# Тема №7 - Триггеры.

**Триггер** *–* это специальный тип хранимой процедуры, которая запускается автома­тически системой SQL Server при модифицировании какой-либо таблицы одним из трех операторов: UPDATE, INSERT или DELETE. Триггеры, как другие хранимые процедуры, могут содержать простые или сложные операторы SQL. Триггер определяет операцию, которая должна выполняться при наступлении некоторого события в базе данных. Триггеры срабатывают при выполнении с таблицей команды SQL INSERT, UPDATE или DELETE.

В PostgreSQL триггеры создаются на основе существующих функции, т.е. сначала командой CREATE FUNCTION определяется триггерная функция, затем на ее основе командой CREATE TRIGGER определяется собственно триггер. Триггер будет связан с указанной таблицей, представлением или сторонней таблицей и будет выполнять заданную функцию *имя\_функции* при определённых событиях.

**Синтаксис объявления триггера:**

CREATE [ CONSTRAINT ] TRIGGER имя { BEFORE | AFTER | INSTEAD OF } { событие [ OR ... ] }

ON имя\_таблицы

[ FROM ссылающаяся\_таблица ]

[ NOT DEFERRABLE | [ DEFERRABLE ] [ INITIALLY IMMEDIATE | INITIALLY DEFERRED ] ]

[ FOR [ EACH ] { ROW | STATEMENT } ]

[ WHEN ( условие ) ]

EXECUTE PROCEDURE имя\_функции ( аргументы )

Здесь допускается событие:

INSERT

UPDATE [ OF имя\_столбца [, ... ] ]

DELETE

**CREATE TRIGGER** *триггер*. В аргументе *триггер* указывается произвольное имя создаваемого триггера. Имя может совпадать с именем триггера, уже существующего в базе данных при условии, что этот триггер установлен для другой таблицы. Кроме того, по аналогии с большинством других несистемных объектов баз данных, имя триггера (в сочетании с таблицей, для которой он устанавливается) должно быть уникальным лишь в контексте базы данных, в которой он создается.

**{BEFORE | AFTER}.** Ключевое слово BEFORE означает, что функция должна выполняться передпопыткой выполнения операции, включая все встроенные проверки ограничений данных, реализуемые при выполнении команд INSERT и DELETE. Ключевое слово AFTER означает, что функция вызывается после завершения операции, приводящей в действие триггер.

**{событие[OR *событие ...*] }.** События SQL, поддерживаемые в PostgreSQL: INSERT, UPDATE или DELETE. При перечислении нескольких событий в качестве разделителя используется ключевое слово OR.

**ON *таблица.***Имя таблицы, модификация которой заданным событием приводит к срабатыванию триггера.

**FOR EACH { ROW | STATEMENT }.** Ключевое слово, следующее за конструкцией FOR EACH и определяющее количество вызовов функции при наступлении указанного события. Ключевое слово ROW означает, что функция вызывается *для каждой  модифицируемой записи.*Если функция должна вызываться всего один раз для всей команды, используется ключевое слово STATEMENT.

**EXECUTE PROCEDURE *функция*(аргументы*)*.** Имя вызываемой функции с аргументами. На практике аргументы при вызове триггерных функций не используются.

**Синтаксис определения триггерной функции:**

CREATE FUNCTION **функция** () RETURNS **trigger** AS $$

DECLARE

**объявления**;

BEGIN

**команды**;

END;

$$ LANGUAGE plpgsql;

В триггерных функциях используются специальные переменные, содержащие информацию о сработавшем триггере. С помощью этих переменных триггерная функция работает с данными таблиц.

Когда функция на PL/pgSQL срабатывает как триггер, в блоке верхнего уровня автоматически создаются несколько специальных переменных:

**NEW**

Тип данных RECORD. Переменная содержит новую строку базы данных для команд INSERT/UPDATE в триггерах уровня строки. В триггерах уровня оператора и для команды DELETE эта переменная имеет значение null.

**OLD**

Тип данных RECORD. Переменная содержит старую строку базы данных для команд UPDATE/DELETE в триггерах уровня строки. В триггерах уровня оператора и для команды INSERT эта переменная имеет значение null.

**TG\_NAME**

Тип данных name. Переменная содержит имя сработавшего триггера.

**TG\_WHEN**

Тип данных text. Строка, содержащая BEFORE, AFTER или INSTEAD OF, в зависимости от определения триггера.

**TG\_LEVEL**

Тип данных text. Строка, содержащая ROW или STATEMENT, в зависимости от определения триггера.

**TG\_OP**

Тип данных text. Строка, содержащая INSERT, UPDATE, DELETE или TRUNCATE, в зависимости от того, для какой операции сработал триггер.

**TG\_RELID**

Тип данных oid. OID таблицы, для которой сработал триггер.

**TG\_RELNAME**

Тип данных name. Имя таблицы, для которой сработал триггер. Эта переменная устарела и может стать недоступной в будущих релизах. Вместо неё нужно использовать TG\_TABLE\_NAME.

**TG\_TABLE\_NAME**

Тип данных name. Имя таблицы, для которой сработал триггер.

**TG\_TABLE\_SCHEMA**

Тип данных name. Имя схемы, содержащей таблицу, для которой сработал триггер.

**TG\_NARGS**

Тип данных integer. Число аргументов в команде CREATE TRIGGER, которые передаются в триггерную функцию.

**TG\_ARGV[]**

Тип данных массив text. Аргументы от оператора CREATE TRIGGER. Индекс массива начинается с 0. Для недопустимых значений индекса ( < 0 или >= tg\_nargs) возвращается NULL.

Триггерная функция должна вернуть либо NULL, либо запись/строку, соответствующую структуре таблице, для которой сработал триггер.

Если BEFORE триггер уровня строки возвращает NULL, то все дальнейшие действия с этой строкой прекращаются (т. е. не срабатывают последующие триггеры, команда INSERT/UPDATE/DELETE для этой строки не выполняется). Если возвращается не NULL, то дальнейшая обработка продолжается именно с этой строкой. Возвращение строки отличной от начальной NEW, изменяет строку, которая будет вставлена или изменена. Поэтому, если в триггерной функции нужно выполнить некоторые действия и не менять саму строку, то нужно возвратить переменную NEW (или её эквивалент). Для того чтобы изменить сохраняемую строку, можно поменять отдельные значения в переменной NEW и затем её вернуть. Либо создать и вернуть полностью новую переменную. В случае строчного триггера BEFORE для команды DELETE само возвращаемое значение не имеет прямого эффекта, но оно должно быть отличным от NULL, чтобы не прерывать обработку строки. Обратите внимание, что переменная NEW всегда NULL в триггерах на DELETE, поэтому возвращать её не имеет смысла. Традиционной идиомой для триггеров DELETE является возврат переменной OLD.

Триггеры INSTEAD OF (это всегда триггеры уровня строк, и они могут применяться только с представлениями) могут возвращать NULL, чтобы показать, что они не выполняли никаких изменений, так что обработку этой строки можно не продолжать (то есть, не вызывать последующие триггеры и не считать строку в числе обработанных строк для окружающих команд INSERT/UPDATE/DELETE). В противном случае должно быть возвращено значение, отличное от NULL, показывающее, что триггер выполнил запрошенную операцию. Для операций INSERT и UPDATE возвращаемым значением должно быть NEW, которое триггерная функция может модифицировать для поддержки предложений INSERT RETURNING и UPDATE RETURNING (это также повлияет на значение строки, передаваемое последующим триггерам, или доступное под специальным псевдонимом EXCLUDED в операторе INSERT с предложением ON CONFLICT DO UPDATE). Для операций DELETE возвращаемым значением должно быть OLD.

Возвращаемое значение для строчного триггера AFTER и триггеров уровня оператора (BEFORE или AFTER) всегда игнорируется. Это может быть и NULL. Однако, в этих триггерах по-прежнему можно прервать вызвавшую их команду, для этого нужно явно вызвать ошибку.

Приведем пример создания триггера на примере вставки для таблицы users. Необходим триггер, который будет срабатывать на операцию добавления фамилии и имени коммерческого студента. В его задачу будет входить добавление аналогичных записей в таблицу с общим списком студентов:

CREATE TRIGGER users\_insert

ON Student\_c

FOR INSERT

AS

INSERT INTO Student (S\_lastname, S\_firstname)

VALUES (Inserted.Lastname, Inserted.Firstname)

CREATE TRIGGER users\_insert

AFTER INSERT

ON users

FOR EACH ROW

EXECUTE PROCEDURE update\_products\_amount();

Когда срабатывает триггер на удаление (по команде DELETE), удаленные записи размещаются в специальной таблицеdeleted. Необходимо учитывать следующие особенности триггера на удаление:

* Таблица deleted и (результирующая) таблица базы данных не имеют общих записей.
* Когда запись добавляется к deleted, она больше не существует в основной таблице.
* Основная таблица размещается в пространстве, зарезервированном для текущей базы данных.
* Таблицаdeleted всегда хранится в кэше.
* Таблицы deleted и inserted — логические. Структурно они всегда похожи на таблицы, для которых определен триггер. Они содержат информацию, которую изменил пользователь.

Примером для данного триггера может стать аналогичный вышеприведенному триггер, срабатывающий на удаление записей из таблицы Student\_c – удаляющий аналогичные записи в таблице Student. При необходимости можно усложнять триггер для срабатывания при каких либо условиях:

CREATE TRIGGER Table\_delete

ON Student\_c

FOR DELETE

###### AS

DELETE S\_lastname

FROM Student, Deleted

WHERE EXISTS Student.S\_lastname = Deleted.Lastname

Когда выполняется команда UPDATE для таблицы, которая имеет соответствующий триггер, исходные записи перемещаются в таблицу deleted, в то время как измененные записи помещаются в таблицу inserted. Сама таблица тоже обновляется. При этом учитывайте следующее:

* После того как все записи будут модифицированы, загружаются таблицы deleted и inserted, после чего срабатывает триггер на модификацию данных текущей таблицы.
* Триггер может обращаться как к таблицам deleted и inserted, так и к модифицируемой, для того чтобы определить, как много данных было обновлено и какие действия следует предпринять.