEE_NO   EMPLOYEE_NAME SEX   ENTRY_DATE
------ ---------- ------------- ------------- ----- --------------------------------
0101   008        0201008003    Jim           1     2021-07-05
0101   000        0101000002    John          1     2021-11-17

--添加DELETE语句后重新merge，employees_merge表的Jim员工数据先被更新然后被删除
MERGE INTO employees_merge b
USING (SELECT * FROM employees) a
ON (a.employee_no=b.employee_no)
WHEN MATCHED THEN UPDATE SET b.sex=a.sex,b.entry_date=a.entry_date
WHERE a.department='008' AND a.branch='0201'
DELETE WHERE b.department='008';

SELECT * FROM employees_merge;
BRANCH DEPARTMENT EMPLOYEE_NO   EMPLOYEE_NAME SEX   ENTRY_DATE
------ ---------- ------------- ------------- ----- --------------------------------
0101   000        0101000002    John          1     2021-11-17
```

merge_insert_clause

该语句用于在condition结果为假时，对目标表执行INSERT语句。不指定列字段表示按目标表定义的列字段顺序逐个匹配，指定多个列字段用`,`分隔。

对列字段赋值的数据可以为：

- expr表达式
- DEFAULT：与INSERT语句中的DEFAULT使用方法一致。

where_clause

对于目标表中根据上面ON condition进入到INSERT语句的行记录，执行一次条件判断（语法同condition），满足条件的行记录才插入到目标表中。

示例（单机HEAP/TAC表、共享集群部署）

```
--employees为一张员工信息表，包含如下五条数据
SELECT * FROM employees;
BRANCH DEPARTMENT EMPLOYEE_NO   EMPLOYEE_NAME SEX   ENTRY_DATE
------ ---------- ------------- ------------- ----- --------------------------------
0101   000        0101000001    Mask          1     2020-09-08
0101   000        0101000002    John          1     2017-12-13
0201   010        0201010011    Anna          0     2022-08-09
0201   008        0201008003    Jack          1     2021-07-05
0101   008        0201008004    Jim           1     2022-11-17

--创建与employees同构的employees3表
CREATE TABLE employees3 AS SELECT * FROM employees WHERE 1=2;
INSERT INTO employees3 VALUES ('0101','000','0101000002','John','1',DATE '2021-11-17');
COMMIT;

--将empoyees1表中的数据merge
MERGE INTO employees3 b
USING (SELECT * FROM employees) a
ON (a.employee_no=b.employee_no)
WHEN MATCHED THEN UPDATE SET b.sex=a.sex,b.entry_date=a.entry_date
```

```
WHEN NOT MATCHED THEN INSERT VALUES (a.branch,a.department,a.employee_no,a.employee_name,a.sex,a.entry_date);

--employees3中的John员工记录被更新，且在employees中没有的记录被插入
SELECT * FROM employees3;
BRANCH DEPARTMENT EMPLOYEE_NO   EMPLOYEE_NAME SEX   ENTRY_DATE
------ ---------- ------------- ------------- ----- ---------------------------------
0101   000        0101000002    John          1     2017-12-13
0101   000        0101000001    Mask          1     2020-09-08
0201   010        0201010011    Anna          0     2022-08-09
0201   008        0201008003    Jack          1     2021-07-05
0101   008        0201008004    Jim           1     2022-11-17
```

NOAUDIT POLICY

通用描述

NOAUDIT POLICY对审计策略取消使能，取消审计策略对应的审计项的审计。

只有拥有AUDIT_ADMIN审计管理员角色，或者拥有AUDIT SYSTEM系统权限的用户，才能对一个审计策略取消使能。

语句定义

noaudit_policy::=

```
syntax::= NOAUDIT POLICY policy_name [(BY (user_name) {"," (user_name)})] [WHENEVER [ NOT] SUCCESSFUL]
```

policy_name

要取消使能的审计策略的名称。

by

指定要取消使能的操作用户，该用户必须在审计策略使能时指定的用户列表中。

不指定本语句时，在该审计策略是针对所有用户或者该审计策略是except的情况下才能取消使能 。

whenever [not] successful

取消审计策略在数据库操作执行成功|失败时进行审计记录。

不指定本语句时，表示取消审计策略在数据库操作执行成功或失败的审计。

示例

```
-- 取消用户sales对应的审计策略up1的审计
NOAUDIT POLICY up1 BY sales;

-- 取消用户sales对应的审计策略up2在数据库操作执行失败的审计
NOAUDIT POLICY up2 BY sales WHENEVER NOT SUCCESSFUL;
```

PURGE

通用描述

PURGE语句用于清理回收站数据。

YashanDB中，某个表被DROP或TRUNCATE时，如果开启了回收站功能（修改配置参数RECYCLEBIN_ENABLED为ON），这个表的数据将被放入回收站，用于误操作的恢复。

PURGE仅适用于HEAP表。

语句定义

purge::=

```
syntax::= PURGE (TABLE name|INDEX name|TABLESPACE name [USER username]|RECYCLEBIN|DBA_RECYCLEBIN)
```

table

该语句用于清理回收站中指定的表，清理表的同时会清理表上的其他对象，如索引，LOB等。

index

该语句用于清理回收站中指定的索引。

tablespace name [user username]

该语句用于清理回收站中指定的表空间，指定user时将只清理指定表空间下指定用户下的对象。

recyclebin

该语句用于清理回收站中当前用户下的所有对象。

dba_recyclebin

该语句用于清理回收站中的全部对象。

示例（HEAP表）

```
--开启回收站
ALTER SYSTEM SET RECYCLEBIN_ENABLED=ON;

--删除某个对象
DROP TABLE finance_info;
--清理回收站中指定的对象数据
PURGE TABLE finance_info;

--清理指定表空间的回收站数据
PURGE TABLESPACE yashan;
--清理回收站中当前用户下的所有对象数据
PURGE RECYCLEBIN;
--清理回收站中的全部对象数据
PURGE DBA_RECYCLEBIN;
```

RECOVER DATABASE

通用描述

RECOVER DATABASE用于从备份集和归档日志/在线日志还原数据库到指定时间点的状态，其中：

RECOVER DATABASE (FROM)=完全恢复，即尽可能回放所有redo，恢复到最新状态。

RECOVER DATABASE UNTIL ...=不完全恢复，即恢复到指定的目标点就停止。

只能在数据库状态为MOUNT时执行RECOVER。

通过备份集恢复数据库时，在RECOVER之前必须先执行RESTORE将数据库恢复到备份集的状态。执行PITR恢复时，RECOVER操作要求在备份集生成时间之后且指定时间点之前的归档日志必须存在。

执行不完全恢复后，需执行ALTER DATABASE OPEN RESETLOGS语句开启数据库并重置redo时间线。

关于备份恢复的详细操作描述请参考备份恢复。

该语句不适用于分布式部署。

语句定义

recover database::=

```
syntax::= RECOVER DATABASE [(UNTIL (TIME date|SCN scn|CANCEL))|(FROM SEQUENCE integer)]
```

until time

该语句用于将数据库恢复到此指定的时间点，date格式需与DATE_FORMAT参数定义的格式一致。

指定的时间点必须在备份集生成时间之后。

> **Note**：
> 最小时间点是保证数据库恢复到一致性的时间点，该时间点对应与备份完datafile，开始备份归档时的SCN对应的时间点，略小于备份完成时间。
> 如果归档日志不够，或者指定的时间点过大，回放完毕后也达不到指定的时间点，数据库会报错，用户可以查询V$DATABASE的current_scn来查看实际回放进度。如果有可用的归档日志副本，可以注册归档，重新执行restore操作。如果没有多余的归档，当前时间点是唯一的选择，继续执行resetlogs操作。

示例（单机、共享集群部署）

```
--启动数据库到NOMOUNT状态，然后执行restore操作
RESTORE DATABASE FROM 'backup';

--RESTORE操作成功后数据库启动到MOUNT状态
RECOVER DATABASE UNTIL TIME TO_DATE('2021-11-25 18:55:00','yyyy-mm-dd hh24:mi:ss');

--打开数据库
ALTER DATABASE OPEN RESETLOGS;
```

until scn

该语句用于将数据库恢复到此指定的SCN号。指定的SCN号必须大于备份集中记录的SCN号。

> **Note**：
> 最小SCN是保证数据库恢复到一致性的SCN，该SCN就是备份完datafile，开始备份归档时的SCN，可能略小于备份完成时的SCN。
> 如果归档日志不够，或者指定的SCN过大，回放完毕后也达不到指定的SCN，数据库会报错，用户可以查询V$DATABASE的current_scn来查看实际回放进度。如果有可用的归档日志副本，可以注册归档，重新执行restore操作。如果没有多余的归档，当前时间点是唯一的选择，继续执行resetlogs操作。

until cancel

该语句用于数据库正常启动失败，无法恢复到一致性状态，没有其他恢复手段的情况下，可以使用该语句强制启动。执行该语句后，执行OPEN命令需要带RESETLOGS。

> **Warning**：
> 该语句为高危操作，使用该语句强制启动数据库后，数据库可能会产生各种无法预料的故障，例如：宕机、事务不一致等。
> 当数据库无法恢复到一致性状态时，优先考虑其他恢复手段，最后考虑该恢复手段。
> 使用该语句强制启动数据库后，建议导出必要的数据后，重建数据库，不要继续使用原来的数据库，以防出现未知的故障。

示例（单机、共享集群部署）

```
--先将数据库启动到MOUNT，然后执行以下语句，强制启动
RECOVER DATABASE UNTIL CANCEL;

--打开数据库
ALTER DATABASE OPEN RESETLOGS;
```

from sequence

该语句用于数据库从指定的日志序列号开始恢复，指定的日志序列号不能大于当前数据库的恢复开始点（对应于V$DATABASE的RCY_POINT），需保证指定序列号之后的日志存在。

RELEASE SAVEPOINT

通用描述

RELEASE SAVEPOINT用于从当前事务的一组保存点中删除某个保存点，删除后将无法再回退到该保存点。

语句定义

release savepoint::=

```
syntax::= RELEASE SAVEPOINT savepoint_name
```

示例

```
-- 设置保存点
SAVEPOINT sp_1;

-- 删除保存点
RELEASE SAVEPOINT sp_1;
```

RESTORE ARCHIVELOG

通用描述

RESTORE ARCHIVELOG用于从备份恢复指定归档范围的归档日志文件至目标位置。

只能在数据库状态为MOUNT或者OPEN状态下执行恢复。

关于备份恢复的详细操作描述请参考备份恢复。

该语句仅适用于在单机和共享集群部署。

语句定义

restore archivelog::=

```
syntax::= RESTORE archivelog archivelogRangeSpecifier [DECRYPTION  password] [PARALLELISM integer] [FROM
(SEARCHDIR|BACKUPSET) "archdirpath"] [TO "destpath"]
```

archivelog_range_specifier::=

```
syntax::= ((
(FROM SCN | SCN BETWEEN interger AND | UNTIL SCN ) interger |
(FROM SEQUENCE | SEQUENCE BETWEEN interger AND | UNTIL SEQUENCE ) interger [THREAD interger] |
(FROM TIME | TIME BETWEEN date_string AND | UNTIL TIME ) date_string | ALL))
```

archivelograngespecifier

用于指定需恢复的归档日志范围，使用方法同BACKUP ARCHIVELOG。

若范围区间内有指定的归档日志文件不存在（例如文件已被删除、某一序列号对应的归档日志文件暂未生成等）现象，恢复操作会失败并提示错误码YAS-02540。

from

在归档恢复语句中，与SCN、SEQUENCE或TIME配合使用指定为归档日志恢复的起点。

指定FROM指令恢复归档日志时，恢复的终点为当前数据库的日志恢复点的上一个归档文件，具体取值可以查询V$DATABASE视图的RCY_POINT字段中的ASN。

between … and …

在归档恢复语句中，与SCN、SEQUENCE或TIME配合使用指定归档日志备份的区间。

指定BETWEEN …（起点） AND …（终点）指令恢复归档时，恢复的起点和终点均为指定值，区间范围应为[当前数据库的日志刷盘点ASN,当前数据库的日志恢复点的上一个归档文件]的子集。

until

在归档备份语句中，与SCN、SEQUENCE或TIME配合使用指定为归档日志恢复的终点。

指定UNTIL指令恢复归档日志时，恢复的起点为当前数据库的日志刷盘点ASN，具体取值可以查询V$DATABASE视图的FLUSH_POINT字段中的ASN。

decryption

该语句用于指定备份集恢复时解密，此时需要同时指定解密密码。

parallelism

该语句用于指定多线程恢复的并行度，取值范围为[1,8]，省略时默认为2。

from searchdir/backupset

该语句指定归档备份集的集合，若目录下存在多个归档备份集的集合，可使用SEARCHDIR关键字指定集合的文件夹，若只指定单个备份集，使用关键字BACKUPSET即可。

to destpath

该语句指定归档的目标恢复路径，需保证该路径存在。

> **Note**：
> 若对归档日志文件进行了加密备份，在恢复语句中需要指定解密密码。若备份集集合文件夹中各备份集的加密密码不一致，则在恢复过程中会自动跳过不一致的备份集。
> 若要分别恢复不同解密密码的的归档备份集，则需要多次执行符合目标备份集秘钥的恢复语句。

RESTORE DATABASE

通用描述

RESTORE DATABASE用于从备份集还原出数据库文件，恢复到备份时的状态。

只能在数据库状态为NOMOUNT，且旧的数据文件必须已被删除时执行恢复。

成功恢复后数据库变更为MOUNT状态，此时还应该执行RECOVER操作，以使系统根据日志还原到指定时间点。

关于备份恢复的详细操作描述请参考备份恢复。

该语句仅适用于在单机和共享集群部署。

语句定义

restore database::=

```
syntax::= RESTORE DATABASE [INCREMENTAL [NOREDO]] [DECRYPTION  password] FROM backup_path [PARALLELISM integer]
```

incremental [noredo]

该语句用于指定增量RESTORE恢复数据库，仅在指定INCREMENTAL字段后才可使用NOREDO字段，指定NOREDO字段可在恢复时跳过恢复redo和归档文件，提升RESTORE效率。

backup_path

该语句用于指定用于恢复的备份集名称。

decryption

该语句用于指定备份集恢复时解密，此时需要同时指定解密密码。

parallelism

该语句用于指定多线程恢复的并行度，该值范围[1, 8]，省略时默认为2。

示例（单机、共享集群部署）

```
RESTORE DATABASE DECRYPTION 12345 FROM 'backup';
```

REVOKE

通用描述

REVOKE用于对某一用户或角色收回授予其的权限，权限内容包括：

- 系统级权限SYSTEM PRIVILEGE
- 对象级权限OBJECT PRIVILEGE
- 角色ROLE

收回授予给用户上的权限立即生效。

收回授予给角色上的权限立即生效，如该角色已被授权给某个用户，则该用户的权限收回也立即生效。

回收授予给用户或角色的角色在用户再次连接时生效，即回收用户A的角色B，回收前已登录的用户A依然拥有B角色，A用户再次登录时失去B角色。

普通用户只能收回自己授予的权限，即使对于指定了WITH GRANT|ADMIN OPTION语句实现跨用户授权的场景，也不能跨用户收回权限。拥有SECURITY_ADMIN安全管理员角色的用户可收回任何权限。

YashanDB支持在单机/集群部署中回收授予给角色的角色，在分布式部署形态中执行此操作返回错误。

YashanDB的权限体系管理具体描述请查看权限管理。

语句定义

revoke::=

```
syntax::= revoke_object_privilege|revoke_system_privilege|revoke_role_privilege
```

revoke_object_privilege::=

```
syntax::= REVOKE (object_privilege|ALL PRIVILEGES) ON [schema "."] table_name  FROM (user|role)
```

revoke_system_privilege::=

```
syntax::= REVOKE (system_privilege|ALL PRIVILEGES) FROM (user|role)
```

revoke_role_privilege::=

```
syntax::= REVOKE role FROM (user|role)
```

revoke_object_privilege

该语句用于将指定的对象级权限从用户/角色收回，如指定ALL PRIVILEGES表示收回所有对象级权限。

- 若打开三权分立功能，可以通过拥有SECURITY_ADMIN安全管理员角色的用户收回。
- 若关闭三权分立功能，可以通过拥有SECURITY_ADMIN安全管理员角色或拥有DBA角色的用户收回。
- 如某个普通用户通过WITH ADMIN OPTION语句被授予了权限，那么此用户可以收回自己授予给其他用户的权限。
- 可以通过拥有GRANT ANY OBJECT PRIVILEGE系统级权限的普通用户收回。

table_name|user_name|role

对象|用户|角色名称。

示例

```
--在用户sales下收回其授予给sales1的SELECT ON area TO sales1 WITH GRANT OPTION权限
REVOKE SELECT ON area FROM sales1;

--sales2的SELECT ON sales.area权限由sales1授权，因此只能由sales1或系统用户收回
```

```
conn sales1/1%2
REVOKE SELECT ON sales.area FROM sales2;
```

revoke_system_privilege

该语句用于将指定的系统级权限从用户/角色收回，如指定ALL PRIVILEGES表示收回所有系统级权限。

- 若打开三权分立功能，可以通过拥有SECURITY_ADMIN安全管理员角色的用户收回。
- 若关闭三权分立功能，可以通过拥有SECURITY_ADMIN安全管理员角色或拥有DBA角色的用户收回。
- 如某个普通用户通过WITH ADMIN OPTION语句被授予了权限，那么此用户可以收回自己授予给其他用户的权限。
- 可以通过拥有GRANT ANY PRIVILEGE系统级权限的普通用户收回。

user_name|role

用户|角色名称。

示例

```
--在系统用户下将授予sales1用户的ALTER SYSTEM权限收回
REVOKE ALTER SYSTEM FROM sales1;
```

revoke_role_privilege

该语句用于将指定的角色权限从用户/角色收回。

- 若打开三权分立功能，可以通过拥有SECURITY_ADMIN安全管理员角色的用户收回。
- 若关闭三权分立功能，可以通过拥有SECURITY_ADMIN安全管理员角色或拥有DBA角色的用户收回。
- 可以通过拥有GRANT ANY ROLE的系统级权限的普通用户收回。

示例

```
--在用户sales下收回其授予给sales1的ya_rol1角色权限
REVOKE ya_rol1 FROM sales1;
```

ROLLBACK

通用描述

ROLLBACK用于回滚一个事务。以下为不指定SAVEPOINT时的回滚规则：

- 执行ROLLBACK操作前，用户可以实时看到自己修改后的数据，其他用户无法看到。
- 执行ROLLBACK操作时，系统将Data Buffer里的数据恢复为修改之前的值，同时清除事务所在会话所有的SAVEPOINT，释放本事务级别的所有锁，本次事务结束。
- 执行ROLLBACK操作后，所有用户都只能看到修改前的数据。

在YashanDB中，系统发生任何异常情况时，均会自动调用ROLLBACK。

语句定义

rollback::=

```
syntax::= ROLLBACK [WORK] [TO [SAVEPOINT] savepoint_name]
```

work

该语句用于和标准SQL的语法兼容，无实际意义。

to savepoint

该语句用于回滚指定保存点之后的操作，执行此语句时：

- 系统将Data Buffer里的数据恢复为该保存点时的数值。
- 清除该保存点之后的所有的SAVEPOINT。
- 不结束事务。

示例

```
--回滚整个事务
ROLLBACK;

--回滚到指定保存点
ROLLBACK TO SAVEPOINT sa_area_1;
```

SAVEPOINT

通用描述

SAVEPOINT用于在事务过程中建立一个保存点，在ROLLBACK语句中可以根据此保存点来指定回滚在它之后的操作。

如果事务被执行了COMMIT语句，则包括指定了保存点的回滚操作都将无效。

如果保存点的名称出现重复，旧的保存点将被覆盖而导致失效。

如果按某一保存点进行了回滚，其后面定义的保存点将被清除。

语句定义

savepoint::=

```
syntax::= SAVEPOINT name
```

savepoint_name

该语句用于指定保存点的名称，不可省略，且需符合YashanDB的对象命名规范。

分布式部署中不允许创建名称以`SYS_S_`开头的保存点。

示例

```
--初始数据
SELECT * FROM area WHERE area_no='01';
AREA_NO AREA_NAME                                                        DHQ
------- ---------------------------------------------------------------- --------------------
01      华东                                                             Shanghai

--设置保存点
UPDATE area SET DHQ='Hangzhou' WHERE area_no='01';
SAVEPOINT sa_area_1;
UPDATE area SET DHQ='Nanjing' WHERE area_no='01';
SAVEPOINT sa_area_1;
UPDATE area SET DHQ='Suzhou' WHERE area_no='01';
SAVEPOINT sa_area_3;

--按保存点回滚,由于重名，第一个保存点无效
ROLLBACK TO SAVEPOINT sa_area_1;
SELECT * FROM area WHERE area_no='01';
AREA_NO AREA_NAME                                                        DHQ
------- ---------------------------------------------------------------- --------------------
01      华东                                                             Nanjing

--由于已执行了到前面保存点的回滚，sa_area_3被清除
ROLLBACK TO SAVEPOINT sa_area_3;
YAS-02022 savepoint sa_area_3 not exist
```

SELECT

通用描述

SELECT用于执行对数据库的表、视图、AC等的查询操作。

YashanDB支持丰富多样的查询操作，包括但不限于：

- 多表连接（子查询、SET、JOIN等）
- 排序ORDER BY
- 分组GROUP BY
- CASE
- LIMIT
- CTE
- 层次化/递归
- 抽样
- 指定分区
- 指定切片（Slice）

其中，对AC可以执行上述列示的多表连接、排序、分组等基本查询。

对于分布式系统视图（DV$开头的系统视图），所有操作均在各节点本地执行，最后再汇总至CN。

语句定义

select::=

```
syntax::= subquery [for_update_clause]
```

subquery**::=**

```
syntax::= (query_block
|(subquery (set_oper subquery) {" " (set_oper subquery)})
|"(" subquery ")")
[order_by_clause]
[row_limiting_clause]
```

set_oper**::=**

```
syntax::= (UNION|INTERSECT|MINUS|EXCEPT) [ALL]
```

query_block**::=**

```
syntax::= [with_clause]
SELECT [hint] [DISTINCT] select_list
FROM (table_reference|join_clause|"(" join_clause ")")
{" " (table_reference|join_clause|"(" join_clause ")")}
[where_clause]
[hierarchical_query_clause]
[group_by_clause]
```

with_clause**::=**

```
syntax::= WITH (cte_clause) {"," (cte_clause)}
```

cte_clause**::=**

```
syntax::= cte_name ["(" (column_alias) {"," (column_alias)} ")"] AS "(" subquery ")"
```

select_list::=

```
syntax::= ("*"|expr_clause [[AS] alias]) {"," ("*"|expr_clause [[AS] alias])}
```

expr_clause::=

```
syntax::= ["("] (query_block|expr|case_clause|udt_expr) [")"]
```

case_clause::=

```
syntax::= CASE (simple_case_expression|searched_case_expression)
{"," (simple_case_expression|searched_case_expression)}
[else_clause] END
```

simple_case_expression::=

```
syntax::= expr (WHEN comparison_expr THEN return_expr)
{"," (WHEN comparison_expr THEN return_expr)}
```

searched_case_expression::=

```
syntax::= (WHEN condition THEN return_expr) {"," (WHEN condition THEN return_expr)}
```

else_clause::=

```
syntax::= ELSE else_expr
```

table_reference::=

```
syntax::= query_table_expression [flashback_query_clause]
```

query_table_expression::=

```
syntax::= ((query_name|subquery) [t_alias]) {"," ((query_name|subquery) [t_alias])}
```

query_name::=

```
syntax::= [schema "."]  (((table_name[dblink]|synonym_name)[partition_extension_clause][SLICE "(" slice_id ")" ]  )
|view_name
|table_collection_expression)
[sample_clause]
[pivot_clause]
```

partition_extension_clause::=

```
syntax::= PARTITION ("(" partition ")"|FOR "(" (partition_key_value) {","(partition_key_value)} ")")|SUBPARTITION ("("
subpartition ")"|FOR "(" (subpartition_key_value) {","(subpartition_key_value)} ")")
```

table_collection_expression::=

```
syntax::= TABLE "(" collection_expression ")"
```

sample_clause::=

```
syntax::= SAMPLE "(" sample_percent ")" [SEED "(" seed_value ")"]
```

flashback_query_clause::=

```
syntax::= AS OF ((SCN|TIMESTAMP) expr)
```

pivot_clause::=

```
syntax::= PIVOT "(" aggregate_expression [[AS] alias] { "," aggregate_expression [[AS] alias]} FOR column_expression IN "("
const_expression [[AS] alias] { "," const_expression [[AS] alias]}  ")" ")"
```

join_clause::=

```
syntax::= table_reference (inner_cross_join_clause|outer_join_clause )
{"," (inner_cross_join_clause|outer_join_clause)}
```

inner_cross_join_clause::=

```
syntax::= ([INNER] JOIN table_reference ON condition)|(CROSS JOIN table_reference)
```

outer_join_clause::=

```
syntax::= outer_join_type JOIN table_reference [ON condition]
```

outer_join_type::=

```
syntax::= (LEFT|RIGHT) [OUTER]
```

where_clause::=

```
syntax::= WHERE condition
```

hierarchical_query_clause::=

```
syntax::= (connect_by_clause [start_with_clause] [order_siblings_by_clause]
| start_with_clause connect_by_clause [order_siblings_by_clause])
```

connect_by_clause::=

```
syntax::= CONNECT BY [NOCYCLE] condition
```

start_with_clause::=

```
syntax::= START WITH condition
```

order_siblings_by_clause::=

```
syntax::= ORDER SIBLINGS BY ((expr|position|c_alias) [ASC|DESC] [(NULLS FIRST)|(NULLS LAST)])
{"," ((expr|position|c_alias) [ASC|DESC] [(NULLS FIRST)|(NULLS LAST)])}
```

group_by_clause::=

```
syntax::= (group_by_clause1|group_by_clause2)
```

group_by_clause1::=

```
syntax::= GROUP BY (((expr)){"," (expr)}|rollup_cube_clause|grouping_sets_clause) [HAVING condition]
```

rollup_cube_clause::=

```
syntax::= (ROLLUP|CUBE) "("((expr)){"," (expr)}")"
```

grouping_sets_clause::=

```
syntax::= GROUPING SETS "("((expr)){"," (expr)}")"
```

group_by_clause2::=

```
syntax::= [HAVING condition] [GROUP BY] (((expr)){"," (expr)}|rollup_cube_clause|grouping_sets_clause)
```

order_by_clause::=

```
syntax::= ORDER BY ((expr|position|c_alias) [ASC|DESC] [(NULLS FIRST)|(NULLS LAST)])
{"," ((expr|position|c_alias) [ASC|DESC] [(NULLS FIRST)|(NULLS LAST)])}
```

row_limiting_clause::=

```
syntax::= [LIMIT expr [offset expr]]
```

offset_fetch_clause::=

```
syntax::= [OFFSET offset (ROW | ROWS)] [ FETCH (FIRST | NEXT) rowcount (ROW | ROWS) ONLY]
```

for_udpate_clause::=

```
syntax::= FOR UPDATE [OF [([schema "."] (table_name|view_name|synonym_name) ".")|t_alias] column_name]
[(NO WAIT)|(WAIT ntimes)|(SKIP LOCKED)]
```

subquery

YashanDB的SELECT语句支持嵌套查询（对封装在`()`中的SELECT语句的结果进行SELECT查询）和合并查询（用集合合并多个SELECT语句的查询结果），不同方式的查询可交叉及多层级使用，例如：

```
SELECT * FROM (SELECT a,b FROM (SELECT cl_a a,cl_b b FROM table_a WHERE ...
                                UNION ALL
                                SELECT cl_c a,cl_d b FROM table_b WHERE ...
                               )
               WHERE ...
               )
WHERE ...
```

每一个完整的SELECT查询语句（包括其下级）均可称为一个子查询（subQuery）。 对于子查询中的select_list列项：

- 若这些列项显式指定了别名（例如为列c1_a定义别名a），系统以此别名作为子查询返回结果集的列名，YashanDB对显式指定别名的长度限制为64字节。
- 若这些列项未显式指定别名，系统默认以该项列名作为子查询返回结果集的列名，且超过20字节截断，此时，如子查询select_list中存在名称前20字节相同的列，将导致返回结果集产生相同名称的列而触发系统报错。

分布式部署中的子查询不能为多表连接。

示例

```
SELECT * FROM
(SELECT CAST(CAST(a.area_no AS INT) AS BIGINT),
CAST(CAST(a.area_no AS VARCHAR(100)) AS VARCHAR(100))
FROM area a);
[3:1]YAS-04301 ambiguous column
```

query_block

含有单个SELECT的查询语句称为一个查询块（queryBlock）。

set_oper

集合操作，用于将两个或两个以上的查询结果集组合成单个结果集，包括如下类型：

- **UNION**：合并且过滤掉重复值。
- **UNION ALL**：合并且保留重复值。
- **INTERSECT**：比较两个查询的结果，选择所有查询结果集中相同行的记录，且过滤掉重复值。
- **INTERSECT ALL**：比较两个查询的结果，选择所有查询结果集中相同行的记录，且保留重复值。
- **MINUS/EXCEPT**：比较两个查询的结果，返回在第一个查询结果集中，但不是第二个查询结果集中的行的记录，且过滤掉重复值。
- **MINUS ALL/EXCEPT ALL**：比较两个查询的结果，返回在第一个查询结果集中，但不是第二个查询结果集中的行的记录，且保留重复值。

集合操作中MINUS与EXCEPT完全等价。

进行集合操作的查询结果集的数据类型必须处于下面同一个分类中，否则函数返回Query column mismatch错误：

- 数值型：按优先度从高到低为DOUBLE、FLOAT、NUMBER、BIGINT、INT、SMALLINT、TINYINT，但BIT类型只能与同类型在同一个操作列表中。
- 字符型：按优先度从高到低为VARCHAR、CHAR。
- 日期时间型：按优先度从高到低为TIMESTAMP、DATE、TIME，但INTERVAL YEAR TO MONTH、INTERVAL DAY TO SECOND只能与同类型在同一个参数列表中。
- 布尔型：BOOLEAN。
- 其他类型：RAW。

> **Note**：
> 1.同一分类中数据类型不同时，低优先度类型向高优先度类型转换。
> 2.结果集为常量NULL（查询列为常量NULL所生成的结果集）时，不受上述分类限制。
> 3.所有集合运算为统一运算优先级，运算顺序以书写顺序为准，可通过双括号（）调整想要达到的运算优先级。

示例

```
--UNION
SELECT *
FROM (SELECT * FROM area WHERE area_no IN ('01','02')
      UNION
      SELECT * FROM area WHERE area_no IN ('01','03')
     );
AREA_NO AREA_NAME          DHQ
------- ------------------ ----------
01      华东               Shanghai
02      华西               Chengdu
03      华南               Guangzhou

--UNION ALL
SELECT *
FROM (SELECT * FROM area WHERE area_no IN ('01','02')
      UNION ALL
      SELECT * FROM area WHERE area_no IN ('01','03')
     );
```

```
AREA_NO AREA_NAME          DHQ
------- ------------------ ----------
01      华东               Shanghai
03      华南               Guangzhou
01      华东               Shanghai
02      华西               Chengdu

--INTERSECT
SELECT *
FROM (SELECT * FROM area WHERE area_no IN ('01','02')
      INTERSECT
      SELECT * FROM area WHERE area_no IN ('01','03')
     );
AREA_NO AREA_NAME          DHQ
------- ------------------ ----------
01      华东               Shanghai

--INTERSECT ALL
SELECT *
FROM (SELECT * FROM area WHERE area_no IN ('01','02')
      INTERSECT ALL
      SELECT * FROM area WHERE area_no IN ('01','03')
     );
AREA_NO AREA_NAME          DHQ
------- ------------------ ----------
01      华东               Shanghai

--MINUS
SELECT *
FROM (SELECT * FROM area WHERE area_no IN ('01','02')
      MINUS
      SELECT * FROM area WHERE area_no IN ('01','03')
     );
AREA_NO AREA_NAME          DHQ
------- ------------------ ----------
02      华西               Chengdu

--MINUS ALL
SELECT *
FROM (SELECT * FROM area WHERE area_no IN ('01','02')
      MINUS ALL
      SELECT * FROM area WHERE area_no IN ('01','03')
     );
AREA_NO AREA_NAME          DHQ
------- ------------------ ----------
02      华西               Chengdu
```

with_clause

该语句用于定义CTE（Common Table Expression），在某个子查询被多次使用时，可以直接使用CTE名称而不需要重复写该SELECT语句。

cte_clause

CTE定义，包括：

- cte_name：名称，作为其他地方引用的标识。
- column_alias：定义CTE的列项，其中列字段应该来自于其后的子查询中定义的列项。
- subquery：用于创建CTE的子查询。

示例

```
--WITH语句用于获取area表中'01'和'04'地区的代码和名称，在后面的查询中对branches表获取关联地区名称数据时，不需要多次写该语句，而只用和q_area做关联查询即可
WITH q_area(ano,aname) AS (SELECT area_no ano,area_name aname FROM area WHERE area_no IN ('01','04'))
SELECT bno,bname,aname FROM (
SELECT a.branch_no bno,a.branch_name bname,b.aname aname FROM branches a,q_area b WHERE a.area_no=b.ano AND a.area_no='01'
UNION
SELECT a.branch_no bno,a.branch_name bname,b.aname aname FROM branches a,q_area b WHERE a.area_no=b.ano AND a.area_no='04'
```

```
);
BNO   BNAME             ANAME
----- ----------------- -------------
0101  上海              华东
0102  南京              华东
0103  福州              华东
0104  厦门              华东
0401  北京              华北
0402  天津              华北
0403  大连              华北
0404  沈阳              华北
```

hint

该语句用于提出给定的方案到优化器（Optimizer），使其按照此方案生成语句的执行计划。查看hint说明。

distinct

该语句用于对查询结果进行过滤设置，对查询结果中重复的多条记录只返回一条记录。

select_list

该语句用于指定查询语句中要查询的列项，例如FROM后面某个表的某个列字段（Column）等。查询多项以`,`分隔。`*`表示查询FROM后面所有表|视图|AC|子查询等的所有列。

AS alias用于定义别名，AS可省略。

expr_clause

指定具体的列项，可以为：

- query_block：将一个查询块语句的结果作为列项
- expr：表达式
- CASE语句
- udt_expr：对于Object UDT列，udt_expr格式为表别名.属性名；对于Varray UDT和Nested Table UDT列，udt_expr格式为表别名.*。详细使用方法和示例见用户自定义类型中描述。

示例（单机、分布式部署）

```
--单机部署中
SELECT SYSDATE 查询日期,
a.branch_name 机构,
b.area_name 区域,
(SELECT DHQ FROM area WHERE area_no='10') 默认总部
FROM branches a, area b
WHERE a.area_no IN (SELECT area_no FROM area )
AND a.area_no=b.area_no;
查询日期                      机构          区域                默认总部
----------------------------- ------------- ------------------- ------
2022-01-10 10:24:35           上海          华东
2022-01-10 10:24:35           南京          华东
2022-01-10 10:24:35           福州          华东
2022-01-10 10:24:35           厦门          华东
2022-01-10 10:24:35           北京          华北
2022-01-10 10:24:35           天津          华北
2022-01-10 10:24:35           大连          华北
2022-01-10 10:24:35           沈阳          华北
2022-01-10 10:24:35           成都          华西
2022-01-10 10:24:35           长沙          华中

--分布式部署中
SELECT SYSDATE 查询日期,
a.branch_name 机构,
b.area_name 区域,
'ShenZhen' 默认总部
FROM branches a, area b
WHERE a.area_no IN ('01','02','03')
AND a.area_no=b.area_no;
```

```
查询日期         机构               区域                 默认总部
------------------ ------------------ -------------------- ----------
2022-01-23         上海               华东                 ShenZhen
2022-01-23         南京               华东                 ShenZhen
2022-01-23         福州               华东                 ShenZhen
2022-01-23         厦门               华东                 ShenZhen
2022-01-23         成都               华西                 ShenZhen
```

case_clause
case_clause = CASE { simple_case_expression | searched_case_expression } [else_clause] END.

CASE表达式相当于编程语言中的IF ELSE，其写法有两种：Simple Case和Searched Case。

simple_case_expression = expr { WHEN comparison_expr THEN return_expr }.

searched_case_expression = { WHEN condition THEN return_expr }.

else_clause = ELSE else_expr.

在Simple Case写法中，系统搜寻第一个与expr等值的comparision_expr，并返回相对应的return_expr。如果没有等值的comparision_expr，并且存在ELSE子句，则返回else_expr。否则返回NULL。

在Searched Case写法中，系统从左至右依次检查condition，并返回第一个相对应的return expr。如果所有condition皆为FALSE，并且存在ELSE子句，则返回else_expr。否则返回NULL。

需要注意的是，CASE语句中出现的所有表达式（else_expr/return_expr/expr/comparison_expr）的值在同一个语句中必须同属下列几种大类之一：

- 数值型，优先度从低到高的排序为：TINYINT、SMALLINT、INT、BIGINT、NUMBER、FLOAT、DOUBLE。
- 字符型，优先度从低到高的排序为：CHAR、VARCHAR。
- 日期时间型，优先度从低到高的排序为：DATE、TIMESTAMP。
- 其他数据类型归成一个大类。

在每个大类中，表达式的值将先统一为优先度高的数据类型，再进行匹配或返回。因此CASE表达式返回值的数据类型为所有表达式中优先度最高的类型。

示例

```
SELECT employee_name 姓名, CASE sex
WHEN '0' THEN '女'
WHEN '1' THEN '男'
END 性别
FROM employees;
姓名        性别
------------ ------
Mask         男
John         男
Anna         女
Jack         男
Jim          男

--存在未列出的条件结果时，返回NULL
SELECT employee_name 姓名, CASE sex
WHEN '2' THEN '女'
WHEN '1' THEN '男'
END 性别
FROM employees;
姓名        性别
------------ ------
Mask         男
John         男
Anna
Jack         男
Jim          男

--对所有不在列出条件的情况，给定默认值
SELECT employee_name 姓名, CASE sex
WHEN '2' THEN '女'
```

```
WHEN '1' THEN '男'
ELSE '未知'
END 性别
FROM employees;
姓名        性别
------------- ---------
Mask          男
John          男
Anna          未知
Jack          男
Jim           男
```

table_reference

该语句用于指定FROM语句中对查询对象的定义。

query_table_expression

该语句用于指定要查询的对象，YashanDB支持为查询对象分别定义别名，查询对象可为如下类型：

- query_name：表、视图、同义词、分区、子分区、AC等
- subquery：一个子查询的结果
- dblink：远端表

查询对象不能同时存在行存表和列存表，否则返回错误。

query_name

该语句用于指定表、视图、同义词、分区、子分区、AC等查询对象。

partition_extension_clause

对表分区和子分区的指定方式包括：

- 根据名称：直接获取分区或子分区对象。
- 根据提供的键值（Key Value）： 将键值与表分区界值比较，计算得到分区或子分区对象，键值可为界值范围内的任意值。键值的个数须与分区列个数对应，指定子分区时键值个数须与分区列和子分区列个数对应，多项用 `,` 分隔。

> **Note**：
> 分布式样例表中未指定分区名，由数据库定义分区名及子分区名，可通过查看DBA_TAB_PARTITIONS/DBA_TAB_SUBPARTITIONS查看分区及子分区信息。

示例

```
--执行如下语句查询分区名称
SELECT partition_name
FROM DBA_TAB_PARTITIONS
WHERE table_name='SALES_INFO';
PARTITION_NAME
----------------------------------------------------------------
P_SALES_INFO_1
P_SALES_INFO_2
P_SALES_INFO_3

--sales_info为一张分区表，可指定分区来查询其数据
SELECT * FROM sales_info PARTITION (P_SALES_INFO_1);
YEAR  MONTH BRANCH PRODUCT      QUANTITY      AMOUNT SALSPERSON
----- ----- ------ --------- ----------- ----------- -------------
2021  10    0402   11001              20         300

SELECT * FROM sales_info PARTITION FOR('0102');
YEAR  MONTH BRANCH PRODUCT      QUANTITY      AMOUNT SALSPERSON
----- ----- ------ --------- ----------- ----------- -------------
2001  01    0201   11001              30         500 0201010011
2000  12    0102   11001              20         300
2015  03    0102   11001              20         300
```

```
--获得sales_info表的子分区名称
SELECT partition_name,subpartition_name
FROM DBA_TAB_SUBPARTITIONS
WHERE table_name='SALES_INFO';
--以下输出以单机为例
PARTITION_NAME               SUBPARTITION_NAME
---------------------------- ------------------------------------
P_SALES_INFO_1               P_SALES_INFO_1_SP_SALES_INFO_1
P_SALES_INFO_1               P_SALES_INFO_1_SP_SALES_INFO_2
P_SALES_INFO_1               P_SALES_INFO_1_SP_SALES_INFO_3
P_SALES_INFO_2               P_SALES_INFO_2_SP_SALES_INFO_1
P_SALES_INFO_2               P_SALES_INFO_2_SP_SALES_INFO_2
P_SALES_INFO_2               P_SALES_INFO_2_SP_SALES_INFO_3
P_SALES_INFO_3               P_SALES_INFO_3_SP_SALES_INFO_1
P_SALES_INFO_3               P_SALES_INFO_3_SP_SALES_INFO_2
P_SALES_INFO_3               P_SALES_INFO_3_SP_SALES_INFO_3

--选择其中一个子分区指定查询数据
SELECT * FROM sales_info SUBPARTITION (P_SALES_INFO_1_SP_SALES_INFO_2);
YEAR  MONTH BRANCH PRODUCT      QUANTITY      AMOUNT SALSPERSON
----- ----- ------ --------- ----------- ----------- -------------
2021  10    0402   11001              20         300
```

subquery
该语句用于指定一个子查询结果。

示例（单机、共享集群部署）

```
--为sales_info表创建一个同义词，也可指定分区来查询其数据
CREATE SYNONYM sy_sales_info FOR sales_info;
SELECT * FROM sy_sales_info PARTITION(p_sales_info_1);
YEAR  MONTH BRANCH PRODUCT      QUANTITY      AMOUNT SALSPERSON
----- ----- ------ --------- ----------- ----------- -------------
2021  10    0402   11001              20         300

--用于子查询中
SELECT b.branch_name 机构,
SUM(s.amount) 销售额
FROM branches b,
(SELECT * FROM sy_sales_info PARTITION(p_sales_info_2)) s
WHERE b.branch_no=s.branch
GROUP BY b.branch_name;
机构                                                             销售额
---------------------------------------------------------------- -----------
成都                                                                 500
南京                                                                 600
```

指定切片查询

对于LSC表，可以通过指定Slice ID查询某个切片的数据，具体规则为：

- 只能对LSC表指定切片查询，否则报错。
- 对于分区表，指定切片的同时，必须指定分区，否则报错。
- Slice ID为0表示查询可变数据区（MCOL）的数据。
- 通过查询V$LSC_SLICE_STAT视图可获得某张LSC表（分区）包含的所有Slice ID。

示例（LSC表）

```
--sales_info表的三个分区中分别存在如下稳态数据
SELECT * FROM sales_info PARTITION(p_sales_info_1);
YEAR  MONTH BRANCH PRODUCT      QUANTITY      AMOUNT SALSPERSON
----- ----- ------ --------- ----------- ----------- -------------
2021  10    0402   11001              20         300

SELECT * FROM sales_info PARTITION(p_sales_info_2);
```

```
YEAR  MONTH BRANCH PRODUCT      QUANTITY      AMOUNT SALSPERSON
----- ----- ------ --------- ----------- ----------- -------------
2001  01    0201   11001              30         500 0201010011
2000  12    0102   11001              20         300
2015  03    0102   11001              20         300

SELECT * FROM sales_info PARTITION(p_sales_info_3);
YEAR  MONTH BRANCH PRODUCT      QUANTITY      AMOUNT SALSPERSON
----- ----- ------ --------- ----------- ----------- -------------
2021  05    0101   11001              40         600
2015  11    0101   11001              20         300

--新插入sales_info表一条数据，将存储在可变数据区
INSERT INTO sales_info VALUES ('2002','03','0301','11001',40,700,'');

--查询V$LSC_SLICE_STAT视图获得Slice ID
SELECT utp.table_name, utp.partition_name, lss.slice_id
FROM USER_TAB_PARTITIONS utp, V$LSC_SLICE_STAT lss, USER_OBJECTS uo
WHERE utp.table_name = 'SALES_INFO_RANGE'
AND uo.subobject_name = utp.partition_name
AND uo.object_id = lss.obj;
TABLE_NAME             PARTITION_NAME              SLICE_ID
---------------------- ----------------------- ------------
SALES_INFO_RANGE       P_SALES_INFO_RANGE_1               0
SALES_INFO_RANGE       P_SALES_INFO_RANGE_2               0
SALES_INFO_RANGE       P_SALES_INFO_RANGE_3               0


--查询指定切片的数据
SELECT * FROM sales_info PARTITION(p_sales_info_1) SLICE(1024);
YEAR  MONTH BRANCH PRODUCT      QUANTITY      AMOUNT SALSPERSON
----- ----- ------ --------- ----------- ----------- -------------
2001  01    0201   11001              30         500 0201010011

--查询MCOL数据
SELECT * FROM sales_info PARTITION(p_sales_info_1) SLICE(0);
YEAR  MONTH BRANCH PRODUCT      QUANTITY      AMOUNT SALSPERSON
----- ----- ------ --------- ----------- ----------- -------------
2002  03    0301   11001              40         700
```

table_collection_expression

该语句可用于执行如下两种情况：

- UDT
- 表函数

UDT

可用于查询Varray UDT或Nested Table UDT数据，UDT数据类型的详细描述请参考用户自定义类型。

执行此类查询时，collection_expression为一个Varray对象或Nested Table对象，语法格式为表别名.UDT列名。

Varray对象依赖于所在表，因此应保证执行计划最先访问主表。

示例（HEAP表）

```
DROP TABLE IF EXISTS city_intro;
CREATE OR REPLACE TYPE brc_array AS VARRAY(20) OF CHAR(15);
/
CREATE OR REPLACE TYPE city_table AS TABLE OF CHAR(10);
/

--创建包含Varray UDT和Nested Table UDT列的city_intro表
CREATE TABLE city_intro (id INT, branches brc_array, citys city_table)
NESTED TABLE citys STORE AS nt_citys;

--插入数据
```

```
INSERT INTO city_intro
VALUES (1,
        brc_array('branch01','branch02','branch03'),
        city_table('shenzhen','guangzhou','dongguan'));

--查询数据
SELECT /*+ LEADING(c) */ c.id, cbranch.*, ccity.*
FROM city_intro c,
TABLE(c.branches) cbranch,
TABLE(c.citys) ccity;
          ID COLUMN_VALUE      COLUMN_VALUE
------------ ----------------- -------------
           1 branch01          shenzhen
           1 branch02          shenzhen
           1 branch03          shenzhen
           1 branch01          guangzhou
           1 branch02          guangzhou
           1 branch03          guangzhou
           1 branch01          dongguan
           1 branch02          dongguan
           1 branch03          dongguan
```

表函数

可使用表函数进行查询，表函数的具体描述可见内置函数章节中内置表函数部分描述。

执行此类查询时，collection_expression为所需表函数。

sample_clause

SAMPLE用于从表中按比例抽取一个随机样本，后续WHERE条件将基于这个样本数据而不是整张表数据。

本语句可作用于单个物理表，或者基于单个物理表的视图，对于多表连接的视图执行SAMPLE将会报错。

分布式部署中不可使用本语句。

sample_percent

指定样本所占的百分比，其值必须为一个[0.000001, 100]区间的常量数值（支持可以隐式转换为数值的其他类型）。此百分比代表采样时每个数据行被选为样本的概率，该概率为统计学上的概念，这意味着系统并不会精确返回基于sample_percent计算出的行数的记录。

SEED seed_value

本语句定义一个种子值，系统将尝试对相同的种子值返回相同的样本，未指定本语句时系统每次返回的样本是随机的。seed_value的值必须为一个[0, 4294967295]区间的常量整数（支持可以隐式转换为数值的其他类型，支持将小数截断为整数），其中大于4294967295的数值按4294967295处理。

示例（HEAP表）

```
--随机返回样本
SELECT * FROM area SAMPLE(60);
AREA_NO AREA_NAME                DHQ
------- ------------------------ --------------------
03      华南                     Guangzhou
04      华北                     Beijing
05      华中                     Wuhan

--指定相同seed返回相同样本
SELECT * FROM area SAMPLE(60) SEED(7);
AREA_NO AREA_NAME                DHQ
------- ------------------------ --------------------
01      华东                     Shanghai
02      华西                     Chengdu
03      华南                     Guangzhou
04      华北                     Beijing
05      华中                     Wuhan

SELECT * FROM area SAMPLE(60) SEED(10);
AREA_NO AREA_NAME                DHQ
------- ------------------------ --------------------
01      华东                     Shanghai
03      华南                     Guangzhou
```

```
04      华北                Beijing

SELECT * FROM area SAMPLE(60) SEED(7);
AREA_NO AREA_NAME               DHQ
------- ----------------------- --------------------
01      华东                    Shanghai
02      华西                    Chengdu
03      华南                    Guangzhou
04      华北                    Beijing
05      华中                    Wuhan
```

flashback_query_clause

YashanDB支持对近期历史数据的追溯查询，例如某条记录在某个时间点已被UPDATE或DELETE更改，但通过flashback_query_clause可以获得该时间点之前一段时间范围内（这取决于undo空间的清理规则），该条记录被更改之前的数据值。

可以使用SCN号或TIMESTAMP标签来定位到想要追溯查询的行记录。

系统在记录TIMESTAMP时使用的是北京时间。

分布式部署中不可使用本语句。

示例（单机、共享集群部署）

```
--area表中存在的一条记录
SELECT * FROM area WHERE area_no='03';
AREA_NO AREA_NAME          DHQ
------- ------------------ ------------
03      华南             Guangzhou

--获取当前时间
SELECT SYSTIMESTAMP res FROM dual;
RES
---------------------------------------------------------------
2023-12-17 14:14:08.498126

--删除此条记录并提交
DELETE FROM area WHERE area_no='03';
COMMIT;
SELECT * FROM area WHERE area_no='03';
AREA_NO AREA_NAME          DHQ
------- ------------------ ------------


--利用flashback可查询到快照历史数据
SELECT * FROM area AS OF TIMESTAMP TIMESTAMP('2023-12-17 14:14:08.498126')
WHERE area_no='03';
AREA_NO AREA_NAME          DHQ
------- ------------------ ------------
03      华南             Guangzhou
```

pivot_clause

PIVOT用于行列转换，通过从结果集中抽出FOR后的列，根据IN里的常量表达式列表构成新的投影列，并结合聚合函数输出对应聚合列的结果。

PIVOT可以对任意结果集（即单表查询或多表连接的结果）进行操作。

pivot_clause前查询的结果集若为多表，只允许使用JOIN ON进行显式join，不允许使用逗号进行隐式join。

IN后的列名的长度大于20字符时，输出的列名会自动截取长度20后的字符。

column_expression

必须为简单列名，即不允许使用table.column的形式。

const_expression

必须为常量表达式。

aggregate_expression

aggregate_expression的聚合列与FOR后的列必须出现在查询的投影列中。

aggregate_expression中使用的聚合函数不允许出现GROUPING_ID，GROUP_ID，GROUPING函数以及GROUPING SETS的相关函数（GROUP_CONCAT，WM_CONCAT，LISTAGG，STRINGAGG），非聚合函数或无聚合函数时报错。

aggregate_expression的聚合列与IN后的列的别名长度上限为64字符，超过64字符时报错。

示例

```
-- 抽取区域编号列，并统计对应区域的数量
SELECT * FROM (SELECT branch_name, area_no FROM branches)
PIVOT(
  COUNT(branch_name)
  FOR area_no
  IN ('01' AS 华东, '02' AS 华西, '03' AS 华南, '04' AS 华北, '05' AS 华中)
);
华东                 华西                 华南                 华北                 华中
-------------------- -------------------- -------------------- -------------------- --------------------
                   4                    1                    0                    4                    1


-- 显示华东和华西地区最大编号与城市数量
SELECT * FROM (SELECT branch_no, branch_name, area_no FROM branches)
PIVOT(
  COUNT(branch_name) 城市数量,
  MAX(branch_no) 最大编号
  FOR area_no
  IN ('01' AS 华东, '02' AS 华西)
);
华东_城市数量 华东_最大编号    华西_城市数量 华西_最大编号
-------------------- -------------------- -------------------- --------------------
                   4 0104                                 1 0201
```

join_clause

该语句用于将多个表、视图、AC和子查询进行连接（Join）查询，有如下几种Join方式：

- 显示指定JOIN关键字进行连接查询，并通过指定INNER|OUTER关键字分为内连接（Inner Join）和外连接（Outer Join）两种方式，未指定时缺省为内连接。
- 在FROM后面将各表、视图、子查询以`,`分隔，而在WHERE后指定条件以获取交叉连接后的查询结果，此种方式等同于内连接。
- 在WHERE后条件子句中指定(+)操作符，或者在JOIN ON后子句中指定(+)操作符，表示外连接。

分布式部署中遵循如下规则：

- 多表连接查询分布式系统视图和DBA视图、系统表时，使用的是各节点本地数据。
- 分布式视图仅允许和分布式视图、DBA视图、USER视图、ALL视图、系统表和单机视图进行多表查询。
- 多表查询分布式系统视图时，如使用order by、group by、窗口函数、聚集函数和limit，都仅在各节点内生效，最终结果可能不满足有序等约束。

inner_cross_join_clause

内连接查询是一种交叉连接查询方式，对A和B进行内连接查询时，将A的每一行与B的每一行按照连接条件进行比较，返回的是A、B均满足条件的结果。其中：

- INNER：产生的结果是AB的交集。此时必须在后面指定ON条件。
- CROSS：产生的结果是AB的笛卡尔积。此时不可以在后面指定ON条件。

示例（分布式部署）

```
--INNER JOIN,INNER关键字可省略
SELECT b.branch_name, a.area_name
FROM branches b
INNER JOIN area a
```

```
ON a.area_no = b.area_no
WHERE b.branch_no LIKE '01%';
BRANCH_NAME       AREA_NAME
----------------- --------------------
上海               华东
南京               华东
福州               华东
厦门               华东

--等同于上面的内连接
SELECT b.branch_name, a.area_name
FROM branches b, area a
WHERE a.area_no=b.area_no AND b.branch_no LIKE '01%';
BRANCH_NAME       AREA_NAME
----------------- --------------------
上海               华东
南京               华东
福州               华东
厦门               华东

--CROSS JOIN
SELECT b.branch_name, a.area_name
FROM branches b
CROSS JOIN area a
WHERE b.branch_no LIKE '01%';
BRANCH_NAME       AREA_NAME
----------------- --------------------
上海               华东
上海               华西
上海               华南
上海               华北
上海               华中
南京               华东
南京               华西
南京               华南
南京               华北
南京               华中
福州               华东
福州               华西
福州               华南
福州               华北
福州               华中
厦门               华东
厦门               华西
厦门               华南
厦门               华北
厦门               华中

--分布式视图查询
SELECT * FROM (SELECT * FROM DV$SYSTEM_PARAMETER WHERE GROUP_ID=0);
GROUP_ID     GROUP_NODE_ID      NAME               VALUE
------------ ------------------ ------------------ --------------------------
```

outer_join_clause

外连接查询指定了一张主表，对主表中的记录不要求在对方表里能找到匹配记录，而是全部做为有效结果返回，在返回结果集中对于对方未找到匹配值的列项设为NULL值。外连接需要显式地指定左连接或右连接。

outer_join_type

LEFT [OUTER] JOIN

左连接查询，当A LEFT OUTER JOIN B时，以A为主表，A的每一行记录均会出现在查询结果集中，对于在B中匹配不到的值设为NULL。

RIGHT [OUTER] JOIN

右连接查询，当A RIGHT OUTER JOIN B时，以B为主表，B的每一行记录均会出现在查询结果集中，对于在A中匹配不到的值设为NULL。

FULL [OUTER] JOIN

全连接查询，当A FULL OUTER JOIN B时，A和B的每一行记录均会出现在查询结果集中，对于在A中和B中匹配不到的值设为NULL。

示例

```
--左连接，对于branches表中的武汉，未在area表中匹配到对应的地区记录，则在结果集中将地区设为NULL
SELECT b.branch_name, a.area_name
FROM branches b
LEFT OUTER JOIN area a
ON a.area_no = b.area_no
WHERE b.branch_no LIKE '01%' OR b.branch_no LIKE '05%';
BRANCH_NAME        AREA_NAME
------------------ ---------------
上海               华东
南京               华东
福州               华东
厦门               华东
武汉
长沙               华中

--右连接，对于area表中的华南地区，未在branches表中匹配到下属的城市记录，则在结果集中将城市设为NULL
SELECT b.branch_name, a.area_name
FROM branches b
RIGHT OUTER JOIN area a
ON a.area_no = b.area_no
WHERE a.area_no IN ('01','03','05');
BRANCH_NAME         AREA_NAME
------------------- -------------
厦门                华东
福州                华东
南京                华东
上海                华东
                      华南
长沙                华中

--全连接，对于area表中的华南地区，未在branches表中匹配到下属的城市记录，则在结果集中将城市设为NULL
SELECT b.branch_name, a.area_name
FROM branches b
FULL OUTER JOIN area a
ON a.area_no = b.area_no
WHERE a.area_no IN ('01','03','05');
BRANCH_NAME         AREA_NAME
------------------- -------------
厦门                华东
福州                华东
南京                华东
上海                华东
                      华南
长沙                华中
```

(+) operator

YashanDB支持通过(+)操作符指定外连接。

语法位置

(+)可以使用在where_clause和join_clause的condition子句的比较条件中。

使用方法

在比较条件中，(+)作用于某一个列字段上，表示如果比较条件无法满足，则对该列对应的表进行补空，与外连接同义，即比较条件对端所涉及的表（可能为一个或多个）对该表进行外连接。

使用限制

1. (+)操作符只能作用于列字段上，不可以作用于表达式。
2. 在where_clause条件中使用(+)操作符时，不能与ANSI标准的Join语法一起使用。
3. 在join_clause条件中使用(+)操作符时，连接的两个表必须为普通表，不能为多表Join之后的中间结果。

4. 不允许使用(+)操作符对两个表互相外连接。
5. 某个条件中存在(+)操作符时，该条件的同级条件之间只允许使用AND关键词连接。

示例

```
--(+)作用于area表的area_no上，意为branches表对area表进行左外连接，结果集对area表中没有匹配上的数据进行补空。
--对于branches表中的武汉，未在area表中匹配到对应的地区记录，则在结果集中将地区设为NULL
SELECT a.area_name,b.branch_name
FROM area a, branches b
WHERE b.area_no = a.area_no(+)
AND (b.branch_no LIKE '01%' OR b.branch_no LIKE '05%');
AREA_NAME        BRANCH_NAME
---------------- -------------
华东             上海
华东             南京
华东             福州
华东             厦门
                   武汉
华中             长沙

SELECT a.area_name, b.branch_name
FROM branches b
JOIN area a
ON b.area_no = a.area_no(+)
WHERE b.branch_no LIKE '01%' OR b.branch_no LIKE '05%';
AREA_NAME        BRANCH_NAME
---------------- -------------
华东             上海
华东             南京
华东             福州
华东             厦门
                   武汉
华中             长沙
```

where_clause

该语句用于指定查询的condition。

hierarchical_query_clause

该语句用于指定层次（父子）关系的条件查询。通过START WITH获取根数据，通过 CONNECT BY指定父子关系，即可以将所有满足层次关系的数据全部查找出来。

列存表不适用本语句。

YashanDB提供如下虚拟列（非表结构中定义的列，但可以被当作列字段进行查询）供层次关系查询中使用：

- **LEVEL**：LEVEL表示层次查询形成的树结构中的当前层数。该列值一直从1开始，即START WITH对应的数据的层数一直为1，之后子孙节点的LEVEL值依次递增。
- **CONNECT_BY_ISLEAF**：CONNECT_BY_ISLEAF表示当前数据是否是层次查询形成的树结构中的叶子节点。若是叶子节点值为1，否则为0。
- **CONNECT_BY_ISCYCLE**：CONNECT_BY_ISCYCLE表示在层次查询中当前数据是否会导致形成环，即根据层次关系，当前层数据是否存在其叶子节点也是其父节点。该列只有在同时指定NOCYCLE关键字时才有意义，当前数据会导致形成环则结果为1，否则为0。

YashanDB提供如下标识符用来指定层次关系中的某个节点属性：

- **PRIOR**：PRIOR操作符之后的参数将作为层次查询中的父节点。参数不能为虚拟列、层次查询函数、操作符、伪列及子查询。PRIOR通常应用于connect_by_clause中，且不能在start_with_clause中使用。
- **CONNECT_BY_ROOT**：CONNECT_BY_ROOT操作符之后的参数将作为在层次查询中的根节点。参数不能为虚拟列、层次查询函数、操作符、伪列及子查询。CONNECT_BY_ROOT不能在connect_by_clause和start_with_clause中使用。

YashanDB提供如下层次查询中的专用函数：

- **SYS_CONNECT_BY_PATH**：`SYS_CONNECT_BY_PATH(col_name, delimiter)`，其中delimiter 表示分隔符，只能使用字面量。该函数将获取从根节点到当前节点的路径上所有节点名为col_name的值，中间用delimiter进行分隔开。SYS_CONNECT_BY_PATH不能在connect_by_clause、start_with

_clause和group_by_clause中使用。

connect_by_clause

通过CONNECT BY后跟的condition，指定层次关系查询条件，其中必须至少有一个条件用于指定父子关系，该条件通过PRIOR标识符识别。

nocycle

当根据指定的条件关系获得的查询结果存在环时，如不指定NOCYCLE，系统将返回错误，指定时系统将忽略环的问题，仍返回所有数据。

start_with_clause

通过START WITH后跟的condition，指定以满足该条件的数据作为层次关系中根节点，其LEVEL为1。condition中不允许使用rownum。

本语句可省略，则表示将所有数据均作为根节点进行层次关系查询。

使用本语句时，其前面或后面必须存在connect_by_clause，不能独立使用。

order_siblings_by_clause

对同一父节点下的同一层次数据，通过ORDER SIBLINGS BY指定排序列和排序规则进行排序，本语句功能同order_by_clause。

使用本语句时，其前面必须存在connect_by_clause，不能独立使用。

示例（HEAP表）

```
--建立包含层次关系的地区表area_info
CREATE TABLE area_info (id INT, father_id INT, area_name VARCHAR(20));
INSERT INTO area_info VALUES(1,   0, '广东');
INSERT INTO area_info VALUES(755, 1, '深圳');
INSERT INTO area_info VALUES(756, 755, '龙华');
INSERT INTO area_info VALUES(757, 755, '福田');
INSERT INTO area_info VALUES(2,   0, '浙江');
INSERT INTO area_info VALUES(571, 2, '杭州');
COMMIT;

--显示地区层次关系（指定根节点）
SELECT id, father_id, LEVEL,
CONNECT_BY_ROOT area_name AS name,
SYS_CONNECT_BY_PATH(area_name, '/') path
FROM area_info
CONNECT BY id<>757
AND PRIOR id = father_id
START WITH father_id = 0;
   ID    FATHER_ID      LEVEL NAME             PATH
----- ------------ ---------- ---------------- ------------------
    1            0          1 广东             /广东
  755            1          2 广东             /广东/深圳
  756          755          3 广东             /广东/深圳/龙华
    2            0          1 浙江             /浙江
  571            2          2 浙江             /浙江/杭州

--显示地区层次关系（不指定根节点，则所有数据都将作为根节点）
SELECT id, father_id, LEVEL,
CONNECT_BY_ROOT area_name AS name,
SYS_CONNECT_BY_PATH(area_name, '/') path
FROM area_info
CONNECT BY id<>757
AND PRIOR id = father_id;
    ID    FATHER_ID      LEVEL NAME          PATH
------ ------------ ---------- ------------- ----------------
     1            0          1 广东          /广东
   755            1          2 广东          /广东/深圳
   756          755          3 广东          /广东/深圳/龙华
   755            1          1 深圳          /深圳
   756          755          2 深圳          /深圳/龙华
     2            0          1 浙江          /浙江
   571            2          2 浙江          /浙江/杭州
   571            2          1 杭州          /杭州
   756          755          1 龙华          /龙华
```

```
   757          755          1 福田        /福田

--显示节点是否为叶子节点及是否出现循环
SELECT id, father_id, LEVEL,
CONNECT_BY_ISLEAF AS leaf, CONNECT_BY_ISCYCLE AS iscycle
FROM area_info
CONNECT BY NOCYCLE id<>757
AND PRIOR id = father_id
START WITH father_id = 0;
    ID    FATHER_ID     LEVEL LEAF      ISCYCLE
------ ------------ --------- --------- -----------
     1            0         1 false     false
   755            1         2 false     false
   756          755         3 true      false
     2            0         1 false     false
   571            2         2 true      false

--按指定列对同层次数据排序
SELECT  id, father_id, LEVEL,
CONNECT_BY_ROOT area_name AS name,
SYS_CONNECT_BY_PATH(area_name, '/') path
FROM area_info
CONNECT BY PRIOR id = father_id START WITH father_id = 0
ORDER SIBLINGS BY id DESC;
    ID    FATHER_ID    LEVEL NAME         PATH
------ ------------ -------- ------------ ---------------
     2            0        1 浙江        /浙江
   571            2        2 浙江        /浙江/杭州
     1            0        1 广东        /广东
   755            1        2 广东        /广东/深圳
   757          755        3 广东        /广东/深圳/福田
   756          755        3 广东        /广东/深圳/龙华
```

group_by_clause

该语句用于对查询结果集进行按条件的分类聚集（Aggregation）。

group by

在GROUP BY后定义分组列，多个列用`,`分隔，同时需满足如下规则：

- 在select_list中出现的查询列或列数据，必须为分组列或列数据的子集。

列是子集："SELECT col ,COUNT(*) FROM table GROUP BY col, col2；"

列数据是子集："SELECT LPAD(col), COUNT(*) FROM table GROUP BY col；"

- 查询列与分组列里出现函数时，函数的参数必须一致。

此语句错误："SELECT SUBSTR(col, 1,1), COUNT(*) FROM table GROUP BY SUBSTR(col, 1,2);"

- 如果同时出现了DISTINCT或ORDER BY子语句，则在它们中出现的列遵循上两条规则。
- 分组列不能为*（星号），SEQUENCE，子查询及聚集函数。
- 当分组列为数字时，与ORDER BY不同的是，本语句不会将数字释义成列的位置，而是作为字面量处理。

having

HAVING子句约束SELECT查询语句中GROUP BY的结果，该约束应用于查询结果中的每个分组，类似WHERE条件应用于select_list。使用规则如下：

- HAVING子句可以放在GROUP BY子句的前面或后面。
- HAVING后的condition是一个布尔表达式，语法同WHERE子句中的filter_clause，但是它只能包含分组列、聚集函数（可以与select_list中的聚集函数不一致）、字面量和子查询（子查询中的列不需要为分组列）。其中分布式部署中不可以使用子查询。
- 如果没有GROUP BY，直接使用HAVING子句，表示该约束作用于整个查询结果，此时select_list和condition中不能出现分组列。

示例（单机、分布式部署）

```
--单机部署中，按年份对销售额进行汇总，HAVING后可以指定其他聚集函数，且可以指定对其他表的子查询
SELECT 1,year,month,CONCAT(year,month),SUM(amount) FROM sales_info HAVING MAX(amount)>100 GROUP BY year,month;
          1 YEAR  MONTH CONCAT(YEAR,MONTH)          SUM(AMOUNT)
```

```
----------- ----- ----- ------------------ --------------------
          1 2000  12    200012                              300
          1 2001  01    200101                              500
          1 2015  03    201503                              300
          1 2015  11    201511                              300
          1 2021  05    202105                              600
          1 2021  10    202110                              300

SELECT 1,year,month,CONCAT(year,month),SUM(amount)
FROM sales_info
GROUP BY year,month
HAVING SUM(amount)>(SELECT MIN(price*50) FROM product);
          1 YEAR  MONTH CONCAT(YEAR,MONTH)          SUM(AMOUNT)
----------- ----- ----- ------------------ --------------------
          1 2021  05    202105                              600

--HAVING定义分组列的条件
SELECT 1,year,month,CONCAT(year,month),SUM(amount)
FROM sales_info
GROUP BY year,month
HAVING year>'2015';
          1 YEAR  MONTH CONCAT(YEAR,MONTH)          SUM(AMOUNT)
----------- ----- ----- ------------------ --------------------
          1 2021  05    202105                              600
          1 2021  10    202110                              300

--单独使用HAVING语句，定义对整个查询结果的聚集约束条件,此时不能出现分组列
SELECT 1,SUM(amount)
FROM sales_info
HAVING MAX(amount)>100;
          1           SUM(AMOUNT)
----------- --------------------
          1                 2300

--分布式部署中，按年份对销售额进行汇总，HAVING后指定相同聚集函数
SELECT 1,year,month,CONCAT(year,month),SUM(amount)
FROM sales_info
HAVING SUM(amount)>300
GROUP BY year,month;
          1 YEAR  MONTH CONCAT(YEAR,MONTH) SUM(AMOUNT)
----------- ----- ----- ------------------ -----------
          1 2001  01    200101                      500
          1 2021  05    202105                      600

SELECT 1,year,month,CONCAT(year,month),SUM(amount)
FROM sales_info
GROUP BY year,month
HAVING SUM(amount)>500;
          1 YEAR  MONTH CONCAT(YEAR,MONTH)          SUM(AMOUNT)
----------- ----- ----- ------------------ --------------------
          1 2021  05    202105                              600

--HAVING定义分组列的条件
SELECT 1,year,month,CONCAT(year,month),SUM(amount)
FROM sales_info
GROUP BY year,month
HAVING year>'2015';
          1 YEAR  MONTH CONCAT(YEAR,MONTH)          SUM(AMOUNT)
----------- ----- ----- ------------------ --------------------
          1 2021  05    202105                              600
          1 2021  10    202110                              300

--单独使用HAVING语句，定义对整个查询结果的聚集约束条件,此时不能出现分组列
SELECT 1,SUM(amount)
FROM sales_info
HAVING SUM(amount)>300;
          1           SUM(AMOUNT)
----------- --------------------
          1                 2300
```

rollup_cube_clause

rollup

该关键字用于指定ROLLUP执行算子，该算子用于拓展GROUP BY聚集功能，区别在于GROUP BY仅返回每个分组的结果，指定了ROLLUP关键字会返回总计和每个分组的结果。

其等价于对ROLLUP后指定的列字段的每个层次级别创建grouping set，如 `ROLLUP(A,B,C) == GROUPING SETS((A, B, C), (A, B), (A), ())` 。

本语句不适用于HEAP表。

示例（LSC表、TAC表）

```
SELECT year,product,salsperson,SUM(quantity) FROM sales_info GROUP BY ROLLUP(year,product,salsperson);

YEAR  PRODUCT   SALSPERSON    SUM(QUANTITY)
----- --------- ------------- -------------
2000  11001                             20
2001  11001     0201010011              30
2015  11001                             40
2021  11001                             60
2000  11001                             20
2001  11001                             30
2015  11001                             40
2021  11001                             60
2000                                    20
2001                                    30
2015                                    40
2021                                    60
                                       150
```

cube

该关键字用于指定CUBE执行算子，该算子用于拓展GROUP BY聚集功能，区别在于GROUP BY仅返回每个分组的结果，指定了CUBE关键字会返回所有组合的结果，其中包括每个分组的结果。

其等价于对CUBE后指定的列字段的所有组合创建grouping set，如 `CUBE(A,B,C) == GROUPING SETS((A, B, C), (A, B), (A, C), (B, C), (A), (B), (C), ())` 。

本语句不适用于HEAP表。

示例（LSC表、TAC表）

```
SELECT year,product,salsperson,SUM(quantity) FROM sales_info GROUP BY CUBE(year,product,salsperson);

YEAR  PRODUCT   SALSPERSON    SUM(QUANTITY)
----- --------- ------------- -------------
2000  11001                             20
2001  11001     0201010011              30
2015  11001                             40
2021  11001                             60
2000  11001                             20
2001  11001                             30
2015  11001                             40
2021  11001                             60
      11001                            120
      11001     0201010011              30
      11001                            150
2000                                    20
2001            0201010011              30
2015                                    40
2021                                    60
2000                                    20
2001                                    30
2015                                    40
2021                                    60
```

```
                                    150
                                    120
                0201010011           30
```

grouping_sets_clause

该语句用于指定GROUP BY的分组规则，并将结果聚合，等价于对指定组合执行GROUP BY分组，然后通过UNION ALL将结果联合起来。

本语句不适用于HEAP表。

示例（LSC表、TAC表）

```
SELECT year,product,salsperson,SUM(quantity) FROM sales_info GROUP BY GROUPING SETS(year,product,salsperson);

YEAR  PRODUCT   SALSPERSON    SUM(QUANTITY)
----- --------- ------------- -------------
                                    120
                0201010011           30
2000                                 20
2001                                 30
2015                                 40
2021                                 60
      11001                         150
```

order_by_clause

该语句用于对查询结果集按照排序键进行排序。

排序键可以为：

- expr：表达式
- position：select_list中的列项位置，默认按此项排序
- c_alias：select_list中为列项定义的别名

排序键不允许为如下类型：

- CLOB
- BLOB
- NCLOB
- BIT
- ROWID
- CURSOR
- JSON
- UDT

nulls (first|last)

对于空值，用NULLS FIRST|NULLS LAST语句来指定空值排列在最前或最后。当未指定此语句时，对升序排列缺省为NULLS LAST，对降序排列缺省为NULLS FIRST。

asc|desc

排序方式可指定为升序（ASC）或降序（DESC），未指定则缺省为升序。

示例

```
SELECT year,branch,SUM(amount) FROM sales_info GROUP BY year,branch ORDER BY SUM(amount)/SUM(quantity);
YEAR  BRANCH           SUM(AMOUNT)
----- ------ --------------------
2015  0101                    300
2015  0102                    300
2000  0102                    300
2021  0101                    900
2001  0201                    500
```

```
SELECT year,branch,SUM(amount) FROM sales_info GROUP BY year,branch ORDER BY 2;
YEAR  BRANCH           SUM(AMOUNT)
----- ------ --------------------
2015  0101                    300
2021  0101                    900
2015  0102                    300
2000  0102                    300
2001  0201                    500
```

row_limiting_clause

该语句用于抓取查询结果集的指定行。

limit

指定抓取的行数。LIMIT满足如下规则：

- LIMIT可以为一个表达式，表达式的结果必须为一个数字。
- 查询HEAP表时，LIMIT可以使用标量子查询和绑定参数。
- 如果LIMIT是负值，则将其视为0。
- 如果LIMIT是NULL，则不返回任何行。
- 如果LIMIT大于查询的行数，则返回所有行。
- 如果LIMIT是小数，则小数部分被截断。

offset

指定抓取的起始行偏离数，缺省为第一行的偏离数0。OFFSET满足如下规则：

- OFFSET可以为一个表达式，表达式的结果必须为一个数字。
- 查询HEAP表时，OFFSET可以使用标量子查询和绑定参数。
- 如果OFFSET是负值，则将其视为0。
- 如果OFFSET是NULL，或者大于查询的行数，则不返回任何行。
- 如果OFFSET是小数，则小数部分被截断。

执行本语句将对优化器产生以下影响：

- 如果SQL中同时出现了order_by_clause和row_limiting_clause，优化器会将排序计划改写成TOP SORT计划。
- 如果SQL中同时出现了group_by_clause、distinct、order_by_clause、以及row_limiting_clause，优化器将基于cost可能生成TOP SORT GROUP/TOP SORT DISTINCT计划。

示例

```
--从branches表的第一行开始抓取前四行数据
SELECT * FROM branches LIMIT 4;
BRANCH_NO BRANCH_NAME         AREA_NO ADDRESS
--------- ------------------- ------- -----------------
0001      深圳
0101      上海                01      上海市静安区
0102      南京                01      City of Nanjing
0103      福州                01

--从branches表的第四行开始抓取前四行数据
SELECT * FROM branches LIMIT 4 OFFSET 3;
BRANCH_NO BRANCH_NAME         AREA_NO ADDRESS
--------- ------------------- ------- -----------------
0103      福州                01
0104      厦门                01      Xiamen
0401      北京                04
0402      天津                04
```

offset_fetch_clause

OFFSET FETCH的功能实现和LIMIT OFFSET完全一致，用于抓取查询结果集的指定行。

rowcount

指定抓取的行数。ROWCOUNT满足如下规则：

- ROWCOUNT可以为一个表达式，表达式的结果必须为一个数字。
- 查询HEAP表时，LIMIT可以使用标量子查询和绑定参数。
- 如果ROWCOUNT是负值，则将其视为0。
- 如果ROWCOUNT是NULL，则不返回任何行。
- 如果ROWCOUNT大于查询的行数，则返回所有行。
- 如果ROWCOUNT是小数，则小数部分被截断。

offset

指定抓取的起始行偏离数，缺省为第一行的偏离数0。OFFSET满足如下规则：

- OFFSET可以为一个表达式，表达式的结果必须为一个数字。
- 查询HEAP表时，OFFSET可以使用标量子查询和绑定参数。
- 如果OFFSET是负值，则将其视为0。
- 如果OFFSET是NULL，或者大于查询的行数，则不返回任何行。
- 如果OFFSET是小数，则小数部分被截断。

示例

```
--从branches表的第一行开始抓取前四行数据
SELECT * FROM branches FETCH FIRST 4 ROW ONLY;
BRANCH_NO BRANCH_NAME         AREA_NO ADDRESS
--------- ------------------- ------- -----------------
0001      深圳
0101      上海                01      上海市静安区
0102      南京                01      City of Nanjing
0103      福州                01

--从branches表的第四行开始抓取前四行数据
SELECT * FROM branches OFFSET 3 ROW FETCH FIRST 4 ROW ONLY;
BRANCH_NO BRANCH_NAME         AREA_NO ADDRESS
--------- ------------------- ------- -----------------
0103      福州                01
0104      厦门                01      Xiamen
0401      北京                04
0402      天津                04
```

for_update_clause

该语句用于将查询结果集对应的行记录锁定，锁定后其他用户将只能对这些行进行查询，而不能执行锁定或修改等操作，直至当前事务结束。

只能在顶层的SELECT语句中指定FOR UPDATE，子查询和CTE不能指定FOR UPDATE。

本语句不可与DISTINCT，GROUP BY，FLASHBACK，聚集函数，游标表达式等组合使用。

of column_name

用OF后的列字段名来指定锁定该列字段所在的表，而不需要将FROM后的表都锁定。 本语句一般在多表连接情况下使用，指定对多表中特定的某些表执行FOR UPDATE。省略本语句时锁定当前查询中所有涉及到的表对象对应的记录。

通过column_name标识出其所在的表对象，必须是一个真实的列字段名称，不能为别名。

可联合表、视图、同义词，或者在FROM语句中定义的别名，来指定列字段名。

wait|no wait|skip locked

指定当要锁定的记录里至少有部分行正被其他用户锁定时的操作，缺省为一直等待到那些行被解锁后，再执行操作并返回。

- WAIT ntimes：等待指定的时间，ntimes的单位为秒，超过此时间后报错。
- NO WAIT：不等待且不执行操作，直接返回。
- SKIP LOCKED：跳过被锁定的行，继续执行其他的行。不适用于LSC表的稳态数据。

本语句用来定义行锁时的操作，当出现表锁时，直接进入表锁等待。

示例（单机HEAP表、TAC表）

```
SELECT TRIM(address) address FROM branches FOR UPDATE;

SELECT * FROM area a,branches b WHERE a.area_no=b.area_no AND a.area_no='01' FOR UPDATE OF a.DHQ;          --此时仅锁定area表中的对应行

SELECT TRIM(address) address FROM branches FOR UPDATE NOWAIT;

SELECT TRIM(address) address FROM branches FOR UPDATE WAIT 10;

SELECT TRIM(address) address FROM branches FOR UPDATE SKIP LOCKED;
```

SET AUTOTRACE

通用描述

SET AUTOTRACE为在yasql（YashanDB的SQL客户端工具）中实现的命令，用于生成trace报告，得到DML类型SQL语句（SELECT/INSERT/UPDATE/DELETE）的执行计划，以及执行计划实际产生的资源消耗和执行时间等信息。

当开启AUTOTRACE后，在客户端执行的DML类SQL语句，除输出SQL结果外，还将输出trace报告中的信息，信息内容依据AUTOTRACE的开启选项决定。

AUTOTRACE是常见的SQL调优工具，可在调优手册中获取更多信息。

语句定义

set autotrace::=

```
syntax::= SET AUTOT[RACE] {ON | OFF | TRACE[ONLY]} [EXP[LAIN]] [STAT[ISTICS]]
```

set autotrace off

默认模式，不生成trace报告。

示例

```
SET AUTOTRACE OFF;
```

set autotrace on explain

生成trace报告，且输出语句的执行结果和执行计划。

开启此设置后，后续执行的DML语句将输出执行计划，且执行计划中新增如下信息：

- A-ROWS：实际执行行数
- A-TIME：实际执行时间
- LOOP：数据提取次数
- MEMORY：算子占用的内存大小
- DISK：算子占用的磁盘空间大小
- PX RemoteInfo/PX LocalInfo：各节点数据交互的PX执行算子具体信息
  - total time：激活时间，即开始执行到结束时间，单位毫秒，取各子线程最大值
  - send_bytes：采样时间内发送的数据量，单位字节
  - send_packets：采样时间内发送包的个数
  - send_acks：采样时间内发送ack的个数
  - wait_space_times：采样时间内因发送缓存不足而等待的次数
  - wait_space_timeout：采样时间内因发送缓存不足而等待超时的次数
  - recv_bytes：采样时间内接收的数据量，单位字节
  - recv_packets：采样时间内接收包的个数
  - recv_acks：采样时间内接收ack的个数
  - wait_data_times：采样时间内接收端因数据未到来而等待的次数
  - wait_data_timeout：采样时间内接收端因数据未到来而等待超时的次数

上述新增相比的是在开启设置之前由EXPLAIN语句输出的内容，且在开启设置之后执行EXPLAIN语句将同样新增上述信息。

示例

```
SET AUTOTRACE ON EXPLAIN;
```

set autotrace on statistics

生成trace报告，且输出语句的执行结果和SQL执行统计信息（统计信息仅在STATISTICS_LEVEL参数为'ALL'模式下记录）。

开启此设置后，后续执行的DML语句将输出SQL执行所产生的redo日志大小、物理读、一致性读和递归调用等资源消耗数据。

示例

```
SET AUTOTRACE ON STATISTICS;
```

set autotrace on

生成trace报告，且输出语句的执行结果、执行计划和执行统计信息。

示例

```
SET AUTOTRACE ON;
```

set autotrace traceonly

生成trace报告，且不输出语句的执行结果，只输出语句的执行计划和执行统计信息。

示例

```
SET AUTOTRACE TRACEONLY;
```

SET TRANSACTION

通用描述

SET TRANSACTION用于设置事务的隔离级别，事务的隔离级别有：

- READ COMMITTED：读已提交，系统默认的隔离级别，事务每条语句支持语句级一致性读。
- CURRENT COMMITTED：读当前提交，为读已提交的一种，但不提供语句内的读一致性。
- SERIALIZABLE：序列化读，事务串行化，提供事务级一致性读，和完整的写写串行化冲突检测机制。

分布式部署中的事务隔离级别默认为READ COMMITTED，用户无法执行本语句。

语句定义

set transaction::=

```
syntax::= SET TRANSACTION [ISOLATION LEVEL (READ COMMITTED|CURRENT COMMITTED|SERIALIZABLE)] [NAME comment_string]
```

read committed

将事务隔离级别设置为读已提交。

示例（单机、共享集群部署）

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
```

current committed

将事务隔离级别设置当前提交。

示例（单机、共享集群部署）

```
SET TRANSACTION ISOLATION LEVEL CURRENT COMMITTED;
```

serializable

将事务隔离级别设置为串行化。

示例（单机、共享集群部署）

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
```

SHUTDOWN

通用描述

SHUTDOWN用于关闭当前正在运行的数据库。

只有拥有SYSDBA或SYSOPER角色的用户才可执行SHUTDOWN操作。

语句定义

shutdown::=

```
syntax::= SHUTDOWN [NORMAL | IMMEDIATE | ABORT] [WAIT STANDBY]
```

normal

默认的关闭选项，必须等待现有的会话结束后才执行关闭，在数据库重启之后不会进行实例恢复。

immediate

尽可能快速的关闭数据库，不会等待现有的会话结束，但是会回滚未提交的事务。 此种关闭模式下所有的客户端会话将被断开，且数据库重启之后不会进行实例恢复。

abort

尽可能快速的关闭数据库，不会等待现有的会话结束，也不回滚未提交的事务。 此种关闭模式下所有的客户端会话将被断开，在数据库重启之后会进行实例恢复。

wait standby

该选项表示关闭数据库前，日志要同步到所有备库，并且等待被备库回放。

示例

```
SHUTDOWN;

SHUTDOWN IMMEDIATE;

SHUTDOWN ABORT;

SHUTDOWN WAIT STANDBY;
```

TRUNCATE TABLE

通用描述

TRUNCATE TABLE语句用于删除表的所有数据行（索引，AC里的数据也将被删除），同时该表及索引，AC所占的数据空间将被释放并被重置到其初始大小（除非指定了REUSE STORAGE）。

TRUNCATE操作将会一次性删除表的所有数据，不能回滚（Rollback），也不能通过flashback_query_clause获得操作之前的数据。

对于被子表定义了外键约束的父表，如子表中已存在数据，则无法TRUNCATE此父表。

本文所列REUSE STORAGE选项不适用于分布式部署。

语句定义

truncate table::=

```
syntax::= TRUNCATE TABLE [schema "."] table_name [DROP STORAGE|REUSE STORAGE]
```

drop storage

该语句用于指定表被TRUNCATE后，其所占数据空间将被释放并被重置到该表的初始大小值。

reuse storage

该语句用于指定表被TRUNCATE后，其所占数据空间仍保留。

示例

```
TRUNCATE TABLE finance_info;
```

UPDATE

通用描述

UPDATE用于修改数据库的表中的数据，包括HEAP表、TAC表和LSC表，其中，对LSC表的UPDATE存在如下约束限制：

- 不能对LSC表执行跨分区更新。
- 不能对AC执行UPDATE操作。
- 更新LSC表冷数据的数据，需要开启ROW MOVEMENT。

在UPDATE的事务（Transaction）被提交（Commit）之前，其他会话无法查询到这些被修改的数据。在YashanDB里，可以通过打开自动提交（SET AUTOCOMMIT ON）的开关，这样其他会话将可以及时查询到这些数据。

对于被子表定义了外键约束的父表，如果要被更新的列字段为该外键项且其数据值已在子表中存在，则无法更新该父表中的此项数据。

语句定义

update::=

```
syntax::= UPDATE [hint] dml_table_expression_clause [t_alias] update_set_clause [WHERE condition]
```

dml_table_expression_clause**::=**

```
syntax::= [schema "."] table_name [dblink] [partition_extension_clause]
```

partition_extension_clause**::=**

```
syntax::= PARTITION ("(" partition ")"|FOR "(" (partition_key_value) {","(partition_key_value)} ")")|SUBPARTITION ("(" subpartition ")"|FOR "(" (subpartition_key_value) {","(subpartition_key_value)} ")")
```

update_set_clause**::=**

```
syntax::= SET (column "=" (expr|DEFAULT|"(" subquery ")")|("("(column){"," (column)}")" "=""("(((expr)|DEFAULT|("(" subquery ")")){","((expr)|DEFAULT|("(" subquery ")"))}")"))){","(column "=" (expr|DEFAULT|"(" subquery ")")|("("(column){"," (column)}")" "=""("(((expr)|DEFAULT|("(" subquery ")")){","((expr)|DEFAULT|("(" subquery ")"))}")"))))}
```

hint

该语句用于提出给定的方案到优化器（Optimizer ），使其按照此方案生成语句的执行计划。查看hint说明。

dml_table_expression_clause

该语句用于指定要更新数据的对象，即表（包括本地数据库的表或或远端表）或表分区，可对其指定一个别名。

对于分区表，如未显式指定分区对象，由系统根据分区项字段的值判断要更新的表分区。在显式指定且表未被定义为ENABLE ROW MOVEMENT时：

- 当分区表类型为RANGE时，如更新该表的分区项字段，请注意不要超出其所在分区的界值，否则会更新失败并提示错误。
- 当分区表类型为LIST时，如更新该表的分区项字段，请注意数据值在其所在分区的列表项中，否则会更新失败并提示错误。

对于分布式部署中的分布表，不允许修改其分区键字段的值。

示例1（单机/共享集群部署）

```
SELECT * FROM sales_info_range PARTITION (p_sales_info_range_1);
YEAR  MONTH BRANCH PRODUCT               QUANTITY               AMOUNT SALSPERSON
----- ----- ------ --------- -------------------- -------------------- -------------
2001  01    0201   11001                     30                  500 0201010011
2000  12    0102   11001                     20                  300
--要更新数据的此行记录位于sales_info_range 表的p_sales_info_range_1分区，其界值为'2011'，对year字段的超出界值更新将会失败
UPDATE sales_info_range PARTITION (p_sales_info_range_1) SET year='2023' WHERE year='2001';
```

```
YAS-02209 ROW MOVEMENT is not enabled

--未指定分区时，按分区项判断更新值不在当前分区列表中，更新数据失败
UPDATE sales_info_list SET year='2021' WHERE year='2018';
YAS-02209 ROW MOVEMENT is not enabled

 --指定分区时，按分区项判断其原值和更新值均在指定分区，更新数据成功
UPDATE sales_info_list PARTITION (p_sales_info_list_1) SET year='2019' WHERE year='2018';
SELECT * FROM sales_info_list PARTITION (p_sales_info_list_1);
YEAR  MONTH BRANCH PRODUCT               QUANTITY               AMOUNT SALSPERSON
----- ----- ------ --------- -------------------- -------------------- -------------
2019  10    0101   11001                     20                   300
```

示例2（分布式部署）

```
--sales_info_range为一张范围分区表，该表以branch字段作为分区键,对分布表的分区键执行update将失败
UPDATE sales_info_range SET branch='0101' WHERE branch='0201';
YAS-04510 cannot update partitioning column for sharded table
```

partition_extension_clause

与INSERT语句中partition_extension_clause的描述一致。

update_set_clause

该语句用于指定要更新数据的列字段，并对其赋值。

对于UDT列字段，通过对象初始化方法赋值，详见用户自定义类型中描述。

YashanDB支持通过两种方式更新多个列字段：

方式一：`UPDATE table_name SET column = value, column = value;`

方式二：`UPDATE table_name SET (column,...,column) = (value,...value);`

YashanDB支持按如下语句对列字段赋值：

- expr表达式
- 子查询
- DEFAULT

当上述语句的结果与列字段定义的数据类型不一致时，系统会先进行数据类型转换，转换失败则返回错误。

使用方式二更新多个列字段有如下约束限制：

- 括号内指定的列字段需位于同一张表，否则返回错误。
- 除对所有指定列字段均赋值为DEFAULT情况外，其他情况下值的数量需与列字段数量相同。
- value中不允许同时出现子查询和表达式。

示例

```
--在area表中插入数据
INSERT INTO area VALUES ('09','Other','Shanghai1');

--更新多个列字段（方式一）
UPDATE area SET area_name=23*67,DHQ='Shanghai1' WHERE area_no='09';

--result
SELECT * FROM area WHERE area_no='09';
AREA_NO AREA_NAME      DHQ
------- -------------- ------------
09      1541           Shanghai1

--更新多个列字段（方式二）
UPDATE area SET (area_name,DHQ) = ('华西','Shanxi') WHERE area_no='09';
```

其中，当使用子查询对列字段赋值时，更新的列字段须与子查询返回的列项必须按顺序一一对应，且子查询的返回结果不能为多条。如子查询无返回结果，将以NULL值返回，此时如对应的列存在非空约束，则更新数据失败。

示例

```
--将上述示例插入记录的DHQ更新为'Shanghai2'
UPDATE area SET DHQ='Shanghai2' WHERE area_no='09';

--在area表中未检索到DHQ='Shanghai1'的记录,以NULL值对DHQ字段进行更新触发非空约束错误
UPDATE area SET DHQ = (SELECT a.DHQ FROM area a WHERE a.DHQ='Shanghai1') WHERE area_no='09';
YAS-04006 cannot insert NULL value to column DHQ

--子查询返回多行记录，更新提示错误
UPDATE area SET DHQ = (SELECT a.DHQ FROM area a WHERE a.DHQ LIKE 'Shanghai%') WHERE area_no='09';
[1:24]YAS-04402 query expression return multiple rows
```

当使用DEFAULT对列字段赋值时，如对应的列上已定义了DEFAULT值，则更新的数据为该DEFAULT值，否则为NULL，基于这个规则，如对应的列存在非空约束，则更新数据失败。

示例

```
--area表上的DHQ字段已定义了DEFAULT值'ShenZhen',则按此值更新数据成功
UPDATE area SET DHQ=DEFAULT WHERE area_no='09';

--branches表上的BRANCH_NAME字段非空且无DEFAULT值，则按NULL值更新数据失败
UPDATE branches SET branch_name=DEFAULT WHERE branch_no='0201';
YAS-04006 cannot insert NULL value to column BRANCH_NAME
```

where condition

该语句用于指定condition，按此条件过滤出的记录行被执行更新操作。可省略，则表示更新表的所有行。

附：EBNF语法图

本产品文档中，在描述每一个SQL语句或PL语句定义时，依据其EBNF巴科斯范式，同时生成了可直观展现的语法图。本文将对这些语法图的基本构成元素进行解释。

最简单的语法图

示例：定义删除表的SQL关键字。

```
drop_table::= DROP TABLE table_name
```

|：定义多个分支，且必须选择一个分支

示例：定义更改表时可选择的操作，添加列字段或者删除列字段。

```
alter_table::= ALTER TABLE table_name (add_column_clause | drop_column_clause)
```

""：当要显示的操作里含有与EBNF重叠的关键字，或者多个操作要显示在一个框内时，用双引号进行包围

示例：在添加列字段的具体语法中，ADD关键字后需要跟"("，此时没有双引号的话会按照EBNF的含义来解析，用双引号则解析成其字面含义。

```
alter_table::= ALTER TABLE table_name (ADD "(" column_name dataType ")" | drop_column_clause)
```

[]：[]内的操作是可选的，可以略过

示例：在添加列字段的具体语法中，为列字段定义缺省值是一个可选项。

```
alter_table::= ALTER TABLE table_name (ADD "(" column_name dataType [default_expr] ")" | drop_column_clause)
```

{}：循环{}内定义的操作

示例：更改表时可以同时添加多个列字段，语法中用','分隔，且每个列字段的定义语法完全相同，此时可以定义成相同操作的循环。

```
alter_table::= ALTER TABLE table_name (ADD "(" column_name dataType [default_expr] { "," column_name dataType [default_expr]}
")" | drop_column_clause)
```

附：样例表

单机HEAP样例表

```
DROP TABLE IF EXISTS finance_info;
DROP TABLE IF EXISTS sales_info;
DROP TABLE IF EXISTS sales_info_range;
DROP TABLE IF EXISTS sales_info_list;
DROP TABLE IF EXISTS sales_info_hash;
DROP TABLE IF EXISTS orders_info;
DROP TABLE IF EXISTS product;
DROP TABLE IF EXISTS employees;
DROP TABLE IF EXISTS department;
DROP TABLE IF EXISTS branches;
DROP TABLE IF EXISTS area;
DROP TABLE IF EXISTS numbers;
DROP TABLE IF EXISTS numbers_nobit;
DROP TABLE IF EXISTS area1;
DROP TABLE IF EXISTS branches1;

--区域信息表
CREATE TABLE area
(area_no CHAR(2) NOT NULL PRIMARY KEY,
 area_name VARCHAR2(60),
 DHQ VARCHAR2(20) DEFAULT 'ShenZhen' NOT NULL);
INSERT INTO area VALUES ('01','华东','Shanghai');
INSERT INTO area VALUES ('02','华西','Chengdu');
INSERT INTO area VALUES ('03','华南','Guangzhou');
INSERT INTO area VALUES ('04','华北','Beijing');
INSERT INTO area VALUES ('05','华中','Wuhan');

--机构信息表
CREATE TABLE branches
(branch_no CHAR(4) PRIMARY KEY,
 branch_name VARCHAR2(200) NOT NULL,
 area_no CHAR(2)  CONSTRAINT c_branches_1 REFERENCES area(area_no) ON DELETE SET NULL,
 address VARCHAR2(200));
INSERT INTO branches VALUES ('0001','深圳','','');
INSERT INTO branches VALUES ('0101','上海','01','上海市静安区');
INSERT INTO branches VALUES ('0102','南京','01','City of Nanjing');
INSERT INTO branches VALUES ('0103','福州','01','');
INSERT INTO branches VALUES ('0104','厦门','01','Xiamen');
INSERT INTO branches VALUES ('0401','北京','04','');
INSERT INTO branches VALUES ('0402','天津','04','');
INSERT INTO branches VALUES ('0403','大连','04','大连市');
INSERT INTO branches VALUES ('0404','沈阳','04','');
INSERT INTO branches VALUES ('0201','成都','02','');
INSERT INTO branches VALUES ('0501','武汉','','');
INSERT INTO branches VALUES ('0502','长沙','05','');

--部门信息表
CREATE TABLE department
(deparment_no CHAR(3) PRIMARY KEY,
 department_name VARCHAR2(20));
INSERT INTO department VALUES('000','不分部门');
INSERT INTO department VALUES('010','销售部');
INSERT INTO department VALUES('008','采购部');
INSERT INTO department VALUES('002','财务部');

--员工信息表
CREATE TABLE employees
(branch CHAR(4)   CONSTRAINT c_employees_1 REFERENCES branches(branch_no) ON DELETE SET NULL,
 department CHAR(3)   CONSTRAINT c_employees_2 REFERENCES department(deparment_no) ON DELETE SET NULL,
 employee_no CHAR(10) NOT NULL PRIMARY KEY,
 employee_name VARCHAR2(10),
 sex CHAR(1),
 entry_date DATE
 );
```

```
INSERT INTO employees VALUES ('0101','000','0101000001','Mask','1',SYSDATE-1000);
INSERT INTO employees VALUES ('0101','000','0101000002','John','1',SYSDATE-2000);
INSERT INTO employees VALUES ('0201','010','0201010011','Anna','0',SYSDATE-300);
INSERT INTO employees VALUES ('0201','008','0201008003','Jack','1',SYSDATE-700);
INSERT INTO employees VALUES ('0101','008','0201008004','Jim','1',SYSDATE-200);

--产品信息表
CREATE TABLE product
(product_no CHAR(5) PRIMARY KEY,
 product_name VARCHAR2(30),
 cost NUMBER,
 price NUMBER
);
INSERT INTO product VALUES ('11001','product001',8,10);
INSERT INTO product VALUES ('11002','product002',13,16);
INSERT INTO product VALUES ('10001','product001',99,100);
INSERT INTO product VALUES ('10002','product002',199,200);

--订单信息表
CREATE TABLE orders_info (order_no CHAR(14) NOT NULL,
             product_no CHAR(5) CONSTRAINT c_orders_info_3 REFERENCES product(product_no) ON DELETE SET NULL,
             area CHAR(2) CONSTRAINT c_orders_info_2 REFERENCES area(area_no) ON DELETE SET NULL,
             branch CHAR(4) CONSTRAINT c_orders_info_1 REFERENCES branches(branch_no) ON DELETE SET NULL,
             order_date DATE DEFAULT SYSDATE NOT NULL,
             salesperson CHAR(10) CONSTRAINT c_orders_info_4 REFERENCES employees(employee_no) ON DELETE SET NULL,
             id NUMBER)
    PARTITION BY RANGE (id )
    INTERVAL (1000)
    (PARTITION p_orders_info_1 VALUES LESS THAN (800));
INSERT INTO orders_info VALUES ('20010102020001','11001','01','0101',SYSDATE-400,'0201010011',300);
INSERT INTO orders_info VALUES ('20010102020001','11002','02','0201',SYSDATE-400,'0201008003',1300);
INSERT INTO orders_info VALUES ('20010102020001','10001','04','0402',SYSDATE-400,'0201010011',2300);
INSERT INTO orders_info VALUES ('20210102020002','11001','04','0401',SYSDATE-400,'0201008003',400);
INSERT INTO orders_info VALUES ('20210102020002','11002','05','0501',SYSDATE-400,'0201010011',200);
INSERT INTO orders_info VALUES ('20210102020002','10001','01','0102',SYSDATE-400,'0201008003',100);

--销售信息表
CREATE TABLE sales_info
(year CHAR(4) NOT NULL,
 month CHAR(2) NOT NULL,
 branch CHAR(4) CONSTRAINT c_sales_info_1 REFERENCES branches(branch_no) ON DELETE SET NULL,
 product CHAR(5) CONSTRAINT c_sales_info_2 REFERENCES product(product_no) ON DELETE SET NULL,
 quantity NUMBER DEFAULT 0 NOT NULL,
 amount NUMBER(10,2) DEFAULT 0 NOT NULL,
 salsperson CHAR(10) CONSTRAINT c_sales_info_3 REFERENCES employees(employee_no) ON DELETE SET NULL)
PARTITION BY HASH(branch)
SUBPARTITION BY LIST(year)
SUBPARTITION template (SUBPARTITION sp_sales_info_1 VALUES ('2001','2002','2010'),
 SUBPARTITION sp_sales_info_2 VALUES ('2021','2020','2019'),
 SUBPARTITION sp_sales_info_3 VALUES (DEFAULT))
(PARTITION p_sales_info_1,PARTITION p_sales_info_2,PARTITION p_sales_info_3);

CREATE INDEX idx_sales_info_1 ON sales_info (year,month,branch,product) LOCAL
(PARTITION ip_sales_info_1 (SUBPARTITION isp_sales_info_11,SUBPARTITION isp_sales_info_12,SUBPARTITION isp_sales_info_13),
PARTITION ip_sales_info_2 (SUBPARTITION isp_sales_info_21,SUBPARTITION isp_sales_info_22,SUBPARTITION isp_sales_info_23),
PARTITION ip_sales_info_3 (SUBPARTITION isp_sales_info_31,SUBPARTITION isp_sales_info_32,SUBPARTITION isp_sales_info_33));

INSERT INTO sales_info VALUES ('2001','01','0201','11001',30,500,'0201010011');
INSERT INTO sales_info VALUES ('2015','11','0101','11001',20,300,'');
INSERT INTO sales_info VALUES ('2021','10','0402','11001',20,300,'');
INSERT INTO sales_info VALUES ('2000','12','0102','11001',20,300,'');
INSERT INTO sales_info VALUES ('2015','03','0102','11001',20,300,'');
INSERT INTO sales_info VALUES ('2021','05','0101','11001',40,600,'');

--范围分区的销售信息表
CREATE TABLE sales_info_range
(year CHAR(4) NOT NULL,
 month CHAR(2) NOT NULL,
 branch CHAR(4),
 product CHAR(5),
```

```
 quantity NUMBER DEFAULT 0 NOT NULL,
 amount NUMBER(10,2) DEFAULT 0 NOT NULL,
 salsperson CHAR(10))
PARTITION BY RANGE(year)
(PARTITION p_sales_info_range_1 VALUES LESS THAN('2011'),
 PARTITION p_sales_info_range_2 VALUES LESS THAN('2021'),
 PARTITION p_sales_info_range_3 VALUES LESS THAN('2031'));
INSERT INTO sales_info_range VALUES ('2001','01','0201','11001',30,500,'0201010011');
INSERT INTO sales_info_range VALUES ('2015','11','0101','11001',20,300,'');
INSERT INTO sales_info_range VALUES ('2021','10','0101','11001',20,300,'');
INSERT INTO sales_info_range VALUES ('2000','12','0102','11001',20,300,'');
INSERT INTO sales_info_range VALUES ('2015','03','0102','11001',20,300,'');
INSERT INTO sales_info_range VALUES ('2021','05','0101','11001',40,600,'');

--列表分区的销售信息表
CREATE TABLE sales_info_list
(year CHAR(4) NOT NULL,
 month CHAR(2) NOT NULL,
 branch CHAR(4),
 product CHAR(5),
 quantity NUMBER DEFAULT 0 NOT NULL,
 amount NUMBER(10,2) DEFAULT 0 NOT NULL,
 salsperson CHAR(10))
PARTITION BY LIST(year)
(PARTITION p_sales_info_list_1 VALUES('2018','2019'),
 PARTITION p_sales_info_list_2 VALUES('2021'));
INSERT INTO sales_info_list VALUES ('2018','10','0101','11001',20,300,'');
INSERT INTO sales_info_list VALUES ('2021','05','0101','11001',40,600,'');

--哈希分区的销售信息表
CREATE TABLE sales_info_hash
(year CHAR(4) NOT NULL,
 month CHAR(2) NOT NULL,
 branch CHAR(4),
 product CHAR(5),
 quantity NUMBER DEFAULT 0 NOT NULL,
 amount NUMBER(10,2) DEFAULT 0 NOT NULL,
 salsperson CHAR(10))
PARTITION BY HASH(year)
(PARTITION p_sales_info_hash_1,
 PARTITION p_sales_info_hash_2);

--财务信息表
CREATE TABLE finance_info
(year CHAR(4) NOT NULL,
 month CHAR(2) NOT NULL,
 branch CHAR(4) CONSTRAINT c_finance_info_1 REFERENCES branches(branch_no) ON DELETE SET NULL,
 revenue_total NUMBER(10,2),
 cost_total NUMBER(10,2),
 fee_total NUMBER(10,2)
);
CREATE INDEX idx_finance_info_1 ON finance_info (year,month,branch);
INSERT INTO finance_info VALUES ('2001','01','0201',2888,2000,300);
INSERT INTO finance_info VALUES ('2021','01','0201',28888,24000,3000);
INSERT INTO finance_info VALUES ('2021','01','0101',38888,34000,4000);
INSERT INTO finance_info VALUES ('2021','02','0101',37778,33000,6000);

--在数学函数描述中经常使用的表numbers,包含TINYINT、SMALLINT、INT、BIGINT、FLOAT、DOUBLE、NUMBER、BIT字段
CREATE TABLE numbers(numbera INT,numberb TINYINT,numberc SMALLINT,numberd BIGINT,numbere FLOAT,numberf DOUBLE,numberg
NUMBER,numberh BIT);
INSERT INTO numbers VALUES(-5,55,5555,5555555555555555555,5.555,-5.55555555,555,1);

--在数学函数描述中经常使用的表numbers_nobit,包含INT、TINYINT、SMALLINT、BIGINT、FLOAT、DOUBLE、NUMBER字段
CREATE TABLE numbers_nobit(numbera INT,numberb TINYINT,numberc SMALLINT,numberd BIGINT,numbere FLOAT,numberf DOUBLE,numberg
NUMBER);
INSERT INTO numbers_nobit VALUES(-5,55,5555,5555555555555555555,5.555,5.55555555555,555);

--在聚集函数描述中经常使用的表area1
CREATE TABLE area1
(area_no CHAR(2) NOT NULL PRIMARY KEY,
```

```
 area_name VARCHAR2(60),
 DHQ VARCHAR2(20) DEFAULT 'ShenZhen' NOT NULL,
 employee_count INT);
INSERT INTO area1 VALUES ('01','华东','Shanghai','');
INSERT INTO area1 VALUES ('02','华西','Chengdu',300);
INSERT INTO area1 VALUES ('03','华南','Guangzhou',400);
INSERT INTO area1 VALUES ('04','华北','Beijing',300);
INSERT INTO area1 VALUES ('05','华中','Wuhan','');

--在聚集函数描述中经常使用的表branches1
CREATE TABLE branches1
(branch_no CHAR(4) PRIMARY KEY,
 branch_name VARCHAR2(200) NOT NULL,
 area_no CHAR(2),
 address VARCHAR2(200),
 employee_count INT);
INSERT INTO branches1 VALUES ('0001','深圳','','',40);
INSERT INTO branches1 VALUES ('0101','上海','01','上海市静安区','');
INSERT INTO branches1 VALUES ('0102','南京','01','City of Nanjing',70);
INSERT INTO branches1 VALUES ('0103','福州','01','','');
INSERT INTO branches1 VALUES ('0104','厦门','01','Xiamen','');
INSERT INTO branches1 VALUES ('0401','北京','04','','');
INSERT INTO branches1 VALUES ('0402','天津','04','',10);
INSERT INTO branches1 VALUES ('0403','大连','04','大连市',30);
INSERT INTO branches1 VALUES ('0404','沈阳','04','','');
INSERT INTO branches1 VALUES ('0201','成都','02','','');
INSERT INTO branches1 VALUES ('0501','武汉','','',30);
INSERT INTO branches1 VALUES ('0502','长沙','05','',40);

COMMIT;
```

单机TAC样例表

```
DROP TABLE IF EXISTS finance_info;
DROP TABLE IF EXISTS sales_info;
DROP TABLE IF EXISTS sales_info_range;
DROP TABLE IF EXISTS sales_info_list;
DROP TABLE IF EXISTS sales_info_hash;
DROP TABLE IF EXISTS orders_info;
DROP TABLE IF EXISTS product;
DROP TABLE IF EXISTS employees;
DROP TABLE IF EXISTS department;
DROP TABLE IF EXISTS branches;
DROP TABLE IF EXISTS area;
DROP TABLE IF EXISTS numbers;
DROP TABLE IF EXISTS numbers_nobit;
DROP TABLE IF EXISTS area1;
DROP TABLE IF EXISTS branches1;

--区域信息表
CREATE TABLE area
(area_no CHAR(2) NOT NULL PRIMARY KEY,
 area_name VARCHAR2(60),
 DHQ VARCHAR2(20) DEFAULT 'ShenZhen' NOT NULL);
INSERT INTO area VALUES ('01','华东','Shanghai');
INSERT INTO area VALUES ('02','华西','Chengdu');
INSERT INTO area VALUES ('03','华南','Guangzhou');
INSERT INTO area VALUES ('04','华北','Beijing');
INSERT INTO area VALUES ('05','华中','Wuhan');

--机构信息表
CREATE TABLE branches
(branch_no CHAR(4) PRIMARY KEY,
 branch_name VARCHAR2(200) NOT NULL,
 area_no CHAR(2),
 address VARCHAR2(200));
INSERT INTO branches VALUES ('0001','深圳','','');
INSERT INTO branches VALUES ('0101','上海','01','上海市静安区');
```

```
INSERT INTO branches VALUES ('0102','南京','01','City of Nanjing');
INSERT INTO branches VALUES ('0103','福州','01','');
INSERT INTO branches VALUES ('0104','厦门','01','Xiamen');
INSERT INTO branches VALUES ('0401','北京','04','');
INSERT INTO branches VALUES ('0402','天津','04','');
INSERT INTO branches VALUES ('0403','大连','04','大连市');
INSERT INTO branches VALUES ('0404','沈阳','04','');
INSERT INTO branches VALUES ('0201','成都','02','');
INSERT INTO branches VALUES ('0501','武汉','','');
INSERT INTO branches VALUES ('0502','长沙','05','');

--部门信息表
CREATE TABLE department
(deparment_no CHAR(3) PRIMARY KEY,
 department_name VARCHAR2(20));
INSERT INTO department VALUES('000','不分部门');
INSERT INTO department VALUES('010','销售部');
INSERT INTO department VALUES('008','采购部');
INSERT INTO department VALUES('002','财务部');

--员工信息表
CREATE TABLE employees
(branch CHAR(4),
 department CHAR(3),
 employee_no CHAR(10) NOT NULL PRIMARY KEY,
 employee_name VARCHAR2(10),
 sex CHAR(1),
 entry_date DATE
 );
INSERT INTO employees VALUES ('0101','000','0101000001','Mask','1',SYSDATE-1000);
INSERT INTO employees VALUES ('0101','000','0101000002','John','1',SYSDATE-2000);
INSERT INTO employees VALUES ('0201','010','0201010011','Anna','0',SYSDATE-300);
INSERT INTO employees VALUES ('0201','008','0201008003','Jack','1',SYSDATE-700);
INSERT INTO employees VALUES ('0101','008','0201008004','Jim','1',SYSDATE-200);

--产品信息表
CREATE TABLE product
(product_no CHAR(5) PRIMARY KEY,
 product_name VARCHAR2(30),
 cost NUMBER,
 price NUMBER
);
INSERT INTO product VALUES ('11001','product001',8,10);
INSERT INTO product VALUES ('11002','product002',13,16);
INSERT INTO product VALUES ('10001','product001',99,100);
INSERT INTO product VALUES ('10002','product002',199,200);

--订单信息表
CREATE TABLE orders_info (order_no CHAR(14) NOT NULL,
             product_no CHAR(5),
             area CHAR(2),
             branch CHAR(4),
             order_date DATE DEFAULT SYSDATE NOT NULL,
             salesperson CHAR(10),
             id NUMBER)
    PARTITION BY RANGE (id )
    INTERVAL (1000)
    (PARTITION p_orders_info_1 VALUES LESS THAN (800));
INSERT INTO orders_info VALUES ('20010102020001','11001','01','0101',SYSDATE-400,'0201010011',300);
INSERT INTO orders_info VALUES ('20010102020001','11002','02','0201',SYSDATE-400,'0201008003',1300);
INSERT INTO orders_info VALUES ('20010102020001','10001','04','0402',SYSDATE-400,'0201010011',2300);
INSERT INTO orders_info VALUES ('20210102020002','11001','04','0401',SYSDATE-400,'0201008003',400);
INSERT INTO orders_info VALUES ('20210102020002','11002','05','0501',SYSDATE-400,'0201010011',200);
INSERT INTO orders_info VALUES ('20210102020002','10001','01','0102',SYSDATE-400,'0201008003',100);

--销售信息表
CREATE TABLE sales_info
(year CHAR(4) NOT NULL,
 month CHAR(2) NOT NULL,
 branch CHAR(4),
```

```
 product CHAR(5),
 quantity NUMBER DEFAULT 0 NOT NULL,
 amount NUMBER(10,2) DEFAULT 0 NOT NULL,
 salsperson CHAR(10))
PARTITION BY HASH(branch)
SUBPARTITION BY LIST(year)
SUBPARTITION template (SUBPARTITION sp_sales_info_1 VALUES ('2001','2002','2010'),
 SUBPARTITION sp_sales_info_2 VALUES ('2021','2020','2019'),
 SUBPARTITION sp_sales_info_3 VALUES (DEFAULT))
(PARTITION p_sales_info_1,PARTITION p_sales_info_2,PARTITION p_sales_info_3);

CREATE INDEX idx_sales_info_1 ON sales_info (year,month,branch,product) LOCAL
(PARTITION ip_sales_info_1 (SUBPARTITION isp_sales_info_11,SUBPARTITION isp_sales_info_12,SUBPARTITION isp_sales_info_13),
PARTITION ip_sales_info_2 (SUBPARTITION isp_sales_info_21,SUBPARTITION isp_sales_info_22,SUBPARTITION isp_sales_info_23),
PARTITION ip_sales_info_3 (SUBPARTITION isp_sales_info_31,SUBPARTITION isp_sales_info_32,SUBPARTITION isp_sales_info_33));

INSERT INTO sales_info VALUES ('2001','01','0201','11001',30,500,'0201010011');
INSERT INTO sales_info VALUES ('2015','11','0101','11001',20,300,'');
INSERT INTO sales_info VALUES ('2021','10','0402','11001',20,300,'');
INSERT INTO sales_info VALUES ('2000','12','0102','11001',20,300,'');
INSERT INTO sales_info VALUES ('2015','03','0102','11001',20,300,'');
INSERT INTO sales_info VALUES ('2021','05','0101','11001',40,600,'');

--范围分区的销售信息表
CREATE TABLE sales_info_range
(year CHAR(4) NOT NULL,
 month CHAR(2) NOT NULL,
 branch CHAR(4),
 product CHAR(5),
 quantity NUMBER DEFAULT 0 NOT NULL,
 amount NUMBER(10,2) DEFAULT 0 NOT NULL,
 salsperson CHAR(10))
PARTITION BY RANGE(year)
(PARTITION p_sales_info_range_1 VALUES LESS THAN('2011'),
 PARTITION p_sales_info_range_2 VALUES LESS THAN('2021'),
 PARTITION p_sales_info_range_3 VALUES LESS THAN('2031'));
INSERT INTO sales_info_range VALUES ('2001','01','0201','11001',30,500,'0201010011');
INSERT INTO sales_info_range VALUES ('2015','11','0101','11001',20,300,'');
INSERT INTO sales_info_range VALUES ('2021','10','0101','11001',20,300,'');
INSERT INTO sales_info_range VALUES ('2000','12','0102','11001',20,300,'');
INSERT INTO sales_info_range VALUES ('2015','03','0102','11001',20,300,'');
INSERT INTO sales_info_range VALUES ('2021','05','0101','11001',40,600,'');

--列表分区的销售信息表
CREATE TABLE sales_info_list
(year CHAR(4) NOT NULL,
 month CHAR(2) NOT NULL,
 branch CHAR(4),
 product CHAR(5),
 quantity NUMBER DEFAULT 0 NOT NULL,
 amount NUMBER(10,2) DEFAULT 0 NOT NULL,
 salsperson CHAR(10))
PARTITION BY LIST(year)
(PARTITION p_sales_info_list_1 VALUES('2018','2019'),
 PARTITION p_sales_info_list_2 VALUES('2021'));
INSERT INTO sales_info_list VALUES ('2018','10','0101','11001',20,300,'');
INSERT INTO sales_info_list VALUES ('2021','05','0101','11001',40,600,'');

--哈希分区的销售信息表
CREATE TABLE sales_info_hash
(year CHAR(4) NOT NULL,
 month CHAR(2) NOT NULL,
 branch CHAR(4),
 product CHAR(5),
 quantity NUMBER DEFAULT 0 NOT NULL,
 amount NUMBER(10,2) DEFAULT 0 NOT NULL,
 salsperson CHAR(10))
PARTITION BY HASH(year)
(PARTITION p_sales_info_hash_1,
 PARTITION p_sales_info_hash_2);
```

```
--财务信息表
CREATE TABLE finance_info
(year CHAR(4) NOT NULL,
 month CHAR(2) NOT NULL,
 branch CHAR(4),
 revenue_total NUMBER(10,2),
 cost_total NUMBER(10,2),
 fee_total NUMBER(10,2)
);
CREATE INDEX idx_finance_info_1 ON finance_info (year,month,branch);
INSERT INTO finance_info VALUES ('2001','01','0201',2888,2000,300);
INSERT INTO finance_info VALUES ('2021','01','0201',28888,24000,3000);
INSERT INTO finance_info VALUES ('2021','01','0101',38888,34000,4000);
INSERT INTO finance_info VALUES ('2021','02','0101',37778,33000,6000);

--在数学函数描述中经常使用的表numbers,包含TINYINT、SMALLINT、INT、BIGINT、FLOAT、DOUBLE、NUMBER字段
CREATE TABLE numbers(numbera INT,numberb TINYINT,numberc SMALLINT,numberd BIGINT,numbere FLOAT,numberf DOUBLE,numberg
NUMBER);
INSERT INTO numbers VALUES(-5,55,5555,5555555555555555555,5.555,-5.55555555,555);

--在数学函数描述中经常使用的表numbers_nobit,包含INT、TINYINT、SMALLINT、BIGINT、FLOAT、DOUBLE、NUMBER字段
CREATE TABLE numbers_nobit(numbera INT,numberb TINYINT,numberc SMALLINT,numberd BIGINT,numbere FLOAT,numberf DOUBLE,numberg
NUMBER);
INSERT INTO numbers_nobit VALUES(-5,55,5555,5555555555555555555,5.555,5.55555555555,555);

--在聚集函数描述中经常使用的表area1
CREATE TABLE area1
(area_no CHAR(2) NOT NULL PRIMARY KEY,
 area_name VARCHAR2(60),
 DHQ VARCHAR2(20) DEFAULT 'ShenZhen' NOT NULL,
 employee_count INT);
INSERT INTO area1 VALUES ('01','华东','Shanghai','');
INSERT INTO area1 VALUES ('02','华西','Chengdu',300);
INSERT INTO area1 VALUES ('03','华南','Guangzhou',400);
INSERT INTO area1 VALUES ('04','华北','Beijing',300);
INSERT INTO area1 VALUES ('05','华中','Wuhan','');

--在聚集函数描述中经常使用的表branches1
CREATE TABLE branches1
(branch_no CHAR(4) PRIMARY KEY,
 branch_name VARCHAR2(200) NOT NULL,
 area_no CHAR(2),
 address VARCHAR2(200),
 employee_count INT);
INSERT INTO branches1 VALUES ('0001','深圳','','',40);
INSERT INTO branches1 VALUES ('0101','上海','01','上海市静安区','');
INSERT INTO branches1 VALUES ('0102','南京','01','City of Nanjing',70);
INSERT INTO branches1 VALUES ('0103','福州','01','','');
INSERT INTO branches1 VALUES ('0104','厦门','01','Xiamen','');
INSERT INTO branches1 VALUES ('0401','北京','04','','');
INSERT INTO branches1 VALUES ('0402','天津','04','',10);
INSERT INTO branches1 VALUES ('0403','大连','04','大连市',30);
INSERT INTO branches1 VALUES ('0404','沈阳','04','','');
INSERT INTO branches1 VALUES ('0201','成都','02','','');
INSERT INTO branches1 VALUES ('0501','武汉','','',30);
INSERT INTO branches1 VALUES ('0502','长沙','05','',40);

COMMIT;
```

单机LSC样例表

```
DROP TABLE IF EXISTS finance_info;
DROP TABLE IF EXISTS sales_info;
DROP TABLE IF EXISTS sales_info_range;
DROP TABLE IF EXISTS sales_info_list;
DROP TABLE IF EXISTS sales_info_hash;
DROP TABLE IF EXISTS orders_info;
```

```
DROP TABLE IF EXISTS product;
DROP TABLE IF EXISTS employees;
DROP TABLE IF EXISTS department;
DROP TABLE IF EXISTS branches;
DROP TABLE IF EXISTS area;
DROP TABLE IF EXISTS numbers;
DROP TABLE IF EXISTS numbers_nobit;
DROP TABLE IF EXISTS area1;
DROP TABLE IF EXISTS branches1;

--区域信息表
CREATE TABLE area
(area_no CHAR(2) NOT NULL PRIMARY KEY,
 area_name VARCHAR2(60),
 DHQ VARCHAR2(20) DEFAULT 'ShenZhen' NOT NULL);
INSERT INTO area VALUES ('01','华东','Shanghai');
INSERT INTO area VALUES ('02','华西','Chengdu');
INSERT INTO area VALUES ('03','华南','Guangzhou');
INSERT INTO area VALUES ('04','华北','Beijing');
INSERT INTO area VALUES ('05','华中','Wuhan');

--机构信息表
CREATE TABLE branches
(branch_no CHAR(4),
 branch_name VARCHAR2(200) NOT NULL,
 area_no CHAR(2),
 address VARCHAR2(200));
INSERT INTO branches VALUES ('0001','深圳','','');
INSERT INTO branches VALUES ('0101','上海','01','上海市静安区');
INSERT INTO branches VALUES ('0102','南京','01','City of Nanjing');
INSERT INTO branches VALUES ('0103','福州','01','');
INSERT INTO branches VALUES ('0104','厦门','01','Xiamen');
INSERT INTO branches VALUES ('0401','北京','04','');
INSERT INTO branches VALUES ('0402','天津','04','');
INSERT INTO branches VALUES ('0403','大连','04','大连市');
INSERT INTO branches VALUES ('0404','沈阳','04','');
INSERT INTO branches VALUES ('0201','成都','02','');
INSERT INTO branches VALUES ('0501','武汉','','');
INSERT INTO branches VALUES ('0502','长沙','05','');

--部门信息表
CREATE TABLE department
(deparment_no CHAR(3),
 department_name VARCHAR2(20));
INSERT INTO department VALUES('000','不分部门');
INSERT INTO department VALUES('010','销售部');
INSERT INTO department VALUES('008','采购部');
INSERT INTO department VALUES('002','财务部');

--员工信息表
CREATE TABLE employees
(branch CHAR(4),
 department CHAR(3),
 employee_no CHAR(10) NOT NULL,
 employee_name VARCHAR2(10),
 sex CHAR(1),
 entry_date DATE
 );
INSERT INTO employees VALUES ('0101','000','0101000001','Mask','1',SYSDATE-1000);
INSERT INTO employees VALUES ('0101','000','0101000002','John','1',SYSDATE-2000);
INSERT INTO employees VALUES ('0201','010','0201010011','Anna','0',SYSDATE-300);
INSERT INTO employees VALUES ('0201','008','0201008003','Jack','1',SYSDATE-700);
INSERT INTO employees VALUES ('0101','008','0201008004','Jim','1',SYSDATE-200);

--产品信息表
CREATE TABLE product
(product_no CHAR(5),
 product_name VARCHAR2(30),
 cost NUMBER,
 price NUMBER
```

```
);
INSERT INTO product VALUES ('11001','product001',8,10);
INSERT INTO product VALUES ('11002','product002',13,16);
INSERT INTO product VALUES ('10001','product001',99,100);
INSERT INTO product VALUES ('10002','product002',199,200);

--订单信息表
CREATE TABLE orders_info (order_no CHAR(14) NOT NULL,
            product_no CHAR(5),
            area CHAR(2),
            branch CHAR(4),
            order_date DATE DEFAULT SYSDATE NOT NULL,
            salesperson CHAR(10),
            id NUMBER)
    PARTITION BY RANGE (id )
    INTERVAL (1000)
    (PARTITION p_orders_info_1 VALUES LESS THAN (800));
INSERT INTO orders_info VALUES ('20010102020001','11001','01','0101',SYSDATE-400,'0201010011',300);
INSERT INTO orders_info VALUES ('20010102020001','11002','02','0201',SYSDATE-400,'0201008003',1300);
INSERT INTO orders_info VALUES ('20010102020001','10001','04','0402',SYSDATE-400,'0201010011',2300);
INSERT INTO orders_info VALUES ('20210102020002','11001','04','0401',SYSDATE-400,'0201008003',400);
INSERT INTO orders_info VALUES ('20210102020002','11002','05','0501',SYSDATE-400,'0201010011',200);
INSERT INTO orders_info VALUES ('20210102020002','10001','01','0102',SYSDATE-400,'0201008003',100);

--销售信息表
CREATE TABLE sales_info
(year CHAR(4) NOT NULL,
 month CHAR(2) NOT NULL,
 branch CHAR(4),
 product CHAR(5),
 quantity NUMBER DEFAULT 0 NOT NULL,
 amount NUMBER(10,2) DEFAULT 0 NOT NULL,
 salsperson CHAR(10))
PARTITION BY HASH(branch)
SUBPARTITION BY LIST(year)
SUBPARTITION template (SUBPARTITION sp_sales_info_1 VALUES ('2001','2002','2010'),
 SUBPARTITION sp_sales_info_2 VALUES ('2021','2020','2019'),
 SUBPARTITION sp_sales_info_3 VALUES (DEFAULT))
(PARTITION p_sales_info_1,PARTITION p_sales_info_2,PARTITION p_sales_info_3);
INSERT INTO sales_info VALUES ('2001','01','0201','11001',30,500,'0201010011');
INSERT INTO sales_info VALUES ('2015','11','0101','11001',20,300,'');
INSERT INTO sales_info VALUES ('2021','10','0402','11001',20,300,'');
INSERT INTO sales_info VALUES ('2000','12','0102','11001',20,300,'');
INSERT INTO sales_info VALUES ('2015','03','0102','11001',20,300,'');
INSERT INTO sales_info VALUES ('2021','05','0101','11001',40,600,'');

--范围分区的销售信息表
CREATE TABLE sales_info_range
(year CHAR(4) NOT NULL,
 month CHAR(2) NOT NULL,
 branch CHAR(4),
 product CHAR(5),
 quantity NUMBER DEFAULT 0 NOT NULL,
 amount NUMBER(10,2) DEFAULT 0 NOT NULL,
 salsperson CHAR(10))
PARTITION BY RANGE(year)
(PARTITION p_sales_info_range_1 VALUES LESS THAN('2011'),
 PARTITION p_sales_info_range_2 VALUES LESS THAN('2021'),
 PARTITION p_sales_info_range_3 VALUES LESS THAN('2031'));
INSERT INTO sales_info_range VALUES ('2001','01','0201','11001',30,500,'0201010011');
INSERT INTO sales_info_range VALUES ('2015','11','0101','11001',20,300,'');
INSERT INTO sales_info_range VALUES ('2021','10','0101','11001',20,300,'');
INSERT INTO sales_info_range VALUES ('2000','12','0102','11001',20,300,'');
INSERT INTO sales_info_range VALUES ('2015','03','0102','11001',20,300,'');
INSERT INTO sales_info_range VALUES ('2021','05','0101','11001',40,600,'');

--列表分区的销售信息表
CREATE TABLE sales_info_list
(year CHAR(4) NOT NULL,
 month CHAR(2) NOT NULL,
```

```
 branch CHAR(4),
 product CHAR(5),
 quantity NUMBER DEFAULT 0 NOT NULL,
 amount NUMBER(10,2) DEFAULT 0 NOT NULL,
 salsperson CHAR(10))
PARTITION BY LIST(year)
(PARTITION p_sales_info_list_1 VALUES('2018','2019'),
 PARTITION p_sales_info_list_2 VALUES('2021'));
INSERT INTO sales_info_list VALUES ('2018','10','0101','11001',20,300,'');
INSERT INTO sales_info_list VALUES ('2021','05','0101','11001',40,600,'');

--哈希分区的销售信息表
CREATE TABLE sales_info_hash
(year CHAR(4) NOT NULL,
 month CHAR(2) NOT NULL,
 branch CHAR(4),
 product CHAR(5),
 quantity NUMBER DEFAULT 0 NOT NULL,
 amount NUMBER(10,2) DEFAULT 0 NOT NULL,
 salsperson CHAR(10))
PARTITION BY HASH(year)
(PARTITION p_sales_info_hash_1,
 PARTITION p_sales_info_hash_2);

--财务信息表
CREATE TABLE finance_info
(year CHAR(4) NOT NULL,
 month CHAR(2) NOT NULL,
 branch CHAR(4),
 revenue_total NUMBER(10,2),
 cost_total NUMBER(10,2),
 fee_total NUMBER(10,2)
);
INSERT INTO finance_info VALUES ('2001','01','0201',2888,2000,300);
INSERT INTO finance_info VALUES ('2021','01','0201',28888,24000,3000);
INSERT INTO finance_info VALUES ('2021','01','0101',38888,34000,4000);
INSERT INTO finance_info VALUES ('2021','02','0101',37778,33000,6000);

--在数学函数描述中经常使用的表numbers,包含TINYINT、SMALLINT、INT、BIGINT、FLOAT、DOUBLE、NUMBER字段
CREATE TABLE numbers(numbera INT,numberb TINYINT,numberc SMALLINT,numberd BIGINT,numbere FLOAT,numberf DOUBLE,numberg
NUMBER);
INSERT INTO numbers VALUES(-5,55,5555,5555555555555555555,5.555,-5.55555555,555);

--在数学函数描述中经常使用的表numbers_nobit,包含INT、TINYINT、SMALLINT、BIGINT、FLOAT、DOUBLE、NUMBER字段
CREATE TABLE numbers_nobit(numbera INT,numberb TINYINT,numberc SMALLINT,numberd BIGINT,numbere FLOAT,numberf DOUBLE,numberg
NUMBER);
INSERT INTO numbers_nobit VALUES(-5,55,5555,5555555555555555555,5.555,5.55555555555,555);

--在聚集函数描述中经常使用的表area1
CREATE TABLE area1
(area_no CHAR(2) NOT NULL,
 area_name VARCHAR2(60),
 DHQ VARCHAR2(20) DEFAULT 'ShenZhen' NOT NULL,
 employee_count INT);
INSERT INTO area1 VALUES ('01','华东','Shanghai','');
INSERT INTO area1 VALUES ('02','华西','Chengdu',300);
INSERT INTO area1 VALUES ('03','华南','Guangzhou',400);
INSERT INTO area1 VALUES ('04','华北','Beijing',300);
INSERT INTO area1 VALUES ('05','华中','Wuhan','');

--在聚集函数描述中经常使用的表branches1
CREATE TABLE branches1
(branch_no CHAR(4),
 branch_name VARCHAR2(200) NOT NULL,
 area_no CHAR(2),
 address VARCHAR2(200),
 employee_count INT);
INSERT INTO branches1 VALUES ('0001','深圳','','',40);
INSERT INTO branches1 VALUES ('0101','上海','01','上海市静安区','');
INSERT INTO branches1 VALUES ('0102','南京','01','City of Nanjing',70);
```

```
INSERT INTO branches1 VALUES ('0103','福州','01','','');
INSERT INTO branches1 VALUES ('0104','厦门','01','Xiamen','');
INSERT INTO branches1 VALUES ('0401','北京','04','','');
INSERT INTO branches1 VALUES ('0402','天津','04','',10);
INSERT INTO branches1 VALUES ('0403','大连','04','大连市',30);
INSERT INTO branches1 VALUES ('0404','沈阳','04','','');
INSERT INTO branches1 VALUES ('0201','成都','02','','');
INSERT INTO branches1 VALUES ('0501','武汉','','',30);
INSERT INTO branches1 VALUES ('0502','长沙','05','',40);

COMMIT;
```

分布式TAC样例表

```
DROP TABLE IF EXISTS finance_info;
DROP TABLE IF EXISTS sales_info;
DROP TABLE IF EXISTS sales_info_range;
DROP TABLE IF EXISTS sales_info_list;
DROP TABLE IF EXISTS sales_info_hash;
DROP TABLE IF EXISTS orders_info;
DROP TABLE IF EXISTS product;
DROP TABLE IF EXISTS employees;
DROP TABLE IF EXISTS department;
DROP TABLE IF EXISTS branches;
DROP TABLE IF EXISTS area;
DROP TABLE IF EXISTS numbers;
DROP TABLE IF EXISTS numbers_nobit;
DROP TABLE IF EXISTS area1;
DROP TABLE IF EXISTS branches1;

--区域信息表
CREATE TABLE area
(area_no CHAR(2) NOT NULL PRIMARY KEY,
 area_name VARCHAR2(60),
 DHQ VARCHAR2(20) DEFAULT 'ShenZhen' NOT NULL);
INSERT INTO area VALUES ('01','华东','Shanghai');
INSERT INTO area VALUES ('02','华西','Chengdu');
INSERT INTO area VALUES ('03','华南','Guangzhou');
INSERT INTO area VALUES ('04','华北','Beijing');
INSERT INTO area VALUES ('05','华中','Wuhan');

--机构信息表
CREATE TABLE branches
(branch_no CHAR(4) PRIMARY KEY,
 branch_name VARCHAR2(200) NOT NULL,
 area_no CHAR(2),
 address VARCHAR2(200));
INSERT INTO branches VALUES ('0001','深圳','','');
INSERT INTO branches VALUES ('0101','上海','01','上海市静安区');
INSERT INTO branches VALUES ('0102','南京','01','City of Nanjing');
INSERT INTO branches VALUES ('0103','福州','01','');
INSERT INTO branches VALUES ('0104','厦门','01','Xiamen');
INSERT INTO branches VALUES ('0401','北京','04','');
INSERT INTO branches VALUES ('0402','天津','04','');
INSERT INTO branches VALUES ('0403','大连','04','大连市');
INSERT INTO branches VALUES ('0404','沈阳','04','');
INSERT INTO branches VALUES ('0201','成都','02','');
INSERT INTO branches VALUES ('0501','武汉','','');
INSERT INTO branches VALUES ('0502','长沙','05','');

--部门信息表
CREATE TABLE department
(deparment_no CHAR(3) PRIMARY KEY,
 department_name VARCHAR2(20));
INSERT INTO department VALUES('000','不分部门');
INSERT INTO department VALUES('010','销售部');
INSERT INTO department VALUES('008','采购部');
INSERT INTO department VALUES('002','财务部');
```

```
--员工信息表
CREATE TABLE employees
(branch CHAR(4),
 department CHAR(3),
 employee_no CHAR(10) NOT NULL PRIMARY KEY,
 employee_name VARCHAR2(10),
 sex CHAR(1),
 entry_date DATE
 );
INSERT INTO employees VALUES ('0101','000','0101000001','Mask','1',SYSDATE-1000);
INSERT INTO employees VALUES ('0101','000','0101000002','John','1',SYSDATE-2000);
INSERT INTO employees VALUES ('0201','010','0201010011','Anna','0',SYSDATE-300);
INSERT INTO employees VALUES ('0201','008','0201008003','Jack','1',SYSDATE-700);
INSERT INTO employees VALUES ('0101','008','0201008004','Jim','1',SYSDATE-200);

--产品信息表
CREATE TABLE product
(product_no CHAR(5) PRIMARY KEY,
 product_name VARCHAR2(30),
 cost NUMBER,
 price NUMBER
);
INSERT INTO product VALUES ('11001','product001',8,10);
INSERT INTO product VALUES ('11002','product002',13,16);
INSERT INTO product VALUES ('10001','product001',99,100);
INSERT INTO product VALUES ('10002','product002',199,200);

--订单信息表
CREATE TABLE orders_info (order_no CHAR(14) NOT NULL,
             product_no CHAR(5),
             area CHAR(2),
             branch CHAR(4),
             order_date DATE DEFAULT SYSDATE NOT NULL,
             salesperson CHAR(10),
             id NUMBER)
PARTITION BY HASH (id)
SUBPARTITION BY RANGE (area)
SUBPARTITION TEMPLATE
(SUBPARTITION sp_orders_info_1 VALUES LESS THAN (2),
 SUBPARTITION sp_orders_info_2 VALUES LESS THAN (4),
 SUBPARTITION sp_orders_info_3 VALUES LESS THAN (MAXVALUE)
)
PARTITIONS AUTO;
INSERT INTO orders_info VALUES ('20010102020001','11001','01','0101',SYSDATE-400,'0201010011',300);
INSERT INTO orders_info VALUES ('20010102020001','11002','02','0201',SYSDATE-400,'0201008003',1300);
INSERT INTO orders_info VALUES ('20010102020001','10001','04','0402',SYSDATE-400,'0201010011',2300);
INSERT INTO orders_info VALUES ('20210102020002','11001','04','0401',SYSDATE-400,'0201008003',400);
INSERT INTO orders_info VALUES ('20210102020002','11002','05','0501',SYSDATE-400,'0201010011',200);
INSERT INTO orders_info VALUES ('20210102020002','10001','01','0102',SYSDATE-400,'0201008003',100);

--销售信息表
CREATE TABLE sales_info
(year CHAR(4) NOT NULL,
 month CHAR(2) NOT NULL,
 branch CHAR(4),
 product CHAR(5),
 quantity NUMBER DEFAULT 0 NOT NULL,
 amount NUMBER(10,2) DEFAULT 0 NOT NULL,
 salsperson CHAR(10))
PARTITION BY HASH(branch)
SUBPARTITION BY LIST(year)
SUBPARTITION TEMPLATE
(SUBPARTITION sp_sales_info_1 VALUES ('2001','2002','2010'),
 SUBPARTITION sp_sales_info_2 VALUES ('2021','2020','2019'),
 SUBPARTITION sp_sales_info_3 VALUES (DEFAULT))
PARTITIONS AUTO;
CREATE INDEX idx_sales_info_1 ON sales_info (year,month,branch,product);
INSERT INTO sales_info VALUES ('2001','01','0201','11001',30,500,'0201010011');
INSERT INTO sales_info VALUES ('2015','11','0101','11001',20,300,'');
```

```
INSERT INTO sales_info VALUES ('2021','10','0402','11001',20,300,'');
INSERT INTO sales_info VALUES ('2000','12','0102','11001',20,300,'');
INSERT INTO sales_info VALUES ('2015','03','0102','11001',20,300,'');
INSERT INTO sales_info VALUES ('2021','05','0101','11001',40,600,'');

--范围分区的销售信息表
CREATE TABLE sales_info_range
(year CHAR(4) NOT NULL,
 month CHAR(2) NOT NULL,
 branch CHAR(4),
 product CHAR(5),
 quantity NUMBER DEFAULT 0 NOT NULL,
 amount NUMBER(10,2) DEFAULT 0 NOT NULL,
 salsperson CHAR(10))
PARTITION BY HASH(branch)
SUBPARTITION BY RANGE(year)
SUBPARTITION TEMPLATE
(SUBPARTITION sp_sales_info_range_1 VALUES LESS THAN('2011'),
SUBPARTITION sp_sales_info_range_2 VALUES LESS THAN('2021'),
SUBPARTITION sp_sales_info_range_3 VALUES LESS THAN('2031'))
PARTITIONS AUTO;
INSERT INTO sales_info_range VALUES ('2001','01','0201','11001',30,500,'0201010011');
INSERT INTO sales_info_range VALUES ('2015','11','0101','11001',20,300,'');
INSERT INTO sales_info_range VALUES ('2021','10','0101','11001',20,300,'');
INSERT INTO sales_info_range VALUES ('2000','12','0102','11001',20,300,'');
INSERT INTO sales_info_range VALUES ('2015','03','0102','11001',20,300,'');
INSERT INTO sales_info_range VALUES ('2021','05','0101','11001',40,600,'');

--列表分区的销售信息表
CREATE TABLE sales_info_list
(year CHAR(4) NOT NULL,
 month CHAR(2) NOT NULL,
 branch CHAR(4),
 product CHAR(5),
 quantity NUMBER DEFAULT 0 NOT NULL,
 amount NUMBER(10,2) DEFAULT 0 NOT NULL,
 salsperson CHAR(10))
PARTITION BY HASH(branch)
SUBPARTITION BY LIST(year)
SUBPARTITION TEMPLATE
(SUBPARTITION sp_sales_info_list_1 VALUES('2018','2019'),
SUBPARTITION sp_sales_info_list_2 VALUES('2021'))
PARTITIONS AUTO;
INSERT INTO sales_info_list VALUES ('2018','10','0101','11001',20,300,'');
INSERT INTO sales_info_list VALUES ('2021','05','0101','11001',40,600,'');

--哈希分区的销售信息表
CREATE TABLE sales_info_hash
(year CHAR(4) NOT NULL,
 month CHAR(2) NOT NULL,
 branch CHAR(4),
 product CHAR(5),
 quantity NUMBER DEFAULT 0 NOT NULL,
 amount NUMBER(10,2) DEFAULT 0 NOT NULL,
 salsperson CHAR(10))
PARTITION BY HASH(branch)
SUBPARTITION BY HASH(year)
SUBPARTITION TEMPLATE
(SUBPARTITION sp_sales_info_hash_1,
 SUBPARTITION sp_sales_info_hash_2)
PARTITIONS AUTO;

--财务信息表
CREATE TABLE finance_info
(year CHAR(4) NOT NULL,
 month CHAR(2) NOT NULL,
 branch CHAR(4),
 revenue_total NUMBER(10,2),
 cost_total NUMBER(10,2),
 fee_total NUMBER(10,2)
```

```
);
CREATE INDEX idx_finance_info_1 ON finance_info (year,month,branch);
INSERT INTO finance_info VALUES ('2001','01','0201',2888,2000,300);
INSERT INTO finance_info VALUES ('2021','01','0201',28888,24000,3000);
INSERT INTO finance_info VALUES ('2021','01','0101',38888,34000,4000);
INSERT INTO finance_info VALUES ('2021','02','0101',37778,33000,6000);

--在数学函数描述中经常使用的表numbers,包含TINYINT、SMALLINT、INT、BIGINT、FLOAT、DOUBLE、NUMBER字段
CREATE TABLE numbers(numbera INT,numberb TINYINT,numberc SMALLINT,numberd BIGINT,numbere FLOAT,numberf DOUBLE,numberg
NUMBER);
INSERT INTO numbers VALUES(-5,55,5555,5555555555555555555,5.555,-5.55555555,555);

--在数学函数描述中经常使用的表numbers_nobit,包含INT、TINYINT、SMALLINT、BIGINT、FLOAT、DOUBLE、NUMBER字段
CREATE TABLE numbers_nobit(numbera INT,numberb TINYINT,numberc SMALLINT,numberd BIGINT,numbere FLOAT,numberf DOUBLE,numberg
NUMBER);
INSERT INTO numbers_nobit VALUES(-5,55,5555,5555555555555555555,5.555,5.55555555555,555);

--在聚集函数描述中经常使用的表area1
CREATE TABLE area1
(area_no CHAR(2) NOT NULL PRIMARY KEY,
 area_name VARCHAR2(60),
 DHQ VARCHAR2(20) DEFAULT 'ShenZhen' NOT NULL,
 employee_count INT);
INSERT INTO area1 VALUES ('01','华东','Shanghai','');
INSERT INTO area1 VALUES ('02','华西','Chengdu',300);
INSERT INTO area1 VALUES ('03','华南','Guangzhou',400);
INSERT INTO area1 VALUES ('04','华北','Beijing',300);
INSERT INTO area1 VALUES ('05','华中','Wuhan','');

--在聚集函数描述中经常使用的表branches1
CREATE TABLE branches1
(branch_no CHAR(4) PRIMARY KEY,
 branch_name VARCHAR2(200) NOT NULL,
 area_no CHAR(2),
 address VARCHAR2(200),
 employee_count INT);
INSERT INTO branches1 VALUES ('0001','深圳','','',40);
INSERT INTO branches1 VALUES ('0101','上海','01','上海市静安区','');
INSERT INTO branches1 VALUES ('0102','南京','01','City of Nanjing',70);
INSERT INTO branches1 VALUES ('0103','福州','01','','');
INSERT INTO branches1 VALUES ('0104','厦门','01','Xiamen','');
INSERT INTO branches1 VALUES ('0401','北京','04','','');
INSERT INTO branches1 VALUES ('0402','天津','04','',10);
INSERT INTO branches1 VALUES ('0403','大连','04','大连市',30);
INSERT INTO branches1 VALUES ('0404','沈阳','04','','');
INSERT INTO branches1 VALUES ('0201','成都','02','','');
INSERT INTO branches1 VALUES ('0501','武汉','','',30);
INSERT INTO branches1 VALUES ('0502','长沙','05','',40);

COMMIT;
```

分布式LSC样例表

```
DROP TABLE IF EXISTS finance_info;
DROP TABLE IF EXISTS sales_info;
DROP TABLE IF EXISTS sales_info_range;
DROP TABLE IF EXISTS sales_info_list;
DROP TABLE IF EXISTS sales_info_hash;
DROP TABLE IF EXISTS orders_info;
DROP TABLE IF EXISTS product;
DROP TABLE IF EXISTS employees;
DROP TABLE IF EXISTS department;
DROP TABLE IF EXISTS branches;
DROP TABLE IF EXISTS area;
DROP TABLE IF EXISTS numbers;
DROP TABLE IF EXISTS numbers_nobit;
DROP TABLE IF EXISTS area1;
DROP TABLE IF EXISTS branches1;
```

```
--区域信息表
CREATE TABLE area
(area_no CHAR(2) NOT NULL PRIMARY KEY,
 area_name VARCHAR2(60),
 DHQ VARCHAR2(20) DEFAULT 'ShenZhen' NOT NULL);
INSERT INTO area VALUES ('01','华东','Shanghai');
INSERT INTO area VALUES ('02','华西','Chengdu');
INSERT INTO area VALUES ('03','华南','Guangzhou');
INSERT INTO area VALUES ('04','华北','Beijing');
INSERT INTO area VALUES ('05','华中','Wuhan');
--机构信息表
CREATE TABLE branches
(branch_no CHAR(4),
branch_name VARCHAR2(200) NOT NULL,
area_no CHAR(2),
address VARCHAR2(200));
INSERT INTO branches VALUES ('0001','深圳','','');
INSERT INTO branches VALUES ('0101','上海','01','上海市静安区');
INSERT INTO branches VALUES ('0102','南京','01','City of Nanjing');
INSERT INTO branches VALUES ('0103','福州','01','');
INSERT INTO branches VALUES ('0104','厦门','01','Xiamen');
INSERT INTO branches VALUES ('0401','北京','04','');
INSERT INTO branches VALUES ('0402','天津','04','');
INSERT INTO branches VALUES ('0403','大连','04','大连市');
INSERT INTO branches VALUES ('0404','沈阳','04','');
INSERT INTO branches VALUES ('0201','成都','02','');
INSERT INTO branches VALUES ('0501','武汉','','');
INSERT INTO branches VALUES ('0502','长沙','05','');
--部门信息表
CREATE TABLE department
(deparment_no CHAR(3),
department_name VARCHAR2(20));
INSERT INTO department VALUES('000','不分部门');
INSERT INTO department VALUES('010','销售部');
INSERT INTO department VALUES('008','采购部');
INSERT INTO department VALUES('002','财务部');
--员工信息表
CREATE TABLE employees
(branch CHAR(4),
department CHAR(3),
employee_no CHAR(10) NOT NULL,
employee_name VARCHAR2(10),
sex CHAR(1),
entry_date DATE
);
INSERT INTO employees VALUES ('0101','000','0101000001','Mask','1',SYSDATE-1000);
INSERT INTO employees VALUES ('0101','000','0101000002','John','1',SYSDATE-2000);
INSERT INTO employees VALUES ('0201','010','0201010011','Anna','0',SYSDATE-300);
INSERT INTO employees VALUES ('0201','008','0201008003','Jack','1',SYSDATE-700);
INSERT INTO employees VALUES ('0101','008','0201008004','Jim','1',SYSDATE-200);
--产品信息表
CREATE TABLE product
(product_no CHAR(5),
product_name VARCHAR2(30),
cost NUMBER,
price NUMBER
);
INSERT INTO product VALUES ('11001','product001',8,10);
INSERT INTO product VALUES ('11002','product002',13,16);
INSERT INTO product VALUES ('10001','product001',99,100);
INSERT INTO product VALUES ('10002','product002',199,200);
--订单信息表
CREATE TABLE orders_info (order_no CHAR(14) NOT NULL,
product_no CHAR(5),
area CHAR(2),
branch CHAR(4),
order_date DATE DEFAULT SYSDATE NOT NULL,
salesperson CHAR(10),
id NUMBER)
PARTITION BY HASH(id)
```

```
SUBPARTITION BY RANGE (area )
SUBPARTITION TEMPLATE
(SUBPARTITION sp_orders_info_1 VALUES LESS THAN (2),
SUBPARTITION sp_orders_info_2 VALUES LESS THAN (4),
SUBPARTITION sp_orders_info_3 VALUES LESS THAN (MAXVALUE))
PARTITIONS AUTO;
INSERT INTO orders_info VALUES ('20010102020001','11001','01','0101',SYSDATE-400,'0201010011',300);
INSERT INTO orders_info VALUES ('20010102020001','11002','02','0201',SYSDATE-400,'0201008003',1300);
INSERT INTO orders_info VALUES ('20010102020001','10001','04','0402',SYSDATE-400,'0201010011',2300);
INSERT INTO orders_info VALUES ('20210102020002','11001','04','0401',SYSDATE-400,'0201008003',400);
INSERT INTO orders_info VALUES ('20210102020002','11002','05','0501',SYSDATE-400,'0201010011',200);
INSERT INTO orders_info VALUES ('20210102020002','10001','01','0102',SYSDATE-400,'0201008003',100);
--销售信息表
CREATE TABLE sales_info
(year CHAR(4) NOT NULL,
month CHAR(2) NOT NULL,
branch CHAR(4),
product CHAR(5),
quantity NUMBER DEFAULT 0 NOT NULL,
amount NUMBER(10,2) DEFAULT 0 NOT NULL,
salsperson CHAR(10))
PARTITION BY HASH(branch)
SUBPARTITION BY LIST(year)
SUBPARTITION TEMPLATE
(SUBPARTITION sp_sales_info_1 VALUES ('2001','2002','2010'),
SUBPARTITION sp_sales_info_2 VALUES ('2021','2020','2019'),
SUBPARTITION sp_sales_info_3 VALUES (DEFAULT))
PARTITIONS AUTO;
INSERT INTO sales_info VALUES ('2001','01','0201','11001',30,500,'0201010011');
INSERT INTO sales_info VALUES ('2015','11','0101','11001',20,300,'');
INSERT INTO sales_info VALUES ('2021','10','0402','11001',20,300,'');
INSERT INTO sales_info VALUES ('2000','12','0102','11001',20,300,'');
INSERT INTO sales_info VALUES ('2015','03','0102','11001',20,300,'');
INSERT INTO sales_info VALUES ('2021','05','0101','11001',40,600,'');
--范围分区的销售信息表
CREATE TABLE sales_info_range
(year CHAR(4) NOT NULL,
month CHAR(2) NOT NULL,
branch CHAR(4),
product CHAR(5),
quantity NUMBER DEFAULT 0 NOT NULL,
amount NUMBER(10,2) DEFAULT 0 NOT NULL,
salsperson CHAR(10))
PARTITION BY HASH(branch)
SUBPARTITION BY RANGE(year)
SUBPARTITION TEMPLATE
(SUBPARTITION sp_sales_info_range_1 VALUES LESS THAN('2011'),
SUBPARTITION sp_sales_info_range_2 VALUES LESS THAN('2021'),
SUBPARTITION sp_sales_info_range_3 VALUES LESS THAN('2031'))
PARTITIONS AUTO;
INSERT INTO sales_info_range VALUES ('2001','01','0201','11001',30,500,'0201010011');
INSERT INTO sales_info_range VALUES ('2015','11','0101','11001',20,300,'');
INSERT INTO sales_info_range VALUES ('2021','10','0101','11001',20,300,'');
INSERT INTO sales_info_range VALUES ('2000','12','0102','11001',20,300,'');
INSERT INTO sales_info_range VALUES ('2015','03','0102','11001',20,300,'');
INSERT INTO sales_info_range VALUES ('2021','05','0101','11001',40,600,'');
--列表分区的销售信息表
CREATE TABLE sales_info_list
(year CHAR(4) NOT NULL,
month CHAR(2) NOT NULL,
branch CHAR(4),
product CHAR(5),
quantity NUMBER DEFAULT 0 NOT NULL,
amount NUMBER(10,2) DEFAULT 0 NOT NULL,
salsperson CHAR(10))
PARTITION BY HASH(branch)
SUBPARTITION BY LIST(year)
SUBPARTITION TEMPLATE
(SUBPARTITION sp_sales_info_list_1 VALUES('2018','2019'),
SUBPARTITION sp_sales_info_list_2 VALUES('2021'))
```

```
PARTITIONS AUTO;
INSERT INTO sales_info_list VALUES ('2018','10','0101','11001',20,300,'');
INSERT INTO sales_info_list VALUES ('2021','05','0101','11001',40,600,'');
--哈希分区的销售信息表
CREATE TABLE sales_info_hash
(year CHAR(4) NOT NULL,
month CHAR(2) NOT NULL,
branch CHAR(4),
product CHAR(5),
quantity NUMBER DEFAULT 0 NOT NULL,
amount NUMBER(10,2) DEFAULT 0 NOT NULL,
salsperson CHAR(10))
PARTITION BY HASH(product)
SUBPARTITION BY HASH(year)
SUBPARTITION TEMPLATE
(SUBPARTITION sp_sales_info_hash_1,
SUBPARTITION sp_sales_info_hash_2)
PARTITIONS AUTO;
--财务信息表
CREATE TABLE finance_info
(year CHAR(4) NOT NULL,
month CHAR(2) NOT NULL,
branch CHAR(4),
revenue_total NUMBER(10,2),
cost_total NUMBER(10,2),
fee_total NUMBER(10,2)
);
INSERT INTO finance_info VALUES ('2001','01','0201',2888,2000,300);
INSERT INTO finance_info VALUES ('2021','01','0201',28888,24000,3000);
INSERT INTO finance_info VALUES ('2021','01','0101',38888,34000,4000);
INSERT INTO finance_info VALUES ('2021','02','0101',37778,33000,6000);
--在数学函数描述中经常使用的表numbers,包含TINYINT、SMALLINT、INT、BIGINT、FLOAT、DOUBLE、NUMBER字段
CREATE TABLE numbers(numbera INT,numberb TINYINT,numberc SMALLINT,numberd
BIGINT,numbere FLOAT,numberf DOUBLE,numberg NUMBER);
INSERT INTO numbers VALUES(-5,55,5555,5555555555555555555,5.555,-5.55555555,555);
--在数学函数描述中经常使用的表numbers_nobit,包含INT、TINYINT、SMALLINT、BIGINT、FLOAT、DOUBLE、NUMBER字段
CREATE TABLE numbers_nobit(numbera INT,numberb TINYINT,numberc SMALLINT,
numberd BIGINT,numbere FLOAT,numberf DOUBLE,numberg NUMBER);
INSERT INTO numbers_nobit VALUES
(-5,55,5555,5555555555555555555,5.555,5.55555555555,555);
--在聚集函数描述中经常使用的表area1
CREATE TABLE area1
(area_no CHAR(2) NOT NULL,
area_name VARCHAR2(60),
DHQ VARCHAR2(20) DEFAULT 'ShenZhen' NOT NULL,
employee_count INT);
INSERT INTO area1 VALUES ('01','华东','Shanghai','');
INSERT INTO area1 VALUES ('02','华西','Chengdu',300);
INSERT INTO area1 VALUES ('03','华南','Guangzhou',400);
INSERT INTO area1 VALUES ('04','华北','Beijing',300);
INSERT INTO area1 VALUES ('05','华中','Wuhan','');
--在聚集函数描述中经常使用的表branches1
CREATE TABLE branches1
(branch_no CHAR(4),
branch_name VARCHAR2(200) NOT NULL,
area_no CHAR(2),
address VARCHAR2(200),
employee_count INT);
INSERT INTO branches1 VALUES ('0001','深圳','','',40);
INSERT INTO branches1 VALUES ('0101','上海','01','上海市静安区','');
INSERT INTO branches1 VALUES ('0102','南京','01','City of Nanjing',70);
INSERT INTO branches1 VALUES ('0103','福州','01','','');
INSERT INTO branches1 VALUES ('0104','厦门','01','Xiamen','');
INSERT INTO branches1 VALUES ('0401','北京','04','','');
INSERT INTO branches1 VALUES ('0402','天津','04','',10);
INSERT INTO branches1 VALUES ('0403','大连','04','大连市',30);
INSERT INTO branches1 VALUES ('0404','沈阳','04','','');
INSERT INTO branches1 VALUES ('0201','成都','02','','');
INSERT INTO branches1 VALUES ('0501','武汉','','',30);
```

```
INSERT INTO branches1 VALUES ('0502','长沙','05','',40);
COMMIT;
```

共享集群HEAP样例表

```
DROP TABLE IF EXISTS finance_info;
DROP TABLE IF EXISTS sales_info;
DROP TABLE IF EXISTS sales_info_range;
DROP TABLE IF EXISTS sales_info_list;
DROP TABLE IF EXISTS sales_info_hash;
DROP TABLE IF EXISTS orders_info;
DROP TABLE IF EXISTS product;
DROP TABLE IF EXISTS employees;
DROP TABLE IF EXISTS department;
DROP TABLE IF EXISTS branches;
DROP TABLE IF EXISTS area;
DROP TABLE IF EXISTS numbers;
DROP TABLE IF EXISTS numbers_nobit;
DROP TABLE IF EXISTS area1;
DROP TABLE IF EXISTS branches1;

--区域信息表
CREATE TABLE area
(area_no CHAR(2) NOT NULL PRIMARY KEY,
 area_name VARCHAR2(60),
 DHQ VARCHAR2(20) DEFAULT 'ShenZhen' NOT NULL);
INSERT INTO area VALUES ('01','华东','Shanghai');
INSERT INTO area VALUES ('02','华西','Chengdu');
INSERT INTO area VALUES ('03','华南','Guangzhou');
INSERT INTO area VALUES ('04','华北','Beijing');
INSERT INTO area VALUES ('05','华中','Wuhan');

--机构信息表
CREATE TABLE branches
(branch_no CHAR(4) PRIMARY KEY,
 branch_name VARCHAR2(200) NOT NULL,
 area_no CHAR(2)  CONSTRAINT c_branches_1 REFERENCES area(area_no) ON DELETE SET NULL,
 address VARCHAR2(200));
INSERT INTO branches VALUES ('0001','深圳','','');
INSERT INTO branches VALUES ('0101','上海','01','上海市静安区');
INSERT INTO branches VALUES ('0102','南京','01','City of Nanjing');
INSERT INTO branches VALUES ('0103','福州','01','');
INSERT INTO branches VALUES ('0104','厦门','01','Xiamen');
INSERT INTO branches VALUES ('0401','北京','04','');
INSERT INTO branches VALUES ('0402','天津','04','');
INSERT INTO branches VALUES ('0403','大连','04','大连市');
INSERT INTO branches VALUES ('0404','沈阳','04','');
INSERT INTO branches VALUES ('0201','成都','02','');
INSERT INTO branches VALUES ('0501','武汉','','');
INSERT INTO branches VALUES ('0502','长沙','05','');

--部门信息表
CREATE TABLE department
(deparment_no CHAR(3) PRIMARY KEY,
 department_name VARCHAR2(20));
INSERT INTO department VALUES('000','不分部门');
INSERT INTO department VALUES('010','销售部');
INSERT INTO department VALUES('008','采购部');
INSERT INTO department VALUES('002','财务部');

--员工信息表
CREATE TABLE employees
(branch CHAR(4)   CONSTRAINT c_employees_1 REFERENCES branches(branch_no) ON DELETE SET NULL,
 department CHAR(3)   CONSTRAINT c_employees_2 REFERENCES department(deparment_no) ON DELETE SET NULL,
 employee_no CHAR(10) NOT NULL PRIMARY KEY,
 employee_name VARCHAR2(10),
 sex CHAR(1),
 entry_date DATE
```

```
 );
INSERT INTO employees VALUES ('0101','000','0101000001','Mask','1',SYSDATE-1000);
INSERT INTO employees VALUES ('0101','000','0101000002','John','1',SYSDATE-2000);
INSERT INTO employees VALUES ('0201','010','0201010011','Anna','0',SYSDATE-300);
INSERT INTO employees VALUES ('0201','008','0201008003','Jack','1',SYSDATE-700);
INSERT INTO employees VALUES ('0101','008','0201008004','Jim','1',SYSDATE-200);

--产品信息表
CREATE TABLE product
(product_no CHAR(5) PRIMARY KEY,
 product_name VARCHAR2(30),
 cost NUMBER,
 price NUMBER
);
INSERT INTO product VALUES ('11001','product001',8,10);
INSERT INTO product VALUES ('11002','product002',13,16);
INSERT INTO product VALUES ('10001','product001',99,100);
INSERT INTO product VALUES ('10002','product002',199,200);

--订单信息表
CREATE TABLE orders_info (order_no CHAR(14) NOT NULL,
             product_no CHAR(5) CONSTRAINT c_orders_info_3 REFERENCES product(product_no) ON DELETE SET NULL,
             area CHAR(2) CONSTRAINT c_orders_info_2 REFERENCES area(area_no) ON DELETE SET NULL,
             branch CHAR(4) CONSTRAINT c_orders_info_1 REFERENCES branches(branch_no) ON DELETE SET NULL,
             order_date DATE DEFAULT SYSDATE NOT NULL,
             salesperson CHAR(10) CONSTRAINT c_orders_info_4 REFERENCES employees(employee_no) ON DELETE SET NULL,
             id NUMBER)
    PARTITION BY RANGE (id )
    INTERVAL (1000)
    (PARTITION p_orders_info_1 VALUES LESS THAN (800));
INSERT INTO orders_info VALUES ('20010102020001','11001','01','0101',SYSDATE-400,'0201010011',300);
INSERT INTO orders_info VALUES ('20010102020001','11002','02','0201',SYSDATE-400,'0201008003',1300);
INSERT INTO orders_info VALUES ('20010102020001','10001','04','0402',SYSDATE-400,'0201010011',2300);
INSERT INTO orders_info VALUES ('20210102020002','11001','04','0401',SYSDATE-400,'0201008003',400);
INSERT INTO orders_info VALUES ('20210102020002','11002','05','0501',SYSDATE-400,'0201010011',200);
INSERT INTO orders_info VALUES ('20210102020002','10001','01','0102',SYSDATE-400,'0201008003',100);

--销售信息表
CREATE TABLE sales_info
(year CHAR(4) NOT NULL,
 month CHAR(2) NOT NULL,
 branch CHAR(4) CONSTRAINT c_sales_info_1 REFERENCES branches(branch_no) ON DELETE SET NULL,
 product CHAR(5) CONSTRAINT c_sales_info_2 REFERENCES product(product_no) ON DELETE SET NULL,
 quantity NUMBER DEFAULT 0 NOT NULL,
 amount NUMBER(10,2) DEFAULT 0 NOT NULL,
 salsperson CHAR(10) CONSTRAINT c_sales_info_3 REFERENCES employees(employee_no) ON DELETE SET NULL)
PARTITION BY HASH(branch)
SUBPARTITION BY LIST(year)
SUBPARTITION template (SUBPARTITION sp_sales_info_1 VALUES ('2001','2002','2010'),
 SUBPARTITION sp_sales_info_2 VALUES ('2021','2020','2019'),
 SUBPARTITION sp_sales_info_3 VALUES (DEFAULT))
(PARTITION p_sales_info_1,PARTITION p_sales_info_2,PARTITION p_sales_info_3);

CREATE INDEX idx_sales_info_1 ON sales_info (year,month,branch,product) LOCAL
(PARTITION ip_sales_info_1 (SUBPARTITION isp_sales_info_11,SUBPARTITION isp_sales_info_12,SUBPARTITION isp_sales_info_13),
PARTITION ip_sales_info_2 (SUBPARTITION isp_sales_info_21,SUBPARTITION isp_sales_info_22,SUBPARTITION isp_sales_info_23),
PARTITION ip_sales_info_3 (SUBPARTITION isp_sales_info_31,SUBPARTITION isp_sales_info_32,SUBPARTITION isp_sales_info_33));

INSERT INTO sales_info VALUES ('2001','01','0201','11001',30,500,'0201010011');
INSERT INTO sales_info VALUES ('2015','11','0101','11001',20,300,'');
INSERT INTO sales_info VALUES ('2021','10','0402','11001',20,300,'');
INSERT INTO sales_info VALUES ('2000','12','0102','11001',20,300,'');
INSERT INTO sales_info VALUES ('2015','03','0102','11001',20,300,'');
INSERT INTO sales_info VALUES ('2021','05','0101','11001',40,600,'');

--范围分区的销售信息表
CREATE TABLE sales_info_range
(year CHAR(4) NOT NULL,
 month CHAR(2) NOT NULL,
 branch CHAR(4),
```

```
 product CHAR(5),
 quantity NUMBER DEFAULT 0 NOT NULL,
 amount NUMBER(10,2) DEFAULT 0 NOT NULL,
 salsperson CHAR(10))
PARTITION BY RANGE(year)
(PARTITION p_sales_info_range_1 VALUES LESS THAN('2011'),
 PARTITION p_sales_info_range_2 VALUES LESS THAN('2021'),
 PARTITION p_sales_info_range_3 VALUES LESS THAN('2031'));
INSERT INTO sales_info_range VALUES ('2001','01','0201','11001',30,500,'0201010011');
INSERT INTO sales_info_range VALUES ('2015','11','0101','11001',20,300,'');
INSERT INTO sales_info_range VALUES ('2021','10','0101','11001',20,300,'');
INSERT INTO sales_info_range VALUES ('2000','12','0102','11001',20,300,'');
INSERT INTO sales_info_range VALUES ('2015','03','0102','11001',20,300,'');
INSERT INTO sales_info_range VALUES ('2021','05','0101','11001',40,600,'');

--列表分区的销售信息表
CREATE TABLE sales_info_list
(year CHAR(4) NOT NULL,
 month CHAR(2) NOT NULL,
 branch CHAR(4),
 product CHAR(5),
 quantity NUMBER DEFAULT 0 NOT NULL,
 amount NUMBER(10,2) DEFAULT 0 NOT NULL,
 salsperson CHAR(10))
PARTITION BY LIST(year)
(PARTITION p_sales_info_list_1 VALUES('2018','2019'),
 PARTITION p_sales_info_list_2 VALUES('2021'));
INSERT INTO sales_info_list VALUES ('2018','10','0101','11001',20,300,'');
INSERT INTO sales_info_list VALUES ('2021','05','0101','11001',40,600,'');

--哈希分区的销售信息表
CREATE TABLE sales_info_hash
(year CHAR(4) NOT NULL,
 month CHAR(2) NOT NULL,
 branch CHAR(4),
 product CHAR(5),
 quantity NUMBER DEFAULT 0 NOT NULL,
 amount NUMBER(10,2) DEFAULT 0 NOT NULL,
 salsperson CHAR(10))
PARTITION BY HASH(year)
(PARTITION p_sales_info_hash_1,
 PARTITION p_sales_info_hash_2);

--财务信息表
CREATE TABLE finance_info
(year CHAR(4) NOT NULL,
 month CHAR(2) NOT NULL,
 branch CHAR(4) CONSTRAINT c_finance_info_1 REFERENCES branches(branch_no) ON DELETE SET NULL,
 revenue_total NUMBER(10,2),
 cost_total NUMBER(10,2),
 fee_total NUMBER(10,2)
);
CREATE INDEX idx_finance_info_1 ON finance_info (year,month,branch);
INSERT INTO finance_info VALUES ('2001','01','0201',2888,2000,300);
INSERT INTO finance_info VALUES ('2021','01','0201',28888,24000,3000);
INSERT INTO finance_info VALUES ('2021','01','0101',38888,34000,4000);
INSERT INTO finance_info VALUES ('2021','02','0101',37778,33000,6000);

--在数学函数描述中经常使用的表numbers,包含TINYINT、SMALLINT、INT、BIGINT、FLOAT、DOUBLE、NUMBER、BIT字段
CREATE TABLE numbers(numbera INT,numberb TINYINT,numberc SMALLINT,numberd BIGINT,numbere FLOAT,numberf DOUBLE,numberg
NUMBER,numberh BIT);
INSERT INTO numbers VALUES(-5,55,5555,5555555555555555555,5.555,-5.55555555,555,1);

--在数学函数描述中经常使用的表numbers_nobit,包含INT、TINYINT、SMALLINT、BIGINT、FLOAT、DOUBLE、NUMBER字段
CREATE TABLE numbers_nobit(numbera INT,numberb TINYINT,numberc SMALLINT,numberd BIGINT,numbere FLOAT,numberf DOUBLE,numberg
NUMBER);
INSERT INTO numbers_nobit VALUES(-5,55,5555,5555555555555555555,5.555,5.55555555555,555);

--在聚集函数描述中经常使用的表area1
CREATE TABLE area1
```

```
(area_no CHAR(2) NOT NULL PRIMARY KEY,
 area_name VARCHAR2(60),
 DHQ VARCHAR2(20) DEFAULT 'ShenZhen' NOT NULL,
 employee_count INT);
INSERT INTO area1 VALUES ('01','华东','Shanghai','');
INSERT INTO area1 VALUES ('02','华西','Chengdu',300);
INSERT INTO area1 VALUES ('03','华南','Guangzhou',400);
INSERT INTO area1 VALUES ('04','华北','Beijing',300);
INSERT INTO area1 VALUES ('05','华中','Wuhan','');

--在聚集函数描述中经常使用的表branches1
CREATE TABLE branches1
(branch_no CHAR(4) PRIMARY KEY,
 branch_name VARCHAR2(200) NOT NULL,
 area_no CHAR(2),
 address VARCHAR2(200),
 employee_count INT);
INSERT INTO branches1 VALUES ('0001','深圳','','',40);
INSERT INTO branches1 VALUES ('0101','上海','01','上海市静安区','');
INSERT INTO branches1 VALUES ('0102','南京','01','City of Nanjing',70);
INSERT INTO branches1 VALUES ('0103','福州','01','','');
INSERT INTO branches1 VALUES ('0104','厦门','01','Xiamen','');
INSERT INTO branches1 VALUES ('0401','北京','04','','');
INSERT INTO branches1 VALUES ('0402','天津','04','',10);
INSERT INTO branches1 VALUES ('0403','大连','04','大连市',30);
INSERT INTO branches1 VALUES ('0404','沈阳','04','','');
INSERT INTO branches1 VALUES ('0201','成都','02','','');
INSERT INTO branches1 VALUES ('0501','武汉','','',30);
INSERT INTO branches1 VALUES ('0502','长沙','05','',40);

COMMIT;
```