Авторы: Егор Рогов, Павел Лузанов

Обратная связь
Отзывы, замечания и предложения направляйте по адресу:
edu@postgrespro.ru

Темы

Триггеры и триггерные функции

В какой момент срабатывают триггеры

Контекст выполнения триггерной функции

Возвращаемое значение

Для чего можно применять триггеры и для чего не нужно

Событийные триггеры

Триггеры и функции

Триггер

объект базы данных — список обрабатываемых событий

при возникновении события вызывается триггерная функция и ей передается контекст

Триггерная функция

объект базы данных — код обработки события

выполняется в той же транзакции, что и основная операция

соглашение: функция не принимает параметры, возвращает значение псевдотипа `trigger` (фактически `record`)

может использоваться в нескольких триггерах

Механизм триггеров позволяет выполнять определенные действия «в ответ» на определенные события. Триггер состоит из двух частей: собственно триггера (который определяет события) и триггерной функции (которая определяет действия). И триггер, и функция являются самостоятельными объектами БД.

Когда возникает событие, на которое «подписан» триггер, вызывается триггерная функция. Ей передается контекст вызова, чтобы можно было определить, какой именно триггер и в каких условиях вызвал функцию.

Триггерная функция — это обычная функция, которая написана с учетом некоторых соглашений:

- она пишется на любом языке, кроме чистого SQL;

- она не имеет параметров;

- она возвращает значение типа trigger (на самом деле это псевдотип, по факту возвращается запись, соответствующая строке таблицы).

Триггерная функция выполняется в той же транзакции, что и основная операция. Таким образом, если триггерная функция завершится с ошибкой, вся транзакция будет прервана.

https://postgrespro.ru/docs/postgresql/9.6/trigger-definition.html

События

INSERT, UPDATE, DELETE

таблицы	before/after	statement
	before/after	row
представления	before/after	statement
	instead of	row

TRUNCATE

таблицы	before/after	statement

Условие WHEN

устанавливает дополнительный фильтр

Триггеры могут срабатывать на вставку (insert), обновление (update) или удаление (delete) строк в таблице или представлении, а также на опустошение (truncate) таблиц.

Триггер может срабатывать до выполнения действия (before), после него (after), или вместо него (instead of).

Триггер может срабатывать один раз для всей операции (for each statement), или каждый раз для каждой затронутой строки (for each row).

Не для любой комбинации этих условий можно создать триггер. Например, instead-of-триггеры можно определить только для представлений на уровне строк, а truncate-триггер можно определить только для таблиц и только на уровне оператора. Допустимые варианты перечислены на слайде.

Кроме того, можно сузить область действия триггера, указав дополнительное условие when: если условие не выполняется — триггер не срабатывает.

https://postgrespro.ru/docs/postgresql/9.6/sql-createtrigger

Before/after statement

Срабатывает

до/после операции
(даже если не затронута ни одна строка)

Возвращаемое значение

игнорируется

Контекст

TG-переменные

Рассмотрим подробнее разные типы триггеров.

Триггеры BEFORE и AFTER STATEMENT срабатывают один раз для операции независимо от того, сколько строк будет затронуто (возможно, что и не одной). BEFORE-триггер срабатывает до того, как операция начала выполняться, а AFTER-триггер — после того, как вся операция завершена.

Возвращаемое значение триггерной функции игнорируется, можно возвращать просто NULL.

Поскольку триггерная функция не имеет параметров, контекст вызова в PL/pgSQL передается ей с помощью предопределенных TG-переменных, таких, как:

- TG_WHEN = «BEFORE»/«AFTER»

- TG_LEVEL = «STATEMENT»

- TG_OP = «INSERT»/«UPDATE»/«DELETE»/«TRUNCATE»

и др.

https://postgrespro.ru/docs/postgresql/9.6/plpgsql-trigger.html

Before row

Срабатывает

перед действием со строкой
в процессе выполнения операции

Возвращаемое значение

строка (возможно, измененная)
null отменяет действие

Контекст

OLD update, delete
NEW insert, update

TG-переменные

Триггеры BEFORE ROW срабатывают каждый раз перед тем, как операция затронет строку. Это происходит непосредственно в процессе выполнения операции.

В качестве контекста триггерная функция получает переменные:

- OLD — старая строка (не определено для операции вставки),
- NEW — измененная строка (не определено для удаления),
- TG_WHEN = «BEFORE»
- TG_LEVEL = «ROW»
- TG_OP = «INSERT»/«UPDATE»/«DELETE»

и др.

Возврат неопределенного значения NULL воспринимается как отмена действия над данной строкой. Сама операция продолжит выполнение, но текущая строка не будет обработана и другие триггеры для этой строки не сработают.

Чтобы не вмешиваться в работу операции, триггер должен вернуть строку в том виде, в котором ее собирается изменить операция: NEW для вставки и обновления, любое значение (но не NULL) для удаления (обычно используют OLD).

Но триггерная функция может и изменить значение NEW, чтобы повлиять на результат операции — часто именно для этого такой триггер и создают.

Instead of row

Срабатывает

вместо действия со строкой
для представлений

Возвращаемое значение

строка (возможно, измененная) — будет видна в RETURNING

null отменяет действие

Контекст

OLD update, delete
NEW insert, update

TG-переменные

Триггеры INSTEAD OF очень похожи на триггеры BEFORE, но определяются только для представлений и срабатывают не до, а вместо операции.

В задачу таких триггеров обычно входит выполнение необходимых операций над базовыми таблицами представления. Также триггер может вернуть измененное значение NEW — именно оно будет видно при выполнении операции с указанием фразы RETURNING.

After row

Срабатывает

после выполнения операции,
но перед AFTER STATEMENT

очередь из прошедших условие WHEN

Возвращаемое значение

игнорируется

Контекст

OLD	update, delete
NEW	insert, update

TG-переменные

Триггеры AFTER ROW, как и BEFORE ROW, срабатывают для каждой затрагиваемой строки, но уже после того, как действие над строкой выполнено.

Более того, они срабатывают уже после того, как выполнена вся операция — таким образом, триггерная функция может «увидеть» таблицу уже в ее окончательном виде. Для этого события помещаются в очередь и обрабатываются после того, как выполнена операция (но до того, как сработают триггеры AFTER STATEMENT). Понятно, что чем меньше событий попадет в очередь, тем меньше будет накладных расходов — поэтому именно в этом случае очень полезно использовать условие WHEN, чтобы отсечь ненужные строки.

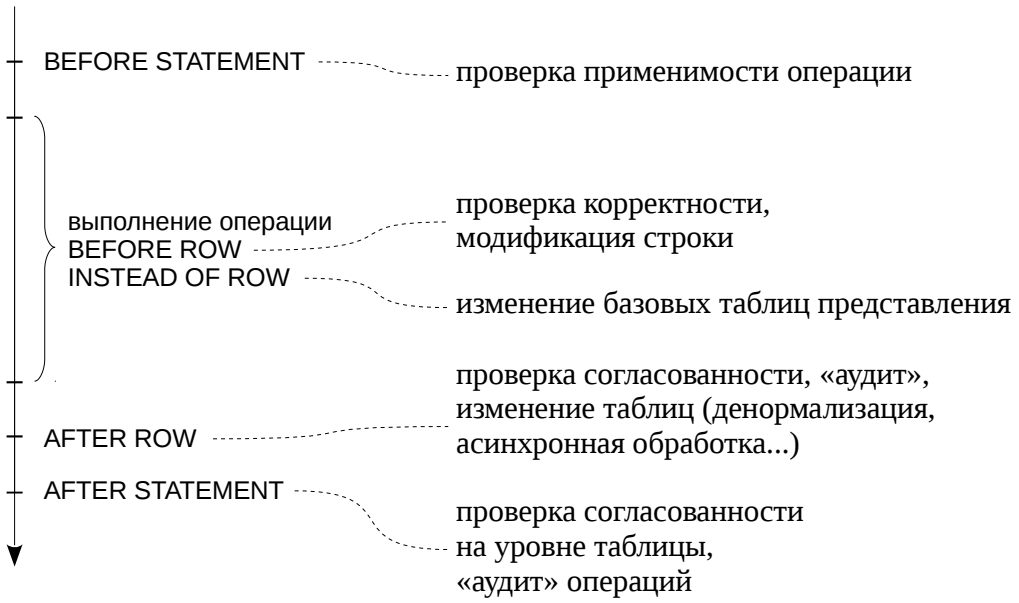
Возвращаемое значение триггеров AFTER ROW игнорируется (поскольку операция уже выполнена).

Контекст триггерной функции составляют:

- OLD — старая строка (не определено для операции вставки),
- NEW — новое значение строки (не определено для удаления),
- TG_WHEN = «AFTER»
- TG_LEVEL = «ROW»
- TG_OP = «INSERT»/«UPDATE»/«DELETE»

и др.

Возможное использование



Каково практическое применение триггеров?

Триггеры BEFORE STATEMENT и BEFORE ROW можно использовать, чтобы проверить корректность операции и при необходимости вызвать ошибку.

Триггеры BEFORE ROW можно применять для модификации строки (например, заполнить пустое поле нужным значением). Это бывает удобно, чтобы не повторять логику заполнения «технических» полей в каждой операции, а также позволяет вмешаться в работу приложения, код которого недоступен для изменения.

Триггеры INSTEAD OF ROW применяют для того, чтобы отобразить операции над представлением в операции над базовыми таблицами.

Триггеры AFTER ROW и AFTER STATEMENT можно применять для проверки согласованности операции и для «аудита» изменений (BEFORE-триггеры могут не подойти для этой роли из-за того, что при их выполнении еще не понятно, в каком состоянии окажется таблица в конце операции).

Триггеры AFTER ROW можно применять для каскадного изменения других таблиц (например, обновлять денормализованные данные при изменении базовых таблиц, или записывать изменения в очередь для последующей обработки вне данной транзакции).

Не стоит полагаться

На правила видимости

- триггерная VOLATILE-функция видит результат работы триггеров на уровне BEFORE ROW или INSTEAD OF ROW

На порядок вызова триггеров для одного события

- триггеры отрабатывают в алфавитном порядке

На передачу дополнительного контекста

- триггерной функции можно передать «параметры» через TG_ARGV

Сложную логику на триггерах почти невозможно отладить

- код вызывается неявно, трудно проследить зависимости
- крайне усложняется поддержка приложения

Однако не стоит злоупотреблять триггерами. Триггеры срабатывают неявно, что сильно затрудняет понимание логики работы и усложняет поддержку приложения. Попытки реализовать сложную логику на триггерах обычно заканчиваются плачевно.

Есть ряд тонкостей, связанных с триггерами, которые мы сознательно не рассматриваем подробно:

- правила видимости в триггерах BEFORE ROW и INSTEAD OF ROW (не стоит полагаться на порядок, в котором сработают триггеры — он неопределен);

- порядок вызова нескольких триггеров, обрабатывающих одно и то же событие (не стоит усугублять и без того неявное срабатывание триггеров завязкой на последовательность обработки);

- передача дополнительного контекста в триггерную функцию с помощью массива TG_ARGV (не стоит писать сверхуниверсальную триггерную функцию, лучше сделать несколько более простых).

Если вы столкнулись с тем, что эти тонкости важны для вашего приложения — серьезно задумайтесь.

Событийные триггеры

Событийный триггер

похож на обычный «табличный» триггер, но другой объект

Триггерная функция

соглашение: функция не принимает параметры,
возвращает значение псевдотипа `event_trigger`

для получения контекста служат специальные функции

События

DDL_COMMAND_START	перед выполнением команды
DDL_COMMAND_END	после выполнения команды
TABLE_REWRITE	перед перезаписью таблицы
SQL_DROP	после удаления объектов

Событийные триггеры — по сути те же триггеры, но срабатывают они не на DML-, а на DDL-операции (CREATE, ALTER, DROP, COMMENT, GRANT, REVOKE).

Такие триггеры являются не инструментом разработки приложений, а скорее служат для решения задач администрирования. Поэтому здесь мы упоминаем их только для полноты картины и не рассматриваем подробно.

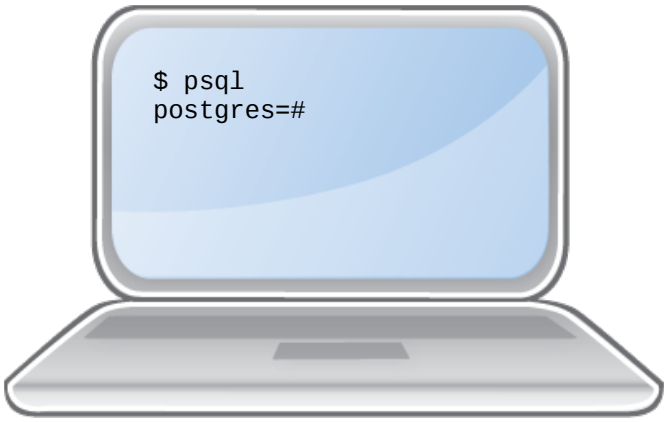
https://postgrespro.ru/docs/postgresql/9.6/event-trigger-definition.html

https://postgrespro.ru/docs/postgresql/9.6/event-trigger-matrix.html

https://postgrespro.ru/docs/postgresql/9.6/sql-createeventtrigger

https://postgrespro.ru/docs/postgresql/9.6/functions-event-triggers.html

Демонстрация



Итоги

Триггер — способ отреагировать на возникновение события

С помощью триггера можно отменить операцию, изменить ее результат или выполнить дополнительные действия

Триггер выполняется как часть транзакции; ошибка в триггере приводит к откату транзакции

Триггеры AFTER ROW обходятся дороже, чем остальные

Все хорошо в меру: сложную логику трудно отлаживать из-за неявного выполнения триггеров

Практика

1. Создайте триггер, обрабатывающий обновление поля onhand_qty представления catalog_v.
2. Проверьте, что в «Каталоге» появилась возможность заказывать книги.
3. Обеспечьте выполнение требования согласованности: количество книг на складе не может быть отрицательным (нельзя купить книгу, которой нет в наличии).
4. Внимательно проверьте правильность реализации, учитывая, что с приложением могут одновременно работать несколько пользователей.

3. Может показаться, что достаточно создать AFTER-триггер на таблице operations, подсчитывающий сумму qty_change. Однако на уровне изоляции read committed, с которым работает приложение «Книжный магазин», нам придется блокировать таблицу operations в эксклюзивном режиме — иначе возможны сценарии, при которых такая проверка не сработает.

Лучше поступить следующим образом: добавить в таблицу books поле onhand_qty и создать триггер, изменяющий это поле при изменении таблицы operations (то есть, фактически, выполнить денормализацию данных). На поле onhand_qty теперь можно наложить ограничение CHECK, реализующее требование согласованности. А функция onhand_qty(), которую мы создавали ранее. больше не нужна.

Особое внимание надо уделить начальной установке значения, учитывая, что одновременно с выполнением наших операций в системе могут работают пользователи.

Практика

1. Напишите триггер, увеличивающий счетчик (поле version) на единицу при каждом изменении строки. При вставке новой строки счетчик должен устанавливаться в единицу. Проверьте правильность работы.
2. Продолжите последний пример из демонстрации, добавив поддержку вставки и удаления строк.
3. В том же примере измените логику триггеров так, чтобы позиция автоматически создавалась, если не существует.