|  |  |  |
| --- | --- | --- |
| Лабораторная работа №5  Индексы, хранимые процедуры, представления, триггеры | Ф.И.О. | Мельников В. Д. |
| Группа | ИВТ-264 |
| Преподаватель | Аль-Мерри Гаис Мохаммед Салех |
| Дата сдачи | 26.05.25 |

**Индексы**

**Запрос 1.** Вывод топ 100 сотрудников по количеству завершенных задач

Код:

SELECT e.employee\_id, e.employee\_email, COUNT(t.task\_id) AS total\_tasks

FROM employees e

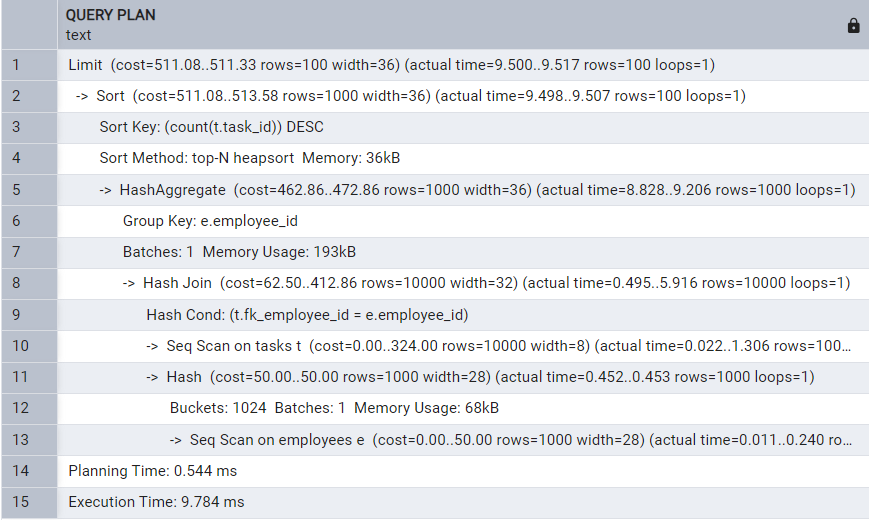
JOIN tasks t ON e.employee\_id = t.fk\_employee\_id

WHERE t.task\_status = 'Завершена'

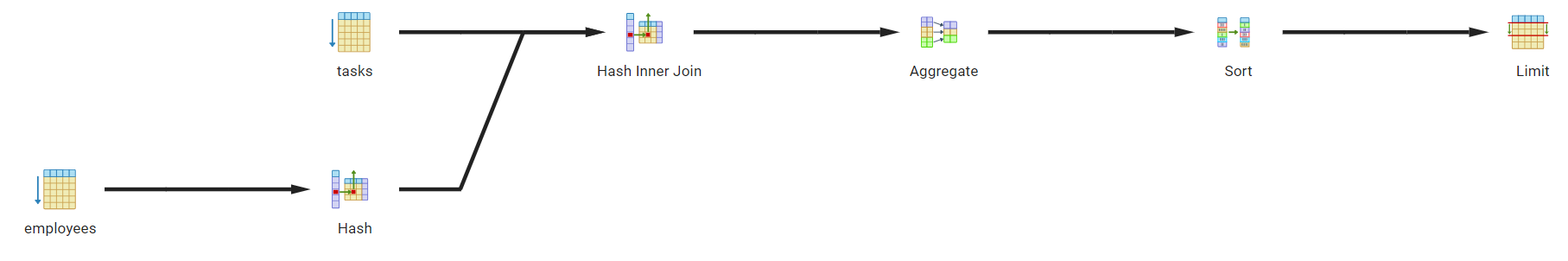
GROUP BY e.employee\_id

ORDER BY total\_tasks DESC

LIMIT 100;  
  
Запрос возвращает почты id и почту сотрудников, имеющее наибольшее количество задач с task\_status = ‘Завершено’.



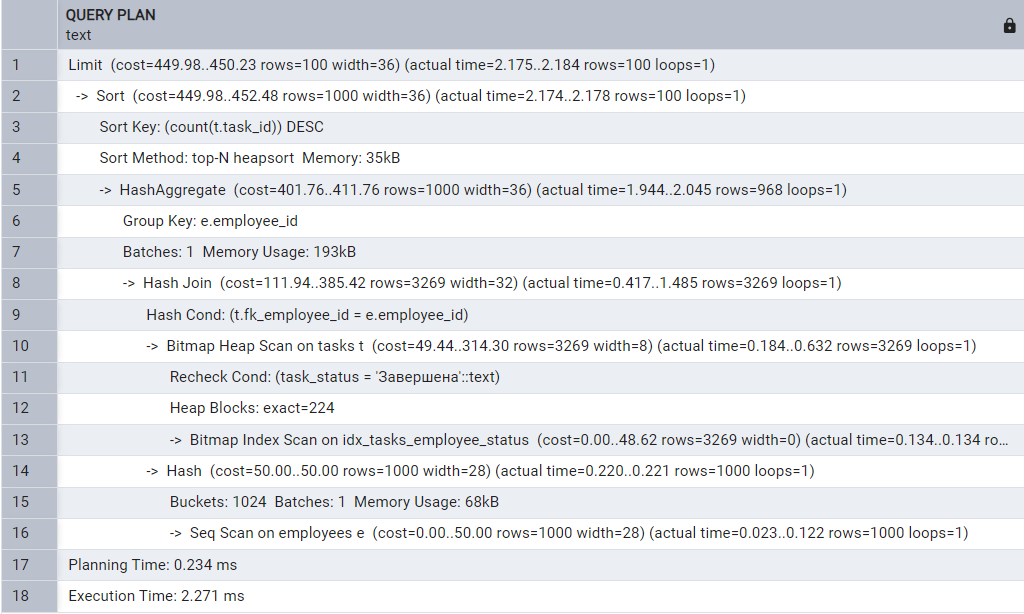
Визуализация до изменений:



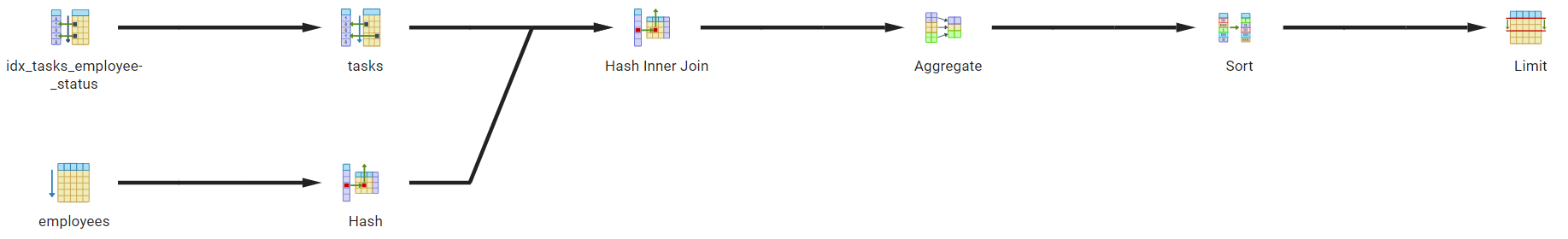
Добавление индекса:

CREATE INDEX idx\_tasks\_status\_employee ON tasks (task\_status, fk\_employee\_id);

Ускоряет фильтрацию по статусу и соединение с employee\_id

Результат запроса после индекса  


Визуализация после изменений



До добавления индексов PostgreSQL выполнял полное сканирование таблиц (Seq Scan). После создания индексов:

- Время сократилось с 5.5 до 2.2

- В плане появился Index Scan

**Запрос 2.** Вывод 10 последних созданных задач (частый запрос)

SELECT task\_description, task\_status, task\_creation\_date

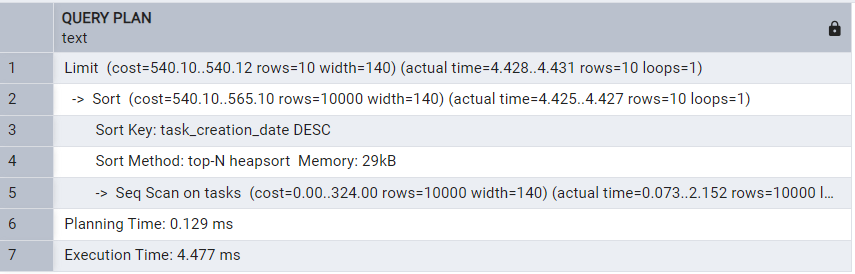
FROM tasks

ORDER BY task\_creation\_date DESC

LIMIT 10

Запрос выводит описание, статус и дату создания 10 последних задач.

Результат до изменений:

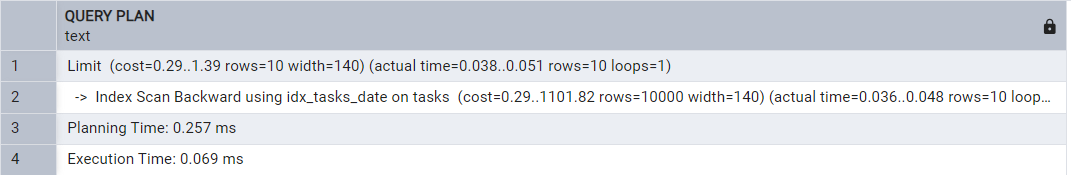


Визуализация до изменений:



Добавление индекса: CREATE INDEX idx\_tasks\_date ON tasks(task\_creation\_date);

Результат после изменений:



Визуализация после изменений:



После создания индексов:

- Время сократилось с 4.477 до 0.069

- До индексов сортировка была всей таблицы с cost = 540.10, после создания индексов PostgreSQL начал использовать Index Scan Backward с cost = 0.29

**Запрос 3.** Сотрудники, у которых более 5 клиентов и хотя бы одна сделка с дорогим тарифом (>3000)

SELECT e.employee\_id, e.employee\_first\_name, COUNT(DISTINCT c.client\_id) AS client\_count

FROM employees e

JOIN clients c ON c.fk\_employee\_id = e.employee\_id

JOIN deals d ON d.fk\_employee\_id = e.employee\_id

JOIN tariffs t ON t.tariff\_id = d.fk\_tariff\_id

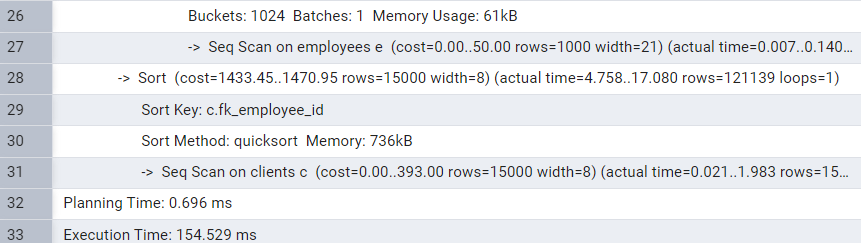
WHERE t.tariff\_price > 3000

GROUP BY e.employee\_id, e.employee\_first\_name

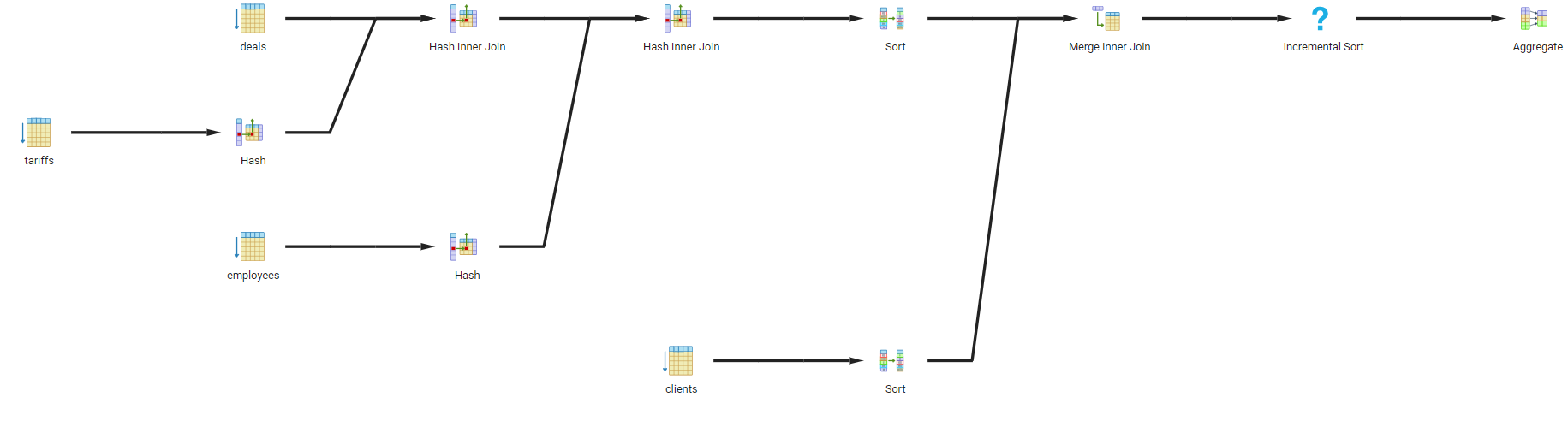
HAVING COUNT(DISTINCT c.client\_id) > 5;

Результат до изменений:





Визуализация до изменений:



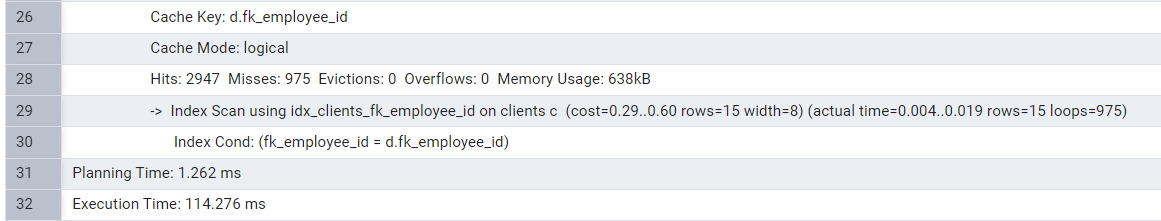
Добавление индексов

CREATE INDEX idx\_clients\_fk\_employee\_id ON clients(fk\_employee\_id);

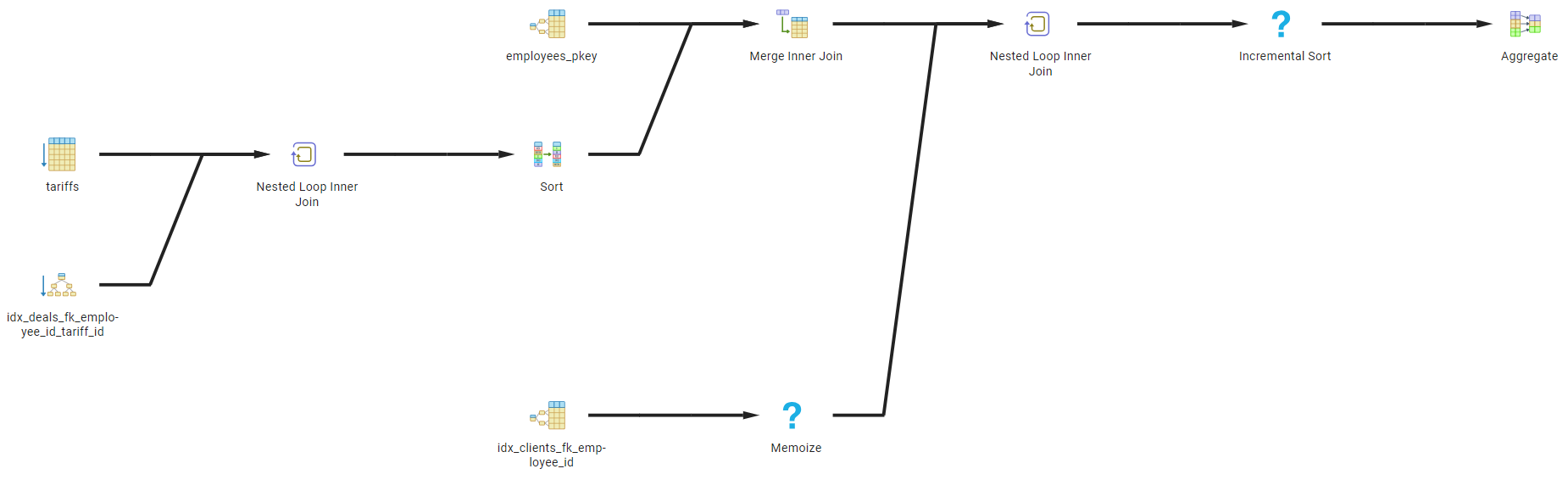
CREATE INDEX idx\_deals\_fk\_employee\_id\_tariff\_id ON deals(fk\_employee\_id, fk\_tariff\_id);

CREATE INDEX idx\_tariffs\_price ON tariffs(tariff\_price);

Результат после изменений:

Визуализация после изменений:



- Вместо медленного Nested Loop используется Merge Join.

- В плане появился Index Scan

**Запрос 4.** Клиенты с успешными сделками и включённым mail-сервером

SELECT c.client\_id, c.client\_email, COUNT(\*) AS deal\_count

FROM clients c

JOIN deals d ON c.client\_id = d.fk\_client\_id

JOIN additional\_services a ON d.deal\_id = a.fk\_deal\_id

WHERE d.deal\_status = 'Заключена'

AND a.mail\_server = TRUE

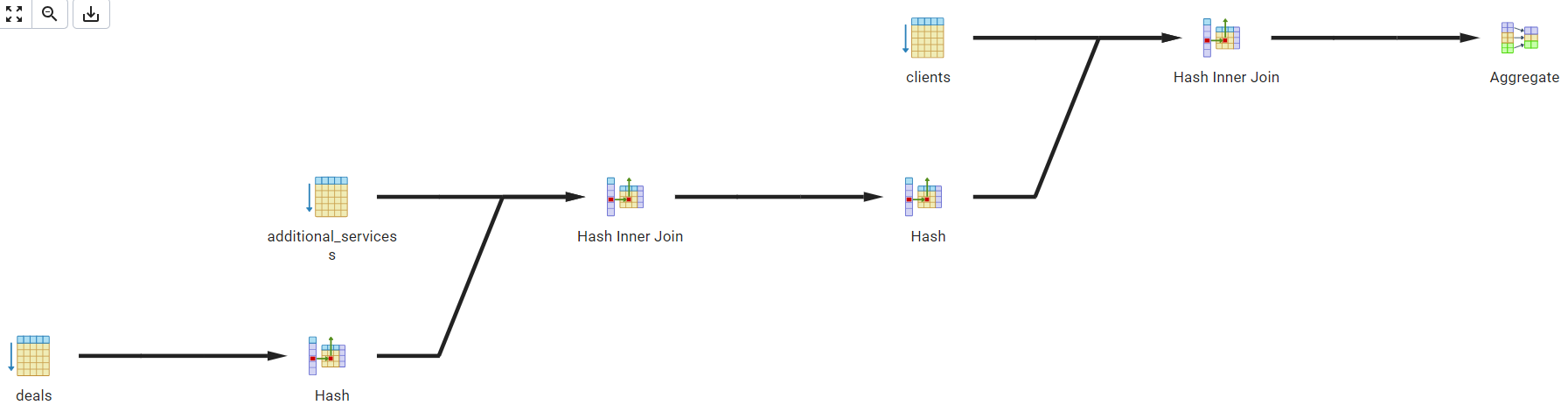
GROUP BY c.client\_id, c.client\_email

HAVING COUNT(\*) > 1;

Резульат до изменений:



Визуализация до изменений



Добавленные индексы:

CREATE INDEX idx\_deals\_client\_status ON deals(fk\_client\_id, deal\_status);

CREATE INDEX idx\_additional\_services\_mail ON additional\_services(fk\_deal\_id)

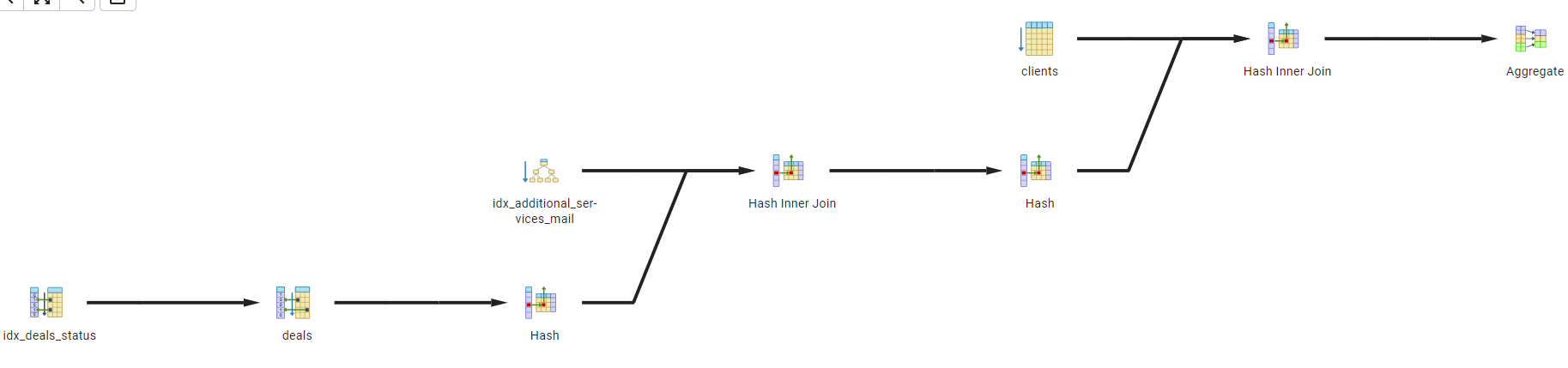
WHERE mail\_server = TRUE;

CREATE INDEX idx\_deals\_status ON deals(deal\_status);

Результат после изменений:



Визуализация после изменений:



- Индекс Only Scan используется для таблицы additional\_services (используется индекс idx\_additional\_services\_mail)

- Bitmap Index Scan используется для таблицы deals (используется индекс idx\_deals\_status)

**Запрос 5.** 100 клиентов по суммарным сделкам в ожидании, у которых указан Telegram.

SELECT

c.client\_id,

c.client\_email,

c.client\_first\_name || ' ' || c.client\_last\_name AS full\_name,

COUNT(d.deal\_id) AS active\_deals,

SUM(t.tariff\_price) AS total\_payment

FROM clients c

JOIN deals d ON c.client\_id = d.fk\_client\_id

JOIN tariffs t ON d.fk\_tariff\_id = t.tariff\_id

JOIN social\_networks sn ON c.client\_id = sn.fk\_client\_id

WHERE d.deal\_status = 'Ожидание'

AND sn.telegram\_link IS NOT NULL

GROUP BY c.client\_id, c.client\_email, c.client\_first\_name, c.client\_last\_name

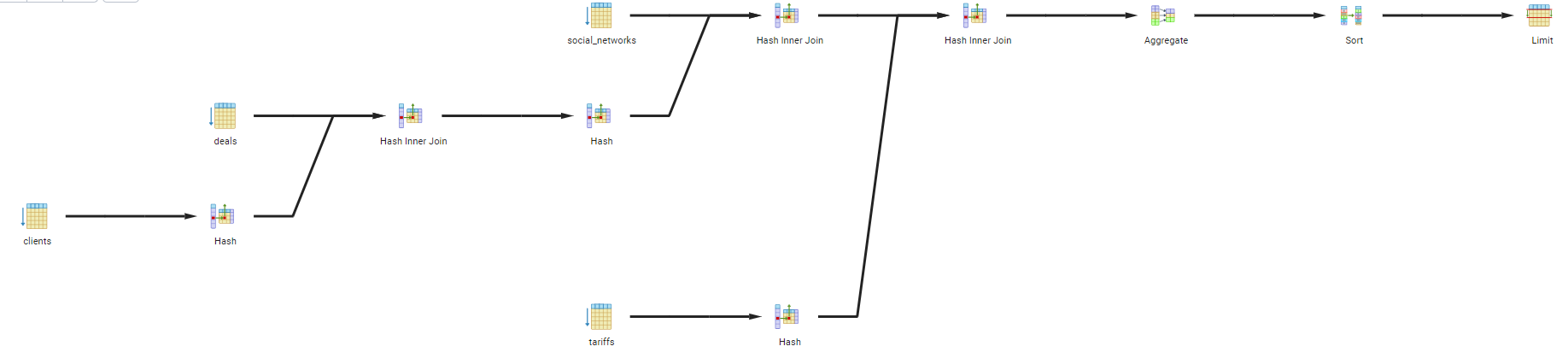
ORDER BY total\_payment DESC

LIMIT 100;

Результат работы до изменений:



Визуализация до изменений:



Использованные индексы:

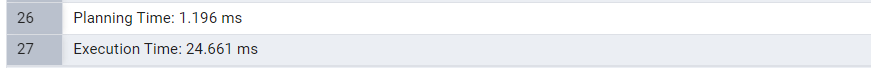
CREATE INDEX idx\_deals\_fk\_client\_id ON deals (fk\_client\_id);

CREATE INDEX idx\_deals\_fk\_tariff\_id ON deals (fk\_tariff\_id);

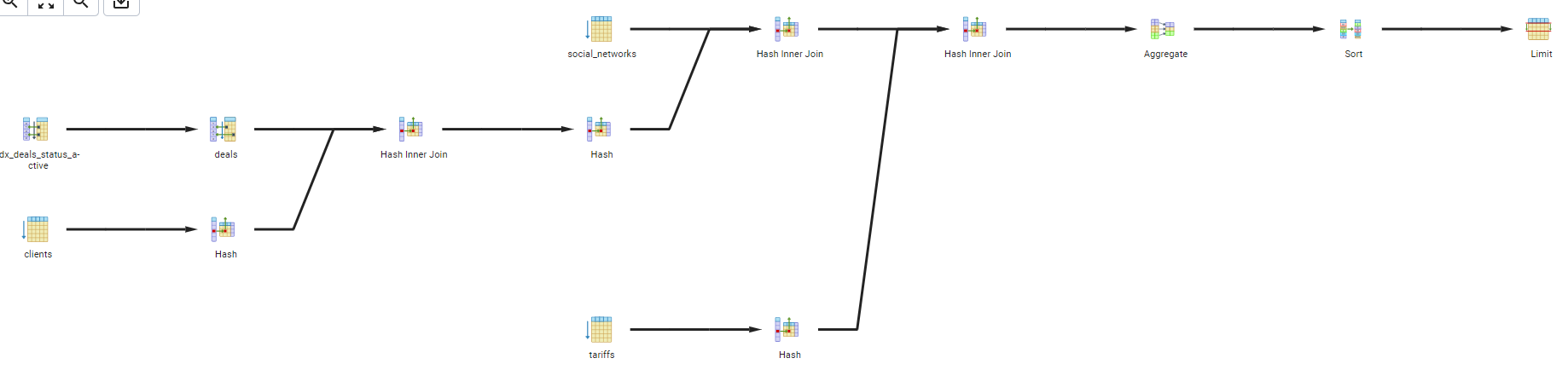
CREATE INDEX idx\_social\_networks\_fk\_client\_id ON social\_networks (fk\_client\_id);

Результат работы после изменений:





Визуализация после изменений:



- Время выполнения запроса снизилось с 3.8 до 1.4

- Во втором плане используется Index Scan using idx\_social\_networks\_fk\_client\_id для поиска по fk\_client\_id

**Хранимые процедуры и функции**

**1. Функция** возвращает полное имя сотрудника по его ID.

CREATE OR REPLACE FUNCTION get\_employee\_full\_name(emp\_id INTEGER)

RETURNS TEXT AS $$

DECLARE

full\_name TEXT;

BEGIN

SELECT CONCAT(e.employee\_last\_name, ' ', e.employee\_first\_name, ' ', e.employee\_patronymic)

INTO full\_name

FROM employees e

WHERE e.employee\_id = emp\_id;

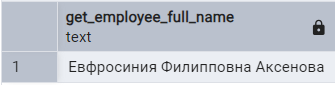
RETURN full\_name;

END;

$$ LANGUAGE plpgsql;

Вызов:

SELECT get\_employee\_full\_name(1);



**2. Функция** возвращает тип клиента ("Юридическое лицо", "Физическое лицо")

CREATE OR REPLACE FUNCTION get\_client\_type(client\_id INTEGER)

RETURNS TEXT AS $$

DECLARE

legal BOOLEAN;

physical BOOLEAN;

BEGIN

SELECT legal\_entity, physical\_entity

INTO legal, physical

FROM right\_statuses

WHERE fk\_client\_id = client\_id;

RETURN CASE

WHEN legal AND NOT physical THEN 'Юридическое лицо'

WHEN NOT legal AND physical THEN 'Физическое лицо'

WHEN legal AND physical THEN 'Смешанный статус'

ELSE 'Не указано'

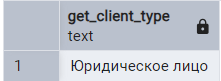
END;

END;

$$ LANGUAGE plpgsql;

Вызов:

SELECT get\_client\_type(1);

****

**3. Функция** возвращает стоимость сделки с учётом выбранных дополнительных услуг (добавляется фиксированная стоимость за каждую услугу).

CREATE OR REPLACE FUNCTION get\_deal\_cost\_with\_services(p\_deal\_id INTEGER)

RETURNS INTEGER AS $$

DECLARE

base\_price INTEGER;

service\_cost INTEGER := 0;

BEGIN

SELECT t.tariff\_price

INTO base\_price

FROM deals d

JOIN tariffs t ON d.fk\_tariff\_id = t.tariff\_id

WHERE d.deal\_id = p\_deal\_id;

SELECT

(CASE WHEN domain\_registration THEN 300 ELSE 0 END) +

(CASE WHEN profiler THEN 150 ELSE 0 END) +

(CASE WHEN mail\_server THEN 200 ELSE 0 END)

INTO service\_cost

FROM additional\_services

WHERE fk\_deal\_id = p\_deal\_id;

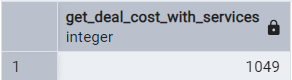
RETURN base\_price + COALESCE(service\_cost, 0);

END;

$$ LANGUAGE plpgsql;

Вызов:

SELECT get\_deal\_cost\_with\_services(1);



**1. Процедура** создаёт новую задачу сотруднику, если у него менее 5 незавершённых задач.

CREATE OR REPLACE PROCEDURE create\_task\_for\_employee(

emp\_id INTEGER,

descr TEXT

)

LANGUAGE plpgsql

AS $$

DECLARE

open\_tasks\_count INTEGER;

BEGIN

SELECT COUNT(\*) INTO open\_tasks\_count

FROM tasks

WHERE fk\_employee\_id = emp\_id AND task\_status != 'Завершена';

IF open\_tasks\_count < 5 THEN

INSERT INTO tasks (task\_description, task\_status, task\_creation\_date, fk\_employee\_id)

VALUES (descr, 'Новая', CURRENT\_DATE, emp\_id);

ELSE

RAISE NOTICE 'Слишком много незавершенных задач у сотрудника %', emp\_id;

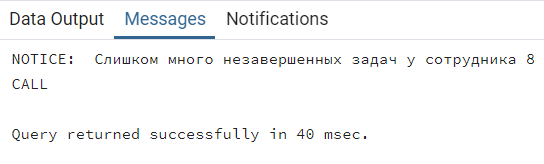
END IF;

END;

$$;

Вызов:

CALL create\_task\_for\_employee(8, 'Создать презентацию для клиента');

****

**Процедура 2.** Назначает сделку сотруднику. Если у сотрудника уже есть больше 10 активных сделок — выдаёт предупреждение и не назначает.

CREATE OR REPLACE PROCEDURE assign\_deal\_to\_employee(

in\_deal\_id INTEGER,

in\_employee\_id INTEGER

)

LANGUAGE plpgsql

AS $$

DECLARE

active\_deals\_count INTEGER;

BEGIN

SELECT COUNT(\*)

INTO active\_deals\_count

FROM deals

WHERE fk\_employee\_id = in\_employee\_id AND deal\_status != 'Закрыта';

IF active\_deals\_count >= 10 THEN

RAISE NOTICE 'Сотрудник % уже имеет 10 или более активных сделок.', in\_employee\_id;

ELSE

UPDATE deals

SET fk\_employee\_id = in\_employee\_id

WHERE deal\_id = in\_deal\_id;

RAISE NOTICE 'Сделка % успешно назначена сотруднику %.', in\_deal\_id, in\_employee\_id;

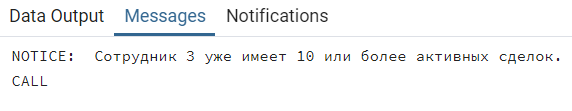
END IF;

END;

$$;

Вызов:

CALL assign\_deal\_to\_employee(2, 3);



**Процедура 3.** Деактивирует все подзадачи (subtasks) у указанной задачи (task\_id), если они ещё активны

CREATE OR REPLACE PROCEDURE deactivate\_all\_subtasks\_for\_task(

in\_task\_id INTEGER

)

LANGUAGE plpgsql

AS $$

DECLARE

subtask\_ids INTEGER[];

i INTEGER := 1;

BEGIN

-- Получаем список ID активных подзадач по задаче

SELECT array\_agg(subtask\_id)

INTO subtask\_ids

FROM subtasks

WHERE fk\_task\_id = in\_task\_id AND subtask\_status = true;

IF subtask\_ids IS NULL THEN

RAISE NOTICE 'Активных подзадач не найдено для задачи %', in\_task\_id;

RETURN;

END IF;

-- LOOP по ID подзадач

LOOP

EXIT WHEN i > array\_length(subtask\_ids, 1);

UPDATE subtasks

SET subtask\_status = false

WHERE subtask\_id = subtask\_ids[i];

RAISE NOTICE 'Подзадача % деактивирована.', subtask\_ids[i];

i := i + 1;

END LOOP;

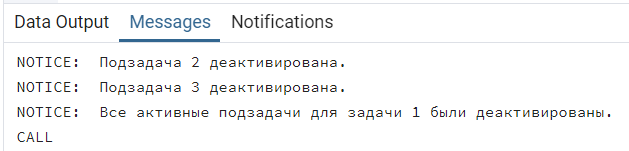
RAISE NOTICE 'Все активные подзадачи для задачи % были деактивированы.', in\_task\_id;

END;

$$;

Вызов:

CALL deactivate\_all\_subtasks\_for\_task(1);

****

**Триггер.** Автоматическая деактивация всех подзадач при закрытии задачи

CREATE OR REPLACE FUNCTION close\_subtasks\_on\_task\_close()

RETURNS trigger

LANGUAGE plpgsql

AS $$

BEGIN

-- Проверка: если задача меняет статус на 'Закрыта'

IF NEW.task\_status = 'Закрыта' AND OLD.task\_status IS DISTINCT FROM 'Закрыта' THEN

UPDATE subtasks

SET subtask\_status = false

WHERE fk\_task\_id = NEW.task\_id AND subtask\_status = true;

RAISE NOTICE 'Все активные подзадачи задачи % были автоматически закрыты.', NEW.task\_id;

END IF;

RETURN NEW;

END;

$$;

CREATE TRIGGER trg\_close\_subtasks\_after\_task\_close

AFTER UPDATE ON tasks

FOR EACH ROW

EXECUTE FUNCTION close\_subtasks\_on\_task\_close();

Вызов:

UPDATE tasks

SET task\_status = 'Закрыта'

WHERE task\_id = 1;

