Федеральное государственное автономное образовательное учреждение высшего образования

Университет ИТМО

Дисциплина: Информационные системы и базы данных

Лабораторная работа 4

Вариант 4590

Выполнил:
Кривоносов Егор Дмитриевич

Группа: P33111

Преподаватель:
Николаев Владимир Вячеславович

2021 г.

Санкт-Петербург

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).

Ссылка на инфологическую модель БД "Учебный Процесс": *https://vk.cc/c7YK2E*

Выполнение

Задание 1

> Сделать запрос для получения атрибутов из указанных таблиц, применив фильтры по указанным условиям:
>
> Таблицы: Н_ОЦЕНКИ, Н_ВЕДОМОСТИ.
> Вывести атрибуты: Н_ОЦЕНКИ.КОД, Н_ВЕДОМОСТИ.ЧЛВК_ИД.
> Фильтры (AND):
> a) Н_ОЦЕНКИ.ПРИМЕЧАНИЕ = незачет.
> b) Н_ВЕДОМОСТИ.ДАТА > 1998-01-05.
> Вид соединения: LEFT JOIN.

Запрос

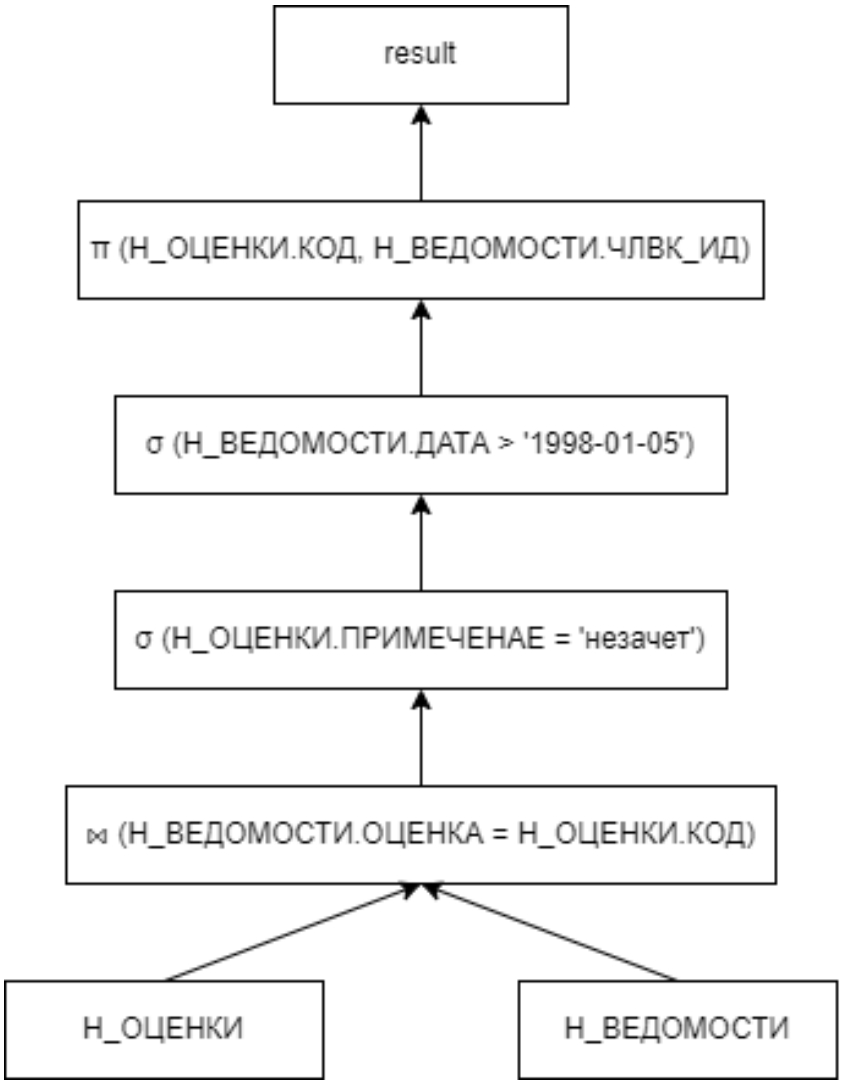
```
SELECT "Н_ОЦЕНКИ"."КОД", "Н_ВЕДОМОСТИ"."ЧЛВК_ИД" FROM "Н_ОЦЕНКИ"
   LEFT JOIN "Н_ВЕДОМОСТИ" ON "Н_ВЕДОМОСТИ"."ОЦЕНКА" = "Н_ОЦЕНКИ"."КОД"
WHERE "Н_ОЦЕНКИ"."ПРИМЕЧАНИЕ" = 'незачет' AND "Н_ВЕДОМОСТИ"."ДАТА" > '1998-01-05';
```

Индексы

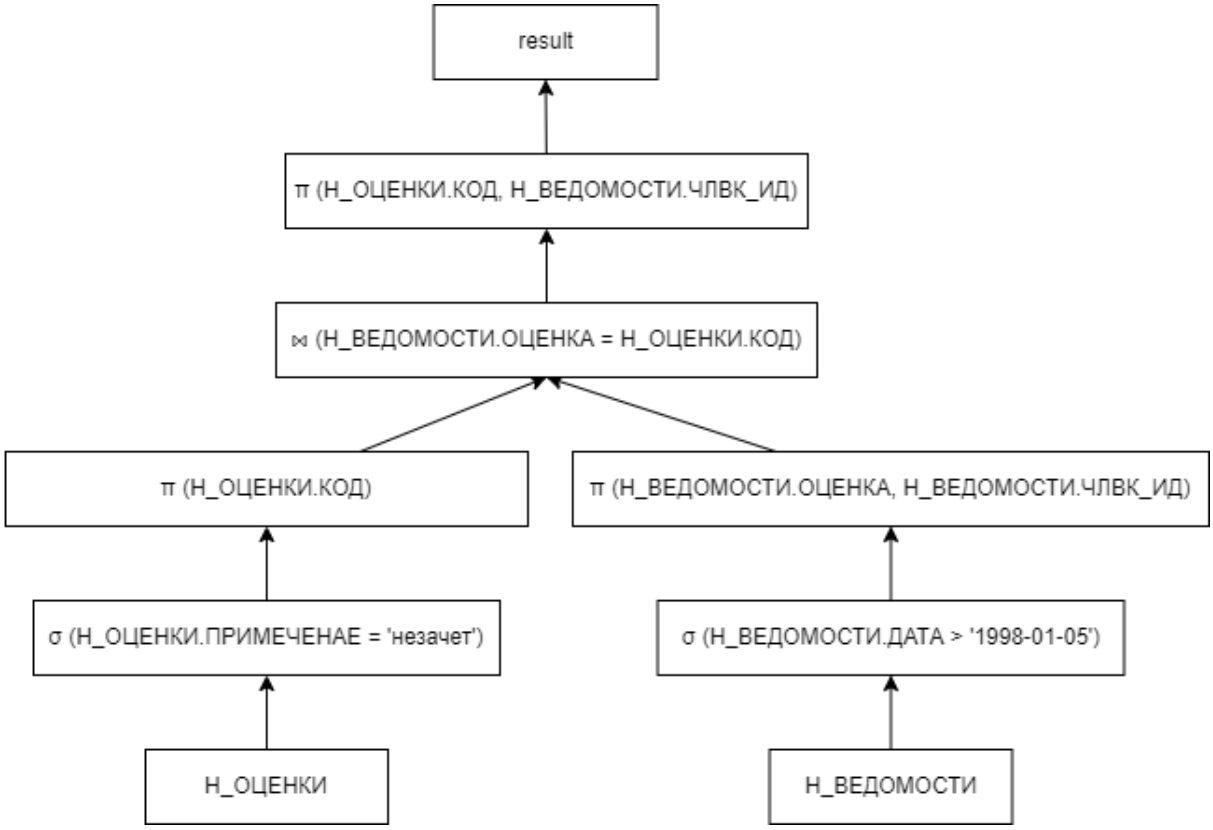
```
-- а) Индекс позволяет эффективно выбирать строки, основываясь на значении хеша столбца.
CREATE INDEX "ИНД_ОЦЕНКА" ON "Н_ВЕДОМОСТИ" USING HASH ("ОЦЕНКА");
-- б) Индекс позволяет эффективно выбирать строки, основываясь на их положении в дереве, а также так как выражения в WHERE отбираются с использованием знака >.
CREATE INDEX "ИНД_ДАТА" ON "Н_ВЕДОМОСТИ" USING BTREE ("ДАТА");
```

Возможные планы выполнения запроса

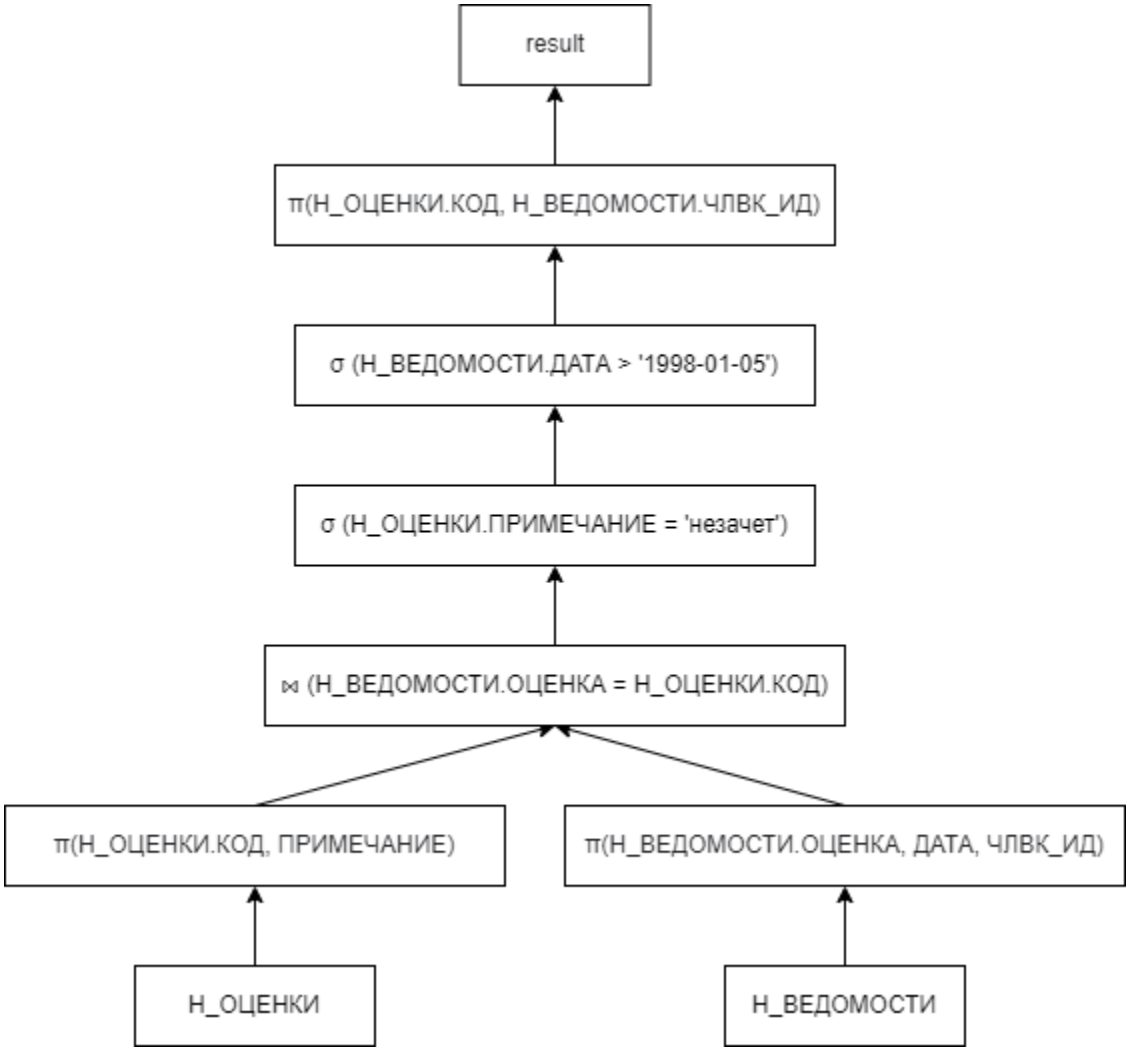
а)



б)


result
π (H_ОЦЕНКИ.КОД, H_ВЕДОМОСТИ.ЧЛВК_ИД)
⋈ (H_ВЕДОМОСТИ.ОЦЕНКА = H_ОЦЕНКИ.КОД)
π (H_ОЦЕНКИ.КОД)
π (H_ВЕДОМОСТИ.ОЦЕНКА, H_ВЕДОМОСТИ.ЧЛВК_ИД)
σ (H_ОЦЕНКИ.ПРИМЕЧЕНАЕ = 'незачет')
σ (H_ВЕДОМОСТИ.ДАТА > '1998-01-05')
H_ОЦЕНКИ
H_ВЕДОМОСТИ

в)



Оптимальным планом является план “б”, потому что сначала производится выбор строк по указанным условиям, а только потом отфильтрованные строки соединяются с помощью LEFT JOIN. Но из-за того, что ДАТА у нас “> 1998-01-05” в это условие попадает большая часть таблицы (217247 строк) мы тащим слишком много с собой данных. Чтобы уменьшить их количество лучше всего сделать сначала фильтрацию по ПРИМЕЧАНИЕ. Поэтому в данном случае менее эффективно использовать план “а” и “в”, когда изначально мы соединим таблицы, а после применяют фильтрацию. При ранней фильтрации и последующем соединении таблиц мы получим на выходе (2950 строк), что почти в 74 раза меньше чем в плане “а” и “в” на начальном этапе.

В запросе используется LEFT JOIN, значит результат запроса должен зависеть от того, какая таблица выбрана правой, а какая левой. Но так как для каждой строки из левой таблицы находится соответствие в правой таблице, результат запроса будет одинаковым вне зависимости от того, какая таблица находится справа, а какая слева. И будет аналогичен INNER JOIN.

При добавлении предложенных индексов, указанный план останется эффективнее первого. Ускорится выборка строк из таблицы Н_ВЕДОМОСТИ с помощью хеш-индекса так как мы используем “=” для соединения таблиц, кроме того ускорится выборка из таблицы Н_ВЕДОМОСТИ за счет использования Btree-индекса для оператора сравнения “>”. Также нет смысла создавать индексы для таблицы Н_ОЦЕНКА так как данная таблица имеет очень маленькое количество данных.

Вывод Explain Analyze

```
ucheb=> EXPLAIN ANALYZE SELECT "Н_ОЦЕНКИ"."КОД", "Н_ВЕДОМОСТИ"."ЧЛВК_ИД" FROM "Н_ОЦЕНКИ"
ucheb->     LEFT JOIN "Н_ВЕДОМОСТИ" ON "Н_ВЕДОМОСТИ"."ОЦЕНКА" = "Н_ОЦЕНКИ"."КОД"
ucheb-> WHERE "Н_ОЦЕНКИ"."ПРИМЕЧАНИЕ" = 'незачет' AND "Н_ВЕДОМОСТИ"."ДАТА" > '1998-01-05';
                                                              QUERY PLAN
----------------------------------------------------------------------------------------------------------------------------------
 Nested Loop  (cost=519.82..5198.80 rows=24113 width=38) (actual time=1.083..12.157 rows=2950 loops=1)
   ->  Seq Scan on "Н_ОЦЕНКИ"  (cost=0.00..1.11 rows=1 width=34) (actual time=0.068..0.071 rows=1 loops=1)
         Filter: (("ПРИМЕЧАНИЕ")::text = 'незачет'::text)
         Rows Removed by Filter: 8
   ->  Bitmap Heap Scan on "Н_ВЕДОМОСТИ"  (cost=519.82..4956.56 rows=24113 width=10) (actual time=1.012..10.544 rows=2950 loops=1)
         Recheck Cond: (("ОЦЕНКА")::text = ("Н_ОЦЕНКИ"."КОД")::text)
         Filter: ("ДАТА" > '1998-01-05 00:00:00'::timestamp without time zone)
         Heap Blocks: exact=732
         ->  Bitmap Index Scan on "ВЕД_ОЦЕНКА_I"  (cost=0.00..513.79 rows=24716 width=0) (actual time=0.837..0.837 rows=2950 loops=1)
               Index Cond: (("ОЦЕНКА")::text = ("Н_ОЦЕНКИ"."КОД")::text)
 Planning time: 1.998 ms
 Execution time: 12.990 ms
(12 rows)
```

Мои предположения предположения на счет использования индексов совпали с выводом Explain Analyze. Из-за того, что в системе на таблицу Н_ВЕДОМОСТИ создан b-tree индекс на столбец ОЦЕНКА выбрано сканирование с использованием битовых карт (вложенных циклов соединения было 1), когда при индексном сканировании нужно выбрать много записей за раз. То есть сначала идёт поиск по индексу, а затем массовое чтение нужных данных из таблицы по найденным в индексе указателям. При этом сначала указатели выстраиваются в порядке, в котором запрашиваемые данные физически расположены в базе, чтобы ускорить их чтение. Так же используется Recheck Cond из-за того, что объем данных в таблице слишком большой. В данном случае у нас хранятся не прямые указатели на строки в базе, а только указатели на страницы, в которых хранятся эти строки. Соответственно, после первоначального прохода по индексу требуется дополнительная проверка выбранных данных на соответствие запросу.
Для фильтрации Н_ВЕДОМОСТИ по ДАТА с оператором “>” был использован B-tree индекс и в данном случае из-за фильтрации у нас не было удалено ни одной строки, потому что самая маленькая дата является: “2008-11-03”. А для фильтрации Н_ОЦЕНКИ индекс не использовался, поскольку таблица имеет всего 9 строк (8 из которых впоследствии были отброшены и из-за этого у нас получилось, что для соединения понадобился всего один цикл).

Если бы для данного запроса использовали бы HASH JOIN, а не Nested Loop, то время бы увеличилось бы больше чем в 10 раз:

```
ucheb=> set enable_nestloop = false;
SET
ucheb=> EXPLAIN ANALYSE SELECT "Н_ОЦЕНКИ"."КОД", "Н_ВЕДОМОСТИ"."ЧЛВК_ИД" FROM "Н_ОЦЕНКИ"
    JOIN "Н_ВЕДОМОСТИ" ON "Н_ВЕДОМОСТИ"."ОЦЕНКА" = "Н_ОЦЕНКИ"."КОД"
WHERE "Н_ОЦЕНКИ"."ПРИМЕЧАНИЕ" = 'незачет' AND "Н_ВЕДОМОСТИ"."ДАТА" > '1998-01-05';
                                                                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------------------------------
 Hash Join  (cost=1.12..7902.58 rows=24113 width=38) (actual time=0.169..184.584 rows=2950 loops=1)
   Hash Cond: (("Н_ВЕДОМОСТИ"."ОЦЕНКА")::text = ("Н_ОЦЕНКИ"."КОД")::text)
   ->  Seq Scan on "Н_ВЕДОМОСТИ"  (cost=0.00..6846.50 rows=217020 width=10) (actual time=0.045..119.461 rows=217247 loops=1)
         Filter: ("ДАТА" > '1998-01-05 00:00:00'::timestamp without time zone)
         Rows Removed by Filter: 5193
   ->  Hash  (cost=1.11..1.11 rows=1 width=34) (actual time=0.014..0.014 rows=1 loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 9kB
         ->  Seq Scan on "Н_ОЦЕНКИ"  (cost=0.00..1.11 rows=1 width=34) (actual time=0.008..0.010 rows=1 loops=1)
               Filter: (("ПРИМЕЧАНИЕ")::text = 'незачет'::text)
               Rows Removed by Filter: 8
 Planning time: 0.236 ms
 Execution time: 185.381 ms
(12 rows)
```

Задание 2

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

Таблицы: Н_ЛЮДИ, Н_ОБУЧЕНИЯ, Н_УЧЕНИКИ.
Вывести атрибуты: Н_ЛЮДИ.ИД, Н_ОБУЧЕНИЯ.НЗК, Н_УЧЕНИКИ.ГРУППА.
Фильтры: (AND)
a) Н_ЛЮДИ.ФАМИЛИЯ < Соколов.
b) Н_ОБУЧЕНИЯ.ЧЛВК_ИД > 113409.
c) Н_УЧЕНИКИ.НАЧАЛО > 2011-11-21.
Вид соединения: RIGHT JOIN.

Запрос

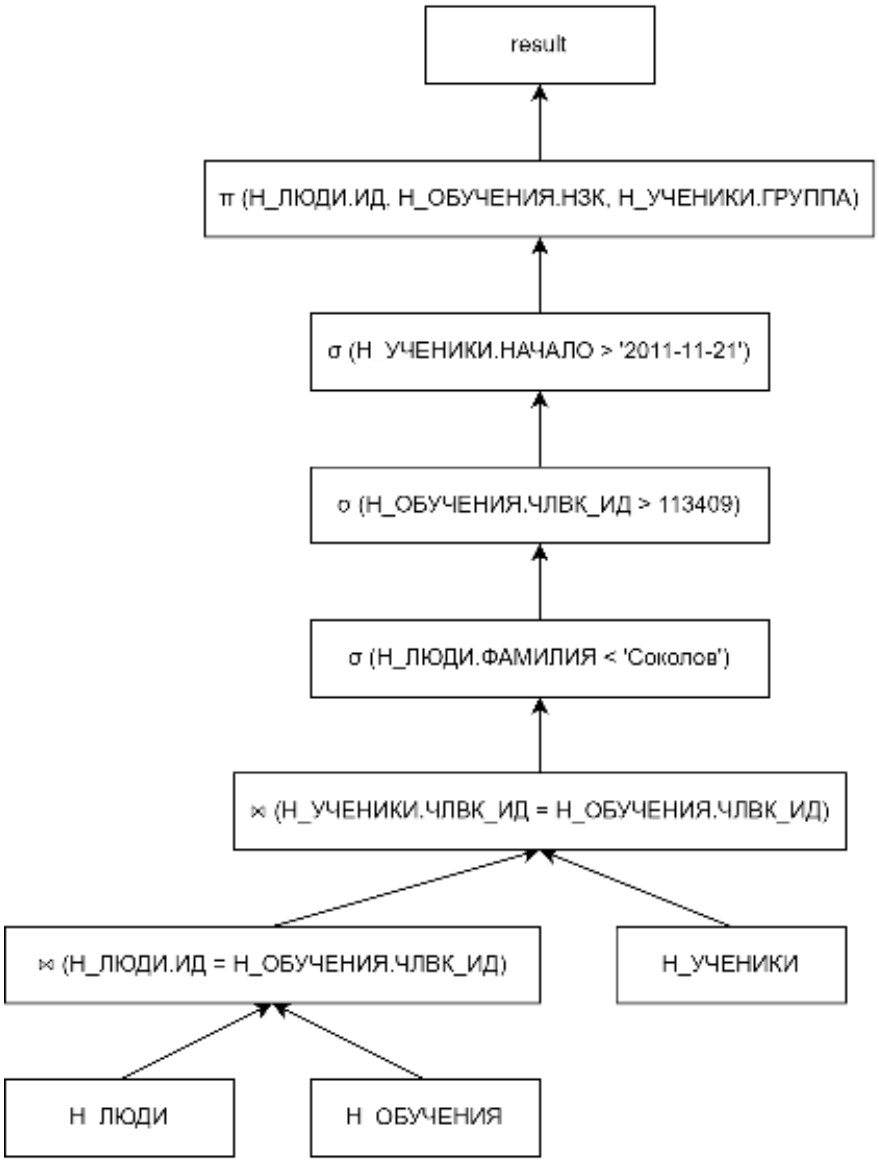
```
SELECT "Н_ЛЮДИ"."ИД", "Н_ОБУЧЕНИЯ"."НЗК", "Н_УЧЕНИКИ"."ГРУППА" FROM "Н_ЛЮДИ"
   RIGHT JOIN "Н_ОБУЧЕНИЯ" ON "Н_ЛЮДИ"."ИД" = "Н_ОБУЧЕНИЯ"."ЧЛВК_ИД"
   RIGHT JOIN "Н_УЧЕНИКИ" ON "Н_УЧЕНИКИ"."ЧЛВК_ИД" = "Н_ОБУЧЕНИЯ"."ЧЛВК_ИД"
WHERE "Н_ЛЮДИ"."ФАМИЛИЯ" < 'Соколов'
   AND "Н_ОБУЧЕНИЯ"."ЧЛВК_ИД" > 113409
   AND "Н_УЧЕНИКИ"."НАЧАЛО" > '2011-11-21';
```

Индексы

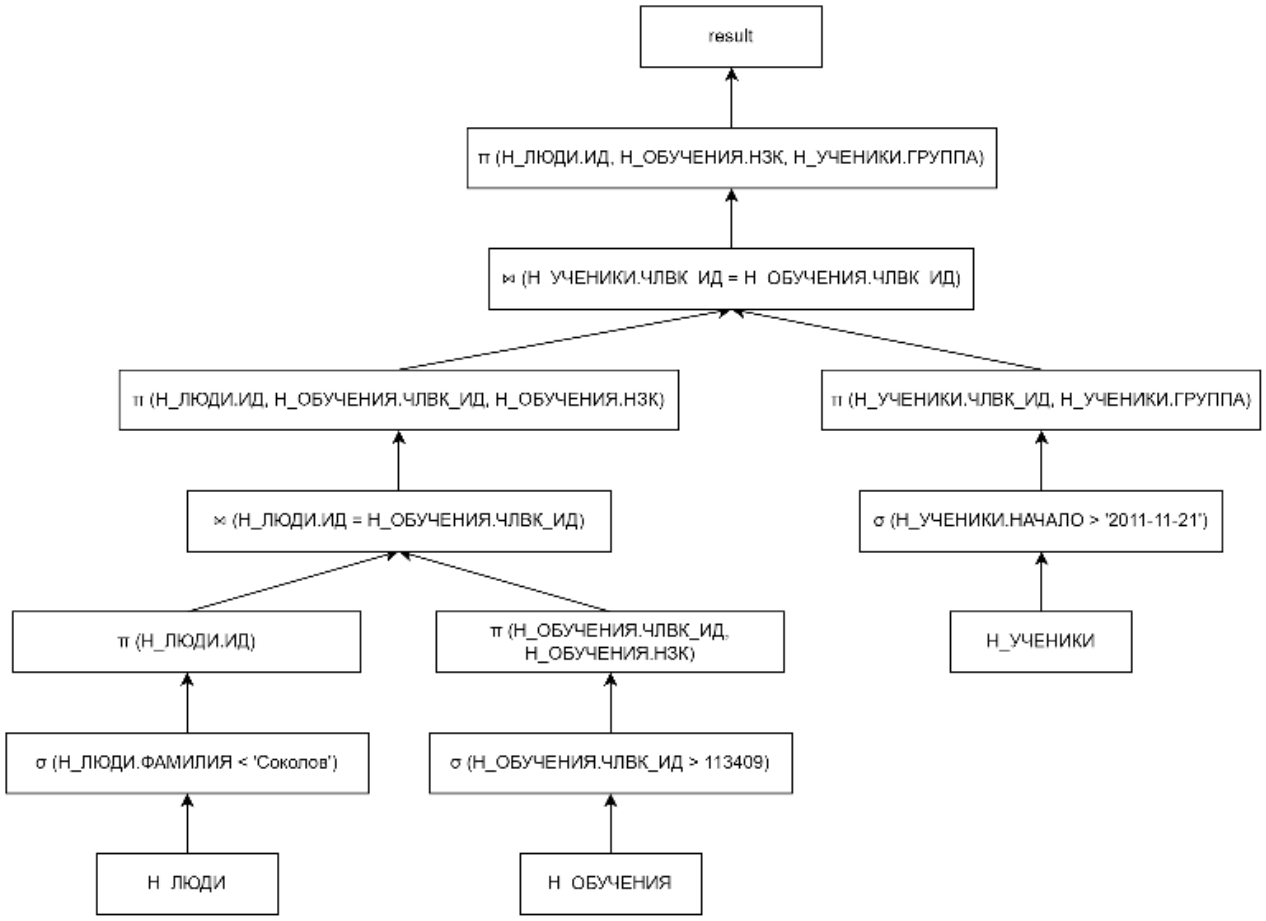
```
-- а) Индекс позволяет эффективно выбирать строки, основываясь на их положении в дереве, а также так как выражения в WHERE отбираются с использованием знака <.
CREATE INDEX "ИНД_ФАМИЛИЯ" ON "Н_ЛЮДИ" USING BTREE ("ФАМИЛИЯ");
-- б) Индекс позволяет эффективно выбирать строки, основываясь на их положении в дереве, а также так как выражения в WHERE отбираются с использованием знака >.
CREATE INDEX "ИНД_ЧЛВК_ИД" ON "Н_ОБУЧЕНИЯ" USING BTREE ("ЧЛВК_ИД");
-- в) Индекс позволяет эффективно выбирать строки, основываясь на их положении в дереве, а также так как выражения в WHERE отбираются с использованием знака >.
CREATE INDEX "ИНД_НАЧАЛО" ON "Н_УЧЕНИКИ" USING BTREE ("НАЧАЛО");
```

Возможные планы выполнения запроса

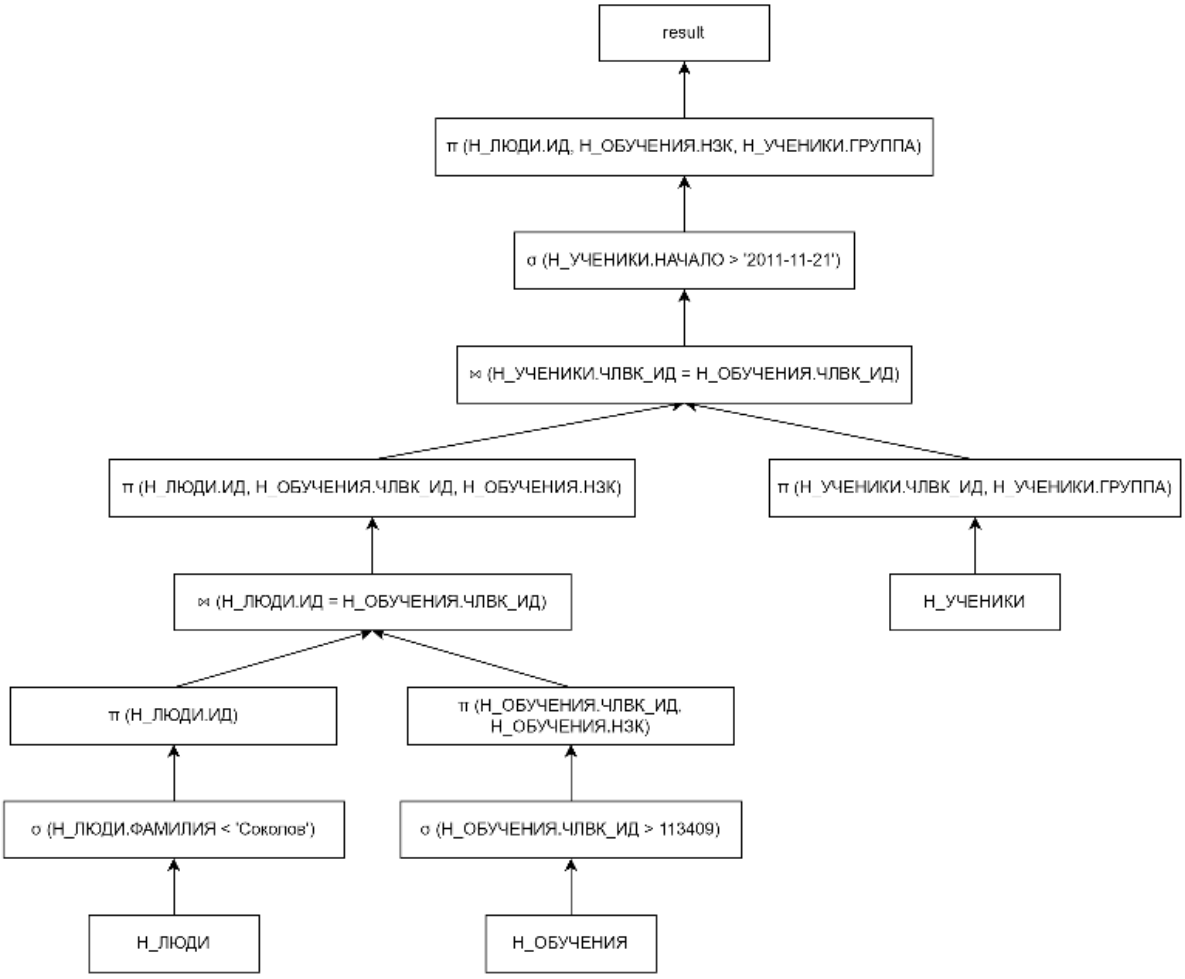
а)



б)


result
π (Н_ЛЮДИ.ИД, Н_ОБУЧЕНИЯ.НЗК, Н_УЧЕНИКИ.ГРУППА)
⋈ (Н_УЧЕНИКИ.ЧЛВК_ИД = Н_ОБУЧЕНИЯ.ЧЛВК_ИД)
π (Н_ЛЮДИ.ИД, Н_ОБУЧЕНИЯ.ЧЛВК_ИД, Н_ОБУЧЕНИЯ.НЗК)
π (Н_УЧЕНИКИ.ЧЛВК_ИД, Н_УЧЕНИКИ.ГРУППА)
⋈ (Н_ЛЮДИ.ИД = Н_ОБУЧЕНИЯ.ЧЛВК_ИД)
σ (Н_УЧЕНИКИ.НАЧАЛО > '2011-11-21')
π (Н_ЛЮДИ.ИД)
π (Н_ОБУЧЕНИЯ.ЧЛВК_ИД, Н_ОБУЧЕНИЯ.НЗК)
Н_УЧЕНИКИ
σ (Н_ЛЮДИ.ФАМИЛИЯ < 'Соколов')
σ (Н_ОБУЧЕНИЯ.ЧЛВК_ИД > 113409)
Н_ЛЮДИ
Н_ОБУЧЕНИЯ

в)



Планы построены в предположении о том, что все три таблицы сопоставимы по объему данных и условия соответствуют не нулевому количеству строк.

Оптимальным планом является план “б”, потому что все три таблицы имеют очень большое количество строк, а так же, после выполнения последнего соединения их останется небольшое количество с точки зрения СУБД (по факту 0). За счет использования проекции и фильтрации на ранних этапах происходит соединение только нужных нам атрибутов, следовательно промежуточные данные меньше. Так как у нас используется RIGHT JOIN логичнее всего поставить таблицу H_УЧЕНИКИ, с количеством после фильтрации 0 строк в самое “право”.

Логично использовать Nested Loop при соединении, так как у нас соединяются относительно небольшие таблицы, а атрибуты, участвующие в нём имеют индексы. Также мы знаем, что после фильтрации мы получим результат, содержащий 0 строк, а значит использование Hash Join привело бы к бесполезным избыточным накладкам.

При добавлении предложенных индексов выбранный план остается эффективнее и они ускоряют выполнение запроса, так как по ним идёт выборка с использованием операторов сравнения.

Вывод Explain Analyze

```
ucheb=> EXPLAIN ANALYZE SELECT "Н_ЛЮДИ"."ИД", "Н_ОБУЧЕНИЯ"."НЗК", "Н_УЧЕНИКИ"."ГРУППА" FROM "Н_ЛЮДИ"
ucheb->     RIGHT JOIN "Н_ОБУЧЕНИЯ" ON "Н_ЛЮДИ"."ИД" = "Н_ОБУЧЕНИЯ"."ЧЛВК_ИД"
ucheb->     RIGHT JOIN "Н_УЧЕНИКИ" ON "Н_УЧЕНИКИ"."ЧЛВК_ИД" = "Н_ОБУЧЕНИЯ"."ЧЛВК_ИД"
ucheb-> WHERE "Н_ЛЮДИ"."ФАМИЛИЯ" < 'Соколов'
ucheb->     AND "Н_ОБУЧЕНИЯ"."ЧЛВК_ИД" > 113409
ucheb->     AND "Н_УЧЕНИКИ"."НАЧАЛО" > '2011-11-21';
                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------
 Nested Loop  (cost=0.85..16.70 rows=1 width=14) (actual time=0.006..0.006 rows=0 loops=1)
   ->  Nested Loop  (cost=0.57..16.31 rows=1 width=12) (actual time=0.005..0.005 rows=0 loops=1)
         ->  Index Scan using "УЧЕН_НАЧ_I" on "Н_УЧЕНИКИ"  (cost=0.29..8.00 rows=1 width=8) (actual time=0.003..0.003 rows=0 loops=1)
               Index Cond: ("НАЧАЛО" > '2011-11-21 00:00:00'::timestamp without time zone)
         ->  Index Scan using "ЧЛВК_PK" on "Н_ЛЮДИ"  (cost=0.28..8.30 rows=1 width=4) (never executed)
               Index Cond: ("ИД" = "Н_УЧЕНИКИ"."ЧЛВК_ИД")
               Filter: (("ФАМИЛИЯ")::text < 'Соколов'::text)
   ->  Index Scan using "ОБУЧ_ЧЛВК_FK_I" on "Н_ОБУЧЕНИЯ"  (cost=0.28..0.38 rows=1 width=10) (never executed)
         Index Cond: (("ЧЛВК_ИД" = "Н_ЛЮДИ"."ИД") AND ("ЧЛВК_ИД" > 113409))
 Planning time: 2.289 ms
 Execution time: 0.049 ms
(11 rows)
```

Мои предположения относительная индексов и типа соединения совпали с выводом Explain Analyze. Был выбран Nested Loop и использовались индексы для соединения, так как ожидалось, что после фильтрации по НАЧАЛО останется хотя бы 1 строка, а это есть самый оптимальный вариант для использования Nested Loop. Но по факту после фильтрации мы получаем 0 строк, и дальнейший план не выполняется.

Вывод

В результате выполнения лабораторной работы были разработаны и проанализированы два SQL запроса и планы их выполнения. В ходе выполнения были изучены особенности составления и обработки планов СУБД PostgreSQL при использовании и без использования индексов. Были изучены основные виды индексов и стратегии соединения таблиц, применяемых в данной СУБД.

Ассоциативность right join примеры, когда ассоциативен и когда нет

Ассоциативный RIGHT JOIN:

```
drop table atab, btab, ctab cascade;

CREATE TABLE atab (id integer, val varchar(10));
CREATE TABLE btab (id integer, val varchar(10));
CREATE TABLE ctab (id integer, val varchar(10));

INSERT INTO atab VALUES (1, 'A1');
INSERT INTO atab VALUES (2, 'A2');
INSERT INTO atab VALUES (3, 'A3');

INSERT INTO btab VALUES (1, 'B1');
INSERT INTO btab VALUES (2, 'B2');
INSERT INTO btab VALUES (3, 'B4');

INSERT INTO ctab VALUES (1, 'C1');
INSERT INTO ctab VALUES (2, 'C2');
INSERT INTO ctab VALUES (3, 'C3');
```

(A ⋈ B) ⋈ C

```
SELECT * FROM (SELECT * FROM atab
  RIGHT JOIN btab USING(id)) ab
  RIGHT JOIN ctab USING (id);
```

	id	atab.val	btab.val	ctab.val
1	1	A1	B1	C1
2	2	A2	B2	C3
3	3	A3	B4	C5

A ⋈ (B ⋈ C)

```
SELECT * FROM atab
  RIGHT JOIN (SELECT * FROM btab
  RIGHT JOIN ctab USING (id)) bc USING(id);
```

	id	val	val	val
1	1	A1	B1	C1
2	2	A2	B2	C3
3	3	A3	B4	C5

НЕ Ассоциативный RIGHT JOIN:

```
drop table atab, btab, ctab cascade;

CREATE TABLE atab (id integer, val varchar(10));
CREATE TABLE btab (id integer, val varchar(10));
CREATE TABLE ctab (id integer, val varchar(10));

INSERT INTO atab VALUES (1, 'A1');
INSERT INTO atab VALUES (2, 'A2');
INSERT INTO atab VALUES (3, 'A3');

INSERT INTO btab VALUES (1, 'B1');
INSERT INTO btab VALUES (2, 'B2');
INSERT INTO btab VALUES (4, 'B4');

INSERT INTO ctab VALUES (1, 'C1');
INSERT INTO ctab VALUES (2, 'C2');
INSERT INTO ctab VALUES (3, 'C3');
```

(A ⋈ B) ⋈ C

```
SELECT * FROM (SELECT * FROM atab
  RIGHT JOIN btab USING(id)) ab
  RIGHT JOIN ctab USING (id);
```

1	1	A1	B1	C1
2	2	A2	B2	C2
3	3	<null>	<null>	C3

A ⋈ (B ⋈ C)

```
SELECT * FROM atab
  RIGHT JOIN (SELECT * FROM btab
  RIGHT JOIN ctab USING (id)) bc USING(id);
```

4	1	A1	B1	C1
5	2	A2	B2	C2
6	3	A3	<null>	C3

Вывод:

Написав такой пример и проанализирова работу RIGHT JOIN можно сделать вывод:
Если по стоблцам которым мы соединяем:

1. Айди у всех 3 таблиц одинаковые - **ассоциативность работает**
2. Айдишники все различные у 3 таблиц - **ассоциативность работает**
3. Если у средней таблицы совпадает айдишиники с любой из крайних таблиц полность - **ассоциативность работает**
4. **Во всех остальных случаях ассоциативность не работает.**

То есть чтобы ассоциативность не сработала, в левой таблице должна быть запись, которую отсечет средняя таблица, но которая сможет сджойниться с правой.

Вспомогательный материал по Explain Analyze

Seq Scan - последовательный перебор строк таблицы (возможно с отбором по условию).
Index Only Scan - поиск по покрывающему индексу без захода в основную таблицу.
Index Scan - поиск по индексу, с заходом в основную таблицу за доп. колонками.
Nested Loops - соединение вложенными циклами. (если не большие объемы нужно соединить)
Hash Join - соединение с помощью хеш-таблицы.
Merge Join - соединение заранее отсортированных наборов с помощью алгоритма слияния. (самое быстрое)
Sort - сортировка УПОРЯДОЧИТЬ ПО.

https://picloud.pw/media/resources/posts/2018/02/20/%D0%9B%D0%B5%D0%BA%D1%86%D0%B8%D1%8F_6.pdf