Методическое пособие по теме:

«Использование триггеров и создание пользовательских процедур»

Цель: Использование триггеров и создание пользовательских процедур, знакомство с PL/pgSQL .

Содержание:

Процедурный язык PL/pgSQL :

- добавляет управляющие конструкции к стандарту SQL;
- допускает сложные вычисления;
- может использовать все объекты БД, определенные пользователем;
- прост в использовании.

Преимущества использования PL/pgSQL:

Стандартный SQL используется в PostgreSQL и других реляционных БД как основной язык для создания запросов. Он переносим и прост, как для изучения, так и для использования. Однако слабое его место — в том, что каждая конструкция языка выполняется сервером отдельно. Это значит, что клиентское приложение должно отправлять каждый запрос серверу, получить его результат, определенным образом согласно логике приложения обработать его, посылать следующий запрос и т. д. В случае, если клиент и сервер БД расположены на разных машинах, это может привести к нежелательному увеличению задержек и объема пересылаемых от клиента серверу и наоборот данных.

При использовании PL/pgSQL все становится проще. Появляется возможность сгруппировать запросы и вычислительные блоки в единую конструкцию, которая будет размещаться и выполняться на сервере, а клиент будет отправлять запрос на её выполнение и получать результат, минуя все промежуточные пересылки данных назад—вперед, что в большинстве случаев очень позитивно сказывается на производительности.

Так же существует функциональность анонимных блоков, позволяющий писать запросы не на SQL, а прямо на любом существующем процедурном языке сервера, в том числе pl/pgSQL, без создания хранимых функций на сервере СУБД.

Триггерные функции

В PL/pgSQL можно создавать триггерные функции, которые будут вызываться при изменениях данных или событиях в базе данных. Триггерная функция создаётся командой CREATE FUNCTION, при этом у функции не должно быть аргументов, а типом возвращаемого значения должен быть trigger (для триггеров, срабатывающих при изменениях данных) или event_trigger (для триггеров, срабатывающих при событиях в базе). Для триггеров автоматически определяются специальные локальные переменные с именами вида TG_имя, описывающие условие, повлёкшее вызов триггера.

Триггеры при изменении данных

Триггер (англ. trigger) — хранимая процедура особого типа, которую пользователь не вызывает непосредственно, а исполнение которой обусловлено действием по модификации данных: добавлением INSERT, удалением DELETE строки в заданной таблице, или изменением UPDATE данных в определённом столбце заданной таблицы реляционной базы данных. Триггеры применяются для обеспечения целостности данных и реализации сложной бизнес-логики. Триггер запускается сервером автоматически при попытке изменения данных в таблице, с которой он связан. Все производимые им модификации данных рассматриваются как выполняемые в транзакции, в которой выполнено действие, вызвавшее срабатывание триггера. Соответственно, в случае обнаружения ошибки или нарушения целостности данных может произойти откат этой транзакции.

Триггер при изменении данных объявляется как функция без аргументов и с типом результата trigger. Заметьте, что эта функция должна объявляться без аргументов, даже если ожидается, что она будет получать аргументы, заданные в команде CREATE TRIGGER — такие аргументы передаются через TG_ARGV, как описано ниже.

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.

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

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.

Триггерная функция на **PL/pgSQL**

Триггер, показанный в этом примере, при любом добавлении или изменении строки в таблице сохраняет в этой строке информацию о текущем пользователе и отметку времени. Кроме того, он требует, чтобы было указано имя сотрудника, и зарплата задавалась положительным числом.

```
CREATE TABLE emp (
    empname text,
    salary integer,
    last_date timestamp,
    last_user text
);
CREATE FUNCTION emp_stamp() RETURNS trigger AS $emp_stamp$
    BEGIN
        -- Проверить, что указаны имя сотрудника и зарплата
        IF NEW.empname IS NULL THEN
            RAISE EXCEPTION 'empname cannot be null';
        END IF;
        IF NEW.salary IS NULL THEN
            RAISE EXCEPTION '% cannot have null salary', NEW.empname;
        END IF;

        -- Кто будет работать, если за это надо будет платить?
        IF NEW.salary < 0 THEN
            RAISE EXCEPTION '% cannot have a negative salary', NEW.empname;
        END IF;
        -- Запомнить, кто и когда изменил запись
        NEW.last_date := current_timestamp;
        NEW.last_user := current_user;
        RETURN NEW;
    END;
$emp_stamp$ LANGUAGE plpgsql;

CREATE TRIGGER emp_stamp BEFORE INSERT OR UPDATE ON emp
    FOR EACH ROW EXECUTE FUNCTION emp_stamp();
```

ТРИГГЕРЫ СОБЫТИЙ

Три́ггер (англ. trigger) — хранимая процедура особого типа, которую пользователь не вызывает непосредственно, а исполнение которой обусловлено действием по модификации данных: добавлением INSERT, удалением DELETE строки в заданной таблице, или изменением UPDATE данных в определённом столбце заданной таблицы реляционной базы данных.

В PL/pgSQL можно создавать событийные триггеры. Postgres Pro требует, чтобы функция, которая вызывается как событийный триггер, объявлялась без аргументов и типом возвращаемого значения был event_trigger.

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

TG_EVENT

Тип данных text. Строка, содержащая событие, для которого сработал триггер.

TG_TAG

Тип данных text. Переменная, содержащая тег команды, для которой сработал триггер.

Функция событийного триггера на **PL/pgSQL**

Триггер в этом примере просто выдаёт сообщение NOTICE каждый раз, когда выполняется поддерживаемая команда.

```
CREATE OR REPLACE FUNCTION snitch() RETURNS event_trigger AS $$
BEGIN
    RAISE NOTICE 'snitch: % %', tg_event, tg_tag;
END;
$$ LANGUAGE plpgsql;

CREATE EVENT TRIGGER snitch ON ddl_command_start EXECUTE FUNCTION snitch();
```

Процедуры

Процедура представляет собой объект базы данных, подобный функции. Отличие состоит в том, что процедура не возвращает значение, и поэтому для неё не определяется возвращаемый тип. Тогда как функция вызывается в составе запроса или команды DML, процедура вызывается явно, оператором CALL.

CREATE PROCEDURE — создать процедуру

Синтаксис:

```
CREATE [ OR REPLACE ] PROCEDURE
    имя ( [ [ режим_аргумента ] [ имя_аргумента ] тип_аргумента [ { DEFAULT |
= } выражение_по_умолчанию ] [, ...] ] )
  { LANGUAGE имя_языка
    | TRANSFORM { FOR TYPE имя_типа } [, ... ]
    | [ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER
    | SET параметр_конфигурации { TO значение | = значение | FROM CURRENT }
    | AS 'определение'
    | AS 'объектный_файл', 'объектный_символ'
  } ...
```

Команда CREATE PROCEDURE определяет новую процедуру. CREATE OR REPLACE PROCEDURE создаёт новую процедуру либо заменяет определение уже существующей. Чтобы определить процедуру, необходимо иметь право USAGE для соответствующего языка.

Если указано имя схемы, процедура создаётся в заданной схеме, в противном случае — в текущей. Имя новой процедуры должно отличаться от имён существующих процедур и функций с такими же типами аргументов в этой схеме. Однако процедуры и функции с аргументами разных типов могут иметь одно имя (это называется перегрузкой).

Команда CREATE OR REPLACE PROCEDURE предназначена для изменения текущего определения существующей процедуры. С её помощью нельзя изменить имя или типы аргументов (если попытаться сделать это, будет создана новая отдельная процедура).

Когда команда CREATE OR REPLACE PROCEDURE заменяет существующую процедуру, владелец и права доступа к этой процедуре не меняются. Все другие свойства процедуры получают значения, задаваемые командой явно или по умолчанию. Чтобы заменить процедуру, необходимо быть её владельцем (или быть членом роли-владельца).

Владельцем процедуры становится создавший её пользователь.

Чтобы создать процедуру, необходимо иметь право USAGE для типов её аргументов.

Параметры:

имя

Имя создаваемой процедуры

режим_аргумента

Режим аргумента: IN, INOUT или VARIADIC. По умолчанию подразумевается IN. (Режим OUT для процедур в настоящее время не поддерживается. Используйте вместо него INOUT.)

имя_аргумента

Имя аргумента.

тип_аргумента

Тип данных аргумента процедуры (возможно, дополненный схемой), при наличии аргументов. Тип аргументов может быть базовым, составным или доменным, либо это может быть ссылка на столбец таблицы.

В зависимости от языка реализации также может допускаться указание «псевдотипов», например, cstring. Псевдотипы показывают, что фактический тип аргумента либо определён не полностью, либо существует вне множества обычных типов SQL.

Ссылка на тип столбца записывается в виде имя_таблицы.имя_столбца%TYPE. Иногда такое указание бывает полезно, так как позволяет создать процедуру, независящую от изменений в определении таблицы.

выражение_по_умолчанию

Выражение, используемое для вычисления значения по умолчанию, если параметр не задан явно. Результат выражения должен сводиться к типу соответствующего параметра. Для всех входных параметров, следующих за параметром с определённым значением по умолчанию, также должны быть определены значения по умолчанию.

имя_языка

Имя языка, на котором реализована функция. Это может быть sql, c, internal либо имя процедурного языка, определённого пользователем, например, plpgsql. Стиль написания этого имени в апострофах считается устаревшим и требует точного совпадения регистра.

определение

Строковая константа, определяющая реализацию процедуры; её значение зависит от языка. Это может быть имя внутренней процедуры, путь к объектному файлу, команда SQL или код на процедурном языке.

объектный_файл, объектный_символ

Эта форма предложения AS применяется для динамически загружаемых процедур на языке C, когда имя процедуры в коде C не совпадает с именем процедуры в SQL. Строка объектный_файл задаёт имя файла, содержащего скомпилированную процедуру на C (данная команда воспринимает эту строку так же, как и LOAD).[1]

Вывод: в данном лабораторном практикуме были рассмотрены триггеры и пользовательские функции

Контрольные вопросы:

1 Что такое PL/pgSQL?

2 Зачем нужны триггеры?

3 Что такое процедура в бд?

Задания для выполнения:

1 Создать триггерную функцию на PL/pgSQL.

2 Создать процедуру с тремя параметрами.