

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

Факультет Программной Инженерии и Компьютерной Техники

Лабораторная работа №4

По Базам Данных

Вариант 34561

Выполнил:

Ларионов Владислав Васильевич

Группа Р3109

Практик:

Воронина Дарья Сергеевна

Санкт-Петербург 2025

# Содержание

<b>Задание .....</b>	<b>3</b>
<b>Выполнение задания .....</b>	<b>4</b>
<b>Запрос 1 .....</b>	<b>4</b>
<b>1.1 Запрос 1 на языке PostgreSQL .....</b>	<b>4</b>
<b>1.2 Индексы для запроса 1.....</b>	<b>4</b>
<b>1.3 План выполнения без индексов для запроса 1 .....</b>	<b>5</b>
<b>1.4 EXPLAIN ANALYZE для запроса 1 .....</b>	<b>5</b>
<b>Запрос 2 .....</b>	<b>7</b>
<b>2.1 Запрос 2 на языке PostgreSQL .....</b>	<b>7</b>
<b>2.2 Индексы для запроса 2.....</b>	<b>7</b>
<b>2.3 План выполнения без индексов для запроса 2 .....</b>	<b>9</b>
<b>2.4 EXPLAIN ANALYZE для запроса 2.....</b>	<b>11</b>
<b>Вывод: .....</b>	<b>12</b>

## 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. Изменяются 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).

1. Сделать запрос для получения атрибутов из указанных таблиц, применив фильтры по указанным условиям:  
Н\_ТИПЫ\_ВЕДОМОСТЕЙ, Н\_ВЕДОМОСТИ.  
Вывести атрибуты: Н\_ТИПЫ\_ВЕДОМОСТЕЙ.НАИМЕНОВАНИЕ, Н\_ВЕДОМОСТИ.ДАТА.  
Фильтры (AND):  
а) Н\_ТИПЫ\_ВЕДОМОСТЕЙ.ИД < 3.  
б) Н\_ВЕДОМОСТИ.ИД = 1457443.  
Вид соединения: LEFT JOIN.
2. Сделать запрос для получения атрибутов из указанных таблиц, применив фильтры по указанным условиям:  
Таблицы: Н\_ЛЮДИ, Н\_ВЕДОМОСТИ, Н\_СЕССИЯ.  
Вывести атрибуты: Н\_ЛЮДИ.ОТЧЕСТВО, Н\_ВЕДОМОСТИ.ЧЛВК\_ИД, Н\_СЕССИЯ.ДАТА.  
Фильтры (AND):  
а) Н\_ЛЮДИ.ИМЯ = Ярослав.  
б) Н\_ВЕДОМОСТИ.ИД > 1250972.

с) Н\_СЕССИЯ.ИД < 27640.

Вид соединения: INNER JOIN.

## Выполнение задания

### Запрос 1

#### 1.1 Запрос 1 на языке PostgreSQL

```
1 SELECT
2     Н_ТИПЫ_ВЕДОМОСТЕЙ.НАИМЕНОВАНИЕ,
3     Н_ВЕДОМОСТИ.ДАТА
4 FROM
5     Н_ТИПЫ_ВЕДОМОСТЕЙ
6 LEFT JOIN
7     Н_ВЕДОМОСТИ ON Н_ТИПЫ_ВЕДОМОСТЕЙ.ИД = Н_ВЕДОМОСТИ.ТВ_ИД
8 WHERE
9     Н_ТИПЫ_ВЕДОМОСТЕЙ.ИД < 3
10    AND Н_ВЕДОМОСТИ.ИД = 1457443;
```

#### 1.2 Индексы для запроса 1

CREATE INDEX idx\_ведомости\_тв\_ид ON Н\_ВЕДОМОСТИ(ТВ\_ИД);

Тип: B-tree

Ускорит соединение с Н\_ВЕДОМОСТИ

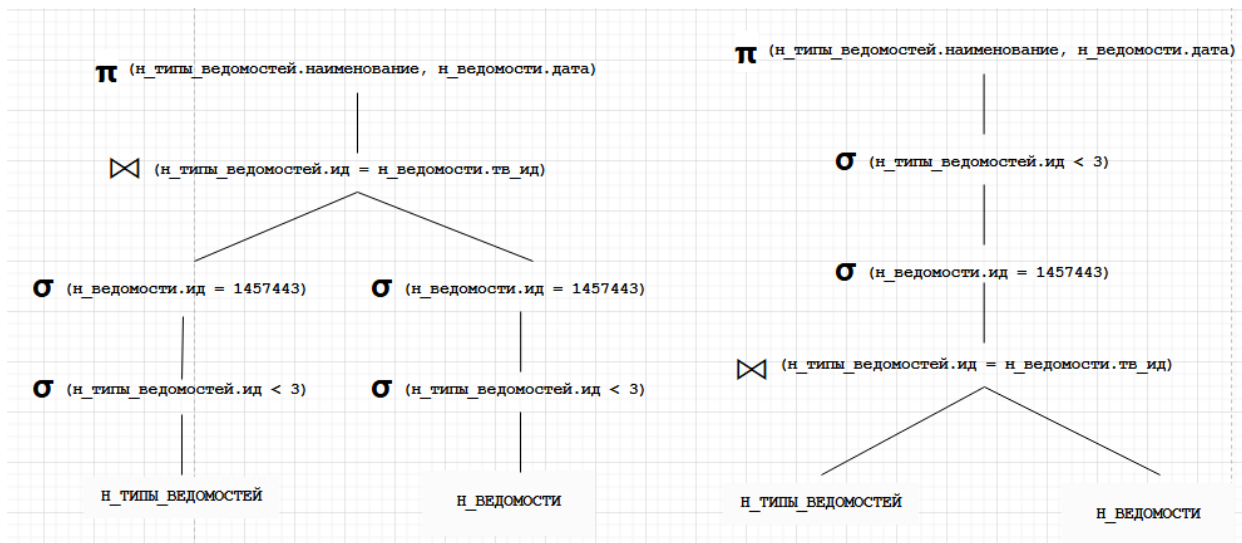
tablename	attname	n_distinct	null_frac	avg_width	correlation
Н_ВЕДОМОСТИ	ТВ_ИД	3	0	4	0.8490781

total_rows
222440

indexname	indexdef
БЕД_РК	CREATE UNIQUE INDEX "БЕД_РК" ON public."Н_ВЕДОМОСТИ" USING btree ("ИД")
БЕД_ДАТА_I	CREATE INDEX "БЕД_ДАТА_I" ON public."Н_ВЕДОМОСТИ" USING btree ("ДАТА")
БЕД_ИП_FK_I	CREATE INDEX "БЕД_ИП_FK_I" ON public."Н_ВЕДОМОСТИ" USING btree ("СЭС_ИД")
БЕД_ОТД_I	CREATE INDEX "БЕД_ОТД_I" ON public."Н_ВЕДОМОСТИ" USING btree ("ОТД_ИД")
БЕД_ОЦЕНКА_I	CREATE INDEX "БЕД_ОЦЕНКА_I" ON public."Н_ВЕДОМОСТИ" USING btree ("ОЦЕНКА")
БЕД_ТВ_FK_I	CREATE INDEX "БЕД_ТВ_FK_I" ON public."Н_ВЕДОМОСТИ" USING btree ("ТВ_ИД")
БЕД_ЧЛВК_FK_IFK	CREATE INDEX "БЕД_ЧЛВК_FK_IFK" ON public."Н_ВЕДОМОСТИ" USING btree ("ЧЛВК_ИД")

Как мы видим, всего 3 уникальных значения (на 222440 записей) и список хорошо отсортирован, то есть индекс не будет полезным (несмотря на это, такой индекс уже существует)

## 1.3 План выполнения без индексов для запроса 1



Как мы видим, первый план выполнения более оптимальный, так как сначала происходит отбор по условиям, а только потом объединение таблиц (то есть объединяется меньшее количество строк)

## 1.4 EXPLAIN ANALYZE для запроса 1

```
QUERY PLAN
-----
Nested Loop (cost=0.42..9.49 rows=1 width=426) (actual time=0.028..0.034 rows=1 loops=1)
  Join Filter: ("н_типы_ведомостей"."ИД" = "н_ведомости"."ТВ_ИД")
  Rows Removed by Join Filter: 1
  -> Seq Scan on "н_типы_ведомостей" (cost=0.00..1.04 rows=1 width=422) (actual time=0.013..0.014 rows=2 loops=1)
        Filter: ("ИД" < 3)
        Rows Removed by Filter: 1
  -> Index Scan using "ВЕД_РК" on "н_ведомости" (cost=0.42..8.44 rows=1 width=12) (actual time=0.006..0.007 rows=1 loops=2)
        Index Cond: ("ИД" = 1457443)
Planning Time: 0.314 ms
Execution Time: 0.071 ms
(10 строк)
```

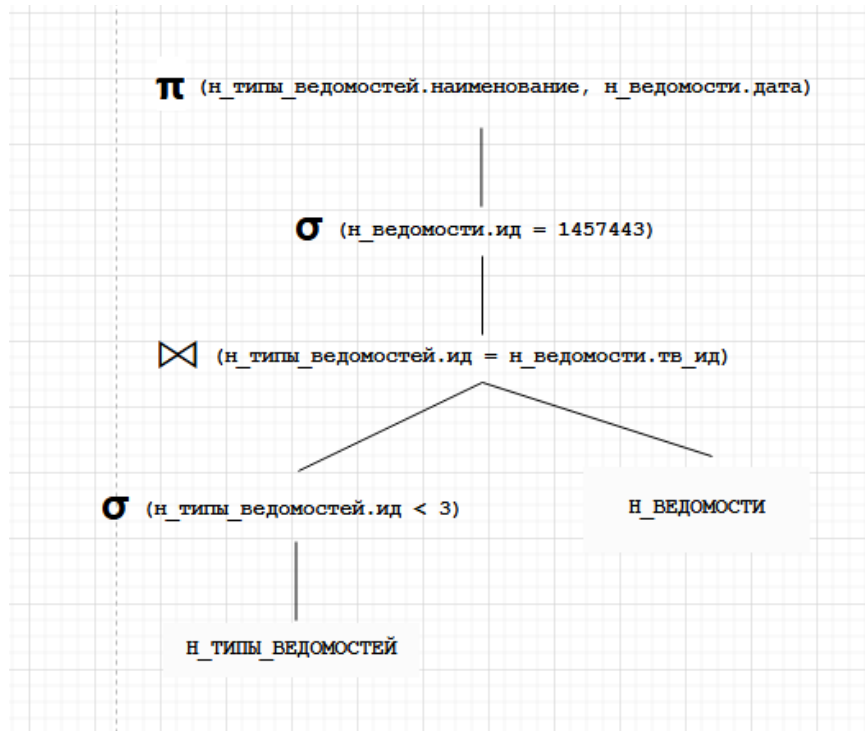
Запрос соединяет таблицы "н\_типы\_ведомостей" и "н\_ведомости" через Nested Loop.

Сначала выполняется полное сканирование "н\_типы\_ведомостей" с фильтром "ИД < 3", которое находит 2 строки и отбрасывает 1.

Затем для каждой из этих строк происходит поиск в "н\_ведомости" по индексу первичного ключа "ВЕД\_РК" с условием "ИД = 1457443", каждый поиск занимает около 0.015 мс.

После этого применяется условие соединения ("Н\_ТИПЫ\_ВЕДОМОСТЕЙ"."ИД" = "Н\_ВЕДОМОСТИ"."ТВ\_ИД"), из двух возможных комбинаций остается одна строка, а вторая отбрасывается.

Общее время выполнения запроса — 0.122 мс, время планирования — 1.117 мс.



## Запрос 2

### 2.1 Запрос 2 на языке PostgreSQL

```
SELECT
  Н_ЛЮДИ.ОТЧЕСТВО,
  Н_ВЕДОМОСТИ.ЧЛВК_ИД,
  Н_СЕССИЯ.ДАТА
FROM
  Н_ЛЮДИ
JOIN
  Н_ВЕДОМОСТИ ON Н_ЛЮДИ.ИД = Н_ВЕДОМОСТИ.ЧЛВК_ИД
JOIN
  Н_СЕССИЯ ON Н_ЛЮДИ.ИД = Н_СЕССИЯ.ЧЛВК_ИД
WHERE
  Н_ЛЮДИ.ИМЯ = 'Ярослав'
  AND Н_ВЕДОМОСТИ.ИД > 1250972
  AND Н_СЕССИЯ.ИД < 27640;
```

### 2.2 Индексы для запроса 2

CREATE INDEX idx\_люди\_имя ON Н\_ЛЮДИ USING HASH (ИМЯ);

Тип: Hash (проверяется всего одно значение, для проверки равенства быстрее, чем B-tree)

tablename	attname	n_distinct	null_frac	avg_width	correlation
Н_ЛЮДИ	ИМЯ	350	0	13	0.032884054

count
5118

Как мы видим, 350 уникальных значений (всего 5118), то есть примерно по 15 позиций на каждое имя, что не оправдывает использование индекса. Но, с другой стороны, значения плохо отсортированы, однако поддержание индекса будет дороже обходиться, чем использование seq scan на все имена. Поэтому индексом можно пренебречь.

```
CREATE INDEX idx_ведомости_члвк_ид ON Н_ВЕДОМОСТИ(ЧЛВК_ИД);
```

Тип: B-tree

Ускоряет соединение с Н\_ЛЮДИ

tablename	attname	n_distinct	null_frac	avg_width	correlation
Н_ВЕДОМОСТИ	ЧЛВК_ИД	3261	0	4	0.51113844

indexname	indexdef
ВЕД_РК	CREATE UNIQUE INDEX "ВЕД_РК" ON public."Н_ВЕДОМОСТИ" USING btree ("ИД")
ВЕД_ИП_FK_I	CREATE INDEX "ВЕД_ИП_FK_I" ON public."Н_ВЕДОМОСТИ" USING btree ("СЭС_ИД")
ВЕД_ОТД_I	CREATE INDEX "ВЕД_ОТД_I" ON public."Н_ВЕДОМОСТИ" USING btree ("ОТД_ИД")
ВЕД_TV_FK_I	CREATE INDEX "ВЕД_TV_FK_I" ON public."Н_ВЕДОМОСТИ" USING btree ("ТВ_ИД")
ВЕД_ЧЛВК_FK_IFK	CREATE INDEX "ВЕД_ЧЛВК_FK_IFK" ON public."Н_ВЕДОМОСТИ" USING btree ("ЧЛВК_ИД")

Индекс уже существует, но если бы его не было, то его создание не было бы лишним – 3261 уникальных значений и отсортировано всего половина списка.

```
CREATE INDEX idx_сессия_члвк_ид ON Н_СЕССИЯ(ЧЛВК_ИД);
```

Тип: B-tree

Ускорит соединение с Н\_ЛЮДИ

tablename	attname	n_distinct	null_frac	avg_width	correlation
Н_СЕССИЯ	ЧЛВК_ИД	180	0.13672708	4	0.13944401

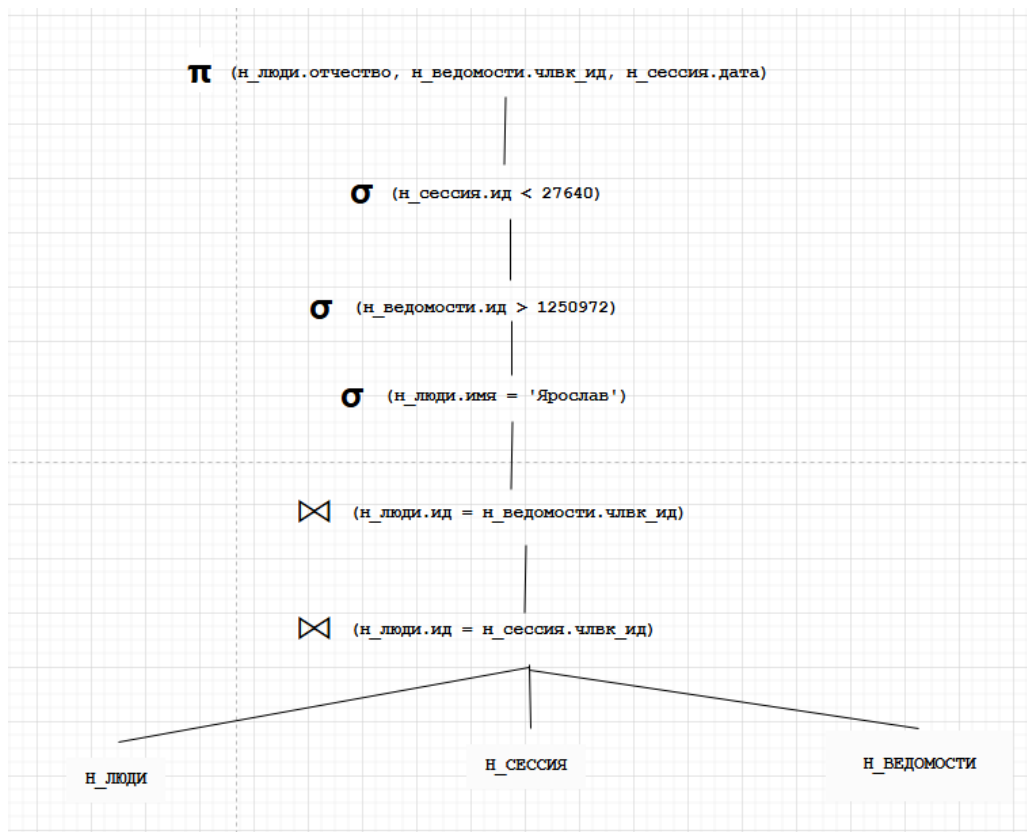
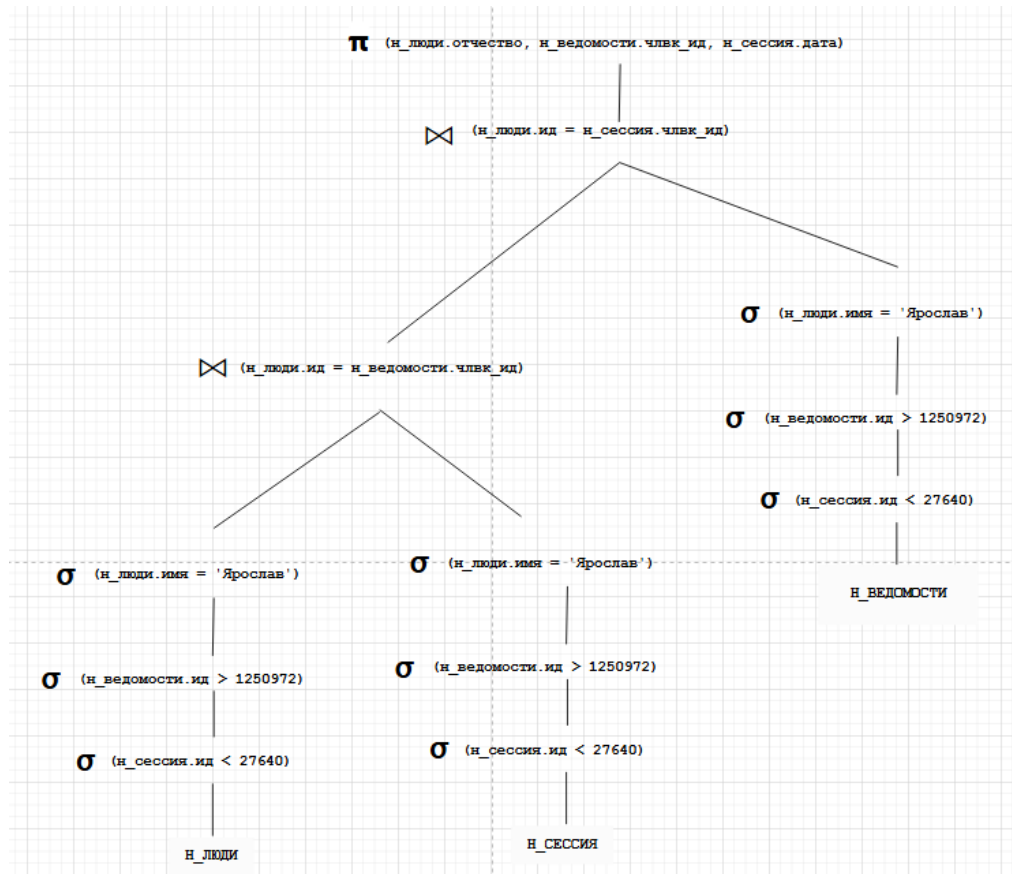
total_rows	unique_values	null_count	uniqueness_percent
3752	180	513	4.80

indexname	indexdef
SYS_C003500_IFK	CREATE INDEX "SYS_C003500_IFK" ON public."Н_СЕССИЯ" USING btree ("ЧЛВК_ИД")
СЕС_СЭС_FK	CREATE INDEX "СЕС_СЭС_FK" ON public."Н_СЕССИЯ" USING btree ("СЭС_ИД")

Индекс уже существует, но оценим, насколько он был бы необходим, если бы его не было. Всего 180 уникальных значений на 3752 (около 20 строк на одно значение), что не оправдывает использование индекса. Но, с другой стороны, значения плохо отсортированы, однако поддержание индекса будет дороже обходиться, чем использование seq scan на все значения. Поэтому индексом можно было бы и пренебречь.



## 2.3 План выполнения без индексов для запроса 2



Как мы видим, первый план выполнения более оптимальный, так как используется левостороннее дерево, мы используем как можно раньше выборку после чего используем объединение таблиц.

Индексы сильно ускоряют запрос благодаря замене Seq Scan на Index Scan.

Для соединения таблиц Н\_ЛЮДИ и Н\_СЕССИЯ рекомендуется использовать Nested Loop Join. Это объясняется небольшим размером обеих таблиц (5,118 и 3,752 строк соответственно) и наличием эффективного индекса по полю соединения. Среднее количество сессий на одного человека составляет около 0.63, что делает такой тип соединения наиболее подходящим для данного случая. Время выполнения подобных запросов обычно не превышает 1 миллисекунды.

Для соединения таблиц Н\_ЛЮДИ и Н\_ВЕДОМОСТИ предпочтительнее применять Hash Join. Несмотря на небольшой размер таблицы Н\_ЛЮДИ (5,118 строк), таблица Н\_ВЕДОМОСТИ значительно больше (222,440 строк). Среднее количество ведомостей на одного человека составляет около 43.5, а система уже использует параллельное выполнение с двумя worker'ами. В таких условиях Hash Join показывает лучшую производительность со временем выполнения около 15-20 миллисекунд.

## 2.4 EXPLAIN ANALYZE для запроса 2

```
QUERY PLAN
-----
Nested Loop (cost=0.59..349.58 rows=51 width=32) (actual time=2.317..2.318 rows=0 loops=1)
  Join Filter: ("Н_ЛЮДИ"."ИД" = "Н_ВЕДОМОСТИ"."ЧЛВК_ИД")
  -> Nested Loop (cost=0.29..283.40 rows=10 width=36) (actual time=0.417..2.293 rows=16 loops=1)
    -> Seq Scan on "Н_СЕССИЯ" (cost=0.00..117.90 rows=3461 width=12) (actual time=0.009..0.672 rows=3460 loops=1)
      Filter: ("ИД" < 27640)
      Rows Removed by Filter: 292
    -> Memoize (cost=0.29..0.45 rows=1 width=24) (actual time=0.000..0.000 rows=0 loops=3460)
      Cache Key: "Н_СЕССИЯ"."ЧЛВК_ИД"
      Cache Mode: logical
      Hits: 3285 Misses: 175 Evictions: 0 Overflows: 0 Memory Usage: 12kB
      -> Index Scan using "ЧЛВК_РК" on "Н_ЛЮДИ" (cost=0.28..0.44 rows=1 width=24) (actual time=0.002..0.002 rows=0 loops=175)
        Index Cond: ("ИД" = "Н_СЕССИЯ"."ЧЛВК_ИД")
        Filter: (("ИМЯ")::text = 'Ярослав')::text
        Rows Removed by Filter: 1
  -> Index Scan using "ВЕД_ЧЛВК_FK_IFK" on "Н_ВЕДОМОСТИ" (cost=0.29..6.52 rows=8 width=4) (actual time=0.001..0.001 rows=0 loops=16)
    Index Cond: ("ЧЛВК_ИД" = "Н_СЕССИЯ"."ЧЛВК_ИД")
    Filter: ("ИД" > 1250972)
Planning Time: 0.998 ms
Execution Time: 2.396 ms
(19 строк)
```

Запрос использует Nested Loop для соединения таблиц "Н\_ЛЮДИ" и "Н\_ВЕДОМОСТИ" по условию "Н\_ЛЮДИ"."ИД" = "Н\_ВЕДОМОСТИ"."ЧЛВК\_ИД", но не находит подходящих строк (rows=0). Внутри этого соединения выполняется еще один Nested Loop между таблицами "Н\_СЕССИЯ" и "Н\_ЛЮДИ".

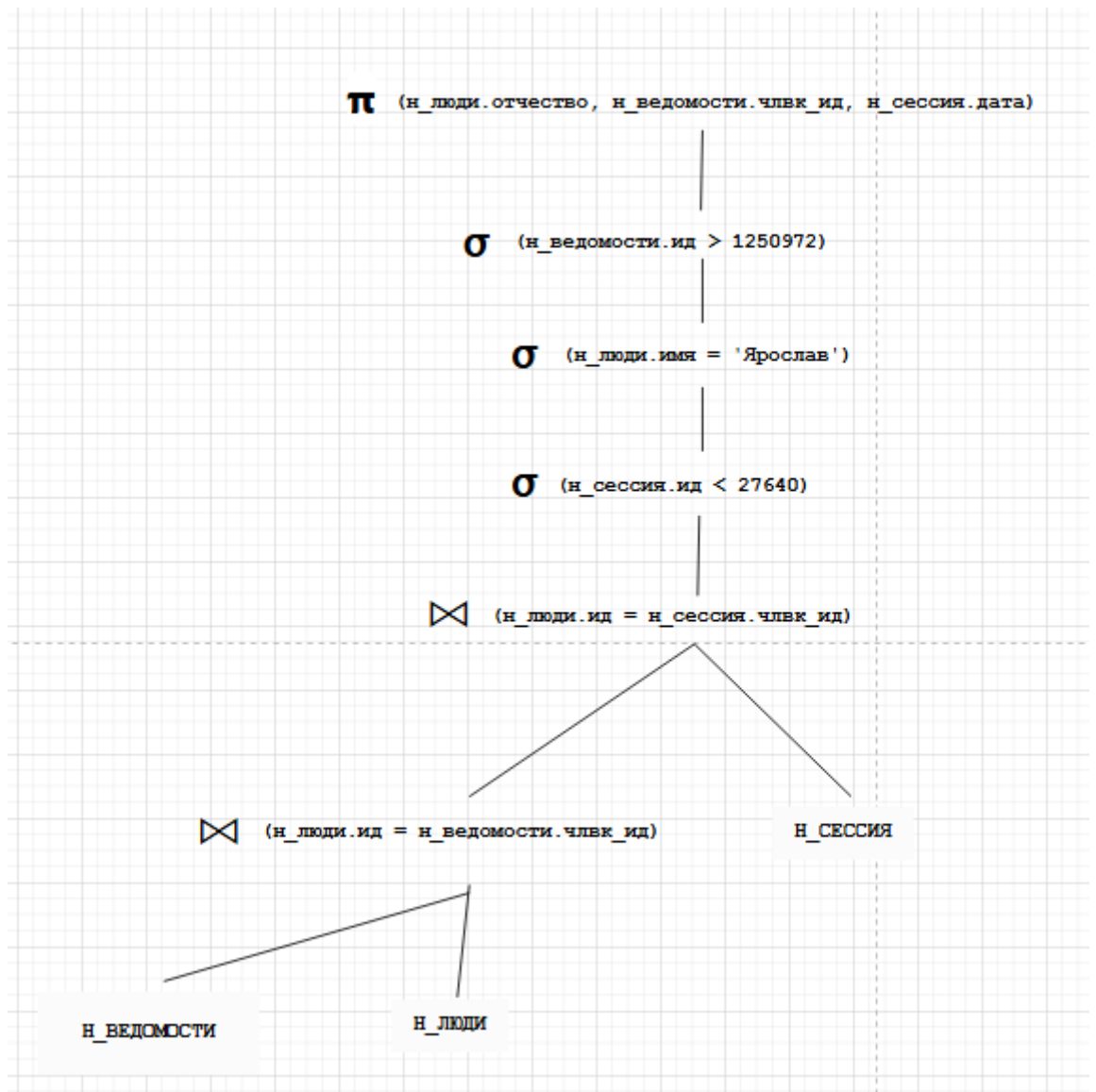
Сначала происходит Seq Scan таблицы "Н\_СЕССИЯ" с фильтром "ИД" < 27640, который обрабатывает 3460 строк и отбрасывает 292.

Для каждой из этих строк выполняется поиск в "Н\_ЛЮДИ" по индексу "ЧЛВК\_РК" с кешированием (Memoize), где проверяется условие "ИД" = "Н\_СЕССИЯ"."ЧЛВК\_ИД" и фильтр "ИМЯ" = 'Ярослав'.

Кеш срабатывает 3285 раз, а 175 раз происходит обращение к индексу, но ни одна строка не проходит фильтр.

Затем для каждой из 16 строк из первого соединения выполняется Index Scan по индексу "ВЕД\_ЧЛВК\_FK\_IFK" таблицы "Н\_ВЕДОМОСТИ" с условием "ЧЛВК\_ИД" = "Н\_СЕССИЯ"."ЧЛВК\_ИД" и фильтром "ИД" > 1250972, но также не находится подходящих строк.

Общее время выполнения запроса составляет 2.535 мс, время планирования — 1.126 мс.



## Вывод:

Во время выполнения данной лабораторной работы я разобрался, что такое индексы, научился их писать и применять. Понял, как можно и нужно оптимизировать sql-запросы.