Практическое задание 12

PostgreSQL

Триггеры

Триггеры

Триггер является указанием, что база данных должна автоматически выполнить заданную функцию, всякий раз, когда выполнен определённый тип операции. Триггеры можно использовать с таблицами, представлениями и с внешними таблицами. Подробную информацию о триггерной функциональности СУБД можно получить по адресу postgrespro.ru/docs/postgresql/15/triggers.

Для таблиц можно определять триггеры, которые будут срабатывать до или после любой из команд INSERT, UPDATE или DELETE; либо один раз для каждой модифицируемой строки, либо один раз для оператора SQL. Триггеры на UPDATE можно установить так, чтобы они срабатывали, только когда в предложении SET оператора UPDATE упоминаются определённые столбцы. Также триггеры могут срабатывать для операторов TRUNCATE. Если происходит событие триггера, для обработки этого события в установленный момент времени вызывается функция триггера.

Для представлений триггеры могут быть определены для выполнения вместо операций INSERT, UPDATE и DELETE. Такие триггеры INSTEAD OF вызываются единожды для каждой строки, которая должна быть изменена в этом представлении.

Триггерная функция должна быть создана до триггера. Она должна быть объявлена без аргументов и возвращать тип trigger.

После создания триггерной функции создаётся триггер с помощью CREATE TRIGGER. Одна и та же триггерная функция может быть использована для нескольких триггеров.

PostgreSQL предлагает как построчные, так и операторные триггеры. В случае построчного триггера триггерная функция вызывается один раз для каждой строки, затронутой оператором, запустившим триггер. Операторный же триггер, напротив, вызывается только один раз при выполнении соответствующего оператора, независимо от количества строк, которые он

затрагивает. В частности оператор, который не затрагивает никаких строк, всё равно приведёт к срабатыванию операторного триггера. Эти два типа триггеров также называют триггерами уровня строк и триггерами уровня оператора, соответственно. Триггеры на TRUNCATE могут быть определены только на уровне оператора, а не на уровне строк.

Триггеры также классифицируются в соответствии с тем, срабатывают ли они до, после или вместо операции. Они называются триггерами BEFORE, AFTER и INSTEAD OF, соответственно. Триггеры BEFORE уровня оператора срабатывают до того, как оператор начинает делать что-либо, тогда как триггеры AFTER уровня оператора срабатывают в самом конце работы оператора. Эти типы триггеров могут быть определены для таблиц, представлений или сторонних таблиц. Триггеры BEFORE уровня строки срабатывают непосредственно перед обработкой конкретной строки, в то время как триггеры AFTER уровня строки срабатывают в конце работы всего оператора (но до любого из триггеров AFTER уровня оператора). Эти типы триггеров могут определяться только для таблиц, в том числе сторонних, но не для представлений. Триггеры INSTEAD OF могут определяться только для представлений и только на уровне строк: они срабатывают для каждой строки сразу после того, как строка представления идентифицирована как подлежащая обработке.

Триггер AFTER — это триггер ограничения, его выполнение может быть отложено не до конца работы оператора, а до конца транзакции. В любом случае триггер выполняется в рамках той же транзакции, к которой относится вызвавший его оператор, поэтому если или оператор, или триггер вызывает ошибку, оба действия отменяются.

Триггерные функции, вызываемые триггерами операторов, должны всегда возвращать NULL. Триггерные функции, вызываемые триггерами строк, могут вернуть строку таблицы.

В определении триггера можно указать логическое условие WHEN, которое будет проверяться, чтобы посмотреть, нужно ли запускать триггер. В триггерах INSTEAD OF не поддерживается использование условий WHEN.

Как правило, триггеры BEFORE уровня строки используются для проверки или модификации данных, которые будут вставлены или изменены. Триггеры AFTER уровня строки наиболее разумно использовать для каскадного обновления данных в других таблицах или проверки согласованности сделанных изменений с данными в других таблицах. Если нет особых причин для выбора между триггерами BEFORE или AFTER, то триггер BEFORE предпочтительнее, так как не требует сохранения информации об операции до конца работы оператора.

Если триггерная функция выполняет команды SQL, эти команды могут заново запускать триггеры. Это известно как каскадные триггеры. Вполне возможно, что каскадные вызовы приведут к рекурсивному срабатыванию одного и того же триггера. Обязанность программиста не допускать бесконечную рекурсию в таких случаях.

Создание предваряющего триггера

В листинге 1 показан предваряющий триггер, который используется для задания значения строки перед обновлением.

Чтобы использовать этот триггер, создайте сначала в таблице TRANSACTION новый столбец под именем AskingPrice типа Number(7, 2).

В листинге 1 триггер New_Price определен фразой

Before Insert or Update of AcquisitionPrice ON TRANSACTION.

Эта фраза, являясь правильной, содержит неопределенность. Она означает «ПЕРЕД (любой) вставкой в транзакции или ПЕРЕД обновлением AcquisitionPrice в транзакции запустить триггер». Таким образом, триггер будет запущен при любой вставке или при обновлении столбца AcquisitionPrice. Обновления других столбцов не вызовут запуска триггера.

Листинг 1. Предваряющий триггер New_Price

```
CREATE OR REPLACE FUNCTION Change_AskingPrice()
RETURNS TRIGGER AS $$
BEGIN
```

```
  /* Устанавливаем новое значение AskingPrice перед вставкой или
обновлением */
NEW.AskingPrice := NEW.AcquisitionPrice * 2;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;


CREATE OR REPLACE TRIGGER New_Price
  BEFORE INSERT OR UPDATE OF Acquisitionprice
  ON TRANSACTION
  FOR EACH ROW
  EXECUTE FUNCTION change_askingprice();
```

Фраза FOR EACH ROW (для каждой строки) указывает, что данный триггер является строчным. Логика его работы проста: новое значение AskingPrice вычисляется как удвоенное новое значение AcquisitionPrice.

Префикс new доступен только для триггеров. Он обозначает новое значение столбца в команде INSERT или UPDATE. Так, new.AcquisitionPrice обозначает новое значение AcquisitionPrice (заданное пользователем). Для обновлений и удалений существует также префикс old, обозначающий значение столбца до выполнения команды UPDATE или DELETE.

Создайте триггерную функцию и триггер. Если не будет ошибок компиляции, триггер будет сохранен в базе данных и станет активным. Обновите столбец столбца AskingPrice будет задано триггером.

Создание завершающего триггера

В листинге 2 показан пример завершающего триггера. Логика его такова: View Ridge определяет произведение, предназначенное для продажи, как любое произведение, имеющее строку в таблице TRANSACTION с пустым значением CustomerID. Задача этого триггера — гарантировать, что такая строка в таблице TRANSACTION будет существовать, когда в базу данных будет записываться

произведение. Этот триггер, запускаемый при создании строки в таблице WORK, проверяет таблицу TRANSACTION и не производит никаких действий, если находит в ней подходящую строку.

Листинг 2. Завершающий триггер On_Work_Insert

```
CREATE OR REPLACE FUNCTION Try_Insert_Transaction_By_Work()
RETURNS TRIGGER AS $$
DECLARE
rowcountNUMERIC(2);
BEGIN
  /* Считаемдоступныестроки */
  SELECT Count(*) INTO rowcount
    FROM TRANSACTION
WHERE
CustomerID IS NULL AND
WorkID=NEW.WorkID;


  IF rowcount> 0 Then
/* Строка существует, ничего не предпринимаем */
    RAISE NOTICE 'Подходящая строка в таблице TRANSACTION существует
-- никаких действий не предпринято.';
    RETURN NULL;
  ELSE
    /* Нужно добавить новую строку */
INSERT INTO TRANSACTION (TransactionID, DateAcquired, WorkID)
      VALUES (nextval('transid'), current_date, NEW.WorkID);
    RETURN NEW;
  END IF;
END;
$$ LANGUAGE plpgsql;


CREATE OR REPLACE TRIGGER On_WORK_Insert
```

```
  AFTER INSERT
  ON WORK
  FOR EACH ROW
  EXECUTE FUNCTION Try_Insert_Transaction_By_Work();
```

Логика работы триггера элементарна. Он подсчитывает количество подходящих строк, и если это количество больше нуля, то ничего не предпринимается, а если оно равно нулю, в таблице TRANSACTION создается новая строка с соответствующими данными. Обратите внимание на то, как префикс new используется в части VALUES оператора INSERT.

Создание замещающего триггера

Замещающие триггеры используются для обновления представлений. Рассмотрим представление, определенное в листинге 3. Оно соединяет в себе четыре таблицы и, как представление соединения, не может быть обновлено без помощи соответствующего триггера.

Листинг 3. Определение представления CustomerPurchases

```
CREATE VIEW CustomerPurchases AS
  SELECT
CUSTOMER.Name AS CustName,
    Copy,
    Title,
ARTIST.Name AS ArtistName
  FROM CUSTOMER, TRANSACTION, WORK, ARTIST
WHERE
CUSTOMER.CustomerID = TRANSACTION.CustomerID AND
TRANSACTION.WorkID = WORK.WorkID AND
WORK.ArtistID = ARTIST.ArtistID;
```

Создайте представление и выведите содержащиеся в нем данные.

Обратите внимание на SELECT в определении этого представления. Оператор задает псевдонимы (Alias) для столбцов CUSTOMER.Name (псевдоним CustName) и ARTIST.Name (псевдоним ArtistName). Если бы это не было сделано, то в представлении оказалось бы два столбца с именем Name, а это недопустимо.

Исследуйте данные, выведенные в представлении, и подумайте, что должно произойти, когда пользователь попытается обновить столбец Title.

Замещающие триггеры позволяют разработчику задать действия, которые необходимо предпринять при попытке обновить представление в конкретном приложении. В листинге 4 показан пример такого триггера, который обрабатывает обновления столбца Title представления CustomerPurchases.
Поскольку замещающие триггеры, в отличие от предваряющих, не могут содержать фразу UPDATE OF, мы должны написать код, определяющий, был ли изменен столбец Title.

Листинг 4. Замещающий триггер Title_Update

```
CREATE OR REPLACE FUNCTION Update_Work_Title()
RETURNS TRIGGER AS $$
BEGIN
  /* Ничего не делаем, кроме случая, когда обновляется столбец Title
*/
IF NEW.Title = OLD.Title THEN
    RETURN NULL;
  END IF;
  UPDATE WORK
    SET Title = NEW.Title
    WHERE Title = OLD.Title;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE OR REPLACE TRIGGER Title_Update
  INSTEAD OF UPDATE
  ON CustomerPurchases
  FOR EACH ROW
  EXECUTE FUNCTION update_work_title();
```

Если пользователь обновляет столбец Title, то соответствующее изменение вносится в таблицу WORK. Обратите внимание, что запуск триггера происходит при обновлении представления CustomerPurchases, но само обновление производится в таблице WORK — одной из таблиц, на которой базируется это представление. Как раз для таких действий и предназначаются замещающие триггеры,

Обработка исключений

По умолчанию любая возникающая ошибка прерывает выполнение функции на PL/pgSQL и транзакцию, в которая она выполняется. Использование в блоке секции EXCEPTION позволяет перехватывать и обрабатывать ошибки. Синтаксис секции EXCEPTION расширяет синтаксис обычного блока:

```
[ <<метка>> ] [
DECLARE
объявления ]
BEGIN
операторы
EXCEPTION
    WHEN      условие      [      OR      условие...      ]THEN
операторы_обработчика
[    WHEN    условие    [    OR    условие    ...    ]    THEN
операторы_обработчика
      ... ]
END;
```

Если ошибок не было, то выполняются все операторы блока и управление переходит к следующему оператору после END. Но если при выполнении оператора происходит ошибка, то дальнейшая обработка прекращается и

управление переходит к списку исключений в секции EXCEPTION. В этом списке ищется первое исключение, условие которого соответствует ошибке. Если исключение найдено, то выполняются соответствующие операторы_обработчика и управление переходит к следующему оператору после END. Если исключение не найдено, то ошибка передаётся наружу, как будто секции EXCEPTION не было. При этом ошибку можно перехватить в секции EXCEPTION внешнего блока. Если ошибка так и не была перехвачена, то обработка функции прекращается.

При обработке исключений часто бывает необходимым получить детальную информацию о произошедшей ошибке. Для этого в PL/pgSQL есть два способа: использование специальных переменных и команда GET STACKED

Внутри секции EXCEPTION специальная переменная SQLSTATE содержит код ошибки, для которой было вызвано исключение (список возможных кодов ошибок приведён в документации). Специальная переменная SQLERRM содержит сообщение об ошибке, связанное с исключением. Эти переменные являются неопределёнными вне секции EXCEPTION.

Также в обработчике исключения можно получить информацию о текущем исключении командой GET STACKED DIAGNOSTICS, которая имеет вид:

```
GET STACKED DIAGNOSTICS переменная{ = | := } элемент [ , ... ];
```

Каждый элемент представляется ключевым словом, указывающим, какое значение состояния нужно присвоить заданной переменной (она должна иметь подходящий тип данных, чтобы принять его). Все доступные в настоящее время элементы состояния перечислены в документации.

Подробную информацию об обработке исключений и ошибок вы можете получить по адресу postgrespro.ru/docs/postgresql/15/plpgsql-