**Рекомендации по выполнению лабораторной работы**

<https://postgrespro.ru/docs/postgresql/14/plpgsql-overview>

PL/pgSQL это процедурный язык для СУБД PostgreSQL. Целью проектирования PL/pgSQL было создание загружаемого процедурного языка, который:

* используется для создания функций, процедур и триггеров,
* добавляет управляющие структуры к языку SQL,
* может выполнять сложные вычисления,
* наследует все пользовательские типы, функции, процедуры и операторы,
* может быть определён как доверенный язык,
* прост в использовании.

Функции PL/pgSQL могут использоваться везде, где допустимы встроенные функции. Например, можно создать функции со сложными вычислениями и условной логикой, а затем использовать их при определении операторов или в индексных выражениях.

В версии PostgreSQL 9.0 и выше PL/pgSQL устанавливается по умолчанию. Тем не менее это по-прежнему загружаемый модуль и администраторы, особо заботящиеся о безопасности, могут удалить его при необходимости.

Для создания процедур и триггеров необходимо, чтобы у пользователя были права на создание этих объектов, для этого возможно будет необходимо «дать» эти права.

<https://postgrespro.ru/docs/postgresql/14/sql-createprocedure> - теория по PostgreSQL

Пример: execute immediate 'grant create any table to system'; - дали права на создание любой таблице пользователю system. Для создание процедуры нужно выполнить оператор CREATE OR REPLACE PROCEDURE, который имеет следующий 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'

| тело\_sql

} ......

Чтобы выполнить процедуру, воспользуйтесь командой CALL.

<https://postgrespro.ru/docs/postgresql/14/sql-call>

**ПРИМЕР**

CREATE PROCEDURE insert\_data(a integer, b integer)

LANGUAGE SQL

AS $$

INSERT INTO tbl VALUES (a);

INSERT INTO tbl VALUES (b);

$$;

**ПРИМЕР ВЫЗОВА**

CALL insert\_data(1, 2);

Чтобы выводить результаты можно воспользоваться командой Raise notice. Пример вывода:

do $$

Begin

Raise notice 'Hello World!';

End;

$$

<https://postgrespro.ru/docs/postgresql/14/ddl-constraints>

Ограничения целостности – это правила, определяющие допустимость и достоверность хранимых и вводимых данных. Ограничения целостности основаны на особенностях модели предметной области, а также обусловлены особенностями реляционной модели. Выделяют статические и динамические ограничения целостности. К первой группе относят:

* ограничение на определенность значения атрибута (NOT NULL);
* ограничения на уникальность значения атрибутов (UNIQUE);
* первичный ключ;
* внешний ключ;
* ограничение на диапазон значений и др. (задается предикатом).

Статические ограничения целостности реализуются в большинстве современных СУБД, поэтому мы не будем останавливаться на них подробно. Отметим только, что они могут быть определены как при создании таблицы, так и при ее изменении командой ALTER TABLE.

Динамические ограничения целостности в отличие от статических подразумевают возможность проверки более сложных условий: например, проверку что фамилии не содержат цифр, или дата рождения не может быть больше текущей даты, число и названия месяцев канонизированы и т.п. Некоторые данные не могут гарантированно быть признаны ошибочными, но могут быть классифицированы как маловероятные или «подозрительные» на ошибки и к их обработке должен быть привлечен оператор. Реализуются динамические ограничения с использованием триггеров – специальные процедуры, срабатывающие при возникновении некоторого события в БД.

<https://postgrespro.ru/docs/postgrespro/14/sql-createtrigger>

Создание триггера реализуется командой CREATE OR REPLACE TRIGGER.

Ниже представлен полный синтаксис по созданию триггера:

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

ON имя\_таблицы

[ FROM ссылающаяся\_таблица ]

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

[ REFERENCING { { OLD | NEW } TABLE [ AS ] имя\_переходного\_отношения } [ ... ] ]

[ FOR [ EACH ] { ROW | STATEMENT } ]

[ WHEN ( условие ) ]

EXECUTE { FUNCTION | PROCEDURE } имя\_функции ( аргументы )

Здесь допускается событие:

INSERT

UPDATE [ OF имя\_столбца [, ... ] ]

DELETE

TRUNCATE

Примеры:

Выполнение функции check\_account\_update перед любым изменением строк в таблице accounts:

CREATE TRIGGER check\_update

BEFORE UPDATE ON accounts

FOR EACH ROW

EXECUTE FUNCTION check\_account\_update();

Изменение определения триггера, чтобы данная функция выполнялась только при указании столбца balance в качестве целевого столбца команды UPDATE:

CREATE OR REPLACE TRIGGER check\_update

BEFORE UPDATE OF balance ON accounts

FOR EACH ROW

EXECUTE FUNCTION check\_account\_update();

В этом примере функция будет выполняться, если значение столбца balance в действительности изменилось:

CREATE TRIGGER check\_update

BEFORE UPDATE ON accounts

FOR EACH ROW

WHEN (OLD.balance IS DISTINCT FROM NEW.balance)

EXECUTE FUNCTION check\_account\_update();

<https://postgrespro.ru/docs/postgrespro/9.6/plpgsql-cursors>

Вместо того чтобы сразу выполнять весь запрос, есть возможность настроить курсор, инкапсулирующий запрос, и затем получать результат запроса по нескольку строк за раз. Одна из причин так делать заключается в том, чтобы избежать переполнения памяти, когда результат содержит большое количество строк. (Пользователям PL/pgSQL не нужно об этом беспокоиться, так как циклы FOR автоматически используют курсоры, чтобы избежать проблем с памятью.) Более интересным вариантом использования является возврат из функции ссылки на курсор, что позволяет вызывающему получать строки запроса. Это эффективный способ получать большие наборы строк из функций.

Доступ к курсорам в PL/pgSQL осуществляется через курсорные переменные, которые всегда имеют специальный тип данных refcursor. Один из способов создать курсорную переменную, просто объявить её как переменную типа refcursor. Другой способ заключается в использовании синтаксиса объявления курсора, который в общем виде выглядит так:

имя [ [ NO ] SCROLL ] CURSOR [ ( аргументы ) ] FOR запрос; (подробнее см. в методичке «Знакомство с языком PL/SQL и его управляющими конструкциями»)

**Задание на лабораторную работу**

1. Реализовать в таблице статические ограничения целостности (NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY).
2. Реализовать триггеры автоматической проверки и расчета динамических ограничений целостности для следующей задачи:

Рассматриваемые объекты (таблицы): группы товаров, товары. Предметная область - складской учет.

## Группы товаров

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Код группы | Имя группы | Количество на складе | Сводная розничная стоимость | Наценка  (0…1) |
| *Вводится вручную* | *Вводится вручную* | *Рассчитыва-ется автомати-чески* | *Рассчитывается автоматически* | *Вводится вручную* |
| 1 | Телевизоры | 15 | 532000,00 | 0.2 |
| 2 | Фотоаппараты | 25 | 212000,00 | 0.3 |
| 3 | Холодильники | 10 | 831200,00 | 0.15 |

## Товары

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Код товара | Наимено-вание | Код груп-пы | Приход-ная цена | Розничная цена | Кол-во на складе |
| *Рассчиты-вается автомати-чески* | *Вводится вручную* | *Вводится вручную* | *Вводится вручную* | *Рассчиты-вается автомати-чески* | *Вводится вручную* |
| 1 | Телевизор Philips | 1 | 10000 | 12000 | 5 |
| 2 | Телевизор Sony | 1 | 12000 | 14400 | 3 |
| 3 | Фото Panasonic | 2 | 5000 | 6500 | 4 |

**Требуется:**

* Организовать с помощью последовательностей автоматический ввод «кода товара» в таблице «Товары».
* Организовать автоматический расчет «Розничной цены» по формуле: «Розничная цена» = «Приходная цена» \* (1 + «Наценка на группу»)
* Написать триггеры, которые при изменении количества товаров в таблице «Товары», меняют «количество товаров» и «сводную стоимость» для соответствующей группы в таблице «Группы товаров».
* Написать триггеры, которые при изменении наценки на товар в таблице «Группы товаров» меняют «розничные цены» в таблице «Товары» для соответствующей группы.
* При создании таблицы Товары нужно по умолчанию заполнить значение столбцов 0, а не NULL
* При изменении розничной цены должна пересчитываться сводная стоимость (создать триггер).