事件触发器(Event Trigger)， 这种触发器实际上就是DDL触发器。 目前， 事件触发器支持以下3种DDL事件。

ddl\_command\_start： DDL开始执行前被触发。

ddl\_command\_end： 一个DDL执行完成后被触发。

sql\_drop： 删除数据库对象前被触发。

由于事件触发器涉及的权限较大， 比如禁止DDL操作等， 所以只有超级用户才能创建和修改事件触发器。

**创建事件触发器**

**语法格式：**

CREATE EVENT TRIGGER name

ON event

EXECUTE PROCEDURE function\_name()

在创建事件触发器之前， 必须先创建触发器函数， 事件触发器函数的返回类型为event\_trigger， 注意， 其与普通触发器函数的返回类型（trigger） 是不一样的。

**示例1**

禁止所有DDL语句：

CREATE OR REPLACE FUNCTION abort\_any\_command()

RETURNS event\_trigger

LANGUAGE plpgsql

AS $$

BEGIN

RAISE EXCEPTION 'command % is disabled', tg\_tag;

END $$;;

CREATE EVENT TRIGGER abort\_ddl ON DDL\_command\_start

EXECUTE PROCEDURE abort\_any\_command();

执行DDL语句：

drop table t1;--ERROR

create table t2(id int);--ERROR

truncate table t1; --SUCCESS

注意：truncate语句执行成功。 因为在PostgreSQL中truncate事件是使用普通触发器触发的， 事件触发器是不会触发truncate的。

另外， 事件触发器本身的操作是不会再触发事件触发器的， 否则禁止DDL语句后，数据库就无法再执行DDL操作了， 这显然是不行的。

**特殊变量**

事件触发器函数中仅仅支持TG\_EVENT和TG\_TAG变量。

TG\_EVENT： 为“ddl\_command\_start”“ddl\_command\_end”“sql\_drop”之一。

TG\_TAG： 指具体的哪种DDL操作， 如“CREATE TABLE”“DROP TABLE”等。

查询事件触发器：

select \* from pg\_event\_trigger ;

**示例2：**

上面示例做DDL操作时还很难分清是操作了哪个数据库对象。

不过， 对于sql\_drop事件， 可以调用函数 pg\_event\_trigger\_dropped\_objects() 来获得删除数据库对象的信息， 该函数会返回一个结果集。

|  |  |  |
| --- | --- | --- |
| **名称** | **类型** | **描述** |
| classid | oid | 对象所属的目录的 OID |
| objid | oid | 对象本身的 OID |
| objsubid | integer | 对象的子-id（例如列的属性号） |
| command\_tag | text | 命令标签 |
| object\_type | text | 对象的类型 |
| schema\_name | text | 该对象所属的模式的名称（如果有），如果没有则为 NULL。 没有引号。 |
| object\_identity | text | 对象标识的文本表现形式，用模式限定。如果必要，出现在 该标识中的每一个标识符都会被引用。 |
| in\_extension | boolean | 如果该命令是一个扩展脚本的一部分则为真 |
| command | pg\_ddl\_command | 以内部格式表达的该命令的一个完整表现形式。这不能被直接输出， 但是可以把它传递给其他函数来得到有关于该命令不同部分的信息。 |

记录表、 索引、 视图等所有数据库对象删除操作的审计日志：

CREATE TABLE log\_drop\_objects(

op\_time timestamp,

ddl\_tag text,

classid oid,

objid oid,

objsubid oid,

object\_type text,

schema\_name text,

object\_name text,

object\_identity text

);

CREATE FUNCTION event\_trigger\_log\_drops()

RETURNS event\_trigger

LANGUAGE plpgsql

AS $$

BEGIN

INSERT INTO log\_drop\_objects SELECT now(), tg\_tag, classid,objid,objsubid,object\_type,schema\_name,object\_name, object\_identity FROM pg\_event\_trigger\_dropped\_objects();

END $$;

CREATE EVENT TRIGGER event\_trigger\_log\_drops ON sql\_drop

EXECUTE PROCEDURE event\_trigger\_log\_drops();

创建一张表：

create table t1(id int primary key, name text,age int);

删除一个字段：

alter table t1 drop column age;

select ddl\_tag, object\_type, object\_name, object\_identity from log\_drop\_objects;--记录了删除字段的操作。

删除表：

drop table t1;

select ddl\_tag, object\_type, object\_name, object\_identity from log\_drop\_objects;

从上面的结果中可以看到， 删除该表后， 数据库实际还删除了表上的主键约束、主键索引及和与表名相同的类型等内容。

**触发器管理**

**修改事件触发器**

语法格式：

ALTER EVENT TRIGGER name DISABLE

ALTER EVENT TRIGGER name ENABLE

ALTER EVENT TRIGGER name OWNER TO new\_owner

ALTER EVENT TRIGGER name RENAME TO new\_name

**删除事件触发器**

DROP EVENT TRIGGER [ IF EXISTS ] name [ CASCADE | RESTRICT ]

DROP EVENT TRIGGER event\_trigger\_log\_drops;