**Просто и доступно про триггеры PostgreSQL**

В свое время, когда только начинал разбираться с функциями, процедурами и триггерами в СУБД [PostgreSQL](http://www.postgresql.org/" \o "PostgreSQL) столкнулся с тем что нормального, простого и не абстрактоного примера найти было невозможно.

Вот мой пример :)

Задача: реализовать самую простую систему логирования пользователей. Она будет следить за изменениями в таблице пользователей и при изменениях добавлять текстовые акшены в таблицу логов.

Так как задача простая, то я не буду приводить рабочие таблицы со всеми полями. Для примера возьмем максимально простые (расширить всегда можно, главное смысл понять):

-- Таблица пользователей

**CREATE** **TABLE** users

(

"name" text

)

-- Таблица логов

**CREATE** **TABLE** **logs**

(

"text" text,

"added" timestamp without time zone

)

Все максимально просто. Что теперь надо сделать для добавления логов ?

Создать триггер. Триггер эта такая штука, которая следит за изменениями таблицы через [DML](http://ru.wikipedia.org/wiki/DML) функции (пока надо знать только то что это изменения таблицы с помощью UPDATE, INSERT и DELETE операций). Триггер определяется для конкретной таблицы и для конкретных операция и должен вызывать процедуру.

В нашем случае мы будем обрабатывать операции Update, Insert и Delete для базы users.

Сам триггер описывается на [PL/pgSQL](http://ru.wikipedia.org/wiki/PL/pgSQL) как:

**CREATE** TRIGGER t\_user

**AFTER** **INSERT** OR **UPDATE** OR **DELETE** **ON** users **FOR** EACH **ROW** EXECUTE **PROCEDURE** add\_to\_log ();

Собственно тут все понятно. Создать триггер t\_user который будет выполняться AFTER (после) команд INSERT, UPDATE, DELETE для каждой строки вызывать функцию add\_to\_log();

Тут есть небольшая особенность. Если попытаться выполнить эту инструкцию то получим ошибку, что функции add\_to\_log() нету. Значит пред тем как создавать триггер нужно создать функцию add\_to\_log().

Собственно вот она:

**CREATE** **OR** **REPLACE** **FUNCTION** add\_to\_log() **RETURNS** **TRIGGER** **AS** $$

**DECLARE**

mstr **varchar**(30);

astr **varchar**(100);

retstr **varchar**(254);

**BEGIN**

**IF** TG\_OP = 'INSERT' **THEN**

astr = **NEW**.name;

mstr := 'Add new user ';

retstr := mstr || astr;

**INSERT** **INTO** logs(text,added) **values** (retstr,NOW());

**RETURN** **NEW**;

**ELSIF** TG\_OP = 'UPDATE' **THEN**

astr = **NEW**.name;

mstr := 'Update user ';

retstr := mstr || astr;

**INSERT** **INTO** logs(text,added) **values** (retstr,NOW());

**RETURN** **NEW**;

**ELSIF** TG\_OP = 'DELETE' **THEN**

astr = **OLD**.name;

mstr := 'Remove user ';

retstr := mstr || astr;

**INSERT** **INTO** logs(text,added) **values** (retstr,NOW());

**RETURN** **OLD**;

**END IF**;

**END**;

$$ **LANGUAGE** plpgsql;

Что тут происходит ? Собственно определяется новая функция, без входящих параметров, возвращает специальный тип TRIGGER. Для внутреннего использования определяем 3-и переменные в разделе DECLARE. В самой процедуре смотрим переменную TG\_OP (внутренняя переменная триггера которая определяет с какой операцией была вызвана процедура). В зависимости от операции определяем переменную mstr, собираем строку retstr которая будет записана в базу данных (обратите внимание как производится контаминация строк в pgsql, через ||) и собственно делаем запись в таблицу логов (INSERT INTO).  Надо пояснить что такое переменные NEW и OLD. Это собственно строки которые обрабатывает триггер. В случае INSERT переменная NEW будет содержать новую строку, а OLD будет пустая, в случае UPDATE обе переменные будут определены (соответствующими данными), а в случае DELETE переменная NEW будет пустая, OLD содержать удаляемую строку.

Если все правильно сделали то теперь при любой работе с таблицей users без нашего участия будет записываться лог действий с таблицей. Все очень прозрачно, ничего вызывать стороннего не надо.

У данного метода есть 2-е очевидные плюшки по сравнению с «программным методом» когда программист в бизнес логике приложения определяет точки вызова записи в лог:

* скорость работы, особенно это будет заметно на средне и высоко нагруженных приложениях,
* абсолютная абстракция от языка приложения. Администратору СУБД плевать с большой колокольни на то, какую там Яву или питон использовал программист приложения. Вносить изменения в логику работы базы без вмешательства в код приложения.