|  | | МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ | | | | | | |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | |  | | |  | | |
|  | «Пермский государственный национальный  исследовательский университет» | | | | | | | |  |
|  | | | |  | | |  | | |
|  | | | ОТЧЕТ  О ВЫПОЛНЕНИИ ЗАДАНИЯ №5  ПО ДИСЦИПЛИНЕ БД И СУБД  по теме:  «Метаданные» | | | | |  | |
|  | | | |  | | |  | | |
|  | |  | | |  | Работу  Выполнил  Студент гр.ПМИ-4-20,  3 курс  Пуховкин В.И. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись)  «\_\_\_\_\_»\_\_\_\_\_\_\_2023г | | |  |
|  | | | | Пермь 2023 | | |  | | |

# **Задание**

Решить проблему неконтролируемых ошибок разработанной ранее хранимой процедуры (ХП) в случае, если в качестве параметров передана несуществующая таблица, несуществующий столбец существующей таблицы или столбец, который не имеет целочисленный тип, посредством проверки корректности параметров на основе метаданных базы данных. В случае некорректных параметров самостоятельно порождать ошибку.

Внести такие изменения в код ХП, чтобы имя триггера формировалось по шаблону "{имя таблицы}\_{имя столбца}\_{номер триггера}", где номер триггера выбирается как количество существовавших ранее триггеров для этой таблицы + 1. Перед созданием требуется проверить существование другого объекта в базе данных с предлагаемым именем. В случае, если такой объект уже есть, требуется сформировать имя любым другим образом.

Провести тестирование корректности работы программы.

# **2. Этапы и описание реализации**

**Описание алгоритма:**

1. Создание исключений, которые будут срабатывать при выполнении неверных действий.
2. В разработке исключений применяем метаданные
3. При выполнении ХП считаем количество вызванных триггеров, для того, чтобы создать новое имя для нового триггера по следующему принципу: "{имя таблицы}\_{имя столбца}\_{номер триггера}". Если такое имя уже существует перейти к шагу 4 иначе перейти к шагу 5.
4. В случае, если триггер с таким названием существует, то мы добавляем в его название текущий момент времени выполнения операции.
5. Создание триггера, обработка триггерной функции, завершение работы ХП.

**Описание реализации:**

1. **Работа с метаданными:**

information\_schema\_catalog\_name

Таблица information\_schema\_catalog\_name всегда содержит одну строку и один столбец с именем текущей базы данных (текущий каталог, в терминологии SQL).

В нашем скрипте мы используем:

information\_schema.tables

Представление tables показывает все таблицы и представления, определённые в текущей базе данных. В нём показываются только те таблицы и представления, к которым имеет доступ текущий пользователь (являясь их владельцем или имея некоторые права).

information\_schema.columns

Представление columns содержит информацию обо всех столбцах таблиц (или столбцах представлений) в базе данных. Системные столбцы (ctid и т. д.) в нём не отображаются. В нём показываются только те столбцы, к которым имеет доступ текущий пользователь (являясь владельцем или имея некоторые права).

1. **Создание исключений:**

Оператор ASSERT представляет удобное средство вставлять отладочные проверки в функции PL/pgSQL.

ASSERT условие [ , сообщение ];

Здесь условие — это логическое выражение, которое, как ожидается, должно быть всегда истинным; если это так, оператор ASSERT больше ничего не делает. Если же оно возвращает ложь или NULL, этот оператор выдаёт исключение ASSERT\_FAILURE. (Если ошибка происходит при вычислении условия, она выдаётся как обычная ошибка.)

Если в нём задаётся необязательное сообщение, результат этого выражения (если он не NULL) заменяет сообщение об ошибке по умолчанию «assertion failed» (нарушение истинности), в случае, если условие не выполняется. В обычном случае, когда условие утверждения выполняется, выражение сообщения не вычисляется.

Проверку утверждений можно включить или отключить с помощью конфигурационного параметра plpgsql.check\_asserts, принимающего логическое значение; по умолчанию она включена (on). Если этот параметр отключён (off), операторы ASSERT ничего не делают.

Учтите, что оператор ASSERT предназначен для выявления программных дефектов, а не для вывода обычных ошибок (для этого используется оператор RAISE, описанный выше).

В нашей программе мы описываем исключения данным способом:

ASSERT EXISTS(SELECT \* FROM information\_schema.tables

WHERE table\_name = TableNameIN), 'Такой таблицы не существует';

ASSERT EXISTS(SELECT \* FROM information\_schema.columns

WHERE table\_name = TableNameIN AND column\_name = ColumnNameIN), 'Такого столбца не существует';

ASSERT EXISTS(SELECT \* FROM information\_schema.columns

WHERE table\_name = TableNameIN AND column\_name = ColumnNameIN AND data\_type = 'integer'), 'Не целочисленный тип данных';

Тем самым мы осуществляем проверку на правильность входных данных, используя метаданные.

1. **Вывод текущей даты и времени для имени триггера:**

В скрипте используется SELECT CURRENT\_TIMESTAMP

current\_timestamp → timestamp with time zone

Текущая дата и время (на момент начала транзакции);

# **Тестирование**

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

-- ОШИБКА: Такой таблицы не существует

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

-- ОШИБКА: Такого столбца не существует

SELECT Search\_Inc('spec', 'id1');

-- ОШИБКА: Не целочисленный тип данных

SELECT Search\_Inc('spec', 'tablename');

-- Создание таблиц и триггеров

CREATE TABLE test(

id integer,

num\_value1 integer

);

CREATE OR REPLACE FUNCTION trig\_function()

RETURNS TRIGGER

AS

$$

BEGIN

RETURN NULL;

END

$$

LANGUAGE plpgsql;

CREATE TRIGGER trigger1

AFTER DELETE ON test

EXECUTE FUNCTION trig\_function();

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

SELECT trigger\_name FROM information\_schema.triggers

WHERE event\_object\_table = 'test';

--новое имя

CREATE TABLE test2(

id integer,

num\_value1 integer

);

--Создаем триггер с уже имеющимся названием

CREATE TRIGGER test2\_id\_2

AFTER DELETE ON test2

EXECUTE FUNCTION trig\_function();

SELECT Search\_Inc('test2', 'id');

SELECT trigger\_name FROM information\_schema.triggers

WHERE event\_object\_table = 'test2';

DROP TABLE spec;

DROP TABLE test;

DROP TABLE test2;

DROP FUNCTION trig\_function();

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

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