|  | | МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ | | | | | | |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | |  | | |  | | |
|  | «Пермский государственный национальный  исследовательский университет» | | | | | | | |  |
|  | | | |  | | |  | | |
|  | | | ОТЧЕТ  О ВЫПОЛНЕНИИ ЗАДАНИЯ №4  ПО ДИСЦИПЛИНЕ БД И СУБД  по теме:  «Триггеры» | | | | |  | |
|  | | | |  | | |  | | |
|  | |  | | |  | Работу  Выполнил  Студент гр.ПМИ-2-20,  3 курс  Пуховкин В.И. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись)  «\_\_\_\_\_»\_\_\_\_\_\_\_2023г | | |  |
|  | | | | Пермь 2023 | | |  | | |

**1. Задание**

Решить проблему рассогласованности спец. таблицы и отслеживаемой таблицы в случае, если в последнюю была проведена вставка в обход разработанной ранее хранимой процедуры (ХП) новой записи с идентификатором, превышающим предыдущий максимум, или было проведено обновление идентификаторов, приведшее к превышению максимума. Для этого автоматически создавать триггер для таблицы, имя которой передаётся в ХП в качестве параметра для каждой новой уникальной пары 'имя таблицы' + 'имя столбца', который только в описанных выше случаях превышения обновляет соответствующее текущее максимальное значение в спец. таблице. Имя триггера, помимо осмысленной части, должно содержать GUID.

Провести тестирование корректности работы программы.

**2. Этапы и описание реализации.**

**Описание алгоритма:**

1. Создание спец. таблицы, в которой есть столбцы `id`, `имя таблицы`, `имя столбца` и `текущее максимальное значение`;
2. Создание функции, которая будет обновлять поле `текущее максимальное значение` в спец. таблице;
3. Внутри ХП выполняем проверку, на наличие в специальной таблице записи с полученными на вход данными об имени таблицы и имени столбца. Если ХП находит данную запись, то инкрементируем значение, находящееся в столбце “CurrentMaxValue” и переходим к шагу 5.

В случае, если такой записи нет, переходим к шагу 4.

1. ХП ищет максимальное число в запрашиваемой таблице и затем инкрементирует данное число и переходим к шагу 5. Если в запрашиваемой таблице отсутствуют значения, то берём число 1 число и переходим к шагу 5.
2. Возвращаем полученное значение.
3. Каждый EXECUTE содержит триггер, который вызывается после соответствующей операции (в нашем случае один после INSERT, другой после UPDATE), если вызвался один из триггеров, переходим к шагу 7.
4. После вызова триггера вызывается функция, которая обновляет значение поля `текущее максимальное значение`
5. Если имеются последующие тестовые команды, то выполнятся они, иначе работа SQL- скрипта завершается.

**Описание реализации:**

1. **Создание функции:**

CREATE [ OR REPLACE ] FUNCTION

***имя*** ( [ [ ***режим\_аргумента*** ] [ ***имя\_аргумента*** ] ***тип\_аргумента*** [ { DEFAULT | = } ***выражение\_по\_умолчанию*** ] [, ...] ] )

[ RETURNS ***тип\_результата***

| RETURNS TABLE ( ***имя\_столбца*** ***тип\_столбца*** [, ...] ) ]

LANGUAGE ***имя\_языка***

CREATE OR REPLACE FUNCTION создаёт новую функцию, либо заменяет определение уже существующей.

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

Чтобы заменить текущее определение существующей функции, используйте команду CREATE OR REPLACE FUNCTION. Но учтите, что она не позволяет изменить имя или аргументы функции (если попытаться сделать это, на самом деле будет создана новая, независимая функция). Кроме того, CREATE OR REPLACE FUNCTION не позволит изменить тип результата существующей функции. Чтобы сделать это, придётся удалить функцию и создать её заново. (Это означает, что если функция имеет выходные параметры (OUT), то изменить типы параметров OUT можно, только удалив функцию.)

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

Если вы удалите и затем вновь создадите функцию, новая функция станет другой сущностью, отличной от старой; вам потребуется так же удалить существующие правила, представления, триггеры и т. п., ссылающиеся на старую функцию. Поэтому, чтобы изменить определение функции, сохраняя ссылающиеся на неё объекты, следует использовать CREATE OR REPLACE FUNCTION.

BEGIN и END показывают начало и конец блока и обрамляются символами “$$”.

В качестве имени языка выбираем plpgsql, это нам необходимо для использования его функционала.

1. **Выполнение динамически формируемых команд:**

EXECUTE ***строка-команды*** [ INTO [STRICT] ***цель*** ] [ USING ***выражение*** [, ... ] ];

Оператор EXECUTE предусмотрен для исполнения динамических команд, где ***строка-команды*** это выражение, формирующее строку (типа text) с текстом команды, которую нужно выполнить. Необязательная ***цель*** — это переменная-запись, переменная-кортеж или разделённый запятыми список простых переменных и полей записи/кортежа, куда будут помещены результаты команды. Необязательные выражения в USING формируют значения, которые будут вставлены в команду.

Все необходимые значения переменных должны быть вставлены в командную строку при её построении, либо нужно использовать параметры, как описано ниже.

Также, нет никакого плана кеширования для команд, выполняемых с помощью EXECUTE. Вместо этого план создаётся каждый раз при выполнении. Таким образом, строка команды может динамически создаваться внутри функции для выполнения действий с различными таблицами и столбцами.

Предложение INTO указывает, куда должны быть помещены результаты SQL-команды, возвращающей строки. Если используется переменная строкового типа или список переменных, то они должны в точности соответствовать структуре результата запроса (когда используется переменная типа record, она автоматически приводится к строковому типу результата запроса). Если возвращается несколько строк, то только первая будет присвоена переменной(ым) в INTO. Если не возвращается ни одной строки, то присваивается NULL. Без предложения INTO результаты запроса отбрасываются.

С указанием STRICT запрос должен вернуть ровно одну строку, иначе выдаётся сообщение об ошибке.

В тексте команды можно использовать значения параметров, ссылки на параметры обозначаются как $1, $2 и т. д. Эти символы указывают на значения, находящиеся в предложении USING. Такой метод зачастую предпочтительнее, чем вставка значений в команду в виде текста: он позволяет исключить во время исполнения дополнительные расходы на преобразования значений в текст и обратно, и не открывает возможности для SQL-инъекций, не требуя применять экранирование или кавычки для спецсимволов.

FORMAT определяет возвращаемое значение.

QUOTE\_IDENT ( text ) → text

Преобразует аргумент в строку, подходящую для использования в качестве идентификатора в SQL-операторе. При необходимости идентификатор заключается в кавычки (например, если он содержит символы, недопустимые в открытом виде, или буквы в разных регистрах). Если переданная строка содержит кавычки, они дублируются.

1. **Получение максимального значения в столбце:**

\*SELECT COALESCE(description, short\_description, '(none)') ...

Функция COALESCE возвращает первый попавшийся аргумент, отличный от NULL. Если же все аргументы равны NULL, результатом тоже будет NULL. Это часто используется при отображении данных для подстановки некоторого значения по умолчанию вместо значений NULL.

Этот \* запрос вернёт значение description, если оно не равно NULL, либо short\_description, если оно не NULL, и строку (none), если оба эти значения равны NULL.

COALESCE вычисляет только те аргументы, которые необходимы для получения результата; то есть, аргументы правее первого отличного от NULL аргумента не вычисляются.

В нашей задаче в поле description мы используем параметр поиска максимального (max(%I)+1) и значение 1 для short\_description.

1. **Создание триггера:**

CREATE [ CONSTRAINT ] TRIGGER ***имя*** { BEFORE | AFTER | INSTEAD OF } { ***событие*** [ OR ... ] }

ON ***имя\_таблицы***

[ FROM ***ссылающаяся\_таблица*** ]

[ NOT DEFERRABLE | [ DEFERRABLE ] [ INITIALLY IMMEDIATE | INITIALLY DEFERRED ] ]

[ FOR [ EACH ] { ROW | STATEMENT } ]

[ WHEN ( ***условие*** ) ]

EXECUTE PROCEDURE ***имя\_функции*** ( *аргументы* )

Здесь допускается ***событие***:

INSERT

UPDATE [ OF ***имя\_столбца*** [, ... ] ]

DELETE

TRUNCATE

CREATE TRIGGER создаёт новый триггер. Триггер будет связан с указанной таблицей, представлением или сторонней таблицей и будет выполнять заданную функцию ***имя\_функции*** при определённых событиях.

Триггер можно настроить так, чтобы он срабатывал до операции со строкой (до проверки ограничений и попытки выполнить INSERT, UPDATE или DELETE) или после её завершения (после проверки ограничений и выполнения INSERT, UPDATE или DELETE), либо вместо операции (при добавлении, изменении и удалении строк в представлении). Если триггер срабатывает до или вместо события, он может пропустить операцию с текущей строкой, либо изменить добавляемую строку (только для операций INSERT и UPDATE). Если триггер срабатывает после события, он «видит» все изменения, включая результат действия других триггеров.

Триггер с пометкой FOR EACH ROW вызывается один раз для каждой строки, изменяемой в процессе операции. Например, операция DELETE, удаляющая 10 строк, приведёт к срабатыванию всех триггеров ON DELETE в целевом отношении 10 раз подряд, по одному разу для каждой удаляемой строки. Триггер с пометкой FOR EACH STATEMENT, напротив, вызывается только один раз для конкретной операции, вне зависимости от того, как много строк она изменила (в частности, при выполнении операции, изменяющей ноль строк, всё равно будут вызваны все триггеры FOR EACH STATEMENT). Заметьте, что при выполнении INSERT с предложением ON CONFLICT DO UPDATE сработают оба триггера уровня операторов, для INSERT и для UPDATE.

Триггеры, срабатывающие в режиме INSTEAD OF, должны быть помечены FOR EACH ROW и могут быть определены только для представлений. Триггеры BEFORE и AFTER для представлений должны быть помечены FOR EACH STATEMENT.

Кроме того, триггеры можно определить и для команды TRUNCATE, но только типа FOR EACH STATEMENT.

Кроме того, в определении триггера можно указать логическое условие WHEN, которое определит, вызывать триггер или нет. В триггерах на уровне строк условия WHEN могут проверять старые и/или новые значения столбцов в строке. Триггеры на уровне оператора так же могут содержать условие WHEN, хотя для них это не столь полезно, так как в этом условии нельзя ссылаться на какие-либо значения в таблице.

Если для одного события определено несколько триггеров одного типа, они будут срабатывать в алфавитном порядке их имён.

Когда указывается параметр CONSTRAINT, эта команда создаёт *триггер ограничения*. Он подобен обычным триггерам, но отличается тем, что время его срабатывания можно изменить командой SET CONSTRAINTS. Триггеры ограничений должны работать в режиме AFTER ROW. Они могут срабатывать либо в конце оператора, вызвавшего целевое событие, либо в конце содержащей его транзакции; в последнем случае они называются *отложенными*. Срабатывание ожидающего отложенного триггера можно вызвать немедленно, воспользовавшись командой SET CONSTRAINTS. Предполагается, что триггеры ограничений будут генерировать исключения при нарушении ограничений.

SELECT не изменяет никакие строки, поэтому создавать триггеры для SELECT нельзя. В случае подобной потребности будут более уместны правила и представления.

**В нашей задачи, используется следующее описание триггера:**

CREATE TRIGGER %s

AFTER INSERT ON %s

REFERENCING NEW TABLE AS NEW

FOR EACH STATEMENT

EXECUTE FUNCTION update\_max\_in\_test\_table(%s,%s)

Данное описание мы используем по целому ряду причин:

* Триггер должен сработать, после того, как выполнится операция INSERT (UPDATE);
* Нам нужно, чтобы триггер вызвался не только для изменения одной строки, а для изменения в определённой таблице в целом;
* Для анализа изменений, нам нужно где-то запомнить новые, модифицированные значения.
* Также триггер должен вызываться один раз, не зависимо от того, сколько строк мы изменим в таблице.

Из первых двух получаем связку AFTER и REFERENСING

Далее, для запоминания модифицированных значений с REFERNCING используем NEW TABLE AS NEW

А для того, чтобы триггер срабатывал не на каждое изменение, а только один раз, на общее изменение, используем FOR EACH STATEMENT.

**Работа с переменными:**

Для использования переменной, ее надо предварительно «объявить», выполнив команду **DECLARE** языка SQL. Команда **DECLARE** имеет следующий формат: **DECLARE** *<имя\_переменной> <тип\_переменной>*, где *<имя\_переменной>* - это название объявляемой переменной, а *<тип\_переменной>* - тип данных, хранимых в ней.

Поддерживаются следующие основные типы данных:

* *Geometry*- пространственный объект;
* *BigInt* - целочисленное значение в диапазоне от -263-1 до 263 (от - 9,223,372,036,854,775,808 по +9,223,372,036,854,775,807);
* *Int* - целочисленное значение в диапазоне от -231 (-2,147,483,648) to 231-1 (2,147,483,647);
* *SmallInt* - целочисленное значение в диапазоне от -215 (-32,768) to 215-1 (32,767);
* *Double*, *Float*, *Real*- число с плавающей точкой;
* *char*, *varchar*- строковое значение.

**Строковая функция экранирования:**

quote\_ident(*string* text)

Переданная строка оформляется для использования в качестве идентификатора в SQL -операторе. При необходимости идентификатор заключается в кавычки (например, если он содержит символы, недопустимые в открытом виде, или буквы в разном регистре). Если переданная строка содержит кавычки, они дублируются.

Пример: quote\_ident('Foo bar') получим "Foo bar"

**4. Тестирование**

До начала тестирования, уже реализована хранимая процедура “ Search\_Inc” с триггерами, специальная таблица, в нашем скрипте это - “ SpecTable” и функция, обновляющая значение в спец. таблице.

--Создание таблицы с одним полем

CREATE TABLE test

(

id INT NOT NULL

);

--Вывод таблицы

SELECT \* FROM spectable;

--Добавляем знчение 12

INSERT INTO test VALUES (12);

--Запускаем ХП от test

SELECT Search\_Inc('test','id');

--Добавляем знчение 20

INSERT INTO test VALUES (20);

--Добавляем сразу 2 знаения

INSERT INTO test VALUES (40), (30);

--Запускаем ХП от test

SELECT Search\_Inc('test','id');

--Вставляем знаяения, которые меньше текущих

INSERT INTO test VALUES (1), (2);

--Изменение данных при помщи Update

UPDATE test

SET id = 50;

SELECT \* FROM spectable ;

--Запускаем ХП от test

SELECT Search\_Inc('test','id');

--Вывод значений таблицы

SELECT \* FROM spectable;

--Создаем новую таблицу с столбцами 'num\_value1', 'num\_value2'

CREATE TABLE test2

(

num\_value1 INT NOT NULL,

num\_value2 INT NOT NULL

);

--Вставка значений

INSERT INTO test2 VALUES (20, 30);

--Запус ХП от num\_value2

SELECT Search\_Inc('test2', 'num\_value2');

--Запус ХП от num\_value1

SELECT Search\_Inc('test2','num\_value1');

--распечатка таблицы test2

SELECT \* FROM test2;

-- распечатка спец таблицы

SELECT \* FROM spectable;

--вызов ХП с параметрами 'test2' и 'num\_value1'

SELECT Search\_Inc('test2','num\_value1');

--Вставка значений меньше имеющихся в таблице

INSERT INTO test2 VALUES(2, 13);

SELECT \* FROM spectable;

--Удаление ХП

DROP FUNCTION Search\_Inc(TableNameIN varchar, ColumnNameIN varchar);

--Удаление таблиц

DROP TABLE SpecTable;

DROP TABLE test;

DROP TABLE test2;

--Удаление вспомогательной функции

drop function update\_max\_in\_test\_table();