



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»

РТУ МИРЭА

**Институт информационных технологий (ИИТ)
Кафедра цифровой трансформации (ЦТ)**

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 4
по дисциплине «Разработка баз данных»

Студент группы

ИНБО-12-23. Албахтин И.В.

(подпись)

Ассистент

Брайловский А.В.

(подпись)

Москва 2025 г.

ПРАКТИЧЕСКАЯ РАБОТА №4.
АНАЛИТИЧЕСКИЕ ЗАПРОСЫ: ОКОННЫЕ
ФУНКЦИИ И ПОСТРОЕНИЕ СВОДНЫХ ТАБЛИЦ

Цель:

Целью данной практической работы является формирование у студентов углубленных навыков работы со сложными аналитическими запросами в СУБД PostgreSQL.

Постановка задачи:

Для выполнения практической работы необходимо последовательно выполнить четыре задачи, используя собственную базу данных. Все примеры в данном документе основаны на демонстрационной базе данных «Аптека», содержащей таблицы manufacturers (производители), medicines (лекарства) и sales (продажи).

Ваша задача — адаптировать каждую из поставленных задач к логической структуре и предметной области вашей базы данных. Приведенные ниже формулировки и последующие примеры кода служат шаблоном для понимания, какой тип аналитического запроса требуется составить.

Задание №1: использование ранжирующих функций

Для каждой основной «родительской» сущности в вашей БД (*например, производитель, категория товара, автор*) определить **три** наиболее значимых по некоторому **числовому признаку** дочерних сущности (*например, три самых дорогих товара, три самые популярные книги по количеству продаж*).

В результирующей таблице должны быть указаны идентификатор группы, идентификатор дочерней сущности, её числовой признак и ранг. Для расчёта ранга использовать функцию **RANK()** или **DENSE_RANK()**.

Задание №2: использование агрегатных оконных функций

Для ключевой сущности, имеющей **транзакции по времени** (*например, товар, услуга*), рассчитать **нарастающий итог** (*кумулятивную сумму*) по

некоторому показателю (*например, объем продаж, количество заказов*) с разбивкой по временным периодам (*месяцам или годам*).

Отчёт должен содержать идентификатор сущности (id/название/...), временной период, сумму за период и кумулятивную сумму.

Задание №3: использование функции смещения

Провести сравнительный анализ общих показателей **по периодам**.

Для **каждого периода** (*например, месяца*), начиная со второго, необходимо вывести **общий показатель** за **текущий** период и аналогичный показатель за **предыдущий** период в одной строке. Это позволит наглядно оценить динамику.

Необходимо использовать функцию **LAG()**.

Задание №4: построение сводной таблицы

Создать сводный отчет, который агрегирует некоторый числовой показатель для основной сущности по категориям, представленным в виде столбцов.

ВЫПОЛНЕНИЕ ПРАКТИЧЕСКОЙ РАБОТЫ

Таблица 1. Таблица worker (Сотрудник)

The screenshot shows a PostgreSQL database interface with the following components:

- Query Editor:** Displays a complex SQL script for generating a crosstab view. The script uses a CTE named `crosstab` to calculate quarterly totals for invoices, then joins this with a sequence of quarters (1-4) to create a final result set. Below this, a simple `SELECT * FROM worker LIMIT 10;` statement is shown.
- Results Viewer:** Shows a table titled "worker 1" with two rows of data. The columns are `worker_id`, `name`, `position`, and `phone`. The data is:

worker_id	name	position	phone
1	Сергей	Механик	+79112223344
2	Ольга	Диагност	+79210009988

Таблица 2. Таблица maintenance (TO)

The screenshot shows a database management interface with a code editor and a results viewer.

Code Editor:

```

GROUP BY year, quarter
ORDER BY 1, 2',
'SELECT q FROM generate_series(1,4) AS q'
) AS ct(year NUMERIC, Q1 NUMERIC, Q2 NUMERIC, Q3 NUMERIC, Q4 NUMERIC);

SELECT * FROM worker LIMIT 10;

-- SELECT maintenance_id, car_id, worker_id, start_date, end_date, status
-- FROM maintenance
-- LIMIT 10;

-- -- 3. Таблица счетов
-- SELECT invoice_id, maintenance_id, total_amount, payment_status

```

Results Viewer:

Table: maintenance 1

	maintenance_id	car_id	worker_id	start_date	end_date	status
1	34	4	4	2025-02-10	[NULL]	waiting
2	35	5	5	2025-03-20	2025-09-10	completed
3	36	6	2	2025-03-20	[NULL]	in progress
4	37	7	3	2025-04-25	2025-09-11	completed
5	38	8	1	2025-04-25	[NULL]	planned
6	39	9	4	2025-05-15	2025-09-15	completed
7	40	10	5	2025-05-15	[NULL]	in progress
8	33	3	2	2025-11-05	2025-11-15	completed
9	31	1	1	2025-07-15	2025-07-20	completed
10	32	2	3	2025-09-10	2025-09-20	in progress

Таблица 3. Таблица invoice (счета)

The screenshot shows the DB Stud software interface. At the top, there's a toolbar with icons for file operations and a tab bar showing 'albakhtin_iv' and 'Script-9'. Below the toolbar, a vertical sidebar contains icons for navigation, search, and other database functions. The main area has a script editor window containing three SQL queries:

```
SELECT * FROM worker LIMIT 10;  
SELECT maintenance_id, car_id, worker_id, start_date, end_date, stat  
FROM maintenance  
LIMIT 10;  
SELECT invoice_id, maintenance_id, total_amount, payment_status  
FROM invoice  
LIMIT 10;
```

Below the script editor is a results viewer titled 'invoice 1'. It displays a table with the following data:

invoice_id	maintenance_id	total_amount	payment_status
1	3	31	35 000 Оплачено
2	4	32	18 000 Ожидает оплаты
3	5	33	26 000 Оплачено
4	6	34	40 000 Оплачено
5	7	35	15 000 Оплачено
6	8	36	17 000 Оплачено
7	9	37	22 000 Ожидает оплаты
8	10	38	19 500 Оплачено
9	11	39	31 000 Оплачено
10	12	40	28 000 Оплачено

The results viewer also includes a text input field at the bottom with the placeholder 'Введите SQL выражение чтобы отфильтровать результаты'.

Задание 1. Создание модифицируемого представления

The screenshot shows the pgAdmin interface for PostgreSQL. A script editor window titled 'albakhtin_iv' contains the following SQL code:

```
CREATE OR REPLACE VIEW paid_invoices AS
SELECT
    invoice_id,
    maintenance_id,
    total_amount,
    payment_status
FROM invoice
WHERE payment_status = 'Оплачено';
```

Below the script editor is a statistics panel titled 'Статистика 1' (Statistics 1) with the following data:

Name	Value
Updated Rows	0
Execute time	0.014s
Start time	Sun Nov 02 14:13:52 MSK 2025
Finish time	Sun Nov 02 14:13:52 MSK 2025
Query	CREATE OR REPLACE VIEW paid_invoices AS SELECT invoice_id, maintenance_id, total_amount, payment_status FROM invoice WHERE payment_status = 'Оплачено'

Рисунок 1 – Все оплаченные счета с фильтром payment_status = 'Оплачено' часть 1

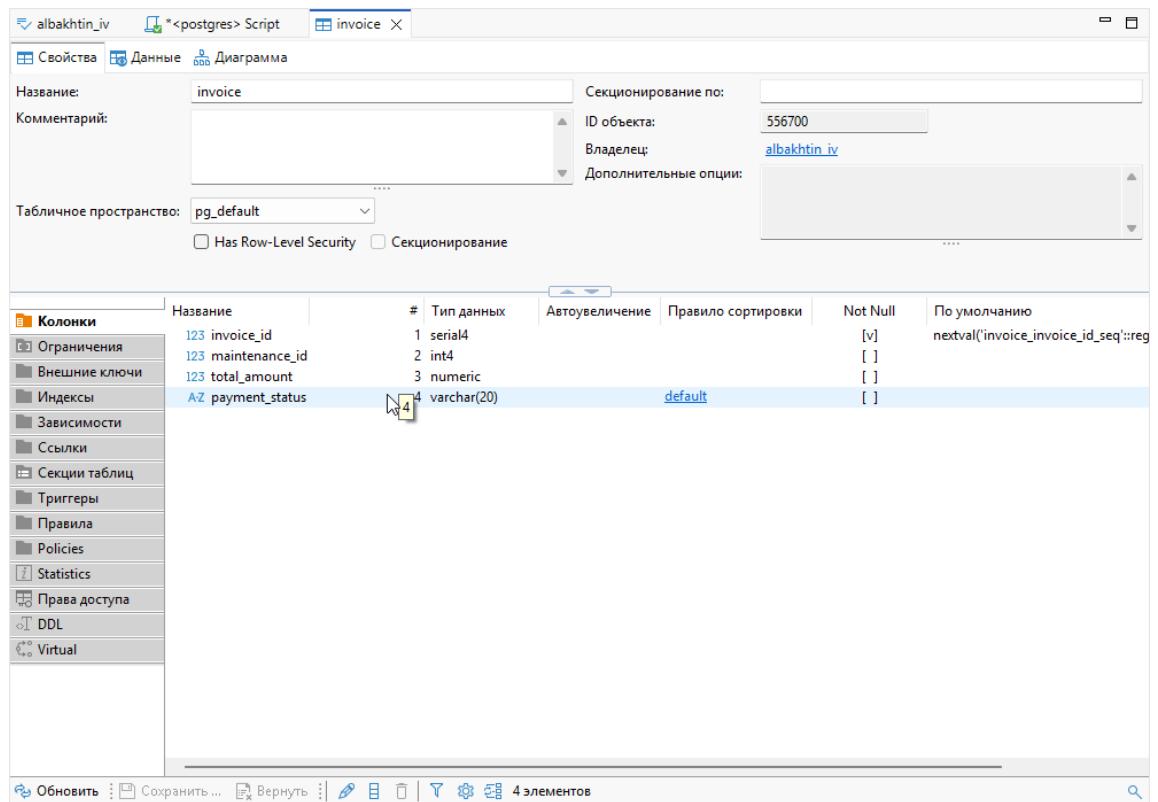


Рисунок 2 - Все оплаченные счета с фильтром payment_status = 'Оплачено' часть 2

Задание 2. Модификация данных через представление

```

JOIN cat_tree ct ON c.parent_id = ct.category_id
)
SELECT
    LPAD(' ', level * 4, ' ') || name AS hierarchy,
    level
FROM cat_tree;

CREATE OR REPLACE VIEW paid_invoices AS
SELECT
    invoice_id,
    maintenance_id,
    total_amount,
    payment_status
FROM invoice
WHERE payment_status = 'Оплачено';

INSERT INTO paid_invoices (maintenance_id, total_amount, payment_status)
VALUES (31, 25000, 'Оплачено');

```

Name	Value
Updated Rows	1
Execute time	0.010s
Start time	Sun Nov 02 14:18:36 MSK 2025
Finish time	Sun Nov 02 14:18:36 MSK 2025
Query	INSERT INTO paid_invoices (maintenance_id, total_amount, payment_status) VALUES (31, 25000, 'Оплачено')

Рисунок 3 – Добавление новой записи через представление

The screenshot shows a PostgreSQL database interface. In the top-left corner, there's a tree view of database objects. Below it is a script editor window titled 'albakhtin_iv' containing several SQL statements. One statement is highlighted in red, indicating an error or warning. The script includes creating a view, inserting data into a temporary table, and deleting from a view. Below the script is a 'Статистика 1' (Statistics 1) panel displaying execution details like 'Updated Rows' (0), 'Execute time' (0.011s), and the executed query.

Рисунок 4 - Удаление записи через представление

This screenshot shows a similar PostgreSQL interface. The script editor contains the same set of SQL statements as in Figure 4. Below the script is a 'invoice 1' data viewer window. It displays a table with columns: invoice_id, maintenance_id, total_amount, and payment_status. The data shows 14 rows of invoices, with the first row selected. A status bar at the bottom indicates '14 строк получено - 0.008s, 2025-11-02 в 14:25:15'.

invoice_id	maintenance_id	total_amount	payment_status
16	31	25 000	Оплачено
15	31	25 000	Оплачено
14	31	25 000	Оплачено
13	31	25 000	Оплачено
12	40	28 000	Оплачено
11	39	31 000	Оплачено
10	38	19 500	Оплачено
9	37	22 000	Ожидает оплаты
8	36	17 000	Оплачено
7	35	15 000	Оплачено
6			
5			
4			
3			
2			
1			

Рисунок 5 - Проверка

Задание 3. Создание немодифицируемого аналитического представления

```


--> DELETE FROM paid_invoices
   WHERE invoice_id = 1;

--> SELECT * FROM invoice WHERE invoice_id = 1;

--> INSERT INTO paid_invoices (maintenance_id, total_amount, payment_status)
VALUES (31, 25000, 'Оплачено');

--> SELECT * FROM invoice ORDER BY invoice_id DESC;

--> CREATE OR REPLACE VIEW worker_summary AS
SELECT
    w.name AS worker_name,
    COUNT(m.maintenance_id) AS total_services,
    COALESCE(SUM(i.total_amount), 0) AS total_revenue,
    ROUND(AVG(i.total_amount), 2) AS average_invoice
FROM worker w
LEFT JOIN maintenance m ON w.worker_id = m.worker_id
LEFT JOIN invoice i ON m.maintenance_id = i.maintenance_id
GROUP BY w.name;


```

Статистика 1

Name	Value
Updated Rows	0
Execute time	0.015s
Start time	Sun Nov 02 14:25:49 MSK 2025
Finish time	Sun Nov 02 14:25:49 MSK 2025
Query	CREATE OR REPLACE VIEW worker_summary AS
	SELECT
	w.name AS worker_name,
	COUNT(m.maintenance_id) AS total_services,
	COALESCE(SUM(i.total_amount), 0) AS total_revenue,
	ROUND(AVG(i.total_amount), 2) AS average_invoice
	FROM worker w
	LEFT JOIN maintenance m ON w.worker_id = m.worker_id
	LEFT JOIN invoice i ON m.maintenance_id = i.maintenance_id
	GROUP BY w.name;

Значение

Select a cell to view/edit value
Press F7 to hide this panel

Рисунок 6 – Создание сводки по каждому сотруднику

```


--> SELECT * FROM paid_invoices
   WHERE invoice_id = 1;

--> SELECT * FROM invoice WHERE invoice_id = 1;

--> INSERT INTO paid_invoices (maintenance_id, total_amount, payment_status)
VALUES (31, 25000, 'Оплачено');

--> SELECT * FROM invoice ORDER BY invoice_id DESC;

--> CREATE OR REPLACE VIEW worker_summary AS
SELECT
    w.name AS worker_name,
    COUNT(m.maintenance_id) AS total_services,
    COALESCE(SUM(i.total_amount), 0) AS total_revenue,
    ROUND(AVG(i.total_amount), 2) AS average_invoice
FROM worker w
LEFT JOIN maintenance m ON w.worker_id = m.worker_id
LEFT JOIN invoice i ON m.maintenance_id = i.maintenance_id
GROUP BY w.name;


```

worker_summary 1

	AZ worker_name	123 total_services	123 total_revenue	123 average_invoice
1	Ольга	2	43 000	21 500
2	Сергей	6	154 500	25 750

Обновить Save Cancel Экспорт данных... 200 2

2 строки получено - 0.032s, 2025-11-02 в 14:27:26

Рисунок 7 - Проверка

Задание 4. Использование аналитического представления в запросах

The screenshot shows the DBeaver IDE interface. On the left, there's a tree view of database objects under the schema 'albakhtin_iv'. A script tab titled '<postgres> Script' contains the following SQL code:

```
VALUES (31, 25000, 'оплачено');

SELECT * FROM invoice ORDER BY invoice_id DESC;

CREATE OR REPLACE VIEW worker_summary AS
SELECT
    w.name AS worker_name,
    COUNT(m.maintenance_id) AS total_services,
    COALESCE(SUM(i.total_amount), 0) AS total_revenue,
    ROUND(AVG(i.total_amount), 2) AS average_invoice
FROM worker w
LEFT JOIN maintenance m ON w.worker_id = m.worker_id
LEFT JOIN invoice i ON m.maintenance_id = i.maintenance_id
GROUP BY w.name;
SELECT * FROM worker_summary;

SELECT *
FROM worker_summary
WHERE average_invoice > 25000
ORDER BY average_invoice DESC;
```

Below the script, a result set titled 'worker_summary 1' is displayed in a table format. The table has four columns: 'worker_name', 'total_services', 'total_revenue', and 'average_invoice'. There is one row for 'Сергей'.

	worker_name	total_services	total_revenue	average_invoice
1	Сергей	6	154 500	25 750

At the bottom of the interface, there are various toolbars and status messages.

Рисунок 8 - Выведем всех работников, у которых средний счёт больше 25 000

Задание 5. Создание и обновление материализованного представления

The screenshot shows the pgAdmin interface. In the top window, there is a script editor with the following SQL code:

```
albakhtin_iv * <postgres> Script X invoice
LEFT JOIN maintenance m ON w.worker_id = m.worker_id
LEFT JOIN invoice i ON m.maintenance_id = i.maintenance_id
GROUP BY w.name;
SELECT * FROM worker_summary;

@ SELECT *
FROM worker_summary
WHERE average_invoice > 25000
ORDER BY average_invoice DESC;

@ CREATE MATERIALIZED VIEW client_total_spent AS
SELECT
    c.client_id,
    c.name,
    SUM(i.total_amount) AS total_spent
FROM client c
JOIN car ca ON c.client_id = ca.client_id
JOIN maintenance m ON ca.car_id = m.car_id
JOIN invoice i ON m.maintenance_id = i.maintenance_id
GROUP BY c.client_id, c.name;
```

In the bottom window, there is a statistics window titled "Статистика 1" (Statistics 1) with the following data:

Name	Value
Updated Rows	2
Execute time	0.024s
Start time	Sun Nov 02 14:30:42 MSK 2025
Finish time	Sun Nov 02 14:30:42 MSK 2025
Query	CREATE MATERIALIZED VIEW client_total_spent AS SELECT c.client_id, c.name, SUM(i.total_amount) AS total_spent FROM client c JOIN car ca ON c.client_id = ca.client_id JOIN maintenance m ON ca.car_id = m.car_id JOIN invoice i ON m.maintenance_id = i.maintenance_id GROUP BY c.client_id, c.name

Рисунок 9 - Создадим быстрое сводное представление по клиентам:
сколько они потратили на ТО

The screenshot shows a PostgreSQL client interface. In the top-left pane, there is a tree view of database objects under the schema 'albakhtin_iv'. A script tab titled 'Script' is open, containing the following SQL code:

```

GROUP BY w.name;
SELECT * FROM worker_summary;

SELECT *
FROM worker_summary
WHERE average_invoice > 25000
ORDER BY average_invoice DESC;

CREATE MATERIALIZED VIEW client_total_spent AS
SELECT
    c.client_id,
    c.name,
    SUM(i.total_amount) AS total_spent
FROM client c
JOIN car ca ON c.client_id = ca.client_id
JOIN maintenance m ON ca.car_id = m.car_id
JOIN invoice i ON m.maintenance_id = i.maintenance_id
GROUP BY c.client_id, c.name;

REFRESH MATERIALIZED VIEW client_total_spent;

```

In the bottom-right pane, there is a 'Статистика 1' (Statistics 1) panel displaying the following information:

Name	Value
Updated Rows	0
Execute time	0.024s
Start time	Sun Nov 02 14:33:02 MSK 2025
Finish time	Sun Nov 02 14:33:02 MSK 2025
Query	REFRESH MATERIALIZED VIEW client_total_spent

Рисунок 10 – Обновим данные

The screenshot shows a PostgreSQL client interface. In the top-left pane, there is a tree view of database objects under the schema 'albakhtin_iv'. A script tab titled 'Script' is open, containing the same SQL code as in Figure 10.

In the bottom-right pane, there is a results grid titled 'client_total_spent 1' showing the data from the materialized view:

client_id	name	total_spent
1	Иван	135 000
2	Мария	18 000

The status bar at the bottom indicates: '2 строк получено - 0.010s, 2025-11-02 в 14:34:27'.

Рисунок 11 – Проверка

Задание 6. Разработка пользовательской функции для аналитических вычислений

The screenshot shows the pgAdmin interface with a query editor window titled 'albakhtin_iv' containing the following SQL code:

```
FROM worker w
ORDER BY revenue DESC;

CREATE OR REPLACE FUNCTION get_worker_total_revenue(p_worker_id INT)
RETURNS NUMERIC(12,2)
LANGUAGE plpgsql
AS $$
DECLARE
    total NUMERIC(12,2);
BEGIN
    SELECT COALESCE(SUM(i.total_amount), 0)
    INTO total
    FROM maintenance m
    JOIN invoice i ON m.maintenance_id = i.maintenance_id
    WHERE m.worker_id = p_worker_id;
    RETURN total;
END;
$$;
```

Below the editor is a statistics panel titled 'Статистика 1' (Statistics 1) showing the following details:

Name	Value
Updated Rows	0
Execute time	0.015s
Start time	Sun Nov 02 14:38:15 MSK 2025
Finish time	Sun Nov 02 14:38:15 MSK 2025
Query	CREATE OR REPLACE FUNCTION get_worker_total_revenue(p_worker_id INT) RETURNS NUMERIC(12,2) LANGUAGE plpgsql AS \$\$ DECLARE total NUMERIC(12,2); BEGIN SELECT COALESCE(SUM(i.total_amount), 0) INTO total FROM maintenance m JOIN invoice i ON m.maintenance_id = i.maintenance_id

Рисунок 12 –Возвращаем общую сумму по конкретному работнику

The screenshot shows the pgAdmin interface with a query editor window containing the following SQL code:

```
GROUP BY c.client_id, c.name;
SELECT * FROM client_total_spent;
```

Below the editor is a results grid titled 'client_total_spent 1' showing the following data:

client_id	name	total_spent
1	Иван	135 000
2	Мария	18 000

Рисунок 13 - Текущее состояние

The screenshot shows a database interface with a query editor and a statistics window.

```
INSERT INTO invoice (maintenance_id, total_amount, payment_status)
VALUES (32, 7000, 'Оплачено');
```

Статистика 1

Name	Value
Updated Rows	1
Execute time	0,026s
Start time	Sun Dec 21 13:40:57 MSK 2025
Finish time	Sun Dec 21 13:40:58 MSK 2025
Query	INSERT INTO invoice (maintenance_id, total_amount, payment_status) VALUES (32, 7000, 'Оплачено')

Рисунок 14 – Меняем исходные данные

The screenshot shows a database interface with a query editor and a table viewer.

```
INSERT INTO invoice (maintenance_id, total_amount, payment_status)
VALUES (32, 7000, 'Оплачено');

SELECT * FROM client_total_spent;
```

client_total_spent 1

Таблица	123 client_id	AZ name	123 total_spent
	1	Иван	135 000
	2	Мария	18 000

Рисунок 15 – Ничего не поменялось

The screenshot shows a database interface with a query editor and a statistics window.

```
REFRESH MATERIALIZED VIEW client_total_spent;
```

Статистика 1

Name	Value
Updated Rows	0
Execute time	0,06s
Start time	Sun Dec 21 13:43:15 MSK 2025
Finish time	Sun Dec 21 13:43:15 MSK 2025
Query	REFRESH MATERIALIZED VIEW client_total_spent

Рисунок 16 – Обновляем представление

client_total_spent

client_id	name	total_spent
1	Иван	32 684 903,83
2	Мария	34 712 442,83

Рисунок 17 – После обновления (данные поменялись слишком сильно, так как переделываю я эту работу после всех последующих)

Задание 7. Разработка хранимой процедуры для выполнения сложной операции

```

CREATE OR REPLACE PROCEDURE process_part_usage(
    p_part_id INT,
    p_quantity INT,
    OUT p_success BOOLEAN,
    OUT p_message TEXT
)
LANGUAGE plpgsql AS $$
DECLARE
    current_stock INT;
BEGIN
    SELECT quantity INTO current_stock FROM part WHERE part_id = p_part_id;
    IF NOT FOUND THEN
        p_success := FALSE;
        p_message := 'Ошибка: запчасть с ID ' || p_part_id || ' не найдена.';
        RETURN;
    END IF;

    IF current_stock >= p_quantity THEN
        UPDATE part SET quantity = quantity - p_quantity WHERE part_id = p_part_id;
        p_success := TRUE;
        p_message := 'Списано ' || p_quantity || ' шт. Остаток: ' || (current_stock - p_quantity);
    ELSE
        p_success := FALSE;
        p_message := 'Недостаточно запчастей. Доступно: ' || current_stock;
    END IF;
END;
$$;

```

Name	Value
Updated Rows	0
Execute time	0.013s
Start time	Sun Nov 02 14:40:44 MSK 2025
Finish time	Sun Nov 02 14:40:44 MSK 2025
Query	CREATE OR REPLACE PROCEDURE process_part_usage(
	p_part_id INT,
	p_quantity INT,
	OUT p_success BOOLEAN,
	OUT p_message TEXT
)
	LANGUAGE plpgsql AS \$\$
	DECLARE
	current_stock INT;
	BEGIN
	SELECT quantity INTO current stock FROM part WHERE part id = o part id:

Рисунок 18 – Списываем запас запчастей (part), если их хватает, и записывает в таблицу maintenance_work.

```


--> *<postgres> Script X invoice
    FROM information_schema.columns
    WHERE table_name = 'part';

    ALTER TABLE part
    ADD COLUMN quantity INT DEFAULT 0;

    UPDATE part SET quantity = 10 WHERE part_id = 1;
    UPDATE part SET quantity = 5 WHERE part_id = 2;

    SELECT * FROM part;

    DO $$ 
    DECLARE
        success BOOLEAN;
        message TEXT;
    BEGIN
        CALL albakhtin_iv.process_part_usage(1, 3, success, message);
        RAISE NOTICE 'Успешно: %, Сообщение: %', success, message;
    END $$;


```

Статистика 1

Name	Value
Updated Rows	0
Execute time	0.013s
Start time	Sun Nov 02 14:46:08 MSK 2025
Finish time	Sun Nov 02 14:46:08 MSK 2025
Query	DO \$\$ DECLARE success BOOLEAN; message TEXT; BEGIN CALL albakhtin_iv.process_part_usage(1, 3, success, message); RAISE NOTICE 'Успешно: %, Сообщение: %', success, message; END \$\$;

Рисунок 19 - Проверка

Задание 8: Демонстрация вызова хранимой процедуры

```


IF current_stock IS NULL THEN
    p_success := FALSE;
    p_message := 'Ошибка: запчасть с ID ' || p_part_id || ' не найдена.';
    RETURN;
END IF;
IF current_stock >= p_quantity THEN
    UPDATE part
    SET quantity = quantity - p_quantity
    WHERE part_id = p_part_id;

    p_success := TRUE;
    p_message := 'Списано ' || p_quantity || ' шт. Остаток: ' || (current_stock - p_quantity);
ELSE
    p_success := FALSE;
    p_message := 'Недостаточно запчастей. Доступно: ' || current_stock;
END IF;
$$;


```

Статистика 1

Name	Value
Updated Rows	0
Execute time	0.011s
Start time	Sun Nov 02 14:48:11 MSK 2025
Finish time	Sun Nov 02 14:48:11 MSK 2025
Query	CREATE OR REPLACE PROCEDURE process_part_usage(p_part_id INT, p_quantity INT, OUT p_success BOOLEAN, OUT p_message TEXT) LANGUAGE plpgsql AS \$\$ DECLARE current_stock INT; BEGIN

Рисунок 20 – Процедура списания запчастей со склада

```


    p_success := FALSE;
    p_message := 'Недостаточно запчастей. Доступно: ' || 
END IF;
END;
$$;

@ SELECT routine_name, routine_type, routine_schema
FROM information_schema.routines
WHERE routine_name = 'process_part_usage';

@ DO $$
DECLARE
    success BOOLEAN;
    message TEXT;
BEGIN
    CALL albakhtin_iv.process_part_usage(1, 3, success, mess
    RAISE NOTICE 'Успешно: %, Сообщение: %', success, messag
END $$;


```

Статистика 1 ×

Name	Value
Updated Rows	0
Execute time	0.011s
Start time	Sun Nov 02 14:50:00 MSK 2025
Finish time	Sun Nov 02 14:50:00 MSK 2025
Query	DO \$\$ DECLARE success BOOLEAN; message TEXT; BEGIN CALL albakhtin_iv.process_part_usage(1, 3, success, message); RAISE NOTICE 'Успешно: %, Сообщение: %', success, message; END \$\$;

Рисунок 21 – Проверка

```


    EW;
$$;

@ SELECT routine_name, routine_type, routine_schema
FROM information_schema.routines
WHERE routine_name = 'process_part_usage';

@ DO $$
DECLARE
    success BOOLEAN;
    message TEXT;
BEGIN
    CALL albakhtin_iv.process_part_usage(1, 3, success, mess
    RAISE NOTICE 'Успешно: %, Сообщение: %', success, messag
END $$;

@ SELECT part_id, name, quantity
FROM part
ORDER BY part_id;


```

part1 ×

part_id	name	quantity
1	Фильтр масляный	7
2	Тормозные колодки	5

Рисунок 22 – Проверка остатков

ВЫВОД

В ходе выполнения работы была создана процедура process_part_usage, выполняющая автоматическое списание запчастей со склада. Процедура корректно обрабатывает случаи успешного списания, нехватки деталей и отсутствия запчасти. Работа подтверждает успешное применение процедур для автоматизации учёта в базе данных.