



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 №8
по дисциплине «Разработка баз данных»

Студент группы *ИКБО-50-23. Павлов Н.С.*

(подпись)

Преподаватель *Мажей Я. В.*

(подпись)

Москва 2025 г.

СОДЕРЖАНИЕ

1. ПОСТАНОВКА ЗАДАЧИ.....	3
2 ВЫПОЛНЕНИЕ РАБОТЫ.....	5
2.1 ИСХОДНЫЕ ТАБЛИЦЫ.....	5
2.2 ЗАДАНИЕ 1. ХРАНЕНИЕ ПАРОЛЕЙ (BYTEA)	6
2.3 ЗАДАНИЕ 2. СЕКЦИОНИРОВАНИЕ ТАБЛИЦЫ ЛОГОВ	7
2.4 ЗАДАНИЕ 3. ГЕНЕРАЦИЯ ДАННЫХ	9
2.5 ЗАДАНИЕ 4. АНАЛИЗ И ПОИСК ПО JSONB	10
2.6 ЗАДАНИЕ 5. МОДИФИКАЦИЯ ДАННЫХ JSONB.....	11

1. ПОСТАНОВКА ЗАДАЧИ

Цель: освоение методов работы со сложными и неструктуризованными типами данных (JSONB, BYTEA) в PostgreSQL, а также получение практических навыков реализации горизонтального масштабирования базы данных с помощью декларативного секционирования.

Задачи:

Задание №1: хранение паролей (BYTEA)

1. Создать справочную таблицу ролей и таблицу системных пользователей (или модифицировать существующую).
2. В таблице пользователей предусмотреть поля для хранения «сырого» пароля (только для учебной демонстрации) и его хеша (тип BYTEA).
3. Сохранить хеши паролей пользователей, используя функцию хеширования digest.
4. Выполнить запрос, проверяющий соответствие введенного пароля сохраненному хешу. Одна проверка должна быть успешной, другая - нет (допустимо сделать это в одном запросе).

Задание №2: секционирование таблицы логов

1. Создать новую секционированную таблицу для хранения логов событий.
2. Использовать стратегию секционирования по диапазонам (RANGE) по полю даты.
3. Создать две секции для периодов:
 - 2-е полугодие 2024 (с 2024-07-01 по 2025-01-01).
 - 1-е полугодие 2025 (с 2025-01-01 по 2025-07-01).
4. Создать секцию по умолчанию, куда будут попадать все данные, выходящие за рамки указанных диапазонов (будущее время).

Задание №3: генерация данных

Написать скрипт на PL/pgSQL для генерации 20 000 записей в диапазоне дат, покрывающем созданные секции. Сгенерированные данные должны содержать JSON-строку с различным набором полей для разных типов

событий (например, чтобы логи «продажи» отличались по структуре от логов «входа»).

Вывести сгенерированные данные – показать запросом SELECT, что данные успешно созданы и физически распределились по разным таблицам секциям.

Задание №4: анализ и поиск по JSONB

Написать три запроса:

1. Фильтрация по значению – найти записи, где числовое поле внутри JSON удовлетворяет математическому условию (например, > 800), используя операторы извлечения $->$ и $->>$.
2. Поиск по вхождению – найти записи, содержащие точный фрагмент JSON-структуры (например, `{"quantity": 5}`), используя оператор `@>`.
3. Агрегация – посчитать сумму или среднее значение числового поля из JSON-документа.

Задание №5: модификация данных JSONB

Продемонстрировать изменение данных внутри JSON-поля:

1. Массовое добавление нового поля во все записи определенного типа.
2. Точечное изменение значения по ключу в конкретной записи.

2 ВЫПОЛНЕНИЕ РАБОТЫ

2.1 ИСХОДНЫЕ ТАБЛИЦЫ

The screenshot shows the 'employee' table in MySQL Workbench. The table has 10 rows and 11 columns. The columns are: employee_id, position_id, manager_id, surname, name, fathers_name, phone, registration_address, and employment_date. The data includes names like Волков, Соколов, Кузнецов, etc., and addresses in Moscow.

id	employee_id	position_id	manager_id	surname	name	fathers_name	phone	registration_address	employment_date
1	1	7	[NULL]	Волков	Александр	Михайлович	+79167654321	г. Москва, ул. Пушкина, 15	2022-
2	2	3	1	Соколов	Игорь	Васильевич	+79167654322	г. Москва, ул. Гагарина, 20	2022-
3	3	2	2	Кузнецов	Павел	Сергеевич	+79167654323	г. Москва, ул. Лермонтова, 2	2022-
4	4	1	2	Попов	Михаил	Андреевич	+79167654324	г. Москва, ул. Чехова, 30	2022-
5	5	4	2	Васильев	Денис	Олегович	+79167654325	г. Москва, ул. Толстого, 35	2022-
6	6	6	1	Петров	Артем	Викторович	+79167654326	г. Москва, ул. Достоевского, 1	2022-
7	7	5	6	Михайлов	Кирилл	[NULL]	+79167654327	г. Москва, ул. Тургенева, 45	2022-
8	8	8	2	Новиков	Евгений	Алексеевич	+79167654328	г. Москва, ул. Гоголя, 50	2022-
9	9	2	2	Фролов	Роман	Дмитриевич	+79167654329	г. Москва, ул. Некрасова, 55	2022-
10	10	2	2	Алексеев	Владимир	Павлович	+79167654330	г. Москва, ул. Островского, 1	2022-

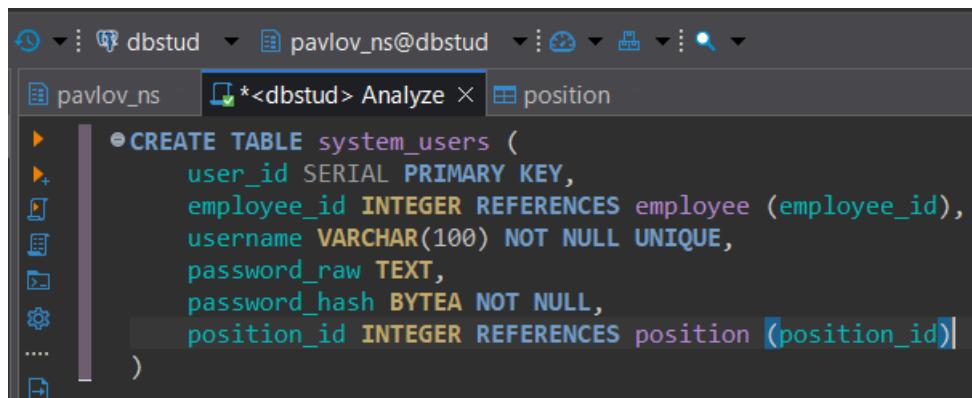
Рисунок 1 – Таблица employee

The screenshot shows the 'position' table in MySQL Workbench. The table has 8 rows and 3 columns. The columns are: position_id, title, and salary. The data includes titles like Мастер-приемщик, Веломеханик, etc., and salaries ranging from 45 000 to 80 000.

id	position_id	title	salary
1	1	Мастер-приемщик	60 000
2	2	Веломеханик	55 000
3	3	Старший механик	70 000
4	4	Диагност	65 000
5	5	Менеджер по закупкам	58 000
6	6	Администратор	45 000
7	7	Главный механик	80 000
8	8	Специалист по электронике	72 000

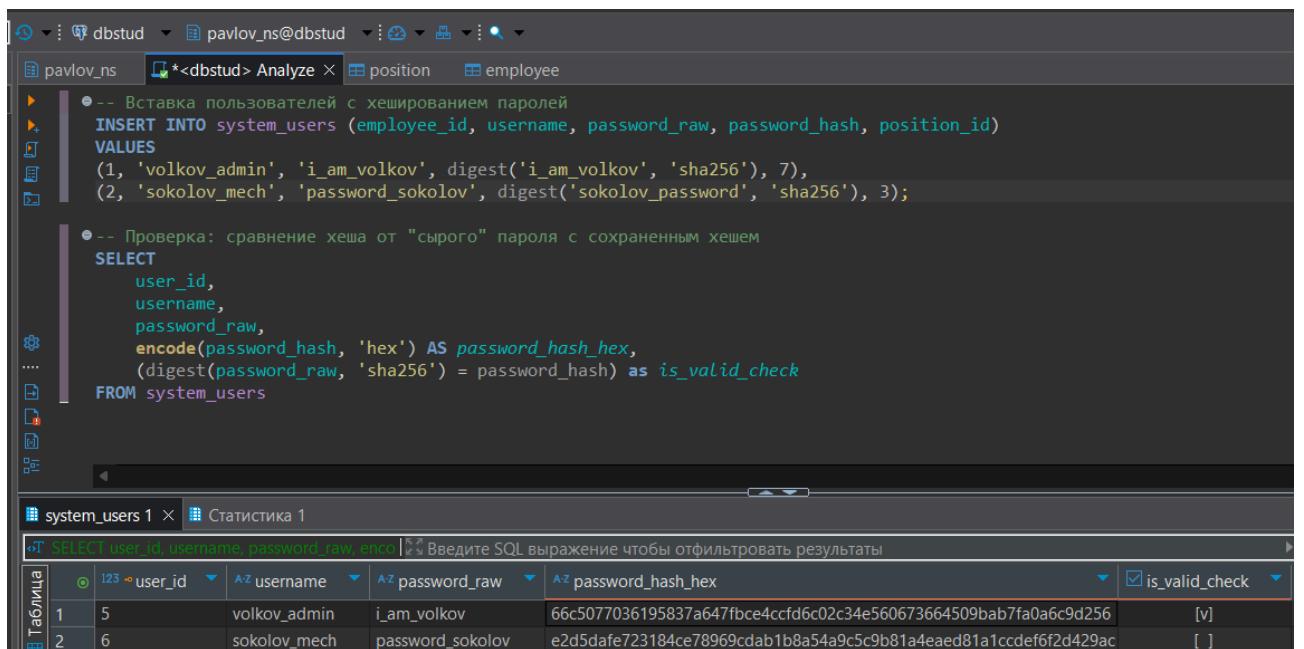
Рисунок 2 – Таблица position

2.2 ЗАДАНИЕ 1. ХРАНЕНИЕ ПАРОЛЕЙ (BYTEA)



```
CREATE TABLE system_users (
    user_id SERIAL PRIMARY KEY,
    employee_id INTEGER REFERENCES employee (employee_id),
    username VARCHAR(100) NOT NULL UNIQUE,
    password_raw TEXT,
    password_hash BYTEA NOT NULL,
    position_id INTEGER REFERENCES position (position_id)
)
```

Рисунок 3 – Создание таблицы пользователей



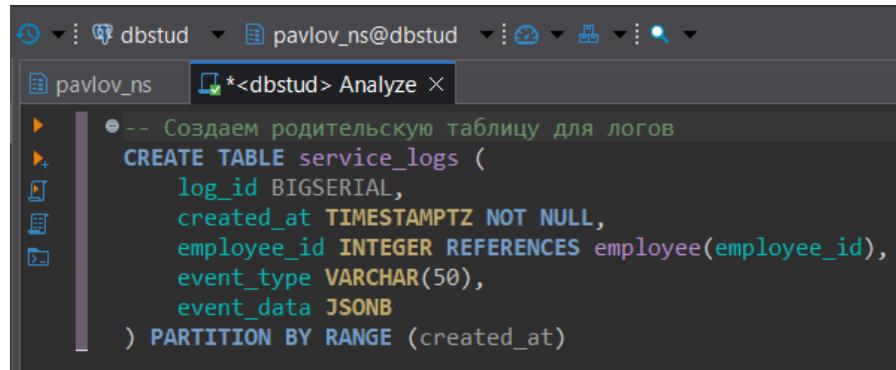
```
-- Вставка пользователей с хешированием паролей
INSERT INTO system_users (employee_id, username, password_raw, password_hash, position_id)
VALUES
(1, 'volkov_admin', 'i_am_volkov', digest('i_am_volkov', 'sha256'), 7),
(2, 'sokolov_mech', 'password_sokolov', digest('sokolov_password', 'sha256'), 3);

-- Проверка: сравнение хеша от "сырого" пароля с сохраненным хешем
SELECT
    user_id,
    username,
    password_raw,
    encode(password_hash, 'hex') AS password_hash_hex,
    (digest(password_raw, 'sha256') = password_hash) AS is_valid_check
FROM system_users
```

Таблица	user_id	username	password_raw	password_hash_hex	is_valid_check
system_users	1	volkov_admin	i_am_volkov	66c5077036195837a647fbce4ccfd6c02c34e560673664509bab7fa0a6c9d256	[v]
system_users	2	sokolov_mech	password_sokolov	e2d5d5afe723184ce78969cdab1b8a54a9c5c9b81a4eaed81a1ccdef6f2d429ac	[]

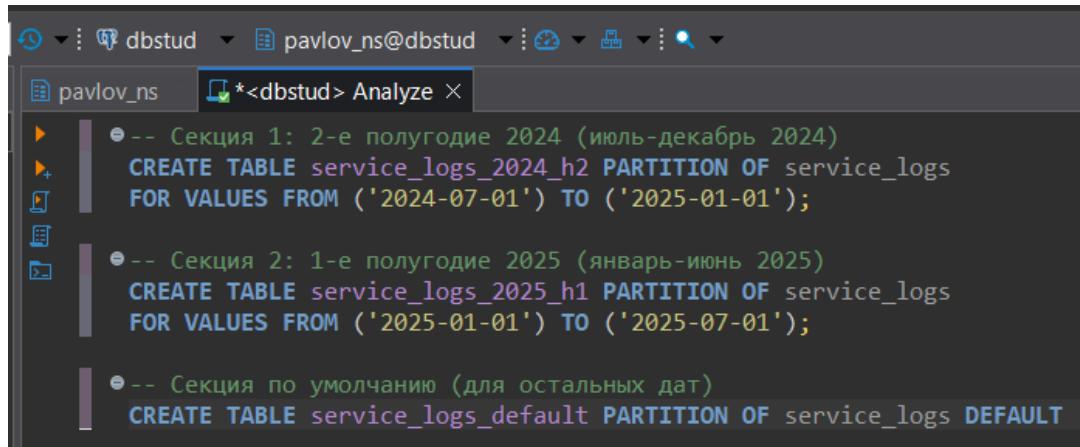
Рисунок 4 – Вставка и проверка данных пользователей

2.3 ЗАДАНИЕ 2. СЕКЦИОНИРОВАНИЕ ТАБЛИЦЫ ЛОГОВ



```
-- Создаем родительскую таблицу для логов
CREATE TABLE service_logs (
    log_id BIGSERIAL,
    created_at TIMESTAMPTZ NOT NULL,
    employee_id INTEGER REFERENCES employee(employee_id),
    event_type VARCHAR(50),
    event_data JSONB
) PARTITION BY RANGE (created_at)
```

Рисунок 5 – Создание секционированной таблицы



```
-- Секция 1: 2-е полугодие 2024 (июль–декабрь 2024)
CREATE TABLE service_logs_2024_h2 PARTITION OF service_logs
FOR VALUES FROM ('2024-07-01') TO ('2025-01-01');

-- Секция 2: 1-е полугодие 2025 (январь–июнь 2025)
CREATE TABLE service_logs_2025_h1 PARTITION OF service_logs
FOR VALUES FROM ('2025-01-01') TO ('2025-07-01');

-- Секция по умолчанию (для остальных дат)
CREATE TABLE service_logs_default PARTITION OF service_logs DEFAULT
```

Рисунок 6 – Создание секций

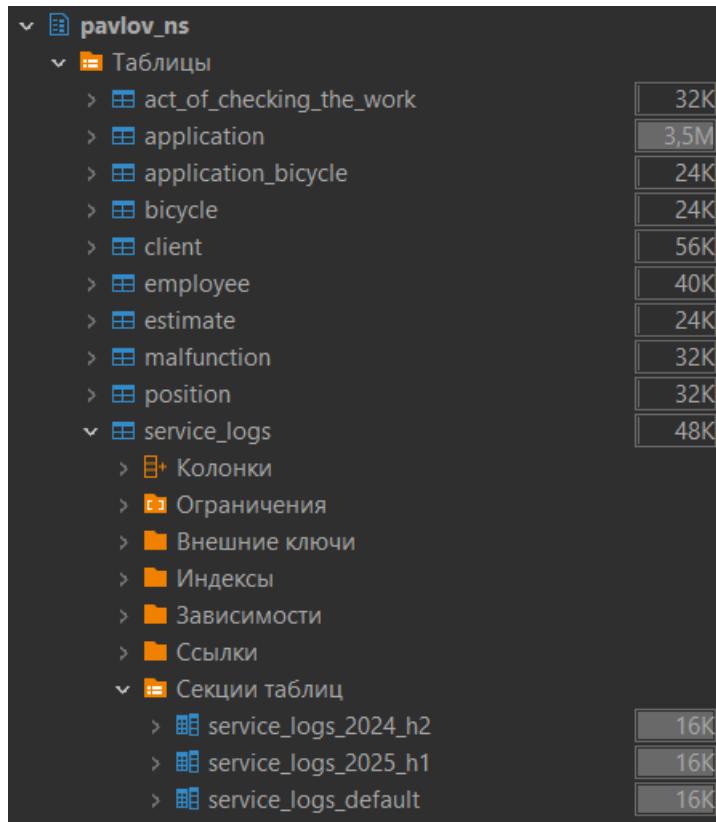


Рисунок 7 – Структура таблиц в навигаторе объектов (видна иерархия)

```

CREATE INDEX idx_service_logs_date ON service_logs(created_at)

```

Рисунок 8 – Создание индекса

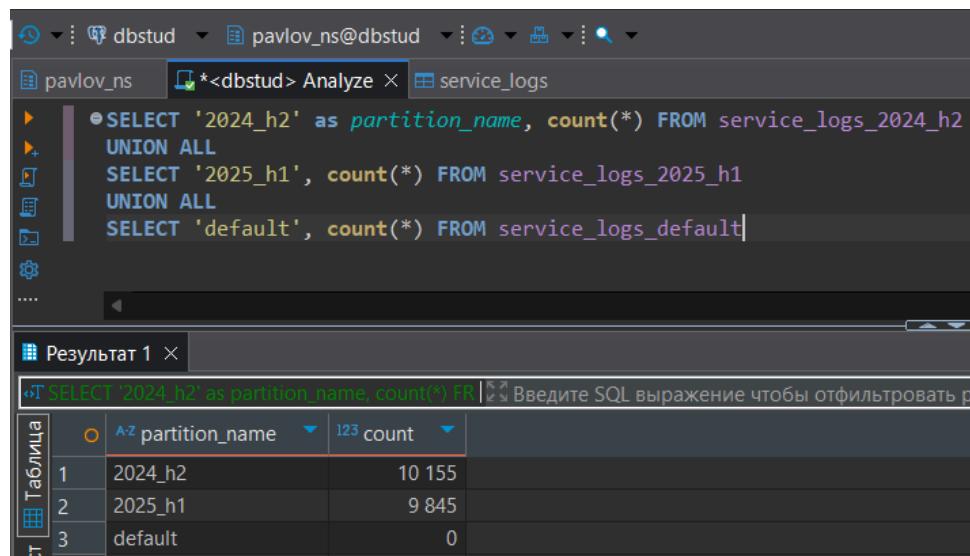
Название	Column	Таблица
application_bicycle_pkey	application_bicycle_id	application_bicycle
bicycle_pkey	bicycle_id	bicycle
client_email_key	email	client
client_phone_key	phone	client
client_pkey	client_id	client
employee_phone_key	phone	employee
employee_pkey	employee_id	employee
estimate_pkey	estimate_id	estimate
malfunction_pkey	malfunction_id	malfunction
position_pkey	position_id	position
idx_service_logs_date	created_at	service logs
service_logs_2024_h2_created_a	created_at	service logs 2024 h2
service_logs_2025_h1_created_a	created_at	service logs 2025 h1
service_logs_default_created_at	created_at	service logs default

Рисунок 9 – Проверка применения индекса ко всем секциям

2.4 ЗАДАНИЕ 3. ГЕНЕРАЦИЯ ДАННЫХ

```
DO $$  
DECLARE  
    i INT;  
    random_date TIMESTAMPTZ;  
    random_employee INT;  
    event_type TEXT;  
    json_payload JSONB;  
BEGIN  
    FOR i IN 1..20000 LOOP  
        random_date := '2024-07-01'::TIMESTAMPTZ + (random() * 365 * INTERVAL '1 day');  
        random_employee := floor(random() * 10 + 1)::INT;  
  
        IF (i % 2 = 0) THEN  
            event_type := 'payment';  
            json_payload := jsonb_build_object(  
                'event', event_type,  
                'details', jsonb_build_object(  
                    'order_id', i,  
                    'amount', floor(random() * 80000 + 500),  
                    'method', CASE floor(random() * 2)::INT  
                        WHEN 0 THEN 'Наличные'  
                        ELSE 'Карта'  
                    END  
                )  
        );  
        ELSE  
            event_type := 'login';  
            json_payload := jsonb_build_object(  
                'event', event_type,  
                'ip', '192.168.1.' || floor(random() * 255)::TEXT,  
                'success', (random() > 0.1)  
            );  
        END IF;  
  
        INSERT INTO service_logs (created_at, employee_id, event_type, event_data)  
        VALUES (random_date, random_employee, event_type, json_payload);  
    END LOOP;  
END $$
```

Рисунок 10 – Скрипт генерации данных



The screenshot shows the pgAdmin interface with a query editor window titled '<dbstud> Analyze' containing the following SQL code:

```
SELECT '2024_h2' as partition_name, count(*) FROM service_logs_2024_h2  
UNION ALL  
SELECT '2025_h1', count(*) FROM service_logs_2025_h1  
UNION ALL  
SELECT 'default', count(*) FROM service_logs_default
```

Below the query editor is a results table titled 'Результат 1' (Result 1) with the following data:

partition_name	count
2024_h2	10 155
2025_h1	9 845
default	0

Рисунок 11 – Проверка количества записей в секциях

2.5 ЗАДАНИЕ 4. АНАЛИЗ И ПОИСК ПО JSONB

```

SELECT log_id, event_data
FROM service_logs
WHERE event_type = 'payment'
AND (event_data -> 'details' -> 'amount')::NUMERIC > 50000
LIMIT 5
    
```

The screenshot shows the pgAdmin interface with a query editor and a results table. The query filters logs where the amount is greater than 50000. The results table contains 5 rows of log data, each with a log ID and a JSONB event data object.

log_id	event_data
20 004	{"event": "payment", "details": {"amount": 73501, "method": "Карта", "order_id": 4}}
20 006	{"event": "payment", "details": {"amount": 75225, "method": "Карта", "order_id": 6}}
20 020	{"event": "payment", "details": {"amount": 57555, "method": "Наличные", "order_id": 20}}
20 024	{"event": "payment", "details": {"amount": 53990, "method": "Карта", "order_id": 24}}
20 028	{"event": "payment", "details": {"amount": 73931, "method": "Наличные", "order_id": 28}}

Рисунок 12 – Фильтрация со сложным условием

```

SELECT log_id, event_data
FROM service_logs
WHERE event_data @> '{"details": {"method": "Карта"}}'
LIMIT 5
    
```

The screenshot shows the pgAdmin interface with a query editor and a results table. The query searches for logs where the method is 'Карта'. The results table contains 5 rows of log data.

log_id	event_data
20 002	{"event": "payment", "details": {"amount": 15938, "method": "Карта", "order_id": 2}}
20 004	{"event": "payment", "details": {"amount": 73501, "method": "Карта", "order_id": 4}}
20 006	{"event": "payment", "details": {"amount": 75225, "method": "Карта", "order_id": 6}}
20 018	{"event": "payment", "details": {"amount": 19045, "method": "Карта", "order_id": 18}}
20 024	{"event": "payment", "details": {"amount": 53990, "method": "Карта", "order_id": 24}}

Рисунок 13 – Поиск по вхождению фрагмента

```

SELECT
    SUM((event_data -> 'details' -> 'amount')::NUMERIC) as total_payments,
    AVG((event_data -> 'details' -> 'amount')::NUMERIC(10, 2)) as avg_payment
FROM service_logs
WHERE event_type = 'payment'
    
```

The screenshot shows the pgAdmin interface with a query editor and a results table. The query aggregates payment data, summing the total amount and calculating the average amount. The results table contains 1 row with the aggregated values.

total_payments	avg_payment
404 492 509	40 449,25

Рисунок 14 – Агрегация данных

2.6 ЗАДАНИЕ 5. МОДИФИКАЦИЯ ДАННЫХ JSONB

The screenshot shows the pgAdmin interface with a connection to dbstud and user pavlov_ns. In the main pane, there are two queries:

```
• UPDATE service_logs
  SET event_data = event_data || '{"source": "web"}'::jsonb;
• SELECT event_data FROM service_logs LIMIT 5
```

Below the queries, a results grid titled "service_logs 1" displays five rows of JSON data:

Таблица	event_data
1	{"event": "payment", "source": "web", "details": {"amount": 15938, "method": "Карта", "order_id": 2}}
2	{"ip": "192.168.1.124", "event": "login", "source": "web", "success": true}
3	{"event": "payment", "source": "web", "details": {"amount": 73501, "method": "Карта", "order_id": 4}}
4	{"event": "payment", "source": "web", "details": {"amount": 75225, "method": "Карта", "order_id": 6}}
5	{"ip": "192.168.1.34", "event": "login", "source": "web", "success": true}

Рисунок 15 – Массовое обновление

The screenshot shows the pgAdmin interface with a connection to dbstud and user pavlov_ns. In the main pane, there are two queries:

```
• UPDATE service_logs
  SET event_data = jsonb_set(event_data, '{success}', 'false')
  WHERE log_id = (SELECT min(log_id) FROM service_logs WHERE event_type = 'login');
• SELECT event_data FROM service_logs WHERE event_data ->> 'success' = 'false'
  LIMIT 1
```

Below the queries, a results grid titled "service_logs 1" displays one row of JSON data:

Таблица	event_data
1	{"ip": "192.168.1.122", "event": "login", "source": "web", "success": false}

Рисунок 16 – Использование jsonb_set