



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 **результатирующей 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 database management interface. At the top, there are tabs for 'albakhtin_iv', '*<dbstud> Script-9', 'review_id', 'maintenance', and 'mair'. The main editor displays two SQL queries. The first query is a complex one using a CROSSTAB function to generate a pivot table of invoice data. The second query, which is highlighted, is 'SELECT * FROM worker LIMIT 10;'. Below the editor, there is a table view for 'worker 1'. It shows a table with columns: worker_id, name, position, and phone. The first two rows of data are visible.

```
SELECT * FROM crosstab(  
    'SELECT  
        EXTRACT(YEAR FROM m.start_date) AS year,  
        EXTRACT(QUARTER FROM m.start_date) AS quarter  
        SUM(i.total_amount)  
    FROM invoice i  
    JOIN maintenance m ON i.maintenance_id = m.mainte  
    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)
```

```
SELECT * FROM worker LIMIT 10;
```

	123 worker_id	A-Z name	A-Z position	A-Z phone
1	1	Сергей	Механик	+79112223344
2	2	Ольга	Диагност	+79210009988

Таблица 2. Таблица maintenance (ТО)

albaktin_iv

*<dbstud> Script-9

review_id

maintenance

maintenance_work

maintenance

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

maintenance 1

SELECT maintenance_id, car_id, worker_id, start_date

Введите SQL выражение чтобы отфильтровать результаты

	123 maintenance_id	123 car_id	123 worker_id	start_date	end_date	A-Z 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)

<div> <div> <div>albakhtin_iv</div> <div>*<dbstud> Script-9 X</div> <div>review_id</div> <div>maintenance</div> <div>maintenance_work</div> <div>ma</div> </div> <div> <div>SELECT * FROM worker LIMIT 10;</div> <div> <div>SELECT maintenance_id, car_id, worker_id, start_date, end_date, stat</div> <div>FROM maintenance</div> <div>LIMIT 10;</div> </div> <div> <div>SELECT invoice_id, maintenance_id, total_amount, payment_status</div> <div>FROM invoice</div> <div>LIMIT 10;</div> </div> </div> </div>					
<div> <div>invoice 1 X</div> <div> <div>SELECT invoice_id, maintenance_id, total_amount, pz</div> <div>Введите SQL выражение чтобы отфильтровать результаты</div> </div> </div>					
<div> <div>Таблица</div> <div>Текст</div> <div>Таблица</div> <div>Текст</div> <div>Таблица</div> <div>Текст</div> <div>Таблица</div> <div>Текст</div> <div>Таблица</div> <div>Текст</div> <div>Таблица</div> <div>Текст</div> </div>	<div> <div>123 invoice_id</div> <div>123 maintenance_id</div> <div>123 total_amount</div> <div>A-Z payment_status</div> </div>				
	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

Задание 1. Создание модифицируемого представления

The screenshot displays a PostgreSQL IDE interface. The top pane shows two SQL queries. The first query is a JOIN operation on the 'cat_tree' table. The second query is a 'CREATE OR REPLACE VIEW' statement for 'paid_invoices', which selects columns from the 'invoice' table where the 'payment_status' is 'Оплачено'. The bottom pane shows the execution statistics for the second query, indicating that 0 rows were updated.

```
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';
```

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

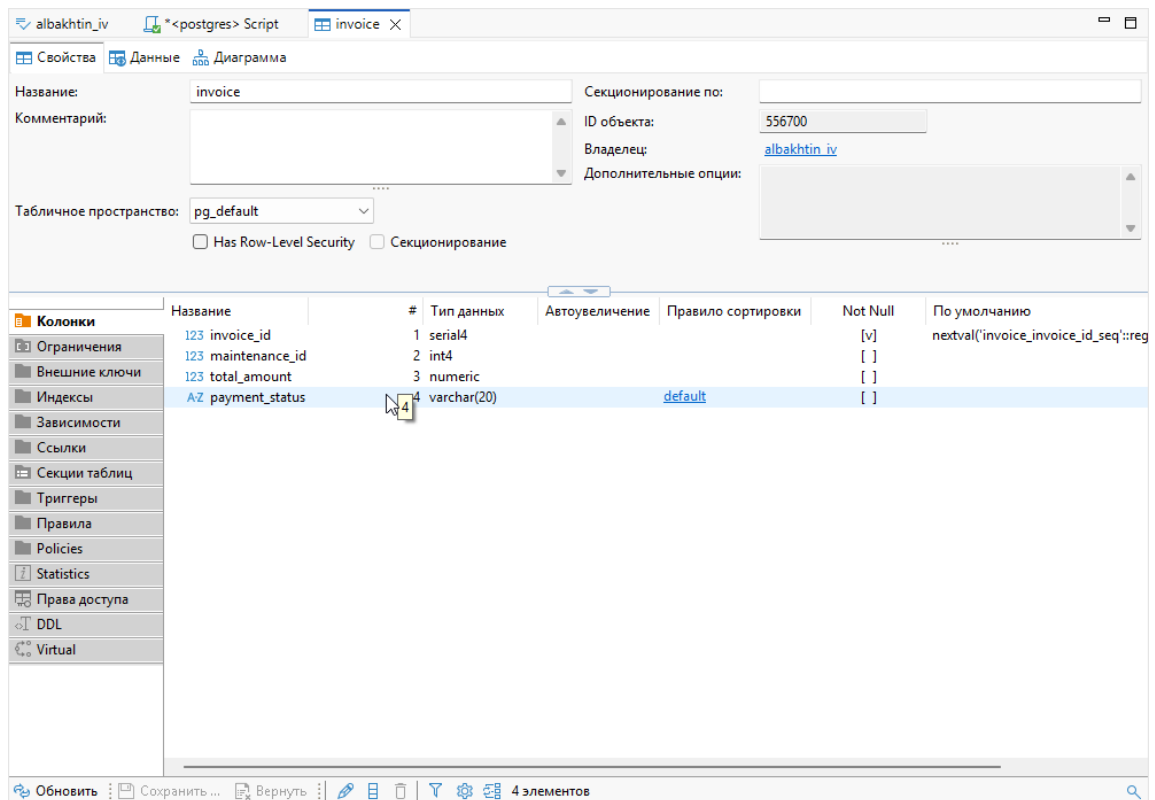


Рисунок 2 - Все оплаченные счета с фильтром payment_status = 'Оплачено' часть 2

Задание 2. Модификация данных через представление

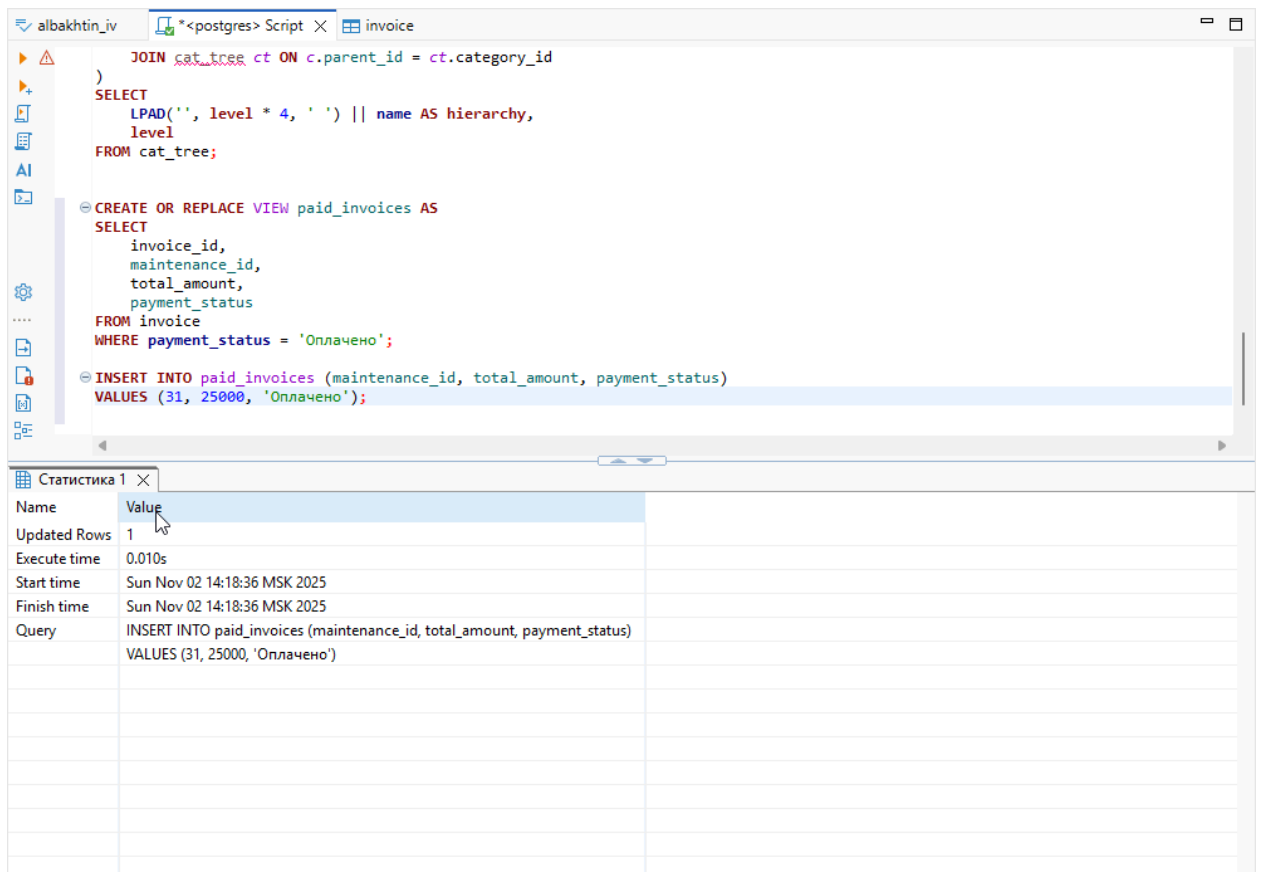


Рисунок 3 – Добавление новой записи через представление

The screenshot shows a PostgreSQL script editor with the following SQL commands:

```

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');

-- DELETE FROM paid_invoices
-- WHERE invoice_id = 1;

```

Below the script, a statistics window titled "Статистика 1" displays the following information:

Name	Value
Updated Rows	0
Execute time	0.011s
Start time	Sun Nov 02 14:19:27 MSK 2025
Finish time	Sun Nov 02 14:19:28 MSK 2025
Query	DELETE FROM paid_invoices WHERE invoice_id = 1

Рисунок 4 - Удаление записи через представление

The screenshot shows a PostgreSQL script editor with the following SQL commands:

```

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');

-- 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;

```

Below the script, a table view titled "invoice 1" displays the results of the query `SELECT * FROM invoice ORDER BY invoice_id DESC`. The table has 14 rows and 5 columns: invoice_id, maintenance_id, total_amount, and payment_status.

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

The bottom of the screenshot shows a status bar indicating "14 строк получено - 0.008s, 2025-11-02 в 14:25:15".

Рисунок 5 - Проверка

Задание 3. Создание немодифицируемого аналитического представления

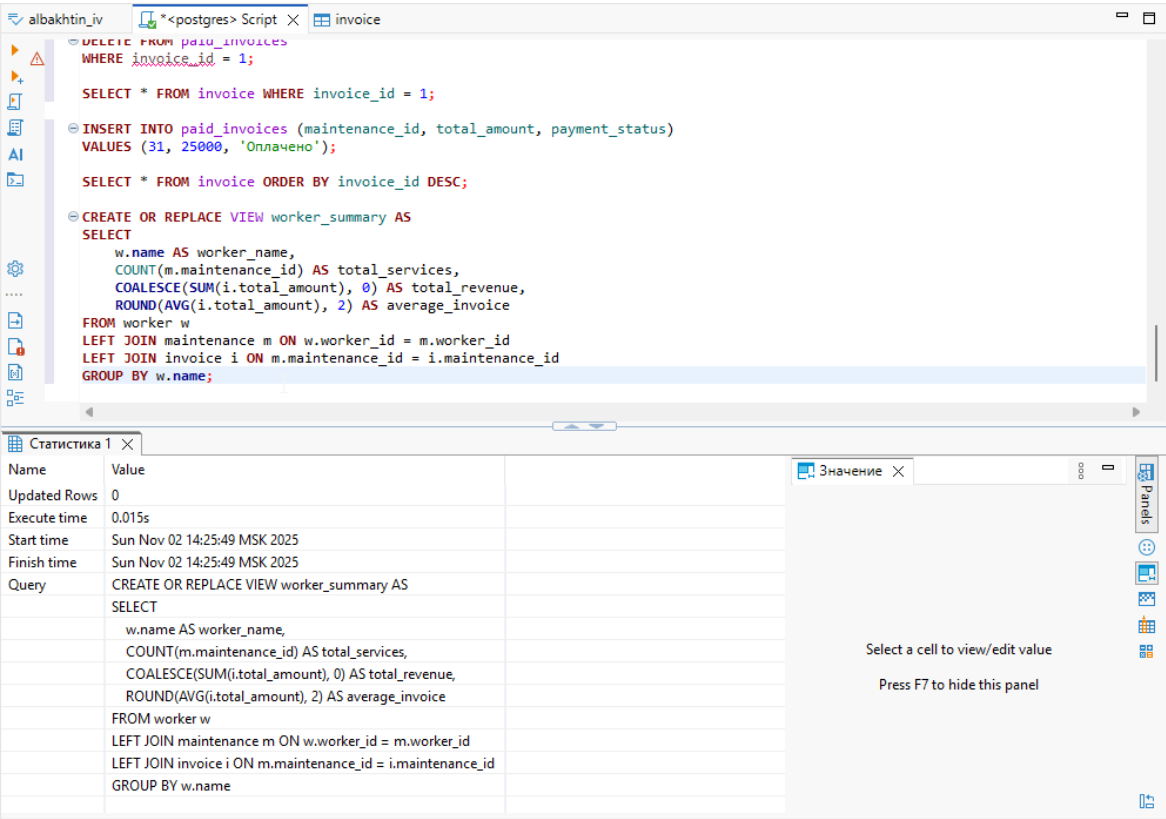


Рисунок 6 – Создание сводки по каждому сотруднику

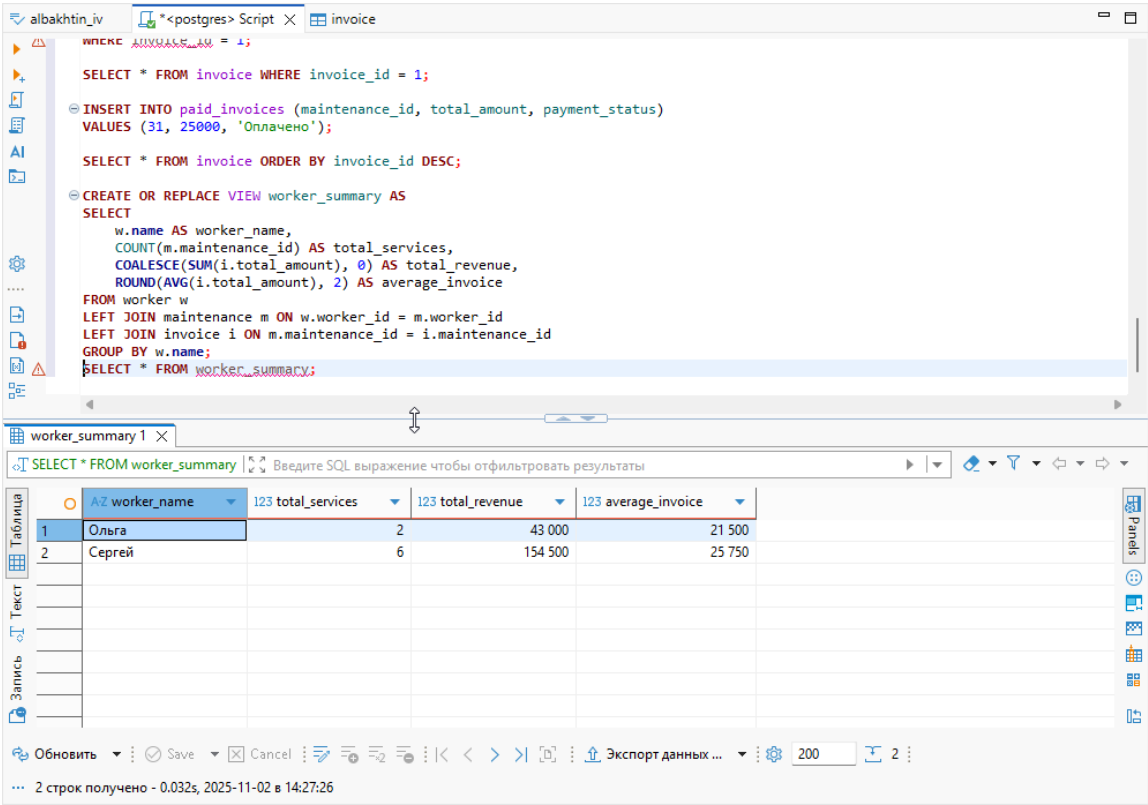
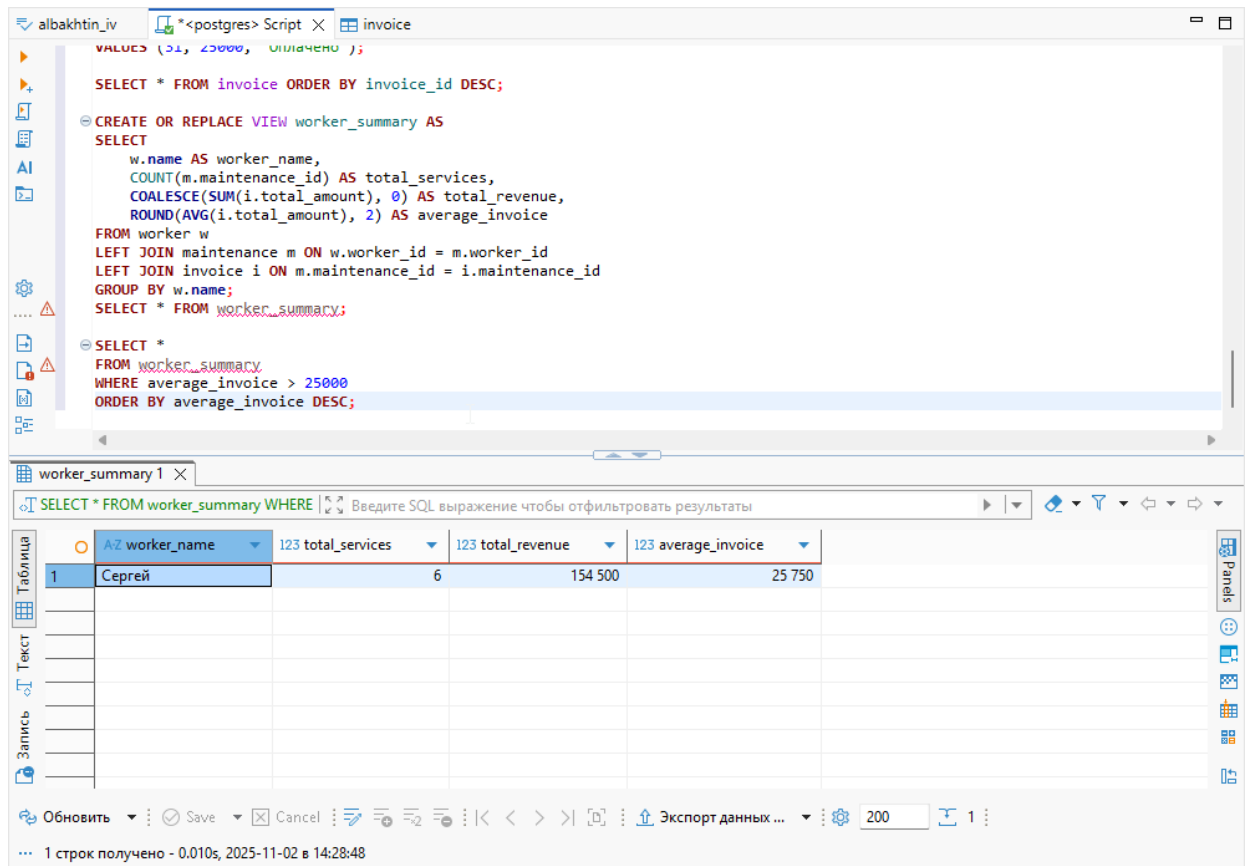


Рисунок 7 - Проверка

Задание 4. Использование аналитического представления в запросах



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

Задание 5. Создание и обновление материализованного Представления

The screenshot displays a PostgreSQL client window with a script editor and a statistics panel. The script editor contains three SQL queries. The first query joins maintenance, worker, and invoice tables. The second query filters for high average invoice amounts. The third query creates a materialized view for client spending.

```
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;
```

The statistics panel, titled "Статистика 1", shows the execution details for the third query. The "Updated Rows" value is 2.

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

albakhitin_iv | *<postgres> Script X invoice

```
UNION 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;
```

Статистика 1 X

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

albakhitin_iv | *<postgres> Script X invoice

```
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;  
  
SELECT * FROM client_total_spent;
```

client_total_spent 1 X

SELECT * FROM client_total_spent | Введите SQL выражение чтобы отфильтровать результаты

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

Обновить Save Cancel Экспорт данных ... 200 2

2 строк получено - 0.010s, 2025-11-02 в 14:34:27

Рисунок 11 – Проверка

Задание 6. Разработка пользовательской функции для аналитических вычислений

albakhitin_iv

*<postgres> Script X invoice

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;
\$\$;

Статистика 1 X

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

GROUP BY c.client_id, c.name;

SELECT * FROM client_total_spent;

client_total_spent 1 X

SELECT * FROM client_total_spent | Введите SQL выражение чтобы отфильтр

123 client_id	A-Z name	123 total_spent
1	Иван	135 000
2	Мария	18 000

Рисунок 13 - Текущее состояние

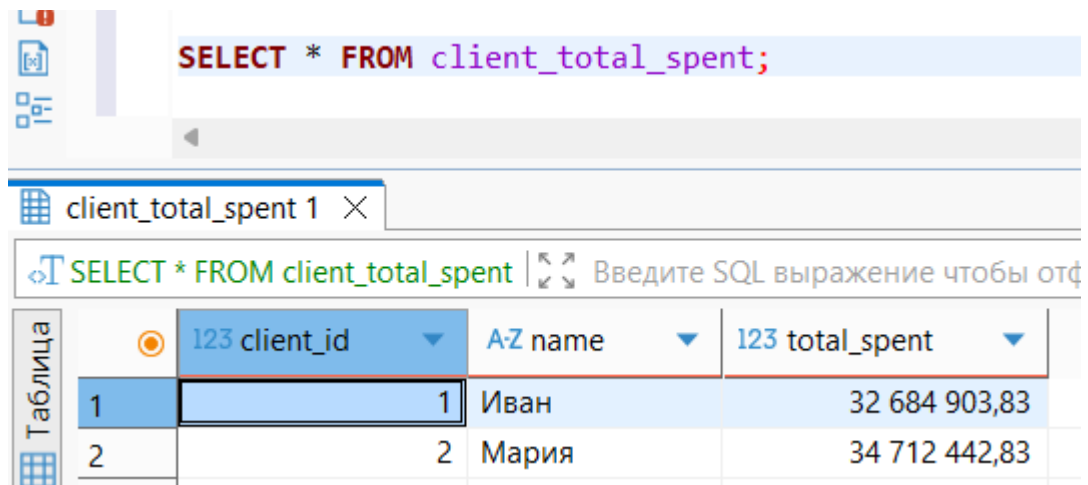
<pre> INSERT INTO invoice (maintenance_id, total_amount, payment_status) VALUES (32, 7000, 'Оплачено'); </pre>	
Статистика 1	
Name	Value
Updated Rows	1
Execute time	0,026s
Start time	Sun Dec 21 13:40:57 MSK 2025
Finish time	Sun Dec 21 13:40:58 MSK 2025
Query	INSERT INTO invoice (maintenance_id, total_amount,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

Рисунок 14 – Меняем исходные данные

Рисунок 15 – Ничего не поменялось

<pre> REFRESH MATERIALIZED VIEW client_total_spent; </pre>	
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

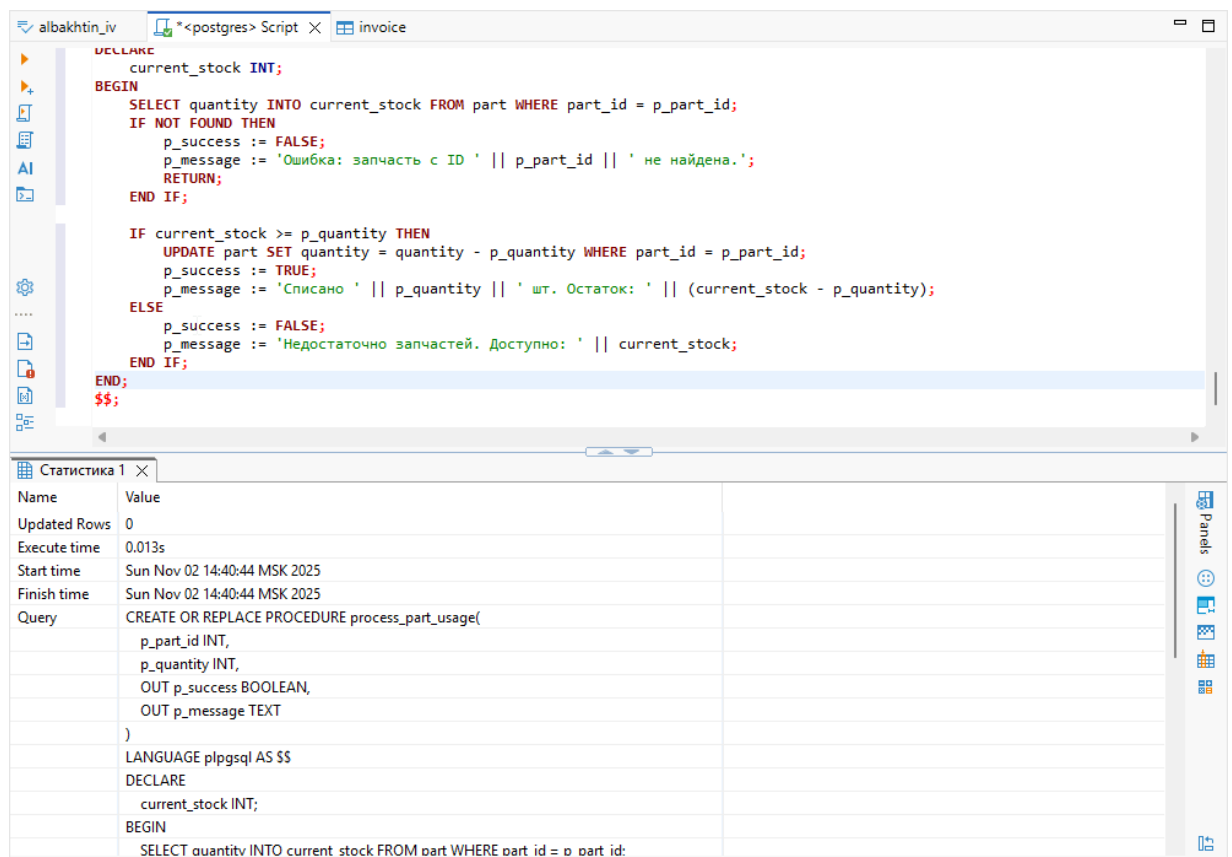


The screenshot shows a database client interface. At the top, a SQL query is entered in a text box: `SELECT * FROM client_total_spent;`. Below the query, a tab labeled "client_total_spent 1" is active. The results of the query are displayed in a table with the following columns: "client_id", "name", and "total_spent". The table contains two rows of data.

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



The screenshot shows a database client interface with a PL/SQL procedure named "process_part_usage" defined. The procedure takes "p_part_id" and "p_quantity" as input parameters and returns "p_success" (BOOLEAN) and "p_message" (TEXT). The procedure logic includes checking the current stock and updating it if necessary. Below the procedure definition, a "Статистика 1" (Statistics 1) panel shows the execution details of the procedure.

```

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 = p_part_id;

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

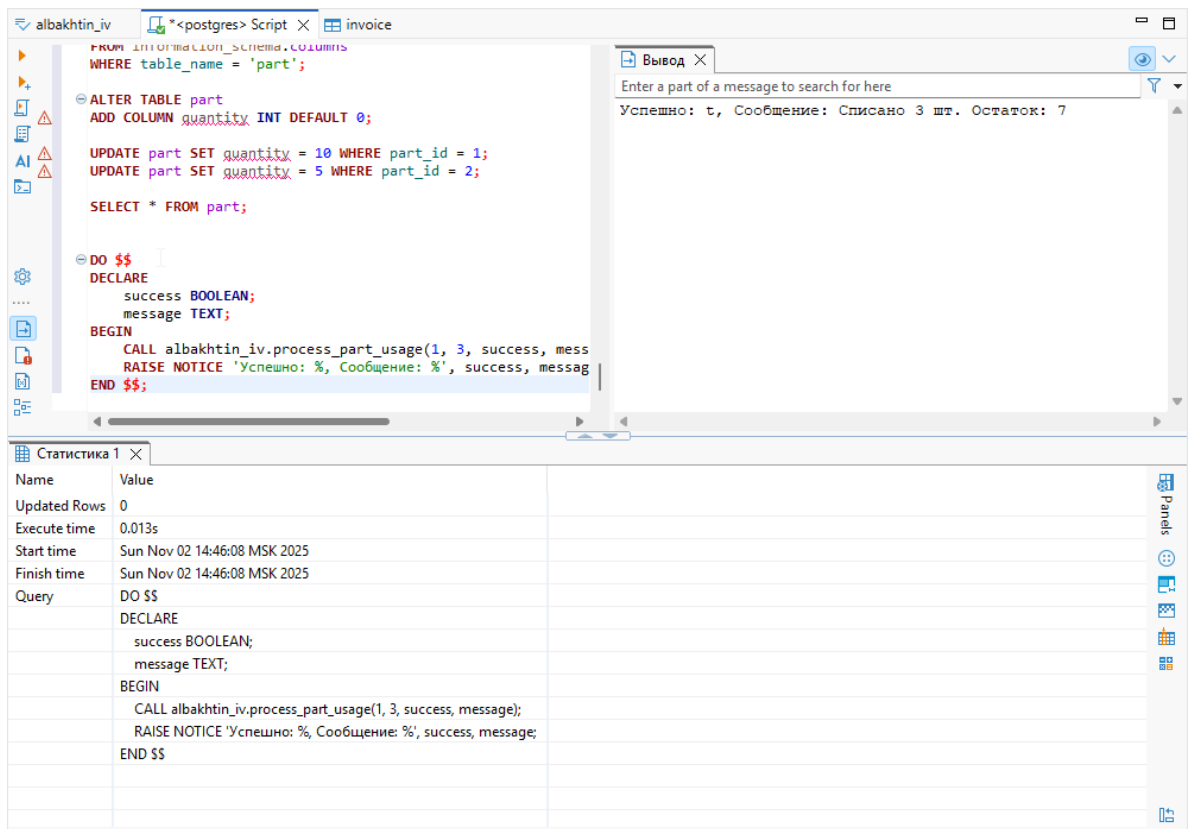


Рисунок 19 - Проверка

Задание 8: Демонстрация вызова хранимой процедуры

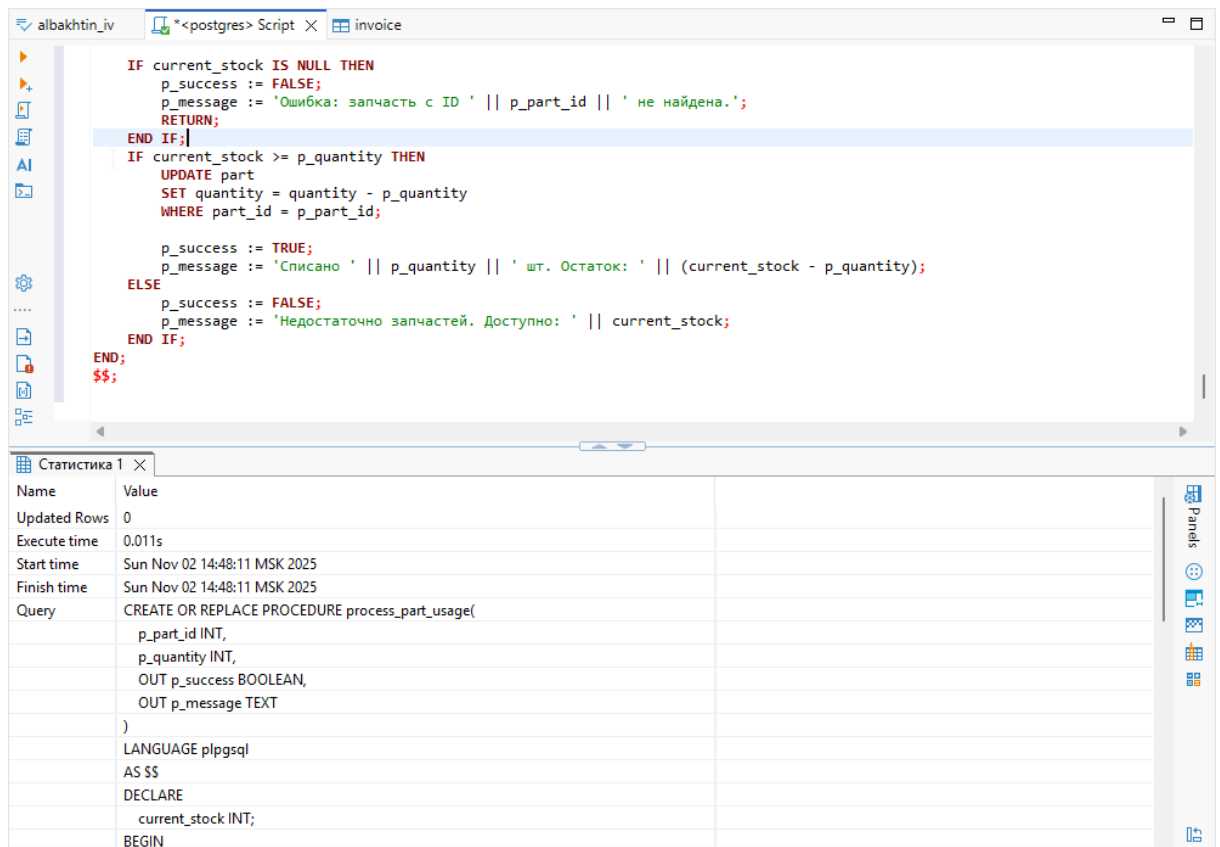


Рисунок 20 – Процедура списания запчастей со склада

The screenshot shows a PostgreSQL IDE window with a script named 'invoice'. The script contains the following SQL code:

```

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

```

The output window shows the following messages:

```

Успешно: t, Сообщение: Списано 3 шт. Остаток: 7
Успешно: t, Сообщение: Списано 3 шт. Остаток: 7

```

The statistics window shows the following data:

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

Рисунок 21 – Проверка

The screenshot shows a PostgreSQL IDE window with a script named 'invoice'. The script contains the following SQL code:

```

END;
$$;

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

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

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

```

The output window shows the following messages:

```

Успешно: t, Сообщение: Списано 3 шт. Остаток: 7
Успешно: t, Сообщение: Списано 3 шт. Остаток: 7

```

The table view shows the following data:

part_id	name	quantity
1	Фильтр масляный	7
2	Тормозные колодки	5

The bottom status bar indicates: 2 строк получено - 0.010s, 2025-11-02 в 14:50:25

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.