Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет ИТМО»

Факультет Программной Инженерии и Компьютерной Техники

Лабораторная работа №4

Выполнил:

Горин Семён Дмитриевич

Группа P3108

Проверила:

Заболотняя Ольга Михайловна

Санкт-Петербург 2025

Содержание

Задание

Составить запросы на языке SQL (пункты 1-2).

Для каждого запроса предложить индексы, добавление которых уменьшит время выполнения запроса (указать таблицы/атрибуты, для которых нужно добавить индексы, написать тип индекса; объяснить, почему добавление индекса будет полезным для данного запроса).

Для запросов 1-2 необходимо составить возможные планы выполнения запросов. Планы составляются на основании предположения, что в таблицах отсутствуют индексы. Из составленных планов необходимо выбрать оптимальный и объяснить свой выбор.

Изменятся ли планы при добавлении индекса и как?

Для запросов 1-2 необходимо добавить в отчет вывод команды EXPLAIN ANALYZE [запрос]

Подробные ответы на все вышеперечисленные вопросы должны присутствовать в отчете (планы выполнения запросов должны быть нарисованы, ответы на вопросы - представлены в текстовом виде).

Сделать запрос для получения атрибутов из указанных таблиц, применив фильтры по указанным условиям:

Таблицы: H_ЛЮДИ, H_ВЕДОМОСТИ.

Вывести атрибуты: H_ЛЮДИ.ОТЧЕСТВО, H_ВЕДОМОСТИ.ИД.

Фильтры (AND):

a) H_ЛЮДИ.ФАМИЛИЯ < Афанасьев.

b) H_ВЕДОМОСТИ.ИД < 1490007.

c) H_ВЕДОМОСТИ.ИД < 1490007.

Вид соединения: INNER JOIN.

Сделать запрос для получения атрибутов из указанных таблиц, применив фильтры по указанным условиям:

Таблицы: H_ЛЮДИ, H_ВЕДОМОСТИ, H_СЕССИЯ.

Вывести атрибуты: H_ЛЮДИ.ОТЧЕСТВО, H_ВЕДОМОСТИ.ИД, H_СЕССИЯ.ДАТА.

Фильтры (AND):

a) H_ЛЮДИ.ОТЧЕСТВО = Георгиевич.

b) H_ВЕДОМОСТИ.ЧЛВК_ИД > 117219.

Вид соединения: LEFT JOIN.

Реализация первого запроса

```
SELECT
    Н_ЛЮДИ.ОТЧЕСТВО AS second_name,
    Н_ВЕДОМОСТИ.ИД as document_id
FROM
    Н_ЛЮДИ INNER JOIN Н_ВЕДОМОСТИ ON Н_ЛЮДИ.ИД = Н_ВЕДОМОСТИ.ЧЛВК_ИД
WHERE
    Н_ЛЮДИ.ФАМИЛИЯ < 'Афанасьев'
    AND Н_ВЕДОМОСТИ.ИД < 1490007;
```

Индексы для уменьшения времени исполнения первого запроса

```
CREATE INDEX PEOPLE_LAST_NAME_INDEX ON Н_ЛЮДИ USING BTREE(ФАМИЛИЯ);
CREATE INDEX REPORTS_ID_INDEX ON Н_ВЕДОМОСТИ USING BTREE(ИД);
```

Выборка происходит с использованием операторов сравнения, поэтому наиболее выгодным я считаю использование B-дерева в качестве индекса по каждому из атрибутов, использующихся в запросе. Это позволит уменьшить временные затраты на фильтрацию элементов.

Возможные планы выполнения первого запроса

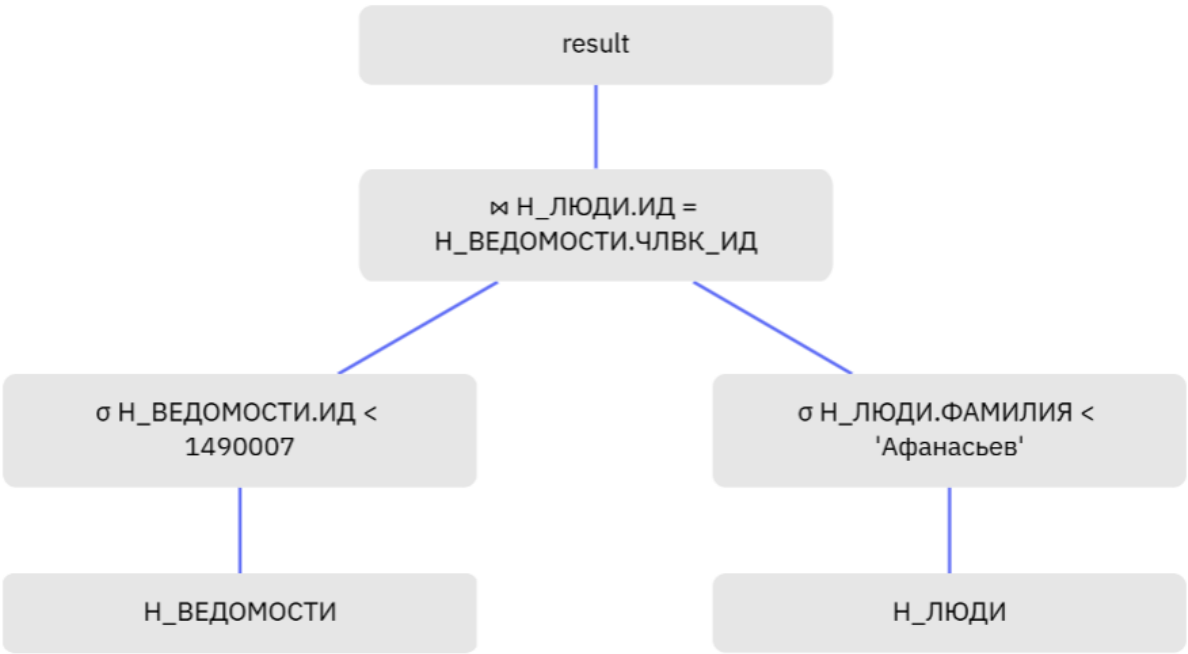

Рисунок 1

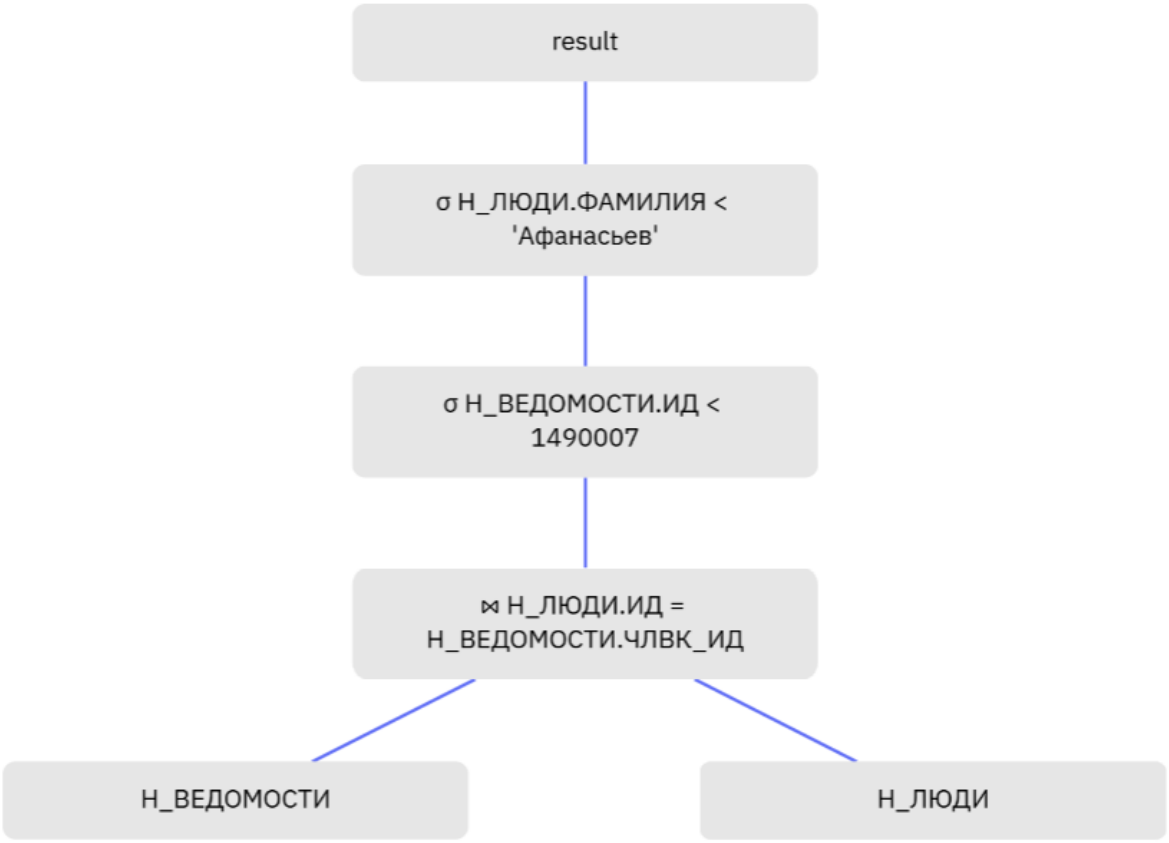

Рисунок 2

План изображенный на рисунке 1 является оптимальным, так как фильтрация происходит до соединения таблиц, что значительно уменьшает затраты на соединение.

EXPLAIN ANALYZE первого запроса

```
                                   QUERY PLAN
-------------------------------------------------------------------------------
 Hash Join  (cost=111.39..7580.27 rows=9431 width=24) (actual
time=0.429..57.889 rows=9366 loops=1)
   Hash Cond: ("Н_ВЕДОМОСТИ"."ЧЛВК_ИД" = "Н_ЛЮДИ"."ИД")
   ->  Seq Scan on "Н_ВЕДОМОСТИ"  (cost=0.00..6884.50 rows=222439 width=8)
(actual time=0.010..33.587 rows=222439 loops=1)
         Filter: ("ИД" < 1490007)
         Rows Removed by Filter: 1
   ->  Hash  (cost=108.68..108.68 rows=217 width=24) (actual
time=0.384..0.387 rows=216 loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 21kB
         ->  Bitmap Heap Scan on "Н_ЛЮДИ"  (cost=5.96..108.68 rows=217
width=24) (actual time=0.139..0.331 rows=216 loops=1)
               Recheck Cond: (("ФАМИЛИЯ")::text < 'Афанасьев'::text)
               Heap Blocks: exact=91
               ->  Bitmap Index Scan on "ФАМ_ЛЮД"  (cost=0.00..5.91 rows=217
width=0) (actual time=0.122..0.123 rows=216 loops=1)
                     Index Cond: (("ФАМИЛИЯ")::text < 'Афанасьев'::text)
 Planning Time: 1.555 ms
 Execution Time: 58.406 ms
```

Разберем ответ снизу-вверх. Postgres применяет индекс “ФАМ_ЛЮД” для фильтрации значений таблицы Н_ЛЮДИ. Затем читает данные из отфильтрованной таблицы. После этого Postgres решает просканировать таблицу Н_ВЕДОМОСТИ с применением фильтра

по атрибуту ИД, не применяя никаких индексов. Затем происходит соединение 2 таблиц по заданному условию с помощью hash join. Т.о. можно действительно подтвердить, что план выполнения, изображенный на рисунке 1, является оптимальным, ведь именно его предпочла СУБД.

Реализация второго запроса

```
SELECT
    Н_ЛЮДИ.ОТЧЕСТВО AS second_name,
    Н_ВЕДОМОСТИ.ИД AS document_id,
    Н_СЕССИЯ.ДАТА AS date_of_exam
FROM
    Н_ВЕДОМОСТИ LEFT JOIN (Н_ЛЮДИ LEFT JOIN Н_СЕССИЯ ON Н_ЛЮДИ.ИД =
Н_СЕССИЯ.ЧЛВК_ИД) ON Н_ЛЮДИ.ИД = Н_ВЕДОМОСТИ.ЧЛВК_ИД
WHERE
    Н_ЛЮДИ.ОТЧЕСТВО = 'Георгиевич'
    AND Н_ВЕДОМОСТИ.ЧЛВК_ИД > 117219;
```

Индексы для уменьшения времени исполнения первого запроса

```
CREATE INDEX PEOPLE_SECOND_NAME_INDEX ON Н_ЛЮДИ USING BTREE(ОТЧЕСТВО);
CREATE INDEX REPORTS_PEOPLE_ID_INDEX ON Н_ВЕДОМОСТИ USING BTREE(ЧЛВК_ИД);
```

Так же, как и в предыдущем примере для обоих индексов было решено использовать B-дерево. В то время как второй индекс аналогичен уже приведенным выше, для первого необходимы некоторые разъяснения. Я выбрал B-дерево, а не хэш, так как в Postgres, хэш не даёт значительного прироста скорости по сравнению с деревом, однако гораздо менее гибок, и реже используется в связи с чем индекс будет сложнее переиспользовать для других запросов к данной таблице.

Возможные планы выполнения второго запроса

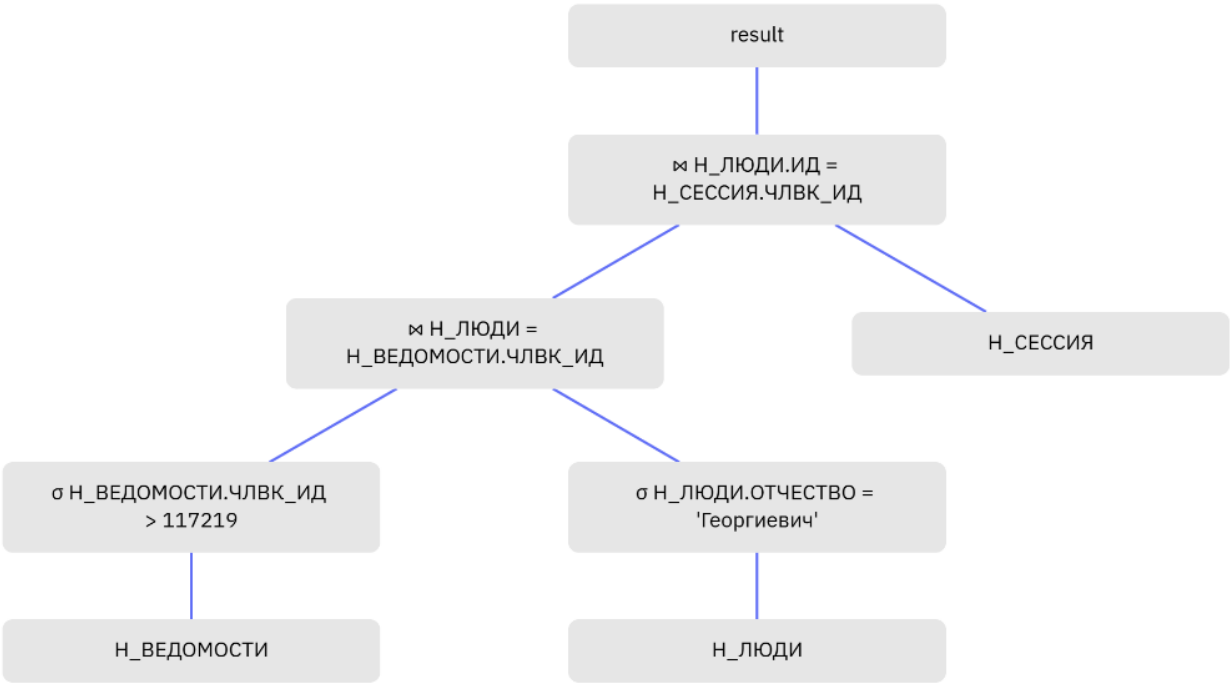

Рисунок 3

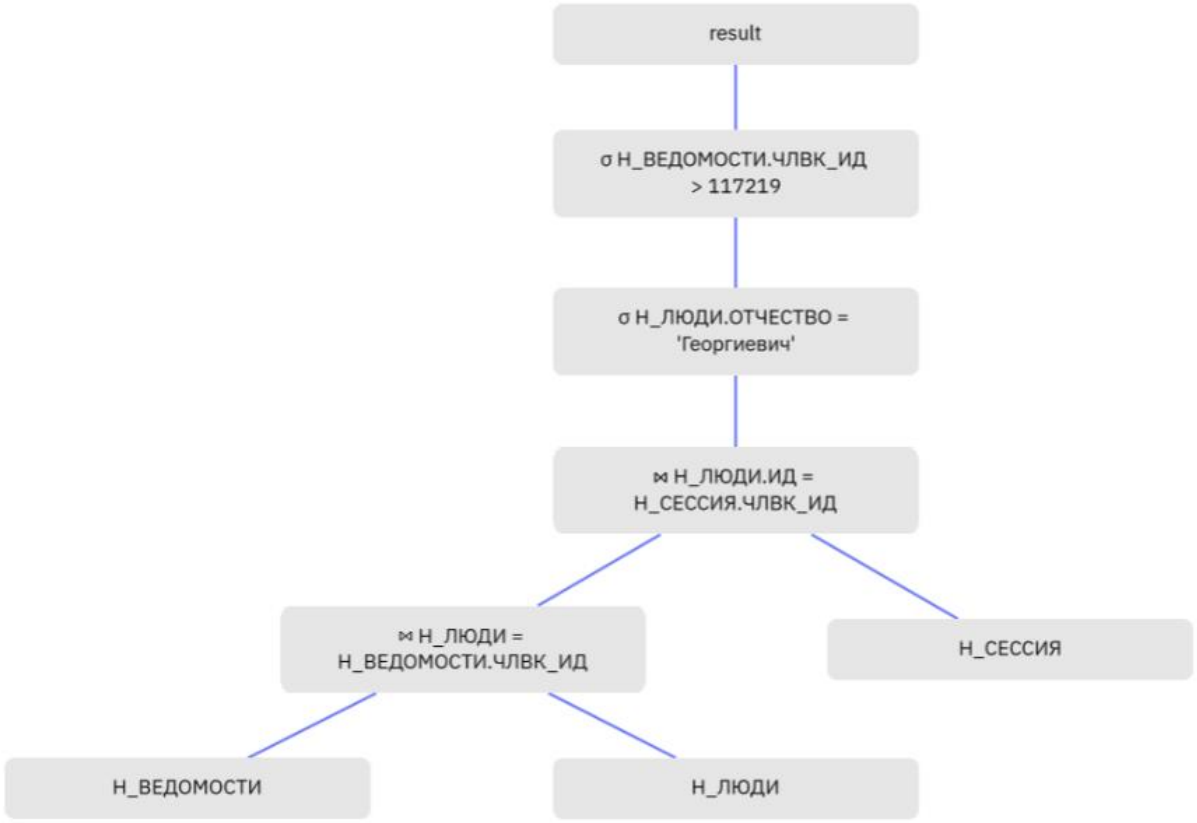

Рисунок 4

План изображенный на рисунке 3 является более оптимальным, так как фильтрация происходит до соединения таблиц, что значительно уменьшает затраты на соединение.

EXPLAIN ANALYZE второго запроса

```
                                                                QUERY PLAN
------------------------------------------------------------------------------------------
 Nested Loop  (cost=164.59..4143.65 rows=920 width=32) (actual
time=1.519..2.256 rows=665 loops=1)
   ->  Hash Right Join  (cost=164.30..282.68 rows=26 width=32) (actual
time=1.043..1.494 rows=42 loops=1)
         Hash Cond: ("Н_СЕССИЯ"."ЧЛВК_ИД" = "Н_ЛЮДИ"."ИД")
         ->  Seq Scan on "Н_СЕССИЯ"  (cost=0.00..108.52 rows=3752 width=12)
(actual time=0.003..0.328 rows=3752 loops=1)
         ->  Hash  (cost=163.97..163.97 rows=26 width=24) (actual
time=0.692..0.692 rows=26 loops=1)
               Buckets: 1024  Batches: 1  Memory Usage: 10kB
               ->  Seq Scan on "Н_ЛЮДИ"  (cost=0.00..163.97 rows=26 width=24)
(actual time=0.026..0.680 rows=26 loops=1)
                     Filter: (("ОТЧЕСТВО")::text = 'Георгиевич'::text)
                     Rows Removed by Filter: 5092
   ->  Index Scan using "ВЕД_ЧЛВК_FK_IFK" on "Н_ВЕДОМОСТИ"
(cost=0.29..147.94 rows=56 width=8) (actual time=0.002..0.016 rows=16
loops=42)
         Index Cond: (("ЧЛВК_ИД" = "Н_ЛЮДИ"."ИД") AND ("ЧЛВК_ИД" > 117219))
 Planning Time: 1.568 ms
 Execution Time: 2.369 ms
```

Разберем ответ. Сначала Postgres выполняет полное сканирование таблицы Н_ЛЮДИ с применением фильтра по атрибуту ОТЧЕСТВО. Затем также полностью прочитал все строки Н_СЕССИЯ, после чего соединил две таблицы с помощью hash join. Стоит заметить, что Postgres заменил Left Join, на Right Join поменяв местами таблицы (видимо так запрос выполится быстрее?). Дальше поиск по Н_ВЕДОМОСТИ с использованием индекса и финальное соединение таблиц посредством nested loop.

Выводы

Во время лабораторной работы я лучше понял как СУБД понимает запросы и оптимизирует их, рассмотрел какие структуры данных используются для ускорения запросов, узнал о внутреннем устройстве соединения таблиц в Postgres.