

## Мелехин Александр Кс-30 Вариант 9 Лабораторная работа 6

### Данные таблицы для лабораторной работы 6

Таблица salers

	saler_id [PK] integer	saler_name character varying (100)	saler_sex character varying (100)	saler_age integer
1	1	Иванов	Мужской	20
2	2	Петрова	Женский	19
3	3	Сидорова	Женский	21

Таблица brands

	brand_id [PK] integer	brand_name character varying (100)
1	1	Самсунг
2	2	Леново
3	3	Сони

Таблица sales

	sale_id [PK] integer	sale_date date	brand integer	price numeric	sale_count integer	saler integer
1	1	2005-01-03	1	12000	5	1
2	2	2005-01-15	2	8000	4	2
3	3	2005-02-02	1	25000	3	3
4	4	2005-03-02	1	10000	5	1
5	5	2005-02-14	3	11000	3	3
6	6	2005-04-04	1	19000	4	2
7	7	2005-01-07	2	16500	4	[null]
8	8	2005-01-07	[null]	12500	3	2
9	10	2015-01-03	1	15000	5	1

## Задание 1

**Задание:** создать функцию Period(..., ...) с 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

	sale_id integer	sale_date date	brand integer	price numeric	sale_count integer	saler integer
1	1	2005-01-03	1	12000	5	1
2	2	2005-01-15	2	8000	4	2
3	7	2005-01-07	2	16500	4	[null]
4	8	2005-01-07	[null]	12500	3	2

## 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

	brand_name character varying 
1	Леново
2	Самсунг

### 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

	row_count integer	
1		4

## 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

```
ЗАМЕЧАНИЕ: Минимальное количество: 3, Максимальное количество: 5, Среднее количество: 4.4000000000000000
CALL
```

```
Query returned successfully in 72 msec.
```

## 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

---

```
ЗАМЕЧАНИЕ: Продавец: Сидорова, Минимум: 3, Максимум: 3, Среднее: 3.0000000000000000
ЗАМЕЧАНИЕ: Продавец: Петрова, Минимум: 3, Максимум: 4, Среднее: 3.6666666666666667
ЗАМЕЧАНИЕ: Продавец: Иванов, Минимум: 5, Максимум: 5, Среднее: 5.0000000000000000
CALL
```

```
Query returned successfully in 84 msec.
```

## 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

```
ЗАМЕЧАНИЕ: Бренд: Леново, Оценка: Значительный объект
ЗАМЕЧАНИЕ: Бренд: Сони, Оценка: Обычный объект
ЗАМЕЧАНИЕ: Бренд: Самсунг, Оценка: Значительный объект
CALL
```

```
Query returned successfully in 62 msec.
```



## 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

```
INSERT INTO public.salers(saler_name, saler_sex, saler_age) VALUES
('Николаев','Мужской', '25');
INSERT INTO public.sales(sale_date, brand, price, sale_count, saler) VALUES
('3.1.2016', 1, 12000, 5, 4);
SELECT * FROM salers;
SELECT * FROM sales;
```

	saler_id [PK] integer	saler_name character varying (100)	saler_sex character varying (100)	saler_age integer
1	1	Иванов	Мужской	20
2	2	Петрова	Женский	19
3	3	Сидорова	Женский	21
4	4	Николаев	Мужской	25

	sale_id [PK] integer	sale_date date	brand integer	price numeric	sale_count integer	saler integer
1	1	2005-01-03	1	12000	5	1
2	2	2005-01-15	2	8000	4	2
3	3	2005-02-02	1	25000	3	3
4	4	2005-03-02	1	10000	5	1
5	5	2005-02-14	3	11000	3	3
6	6	2005-04-04	1	19000	4	2
7	7	2005-01-07	2	16500	4	[null]
8	8	2005-01-07	[null]	12500	3	2
9	9	2015-01-03	1	15000	5	1
10	10	2016-01-03	1	12000	5	4

DELETE FROM salers WHERE saler\_id = 4;

SELECT \* FROM salers;

SELECT \* FROM sales;

	saler_id [PK] integer	saler_name character varying (100)	saler_sex character varying (100)	saler_age integer
1	1	Иванов	Мужской	20
2	2	Петрова	Женский	19
3	3	Сидорова	Женский	21

	sale_id [PK] integer	sale_date date	brand integer	price numeric	sale_count integer	saler integer
1	1	2005-01-03	1	12000	5	1
2	2	2005-01-15	2	8000	4	2
3	3	2005-02-02	1	25000	3	3
4	4	2005-03-02	1	10000	5	1
5	5	2005-02-14	3	11000	3	3
6	6	2005-04-04	1	19000	4	2
7	7	2005-01-07	2	16500	4	[null]
8	8	2005-01-07	[null]	12500	3	2
9	9	2015-01-03	1	15000	5	1

## 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

	sale_id [PK] integer	sale_date date	brand integer	price numeric	sale_count integer	saler integer
1	1	2005-01-03	1	12000	5	1
2	2	2005-01-15	2	8000	4	2
3	3	2005-02-02	1	25000	3	3
4	4	2005-03-02	1	10000	5	1
5	5	2005-02-14	3	11000	3	3
6	6	2005-04-04	1	19000	4	2
7	7	2005-01-07	2	16500	4	[null]
8	8	2005-01-07	[null]	12500	3	2
9	9	2015-01-03	1	15000	5	1

DELETE FROM sales WHERE sale\_id = 9;

ЗАМЕЧАНИЕ: Удаление продажи: Бренд - Самсунг, Продавец - Иванов  
DELETE 1

Query returned successfully in 89 msec.

	sale_id [PK] integer	sale_date date	brand integer	price numeric	sale_count integer	saler integer
1	1	2005-01-03	1	12000	5	1
2	2	2005-01-15	2	8000	4	2
3	3	2005-02-02	1	25000	3	3
4	4	2005-03-02	1	10000	5	1
5	5	2005-02-14	3	11000	3	3
6	6	2005-04-04	1	19000	4	2
7	7	2005-01-07	2	16500	4	[null]
8	8	2005-01-07	[null]	12500	3	2

## 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 FUNCTION increment_total_sales()
RETURNS TRIGGER AS $$
BEGIN
    -- Добавляем значение `price` новой строки к общей сумме
    UPDATE sales_summary
    SET total_sales = total_sales + NEW.price;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
CREATE 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 значения `sale_count`.

### Результат

	total_sales 
1	20000

	sale_id [PK] integer	sale_date date	brand integer	price numeric	sale_count integer	saler integer
1	1	2005-01-03	1	12000	5	1
2	2	2005-01-15	2	8000	4	2
3	3	2005-02-02	1	25000	3	3
4	4	2005-03-02	1	10000	5	1
5	5	2005-02-14	3	11000	3	3
6	6	2005-04-04	1	19000	4	2
7	7	2005-01-07	2	16500	4	[null]
8	8	2005-01-07	[null]	12500	3	2
9	15	2024-01-01	2	20000	3	2

INSERT INTO sales (sale\_date, brand, price, sale\_count, saler) VALUES ('2024-01-02', 2, 5, 3, 1);

SELECT \* FROM sales\_summary;

	total_sales numeric
1	20005

	sale_id [PK] integer	sale_date date	brand integer	price numeric	sale_count integer	saler integer
1	1	2005-01-03	1	12000	5	1
2	2	2005-01-15	2	8000	4	2
3	3	2005-02-02	1	25000	3	3
4	4	2005-03-02	1	10000	5	1
5	5	2005-02-14	3	11000	3	3
6	6	2005-04-04	1	19000	4	2
7	7	2005-01-07	2	16500	4	[null]
8	8	2005-01-07	[null]	12500	3	2
9	15	2024-01-01	2	20000	3	2
10	16	2024-01-02	2	5	3	1

## Задание 10

**Задание:** создать 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

	sale_id [PK] integer	sale_date date	brand integer	price numeric	sale_count integer	saler integer
1	1	2005-01-03	1	12000	5	1
2	2	2005-01-15	2	8000	4	2
3	3	2005-02-02	1	25000	3	3
4	4	2005-03-02	1	10000	5	1
5	5	2005-02-14	3	11000	3	3
6	6	2005-04-04	1	19000	4	2
7	7	2005-01-07	2	16500	4	[null]
8	8	2005-01-07	[null]	12500	3	2
9	15	2024-01-01	2	20000	3	2
10	16	2024-01-02	2	5	3	1

```
UPDATE sales SET saler = 3 WHERE sale_id = 15;
SELECT * FROM sales
```

	sale_id [PK] integer 	sale_date date 	brand integer 	price numeric 	sale_count integer 	saler integer 
1	1	2005-01-03	1	12000	5	1
2	2	2005-01-15	2	8000	4	2
3	3	2005-02-02	1	25000	3	3
4	4	2005-03-02	1	10000	5	1
5	5	2005-02-14	3	11000	3	3
6	6	2005-04-04	1	19000	4	2
7	7	2005-01-07	2	16500	4	[null]
8	8	2005-01-07	[null]	12500	3	2
9	16	2024-01-02	2	5	3	1
10	15	2024-01-01	2	22000.0	3	3