

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего
образования

Санкт-Петербургский национальный исследовательский университет ИТМО

Мегафакультет трансляционных информационных технологий

Факультет информационных технологий и программирования

Лабораторная работа 5. Партицирование в PostgreSQL

По дисциплине «Проектирование баз данных»

Выполнил:

студент группы №М3212

Тимофеев Вячеслав

Проверила:

Чеботарева



УНИВЕРСИТЕТ ИТМО

Санкт-Петербург
2025

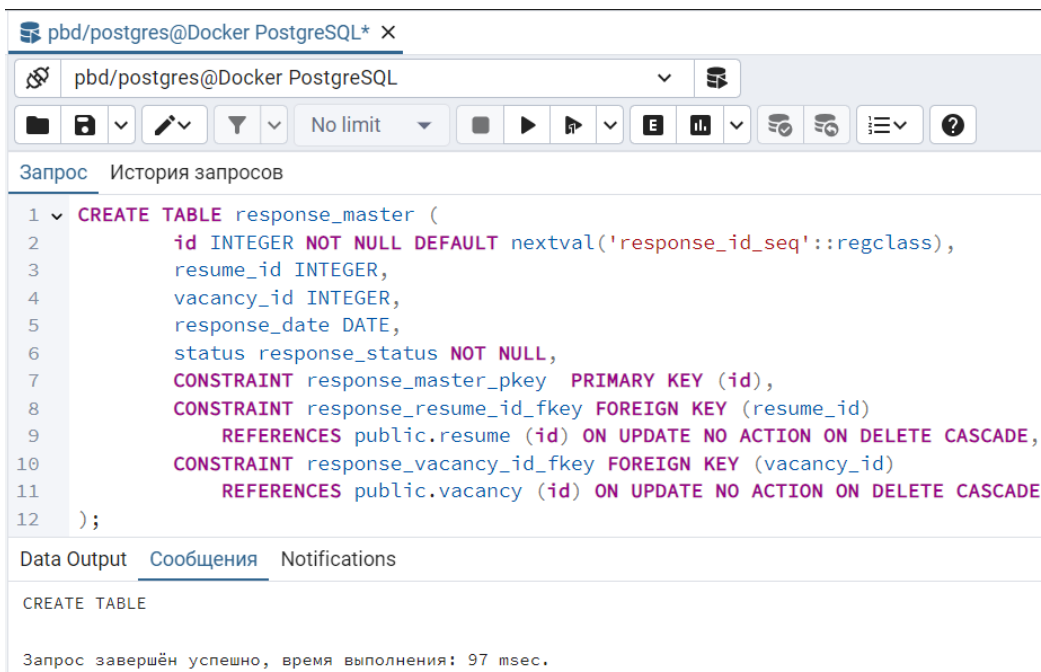
Задача:

1. Сделать партиционирование одной таблицы через наследование.
 - 1.2 Таблица должна быть разбита на не менее, чем 3 партии.
 - 1.3 В каждой партии должно быть не менее 5 записей.

Ход работы:

1. Будем разделять таблицу **response** по временному интервалу в 1 год. На практике партиционирование данных по датам оптимизирует запросы к базе (если разные партии хранить не на одном сервере), => легче масштабировать, очищать бд от мертвых строк (проводить VACUUM).

Создадим мастер-таблицу **response_master** (используя информацию из скрипта CREATE таблицы response):



The screenshot shows a PostgreSQL client window titled "pbd/postgres@Docker PostgreSQL* X". The address bar shows "pbd/postgres@Docker PostgreSQL". Below the address bar is a toolbar with various icons. The main area is titled "Запрос История запросов" and contains a SQL query to create a table named "response_master". The query is as follows:

```
1 CREATE TABLE response_master (  
2     id INTEGER NOT NULL DEFAULT nextval('response_id_seq'::regclass),  
3     resume_id INTEGER,  
4     vacancy_id INTEGER,  
5     response_date DATE,  
6     status response_status NOT NULL,  
7     CONSTRAINT response_master_pkey PRIMARY KEY (id),  
8     CONSTRAINT response_resume_id_fkey FOREIGN KEY (resume_id)  
9         REFERENCES public.resume (id) ON UPDATE NO ACTION ON DELETE CASCADE,  
10    CONSTRAINT response_vacancy_id_fkey FOREIGN KEY (vacancy_id)  
11        REFERENCES public.vacancy (id) ON UPDATE NO ACTION ON DELETE CASCADE  
12 );
```

Below the query, there are tabs for "Data Output", "Сообщения", and "Notifications". The "Сообщения" tab is selected, showing the message "CREATE TABLE". At the bottom, a status bar indicates "Запрос завершён успешно, время выполнения: 97 msec."

2. Создадим три партиционные таблицы, разбивающие отклики на 2023,2024,2025 года (**response_2023**, **response_2024**, **response_2025**):

kbd/postgres@Docker PostgreSQL ×

kbd/postgres@Docker PostgreSQL

Запрос История запросов

```
1 CREATE TABLE response_2023 (  
2     CHECK (response_date >= DATE '2023-01-01' AND response_date < DATE '2024-01-01')  
3 ) INHERITS (response_master);  
4  
5 CREATE TABLE response_2024 (  
6     CHECK (response_date >= DATE '2024-01-01' AND response_date < DATE '2025-01-01')  
7 ) INHERITS (response_master);  
8  
9 CREATE TABLE response_2025 (  
10    CHECK (response_date >= DATE '2025-01-01' AND response_date < DATE '2026-01-01')  
11 ) INHERITS (response_master);
```

Data Output Сообщения Notifications

CREATE TABLE

Запрос завершён успешно, время выполнения: 162 мсек.

3. Создадим триггер-функцию, обеспечивающую разделение откликов по годам при модификации основной таблицы (вставка):

kbd/postgres@Docker PostgreSQL ×

kbd/postgres@Docker PostgreSQL

Запрос История запросов

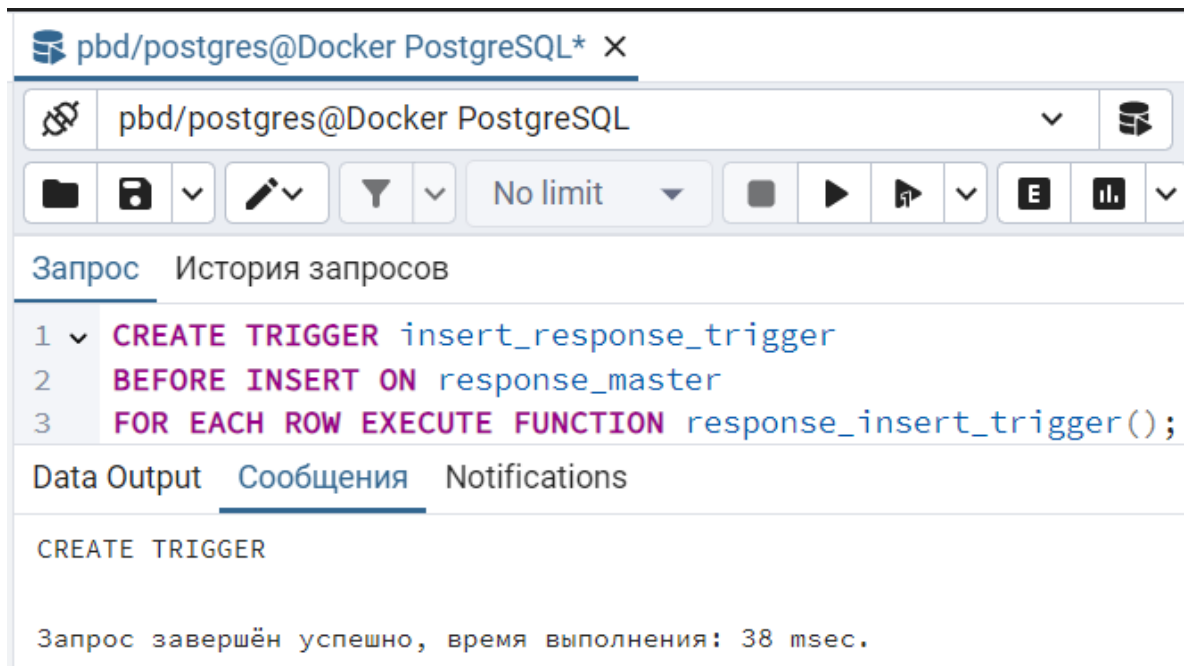
```
1 CREATE OR REPLACE FUNCTION response_insert_trigger()  
2 RETURNS TRIGGER AS $$  
3 BEGIN  
4     IF (NEW.response_date >= DATE '2023-01-01' AND NEW.response_date < DATE '2024-01-01') THEN  
5         INSERT INTO response_2023 VALUES (NEW.*);  
6  
7     ELSEIF (NEW.response_date >= DATE '2024-01-01' AND NEW.response_date < DATE '2025-01-01') THEN  
8         INSERT INTO response_2024 VALUES (NEW.*);  
9  
10    ELSEIF (NEW.response_date >= DATE '2025-01-01' AND NEW.response_date < DATE '2026-01-01') THEN  
11        INSERT INTO response_2025 VALUES (NEW.*);  
12  
13    ELSE  
14        RAISE EXCEPTION 'Неверный диапазон даты:%', NEW.response_date;  
15  
16    END IF;  
17    RETURN NULL;  
18 END;  
19 $$ LANGUAGE plpgsql;
```

Data Output Сообщения Notifications

CREATE FUNCTION

Запрос завершён успешно, время выполнения: 41 мсек.

4. Привязываем созданный триггер к мастер-таблице, устанавливая его запуск перед каждой вставкой новой строки:

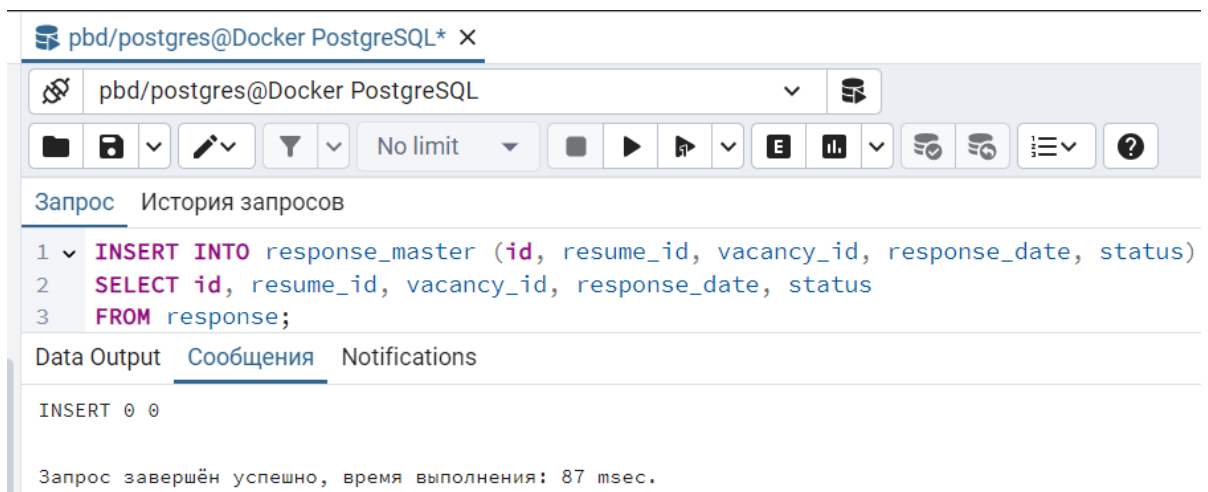


Terminal window titled "pbd/postgres@Docker PostgreSQL* X". The interface shows a query editor with the following SQL code:

```
1 CREATE TRIGGER insert_response_trigger
2 BEFORE INSERT ON response_master
3 FOR EACH ROW EXECUTE FUNCTION response_insert_trigger();
```

Below the query editor, the "Data Output" tab is selected, showing the text "CREATE TRIGGER". At the bottom, a status message reads: "Запрос завершён успешно, время выполнения: 38 msec."

5. Заполним партиции существующими данными из **response**:



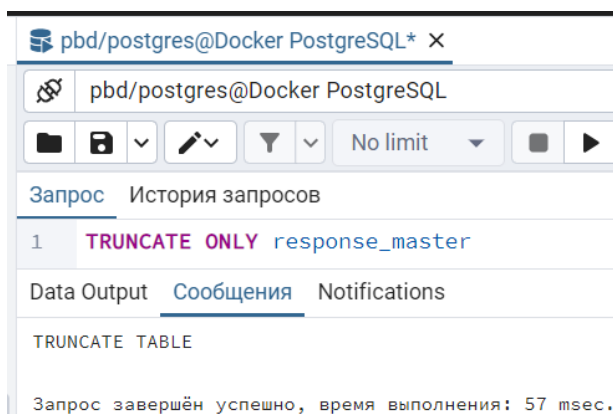
Terminal window titled "pbd/postgres@Docker PostgreSQL* X". The interface shows a query editor with the following SQL code:

```
1 INSERT INTO response_master (id, resume_id, vacancy_id, response_date, status)
2 SELECT id, resume_id, vacancy_id, response_date, status
3 FROM response;
```

Below the query editor, the "Data Output" tab is selected, showing the text "INSERT 0 0". At the bottom, a status message reads: "Запрос завершён успешно, время выполнения: 87 msec."

(INSERT 0 0 так как триггер перехватил вставку в нужную партицию, внутри **response** было 5 строк с датой 2023..)

6. Очистим мастер-таблицу, не затрагивая дочерние:



Terminal window titled "pbd/postgres@Docker PostgreSQL* X". The interface shows a query editor with the following SQL code:

```
1 TRUNCATE ONLY response_master
```

Below the query editor, the "Data Output" tab is selected, showing the text "TRUNCATE TABLE". At the bottom, a status message reads: "Запрос завершён успешно, время выполнения: 57 msec."

Вставим новые данные:

пbd/postgres@Docker PostgreSQL ×

пbd/postgres@Docker PostgreSQL

▼

▼

▼

No limit

▼

▼

E

▼

▼

Запрос

История запросов

1 ▼

INSERT INTO response_master (resume_id, vacancy_id, response_date, status)

2

VALUES

3

(1, 1, '2024-01-01', 'Pending'),

4

(2, 2, '2024-02-02', 'Rejected'),

5

(3, 3, '2024-03-03', 'Invited'),

6

(4, 4, '2024-04-04', 'Pending'),

7

(5, 5, '2024-05-05', 'Rejected'),

8

9

(1, 1, '2025-01-01', 'Pending'),

10

(2, 2, '2025-02-02', 'Rejected'),

11

(3, 3, '2025-03-03', 'Invited'),

12

(4, 4, '2025-04-04', 'Pending'),

13

(5, 5, '2025-05-05', 'Rejected');

Data Output

Сообщения

Notifications

INSERT 0 0

Запрос завершён успешно, время выполнения: 53 мсек.

Проверим данные партиций:
response_2023:

Query: `SELECT * FROM response_2023`

	id integer	resume_id integer	vacancy_id integer	response_date date	status response_status
1	1	1	1	2023-10-16	Pending
2	2	2	2	2023-10-17	Invited
3	3	3	3	2023-10-18	Rejected
4	4	4	4	2023-10-19	Pending
5	5	5	5	2023-10-20	Invited

response_2024:

kbd/postgres@Docker PostgreSQL* X

kbd/postgres@Docker PostgreSQL

Запрос История запросов

1 **SELECT** * **FROM** response_2024

Data Output Сообщения Notifications

	id integer	resume_id integer	vacancy_id integer	response_date date	status response_status
1	1	1	1	2024-01-01	Pending
2	2	2	2	2024-02-02	Rejected
3	3	3	3	2024-03-03	Invited
4	4	4	4	2024-04-04	Pending
5	5	5	5	2024-05-05	Rejected

response_2025:

kbd/postgres@Docker PostgreSQL* X

kbd/postgres@Docker PostgreSQL

Запрос История запросов

1 **SELECT** * **FROM** response_2025

Data Output Сообщения Notifications

	id integer	resume_id integer	vacancy_id integer	response_date date	status response_status
1	6	1	1	2025-01-01	Pending
2	7	2	2	2025-02-02	Rejected
3	8	3	3	2025-03-03	Invited
4	9	4	4	2025-04-04	Pending
5	10	5	5	2025-05-05	Rejected

Вывод: в ходе лабораторной работы научился разделять данные на партии, создавать триггер-функции, в том числе обеспечивающие партиционирование. Убедился в корректности партиционирования на практике.