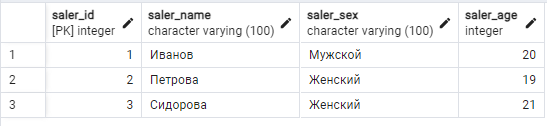
**Мелехин Александр Кс-30 Вариант 9 Лабораторная работа 7**

**Данные таблицы для лабораторной работы 7**

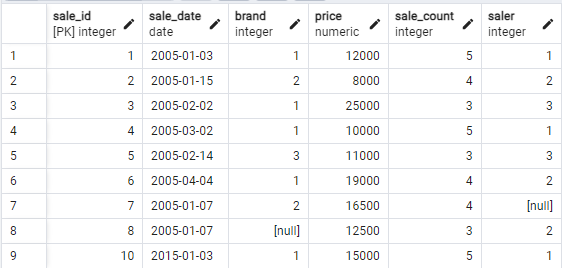
**Таблица salers**



**Таблица brands**



**Таблица sales**



**Задание 1**

**Задание:** создать функцию Period(…, …) c двумя входными параметрами типа date, которая выберет строки из дочерней таблицы в диапазоне дат, указанных первым и вторым аргументами при вызове функции Period(…, …).

**SQL код для задания:**

CREATE FUNCTION Period(date\_from DATE, date\_to DATE)

RETURNS TABLE(sale\_id INT, sale\_date DATE, brand INT, price DECIMAL, sale\_count INT, saler INT) AS $$

BEGIN

RETURN QUERY

SELECT \*

FROM sales

WHERE sales.sale\_date BETWEEN Period.date\_from AND Period.date\_to;

END;

$$ LANGUAGE plpgsql;

SELECT \* FROM Period('2005-01-01', '2005-02-01');

**Пояснение:** функция Period принимает две даты (date\_from и date\_to) и возвращает строки из таблицы sales, где sale\_date попадает в указанный диапазон.

**Результат**

**Задание 2**

**Задание:** создать функцию Sum\_object(…)(с одним параметром), которая возвращает список имен объектов из родительской таблицы на основании данных дочерней таблицы. Список объектов определяется значением параметра, исходя из условия, что суммарное количество объектов должно быть больше, чем заданное значение в параметре.

**SQL код для задания:**

CREATE FUNCTION Sum\_object(min\_sale\_count INT)

RETURNS TABLE(brand\_name VARCHAR) AS $$

BEGIN

RETURN QUERY

SELECT b.brand\_name

FROM sales s

JOIN brands b ON s.brand = b.brand\_id

GROUP BY b.brand\_name

HAVING SUM(s.sale\_count) > min\_sale\_count;

END;

$$ LANGUAGE plpgsql;

SELECT \* FROM Sum\_object(4);

**Пояснение:** функция Sum\_object принимает параметр min\_sale\_count, и возвращает список брендов из таблицы brands, для которых общее количество продаж (по полю sale\_count в таблице sales) превышает это значение.

**Результат**

**Задание 3**

**Задание:** создать функцию row\_count(…), которая подсчитывает количество строк дочерней таблицы, даты которых находятся между параметрами date\_from и date\_to.

**SQL код для задания:**

CREATE FUNCTION row\_count(date\_from DATE, date\_to DATE)

RETURNS INT AS $$

DECLARE

total\_count INT;

BEGIN

SELECT COUNT(\*)

INTO total\_count

FROM sales

WHERE sale\_date BETWEEN date\_from AND date\_to;

RETURN total\_count;

END;

$$ LANGUAGE plpgsql;

SELECT \* FROM row\_count('2005-01-01', '2005-02-01');

**Пояснение:** функция row\_count принимает параметры date\_from и date\_to, и возвращает количество строк из таблицы sales, где дата продажи (sale\_date) находится в указанном диапазоне.

**Результат**

**Задание 4**

**Задание:** создать хранимую процедуру object\_stat(…), которая подсчитывает минимальное, максимальное и среднее значение объектов в дочерней таблице, входным параметром является имя объекта.

**SQL код для задания:**

CREATE PROCEDURE object\_stat(brand\_name\_input VARCHAR)

LANGUAGE plpgsql AS $$

DECLARE

min\_count INT;

max\_count INT;

avg\_count DECIMAL;

BEGIN

SELECT MIN(sale\_count), MAX(sale\_count), AVG(sale\_count)

INTO min\_count, max\_count, avg\_count

FROM sales s

JOIN brands b ON s.brand = b.brand\_id

WHERE b.brand\_name = brand\_name\_input;

RAISE NOTICE 'Минимальное количество: %, Максимальное количество: %, Среднее количество: %', min\_count, max\_count, avg\_count;

END;

$$;

CALL object\_stat('Самсунг');

**Пояснение:** Процедура object\_stat принимает имя бренда и выводит минимальное, максимальное и среднее количество продаж (sale\_count) для этого бренда.

**Результат**

**Задание 5**

**Задание:** создать хранимую процедуру objects\_stat(…), которая подсчитывает минимальное, максимальное и среднее значение каждого объекта в дочерней таблице и выводит имя объекта, входным параметром является имя объекта из родительской таблицы.

**SQL код для задания:**

CREATE PROCEDURE objects\_stat()

LANGUAGE plpgsql AS $$

DECLARE

saler\_record RECORD;

BEGIN

FOR saler\_record IN

SELECT s.saler\_name, MIN(sales.sale\_count) AS min\_count,

MAX(sales.sale\_count) AS max\_count,

AVG(sales.sale\_count) AS avg\_count

FROM sales

JOIN salers s ON sales.saler = s.saler\_id

GROUP BY s.saler\_name

LOOP

RAISE NOTICE 'Продавец: %, Минимум: %, Максимум: %, Среднее: %',

saler\_record.saler\_name, saler\_record.min\_count, saler\_record.max\_count, saler\_record.avg\_count;

END LOOP;

END;

$$;

CALL objects\_stat();CALL objects\_stat();

**Пояснение:** процедура objects\_stat выводит минимальное, максимальное и среднее количество продаж (sale\_count) для каждого продавца из таблицы salers.

**Результат**

**Задание 6**

**Задание:** создать хранимую процедуру Itog(…) с одним входным параметром, которая выводит наименование объекта по суммарному количеству объектов: а) Оценка «Незначительный объект», если число объектов меньше 2 б) Оценка «Обычный объект», если число объектов больше 2 и меньше или равно 3 с) Оценка «Значительный объект», если число объектов больше 3

**SQL код для задания:**

CREATE PROCEDURE Itog()

LANGUAGE plpgsql AS $$

DECLARE

brand\_record RECORD;

total\_sales INT;

BEGIN

FOR brand\_record IN

SELECT b.brand\_name, SUM(s.sale\_count) AS total\_sales

FROM sales s

JOIN brands b ON s.brand = b.brand\_id

GROUP BY b.brand\_name

LOOP

total\_sales := brand\_record.total\_sales;

IF total\_sales < 2 THEN

RAISE NOTICE 'Бренд: %, Оценка: Незначительный объект', brand\_record.brand\_name;

ELSIF total\_sales > 2 AND total\_sales <= 3 THEN

RAISE NOTICE 'Бренд: %, Оценка: Обычный объект', brand\_record.brand\_name;

ELSE

RAISE NOTICE 'Бренд: %, Оценка: Значительный объект', brand\_record.brand\_name;

END IF;

END LOOP;

END;

$$;

CALL Itog();

**Пояснение:** процедура Itog оценивает каждый бренд в таблице brands по общему количеству проданных единиц, выводя оценку в зависимости от значения: "Незначительный объект" для количества меньше 2, "Обычный объект" для количества от 2 до 3, и "Значительный объект" для количества больше 3.

**Результат**

**Задание 7**

**Задание:** создать триггер After\_Delete, который при удалении записи из родительской таблицы удалял бы все связанные записи из дочерней таблицы. Показать результат работы триггера.

**SQL код для задания:**

CREATE FUNCTION delete\_sales\_on\_saler\_delete()

RETURNS TRIGGER AS $$

BEGIN

DELETE FROM sales WHERE saler = OLD.saler\_id;

RETURN OLD;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER after\_delete\_saler

AFTER DELETE ON salers

FOR EACH ROW

EXECUTE FUNCTION delete\_sales\_on\_saler\_delete();

**Пояснение:** триггер after\_delete\_saler удаляет все записи из sales, связанные с удаляемым продавцом.

**Результат**

**Задание 8**

**Задание:** создать триггер Before\_Delete, который при удалении записи из дочерней таблицы выводил бы имя объекта родительской таблицы. Показать результат работы триггера

**SQL код для задания:**

CREATE FUNCTION show\_object\_name\_before\_delete()

RETURNS TRIGGER AS $$

DECLARE

brand\_name VARCHAR;

saler\_name VARCHAR;

BEGIN

-- Получаем имя бренда из таблицы brands

SELECT b.brand\_name INTO brand\_name FROM brands b WHERE b.brand\_id = OLD.brand;

-- Получаем имя продавца из таблицы salers

SELECT s.saler\_name INTO saler\_name FROM salers s WHERE s.saler\_id = OLD.saler;

RAISE NOTICE 'Удаление продажи: Бренд - %, Продавец - %', brand\_name, saler\_name;

RETURN OLD;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER before\_delete\_sales

BEFORE DELETE ON sales

FOR EACH ROW

EXECUTE FUNCTION show\_object\_name\_before\_delete();

**Пояснение:** триггер before\_delete\_sales срабатывает перед удалением записи из sales и выводит сообщение с именами соответствующего бренда и продавца.

**Результат**

**Задание 9**

**Задание:** создать триггер ins\_sum, который связывает триггер с таблицей для инструкций INSERT. Это действует как сумматор, чтобы суммировать значения, вставленные в один из столбцов дочерней таблицы. Триггер должен активироваться перед каждой строкой, вставленной в таблицу. Показать результат работы триггера.

**SQL код для задания:**

CREATE OR REPLACE FUNCTION increment\_total\_sales()

RETURNS TRIGGER AS $$

DECLARE

total\_price DECIMAL;

BEGIN

SELECT SUM(price) INTO total\_price FROM sales;

RAISE NOTICE 'Сумма значений price до вставки: %', total\_price;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER ins\_sum

BEFORE INSERT ON sales

FOR EACH ROW

EXECUTE FUNCTION increment\_total\_sales();

**Пояснение:** триггер ins\_sum срабатывает перед каждой вставкой в sales, суммируя все значения price до вставки.

**Результат**

**Задание 10**

**Задание:** cоздать триггер Before\_Update\_Value на событие UPDATE, который увеличивает значение числового поля дочерней таблицы на 10%. Показать результат работы триггера

**SQL код для задания:**

CREATE FUNCTION increase\_price\_on\_update()

RETURNS TRIGGER AS $$

BEGIN

NEW.price := NEW.price \* 1.1;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER before\_update\_price

BEFORE UPDATE ON sales

FOR EACH ROW

EXECUTE FUNCTION increase\_price\_on\_update();

**Пояснение:** триггер before\_update\_price срабатывает перед обновлением записи в таблице sales и увеличивает значение поля price на 10%.

**Результат**