Национальный Исследовательский Университет ИТМО
Кафедра ВТ

Лабораторная работа №3
Информационные Системы
и
Базы Данных

Преподаватель: Николаев Владимир Вячеславович
Выполнил: Федоров Сергей
Группа: P33113

Санкт-Петербург
2020 г.

Данный текст задания

Вариант: 254178

Составить запросы на языке **SQL**.

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

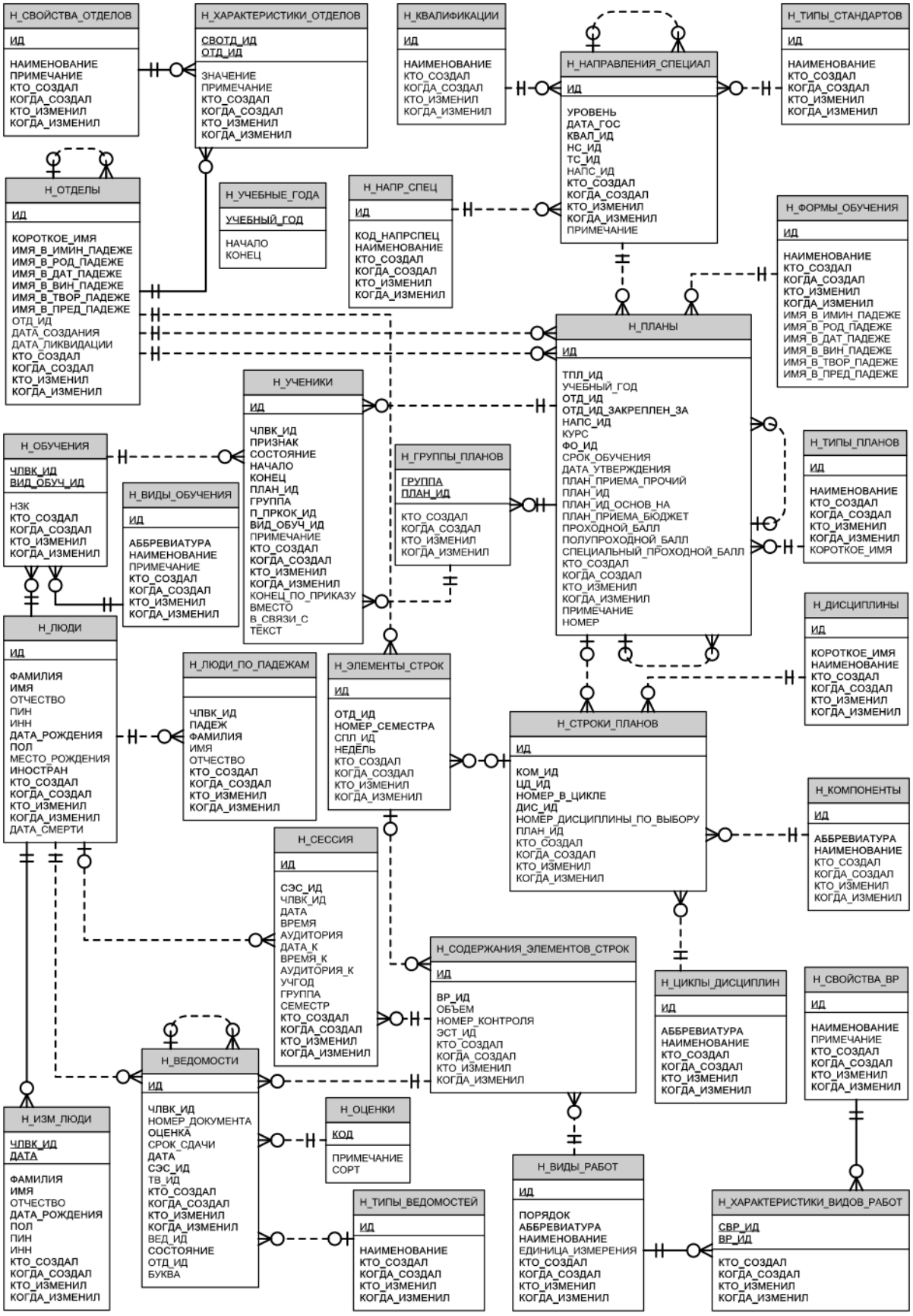
Для запросов `1`-`2` необходимо составить возможные планы выполнения за просов. Планы составляются на основании предположения, что в таблица х отсутствуют индексы. Из составленных планов необходимо выбрать опти мальный и объяснить свой выбор. Изменятся ли планы при добавлении ин декса и как?

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

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

1. Сделать запрос для получения атрибутов из указанных таблиц, применив фильтры по указанным условиям:
   Таблицы: `Н_ОЦЕНКИ`, `Н_ВЕДОМОСТИ`. Вывести атрибуты: `Н_ОЦЕНКИ.ПРИМЕЧАНИЕ`, `Н_ВЕДОМОСТИ.ИД`. Фильтры **AND**:
   a) `Н_ОЦЕНКИ.ПРИМЕЧАНИЕ` = **неудовлетворительно**.
   b) `Н_ВЕДОМОСТИ.ЧЛВК_ИД` = **117219**. Вид соединения: **INNER 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.ФАМИЛИЯ`, `Н_ВЕДОМОСТИ.ДАТА`, `Н_СЕССИЯ.ДАТА`. Фильтры **AND**: a) `Н_ЛЮДИ.ОТЧЕСТВО` = **Владимирович**.
   b) `Н_ВЕДОМОСТИ.ЧЛВК_ИД` < **163249**. c) `Н_СЕССИЯ.ЧЛВК_ИД` = **126631**. Вид соединения: **RIGHT JOIN**.

ER-диаграмма базы данных



Выполнение

DML

Запрос 1

```
SELECT "Н_ОЦЕНКИ"."ПРИМЕЧАНИЕ", "Н_ВЕДОМОСТИ"."ИД" FROM "Н_ОЦЕНКИ"
INNER JOIN "Н_ВЕДОМОСТИ" ON "ОЦЕНКА" = "КОД"
WHERE
      "Н_ОЦЕНКИ"."ПРИМЕЧАНИЕ" = 'неудовлетворительно' AND
      "Н_ВЕДОМОСТИ"."ЧЛВК_ИД" = 117219;
```

<empty>

Запрос 2

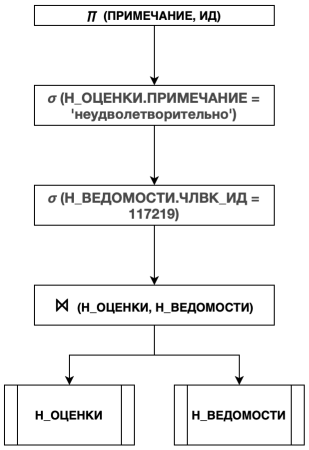
```
SELECT "Н_ЛЮДИ"."ФАМИЛИЯ", "Н_ВЕДОМОСТИ"."ДАТА", "Н_СЕССИЯ"."ДАТА"
FROM "Н_ЛЮДИ"
RIGHT JOIN "Н_ВЕДОМОСТИ" on "Н_ЛЮДИ"."ИД" = "Н_ВЕДОМОСТИ"."ЧЛВК_ИД"
RIGHT JOIN "Н_СЕССИЯ" on "Н_ЛЮДИ"."ИД" = "Н_СЕССИЯ"."ЧЛВК_ИД"
WHERE
        "Н_ЛЮДИ"."ОТЧЕСТВО" = 'Владимирович' AND
        "Н_ВЕДОМОСТИ"."ЧЛВК_ИД" < 163249 AND
        "Н_СЕССИЯ"."ЧЛВК_ИД" = 126631;
```

<empty>

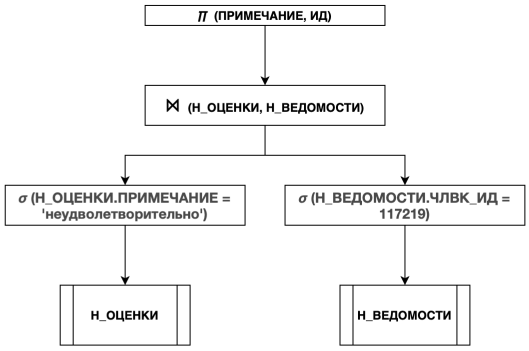
Возможные планы исполнения запросов

Запрос 1

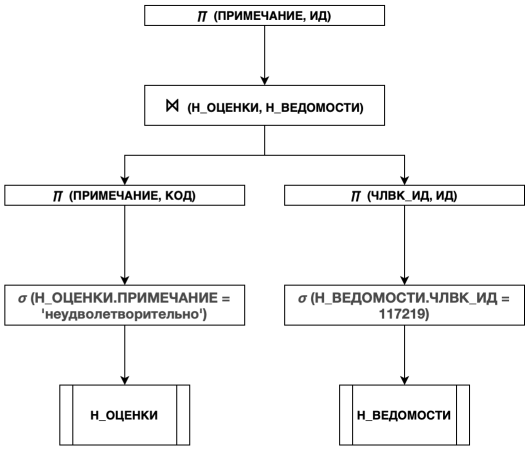
1



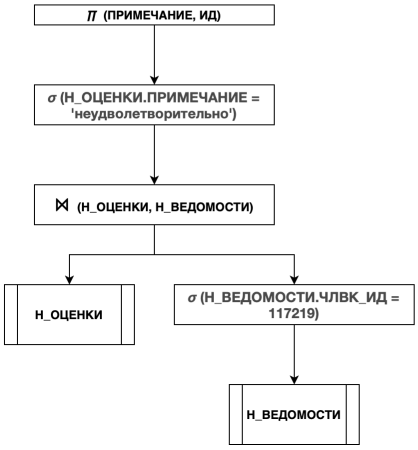
2



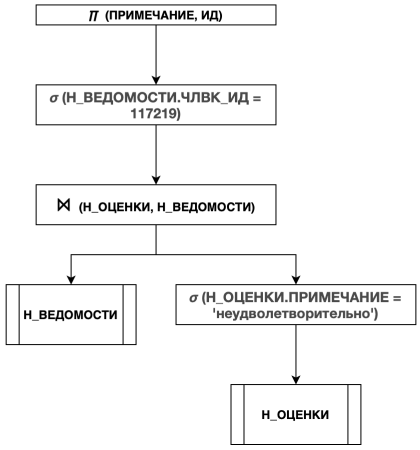
5_{extra}



3



4



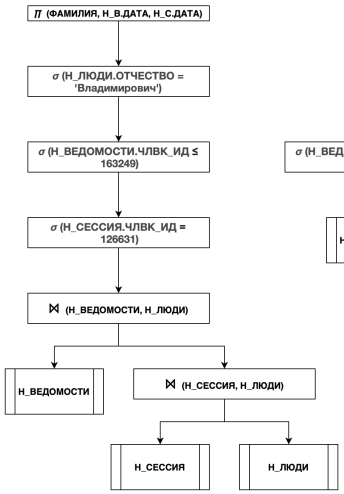
Пояснение:

Лучшим из предоставленных вариантов будет **второй** вариант исполнения команды. В теории если бы после фильтров, по ПРИМЕЧАНИЕ и ЧЛВК_ИД у нас происходила материализация, тогда можно было бы применить еще проекции до JOIN’а, чтобы уменьшить кол-во хранимой информации о промежуточных данных, однако это не так.

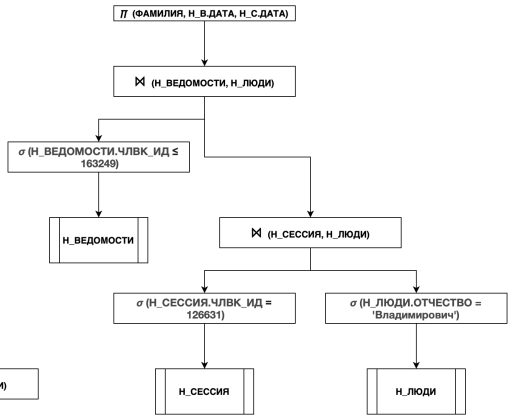
Лучшим **второй** вариант является потому, что мы выполняем фильтрацию строк до начала Nested Loop Join’а, а значит количество итераций на данном этапе будет значительно меньше чем если выполнять запрос в обратном порядке (join -> filter).

Запрос 2

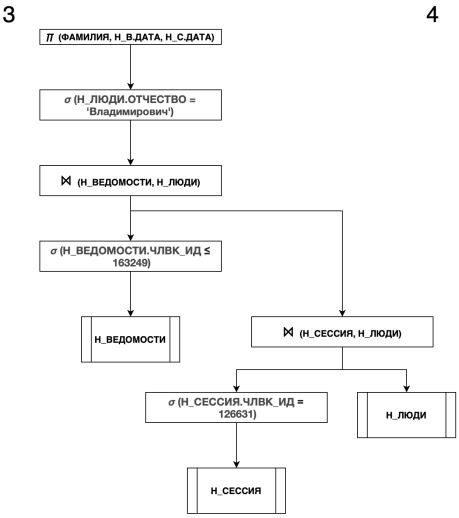
1



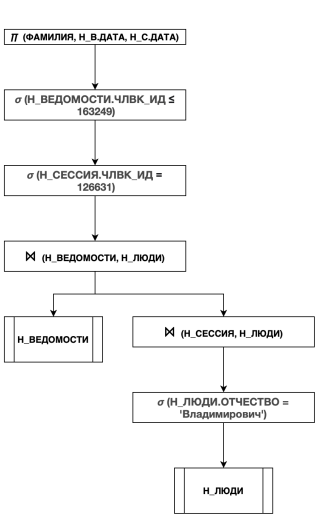
2



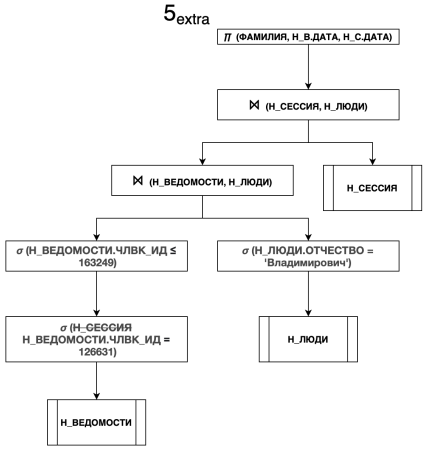
3



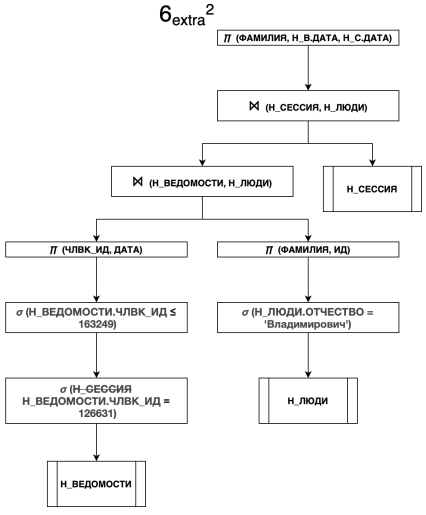
4



5_{extra}



$6_{extra}{}^{2}$



Пояснение:
Лучшим из предоставленных вариантов будет также **второй** вариант исполнения запроса, так как мы опять фильтруем данные из каждой таблицы перед последующим join'ом ее с другой таблицей.

Как еще одним вариантом оптимизации (**пятый** вариант), чисто из тела запроса можно вычленить тот факт что Н_ВЕДОМОСТИ.ЧЛВК_ИД и Н_СЕССИЯ.ЧЛВК_ИД это суть есть одно и тоже (по крайней мере так он требуется по условиям join'a). В таком случае, можно просто взять и устроить фильтрацию сразу по таблице Н_ВЕДОМОСТИ или Н_СЕССИЯ по двум критериям вместо одного, тем самым построив левостороннее дерево исполнения, что позволяет нам использовать конвейерную обработку. Данная оптимизация может помочь с приростом в скорости выполнения в том случае, если например одна таблица сильно больше другой (например Н_ВЕДОМОСТИ >> Н_СЕССИЯ).

Также можно вспомнить еще о проекциях. Так как после первого (в хронологическом порядке) join'a у нас появиться надобность в материализации таблиц, то возможно имеет смысл сделать проекцию на таблицы, чтобы уменьшить размер промежуточного материализованного представления (**шестой** вариант).

P.S. Здесь на предоставленных вариантах данного варианта нету. Но так как **второй** вариант будет самым эффективным, и там будет материализация "посередине", то в такой план исполнения тоже стоит вставить промежуточные проекции.

Разбор предоставленных Postgres планов [`EXPLAIN`]

Запрос 1

```
EXPLAIN ANALYSE SELECT "Н_ОЦЕНКИ"."ПРИМЕЧАНИЕ", "Н_ВЕДОМОСТИ"."ИД" FROM "Н_ОЦЕНКИ"
INNER JOIN "Н_ВЕДОМОСТИ" ON "ОЦЕНКА" = "КОД"
WHERE
      "Н_ОЦЕНКИ"."ПРИМЕЧАНИЕ" = 'неудовлетворительно' AND
      "Н_ВЕДОМОСТИ"."ЧЛВК_ИД" = 117219;
                                                                 QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------
 Nested Loop  (cost=0.42..198.01 rows=7 width=422) (actual time=0.579..0.579 rows=0 loops=1)
   Join Filter: (("Н_ОЦЕНКИ"."КОД")::text = ("Н_ВЕДОМОСТИ"."ОЦЕНКА")::text)
   Rows Removed by Join Filter: 31
   ->  Seq Scan on "Н_ОЦЕНКИ"  (cost=0.00..1.11 rows=1 width=452) (actual time=0.055..0.055 rows=1 loops=1)
         Filter: (("ПРИМЕЧАНИЕ")::text = 'неудовлетворительно'::text)
         Rows Removed by Filter: 8
   ->  Index Scan using "ВЕД_ЧЛВК_FK_IFK" on "Н_ВЕДОМОСТИ"  (cost=0.42..196.10 rows=64 width=10) (actual time=0.139..0.502 rows=31 loops=1)
         Index Cond: ("ЧЛВК_ИД" = 117219)
 Planning time: 1.547 ms
 Execution time: 0.655 ms
(10 rows)
```

Пояснение:
Выбранный Postgres план выполнения совпадает с предложенным мною **вторым** вариантом.
Теперь более детально:

- Nested Loop - конкретные причины почему используется этот метод join'a мне неизвестны, но скорее всего дело в том что в нашем случае таблица Н_ОЦЕНКИ очень мала (~~а после фильтрации и того станет размером 1~~), а значит выполнения данного метода займет примерно O(N), где N - размер таблицы Н_ВЕДОМОСТИ. Тем более, преимущество также заключается и в том для NLJ, не нужна никакая предварительная подготовка.

- Seq Scan Н_ОЦЕНКИ - опять таки из-за слишком малого размера таблицы, неразумно использовать какие-либо index scan'ы. К тому же индекса на "ПРИМЕЧАНИЕ" у нас не создано.
- Index Scan using "ВЕД_ЧЛВК_FK_IFK" Н_ВЕДОМОСТИ - используется btree индекс на Н_ВЕДОМОСТИ, для эквисравнения. Используется именно этот тип сканирования, потому что ЧЛВК_ИД - колонка с уникальными значениями, а значит не стоит применять bitmap сканирование.

При частом сравнении на эквивалентность стоит применить HASH заместо B_TREE для таблицы Н_ВЕДОМОСТИ (`O(1) <= O(log(N)`). Однако оптимизирующий индекс уже присутствует и смысла добавлять новый особо нету.

Запрос 2

```
ucheb=> EXPLAIN ANALYSE SELECT "Н_ЛЮДИ"."ФАМИЛИЯ", "Н_ВЕДОМОСТИ"."ДАТА", "Н_СЕССИЯ"."ДАТА" FROM "Н_ЛЮДИ"
ucheb-> RIGHT JOIN "Н_ВЕДОМОСТИ" on "Н_ЛЮДИ"."ИД" = "Н_ВЕДОМОСТИ"."ЧЛВК_ИД"
ucheb-> RIGHT JOIN "Н_СЕССИЯ" on "Н_ЛЮДИ"."ИД" = "Н_СЕССИЯ"."ЧЛВК_ИД"
ucheb-> WHERE
ucheb->         "Н_ЛЮДИ"."ОТЧЕСТВО" = 'Владимирович' AND
ucheb->         "Н_ВЕДОМОСТИ"."ЧЛВК_ИД" < 163249 AND
ucheb->         "Н_СЕССИЯ"."ЧЛВК_ИД" = 126631;
                                                                QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------
 Nested Loop  (cost=5.04..236.08 rows=448 width=32) (actual time=0.078..0.078 rows=0 loops=1)
   ->  Index Scan using "ВЕД_ЧЛВК_FK_IFK" on "Н_ВЕДОМОСТИ"  (cost=0.42..196.26 rows=64 width=12) (actual time=0.077..0.077 rows=0 loops=1)
         Index Cond: (("ЧЛВК_ИД" < 163249) AND ("ЧЛВК_ИД" = 126631))
   ->  Materialize  (cost=4.62..34.24 rows=7 width=28) (never executed)
         ->  Nested Loop  (cost=4.62..34.20 rows=7 width=28) (never executed)
               ->  Index Scan using "ЧЛВК_PK" on "Н_ЛЮДИ"  (cost=0.28..8.30 rows=1 width=20) (never executed)
                     Index Cond: ("ИД" = 126631)
                     Filter: (("ОТЧЕСТВО")::text = 'Владимирович'::text)
               ->  Bitmap Heap Scan on "Н_СЕССИЯ"  (cost=4.33..25.83 rows=7 width=12) (never executed)
                     Recheck Cond: ("ЧЛВК_ИД" = 126631)
                     ->  Bitmap Index Scan on "SYS_C003500_IFK"  (cost=0.00..4.33 rows=7 width=0) (never executed)
                           Index Cond: ("ЧЛВК_ИД" = 126631)
 Planning time: 1.319 ms
 Execution time: 0.232 ms
(14 rows)
```

Пояснение:
Выбранный Postgres план соответствует моему **второму** варианту. Оптимизации предложенные мною в **5м** и **6м** вариантах не учитываются, возможно потому что Н_СЕССИЯ - не маленькая таблица и лучше уже по отдельности отфильтровать таблицы.

Теперь более детально:
- Nested Loop #1/Nested Loop #2 - об join'а у нас имеют фильтр на эквивалентность, однако видимо Postgres считает, что чтения из индексов и в принципе предварительная подготовка, могут лишь только ухудшить итоговую производительность, поэтому используем вложенные циклы.
- Index Scan using "ВЕД_ЧЛВК_FK_IFK" Н_ВЕДОМОСТИ - предоставленный b_tree индекс идеально подходить для данных фильтрующий операций сравнения и эквивалентности.
- Materialize - после первого хронологического join'а стоит сохранить промежуточный результат.
- Index Scan using "ЧЛВК_PK" Н_ЛЮДИ - идет фильтрация по ИД (пусть она даже явно и не указывается в запросе, Postgres сам вывел данную зависимость), затем там же идет фильтрация на эквивалентность строке. Зачем так? Легче отфильтровать большой обьем данных по косвенному, с легкой для выполнения точки зрения предикатом по ИД, чтобы потом уже использовать непосредственно указанный признак эквивалентности по ОТЧЕСТВО.

- Bitmap Heap/Index Scan using “SYS_C003500_IFK” Н_СЕССИЯ - данный тип сканирования применяется в промежуточном случае между Seq Scan и Index Scan. Seq Scan по объективным причинам неэффективен при выборке малого количества строк из таблицы. Index Scan неэффективен при выборке большого количества строк, так как будет большое количества случайных доступов к памяти, что в свою очередь в силу устройства обычных HDD дисков (да и SSD в том числа на самом деле) может привести к большому и неэффективному io_wait (пока устройство будет искать нужный блок). В промежуточной ситуации, когда предполагается что будет выбрано немалое, но и не большое кол-во строк, и при наличие индекса, применяется Bitmap Index Scan.

В случае данного запроса, при сравнении на эквивалентность атрибутов у таблиц Н_ЛЮДИ и Н_СЕССИЯ, вместо имеющихся индексов:

```
CREATE INDEX "ЧЛВК_ПАДЕЖ_FK_IFK" ON "Н_ЛЮДИ_ПО_ПАДЕЖАМ" USING btree ("ЧЛВК_ИД");
CREATE INDEX "SYS_C003500_IFK" ON "Н_СЕССИЯ" USING btree (“ЧЛВК_ИД");
```

Можно попытаться использовать Hash index на ЧЛВК_ИД. Однако, опять таки, прирост по скорости, если и будет, то скорее всего будет минимальным.

Вывод

Данная работа, показалась сложнее предыдущих, так как ее выполнение предполагает наличие хороших знаний о внутреннем устройстве базы данных (в особенности планировщика), а так же некоторого знания архитектуры вычислительных систем.

Сами по себе методы оптимизации выполнения запросов в Postgres, показались достаточно прямолинейными и простыми, однако если копнуть глубже, оказывается, что при применении оптимизации, результат может быть совершенно иным, а значит надо иметь большой опыт работы с SQL, а так же хорошо знать предметную область (селективность данных, отношение операций чтения к операциям записи или изменения, требования к данным операциям) для того чтобы вывести эффективный план выполнения запросов.