



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 *ИНБО-12-23. Албахтин И.В.*

(подпись)

Ассистент *Брайловский А.В.*

(подпись)

Москва 2025 г.

ПРАКТИЧЕСКАЯ РАБОТА №7. СЛОЖНЫЕ ТИПЫ ДАННЫХ И МАСШТАБИРОВАНИЕ POSTGRESQL

Цель работы:

Целью данной практической работы является 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.

По завершении работы студент должен уметь:

- Проектировать таблицы для хранения чувствительных данных (например, паролей) с использованием бинарного типа данных BYTEA и криптографических функций.
- Реализовывать хранение полуструктурных данных (логов, событий) в формате JSONB.
- Выполнять выборку, фильтрацию и агрегацию данных из JSON-документов с использованием специализированных операторов (->>, @>, ?) и условий сравнения.
- Модифицировать структуру JSON-документов «на лету» без изменения схемы таблицы.
- Настраивать декларативное секционирование (Partitioning) таблиц по диапазонам для оптимизации производительности при работе с большими объемами данных.

Постановка задачи:

Для выполнения практической работы необходимо последовательно выполнить следующие шаги, адаптируя примеры из БД «Аптека» к вашей собственной базе данных.

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-структуру 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.

ВЫПОЛНЕНИЕ ПРАКТИЧЕСКОЙ РАБОТЫ

Таблица 1. Данные таблицы system_roles

The screenshot shows a database interface with a toolbar on the left containing icons for file operations. A query editor window is open with the following SQL code:

```
SELECT column_name, data_type
FROM information_schema.columns
WHERE table_name = 'system_roles';
```

Below the query editor is a results grid titled "columns 1". The grid displays two columns: "column_name" and "data_type". The data is as follows:

	column_name	data_type
1	role_id	integer
2	role_name	character varying

Таблица 2. Структура таблицы system_roles

The screenshot shows a database interface with a toolbar on the left containing icons for file operations. A query editor window is open with the following SQL code:

```
SELECT * FROM system_roles;
```

Below the query editor is a results grid titled "system_roles 1". The grid displays two columns: "role_id" and "role_name". The data is as follows:

	role_id	role_name
1	1	Администратор
2	2	Мастер
3	3	Менеджер

Таблица 3. Структура таблицы system_users

The screenshot shows the MySQL Workbench interface with the following details:

- SQL Editor:** Contains the SQL query: `SELECT column_name, data_type FROM information_schema.columns WHERE table_name = 'system_users';`
- Result Set:** A table titled "columns 1" showing the columns of the system_users table.
- Columns:**

	column_name	data_type
1	user_id	integer
2	password_hash	bytea
3	role_id	integer
4	username	character varying
5	password_raw	text

Таблица 4. Данные таблицы system_users

The screenshot shows the MySQL Workbench interface with the following details:

- SQL Editor:** Contains the SQL query: `SELECT * FROM system_users;`
- Result Set:** A table titled "system_users 1" showing the data rows.
- Columns:**

	user_id	username	password_raw	password_hash	role_id
1	2	worker	Pharma2025	6PGÃ¥ yÜ Úlý :é i Á í D8eöñý [32]	2
2	4	test_user	correctPass	½ Ü 0 lo ÁúDe IGÖÄö2üg xæë [32]	1
3	1	admin_user	VeryLongAndStrongPassword123	[ÜQw&'óÑ Mñr ³Ö GÅ×?é÷¾-/á Ø	1

Таблица 5. Структура таблицы change_log

The screenshot shows a database interface with a query editor and a results grid. The query in the editor is:

```
SELECT column_name, data_type  
FROM information_schema.columns  
WHERE table_name = 'event_log';
```

The results grid is titled "columns 1" and displays the following data:

Таблица	A-Z column_name	A-Z data_type
1	id	integer
2	event_type	character varying
3	event_date	date
4	details	jsonb

Таблица 6. Секции таблицы event_log

The screenshot shows a database interface with a query editor and a results grid. The query in the editor is:

```
SELECT table_name  
FROM information_schema.tables  
WHERE table_name LIKE 'event_log%';
```

The results grid is titled "tables 1" and displays the following data:

Таблица	A-Z table_name
1	event_log
2	event_log_2024_h2
3	event_log_2025_h1
4	event_log_future

Таблица 7. Данные секции event_log_2024_h2

```
SELECT '2024_H2' AS part, * FROM event_log_2024_h2 LIMIT 10;
SELECT '2025_H1' AS part, * FROM event_log_2025_h1 LIMIT 10;
SELECT 'FUTURE' AS part, * FROM event_log_future LIMIT 10;
```

event_log_2024_h2 1 | event_log_2025_h1 1 (2) | event_log_future 1 (3)

Фильтрация данных не поддерживается

Таблица	AZ part	123 id	AZ event_type	event_date	details
1	2024_H2	4	error	2024-11-30	{"code": 404, "message": "Auto-generated test error"}
2	2024_H2	7	login	2024-08-17	{"ip": "192.168.213.10", "device": "tablet"}
3	2024_H2	8	login	2024-10-13	{"ip": "192.168.93.10", "device": "mobile"}
4	2024_H2	9	login	2024-07-31	{"ip": "192.168.54.10", "device": "tablet"}
5	2024_H2	19	login	2024-10-15	{"ip": "192.168.213.10", "device": "desktop"}
6	2024_H2	25	error	2024-10-14	{"code": 500, "message": "Auto-generated test error"}
7	2024_H2	39	error	2024-09-02	{"code": 500, "message": "Auto-generated test error"}
8	2024_H2	51	error	2024-07-01	{"code": 401, "message": "Auto-generated test error"}
9	2024_H2	53	error	2024-10-21	{"code": 404, "message": "Auto-generated test error"}
10	2024_H2	54	error	2024-08-15	{"code": 403, "message": "Auto-generated test error"}

Таблица 8. Данные секции event_log_2025_h1

```
SELECT '2025_H1' AS part, * FROM event_log_2025_h1 LIMIT 10;
SELECT 'FUTURE' AS part, * FROM event_log_future LIMIT 10;
```

event_log_2024_h2 1 | event_log_2025_h1 1 (2) | event_log_future 1 (3)

Фильтрация данных не поддерживается

Таблица	AZ part	123 id	AZ event_type	event_date	details
1	2025_H1	2	login	2025-02-21	{"ip": "192.168.99.10", "device": "mobile"}
2	2025_H1	10	login	2025-01-18	{"ip": "192.168.153.10", "device": "tablet"}
3	2025_H1	14	error	2025-05-27	{"code": 404, "message": "Auto-generated"}
4	2025_H1	24	error	2025-06-08	{"code": 500, "message": "Auto-generated"}
5	2025_H1	29	error	2025-01-06	{"code": 404, "message": "Auto-generated"}
6	2025_H1	33	login	2025-05-20	{"ip": "192.168.95.10", "device": "desktop"}
7	2025_H1	34	error	2025-05-29	{"code": 400, "message": "Auto-generated"}
8	2025_H1	36	login	2025-02-13	{"ip": "192.168.247.10", "device": "mobile"}
9	2025_H1	37	error	2025-05-21	{"code": 400, "message": "Auto-generated"}
10	2025_H1	38	login	2025-03-26	{"ip": "192.168.28.10", "device": "desktop"}

Таблица 9. Данные секции event_log_future

```
SELECT 'FUTURE' AS part, * FROM event_log_future LIMIT 10;
```

event_log_2024_h2 1 | event_log_2025_h1 1 (2) | event_log_future 1 (3)

Фильтрация данных не поддерживается

Таблица	AZ part	123 id	AZ event_type	event_date	details
1	FUTURE	3	error	2025-07-03	{"code": 403, "message": "Auto-generated test error"}
2	FUTURE	12	login	2025-09-24	{"ip": "192.168.173.10", "device": "desktop"}
3	FUTURE	16	error	2025-07-26	{"code": 404, "message": "Auto-generated test error"}
4	FUTURE	17	login	2025-07-21	{"ip": "192.168.130.10", "device": "tablet"}
5	FUTURE	18	login	2025-08-22	{"ip": "192.168.222.10", "device": "desktop"}
6	FUTURE	23	error	2025-07-12	{"code": 400, "message": "Auto-generated test error"}
7	FUTURE	26	error	2025-08-13	{"code": 401, "message": "Auto-generated test error"}
8	FUTURE	42	error	2025-07-13	{"code": 401, "message": "Auto-generated test error"}
9	FUTURE	45	error	2025-09-22	{"code": 400, "message": "Auto-generated test error"}
10	FUTURE	50	error	2025-09-17	{"code": 400, "message": "Auto-generated test error"}

Таблица 10. Структура таблицы *change_log*

The screenshot shows the MySQL Workbench interface with the following details:

- Left Panel:** Shows icons for Import, Export, and Script.
- Central Area:** A SQL editor window containing the following query:


```
④ SELECT column_name, data_type
FROM information_schema.columns
WHERE table_name = 'change_log';
```
- Bottom Area:** A results grid titled "columns 1" showing the columns of the change_log table. The columns are listed in rows 1 through 6, with "log_id" being the primary key.

Номер	column_name	data_type
1	log_id	integer
2	table_name	text
3	operation	text
4	old_row	jsonb
5	new_row	jsonb
6	changed_at	timestamp without time zone

Таблица 11. Данные таблицы *change_log*

The screenshot shows the MySQL Workbench interface with the following details:

- Left Panel:** Shows icons for Import, Export, and Script.
- Central Area:** A results grid titled "change_log 1" showing the data in the change_log table. The table has 3 rows.
- Bottom Area:** A detailed view of the first row (log_id: 1) with expanded columns for old_row and new_row.

Номер	log_id	table_name	operation	old_row	new_row	changed_at
1	1	system_users	UPDATE	[{"role_id": 1, "user_id": 1, "username": "admin_user", "password_raw": "StrongPass123!", "password": ("role_id": 1, "user_id": 1, "username": "admin_user", "password_raw": "NewPass999", "password": "2025-12-11 15:00:38.052")}]	[{"role_id": 1, "user_id": 4, "username": "test_user", "password_raw": "correctPass", "password": "2025-12-11 15:05:39.770"}]	"role_id": 1, "user_id": 1, "username": "admin_user", "password_raw": "VeryLongAndStrongPasswo 2025-12-11 15:08:16.303"
2	2	system_users	INSERT	[NULL]		
3	3	system_users	UPDATE	[{"role_id": 1, "user_id": 1, "username": "admin_user", "password_raw": "NewPass999", "password": ("role_id": 1, "user_id": 1, "username": "admin_user", "password_raw": "VeryLongAndStrongPasswo 2025-12-11 15:08:16.303")}]		

Задание №1: хранение паролей (BYTEA)

The screenshot shows the pgAdmin interface with a query editor and a statistics window. The query editor contains the SQL code for creating the `system_roles` table:

```
CREATE TABLE system_roles (
    role_id SERIAL PRIMARY KEY,
    role_name VARCHAR(50) NOT NULL UNIQUE
);
```

The statistics window titled "Статистика 1" displays the following information:

Name	Value
Updated Rows	0
Execute time	0,019s
Start time	Thu Dec 11 14:48:17 MSK 2025
Finish time	Thu Dec 11 14:48:17 MSK 2025
Query	CREATE TABLE system_roles (role_id SERIAL PRIMARY KEY, role_name VARCHAR(50) NOT NULL UNIQUE)

Рисунок 1 – Создание таблицы ролей

The screenshot shows the pgAdmin interface with a query editor and a statistics window. The query editor contains the SQL code for creating the `system_users` table:

```
CREATE TABLE system_users (
    user_id SERIAL PRIMARY KEY,
    username VARCHAR(100) NOT NULL UNIQUE,
    password_raw TEXT,
    password_hash BYTEA NOT NULL,
    role_id INT REFERENCES system_roles(role_id)
);
```

The statistics window titled "Статистика 1" displays the following information:

Name	Value
Updated Rows	0
Execute time	0,016s
Start time	Thu Dec 11 14:49:14 MSK 2025
Finish time	Thu Dec 11 14:49:14 MSK 2025
Query	CREATE TABLE system_users (user_id SERIAL PRIMARY KEY, username VARCHAR(100) NOT NULL UNIQUE,

Рисунок 2 – Создание таблицы пользователей

```

INSERT INTO system_roles (role_name)
VALUES ('Администратор'),
       ('Мастер'),
       ('Менеджер');

```

Статистика 1

Name	Value
Updated Rows	3
Execute time	0,012s
Start time	Thu Dec 11 14:50:37 MSK 2025
Finish time	Thu Dec 11 14:50:37 MSK 2025
Query	INSERT INTO system_roles (role_name) VALUES ('Администратор'), ('Мастер'), ('Менеджер')

Рисунок 3 – Заполнение ролей

```

INSERT INTO system_users (username, password_raw, password_hash, role_id)
VALUES
    ('admin_user', 'StrongPass123!', digest('StrongPass123!', 'sha256'), 1),
    ('worker',      'Pharma2025',   digest('Pharma2025_BAD', 'sha256'), 2);

```

Статистика 1

Name	Value
Updated Rows	2
Execute time	0,013s
Start time	Thu Dec 11 14:51:11 MSK 2025
Finish time	Thu Dec 11 14:51:11 MSK 2025
Query	INSERT INTO system_users (username, password_raw, password_hash, role_id) VALUES ('admin_user', 'StrongPass123!', digest('StrongPass123!', 'sha256'), 1), ('worker', 'Pharma2025', digest('Pharma2025_BAD', 'sha256'), 2)

Рисунок 4 – Вставка пользователей с SHA-256

```

SELECT
    username,
    password_raw,
    encode(password_hash, 'hex') AS password_hash_hex,
    digest(password_raw, 'sha256') = password_hash AS is_valid
FROM system_users;

```

system_users 1

Таблица	username	password_raw	password_hash_hex	is_valid
1	admin_user	StrongPass123!	805bd951772627f3d1a607084df1727c6caad60447c5d73febfb7be2d2fe17fd8	[v]
2	worker	Pharma2025	365047c3a515fdd97cdacfff821e3ae87bec1816c11ea6ee8b0d443865f5effd	[]

Рисунок 5 – Проверка корректности хешей

Задание №2. секционирование таблицы логов

```
CREATE TABLE event_log (
    id SERIAL,
    event_type VARCHAR(50) NOT NULL,
    event_date DATE NOT NULL,
    details JSONB NOT NULL
) PARTITION BY RANGE (event_date);
```

Name	Value
Updated Rows	0
Execute time	0,073s
Start time	Thu Dec 11 15:30:08 MSK 2025
Finish time	Thu Dec 11 15:30:08 MSK 2025
Query	CREATE TABLE event_log (id SERIAL, event_type VARCHAR(50) NOT NULL, event_date DATE NOT NULL, details JSONB NOT NULL) PARTITION BY RANGE (event_date)

Рисунок 6 – Создание родительской таблицы (без PK)

```
CREATE TABLE event_log_2024_h2
PARTITION OF event_log
FOR VALUES FROM ('2024-07-01') TO ('2025-01-01');
```

Name	Value
Updated Rows	0
Execute time	0,022s
Start time	Thu Dec 11 15:30:41 MSK 2025
Finish time	Thu Dec 11 15:30:41 MSK 2025
Query	CREATE TABLE event_log_2024_h2 PARTITION OF event_log FOR VALUES FROM ('2024-07-01') TO ('2025-01-01')

Рисунок 7 - Создание первой секции

```
CREATE TABLE event_log_2025_h1
PARTITION OF event_log
FOR VALUES FROM ('2025-01-01') TO ('2025-07-01');
```

Статистика 1

Name	Value
Updated Rows	0
Execute time	0,019s
Start time	Thu Dec 11 15:31:03 MSK 2025
Finish time	Thu Dec 11 15:31:03 MSK 2025
Query	CREATE TABLE event_log_2025_h1 PARTITION OF event_log FOR VALUES FROM ('2025-01-01') TO ('2025-07-01')

Рисунок 8 – Создание второй секции

```
CREATE TABLE event_log_future
PARTITION OF event_log
FOR VALUES FROM ('2025-07-01') TO (MAXVALUE);
```

Статистика 1

Name	Value
Updated Rows	0
Execute time	0,017s
Start time	Thu Dec 11 15:31:36 MSK 2025
Finish time	Thu Dec 11 15:31:36 MSK 2025
Query	CREATE TABLE event_log_future PARTITION OF event_log FOR VALUES FROM ('2025-07-01') TO (MAXVALUE)

Рисунок 9 – Создание секции по умолчанию

The screenshot shows a database interface with a query editor and a results grid. The query editor at the top contains the following SQL code:

```
SELECT table_name  
FROM information_schema.tables  
WHERE table_name LIKE 'event_log%';
```

The results grid below is titled "tables 1" and displays the following data:

	table_name
1	event_log
2	event_log_2024_h2
3	event_log_2025_h1
4	event_log_future

Рисунок 10 – Проверка

Задание №3: генерация данных

The screenshot shows the Oracle SQL Developer interface. The main window displays a PL/SQL script for generating event log data. The script uses a DO loop to iterate 20,000 times. Inside the loop, it selects a random event type from an array ('login', 'sale', 'error'), calculates a random date between 2024-07-01 and 2025-07-01, and generates a JSONB object for each event type. The JSONB object contains details like IP address, device type, quantity, price, or error code and message. Finally, it inserts the generated data into the event_log table.

```
DO $$  
DECLARE  
    i INT;  
    v_event_type TEXT;  
    v_date DATE;  
    v_details JSONB;  
BEGIN  
    FOR i IN 1..20000 LOOP  
  
        -- выбираем случайный тип события  
        v_event_type := (ARRAY['login', 'sale', 'error'])[ceil(random() * 3)];  
  
        -- назначаем дату под существующие секции + будущие даты  
        IF random() < 0.33 THEN  
            v_date := date '2024-07-01' + (random() * 180)::INT; -- 2-е полугодие 2024  
        ELSIF random() < 0.66 THEN  
            v_date := date '2025-01-01' + (random() * 180)::INT; -- 1-е полугодие 2025  
        ELSE  
            v_date := date '2025-07-01' + (random() * 100)::INT; -- future (будущее)  
        END IF;  
  
        -- генерируем JSON под тип события  
        IF v_event_type = 'login' THEN  
            v_details := jsonb_build_object(  
                'ip', '192.168.' || (1 + floor(random()*255))::INT || '.10',  
                'device', (ARRAY['mobile','desktop','tablet'])[ceil(random()*3)]  
            );  
  
        ELSIF v_event_type = 'sale' THEN  
            v_details := jsonb_build_object(  
                'quantity', (1 + floor(random()*50))::INT,  
                'price', (100 + floor(random()*900))::INT  
            );  
  
        ELSE -- error  
            v_details := jsonb_build_object(  
                'code', (ARRAY[400,401,403,404,500])[ceil(random()*5)],  
                'message', 'Auto-generated test error'  
            );  
        END IF;  
  
        -- вставляем строку  
        INSERT INTO event_log(event_type, event_date, details)  
        VALUES (v_event_type, v_date, v_details);  
    END LOOP;  
END $$;
```

Below the code editor, a statistics window titled 'Статистика 1' is open, showing the following data:

Name	Value
Updated Rows	0
Execute time	0,222s
Start time	Thu Dec 11 15:33:21 MSK 2025
Finish time	Thu Dec 11 15:33:21 MSK 2025
Query	DO \$\$ DECLARE i INT; v_event_type TEXT; v_date DATE; v_details JSONB;

Рисунок 11 – Создадим скрипт генерации 20 000 записей

SELECT COUNT(*) FROM event_log;

Результат 1

count
20 000

Рисунок 12 – Проверка отработки скрипта

Рисунок 13 - Проверка распределения по секциям

Задание №4: анализ и поиск по JSONB

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

Query Editor:

```
SELECT *  
FROM event_log  
WHERE event_type = 'sale'  
AND (details->>'price')::INT > 800;
```

Results Table:

	123 id	event_type	event_date	details
1	48	sale	2024-10-29	{"price": 927, "quantity": 27}
2	247	sale	2024-12-26	{"price": 961, "quantity": 1}
3	325	sale	2024-09-28	{"price": 890, "quantity": 36}
4	364	sale	2024-08-31	{"price": 839, "quantity": 4}
5	370	sale	2024-09-19	{"price": 867, "quantity": 23}
6	380	sale	2024-12-16	{"price": 926, "quantity": 30}
7	568	sale	2024-08-05	{"price": 822, "quantity": 50}
8	621	sale	2024-09-12	{"price": 973, "quantity": 25}
9	654	sale	2024-10-02	{"price": 883, "quantity": 4}
10	673	sale	2024-12-22	{"price": 842, "quantity": 2}
11	809	sale	2024-08-22	{"price": 996, "quantity": 40}
12	827	sale	2024-09-23	{"price": 996, "quantity": 36}
13	859	sale	2024-12-26	{"price": 824, "quantity": 11}
14	953	sale	2024-12-15	{"price": 938, "quantity": 14}
15	975	sale	2024-12-20	{"price": 945, "quantity": 47}
16	1 052	sale	2024-12-24	{"price": 929, "quantity": 34}
17	1 108	sale	2024-08-20	{"price": 991, "quantity": 29}
18	1 145	sale	2024-12-06	{"price": 895, "quantity": 33}
19	1 152	sale	2024-10-14	{"price": 846, "quantity": 40}
20	1 199	sale	2024-08-03	{"price": 901, "quantity": 18}
21	1 221	sale	2024-12-08	{"price": 822, "quantity": 3}
22	1 246	sale	2024-10-25	{"price": 827, "quantity": 25}
23	1 297	sale	2024-08-24	{"price": 868, "quantity": 23}
24	1 410	sale	2024-11-12	{"price": 823, "quantity": 22}
25	1 430	sale	2024-08-24	{"price": 903, "quantity": 21}
26	1 436	sale	2024-07-25	{"price": 808, "quantity": 32}

Рисунок 14 – Найти все продажи, где цена > 800

```

④ SELECT *
FROM event_log
WHERE event_type = 'sale'
AND details @> '{"quantity": 5}';

event_log 1 X
SELECT * FROM event_log WHERE event_type = 'sale' AND details @> '{"quantity": 5}' Введите SQL выражение ч

```

Таблица	123 id	AZ event_type	⌚ event_date	☒ details
1	205	sale	2024-10-19	{"price": 294, "quantity": 5}
2	634	sale	2024-12-27	{"price": 596, "quantity": 5}
3	913	sale	2024-07-09	{"price": 565, "quantity": 5}
4	1 379	sale	2024-11-24	{"price": 674, "quantity": 5}
5	1 894	sale	2024-11-22	{"price": 648, "quantity": 5}
6	2 201	sale	2024-08-01	{"price": 525, "quantity": 5}
7	2 281	sale	2024-08-18	{"price": 641, "quantity": 5}
8	2 807	sale	2024-09-10	{"price": 611, "quantity": 5}
9	3 288	sale	2024-10-01	{"price": 965, "quantity": 5}
10	3 439	sale	2024-10-06	{"price": 469, "quantity": 5}
11	3 811	sale	2024-07-21	{"price": 936, "quantity": 5}
12	4 470	sale	2024-08-23	{"price": 216, "quantity": 5}
13	4 601	sale	2024-09-05	{"price": 114, "quantity": 5}
14	4 706	sale	2024-12-19	{"price": 355, "quantity": 5}
15	5 051	sale	2024-10-10	{"price": 586, "quantity": 5}
16	5 187	sale	2024-08-16	{"price": 295, "quantity": 5}
17	5 462	sale	2024-08-06	{"price": 828, "quantity": 5}
18	5 540	sale	2024-11-25	{"price": 302, "quantity": 5}
19	5 552	sale	2024-09-06	{"price": 232, "quantity": 5}
20	6 236	sale	2024-11-12	{"price": 731, "quantity": 5}
21	7 018	sale	2024-10-12	{"price": 797, "quantity": 5}
22	7 025	sale	2024-12-21	{"price": 685, "quantity": 5}
23	8 284	sale	2024-07-09	{"price": 933, "quantity": 5}
24	8 734	sale	2024-12-01	{"price": 315, "quantity": 5}
25	8 773	sale	2024-09-04	{"price": 758, "quantity": 5}
26	9 009	sale	2024-10-19	{"price": 188, "quantity": 5}

Рисунок 15 – Найти продажи, где количество = 5

The screenshot shows a PostgreSQL pgAdmin interface. In the top-left corner, there are several icons: a blue square with a white arrow, a blue square with a white document, a red square with a white exclamation mark, and a blue square with a white question mark. Below these is a small tree icon.

In the main area, a query is displayed in a code editor:

```
SELECT SUM( (details->>'quantity')::INT ) AS total_quantity
FROM event_log
WHERE event_type = 'sale';
```

Below the code editor is a results pane titled "Результат 1" (Result 1). The pane contains a table with one row. The table has two columns: "total_quantity" and a column with the value "172 162".

Таблица	123 total_quantity
1	172 162

Рисунок 16 – Общая сумма quantity по всем ТО

Задание №5: модификация данных JSONB

The screenshot shows a database interface with two main sections. The top section displays an SQL query:

```
UPDATE event_log
SET details = jsonb_set(details, '{version}', '"1.0"', true)
WHERE event_type = 'sale';
```

The bottom section shows the execution statistics for this query:

Name	Value
Updated Rows	6746
Execute time	0,058s
Start time	Thu Dec 11 15:38:00 MSK 2025
Finish time	Thu Dec 11 15:38:00 MSK 2025
Query	UPDATE event_log SET details = jsonb_set(details, '{version}', '"1.0"', true) WHERE event_type = 'sale'

Рисунок 17 – Массовое добавление ключа

The screenshot shows a database interface with two main sections. The top section displays a SQL query:

```
SELECT *
FROM event_log
WHERE event_type = 'sale'
LIMIT 10;
```

The bottom section shows the results of this query in a table:

id	event_type	event_date	details
1	sale	2024-07-17	{"price": 637, "version": "1.0", "quantity": 26}
2	sale	2024-10-24	{"price": 615, "version": "1.0", "quantity": 47}
3	sale	2024-10-29	{"price": 927, "version": "1.0", "quantity": 27}
4	sale	2024-07-03	{"price": 201, "version": "1.0", "quantity": 29}
5	sale	2024-10-09	{"price": 332, "version": "1.0", "quantity": 26}
6	sale	2024-10-30	{"price": 231, "version": "1.0", "quantity": 50}
7	sale	2024-08-09	{"price": 120, "version": "1.0", "quantity": 39}
8	sale	2024-08-11	{"price": 411, "version": "1.0", "quantity": 8}
9	sale	2024-10-12	{"price": 746, "version": "1.0", "quantity": 28}
10	sale	2024-12-09	{"price": 389, "version": "1.0", "quantity": 9}

Рисунок 18 – Проверка результата

The screenshot shows a database interface with a query editor and a statistics window. The query in the editor is:

```
UPDATE event_log
SET details = jsonb_set(details, '{quantity}', '999', false)
WHERE id = 87;
```

The statistics window titled "Статистика 1" displays the following information:

Name	Value
Updated Rows	1
Execute time	0,013s
Start time	Thu Dec 11 15:39:23 MSK 2025
Finish time	Thu Dec 11 15:39:23 MSK 2025
Query	UPDATE event_log SET details = jsonb_set(details, '{quantity}', '999', false) WHERE id = 87

Рисунок 19 - Изменим quantity у записи с id 87

The screenshot shows a database interface with a query editor and a results table. The query in the editor is:

```
SELECT id, event_type, details
FROM event_log
WHERE id = 87;
```

The results table titled "event_log 1" shows one row:

id	event_type	details
87	sale	{"price": 389, "version": "1.0", "quantity": 999}

Рисунок 20 - Проверка

ВЫВОД

В ходе работы были освоены методы хранения чувствительных данных с использованием BYTEA и хеширования паролей через digest. Создана секционированная таблица логов, позволяющая эффективно распределять большие объёмы данных по диапазонам дат.

Сгенерированные JSONB-события продемонстрировали возможности хранения и обработки полуструктурированных данных: фильтрацию, поиск по фрагменту и агрегацию.

Также выполнена модификация JSONB — массовое добавление поля и точечное изменение значения. Работа позволила закрепить навыки использования JSONB и секционирования для повышения гибкости и производительности PostgreSQL.