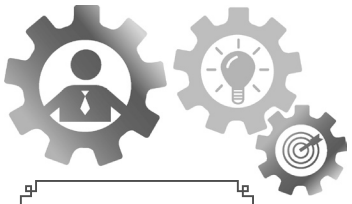

第 5 章

KingbaseES 的 DML 语句

数据操作语言(Data Manipulating Language,DML)是对基本表中的数据进行查询(SELECT)、插入(INSERT)、修改(UPDATE)、删除(DELETE)等操作的 SQL。SELECT 不修改数据,不改变数据库的状态,因而执行时数据库系统不进行完整性约束检查;而 INSERT、UPDATE 和 DELETE 要修改数据,改变数据库的状态,因而执行时数据库系统将进行完整性约束检查,满足相应的完整性约束,则可以成功执行,否则不能成功执行。

DML 语句操作对象主要是数据库基本表,也可以对视图、物化视图进行操作。

5.1 插入语句

在数据库中,一个表被创建后,表中并不包含数据。因此用户在创建表后要使用数据插入语句(INSERT)向表中插入数据。

INSERT ON CONFLICT 是一种特殊的更新,是 INSERT 和 UPDATE 的组合。ON CONFLICT 可以用来指定发生唯一约束或者排除约束违背错误时的替换动作。在关系数据库的上下文中,如果表中已经存在指定值,则更新现有行,如果指定值不存在,则插入新行。

INSERT 符合 SQL 标准,但是 RETURNING 子句是一种 KingbaseES 扩展,在 INSERT 中使用 WITH 也是,用 ON CONFLICT 指定一个替代动作也是扩展。

5.1.1 基本插入与批量插入

1. 基本插入

在进行 INSERT 操作时,数据是以元组的形式被插入表中的。可以一次插入一个元组,也可以插入多个元组或者更多由值表达式指定的元组,或者插入来自一个查询的元组。语法为:

```
INSERT INTO TABLE( [column1, column2, …]) values ( value1, value2, …);
```

属性列与值必须要一一对应。如果任意列的值不是正确的数据类型,将会尝试自动类型转换。对每一个没有出现在显式或者隐式列列表中的列,如果为该列声明过默认值则用默认值填充,否则用空值填充。但要注意,在表定义时说明了 NOT NULL 的列不能取空

值，否则会出错。

如果 INTO 子句中没有指明任何列名，则新插入的元组必须在每个属性列上都有值。

向表中插入数据时，必须拥有表上的 INSERT 特权；若一个列列表被指定，只需要列上的 INSERT 特权。

例 5.1 使用 INSERT 命令向 Categories 表中添加一条数据。

```
INSERT INTO Categories VALUES (1, '图书,音像', 1, 1);
```

说明：若未指定属性列，数据的值按照这些列在表中出现的顺序列出，并且用逗号分隔。通常，数据的值是常量，但也允许使用标量表达式。此语法的缺点是必须知道表中列的顺序和各列的数据类型。

例 5.2 显式地列出属性列，并分别向部分列和全部属性列插入数据。

```
INSERT INTO Categories (catgid, catgname, currlevel) VALUES (2, '电子书刊', 2);
INSERT INTO Categories (catgid, catgname, parentid, currlevel) VALUES (3, '电子书
', 2, 3);
```

插入成功后查询此表的数据内容：

```
SELECT * FROM Categories WHERE catgid in (2, 3);
```

查询结果如下。

```
 catgid |  catgname  | parentid | currlevel
--------+------------+----------+-----------
      2 |电子书刊    |   NULL   |         2
      3 |电子书      |        2 |         3
(2 行记录)
```

说明：该例的第一个 INSERT 语句中省略了 currlevel 列，系统自动赋值 NULL。

例 5.3 插入一个完全由默认值构成的行。

```
INSERT INTO Categories DEFAULT VALUES;
```

执行结果如下。

```
错误:  在字段 "catgid" 中空值违反了非空约束
描述:  失败, 行包含(null, null, null, null).
原因:由于定义了主码,主码不能取空值,所以插入失败。
```

创建从 1 开始的序列 CATEGORIES_SEQ，创建表 Categories2，主码 catgid 的默认值为 CATEGORIES_SEQ 上的值，如下。

```
CREATE SEQUENCE CATEGORIES_SEQ START WITH 1;
CREATE TABLE Categories2(                                    /*商品分类*/
/*分类编码 category id*/
```

```
    catgid INTEGER PRIMARY KEY DEFAULT NEXTVAL('CATEGORIES_SEQ'),
    catgname VARCHAR(128),                                  /*分类名称*/
    parentid INTEGER REFERENCES Categories(catgid),         /*父类编码*/
    currlevel INTEGER                                  /*当前层级 current level*/  );
```

现在向表中插入默认值：

```
INSERT INTO Categories2 DEFAULT VALUES;
```

插入成功后查询此表的数据内容：

```
SELECT * FROM Categories2;
```

查询结果如下。

```
 catgid | catgname | parentid | currlevel
--------+----------+----------+-----------
      1 | NULL     | NULL     | NULL
(1 行记录)
```

2. 批量插入

有时，KingbaseES 数据库需要在单个或最少的步骤中导入大量数据，这通常称为批量数据导入。其中，数据源通常是一个或多个大文件，这个过程有时可能非常慢。

为了兼容 Oracle 特性，KingbaseES 提供了一种将数据同时插入到多个目标表的功能。命令为 INSERT ALL/FIRST。下面介绍四种实现方法。

方法一：使用多行 VALUES 语法向一个表中插入多个行，各行之间用逗号分隔，基本语法为：

```
INSERT INTO tablename(column1,column2,…)VALUES(),(),(),…
```

例 5.4　向 Adminaddrs 表中添加多行数据。

```
INSERT ALL INTO Adminaddrs VALUES (33,'布隆迪',33,0), (35,'喀麦隆',35,0),
  (37,'佛得角',37,0);
```

多行 VALUES 方法的性能受现有索引的影响。建议在运行命令之前删除索引，并在之后重新创建索引。

方法二：通过子查询或函数批量插入数据，基本语法为：

```
INSERT INTO … SELECT …
```

例 5.5　创建一张表结构和 Adminaddrs 相同的表并插入 Adminaddrs 表的全部数据。

```
CREATE TABLE Adminaddrs_batch(          /*行政区划地址 administration address*/
  addrid VARCHAR(12) PRIMARY KEY,       /*地址编码*/
```

```
  addrname VARCHAR(40),                                  /*地址名称*/
  parentid VARCHAR(12) REFERENCES Adminaddrs(addrid),   /*父地址编码*/
  currlevel INTEGER                                      /*当前层级 current level*/);
INSERT INTO Adminaddrs_batch (addrid, addrname, parentid, currlevel)
SELECT addrid, addrname, parentid, currlevel FROM Adminaddrs;
```

说明：表 Adminaddrs_batch 的数据与表 Adminaddrs 相同。

方法三：使用 INSERT ALL 将每行数据同时插入到满足条件的多个目标表中。我们可以插入一个或者多个由值表达式指定的行，或者插入某个查询的结果。INSERT ALL 可以使用条件表达式，或者不使用条件表达式。基本语法为：

```
INSERT ALL into_clause [into_clause]…  subquery;
into_clause ::= INTO [schema.]{table_name|view_name}[t_alias]
   [(column_name [,column_name]..)] [values_clause]
values_clause ::= VALUES ({expre|default}[,{expre|default}]…)
```

例 5.6　向 Adminaddrs、Adminaddrs1 表中分别添加一行数据。

```
INSERT ALL INTO Adminaddrs VALUES(22, '贝宁', 22, 0)
  INTO Adminaddrs1 VALUES(22, '贝宁', 22, 0)
SELECT * FROM DVAL;
```

查询数据是否正确插入，Adminaddrs 表中的数据：

```
SELECT * FROM Adminaddrs;
```

查询结果如下。

```
addrid |  addrname  | parentid | currlevel
------ +---------- +-------- +-----------
     1  |阿富汗      |      1   |      0
     2  |阿尔巴尼亚  |      2   |      0
     3  |安道尔      |      3   |      0
    33  |布隆迪      |     33   |      0
    35  |喀麦隆      |     35   |      0
    37  |佛得角      |     37   |      0
    22  |贝宁        |     22   |      0
(7 行记录)
```

Adminaddrs1 表中的数据：

```
SELECT * FROM Adminaddrs1;
```

查询结果如下。

```
 addrid | addrname | parentid | currlevel
------- +-------- +-------- +-----------
    22  |贝宁      |     22   |      0
(1 行记录)
```

例 5.7　向表 Adminaddrs 和表 Adminaddrs1 分别插入一行新的数据。

```
INSERT ALL INTO Adminaddrs (addrid, addrname, parentid) VALUES (173, '卢旺达', 173)
INTO Adminaddrs1 (addrid, addrname, parentid, currlevel) VALUES (182, '新加坡',
182, 0)
SELECT * FROM DVAL;
```

查询数据是否正确插入，Adminaddrs 表中的数据：

```
SELECT * FROM Adminaddrs;
```

查询结果如下。

```
 addrid |  addrname  | parentid | currlevel
--------+------------+----------+-----------
      1 | 阿富汗     |        1 |         0
      2 | 阿尔巴尼亚 |        2 |         0
      3 | 安道尔     |        3 |         0
     33 | 布隆迪     |       33 |         0
     35 | 喀麦隆     |       35 |         0
     37 | 佛得角     |       37 |         0
     22 | 贝宁       |       22 |         0
    173 | 卢旺达     |      173 |      NULL
(8 行记录)
```

查看 Adminaddrs1 表中的数据：

```
SELECT * FROM Adminaddrs1;
```

查询结果如下。

```
 addrid | addrname | parentid | currlevel
--------+----------+----------+-----------
     22 | 贝宁     |       22 |         0
    182 | 新加坡   |      182 |         0
(2 行记录)
```

例 5.8　向 Adminaddrs 表中插入包含不同列的多行数据，也可使用 INSERT ALL 语句。

```
INSERT ALL INTO Adminaddrs (addrid, addrname, parentid)
             VALUES (187, '苏丹', 187)
          INTO Adminaddrs (addrid, addrname, parentid, currlevel)
             VALUES (190, '瑞典', 190,0)
SELECT * FROM DVAL;
```

查询数据是否正确插入：

```
SELECT * FROM Adminaddrs;
```

查询结果如下。

```
 addrid |  addrname  | parentid | currlevel
--------+------------+----------+-------------
      1 | 阿富汗     |        1 |           0
      2 | 阿尔巴尼亚 |        2 |           0
      3 | 安道尔     |        3 |           0
     33 | 布隆迪     |       33 |           0
     35 | 喀麦隆     |       35 |           0
     37 | 佛得角     |       37 |           0
     22 | 贝宁       |       22 |           0
    173 | 卢旺达     |      173 |        NULL
    187 | 苏丹       |      187 |        NULL
    190 | 瑞典       |      190 |           0
(10 行记录)
```

方法四：使用 INSERT FIRST 将每行数据插入第一个满足条件的目标表。INSERT FIRST 必须和 WHEN 条件表达式结合使用。基本语法为：

```
INSERT {ALL|FIRST} condition_clause [condition_clause] condition_else_clause
subquery;
condition_clause ::= WHEN condition_expr THEN into_clause [, into_clause, …]
condition_else_clause ::= [ELSE into_clause [, into_clause, …]]
```

假设 Categories 表中已插入如下数据。

```
catgid |    catgname     | parentid | currlevel
-------+-----------------+----------+-------------
     1 | 图书,音像       |     NULL |           1
     2 | 电子书刊        |        1 |           2
     7 | 音像            |        1 |           2
    11 | 英文原版        |        1 |           2
    18 | 文艺            |        1 |           2
     3 | 电子书          |        2 |           3
    10 | 教育音像        |        7 |           3
    13 | 商务投资        |       11 |           3
    20 | 文学            |       18 |           3
    21 | 青春文学        |       20 |           4
    23 | 传记            |       21 |           5
(11 行记录)
```

例 5.9 假设已新建了两个与 Categories 表结构相同的空表 Categories1、Categories2，现要从 Categories 表中查询 parentid＜20 的数据，并将 parentid＜10 的行插入到 Categories1 中、将 parentid 介于 10～20 的行插入到 Categories2 中。

```
INSERT FIRST
       WHEN parentid < 10 THEN INTO Categories1
       WHEN parentid > 10 AND parentid < 20 THEN INTO Categories2;
```

命令执行后，查看 Categories1 表的数据：

```
SELECT * FROM Categories1;
```

查询结果如下。

```
catgid |    catgname    | parentid | currlevel
-------+----------------+----------+-----------
     2 | 电子书刊       |        1 |         2
     7 | 音像           |        1 |         2
    11 | 英文原版       |        1 |         2
    18 | 文艺           |        1 |         2
     3 | 电子书         |        2 |         3
    10 | 教育音像       |        7 |         3
(6 行记录)
```

查看 Categories2 表的数据：

```
SELECT * FROM Categories2;
```

查询结果如下。

```
 catgid |    catgname    | parentid | currlevel
--------+----------------+----------+-----------
     13 | 商务投资       |       11 |         3
     20 | 文学           |       18 |         3
(2 行记录)
```

例 5.10　删除 Categories1 和 Categories2 表中的数据，从 Categories 表中查询 currlevel ＜ 5 的所有行，将其中 currlevel ＜＝ 2 的行插入 Categories1，其他行插入 Categories2。

```
INSERT FIRST
    WHEN currlevel <= 2 THEN INTO Categories1
    ELSE INTO Categories2
SELECT * FROM Categories WHERE currlevel < 5;
```

命令执行后，查看 Categories1 表中的数据：

```
SELECT * FROM Categories1;
```

查询结果如下。

```
catgid |    catgname    | parentid | currlevel
-------+----------------+----------+-----------
     1 | 图书,音像      |     NULL |         1
     2 | 电子书刊       |        1 |         2
     7 | 音像           |        1 |         2
```

```
  11  | 英文原版          |      1  |      2
  18  | 文艺              |      1  |      2
(5 行记录)
```

查看 Categories2 表中的数据：

```
SELECT * FROM Categories2;
```

查询结果如下。

```
 catgid |    catgname    | parentid | currlevel
--------+----------------+----------+-----------
     3  | 电子书         |      2   |      3
    10  | 教育音像       |      7   |      3
    13  | 商务投资       |     11   |      3
    20  | 文学           |     18   |      3
    21  | 青春文学       |     20   |      4
(5 行记录)
```

5.1.2 INSERT ON CONFLICT

用类似 5.1.1 节所介绍的插入命令时，如果插入的数据违反了被更新表上的完整性约束时，数据库管理系统会拒绝执行相应的数据插入命令并给出错误提示。如果要插入行违反了主码约束或唯一性约束时，用户希望数据库管理系统不给出错误提示，将该数据插入操作改为数据更新操作，更新与要插入行冲突的已有行，就可以使用带 ON CONFLICT 子句的 INSERT 命令进行数据插入。这种方式对于嵌入在高级程序设计语言或用户自定义函数中的数据插入命令，既可以避免多线程之间的竞争问题，又可以减少代码量，并且会更高效。ON CONFLICT 子句基本语法为：

```
ON CONFLICT [ conflict_target ] conflict_action
```

conflict_target 可以是具有唯一约束或排除约束的属性(组)或约束名称，用来指明要插入的行中哪些约束违反时会执行 conflict_action 指定的替换动作。对于 ON CONFLICT DO NOTHING 来说，“conflict_target”可以省略。在被省略时，与所有有效约束(以及唯一索引)冲突的行都会被处理。对于 ON CONFLICT DO UPDATE，必须提供一个“conflict_target”。

“conflict _ action”指定一个可替换的 ON CONFLICT 动作。它可以是 DO NOTHING，也可以是 DO UPDATE 子句。UPDATE 中的 SET 和 WHERE 子句能够使用被插入表的名称(或者别名)访问表中已存在的行，并且可以用 EXCLUDED 临时表访问要插入的行。这个动作要求被排除列所在目标表的任何列上的 SELECT 特权。

例 5.11 假设 Adminaddrs 中已存在 addrid 为 1 的记录(1,‘阿富汗’, 1, 0)，现要插入一条 addrid 为 1 的记录(1,‘丹麦’, 2, 0)。

分析：因为 addrid 是 Adminaddrs 表的主码，addrid 取值不能重复，用基本的 INSERT

语句会出现错误，因此可以用下面的命令：

```
INSERT INTO Adminaddrs(addrid,addrname,parentid) VALUES(1, '丹麦', 2)
    ON CONFLICT(addrid) do update
      set addrname = EXCLUDED.addrname, parentid = EXCLUDED.parentid;
```

命令执行后查看表中数据：

```
SELECT * FROM Adminaddrs WHERE addrid=1;
```

查询结果如下。

```
 addrid |  addrname  | parentid | currlevel
 ------ +---------- +-------- +------------
    1   | 丹麦       |      2   |       0
(1 行记录)
```

可以看到该命令将 Adminaddrs 中 addrid 为 1 的记录的“阿富汗”更新为“丹麦”。

例 5.12　在 Adminaddrs 表中插入 addrid 为 19999 的记录。

分析：当不确定 Adminaddrs 中是否存在 addrid 为 19999 的记录时，可以用如下命令。

```
INSERT INTO Adminaddrs (addrid, addrname, parentid, currlevel)
  VALUES (19999, '西湖', 19999, 20000) ON CONFLICT do nothing;
```

说明：当表中不存在 addrid 为 19999 的记录时，会插入该记录；如果已存在，因为 conflict_action 是 DO NOTHING，所以也不会给出错误提示。

5.1.3　RETURNING 子句返回值

在用户自定义函数中经常会涉及对表的增、删、改操作，并且在对表操作后，需要获取操作行的某些列的值，或者获取操作行的所有数据，这时就可以在 INSERT、UPDATE、DELETE 命令中使用 RETURNING 或 RETURN 子句来获取这些值。

只能返回被成功地插入或者更新的行。可以在 RETURNING/RETURN 后列出需要返回值的部分列名或所有列名，也可以用 * 代替所有列名。使用 RETURNING 子句需要 RETURNING 中提到的所有列的 SELECT 权限。

例 5.13　在 INSERT 语句中仅返回插入行中一个属性列的值。

```
INSERT INTO Categories (catgid, catgname, parentid, currlevel)
  VALUES ( 106, '对讲机电池', 102, 3) ON CONFLICT(catgid)
  DO UPDATE SET catgname=EXCLUDED.catgname RETURNING catgid, catgname;
```

执行该插入命令前查询 Categories 表中 Categories＝106 的记录：

```
SELECT addrid, addrname, parentid FROM Categories=106;
```

查询结果如下。

```
catgid |    catgname    | parentid | currlevel
------ +--------------- +--------- +------------
  106  | 对讲机配件     |    102   |      3
(1 行记录)
```

执行该插入命令后查询的结果：

```
catgid |    catgname
------ +--------------------
  106  | 对讲机电池
 (1 行记录)
```

说明：用户需要具有 catgid，catgname 属性上的 SELECT 权限，才能成功执行该命令。

例 5.14 在 INSERT 语句中使用返回所有属性列的值。

```
INSERT INTO Adminaddrs(addrid, addrname, parentid) VALUES (3, '安道尔', 3)
  RETURNING * ;
```

执行插入命令后的查询结果：

```
 addrid | addrname | parentid
------- +--------- +-----------
     3  | 安道尔   |      3
(1 行记录)
```

说明：由于 Adminaddrs 表中没有 addrid＝3 的记录，该命令能成功执行返回插入行的所有值。当 Adminaddrs 表中有 addrid＝3 的记录时，该命令不能成功执行，不会返回任何值。

5.2 更新语句

5.2.1 UPDATE 更新语句

修改已经存储在数据库中的数据的行为叫作更新。在进行更新操作时，可以分别更新每个属性列而其他属性列不受影响；可以通过指定被更新的行必须满足的条件，来更新单行或多行的数据。UPDATE 语句的基本语法为：

```
UPDATE <表名> SET <列名> = <表达式>[,<列名> = <表达式>] [where <条件表达式>];
```

其功能是修改指定表中满足 WHERE 子句条件的行。其中，SET 子句给出＜表达式＞的值用来取代相应的列值。如果省略 WHERE 子句，则修改表中的所有行。

例 5.15 将表 Categories 中 catgname 为“电子书”的改为“四大名著”。

```
UPDATE Categories SET catgname = '四大名著' WHERE catgname = '电子书';
```

说明：Categories 中只有一行的 catgname ＝'电子书'，所以该命令只修改了一行数据。