

Триггеры



PostgreSQL

Введение

- Бывает необходимо автоматически «реагировать» на определенные ожидаемые события SQL
- Это может быть необходимость:
 - проверки согласованности набора вставляемых значений
 - форматирования предоставленных данных перед их вставкой
 - изменения записей отдельной таблицы после удаления или изменения набора строк в другой таблице
- Для возможности обработки на стороне сервера PostgreSQL поддерживает нестандартное программное расширение – триггер
 - Позволяет в случае работы с БД различных приложений сохранить кросс-функциональность в базе данных

Триггеры и триггерные функции

- **Триггер** – это именованный блок PL/pgSQL
 - отслеживает заданные события
 - вызывает триггерную функцию для обработки события
 - передает в функцию информацию о событии (**контекст**)
- **Триггерная функция** – содержит команды, которые должны быть выполнены при наступлении заданного события
 - выполняется в той же транзакции, что и основная операция (событие)
- Триггер и триггерная функция хранятся в БД как отдельные объекты
 - Одна триггерная функция может вызываться разными триггерами



Использование триггеров

- Безопасность
 - вызвать ошибку при совершении подозрительных действий
 - ограничить диапазон DDL-команд, доступных пользователю
- Реализация сложных ограничений целостности
 - поддержка сложных правил целостности данных, которые невозможно реализовать где-либо еще, кроме как на уровне базы данных
- Автоматическое вычисление связанных данных
 - поддержка вычисляемых полей на основе значений из других таблиц
- Репликация таблиц
- Аудит и логгирование событий
- **ВАЖНО:**
 - вы должны знать, что триггер существует, и понимать его логику, чтобы понять последствия изменения данных



Типы триггеров

- **DML-триггеры**

- Выполнение DML-команд заданного типа для таблицы или представления
 - INSERT, UPDATE, DELETE
 - TRUNCATE
- **Событие извлечения данных (SELECT) триггером не обрабатывается!**

- **Триггеры событий**

- Выполнение DDL- или DCL-команд
 - CREATE, ALTER, DROP, COMMENT, GRANT, REVOKE
 - SELECT INTO (в значении CREATE TABLE AS)

Создание триггера

- Общий план
 - Создать триггерную функцию
 - Создать вызывающий ее триггер
- Триггер содержит «организационную» информацию:
 - имя триггера
 - имя триггерной функции
 - имя таблицы (представления), с которой связан триггер
 - событие, в ответ на которое должна быть вызвана триггерная функция
 - время выполнения: до или после события
 - уровень: вызвать один раз или для каждой затронутой строки таблицы
 - дополнительное условие вызова (WHEN), позволяющее сузить область действия триггера – триггера срабатывает только если выполняется условие
- Для создания триггера пользователь должен иметь:
 - право **TRIGGER** для таблицы/представления
 - право **EXECUTE** для триггерной функции

<https://postgrespro.ru/docs/postgresql/13/sql-createtrigger>

Управление триггерами

- Отключение / повторное включение **триггера событий**:

```
ALTER EVENT TRIGGER имя_триггера  
{DISABLE | ENABLE}
```

- Отключение / повторное включение **DML-триггера**:

```
ALTER TABLE имя_таблицы  
{DISABLE | ENABLE} TRIGGER имя_триггера | ALL
```

- Удаление триггера

```
DROP [EVENT] TRIGGER [IF EXISTS] имя_триггера  
[ON {имя_таблицы | имя_представления}] [CASCADE];
```

- При удалении таблицы (DROP TABLE) с ключевым словом **CASCADE** все связанные с ней триггеры удаляются автоматически

Просмотр информации о триггерах

- Для получения информации о триггерах используется запрос к системному представлению `information_schema.triggers`

```
SELECT trigger_name,  
       action_timing "when",  
       string_agg(event_manipulation, ',') what,  
       event_object_table "on table/view",  
       action_orientation "level",  
       action_condition "extra filter",  
       action_statement "trigger function"  
FROM information_schema.triggers  
GROUP BY trigger_name, action_timing, event_object_table,  
         action_orientation, action_condition, action_statement  
ORDER BY event_object_table;
```


Замечания

- Порядок срабатывания однотипных триггеров задать нельзя
 - Вызываются по порядку сортировки их имён
- Не рекомендуется реализовывать сложную логику исключительно на основе триггеров
 - Код вызывается неявно, поэтому сложно отследить логику выполнения
 - Усложняется поддержка приложения, т.к. сложно восстановить цепочку событий
- Не следует создавать безусловные рекурсивные триггеры
 - Триггер может вызвать срабатывание других триггеров
- **Триггеры событий не срабатывают в однопользовательском режиме**
 - Эта особенность используется для удаления (отключения) блокирующих работу триггеров, которые невозможно удалить другим способом

DML-триггеры

Базовый синтаксис DML-триггера

```
CREATE TRIGGER имя_триггера  
{ BEFORE | AFTER | INSTEAD OF } событие [OR событие]...  
ON { имя_таблицы | имя_представления }  
[FOR [EACH] { ROW | STATEMENT }]  
[WHEN (условие)]  
EXECUTE { FUNCTION | PROCEDURE }  
имя_функции( );
```

- **Время выполнения**

- **BEFORE** – до выполнения вызвавшей команды и проверки ограничений целостности
- **AFTER** – после выполнения вызвавшей команды и проверки ограничений целостности
- **INSTEAD OF** – вместо выполнения команды (только для представлений)

- **Событие**

- **INSERT, DELETE, UPDATE** [OF имя_столбца[, имя_столбца]...], **TRUNCATE**
- комбинация команд, например: **AFTER INSERT OR UPDATE OF** total_amount

- **Уровень**

- **Команды (FOR EACH STATEMENT)** – выполняется для команды в целом, даже если не было затронуто ни одной строки
 - единственный доступный уровень для события **TRUNCATE**
- **Строки (FOR EACH ROW)** – выполняется для каждой отдельной строки, затрагиваемой командой

DML-триггеры на представления

- Триггер **INSTEAD OF** и предназначены для работы с не обновляемыми представлениями:
 - имеет уровень строки - **FOR EACH ROW**
 - **не поддерживает раздел WHEN**
- Триггеры **BEFORE** и **AFTER**
 - имеют уровень оператора - **FOR EACH STATEMENT**
 - срабатывают, только если операция с представлением обрабатывается триггером уровня строк **INSTEAD OF**
 - вместо внесения изменений в представление вносят их в базовые таблицы
- Для автоматически изменяемого представления выполнение операции сводится к переписыванию оператора в виде операции с базовой таблицей
 - **срабатывать будут триггеры уровня операторов для базовой таблицы!**



Условие WHEN

- Условие **WHEN** определяет логическое выражение, определяющее, будет ли выполняться функция триггера:
 - если для триггера задано указание **WHEN**, функция будет вызываться, только когда условие возвращает **true**
- В триггерах **FOR EACH ROW** условие **WHEN** может ссылаться на значения столбца:
 - **OLD.имя_столбца** – ссылка на столбец в старой версии строки
 - **NEW.имя_столбца** – ссылка на столбец новой версии строки
- **Ограничения:**
 - выражения **WHEN** не могут содержать подзапросы
 - триггеры **INSTEAD OF** не поддерживают условия **WHEN**
 - в триггерах **FOR EACH STATEMENT** нельзя ссылаться на значения столбцов



Сводка характеристик

INSERT, UPDATE, DELETE

Таблицы, внешние таблицы	before/after	statement
	before/ after	row
Представления	before/after	statement
	instead of	row
TRUNCATE		
Таблицы	before/after	statement

- Триггер **AFTER ROW** создает очередь из событий (для каждой записи), которые обрабатываются после окончания операции
- Условие **WHEN** позволяет отфильтровать ненужные строки:
 - Уменьшается число событий в очереди
 - Снижаются накладные расходы

Последовательность срабатывания

```
UPDATE "Production"."Products"  
SET unitprice = unitprice * 0.95  
WHERE categoryid = 5;
```

	productid	productname	supplierid	categoryid	unitprice	discontinued
1	22	Product CPHFY	9	5	\$21.00	0
2	23	Product JLUDZ	9	5	\$9.00	0
3	42	Product RJVNM	20	5	\$14.00	1
4	52	Product QSRXF	24	5	\$7.00	0
5	56	Product VKCMF	26	5	\$38.00	0
6	57	Product OVLQI	26	5	\$19.50	0
7	64	Product HCQDE	12	5	\$33.25	0

→ Триггер **BEFORE STATEMENT**
(1 раз до начала выполнения основной операции)

→ Триггер **BEFORE (INSTEAD OF) ROW**
(в процессе выполнения операции над текущей строкой)

→ Триггер **AFTER ROW**
(для каждой строки, но после выполнения всей операции - **очереди**)

→ Триггер **AFTER STATEMENT**
(1 раз после выполнения операции и отработки AFTER ROW триггеров)

Триггер INSTEAD OF ROW срабатывают не ДО, а ВМЕСТО DML-операции над представлением!

Сценарии использования

- **BEFORE STATEMENT**

- проверка состояния данных и применимости операции
- генерация ошибки при необходимости прервать операцию

- **BEFORE ROW**

- проверка корректности данных
- модификация строки (заполнение недостающих данных, изменение некорректных данных и т.д.)

- **INSTEAD OF ROW**

- изменение базовых таблиц через представления

- **AFTER ROW / AFTER STATEMENT**

- проверка согласованности, в том числе на уровне таблицы
- «аудит» операций (ведение журнала изменений)
- каскадное изменение таблиц (денормализация, асинхронная обработка и т.д.)

Триггерная функция



PostgreSQL

Характеристики

- Может быть написана на одном из подключаемых процедурных языков или на чистом C
- Должна быть определена без параметров
- Тип (псевдотип) возвращаемого значения (RETURNS)
 - **TRIGGER** – для DML-триггеров
 - **EVENT_TRIGGER** – для триггеров событий
- При вызове из триггера ей передаётся текущий контекст (**TriggerData**)
 - какое событие произошло
 - с каким объектом связано и т.п.
- Выполняется в той же транзакции, что и событие, вызвавшее срабатывание триггера
 - Необработанная ошибка отменяет результаты и триггерной функции, и вызвавшей её срабатывание команды



Структура TriggerData

- Используется для хранения **контекста вызова**
- Переменные для DML-триггеров:
 - **OLD** и **NEW** – записи (RECORD), содержащие сведения о, соответственно, исходном и новом состоянии изменяемой строки
 - **используются только в триггерах уровня строки**
 - **TG_OP** – событие ('INSERT' | 'UPDATE' | 'DELETE')
 - **TG_WHEN** – время выполнения (' BEFORE ' | ' AFTER ' | ' INSTEAD OF ')
 - **TG_LEVEL** – уровень ('ROW' | 'STATEMENT')
 - **TG_TABLE_NAME** – имя таблицы, для которой сработал триггер
- Переменные для триггеров событий
 - **TG_EVENT** – событие, для которого сработал триггер
 - **TG_TAG** – тег команды, для которой сработал триггер



Возвращаемое значение триггерной функции

- Триггер **BEFORE STATEMENT**
 - Возвращаемое значение триггерной функции игнорируется (можно возвращать NULL)
 - Если в триггере возникает ошибка, операция отменяется
- Триггер **BEFORE ROW / INSTEAD OF ROW**
 - Возврат значения **NULL** – отмена DML-команды и других триггеров для текущей строки (не отменяет продолжения выполнения операции)
 - Возврат **NEW** при вставке или обновлении – стандартное выполнение операции
 - Возврат **OLD** при удалении – стандартное выполнение операции
 - **Чтобы повлиять на результат операции триггерная функция может изменить значение NEW**
- Триггер **AFTER ROW / AFTER STATEMENT**
 - **Возвращаемое значение триггерной функции игнорируется (можно возвращать NULL)**



Пример использования возвращаемого значения

```
create table t1 (id int);  
create table t2 (id int);
```

```
CREATE or replace FUNCTION trigger_function()  
  RETURNS TRIGGER  
  LANGUAGE PLPGSQL AS $$  
BEGIN  
  if new.id is null then  
    raise 'Передано значение %' , new.id;  
  else  
    insert into t2(id)  
      values(new.id);  
  end if;  
  RETURN NULL;  
END;$$
```

```
CREATE TRIGGER last name changes  
  BEFORE INSERT OR UPDATE  
  ON t1  
  FOR EACH ROW  
  EXECUTE PROCEDURE trigger_function();
```

```
insert into t1(id) values (NULL);
```

```
insert into t1(id) values (5),(6),(null);
```



SQL Error [P0001]: ERROR: Передано значение <NULL>
Где: PL/pgSQL function trigger_function() line 4 at RAISE

```
insert into t1(id) values (1),(2);
```

```
select * from t1;
```

select * from t1	
id	
1	
2	

```
select * from t2;
```

select * from t2	
id	
1	1
2	2

```
Update t1  
set id = 5  
where id = 1;
```

?

Переходные таблицы

- Переходные таблицы содержат только те строки, которые были затронуты операцией:
 - не присутствуют в системном каталоге
 - располагаются в оперативной памяти
 - при большом объеме могут сбрасываться на диск
 - **доступны только внутри триггерной функции**
- Для создания переходных таблиц необходимо при создании триггера:
 - задать их имена (псевдонимы) в секции **REFERENCING**
 - **NEW TABLE** – новые записи (INSERT, UPDATE)
 - **OLD TABLE** – старые записи (UPDATE, DELETE)

Использование переходных таблиц удорожает обработку!

```
CREATE TRIGGER имя_триггера
AFTER UPDATE ON имя_таблицы -- только одно событие на один триггер
REFERENCING
    OLD TABLE AS имя_переходной_таблицы_old
    NEW TABLE AS имя_переходной_таблицы_new
FOR EACH STATEMENT
EXECUTE FUNCTION имя_функции();
```



Встроенные триггерные функции

- В PostgreSQL имеется набор встроенных триггерных функций, которые можно использовать непосредственно в пользовательских триггерах
- **suppress_redundant_updates_trigger ()** - применяется в качестве функции для триггера **BEFORE UPDATE** на уровне строк
 - предотвращает внесение изменений, при которых данные в строке фактически не меняются
 - имя триггера, вызывающего данную функцию, по порядку сортировки должно быть последним среди имён всех триггеров, которые могут быть в таблице, чтобы он не перекрыл другие триггеры, которым может понадобиться изменить строку

Примеры DML-триггеров



Триггер уровня команды

```
CREATE OR REPLACE FUNCTION "Production".restrict_action()  
RETURNS TRIGGER  
LANGUAGE plpgsql AS $$  
BEGIN  
    PERFORM rolname  
    FROM pg_roles  
    WHERE oid IN ( SELECT member FROM pg_auth_members  
                   WHERE roleid = (SELECT oid FROM pg_roles WHERE rolname = 'manager') )  
                   AND rolname = current_user;  
    IF NOT FOUND THEN  
        RAISE EXCEPTION 'Только менеджеры могут % цену товаров активного каталога!', LOWER(TG_OP);  
    END IF;  
    RETURN NULL;  
END; $$
```

```
CREATE TRIGGER price update  
BEFORE UPDATE OF unitprice ON "Production"."Products"  
EXECUTE FUNCTION "Production".restrict_action();
```

Проверка

```
UPDATE "Production"."Products"  
SET unitprice = unitprice * 0.15  
WHERE discontinued = 0::bit;
```



SQL Error [P0001]: ERROR: Только менеджеры могут update цену товаров активного каталога!

Где: PL/pgSQL function
"Production".restrict_action() line 8 at RAISE



Триггер уровня строки - функция

```
CREATE OR REPLACE FUNCTION "Production".restrict_action ()
RETURNS TRIGGER
LANGUAGE plpgsql AS $$
BEGIN
    IF TG_LEVEL = 'STATEMENT' THEN
        --...
        RAISE NOTICE 'STATEMENT SUCCEED';
    ELSE
        RAISE NOTICE 'User % tried to % row in table %',current_user,LOWER(TG_OP),TG_TABLE_NAME;
        RAISE NOTICE 'Old value: %', OLD;
        RAISE NOTICE 'New value: %', NEW;
        IF TG_OP = 'UPDATE' AND NEW.unitprice != OLD.unitprice
            AND NEW.discontinued = 1::bit THEN
            RAISE NOTICE 'FAILED';
            RETURN NULL; --возможный вариант: RETURN OLD;
        END IF;
        RAISE NOTICE 'SUCCEED';
        RETURN NEW;
    END IF;
END; $$
```



Триггер уровня строки - триггер и команды проверки

```
CREATE TRIGGER price change
BEFORE INSERT OR UPDATE OR DELETE ON "Production"."Products"
FOR EACH ROW
EXECUTE FUNCTION "Production".restrict_action();
```

```
UPDATE "Production"."Products" SET unitprice = unitprice * 0.15
WHERE productid = 5;
```

```
User postgres tried to update row in table Products
Old value: (5,"Product EPEIM",2,2,$21.35,1)
New value: (5,"Product EPEIM",2,2,$3.20,1)
FAILED
```

```
UPDATE "Production"."Products" SET unitprice = 0.1
WHERE unitprice = 0::money;
```

```
User postgres tried to update row in table Products
Old value: (3,"Product IMEHJ",1,2,$0.00,0)
New value: (3,"Product IMEHJ",1,2,$0.10,0)
SUCCEED
User postgres tried to update row in table Products
Old value: (13,"Product POXFU",6,8,$0.00,0)
New value: (13,"Product POXFU",6,8,$0.10,0)
SUCCEED
User postgres tried to update row in table Products
Old value: (19,"Product XKXDO",8,3,$0.00,0)
New value: (19,"Product XKXDO",8,3,$0.10,0)
SUCCEED
```



Триггер INSTEAD OF - функция

```
CREATE OR REPLACE FUNCTION "Production".update_product_name()  
RETURNS trigger  
LANGUAGE plpgsql AS $$  
DECLARE  
    v_new_name "Production".v_products_count.productname%TYPE;  
BEGIN  
    IF OLD.productname != NEW.productname and  
       OLD.discontinued = 1::bit and new.discontinued = 1::bit THEN  
        UPDATE "Production"."Products"  
        SET productname = NEW.productname  
        WHERE productid = NEW.productid  
        RETURNING productname INTO v_new_name;  
        RAISE NOTICE 'Было: %. Стало: %', OLD.productname, v_new_name;  
    END IF;  
    RETURN NULL;  
END $$;
```

Триггер INSTEAD OF - триггер и команды проверки

```
CREATE TRIGGER update_product_name_trg
INSTEAD OF UPDATE ON "Production".v_products_count
FOR EACH ROW
EXECUTE FUNCTION "Production".update_product_name();
```

```
create view "Production".v_products_count
as
select P.productid, productname, discontinued, count(*) as products_count
from "Production"."Products" as p join "Sales"."OrderDetails" as od
on p.productid = od.productid
group by P.productid, productname, discontinued;
```

```
UPDATE "Production".v_products_count
SET productname = 'Апельсин'
WHERE productid= 5;
```

Было: Orange. Стало: Апельсин

Триггеры событий

Синтаксис триггера событий

```
CREATE EVENT TRIGGER имя_триггера  
ON событие  
[WHEN переменная_фильтра IN (значение[, значение]...)  
[AND ...]]  
EXECUTE {FUNCTION | PROCEDURE}  
имя_функции()
```

- Триггерная функция выполняется, когда происходит указанное событие и удовлетворяется связанное с триггером условие **WHEN** (если задано)



Характеристики триггеров событий

- **Событие**

- **DDL_COMMAND_START** – перед выполнением любой допустимой команды (см. слайд 5)
- **DDL_COMMAND_END** – после выполнения любой допустимой команды (см. слайд 5)
- **TABLE_REWRITE** – перед перезаписью таблицы
 - например, в результате добавления столбца с вычисляемым значением по умолчанию или изменения типа данных столбца на более короткий
- **SQL_DROP** – после удаления объектов

- **Уровень**

- Команды (statement)



Получение доп. информации в триггерах событий

- Функция **pg_event_trigger_ddl_commands**
 - Событие `ddl_command_end`
 - Пример возвращаемых данных: `command_tag`, `object_type`
- Функция **pg_event_trigger_dropped_objects**
 - Событие `sql_drop`
 - Пример возвращаемых данных: `object_type`, `object_name`
- Функции **pg_event_trigger_table_rewrite_oid** и **pg_event_trigger_table_rewrite_reason**
 - Событие `table_rewrite`
 - Возвращают OID перезаписываемой таблицы и код причины перезаписи соответственно

<https://postgrespro.ru/docs/postgresql/13/functions-event-triggers>

Пример DDL-триггера

- Вывод информации о выполненной DDL-команде

```
CREATE OR REPLACE FUNCTION report_ddl()  
RETURNS event_trigger LANGUAGE plpgsql AS $$  
DECLARE  
    r_command_desc record;  
BEGIN  
    FOR r_command_desc IN SELECT * FROM pg_event_trigger_ddl_commands() LOOP  
        RAISE NOTICE '%. тип: %, OID: %, имя: % ',  
            r_command_desc.command_tag, r_command_desc.object_type,  
            r_command_desc.objid, r_command_desc.object_identity;  
    END LOOP;  
END $$
```

```
CREATE EVENT TRIGGER after_ddl ON ddl_command_end EXECUTE FUNCTION  
report_ddl();
```

```
CREATE TABLE test_trg_tab (idcol int PRIMARY KEY);
```

```
CREATE TABLE. тип: table, OID: 172266, имя: public.test_trg_tab  
CREATE INDEX. тип: index, OID: 172269, имя: public.test_trg_tab_pkey
```



PostgreSQL

Дополнительные материалы

- Учебные материалы

<https://www.postgresqltutorial.com/postgresql-plpgsql/>

- Пример использования триггеров в PostgreSQL

<https://eax.me/postgresql-triggers/>