触发器（Trigger） 是由事件自动触发执行的一种特殊的存储过程， 触发事件可以是对一个表进行INSERT、 UPDATE、 DELETE等操作。

触发器经常用于加强数据的完整性约束和业务规则上的约束等。

**创建触发器**

**语法格式：**

CREATE TRIGGER name { BEFORE | AFTER | INSTEAD OF } { event [ OR ... ] }

ON table\_name

[ FOR [ EACH ] { ROW | STATEMENT } ]

EXECUTE PROCEDURE function\_name

**创建触发器的步骤：**

1.先建一个函数， 此函数的返回类型为触发器类型。

2.然后建一个触发器。

**示例**

有一张学生表和成绩表，如果想要在删除学生表的同时删除该学生在成绩， 就可以使用触发器。

创建学生表和成绩表

CREATE TABLE student(id int primary key,name text,age int);

CREATE TABLE score (id int,chinese\_score int,math\_score int,test\_date date);

先创建函数：

CREATE OR REPLACE FUNCTION student\_delete\_function()

RETURNS TRIGGER

LANGUAGE plpgsql

AS $$

BEGIN

DELETE FROM score WHERE id = OLD.id;

RETURN OLD;

END $$ ;

再创建触发器：

CREATE TRIGGER delete\_student\_trigger

AFTER DELETE ON student

FOR EACH ROW EXECUTE PROCEDURE student\_delete\_function();

测试

INSERT INTO student VALUES(1, 'zhangsan', 18);

INSERT INTO student VALUES(2, 'lisi', 20);

INSERT INTO student VALUES(3, 'wangwu', 22);

INSERT INTO score VALUES(1, 85, 75, date '2022-05-20');

INSERT INTO score VALUES(1, 80, 73, date '2022-09-18');

INSERT INTO score VALUES(2, 68, 83, date '2022-05-20');

INSERT INTO score VALUES(2, 73, 85, date '2022-09-18');

INSERT INTO score VALUES(3, 72, 79, date '2022-05-20');

INSERT INTO score VALUES(3, 78, 82, date '2022-05-20');

删除id为3的学生：

DELETE FROM student where id = 3;

查询成绩表：

SELECT \* FROM score;--id为3的数据均被删除

**触发器函数中的特殊变量**

当把一个函数当作触发器函数调用时， 系统会在顶层声明段中自动创建几个特殊变量， 比如“NEW”“OLD”“TG\_OP”变量等。

NEW： 该变量为INSERT/UPDATE操作触发的行级触发器中存储新的数据行， 数据类型是“RECORD”。 在语句级别的触发器中此变量未分配， DELETE操作触发的行级触发器中此变量也未分配。

OLD： 该变量为UPDATE/DELETE操作触发的行级触发器中存储原有的数据行， 数据类型是“RECORD”。 在语句级别的触发器中此变量未分配， INSERT操作触发的行级触发器中此变量也未分配。

TG\_NAME： 数据类型是name类型， 该变量包含实际触发的触发器名。

TG\_WHEN： 内容为“BEFORE”或“AFTER”字符串用于指定是BEFORE触发器还是AFTER触发器。

TG\_LEVEL： 内容为“ROW”或“STATEMENT”字符串用于指定是语句级触发器还是行级触发器。

TG\_OP： 内容为“INSERT”“UPDATE”“DELETE”“TRUNCATE”之一的字符串， 用于指定DML语句的类型。

TG\_RELID： 触发器所在表的OID。

TG\_RELNAME： 触发器所在表的名称， 该变量即将废弃， 建议使用TG\_TABLE\_NAME变量来替换此变量。

TG\_TABLE\_NAME： 触发器所在表的名称。

TG\_TABLE\_SCHEMA： 触发器所在表的模式。

TG\_NARGS： CREATE TRIGGER语句中赋予触发器过程参数的个数。

TG\_ARGV[]： 为text类型的数组； 是CREATE TRIGGER语句中的参数。

**语句级触发器与行级触发器**

语句级触发器是指执行每个SQL语句时只触发一次， 行级触发器是改变多少行就会触发多少次。 一个修改0行的操作仍然会导致语句级触发器被执行。如果发出一条UPDATE影响 20 行的语句，则行级触发器将被调用 20 次，而语句级触发器将被调用 1 次。

**语句级触发器示例：**

假设要对表“student”的更新情况记录log， 可以为student表建一张log表：

CREATE TABLE log\_student(

update\_time timestamp, --操作的时间

db\_user varchar(40), --操作的数据库用户名

oper\_type varchar(6) --操作类型： insert、 delete、 update);

创建记录log的触发器函数：

CREATE FUNCTION log\_student\_function()

RETURNS trigger

LANGUAGE plpgsql

AS $$

BEGIN

INSERT INTO log\_student values(now(), user, TG\_OP);

RETURN NULL;

END $$ ;

上面函数中的“TG\_OP”是触发器函数中的特殊变量， 代表DML操作类型。

然后在表“student”上创建一个语句级触发器：

CREATE TRIGGER log\_student\_trigger

AFTER INSERT OR DELETE OR UPDATE ON student

FOR STATEMENT EXECUTE PROCEDURE log\_student\_function();

插入两条记：

INSERT INTO student VALUES(1, 'zhangsan', 22), (2, 'lisi', 18);

select \* from student;

select \* from log\_student;--记录一次操作

更新两行记录：

update student set age = 15;

select \* from log\_student;--录了一条记录

以上结果说明语句触发器是按语句进行触发的， 而不管这条语句实际操作了多少行数据。

执行更新， 但没有更新任何数据：

update student set age = 16 where id = 3;

select \* from log\_student;--录了一条记录

从上面结果中可以看出， 语句触发器即使没有更新数据，也会被触发。

**行级触发器示例：**

drop trigger log\_student\_trigger on student;

delete from log\_student;

delete from student;

创建行级触发器：

CREATE TRIGGER log\_student\_trigger2

AFTER INSERT OR DELETE OR UPDATE ON student

FOR ROW EXECUTE PROCEDURE log\_student\_function();

插入2条记录：

INSERT INTO student VALUES(1, 'zhangsan', 22), (2, 'lisi', 18);

select \* from log\_student;--记录两条日志

更新2条记录：

update student set age = 15;

select \* from log\_student;--记录两条日志

执行更新， 但没有更新任何数据：

update student set age = 16 where id = 3;

select \* from log\_student;--没有相应的记录

从上面的结果可以看出， 如果执行的语句没有更新实际的行， 即行触发器不会被触发。

**BEFORE触发器与AFTER触发器**

BEFORE触发器是在语句执行之前被触发，而AFTER触发器是在语句执行完成后才触发。

**BEFORE触发器示例：**

BEFORE触发器可以直接修改NEW值以改变实际的更新值。

先建一个触发器函数：

CREATE FUNCTION student\_use\_new\_name\_funciton()

RETURNS trigger

LANGUAGE plpgsql

AS $$

BEGIN

NEW.name = NEW.name||NEW.id;

RETURN NEW;

END $$ ;

上面函数的作用是， 在插入或更新表时， 把 name 后加上id，也就是直接修改 NEW.name， 语句如下：

NEW.name = NEW.name||NEW.id;

在这种情况下， 只能使用BEFORE触发器， 因为BEFORE触发器是在更新数据之前触发的， 所以这时修改了 NEW.name， 后面实际更新到数据库中的值就变成了 name||id， 示例如下：

CREATE TRIGGER user\_new\_name\_student\_trigger

BEFORE INSERT OR UPDATE ON student

FOR EACH ROW EXECUTE PROCEDURE student\_use\_new\_name\_function();

INSERT INTO student values(3,'wangwu',16);

SELECT \* FROM student;

如果使用AFTER触发器， 则修改NEW是没有意义的：

DROP TRIGGER user\_new\_name\_student\_trigger ON student;

CREATE TRIGGER user\_new\_name\_student\_trigger

AFTER INSERT OR UPDATE ON student

FOR EACH ROW EXECUTE PROCEDURE student\_use\_new\_name\_function();

INSERT INTO student values(3,'wangwu',16);

select \* from student;

**管理触发器**

1.重命名触发器

ALTER TRIGGER trigger\_name ON table\_name RENAME TO new\_trigger\_name;

2.禁用触发器

ALTER TABLE table\_name DISABLE TRIGGER trigger\_name | ALL

-- ALL禁用与该表关联的所有触发器

3.启用触发器

ALTER TABLE table\_name ENABLE TRIGGER trigger\_name | ALL;

4.删除触发器

注意：删除触发器时指定 ON table， 而其他数据库是直接 DROP TRIGGER name。

DROP TRIGGER user\_new\_name\_student\_trigger ON student;

删除触发器时， 触发器的函数不会被删除。 不过， 当删除表时， 表上的触发器也会被删除。

**触发器的行为**

触发器函数有返回值。 语句级触发器应该总是返回NULL， 即必须显式地在触发器函数中写上“RETURN NULL”， 如果没有写将导致报错：

CREATE FUNCTION log\_student\_function()

RETURNS trigger AS

$$

BEGIN

INSERT INTO log\_student values(now(), user, TG\_OP);

END;

$$

LANGUAGE "plpgsql";

CREATE TRIGGER log\_student\_trigger

AFTER INSERT OR DELETE OR UPDATE ON student

FOR STATEMENT EXECUTE PROCEDURE log\_student\_function();

执行报错：

INSERT INTO student VALUES(1, 'zhangsan',22);--ERROR

对于BEFORE触发器来说， 如果返回的是NULL， 则表示忽略对当前行的操作。 如果是返回非NULL的行， 对于INSERT和UPDATE操作来说， 返回的行将成为被插入的行或者是将要更新的行。

对于AFTER级触发器来说， 其返回值会被忽略。

如果同一事件上有多个触发器， 则将按触发器名称的顺序来触发。 如果是BEFORE触发器， 每个触发器返回的行（可能已经被修改） 将成为下一个触发器的输入。 如果BEFORE触发器返回的内容为空， 那么该行上的其他行级触发器也不会被触发。