УНИВЕРСИТЕТ ИТМО

Факультет программной инженерии и компьютерной техники

Направление подготовки 09.03.01 Информатика и вычислительная техника

Дисциплина «Информационный системы и базы данных»

Лабораторная работа №4

Вариант № 3579

Студент

Воронина Д. С.

P33311

Преподаватель

Николаев В. В.

Санкт-Петербург

2023

1. Текст задания.

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:
   Таблицы: Н_ЛЮДИ, Н_СЕССИЯ.
   Вывести атрибуты: Н_ЛЮДИ.ФАМИЛИЯ, Н_СЕССИЯ.ЧЛВК_ИД.
   Фильтры (AND):
   a) Н_ЛЮДИ.ФАМИЛИЯ < Иванов.
   b) Н_СЕССИЯ.ДАТА = 2004-01-17.
   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.ИМЯ, Н_ВЕДОМОСТИ.ЧЛВК_ИД, Н_СЕССИЯ.ЧЛВК_ИД.
   Фильтры (AND):
   a) Н_ЛЮДИ.ИД = 100012.
   b) Н_ВЕДОМОСТИ.ЧЛВК_ИД = 105590.
   c) Н_СЕССИЯ.ЧЛВК_ИД < 100622.
   Вид соединения: LEFT JOIN.

2. Реализация SQL - запросов.

1. Сделать запрос для получения атрибутов из указанных таблиц, применив фильтры по указанным условиям:
   Таблицы: Н_ЛЮДИ, Н_СЕССИЯ.
   Вывести атрибуты: Н_ЛЮДИ.ФАМИЛИЯ, Н_СЕССИЯ.ЧЛВК_ИД.
   Фильтры (AND):
   a) Н_ЛЮДИ.ФАМИЛИЯ < Иванов.
   b) Н_СЕССИЯ.ДАТА = 2004-01-17.
   Вид соединения: INNER JOIN.

```
SELECT Н_ЛЮДИ.ФАМИЛИЯ, Н_СЕССИЯ.ЧЛВК_ИД FROM Н_ЛЮДИ
INNER JOIN Н_СЕССИЯ ON Н_СЕССИЯ.ЧЛВК_ИД = Н_ЛЮДИ.ИД
WHERE Н_ЛЮДИ.ФАМИЛИЯ < 'Иванов'
AND Н_СЕССИЯ.ДАТА = '2004-01-17'::timestamp;
```

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.ИМЯ, Н_ВЕДОМОСТИ.ЧЛВК_ИД, Н_СЕССИЯ.ЧЛВК_ИД.
   Фильтры (AND):
   a) Н_ЛЮДИ.ИД = 100012.
   b) Н_ВЕДОМОСТИ.ЧЛВК_ИД = 105590.
   c) Н_СЕССИЯ.ЧЛВК_ИД < 100622.
   Вид соединения: LEFT JOIN.

```
SELECT Н_ЛЮДИ.ИМЯ, Н_ВЕДОМОСТИ.ЧЛВК_ИД, Н_СЕССИЯ.ЧЛВК_ИД FROM Н_ЛЮДИ
LEFT JOIN Н_ВЕДОМОСТИ ON Н_ВЕДОМОСТИ.ЧЛВК_ИД = Н_ЛЮДИ.ИД
LEFT JOIN Н_СЕССИЯ ON Н_СЕССИЯ.ЧЛВК_ИД = Н_ЛЮДИ.ИД
WHERE Н_ЛЮДИ.ИД = 100012
AND Н_ВЕДОМОСТИ.ЧЛВК_ИД = 105590
AND Н_СЕССИЯ.ЧЛВК_ИД < 100622;
```

3. Планы выполнения запросов.

Запрос № 1

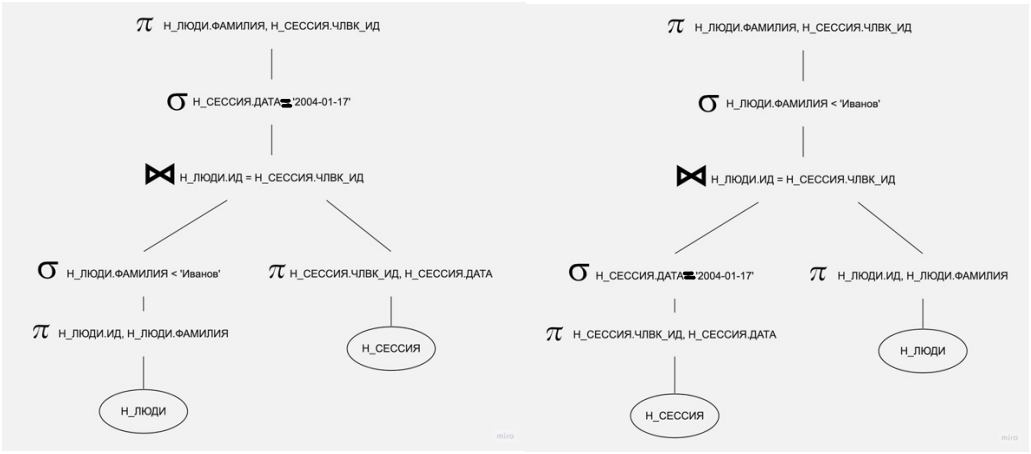

Рис. 1, 2 Планы 1, 2 выполнения запроса № 1

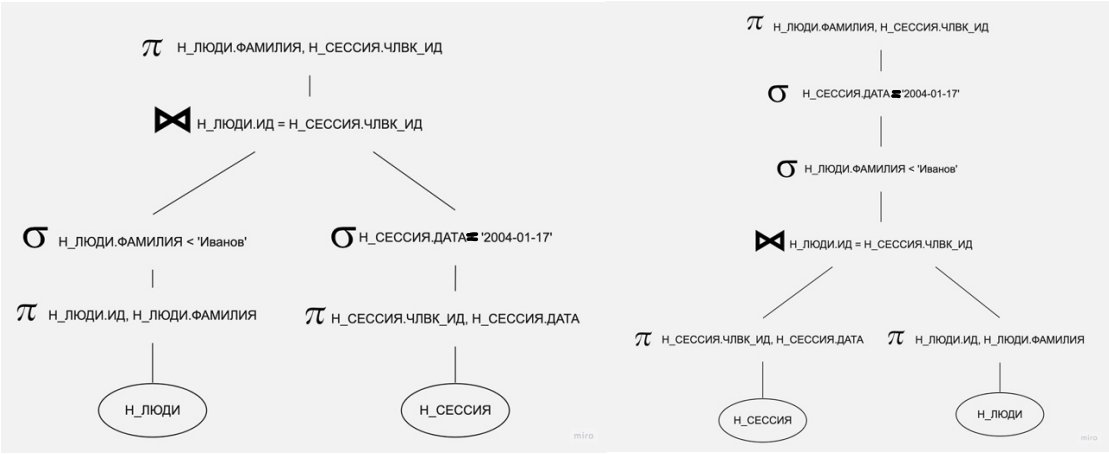

Рис. 3, 4 Планы 3, 4 выполнения запроса № 1

Оптимальный является 3 план, поскольку мы сначала делаем проекции таблиц с нужными атрибутами, фильтруем таблицы, а затем соединяем. В результате соединяется не так много данных, как могло быть, если бы мы не сделали проекции и не отфильтровали таблицы перед соединением. При добавлении индексов планы не изменятся, так как индекс повлияет лишь на эффективность фильтрации данных в таблицах и их соединения.

Алгоритм соединения: Nested Loop Join

Н_ЛЮДИ содержит 5118 строк, Н_СЕССИЯ – 3750. Количество данных довольно не велико, к тому же перед соединением фильтры существенно уменьшают количество строк до Н_ЛЮДИ – 1573, Н_СЕССИЯ – 7 строк. A Nested Loop Join эффективно использовать при небольшом количестве данных.

Hash Join обычно предпочтительнее для больших наборов данных.

Sort-Merge Join эффективен, если данные в таблицах уже отсортированы или могут быть легко отсортированы по ключу соединения. Однако, с учетом correlation = 0.03 для Н_ЛЮДИ.ИД сортировка затратна.

Запрос № 2

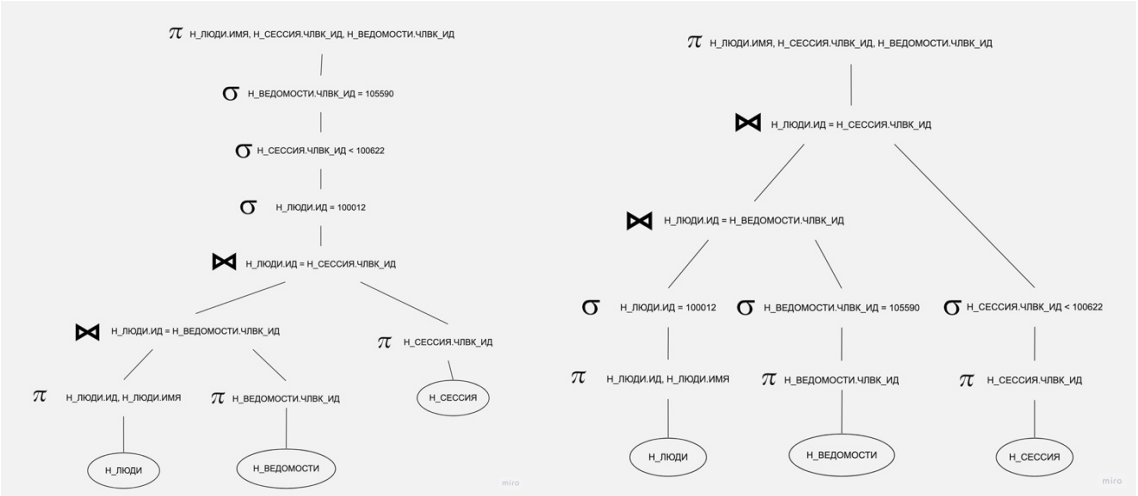

Рис. 5, 6 Планы 1, 2 выполнения запроса № 2

Можно составить ещё много вариантов последовательного расположения операций фильтрации и соединений, но все дальнейшие комбинации не будут эффективны, поэтому их можно не изображать. Оптимальным планом будет план 2, потому что в нём данные сначала берется проекция нужных атрибутов, затем таблицы фильтруются, а потом используются в соединении. При добавлении индексов планы не изменятся.

Таблица Н_ЛЮДИ содержит 5118 строк, в то время как Н_ВЕДОМОСТИ 222440. В таком случае выгодно использовать алгоритм Hash join, так как построение hash-table будет по Н_ЛЮДИ, которая во много раз меньше, а Н_ВЕДОМОСТИ содержит большое количество строк.

Но после фильтрации в таблицах остается Н_ЛЮДИ – 1 строка, Н_ВЕДОМОСТИ – 27 строк, Н_СЕССИЯ – 1087 строк.

И эффективно будет использовать Nested Loop Join, так как данных не очень много.

Sort-Merge Join не будет эффективным, потому что Н_СЕССИЯ довольно велика, процесс сортировки с учетом correlation = 0.13 атрибута ЧЛВК_ИД займет много ресурсов и времени.

Алгоритм соединение: Nested Loop Join

4. Добавление индексов.

1. Добавление индексов

Для **запроса № 1** можно добавить индексы на следующие атрибуты и таблицы:

- Индекс на атрибут ФАМИЛИЯ таблицы Н_ЛЮДИ, тип индекса – B-tree.

```
CREATE INDEX Н_ЛЮДИ_ФАМИЛИЯ_I ON Н_ЛЮДИ
USING btree(ФАМИЛИЯ) WHERE ФАМИЛИЯ < 'Иванов';
```

Статистика данного атрибута в специальной таблице pg_stats:

	null_frac	n_distinct	correlation
1	0	-0.7317311	-0.0006484674

Отрицательное значение n_distinct означает (если взять его по модулю) долю уникальных значений среди всех значений. Это доля довольно большая, поэтому созданный индекс с фильтром отсеет большое количество ненужных запросу значений и ускорит фильтрацию. Но correlation близко к 0, поэтому обслуживание индекса дорогостояще, из-за прыжков для получения данных по физическому адресу. Поэтому тут нужно задуматься о том, что принесет меньше убытков.

Был выбран тип B-tree, так как он поддерживает операции сравнения, а у нас как раз таки сравнение по полю ФАМИЛИЯ.

- Индекс на атрибут ДАТА таблицы Н_СЕССИЯ, тип индекса – Hash.

```
CREATE INDEX Н_СЕССИЯ_ДАТА_I ON Н_СЕССИЯ
USING hash(ДАТА) WHERE ДАТА IS NOT NULL;
```

Статистика данного атрибута в специальной таблице pg_stats:

	null_frac	n_distinct	correlation
1	0.07995736	-0.1369936	0.66200286

Отрицательное значение n_distinct означает, что число различных значений, скорее всего, будет расти по мере роста таблицы. А значение correlation ближе к 1, что означает, что сканирование индекса по колонке будет считаться дешевле, как результат уменьшения случайного доступа к диску.

Причины, почему не были созданы другие индексы:

1) Индекс в таблице Н_ЛЮДИ на атрибут ИД – он уже создан СУБД автоматически как индекс для основного ключа.
2) Индекс на атрибут ЧЛВК_ИД таблицы Н_СЕССИЯ – не создаю, так как после фильтрации остается всего 7 строк. И в соединение идет только 7 строк этой таблицы.

Сначала можно проанализировать **запрос № 2,** и мы поймем, что в целом, запрос прервется на обнаружении заведомо ложного выражения в WHERE-блоке.

```
WHERE Н_ЛЮДИ.ИД = 100012
AND Н_ВЕДОМОСТИ.ЧЛВК_ИД = 105590
AND Н_СЕССИЯ.ЧЛВК_ИД < 100622;
```

Но если бы не было заведомо ложного выражения в WHERE-блоке тогда можно было бы добавить следующие индексы:

- Индекс на атрибут ЧЛВК_ИД таблицы Н_ВЕДОМОСТИ, тип индекса – Hash.

```
CREATE INDEX Н_ВЕДОМОСТИ_ЧЛВК_ИД_I ON Н_ВЕДОМОСТИ
USING hash(ЧЛВК_ИД);
```

Статистика данного атрибута в специальной таблице pg_stats:

	null_frac	n_distinct	correlation
1	0	3272	0.5089833

Мы видим, что значение n_distinct, показывающее, сколько уникальных значений данного атрибута существует в таблице в данный момент, довольно большое, а также correlation ближе к 1, чем к 0, что означает необходимость в применение индекса по данному атрибуту. Также данный атрибут участвует в операции соединения с таблицей Н_ЛЮДИ.

- Индекс на атрибут ЧЛВК_ИД таблицы Н_СЕССИЯ, тип индекса – Hash. С указанием того, что применяем на значениях не NULL.

```
CREATE INDEX Н_СЕССИЯ_ЧЛВК_ИД_I ON Н_СЕССИЯ
USING hash(ЧЛВК_ИД) WHERE ЧЛВК_ИД IS NOT NULL;
```

Статистика данного атрибута в специальной таблице pg_stats:

	null_frac	n_distinct	correlation
1	0.13672708	180	0.13944401

Видим, что n_distinct И correlation не говорят о том, что индекс сильно необходим. Однако данный атрибут участвует в операции соединения с таблицей Н_ЛЮДИ, где строк довольно много. Поэтому к этому атрибуту будет довольного много обращений, что подтверждает полезность индекса. Следовательно, мы можем добавить оптимизацию, исключив из индексации значения NULL.

Был выбран тип индекса Hash, так как в операции соединения постоянно будут использовать сравнения по равенству.

Причины, почему не были созданы другие индексы:

1) Индекс в таблице Н_ЛЮДИ на атрибут ИД – он уже создан СУБД автоматически как индекс для основного ключа.

5. Анализ выбора планировщика

Запрос № 1

```
QUERY PLAN
        Filter: ("ДАТА" = '2004-01-17 00:00:00'::timestamp without time zone)
        Filter: (("ФАМИЛИЯ")::text < 'Иванов'::text)
        Index Cond: ("ИД" = "Н_СЕССИЯ"."ЧЛВК_ИД")
        Rows Removed by Filter: 1
        Rows Removed by Filter: 3745
  ->  Index Scan using "ЧЛВК_РК" on "Н_ЛЮДИ"  (cost=0.28..8.30 rows=1 width=20) (actual time=0.003..0.003 r…
  ->  Seq Scan on "Н_СЕССИЯ"  (cost=0.00..117.90 rows=5 width=4) (actual time=0.015..0.509 rows=7 loops=1)
Execution Time: 0.593 ms
Nested Loop  (cost=0.28..159.42 rows=1 width=20) (actual time=0.039..0.542 rows=2 loops=1)
Planning Time: 1.448 ms
```

Из результатов можно увидеть, что для выполнения запроса был выбран план, в котором выполняется фильтрация таблиц Н_ЛЮДИ по атрибуту ФАМИЛИЯ и Н_СЕССИЯ по атрибуту ДАТА.

Далее выполняется соединение с использованием вложенный цикл (Nested Loop). Он соединяет таблицы Н_ЛЮДИ И Н_СЕССИЯ с использованием условия соединения по полю Н_СЕССИЯ.ЧЛВК_ИД.

Также можно увидеть планируемое время выполнения 1.448. В данном случае запрос был выполнен очень быстро (Execution Time: 0.039 ms), что свидетельствует о том, что он не нагружает базу данных. И посмотрим на cost – оценка ресурсов, равное 0.29. И на общую стоимость выполнения запроса 159.42.

Из описания индексных сканов можно сделать вывод, что для оптимизации запроса были использованы индексы, что позволяет ускорить выполнение запроса.

Запрос № 2

```
QUERY PLAN
Result  (cost=0.00..0.00 rows=0 width=4) (actual time=0.001..0.001 rows=0 loops=1)
  One-Time Filter: false
Planning Time: 0.206 ms
Execution Time: 0.012 ms
```

В документации написано, что когда в запросе заведомо ложное выражение в WHERE-блоке, тогда возникает атрибут One-Time Filter.

В WHERE-блоке действительно находится ложное выражение по условию задания.

If it is false, an empty result set can be returned without further work.

Что означает, что возвращен пустой набор результатов без дальнейшей работы.

Поэтому выполнение запроса занимает 0.012мс.

Вывод

При выполнении данной лабораторной работы я изучила различные виды индексов такие, как Hash и B-tree, узнала, как использовать их для оптимизации скорости выполнения запросов, а также когда использование индексов может быть неэффективно.