|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | 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  Хайруллин Ю.Р. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись)  «\_\_\_\_\_»\_\_\_\_\_\_\_2022г | | |  | Проверил  Старший преподаватель кафедры МОВС  Постаногов И.С. \_\_\_\_\_\_\_\_\_\_  (подпись)  «\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_2022г | | |  |
|  | | | | Пермь 2022 | | |  | | |

**Содержание:**

1. Задание (Что надо сделать?);
2. Анализ предметной области (без терминов из реляционной модели) + предлагаемая схема БД;
3. Этапы и описание реализации (Что сделали?);
4. Тестирование (Как тестировали?).

**1. Задание (Что надо сделать?)**

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.

**2.** **Анализ предметной области (без терминов из реляционной модели) + предлагаемая схема БД.**

В рамках данного задания предметной областью является работа с БД и написание SQL скрипта, который удовлетворяет всем требованиям, заданным в пункте 1.

Схемы БД как таковой нет, но создаётся спец. таблица, в которой есть столбцы `id`, `имя таблицы`, `имя столбца` и `текущее максимальное значение`.

Также создаются триггеры и вспомогательная функция-обработчик, для их успешной работы

**3. Этапы и описание реализации (Что сделали?).**

**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 таблицы:**

Для создания всех таблиц в рамках данного задания, использовалась следующая команда:

CREATE TABLE [ IF NOT EXISTS ] **имя\_таблицы** ( [

{ **имя\_столбца тип\_данных** [ **ограничение\_столбца** [ ... ] ]}

] )

Здесь **ограничение\_столбца**:

[ CONSTRAINT **имя\_ограничения** ]

{ NOT NULL |

UNIQUE **параметры\_индекса** |

PRIMARY KEY **параметры\_индекса** |}

CREATE TABLE создаёт новую, изначально пустую таблицу в текущей базе данных. Владельцем таблицы будет пользователь, выполнивший эту команду.

Если задано имя схемы (например, CREATE TABLE myschema.mytable ...), таблица создаётся в указанной схеме, в противном случае — в текущей. Временные таблицы существуют в специальной схеме, так что при создании таких таблиц имя схемы задать нельзя. Имя таблицы должно отличаться от имён других таблиц, последовательностей, индексов, представлений или сторонних таблиц в этой схеме.

CREATE TABLE также автоматически создаёт составной тип данных, соответствующий одной строке таблицы. Таким образом, имя таблицы не может совпадать с именем существующего типа в этой же схеме.

Необязательные предложения ограничений задают ограничения (проверки), которым должны удовлетворять добавляемые или изменяемые строки, чтобы операция добавления или изменения была выполнена успешно. Ограничение представляет собой SQL-объект, помогающий некоторым способом определить множество допустимых значений в таблице.

Определить ограничения можно двумя способами: в виде ограничений таблицы и в виде ограничений столбца. Ограничение столбца определяется как часть определения столбца, а ограничение таблицы не привязывается к конкретному столбцу и может задействовать несколько столбцов. Любые ограничения столбцов можно также записать в виде ограничения таблицы, они введены просто для удобства записи в случаях, когда ограничение затрагивает только один столбец.

1. **Добавление данных в таблицу:**

INSERT INTO **имя\_таблицы** [ AS **псевдоним** ] [ ( **имя\_столбца** [, ...] ) ]

{ DEFAULT VALUES | VALUES ( { **выражение** | DEFAULT } [, ...] ) [, ...] | **запрос** }

[ ON CONFLICT [ **объект\_конфликта** ] **действие\_при\_конфликте** ]

[ RETURNING \* | **выражение\_результата** [ [ AS ] **имя\_результата** ] [, ...] ]

INSERT добавляет строки в таблицу. Эта команда может добавить одну или несколько строк, сформированных выражениями значений, либо ноль или более строк, выданных дополнительным запросом.

Имена целевых столбцов могут перечисляться в любом порядке. Если список с именами столбцов отсутствует, по умолчанию целевыми столбцами становятся все столбцы заданной таблицы; либо первые N из них, если только N столбцов поступает от предложения VALUES или запроса. Значения, получаемые от предложения VALUES или запроса, связываются с явно или неявно определённым списком столбцов слева направо.

Все столбцы, не представленные в явном или неявном списке столбцов, получат значения по умолчанию, если для них заданы эти значения, либо NULL в противном случае.

Если выражение для любого столбца выдаёт другой тип данных, система попытается автоматически привести его к нужному.

1. **Изменение данных в таблице:**

UPDATE **имя\_таблицы** SET { **имя\_столбца** = **{ выражение** }}

[ FROM **элемент\_FROM** [, ...] ]

[ WHERE **условие** | WHERE CURRENT OF **имя\_курсора** ]

[ RETURNING \* | **выражение\_результата** [ [ AS **] имя\_результата** ] [, ...] ]

UPDATE изменяет значения указанных столбцов во всех строках, удовлетворяющих условию. В предложении SET должны указываться только те столбцы, которые будут изменены; столбцы, не изменяемые явно, сохраняют свои предыдущие значения.

Изменить строки в таблице, используя информацию из других таблиц в базе данных, можно двумя способами: применяя вложенные запросы или указав дополнительные таблицы в предложении FROM. Выбор предпочитаемого варианта зависит от конкретных обстоятельств.

Предложение RETURNING указывает, что команда UPDATE должна вычислить и возвратить значения для каждой фактически изменённой строки. Вычислить в нём можно любое выражение со столбцами целевой таблицы и/или столбцами других таблиц, упомянутых во FROM. При этом в выражении будут использоваться новые (изменённые) значения столбцов таблицы. Список RETURNING имеет тот же синтаксис, что и список результатов SELECT.

Для выполнения этой команды необходимо иметь право UPDATE для таблицы, или как минимум для столбцов, перечисленных в списке изменяемых. Также необходимо иметь право SELECT для всех столбцов, значения которых считываются в выражениях или условии.

1. **Удаление таблиц:**

DROP TABLE [ IF EXISTS ] ***имя*** [, ...] [ CASCADE | RESTRICT ]

DROP TABLE удаляет таблицы из базы данных. Удалить таблицу может только её владелец, владелец схемы или суперпользователь. Чтобы опустошить таблицу, не удаляя её саму, вместо этой команды следует использовать [DELETE](https://postgrespro.ru/docs/postgresql/9.6/sql-delete) или [TRUNCATE](https://postgrespro.ru/docs/postgresql/9.6/sql-truncate).

DROP TABLE всегда удаляет все индексы, правила, триггеры и ограничения, существующие в целевой таблице. Однако чтобы удалить таблицу, на которую ссылается представление или ограничение внешнего ключа в другой таблице, необходимо дополнительно указать CASCADE. (С указанием CASCADE зависимое представление удаляется полностью, тогда как в случае с ограничением внешнего ключа удаляется именно это ограничение, а не вся таблица, к которой оно относится.)

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. **Условный оператор IF:**

IF ***логическое-выражение*** THEN

***операторы***

END IF;

IF-THEN это простейшая форма IF. Операторы между THEN и END IF выполняются, если условие (***логическое-выражение***) истинно. В противном случае они опускаются.

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. **Удаление функции:**

DROP FUNCTION name

DROP FUNCTION удаляет определение существующей функции.

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.

1. **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.

1. **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 (Как тестировали?).**

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.

1. **Вставляем значение в таблицу в обход ранее реализованной ХП, которое больше, чем предыдущее:**

Создаём таблицу test с полем id, вставляем в неё значение 12.

Далее запускаем ХП от данной таблицы, получим значение 13.

Далее, вставим в поле id значение 20

Ожидание: Вызовется триггер INSERT, который изменит значение в поле CurrentMaxValue в таблице “ SpecTable”, значение станет = 20.

Результат: Вызвался триггер INSERT, который изменил значение в поле CurrentMaxValue в таблице “ SpecTable”, значение стало = 20.

1. **Вставляем значение в таблицу в обход ранее реализованной ХП, которое меньше, чем предыдущее:**

Вставляем в ранее созданную таблицу test в поле id значение 10

Ожидание: Вызовется триггер INSERT, который не изменит значение в поле CurrentMaxValue в таблице “ SpecTable”, значение останется = 20.

Результат: Вызвался триггер INSERT, который не изменил значение в поле CurrentMaxValue в таблице “ SpecTable”, значение останется = 20.

1. **Вставляем два значения в таблицу в обход ранее реализованной ХП:**

Вставляем в ранее созданную таблицу test в поле id значение (40), (30)

Ожидание: Вызовется триггер INSERT, который изменит значение в поле CurrentMaxValue в таблице “ SpecTable”, значение станет = 40, так как это наибольшее среди вставляемых;

Результат: Вызвался триггер INSERT, который изменил значение в поле CurrentMaxValue в таблице “ SpecTable”, значение стало = 40.

1. **Обновляем значение в таблице при помощи Update в обход ранее реализованной ХП:**

Выполняем следующий SQL – код :

UPDATE test

SET id = 50;

Ожидание: Вызовется триггер UPDATE, который изменит значение в поле CurrentMaxValue в таблице “ SpecTable”, значение станет = 50.

Результат: Вызвался триггер UPDATE, который изменил значение в поле CurrentMaxValue в таблице “ SpecTable”, значение стало = 50.

1. **Изменение значений в таблице с несколькими полями в обход ранее реализованной ХП:**

Создаём таблицу test2 с полями num\_value1 и num\_value2;

Заполняем поля значениями 20 и 30;

Запускаем ХП от test2 и num\_value2, получим значение 31

Далее выполним: INSERT INTO test2 VALUES (40, 50);

Ожидание: Вызовется триггер INSERT, который изменит значение в поле CurrentMaxValue в таблице “ SpecTable”, значение станет = 50;

Результат: Вызвался триггер INSERT, который изменил значение в поле CurrentMaxValue в таблице “ SpecTable”, значение стало = 50.

1. **Обновление значений в таблице с несколькими полями в обход ранее реализованной ХП при помощи Update:**

Выполним следующий SQL-код

UPDATE test2

SET num\_value2 = 80

WHERE num\_value2 = 50;

Ожидание: Вызовется триггер UPDATE, который изменит значение в поле CurrentMaxValue в таблице “ SpecTable”, значение станет = 80;

Результат: Вызвался триггер UPDATE, который изменил значение в поле CurrentMaxValue в таблице “ SpecTable”, значение стало = 80.