МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное автономное образовательное учреждение высшего образования “Национальный исследовательский университет ИТМО”

**ФАКУЛЬТЕТ ПРОГРАММНОЙ ИНЖЕНЕРИИ И КОМПЬЮТЕРНОЙ ТЕХНИКИ**

**ЛАБОРАТОРНАЯ РАБОТА №4**

по дисциплине

“Информационные системы и базы данных”

*Вариант*: 335110.

выполнил:

**Иванов Матвей Сергеевич**

**группа P33111**

Преподаватель

**Шешуков Дмитрий Михайлович**

**г. Санкт-Петербург, 2023**

# 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).

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

Н\_ТИПЫ\_ВЕДОМОСТЕЙ, Н\_ВЕДОМОСТИ.

Вывести атрибуты: Н\_ТИПЫ\_ВЕДОМОСТЕЙ.ИД, Н\_ВЕДОМОСТИ.ЧЛВК\_ИД.

Фильтры (AND):

a) Н\_ТИПЫ\_ВЕДОМОСТЕЙ.ИД > 2.

b) Н\_ВЕДОМОСТИ.ДАТА = 1998-01-05.

c) Н\_ВЕДОМОСТИ.ДАТА > 2010-06-18.

Вид соединения: INNER JOIN.

1. 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):

a) Н\_ЛЮДИ.ИД > 100865.

b) Н\_ВЕДОМОСТИ.ЧЛВК\_ИД < 163249.

Вид соединения: INNER JOIN.

# Ход работы

**Запрос 1.**

**SELECT**

Н\_ВЕДОМОСТИ.ЧЛВК\_ИД,

Н\_ТИПЫ\_ВЕДОМОСТЕЙ.ИД

**FROM** Н\_ТИПЫ\_ВЕДОМОСТЕЙ

**INNER** **JOIN** Н\_ВЕДОМОСТИ **ON** Н\_ТИПЫ\_ВЕДОМОСТЕЙ.ИД = Н\_ВЕДОМОСТИ.ТВ\_ИД

**WHERE** Н\_ТИПЫ\_ВЕДОМОСТЕЙ.ИД > 2

**AND** Н\_ВЕДОМОСТИ.ДАТА > TO\_DATE('2010-06-18', 'YYYY-MM-DD');

**Индексы:**

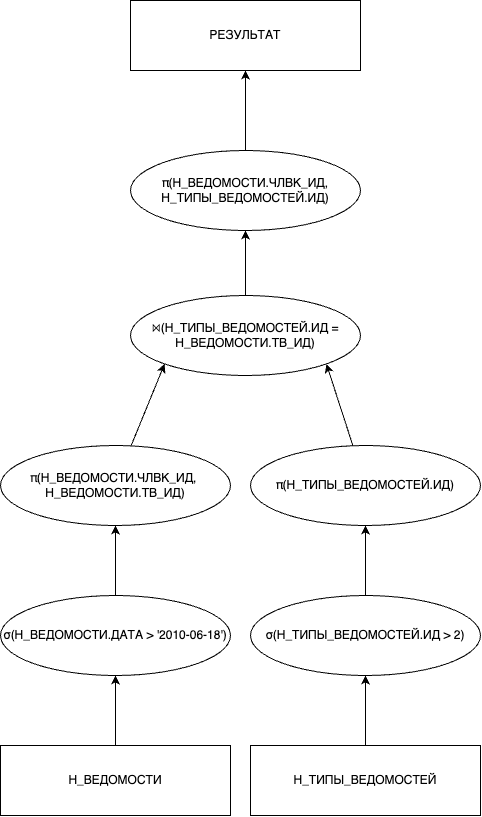
**CREATE** **INDEX** ИНД\_ВЕДОМОСТИ\_ДАТА

**ON** Н\_ВЕДОМОСТИ **USING** BTREE("ДАТА");

Я решил, что будет рационально добавить b-tree индекс на колонку даты, так как в запросе проходит агрегация по этой колонке, выбирая все значения большие заданного значения даты. B-tree индекс поможет эффективно выбирать записи основываясь на их положении в дереве.

Для поля ИД таблицы Н\_ТИПЫ\_ВЕДОМОСТЕЙ я решил не добавлять индексы, так как Н\_ТИПЫ\_ВЕДОМОСТЕЙ это характеристическая таблица и в ней мало записей. Так что использования индексов для этой таблицы будет не эффективно, проще просто пройтись по таблице.

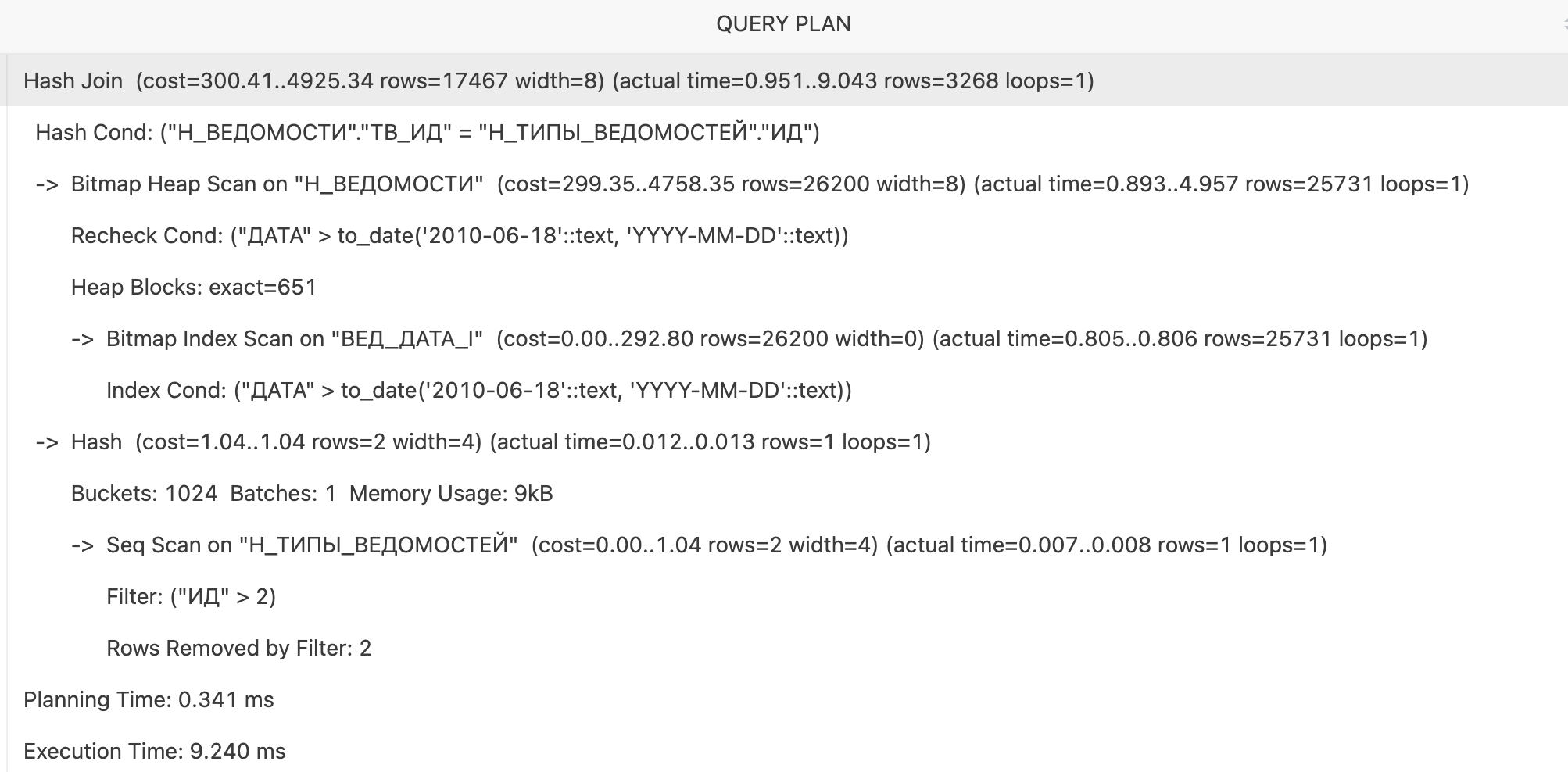
**Планы выполнения:**

1. 
2. 
3. 
4. 

По-моему мнению, самым оптимальным планом для выполнения запроса будет план №3, так как там мы сразу отбираем только нужные строки из каждой сущности, выбираем необходимые поля и только после этого соединяем наши таблицы. Таким образм, для дольно сложной операции соединяния таблиц у нас будет минимальное количество данных и оно будет максимально эффективным.

С добавленнием индексов этот план станет ещё быстрее и останется всё также самым эффективным.

**EXPLAIN ANALYZE**

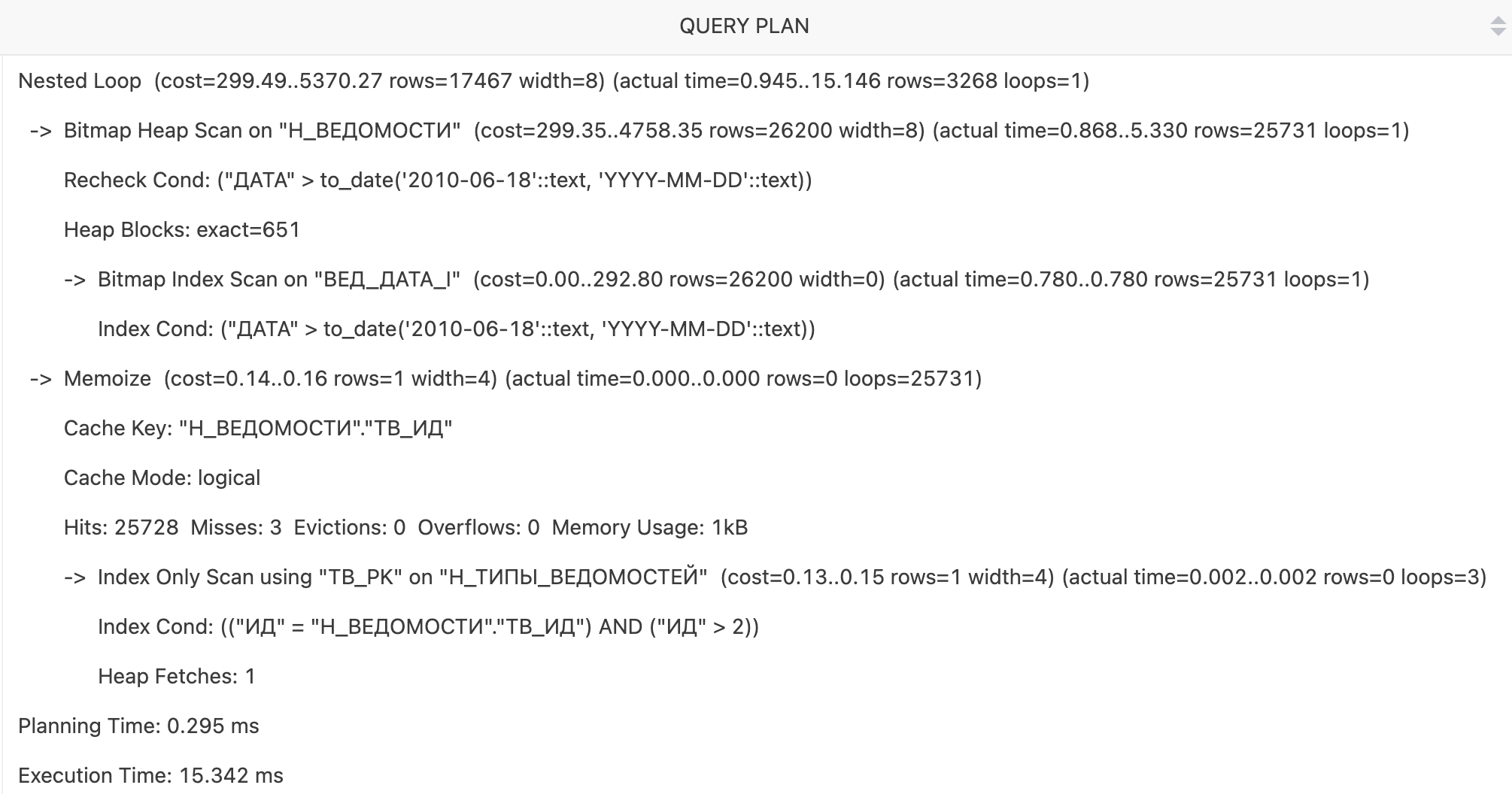


Как видно, планировщик использовал Hash Join что вполне логично, так как у нас характеристическая таблица Н\_ТИПЫ\_ВЕДОМОСТЕЙ и мы можем быстро создать хэш-таблицу для всех значений ключа оттуда и затем просто пройтись по нашей основной таблице Н\_ВЕДОМОСТИ и быстро соединить каждую строку с нужной характеристикой.

Далее мы видим, что для агрегации дат планировщик использует Bitmap Heap Scan на Н\_ВЕДОМОСТИ, чтобы выбирать большие батчи данных по одному индексу. Что логично для данной задачи, где нам нужно выбрать большое количество информации за раз. Всё это заняло у него 1 вложенный цекл соединения, а количество блоков по датам составило 641.

Для выбоки по Н\_ТИПЫ\_ВЕДОМОСТЕЙ он использовал Hash индекс, так как таблица маленькая и характеристическая и отбираем стоки мы по её первичному ключу. А ещё эти хэши дальше очень удобно используются в Hash Join.

Если же отключить использование Hash Join и проверить, как с этим справится Nested Loop то получаем, что Execution Time вырос почти в 2 раза:



**Запрос 2.**

**SELECT**

Н\_ЛЮДИ.ОТЧЕСТВО,

Н\_ВЕДОМОСТИ.ЧЛВК\_ИД,

Н\_СЕССИЯ.ДАТА

**FROM** Н\_ЛЮДИ

**INNER** **JOIN** Н\_ВЕДОМОСТИ **ON** Н\_ЛЮДИ.ИД = Н\_ВЕДОМОСТИ.ЧЛВК\_ИД

**INNER** **JOIN** Н\_СЕССИЯ **ON** Н\_ЛЮДИ.ИД = Н\_СЕССИЯ.ЧЛВК\_ИД

**WHERE** Н\_ЛЮДИ.ИД > **100865**

**AND** Н\_ВЕДОМОСТИ.ЧЛВК\_ИД < **163249**;

**Индексы:**

Для начала проведём небльшой анализ и посмотрим на результаты следующих команд:

**SELECT** **COUNT**(ИД) **FROM** Н\_ЛЮДИ;

# count = 5118

**SELECT** **COUNT**(**DISTINCT** ЧЛВК\_ИД) **FROM** Н\_ВЕДОМОСТИ;

# count = 3475

**SELECT** **COUNT**(**DISTINCT** ЧЛВК\_ИД) **FROM** Н\_СЕССИЯ;

# count = 180

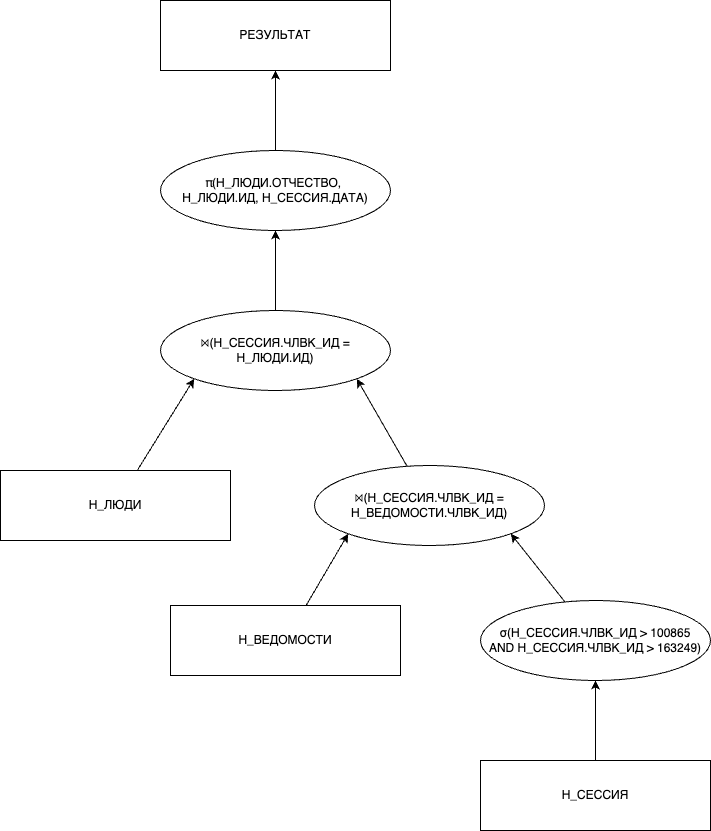
Как видно, в Н\_СЕССИЯ намного меньше уникальных занчений, нежели в Н\_ЛЮДИ и Н\_ВЕДОМОСТИ. А так как в данном контексте всё это одинаковые по смыслу поля в нашем запросе, то составим следующий индекс:

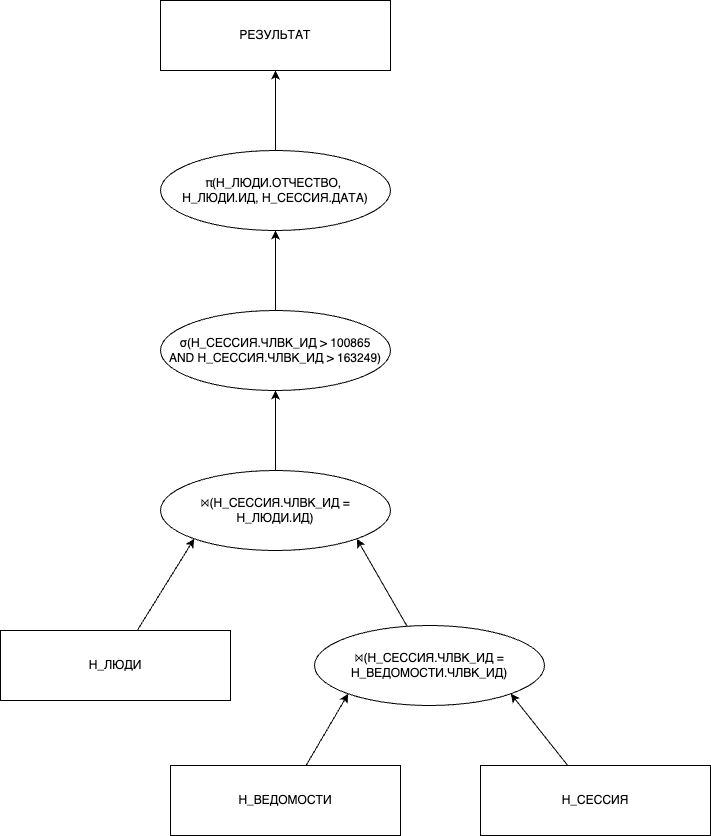
**CREATE** **INDEX** ИНД\_СЕССИЯ\_ЧЛВК\_ИД

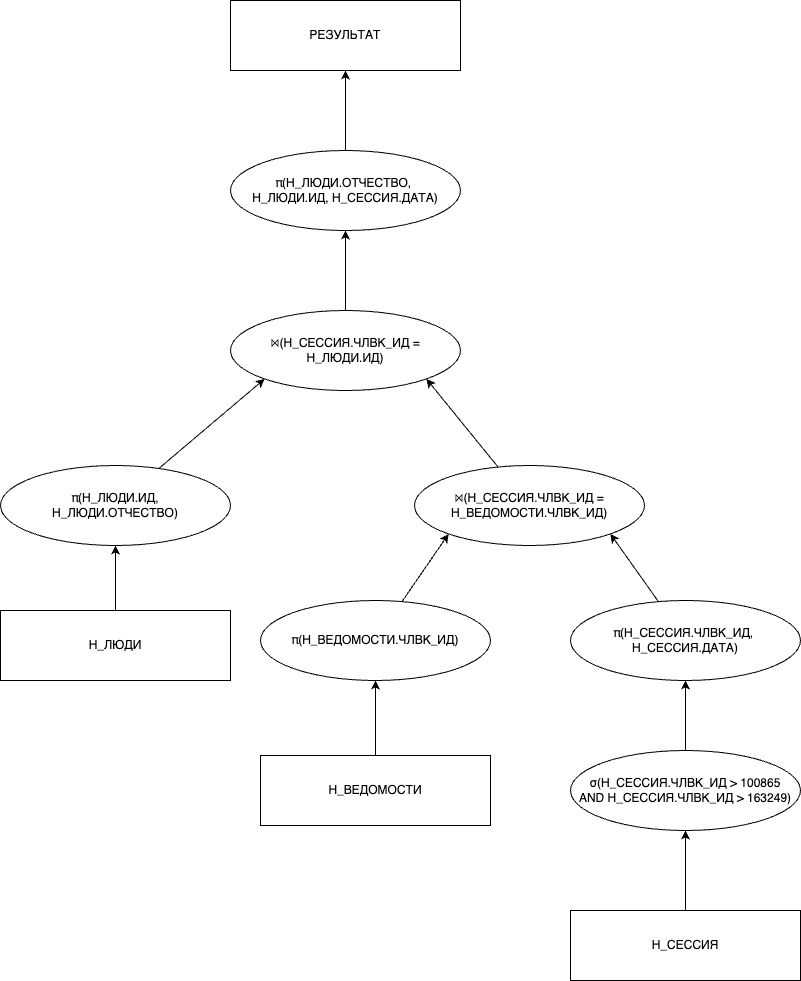
**ON** Н\_СЕССИЯ **USING** BTREE("ЧЛВК\_ИД");

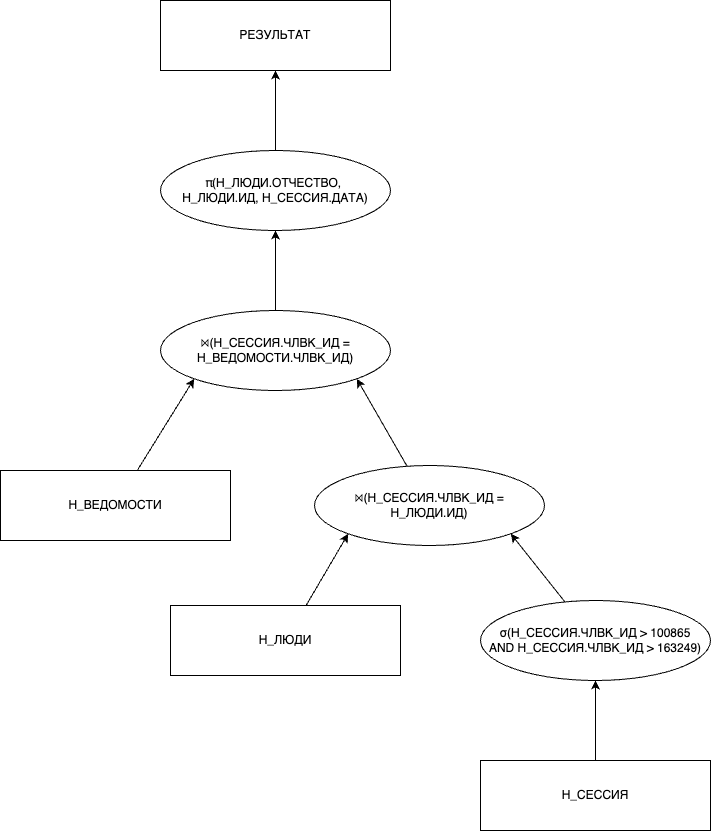
BTREE индекс на Н\_СЕССИЯ.ЧЛВК\_ИД, так как у нас происходит агрегация строк только по этому индексу и это будет максимально оптимально по памяти и скорости работы, так как Н\_СЕССИЯ.ЧЛВК\_ИД содержит меньше всего уникальных записей;

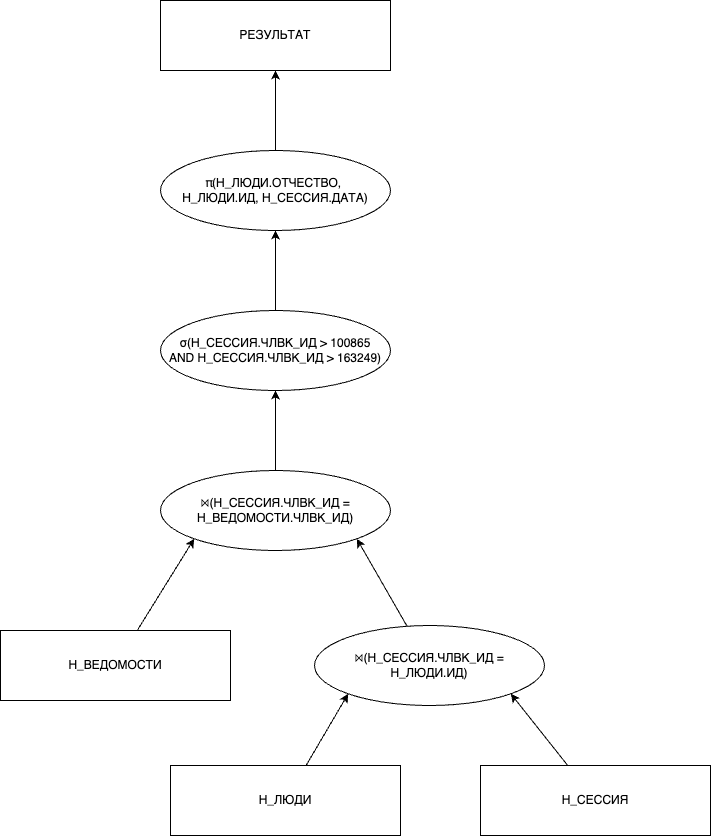
**Планы выполнения:**

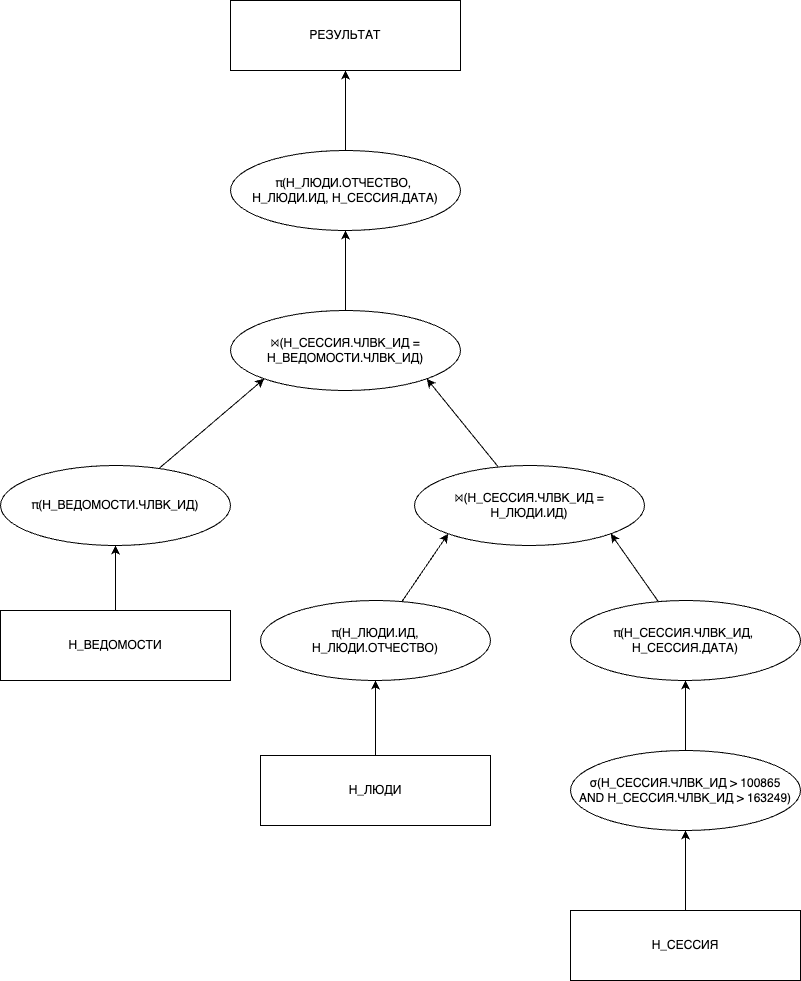
1. 

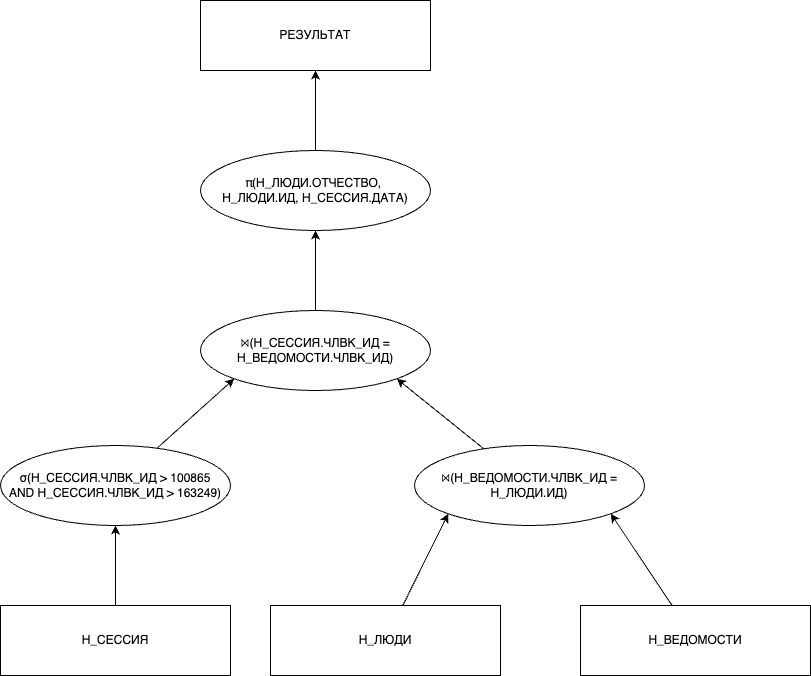
2. 

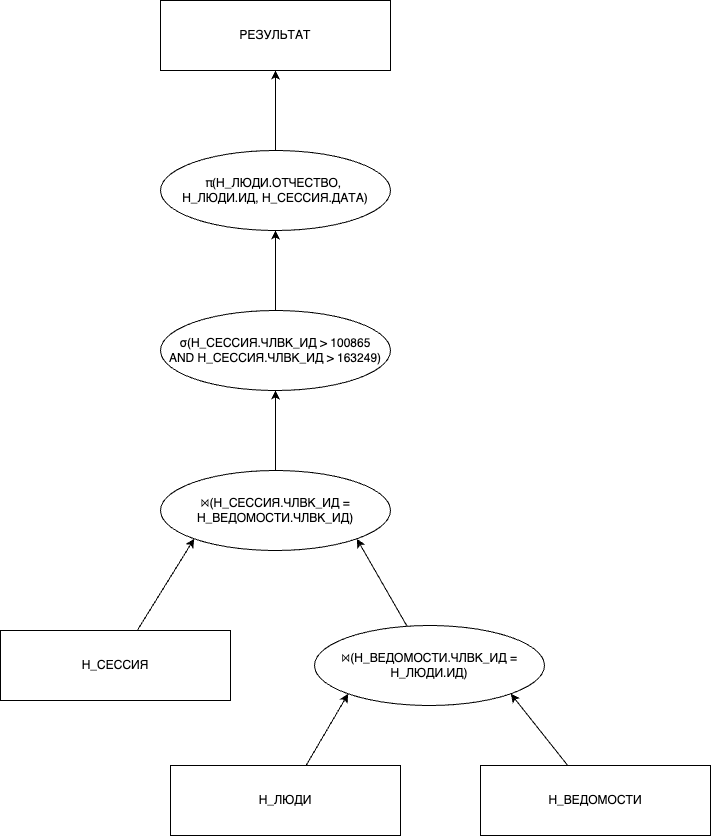
3. 

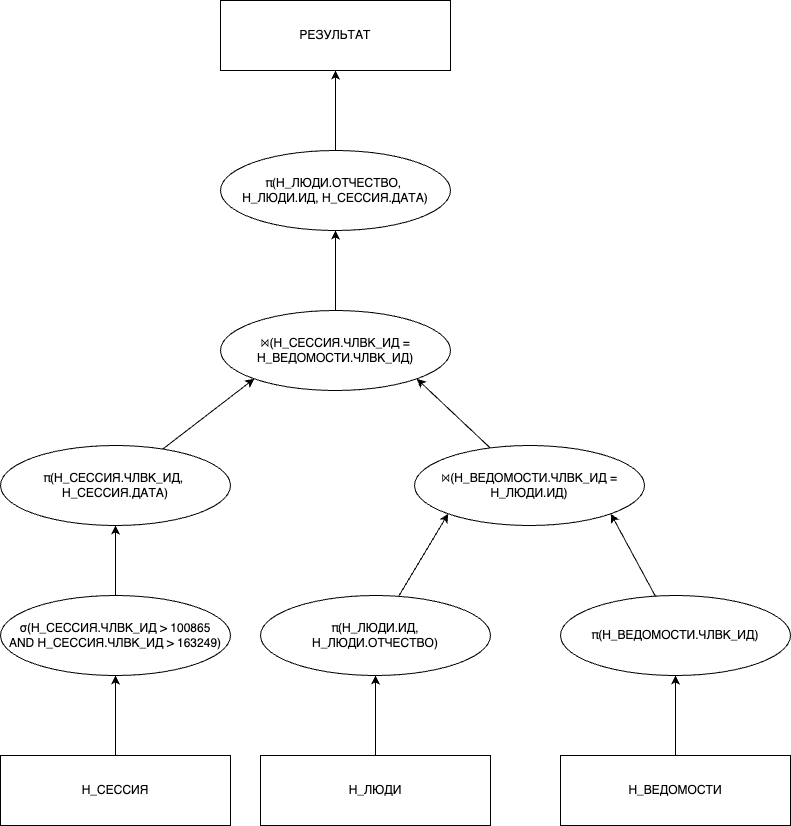
4. 

5. 

6. 

7. 

8.

9.

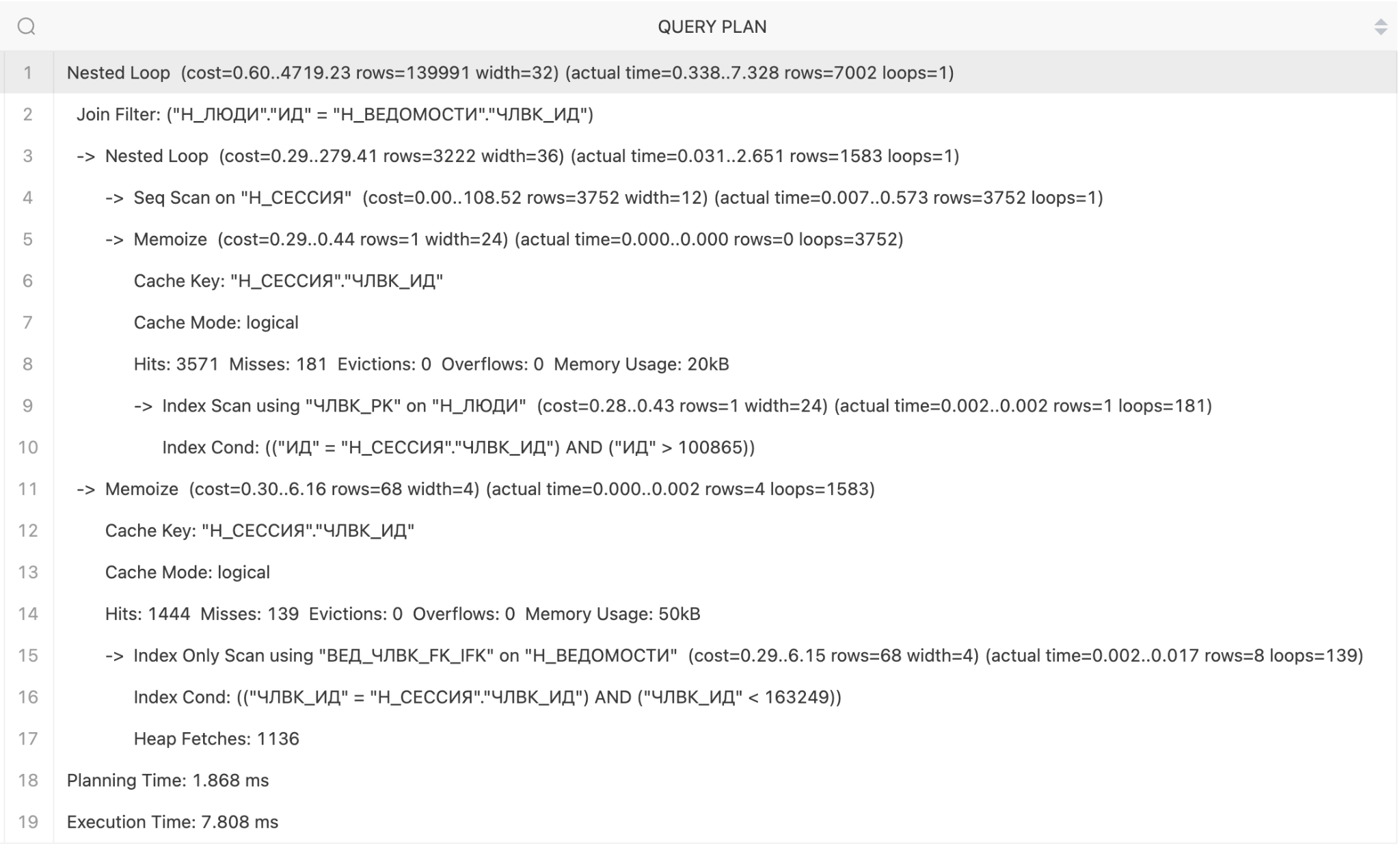
Во первых заметим, что во всех планах отбор строк проходил только по Н\_СЕССИЯ.ЧЛВК\_ИД так как у нас на нём висит индекс и в прошлом пункте подробно разъяснено, почему выгоднее использовать именно его. Если же рассмотреть подробно планы, то можно сразу понять, что так как отбор строк происходит по Н\_СЕССИЯ, а также в этой таблице меньше всего строк, то, очевидно, первый JOIN тоже должен быть её с другой таблицей, а значит планы 7-9 не подходят. Дальше, мы знаем, что таблица Н\_ВЕДОМОСТИ содержит гораздо меньше строк, нежели Н\_ЛЮДИ, так что разумнее будет объединять Н\_СЕССИЯ с Н\_ВЕДОМОСТИ, а только затем с Н\_ЛЮДИ. Отсюда сокращаем рассматриваемые планы до 1 - 3.

В этих же планах мы сразу можем откинуть план 2, так как там отбор строк происходит после всех JOIN операций, но логичнее всего отбирать строки раньше всего (чтобы объединяемая таблица была меньше).

Если же рассматривать планы 1 и 3, то тут эффективнее всё же будет план №3, вель там мы сразу выбираем из каждой таблицы только необходимые поля, тем самым сокращая время на объединение таблиц, ведь нам не нужно тащить очень много лишней информации через все JOIN операции, как в плане №1.

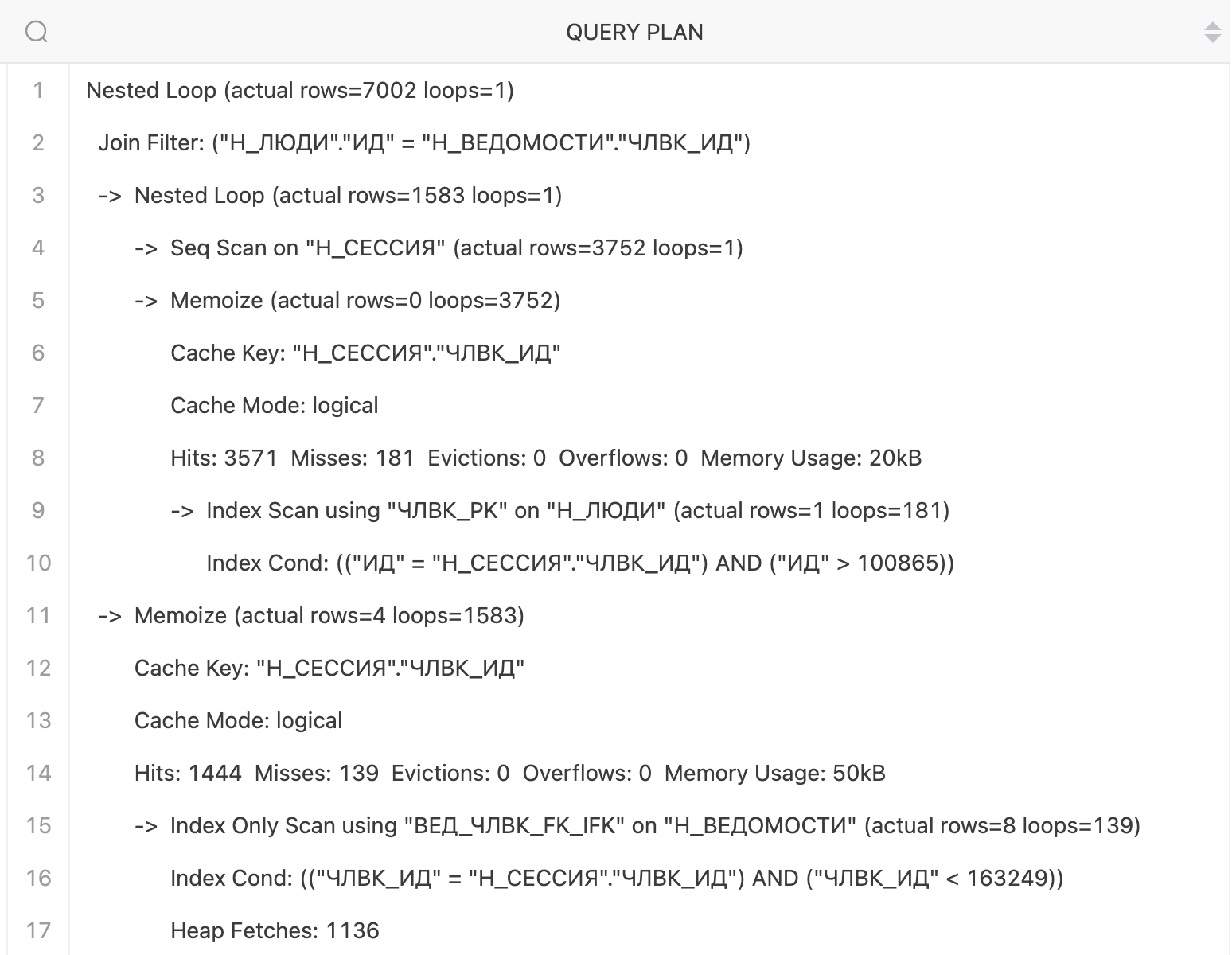
Значит получаем, что самый эффективный план будет №3.

**EXPLAIN ANALYZE**



Как можно заметить, для операции объединения таблиц оба раза использовался Nested Loop. Планировщик в первую очередь решил объединять таблицы Н\_СЕССИЯ и Н\_ЛЮДИ, при том он использовать Memorize на ключ Н\_СЕССИЯ.ЧЛВК\_ИД, чтобы закэшировать наиболее используемые значения таблицы Н\_СЕССИЯ, а так как уникальны значений ключа там не много, а после агрегации стало явно ещё меньше, вероятнее всего вся она смогла влезть в кэш. А понадобилось ему это чтобы при следующем объединении с Н\_ВЕДОМОСТИ мы использовали эти закэшированные знания ключа и быстро могли объединить наши таблицы. Чтобы проверить, как хорошо сработал Memorize для нашей задачи давайте посмотрим на вывод команды:

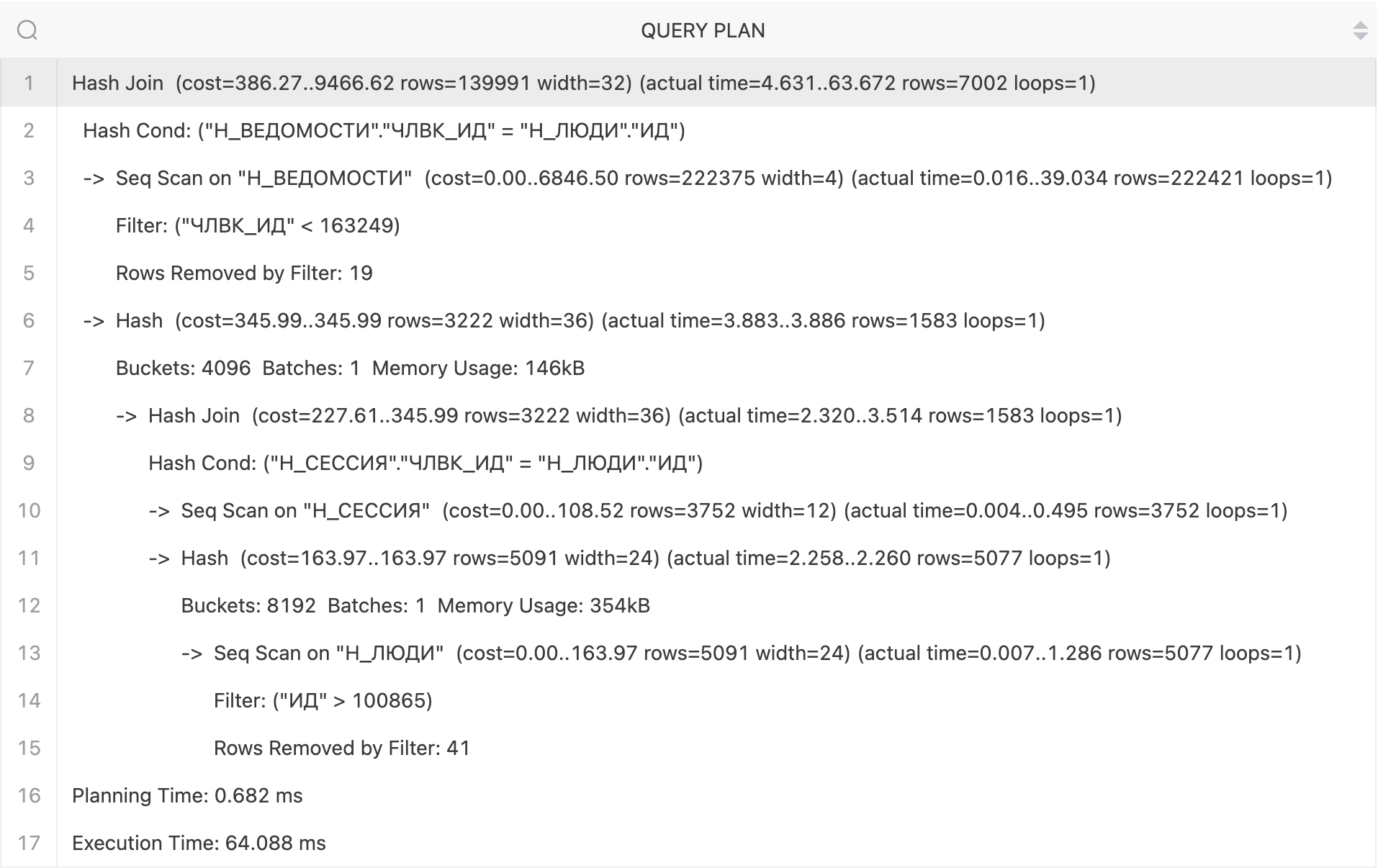
**EXPLAIN** (**analyze**, costs **off**, timing **off**, summary **off**) <QUERY>



Здесь мы можем наблюдать дополнительные параметры (Hits, Misses, Evictions, Overflows, Memory Usage).

Evictions — количество вытеснений из кеша, Overflows — количество переполнений памяти с невозможностью сохранить все строки, относящиеся к одному набору параметров. В нашем запросе они равны 0, а значит Memorize работает эффективно для этого запроса. Ну и также заметно, что Misses намного меньше Hits, что значит что мы чаще всего попадаем при поиске в кэше.

Ну и немного про использование Nested Loop здесь, а не HashJoin, это вполне понятно, так как у нас слишком много различных значений ИД/ЧЛВК\_ИД и хэширование только значительно увеличит памят и ухудшит производительность, так как таблицы будут слишком большие и не удобные. Можем проверить это, отключив использование Nested Loop:



Как видно, время выполнения запроса увеличилось в 8 раз.

**Заключение**

Интересная работа, для изучения того, как внутри устроен планировщик и в целом какие есть стратегии объедиения, создания индексов и агрегаций. Так что за исключением рисования десятков палнов это было весьма весьма познавательно.