Министерство образования и науки РФ

Санкт-Петербургский Политехнический университет Петра Великого

Институт компьютерных наук и кибербезопасности

Высшая школа программной инженерии

ОТЧЕТ

по лабораторной работе №3

Представления, хр. процедуры, триггера и курсоры

по дисциплине «Базы данных»

|  |  |
| --- | --- |
| Выполнила  студентка гр. 5130904/10101  Преподаватель | Никифорова Е. А.  Александрова О. В. |

Санкт-Петербург

2023

***Техническое задание:***

**Представления**

1. Создать представление, отображающее все статьи расхода, по которым за все время сумма превысила некоторую границу
2. Создать представление, отображающее общее количество расходов за последний месяц в разрезе статей расходов

Hints: select, where, count, max, group by, having, like, create view, drop view

**Хранимые процедуры**

* без параметров

1. Создать хранимую процедуру, выводящую все товары и среднюю стоимость их продаж за все время

* с входными параметрами

1. Создать хранимую процедуру, имеющую два параметра «товар1» и «товар2». Она должна возвращать даты продаж, в которых эти два товара продавались одновременно.

* с выходными параметрами

1. Создать хранимую процедуру с двумя входными параметрами «дата1» и «дата2» и двумя выходными параметрами, возвращающими общую сумму дохода и расхода за данный период с «дата1» по «дата2».

Hints: select, where, count, max, group by, having, create procedure, drop procedure

**Триггера**

* Триггера на вставку

1. Создать триггер, который не позволяет добавлять расход, с суммой большей заданной

* Триггера на модификацию

1. Создать триггер, который не позволяет изменять данные в таблице продаж задним числом от сегодняшней даты

* Триггера на удаление

1. Создать триггер, который при удалении расхода в случае, если расход был более чем месяц назад, откатывает транзакцию

Hints: select, where, in, exists, join, commit, rollback, create trigger, drop trigger

**Курсоры**

* Хранимая процедура для расчета суммы предполагаемой прибыли на ближайший месяц

Необходимо реализовать хранимую процедуру, рассчитывающую предполагаемую прибыль на ближайший месяц. Хранимая процедура должна иметь один выходной параметр, в котором возвращать размер прибыли.

Предлагаемый алгоритм: заводим две переменные ср\_расх, ср\_дох типа decimal; создаем курсор, который пробегает по статьям расхода за последние 4 месяца. Для каждой строки проверяем условие, если дата расхода попадает в последний месяц, то суммируем в ср\_расх с коэффициентов 1, если в предыдущий месяц, то с коэффициентом ½, в противном случае, ¼. Аналогично, создаем курсор по продажам за последние 4 месяца и так же с коэффициентами 1, ½, ¼ наращиваем сумму в переменную ср\_дох.

Некоторой переменной, значение которой по окончании работы курсора будет выдано в качестве выходного параметра, присваиваем значение ср\_дох - ср\_расх.

***Архитектура базы данных***

***Изображение выглядит как текст, снимок экрана, диаграмма, программное обеспечение

Автоматически созданное описание***

***Алгоритм решения***

**Представления:**

1. Создать представление, отображающее все статьи расхода, по которым за все время сумма превысила некоторую границу

CREATE VIEW excessive\_expenses AS

SELECT expense\_items.name, SUM(charges.amount)

FROM charges

JOIN expense\_items ON charges.expense\_item\_id = expense\_items.id

GROUP BY expense\_items.name

HAVING SUM(charges.amount) > 10000;

select \* from excessive\_expenses

1. Создать представление, отображающее общее количество расходов за последний месяц в разрезе статей расходов

CREATE VIEW monthly\_expenses AS

SELECT expense\_items.name, SUM(charges.amount)

FROM charges

JOIN expense\_items ON charges.expense\_item\_id = expense\_items.id

WHERE charge\_date >= CURRENT\_DATE - INTERVAL '1 month'

GROUP BY expense\_items.name;

Select \* from monthly\_expenses

**Хранимые процедуры**

* без параметров

1. Создать хранимую процедуру, выводящую все товары и среднюю стоимость их продаж за все время

CREATE OR REPLACE FUNCTION get\_all\_products\_avg\_sales()

RETURNS TABLE (

product\_name character varying(50),

avg\_sale\_price numeric

)

AS $$

BEGIN

RETURN QUERY

SELECT warehouses.name, COALESCE(AVG(sales.amount), 0)

FROM warehouses

LEFT JOIN sales ON warehouses.id = sales.warehouse\_id

GROUP BY warehouses.name;

END;

$$ LANGUAGE plpgsql;

select \* from get\_all\_products\_avg\_sales();

* с входными параметрами

1. Создать хранимую процедуру, имеющую два параметра «товар1» и «товар2». Она должна возвращать даты продаж, в которых эти два товара продавались одновременно.

CREATE OR REPLACE FUNCTION get\_simultaneous\_sales(product1 character varying(50), product2 character varying(50))

RETURNS TABLE (

sale\_date timestamp(3) with time zone

)

AS $$

BEGIN

RETURN QUERY

SELECT DISTINCT s1.sale\_date

FROM sales s1

JOIN sales s2 ON s1.sale\_date = s2.sale\_date

WHERE s1.warehouse\_id = (SELECT id FROM warehouses WHERE name = product1)

AND s2.warehouse\_id = (SELECT id FROM warehouses WHERE name = product2);

END;

$$ LANGUAGE plpgsql;

select \* from get\_simultaneous\_sales('Стол', 'Стул')

* с выходными параметрами

1. Создать хранимую процедуру с двумя входными параметрами «дата1» и «дата2» и двумя выходными параметрами, возвращающими общую сумму дохода и расхода за данный период с «дата1» по «дата2».

CREATE OR REPLACE FUNCTION get\_income\_and\_expenses(

date1 timestamp(3) with time zone,

date2 timestamp(3) with time zone,

OUT income numeric,

OUT expenses numeric

)

AS $$

BEGIN

SELECT SUM(sales.amount) INTO income

FROM sales

WHERE sales.sale\_date BETWEEN date1 AND date2;

SELECT SUM(charges.amount) INTO expenses

FROM charges

WHERE charges.charge\_date BETWEEN date1 AND date2;

END;

$$ LANGUAGE plpgsql;

select \* from get\_income\_and\_expenses('20230101','20231102')

**Триггера**

* Триггера на вставку

1. Создать триггер, который не позволяет добавлять расход, с суммой большей заданной

CREATE OR REPLACE FUNCTION check\_expense\_limit()

RETURNS TRIGGER

AS $$

BEGIN

IF NEW.amount > 100000 THEN -- Здесь установите своё заданное значение

RAISE EXCEPTION 'Сумма расхода превышает допустимый лимит!';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER check\_expense\_limit

BEFORE INSERT ON charges

FOR EACH ROW

EXECUTE FUNCTION check\_expense\_limit();

insert into public.charges (amount, charge\_date, expense\_item\_id)

values (10000000,'2023-11-06',(select public.expense\_items.id from expense\_items where name = 'Зарплата'))

* Триггера на модификацию

1. Создать триггер, который не позволяет изменять данные в таблице продаж задним числом от сегодняшней даты

CREATE OR REPLACE FUNCTION check\_sales\_backdating()

RETURNS TRIGGER AS $$

BEGIN

IF OLD.sale\_date < CURRENT\_DATE THEN

RAISE EXCEPTION 'Нельзя изменять данные в таблице продаж задним числом!';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER check\_sales\_backdating

BEFORE UPDATE ON sales

FOR EACH ROW

EXECUTE FUNCTION check\_sales\_backdating();

update sales set amount = amount + 999

where sale\_date = '20231007'

* Триггера на удаление

1. Создать триггер, который при удалении расхода в случае, если расход был более чем месяц назад, откатывает транзакцию

CREATE OR REPLACE FUNCTION prevent\_old\_expense\_deletion()

RETURNS TRIGGER AS $$

BEGIN

IF OLD.charge\_date < CURRENT\_DATE - INTERVAL '1 month' THEN

RAISE EXCEPTION 'Нельзя удалять старые записи о расходах!';

END IF;

RETURN OLD;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER prevent\_old\_expense\_deletion

BEFORE DELETE ON charges

FOR EACH ROW

EXECUTE FUNCTION prevent\_old\_expense\_deletion();

DELETE from public.charges

where charges.charge\_date = '20230929';

**Курсоры**

* Хранимая процедура для расчета суммы предполагаемой прибыли на ближайший месяц

CREATE OR REPLACE PROCEDURE calculate\_expected\_profit(OUT profit DECIMAL)

AS $$

DECLARE

avg\_expenses DECIMAL := 0;

avg\_income DECIMAL := 0;

expense\_coeff DECIMAL := 0;

income\_coeff DECIMAL := 0;

expense\_cursor CURSOR FOR

SELECT charge\_date, amount FROM charges WHERE charges.charge\_date >= CURRENT\_DATE - INTERVAL '4 months';

income\_cursor CURSOR FOR

SELECT sale\_date, amount FROM sales WHERE sale\_date >= CURRENT\_DATE - INTERVAL '4 months';

expense\_record RECORD;

income\_record RECORD;

BEGIN

FOR expense\_record IN expense\_cursor LOOP

IF date\_trunc('month', expense\_record.charge\_date) = date\_trunc('month', CURRENT\_DATE) THEN

expense\_coeff := 1;

ELSIF date\_trunc('month', expense\_record.charge\_date) = date\_trunc('month', CURRENT\_DATE - INTERVAL '1 month') THEN

expense\_coeff := 0.5;

ELSE

expense\_coeff := 0.25;

END IF;

avg\_expenses := avg\_expenses + (expense\_record.amount \* expense\_coeff);

END LOOP;

FOR income\_record IN income\_cursor LOOP

IF date\_trunc('month', income\_record.sale\_date) = date\_trunc('month', CURRENT\_DATE) THEN

income\_coeff := 1;

ELSIF date\_trunc('month', income\_record.sale\_date) = date\_trunc('month', CURRENT\_DATE - INTERVAL '1 month') THEN

income\_coeff := 0.5;

ELSE

income\_coeff := 0.25;

END IF;

avg\_income := avg\_income + (income\_record.amount \* income\_coeff);

END LOOP;

profit := avg\_income - avg\_expenses;

END;

$$ LANGUAGE plpgsql;

DO $$

DECLARE

result numeric;

BEGIN

CALL calculate\_expected\_profit(result);

RAISE NOTICE 'Предполагаемая прибыль на ближайший месяц: %', result;

END $$;

***Вывод***

С помощью различных sql команд, курсоров были созданы представления, триггеры функции и хранимая процедура в базе данных для получения необходимой статистики, а также для предотвращения запрещенных действий.