**Триггеры PostgreSQL**

Введение в триггеры

Триггер - это набор действий, которые запускаются автоматически, когда указанная операция изменения (оператор SQL INSERT, UPDATE, DELETE или TRUNCATE) выполняется над указанной таблицей. Триггеры полезны для таких задач, как обеспечение соблюдения бизнес-правил, проверка входных данных и ведение контрольного журнала.

**Содержание:**

* [Использует для триггеров](http://kodesource.top/PostgreSQL/postgresql-triggers.php#UFT)
* [Преимущества использования триггеров в бизнесе](http://kodesource.top/PostgreSQL/postgresql-triggers.php#BFT)
* [Создать триггеры PostgreSQL](http://kodesource.top/PostgreSQL/postgresql-triggers.php#CMT)
* [Пример базы данных, таблица, структура таблицы, записи таблицы](http://kodesource.top/PostgreSQL/postgresql-triggers.php#SDT)
* [Триггер PostgreSQL: пример ПОСЛЕ ВСТАВКИ](http://kodesource.top/PostgreSQL/postgresql-triggers.php#MTAI)
* [Триггер PostgreSQL: пример перед вставкой](http://kodesource.top/PostgreSQL/postgresql-triggers.php#MTBI)
* [Триггер PostgreSQL: пример ПОСЛЕ ОБНОВЛЕНИЯ](http://kodesource.top/PostgreSQL/postgresql-triggers.php#MTAU)
* [Триггер PostgreSQL: пример перед обновлением](http://kodesource.top/PostgreSQL/postgresql-triggers.php#MTBU)
* [Триггер PostgreSQL: пример ПОСЛЕ УДАЛЕНИЯ](http://kodesource.top/PostgreSQL/postgresql-triggers.php#MTAD)
* [Удалить / удалить триггер PostgreSQL](http://kodesource.top/PostgreSQL/postgresql-triggers.php#DMT)

Использует для триггеров:

* Применять бизнес-правила
* Проверьте входные данные
* Создайте уникальное значение для вновь вставленной строки в другом файле.
* Записывать в другие файлы для целей аудита
* Запрос из других файлов для перекрестных ссылок
* Доступ к системным функциям
* Репликация данных в разные файлы для обеспечения согласованности данных

Преимущества использования триггеров в бизнесе:

* Ускоренная разработка приложений. Поскольку база данных хранит триггеры, вам не нужно кодировать действия триггера в каждом приложении базы данных.
* Глобальное обеспечение соблюдения бизнес-правил. Определите триггер один раз, а затем повторно используйте его для любого приложения, которое использует базу данных.
* Более простое обслуживание. Если бизнес-политика меняется, вам нужно изменить только соответствующую триггерную программу вместо каждой прикладной программы.
* Улучшить производительность в среде клиент / сервер. Все правила запускаются на сервере до возврата результата.

Реализация триггеров SQL основана на стандарте SQL. Он поддерживает конструкции, которые являются общими для большинства языков программирования. Он поддерживает объявление локальных переменных, операторы для управления потоком процедуры, присвоение результатов выражений переменным и обработку ошибок.

PostgreSQL: создать триггер

Триггер - это именованный объект базы данных, который связан с таблицей, и он активируется, когда происходит конкретное событие (например, вставка, обновление или удаление) для таблицы / представлений. Оператор CREATE TRIGGER создает новый триггер в PostgreSQL. Вот синтаксис:

**Синтаксис:**

СОЗДАТЬ [ОГРАНИЧЕНИЕ] ТРИГГЕР имя {ДО | ПОСЛЕ | INSTEAD OF} {event [OR ...]}

ON table\_name

[FROM referenced\_table\_name]

[НЕ ЗАДЕРЖАЕТ | [НЕЗАВИСИМЫЙ] {ПЕРВОНАЧАЛЬНО НЕМЕДЛЕННЫЙ | ПЕРВОНАЧАЛЬНО ОТЛОЖЕН}

[ДЛЯ [КАЖДОГО] {ROW | ЗАЯВЛЕНИЕ } ]

[КОГДА (условие)]

ВЫПОЛНИТЬ ПРОЦЕДУРУ имя\_функции (аргументы)

**параметры**

|  |  |
| --- | --- |
| **название** | **Описание** |
| название | Название триггера. Триггер должен отличаться от имени любого другого триггера для той же таблицы. Имя не может быть дополнено схемой - триггер наследует схему своей таблицы. |
| ДО ПОСЛЕ ВМЕСТО | Определяет, вызывается ли функция до, после или вместо события. Триггер ограничения может быть указан только как AFTER. |
| событие | Один из INSERT, UPDATE, DELETE или TRUNCATE, который запустит триггер. |
| table\_name | Имя таблицы или представления триггера для. |
| referenced\_table\_name | (Возможно, дополненное схемой) имя другой таблицы, на которую ссылается ограничение. Эта опция используется для ограничений внешнего ключа и не рекомендуется для общего использования. Это может быть указано только для триггеров ограничения. |
| НЕ ЗАДЕРЖАЕТ НЕ откладываемые ПЕРВОНАЧАЛЬНО НЕМЕДЛЕННЫЙ ПЕРВОНАЧАЛЬНО ОТЛОЖЕН | Время по умолчанию для триггера. |
| ДЛЯ КАЖДОГО РЯДА ЗА КАЖДУЮ ЗАЯВЛЕНИЕ | Указывает, должна ли процедура триггера запускаться один раз для каждой строки, на которую воздействует событие триггера, или только один раз для оператора SQL. Если ни то, ни другое не указано, значение FOR EACH STATEMENT используется по умолчанию. |
| состояние | Булево выражение, которое определяет, будет ли фактически выполняться функция триггера. |
| function\_name | Предоставленная пользователем функция, которая объявлена как не имеющая аргументов и возвращающая тип триггера, которая выполняется при срабатывании триггера. |
| аргументы | Необязательный список аргументов через запятую, который должен быть предоставлен функции при выполнении триггера. Аргументы - это строковые константы. |

Триггеры, которые определены для запуска триггерного события INSTEAD OF, должны быть помечены FOR EACH ROW и могут быть определены только для представлений. Триггеры ДО и ПОСЛЕ на виде должны быть помечены как ДЛЯ КАЖДОГО ЗАЯВЛЕНИЯ. Кроме того, триггеры могут быть определены для срабатывания для TRUNCATE, но только для КАЖДОГО ЗАЯВЛЕНИЯ. В следующей таблице приведены типы триггеров, которые можно использовать в таблицах и представлениях:

|  |  |  |  |
| --- | --- | --- | --- |
| **когда** | **Событие** | **На уровне строк** | **Заявление на уровне** |
| ДО | INSERT / UPDATE / DELETE | таблицы | Таблицы и представления |
| TRUNCATE | - | таблицы |
| ПОСЛЕ | INSERT / UPDATE / DELETE | таблицы | Таблицы и представления |
| TRUNCATE | - | таблицы |
| ВМЕСТО | INSERT / UPDATE / DELETE | Просмотры | - |
| TRUNCATE | - | - |

Вот простой пример триггерной функции.

Код:

CREATE OR REPLACE FUNCTION test()

RETURNS trigger AS

$$

BEGIN

INSERT INTO test\_table(col1,col2,col3)

VALUES(NEW.col1,NEW.col2,current\_date);

RETURN NEW;

END;

$$

LANGUAGE 'plpgsql';

Copy

Теперь мы можем создать триггер, который будет запускать во время выполнения события, как указано в триггере для связанных таблиц.

Код:

CREATE TRIGGER test\_trigger

AFTER INSERT

ON test\_table

FOR EACH ROW

EXECUTE PROCEDURE test();

Copy

В вышеупомянутой триггерной функции есть новое ключевое слово « **NEW** », которое является расширением PostgreSQL для триггеров. Существует два расширения PostgreSQL для запуска « **OLD** » и « **NEW** ». СТАРЫЙ и НОВЫЙ не чувствительны к регистру.

* В теле триггера ключевые слова OLD и NEW позволяют получить доступ к столбцам в строках, затронутых триггером.
* В триггере INSERT можно использовать только NEW.col\_name.
* В триггере UPDATE вы можете использовать OLD.col\_name для ссылки на столбцы строки перед ее обновлением и NEW.col\_name для ссылки на столбцы строки после ее обновления.
* В триггере DELETE может использоваться только OLD.col\_name; нового ряда нет

Столбец с именем OLD доступен только для чтения. Вы можете ссылаться на него (если у вас есть привилегия SELECT), но не можете изменять его. Вы можете ссылаться на столбец с именем NEW, если у вас есть привилегия SELECT для него. В триггере BEFORE вы также можете изменить его значение с помощью SET NEW.col\_name = value, если у вас есть привилегия UPDATE для него. Это означает, что вы можете использовать триггер для изменения значений, которые будут вставлены в новую строку или использованы для обновления строки. (Такой оператор SET не действует в триггере AFTER, поскольку изменение строки уже произошло.)

Пример базы данных, таблица, структура таблицы, записи таблицы для различных примеров

Записи таблицы (по некоторым полям): **emp\_details**

postgres = # ВЫБРАТЬ EMPLOYEE\_ID, FIRST\_NAME, LAST\_NAME, JOB\_ID, SALARY, COMMISSION\_PCT FROM emp\_details;

employee\_id | имя\_файла | фамилия | job\_id | зарплата | COMMISSION\_PCT

------------- + ------------- + ----------- + ---------- - + ---------- + ----------------

100 | Стивен | Король | AD\_PRES | 24000,00 | 0,00

101 | Нина | Коххар | AD\_VP | 17000,00 | 0,00

102 | Лекс | Де Хаан | AD\_VP | 17000,00 | 0,00

103 | Александр | Hunold | IT\_PROG | 9000,00 | 0,00

104 | Брюс | Эрнст | IT\_PROG | 6000,00 | 0,00

105 | Дэвид | Остин | IT\_PROG | 4800,00 | 0,00

106 | Валли | Патабала | IT\_PROG | 4800,00 | 0,00

107 | Диана | Лоренц | IT\_PROG | 4200,00 | 0,00

108 | Нэнси | Гринберг | FI\_MGR | 12000,00 | 0,00

109 | Даниэль | Фавиет | FI\_ACCOUNT | 9000,00 | 0,00

110 | Джон | Чен | FI\_ACCOUNT | 8200,00 | 0,00

111 | Исмаэль | Sciarra | FI\_ACCOUNT | 7700,00 | 0,00

112 | Жозе Мануэль | Урман | FI\_ACCOUNT | 7800,00 | 0,00

(13 рядов)

Триггер PostgreSQL: пример ПОСЛЕ ВСТАВКИ

В следующем примере у нас есть две таблицы: emp\_details и emp\_log. Чтобы вставить некоторую информацию в таблицу emp\_logs (которая имеет три поля emp\_id и salary и edttime) каждый раз, когда в таблицу emp\_details входит INSERT, мы использовали следующий триггер:

Сначала нужно создать триггерную функцию. Вот триггерная функция rec\_insert ()

Код:

CREATE OR REPLACE FUNCTION rec\_insert()

RETURNS trigger AS

$$

BEGIN

INSERT INTO emp\_log(emp\_id,salary,edittime)

VALUES(NEW.employee\_id,NEW.salary,current\_date);

RETURN NEW;

END;

$$

LANGUAGE 'plpgsql';

Copy

Вот триггер ins\_same\_rec:

Код:

CREATE TRIGGER ins\_same\_rec

AFTER INSERT

ON emp\_details

FOR EACH ROW

EXECUTE PROCEDURE rec\_insert();

Copy

Записи таблицы (по некоторым столбцам): **emp\_details**

postgres = # ВЫБРАТЬ EMPLOYEE\_ID, FIRST\_NAME, LAST\_NAME, JOB\_ID, SALARY, COMMISSION\_PCT FROM emp\_details;

employee\_id | имя\_файла | фамилия | job\_id | зарплата | COMMISSION\_PCT

------------- + ------------- + ----------- + ---------- - + ---------- + ----------------

100 | Стивен | Король | AD\_PRES | 24000,00 | 0,00

101 | Нина | Коххар | AD\_VP | 17000,00 | 0,00

102 | Лекс | Де Хаан | AD\_VP | 17000,00 | 0,00

103 | Александр | Hunold | IT\_PROG | 9000,00 | 0,00

104 | Брюс | Эрнст | IT\_PROG | 6000,00 | 0,00

105 | Дэвид | Остин | IT\_PROG | 4800,00 | 0,00

106 | Валли | Патабала | IT\_PROG | 4800,00 | 0,00

107 | Диана | Лоренц | IT\_PROG | 4200,00 | 0,00

108 | Нэнси | Гринберг | FI\_MGR | 12000,00 | 0,00

109 | Даниэль | Фавиет | FI\_ACCOUNT | 9000,00 | 0,00

110 | Джон | Чен | FI\_ACCOUNT | 8200,00 | 0,00

111 | Исмаэль | Sciarra | FI\_ACCOUNT | 7700,00 | 0,00

112 | Жозе Мануэль | Урман | FI\_ACCOUNT | 7800,00 | 0,00

(13 рядов)

Записи таблицы (все столбцы): **emp\_log**

postgres = # SELECT \* FROM emp\_log;

emp\_id | зарплата | EDITTIME

-------- + -------- + ------------

100 | 24000 | 2011-01-15

101 | 17000 | 2010-01-12

102 | 17000 | 2010-09-22

103 | 9000 | 2011-06-21

104 | 6000 | 2012-07-05

105 | 4800 | 2011-06-02

(6 рядов)

Теперь вставьте одну запись в таблицу emp\_details, просмотрите записи в таблицах emp\_details и emp\_log:

Код:

INSERT INTO emp\_details VALUES(236, 'RABI', 'CHANDRA', 'RABI',

'590.423.45700', '2013-01-12', 'AD\_VP', 15000, .5);

Copy

postgres = # ВЫБРАТЬ EMPLOYEE\_ID, FIRST\_NAME, LAST\_NAME, JOB\_ID, SALARY, COMMISSION\_PCT FROM emp\_details;

employee\_id | имя\_файла | фамилия | job\_id | зарплата | COMMISSION\_PCT

------------- + ------------- + ----------- + ---------- - + ---------- + ----------------

100 | Стивен | Король | AD\_PRES | 24000,00 | 0,00

101 | Нина | Коххар | AD\_VP | 17000,00 | 0,00

102 | Лекс | Де Хаан | AD\_VP | 17000,00 | 0,00

103 | Александр | Hunold | IT\_PROG | 9000,00 | 0,00

104 | Брюс | Эрнст | IT\_PROG | 6000,00 | 0,00

105 | Дэвид | Остин | IT\_PROG | 4800,00 | 0,00

106 | Валли | Патабала | IT\_PROG | 4800,00 | 0,00

107 | Диана | Лоренц | IT\_PROG | 4200,00 | 0,00

108 | Нэнси | Гринберг | FI\_MGR | 12000,00 | 0,00

109 | Даниэль | Фавиет | FI\_ACCOUNT | 9000,00 | 0,00

110 | Джон | Чен | FI\_ACCOUNT | 8200,00 | 0,00

111 | Исмаэль | Sciarra | FI\_ACCOUNT | 7700,00 | 0,00

112 | Жозе Мануэль | Урман | FI\_ACCOUNT | 7800,00 | 0,00

236 | РАБИ | ЧАНДРА | AD\_VP | 15000,00 | 0,50

(14 рядов)

postgres = # SELECT \* FROM emp\_log;

emp\_id | зарплата | EDITTIME

-------- + -------- + ------------

100 | 24000 | 2011-01-15

101 | 17000 | 2010-01-12

102 | 17000 | 2010-09-22

103 | 9000 | 2011-06-21

104 | 6000 | 2012-07-05

105 | 4800 | 2011-06-02

236 | 15000 | 2014-09-15

(7 рядов)

Триггер PostgreSQL: пример перед вставкой

В следующем примере перед вставкой новой записи в таблицу emp\_details триггер проверяет значение столбца FIRST\_NAME, LAST\_NAME, JOB\_ID и  
- Если есть пробелы перед или после FIRST\_NAME, функция LAST\_NAME, LTRIM () удалит их.  
- Значение JOB\_ID будет преобразовано в верхний регистр функцией UPPER ().  
Вот триггерная функция befo\_insert ():

Код:

CREATE OR REPLACE FUNCTION befo\_insert()

RETURNS trigger AS

$$

BEGIN

NEW.FIRST\_NAME = LTRIM(NEW.FIRST\_NAME);

NEW.LAST\_NAME = LTRIM(NEW.LAST\_NAME);

NEW.JOB\_ID = UPPER(NEW.JOB\_ID);

RETURN NEW;

END;

$$

LANGUAGE 'plpgsql';

Copy

Вот триггер che\_val\_befo\_ins:

Код:

CREATE TRIGGER che\_val\_befo\_ins

BEFORE INSERT

ON emp\_details

FOR EACH ROW

EXECUTE PROCEDURE befo\_insert();

Copy

Теперь вставьте строку в таблицу emp\_details (проверьте столбцы FIRST\_NAME, LAST\_NAME, JOB\_ID):

Код:

INSERT INTO emp\_details VALUES (334, ' Ana ', ' King', 'ANA',

'690.432.45701', '2013-02-05', 'it\_prog', 17000, .50);

Copy

Теперь перечислите следующие поля emp\_details:

postgres = # ВЫБРАТЬ EMPLOYEE\_ID, FIRST\_NAME, LAST\_NAME, JOB\_ID, SALARY, COMMISSION\_PCT FROM emp\_details;

employee\_id | имя\_файла | фамилия | job\_id | зарплата | COMMISSION\_PCT

------------- + ------------- + ----------- + ---------- - + ---------- + ----------------

100 | Стивен | Король | AD\_PRES | 24000,00 | 0,00

101 | Нина | Коххар | AD\_VP | 17000,00 | 0,00

102 | Лекс | Де Хаан | AD\_VP | 17000,00 | 0,00

103 | Александр | Hunold | IT\_PROG | 9000,00 | 0,00

104 | Брюс | Эрнст | IT\_PROG | 6000,00 | 0,00

105 | Дэвид | Остин | IT\_PROG | 4800,00 | 0,00

106 | Валли | Патабала | IT\_PROG | 4800,00 | 0,00

107 | Диана | Лоренц | IT\_PROG | 4200,00 | 0,00

108 | Нэнси | Гринберг | FI\_MGR | 12000,00 | 0,00

109 | Даниэль | Фавиет | FI\_ACCOUNT | 9000,00 | 0,00

110 | Джон | Чен | FI\_ACCOUNT | 8200,00 | 0,00

111 | Исмаэль | Sciarra | FI\_ACCOUNT | 7700,00 | 0,00

112 | Жозе Мануэль | Урман | FI\_ACCOUNT | 7800,00 | 0,00

236 | РАБИ | ЧАНДРА | AD\_VP | 15000,00 | 0,50

334 | Ана | Король | IT\_PROG | 17000,00 | 0,50

(15 рядов)

Смотрите последний ряд:

FIRST\_NAME -> «Ана» заменена на «Ана»

LAST\_NAME -> «Король» изменен на «Король»

JOB\_ID -> 'it\_prog' изменено на 'IT\_PROG'

Триггер PostgreSQL: пример ПОСЛЕ ОБНОВЛЕНИЯ

У нас есть две таблицы student\_mast и stu\_log. student\_mast имеет три столбца: STUDENT\_ID, NAME, ST\_CLASS. Таблица stu\_log имеет два столбца user\_id и описание.

postgres = # SELECT \* FROM student\_mast;

student\_id | имя | st\_class

------------ + --------------------------- + --------- -

1 | Стивен Кинг | 7

2 | Нина Кочхар | 8

3 | Лекс Де Хаан | 8

4 | Александр Хунольд | 10

(4 ряда)

Пусть мы продвинем всех учеников в следующем классе, т.е. 7 будет 8, 8 будет 9 и так далее. После обновления одной строки в таблице student\_mast в таблицу stu\_log будет вставлена новая строка, в которой мы будем хранить текущий идентификатор пользователя и небольшое описание текущего обновления. Вот код триггера:

Код:

CREATE OR REPLACE FUNCTION aft\_update()

RETURNS trigger AS

$$

BEGIN

INSERT into stu\_log VALUES (user, CONCAT('Update Student Record ',

OLD.NAME,' Previous Class :',OLD.ST\_CLASS,' Present Class ',

NEW.st\_class));

RETURN NEW;

END;

$$

LANGUAGE 'plpgsql';

Copy

Вот триггер для этого события

Код:

CREATE TRIGGER updt\_log

AFTER UPDATE

ON student\_mast

FOR EACH ROW

EXECUTE PROCEDURE aft\_update();

Copy

Теперь обновите таблицу student\_mast:

Код:

UPDATE STUDENT\_MAST SET ST\_CLASS = ST\_CLASS + 1;

Copy

Триггер показывает вам обновленные записи в «stu\_log». Вот последняя позиция таблиц STUDENT\_MAST и STU\_LOG:

postgres = # SELECT \* FROM student\_mast;

student\_id | имя | st\_class

------------ + --------------------------- + --------- -

1 | Стивен Кинг | 8

2 | Нина Кочхар | 9

3 | Лекс Де Хаан | 9

4 | Александр Хунольд | 11

(4 ряда)

postgres = # select \* from stu\_log;

user\_id | описание

--------------------------- + ---------------------- -------------------------------------------------- ------------------------------

Postgres | Обновление студенческих записей Стивена Кинга Предыдущий класс: 7 Настоящий класс 8

Postgres | Обновить студенческий билет Neena Kochhar Предыдущий класс: 8 Текущий класс 9

Postgres | Обновить студенческие записи Lex De Haan Предыдущий класс: 8 Настоящий класс 9

Postgres | Обновить студенческие записи Александра Хунольда Предыдущий класс: 10 Настоящий класс 11

(4 ряда)

Триггер PostgreSQL: пример перед обновлением

У нас есть таблица student\_marks с 10 столбцами и 4 строками. Данные есть только в столбцах STUDENT\_ID и NAME.

postgres = # SELECT \* FROM STUDENT\_MARKS;

student\_id | имя | sub1 | sub2 | sub3 | sub4 | sub5 | всего | per\_marks | класс

------------ + --------------------------- + ------ + - ---- + ------ + ------ + ------ + ------- + ----------- + ---- ---

1 | Стивен Кинг | | | | | | | |

2 | Нина Кочхар | | | | | | | |

3 | Лекс Де Хаан | | | | | | | |

4 | Александр Хунольд | | | | | | | |

(4 ряда)

Теперь экзамен закончен, и мы получили все предметные оценки, теперь мы обновим таблицу, итоговые оценки по всем предметам, процент от общих оценок и оценки будет автоматически рассчитан. Для этого примера расчета принимаются следующие условия:

Общее количество баллов (будет храниться в столбце ВСЕГО): ВСЕГО = SUB1 + SUB2 + SUB3 + SUB4 + SUB5

Процент меток (будет храниться в столбце PER\_MARKS): PER\_MARKS = (TOTAL) / 5

Оценка (будет храниться столбец GRADE):

- Если PER\_MARKS> = 90 -> «ОТЛИЧНО»

- Если PER\_MARKS> = 75 И PER\_MARKS <90 -> 'ОЧЕНЬ ХОРОШО'

- Если PER\_MARKS> = 60 И PER\_MARKS <75 -> «ХОРОШО»

- Если PER\_MARKS> = 40 И PER\_MARKS <60 -> «СРЕДНИЙ»

- Если PER\_MARKS <40-> «НЕ РЕКЛАМНО»

Вот код:

Код:

UPDATE STUDENT\_MARKS SET SUB1 = 54, SUB2 = 69, SUB3 = 89, SUB4 = 87,

SUB5 = 59 WHERE STUDENT\_ID = 1;

Copy

Позвольте обновить оценки ученика:

Вот триггерная функция befo\_update:

Код:

CREATE OR REPLACE FUNCTION befo\_update()

RETURNS trigger AS

$$

BEGIN

NEW.TOTAL = NEW.SUB1 + NEW.SUB2 + NEW.SUB3 + NEW.SUB4 + NEW.SUB5;

NEW.PER\_MARKS = NEW.TOTAL/5;

IF NEW.PER\_MARKS >=90 THEN

NEW.GRADE ='EXCELLENT';

ELSEIF NEW.PER\_MARKS>=75 AND NEW.PER\_MARKS<90 THEN

NEW.GRADE ='VERY GOOD';

ELSEIF NEW.PER\_MARKS>=60 AND NEW.PER\_MARKS<75 THEN

NEW.GRADE ='GOOD';

ELSEIF NEW.PER\_MARKS>=40 AND NEW.PER\_MARKS<60 THEN

NEW.GRADE ='AVERAGE';

ELSE

NEW.GRADE ='NOT PROMOTED';

END IF;

RETURN NEW;

END;

$$

LANGUAGE 'plpgsql';

Copy

Вот триггер

Код:

CREATE TRIGGER updt\_marks

BEFORE UPDATE

ON student\_marks

FOR EACH ROW

EXECUTE PROCEDURE befo\_update();

Copy

Теперь проверьте таблицу STUDENT\_MARKS с обновленными данными. Триггер покажет вам обновленные записи в «stu\_log».

postgres = # SELECT \* FROM STUDENT\_MARKS;

student\_id | имя | sub1 | sub2 | sub3 | sub4 | sub5 | всего | per\_marks | класс

------------ + --------------------------- + ------ + - ---- + ------ + ------ + ------ + ------- + ----------- + ---- ------------------

2 | Нина Кочхар | | | | | | | |

3 | Лекс Де Хаан | | | | | | | |

4 | Александр Хунольд | | | | | | | |

1 | Стивен Кинг | 54 | 69 | 89 | 87 | 59 | 358 | 71 | ХОРОШО

(4 ряда)

Триггер PostgreSQL: пример ПОСЛЕ УДАЛЕНИЯ

В нашем примере «AFTER UPDATE» у нас было две таблицы student\_mast и stu\_log. student\_mast состоит из трех столбцов STUDENT\_ID, NAME, ST\_CLASS и таблица stu\_log имеет два столбца user\_id и description. Мы хотим сохранить некоторую информацию в таблице stu\_log после того, как в таблице student\_mast произошла операция удаления. Вот триггер:

Код:

CREATE OR REPLACE FUNCTION aft\_delete()

RETURNS trigger AS

$$

BEGIN

INSERT into stu\_log VALUES (user, CONCAT('Update Student Record ',

OLD.NAME,' Class :',OLD.ST\_CLASS,' -> Deleted on ',

NOW()));

RETURN NEW;

END;

$$

LANGUAGE 'plpgsql';

Copy

Вот триггер

Код:

CREATE TRIGGER delete\_stu

AFTER DELETE

ON student\_mast

FOR EACH ROW

EXECUTE PROCEDURE aft\_delete();

Copy

Позвольте удалить студента из student\_mast

Код:

DELETE FROM STUDENT\_MAST WHERE STUDENT\_ID = 1;

Copy

Вот последняя позиция таблиц student\_mast, stu\_log:

postgres = # SELECT \* FROM STUDENT\_MAST;

student\_id | имя | st\_class

------------ + --------------------------- + --------- -

2 | Нина Кочхар | 9

3 | Лекс Де Хаан | 9

4 | Александр Хунольд | 11

(3 ряда)

postgres = # select \* from stu\_log;

user\_id | описание

--------------------------- + ---------------------- -------------------------------------------------- ------------------------------

Postgres | Обновление студенческих записей Стивена Кинга Предыдущий класс: 7 Настоящий класс 8

Postgres | Обновить студенческий билет Neena Kochhar Предыдущий класс: 8 Текущий класс 9

Postgres | Обновить студенческие записи Lex De Haan Предыдущий класс: 8 Настоящий класс 9

Postgres | Обновить студенческие записи Александра Хунольда Предыдущий класс: 10 Настоящий класс 11

Postgres | Обновление студенческого рекорда Стивена Кинга Класс: 7 -> Удалено 2014-09-16 16: 30: 35.093 + 05: 30

(5 рядов)

DROP триггер PostgreSQL

Чтобы удалить или уничтожить триггер, используйте оператор DROP TRIGGER. Для выполнения этой команды текущий пользователь должен быть владельцем таблицы, для которой определен триггер.

**Синтаксис**

DROP TRIGGER [ЕСЛИ СУЩЕСТВУЕТ] name ON table\_name [CASCADE | ОГРАНИЧЕНИЕ]

**параметры**

|  |  |
| --- | --- |
| **название** | **Описание** |
| ЕСЛИ СУЩЕСТВУЕТ | Не выдавайте ошибку, если триггера не существует. В этом случае выдается уведомление. |
| название | Имя триггера для удаления. |
| table\_name | Имя (возможно, дополненное схемой) таблицы, для которой определен триггер. |
| CASCADE | Автоматически отбрасывать объекты, которые зависят от триггера. |
| RESTRICT | Откажитесь сбросить триггер, если от него зависят какие-либо объекты. Это по умолчанию. |

**Пример:**

Если вы удалите или удалите только что созданный триггер delete\_stu, можно использовать следующий оператор:

Код:

DROP TRIGGER delete\_stu on student\_mast;

Copy

Триггер delete\_stu будет удален.