Университет ИТМО, факультет ПИиКТ

Лабораторная работа №3

Вариант 528432

Дисциплина: Информационные системы и базы данных

Выполнил: Чангалиди Антон

Группа: P33113

Преподаватель:
Николаев Владимир Вячеславович

г. Санкт-Петербург

2020 г.

Plan-reading is an art
that requires some experience to master.
© PostgreSQL documentation

Задание

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).

Запрос 1:

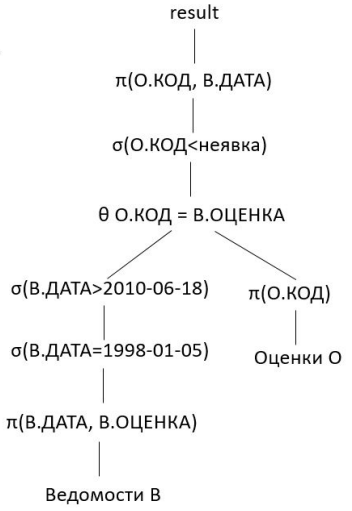
Таблицы: Н_ОЦЕНКИ, Н_ВЕДОМОСТИ.
Вывести атрибуты: Н_ОЦЕНКИ.КОД, Н_ВЕДОМОСТИ.ДАТА.
Фильтры (AND):
A. Н_ОЦЕНКИ.КОД < неявка.
B. Н_ВЕДОМОСТИ.ДАТА = 1998-01-05.
C. Н_ВЕДОМОСТИ.ДАТА > 2010-06-18.
Вид соединения: INNER JOIN.

Запрос:

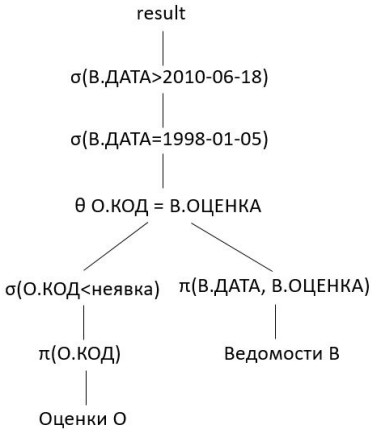
```
select O."КОД", B."ДАТА"
from "Н_ОЦЕНКИ" O
        INNER JOIN "Н_ВЕДОМОСТИ" B ON O."КОД" = B."ОЦЕНКА"
WHERE O."КОД" < 'неявка'
 AND B."ДАТА" = '1998-01-05'
 AND B."ДАТА" > '2010-06-18';
```

Варианты плана выполнения запросов:

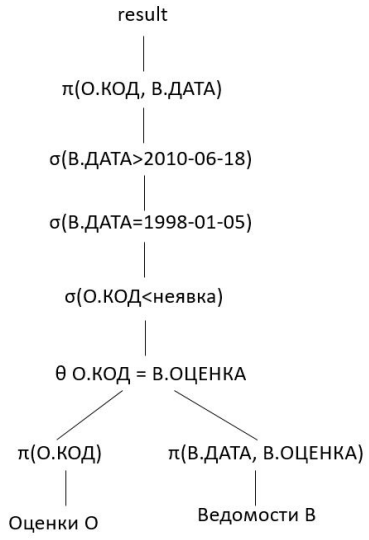
1.



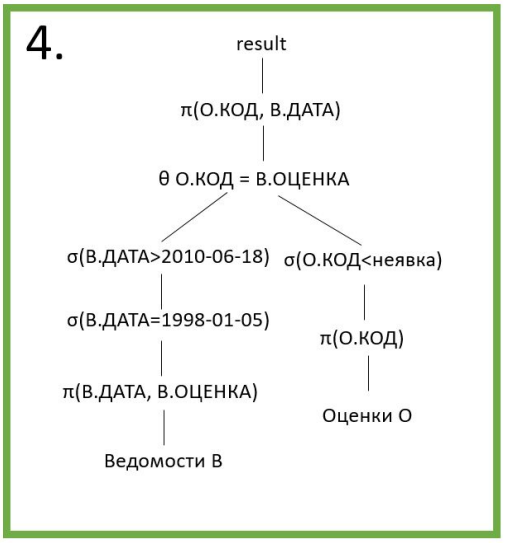
2.



3.



4.



Чтобы оптимизировать запрос, необходимо **делать выборку как можно раньше.** Т.е. оптимальным является план номер 4. Причем имеет смысл сделать сначала выборку по B.ДАТА = , а потом >, так мы сразу сильно сузим множество и не надо будет 2 раза по всем

датам, большим 2010-06-18 ходить (можно заметить, что оно во время второй выборки не будет подходить ни одна строчка, и объединять будет вообще чудесно).

Индексы:

К О."КОД" добавлять индекс смысла нет, так как индексы неэффективны, если в таблице мало строк (в таблице Н_ОЦЕНКИ всего 9 строк), а место занимать будут.

На В."ДАТА" имеет смысл добавить индекс B-TREE (дерево спускается по веткам на основе сравнений, и работает за log(глубина), а обычный поиск за линейное время), так как используется и =, и >.

Так как джойним через В."ОЦЕНКА", туда имеет смысл добавить индекс HASH, так как при join мы будем искать по конкретному значению (проверять на равенство) и это значительно ускорит работу.

```
CREATE INDEX ИН_ВЕД_ДАТА ON "Н_ВЕДОМОСТИ" USING BTREE ("ДАТА");
CREATE INDEX ИН_ВЕД_ДАТА ON "Н_ВЕДОМОСТИ" USING HASH ("ОЦЕНКА");
```

Добавление этих индексов может ускорить запросы, потому что по данным атрибутам идет выборка с использованием операторов (как я говорил = и >) и соединение таблиц.

EXPLAIN ANALYZE:

```
    Hash Join  (cost=1.57..1731.73 rows=212 width=42) (actual time=0.011..0.011
rows=0 loops=1)
    "  Hash Cond: (((""В"".""ОЦЕНКА"")::text = (""О"".""КОД"")::text)"
    "  ->  Index Scan using ""ВЕД_ДАТА_I"" on ""Н_ВЕДОМОСТИ"" ""В""
(cost=0.42..1726.08 rows=636 width=14) (actual time=0.011..0.011 rows=0 loops=1)"
    "        Index Cond: ((""ДАТА"" > '2010-06-18 00:00:00'::timestamp without time
zone) AND (""ДАТА"" = '1998-01-05 00:00:00'::timestamp without time zone))"
     ->  Hash  (cost=1.11..1.11 rows=3 width=34) (never executed)
    "        ->  Seq Scan on ""Н_ОЦЕНКИ"" ""О""  (cost=0.00..1.11 rows=3 width=34)
(never executed)"
    "              Filter: (((""КОД"")::text < 'неявка'::text)"
    Planning time: 0.492 ms
    Execution time: 0.049 ms
```

Запрос 2:

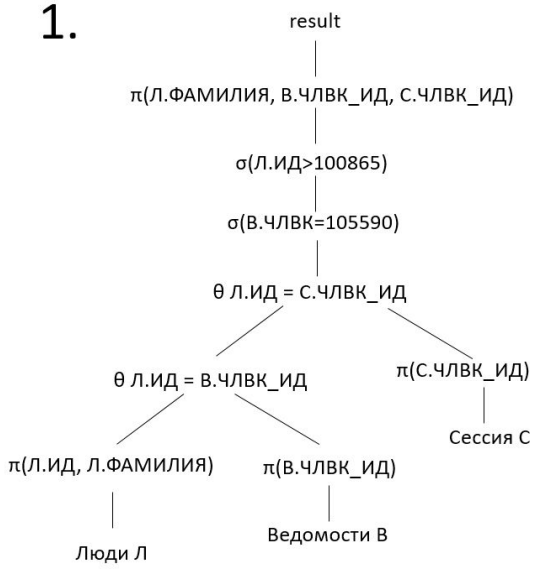
```
Таблицы: Н_ЛЮДИ, Н_ВЕДОМОСТИ, Н_СЕССИЯ.
Вывести атрибуты: Н_ЛЮДИ.ФАМИЛИЯ, Н_ВЕДОМОСТИ.ЧЛВК_ИД,
Н_СЕССИЯ.ЧЛВК_ИД.
Фильтры (AND):
A.   Н_ЛЮДИ.ИД > 100865.
B.   Н_ВЕДОМОСТИ.ЧЛВК_ИД = 105590.
Вид соединения: LEFT JOIN.
```

Запрос:

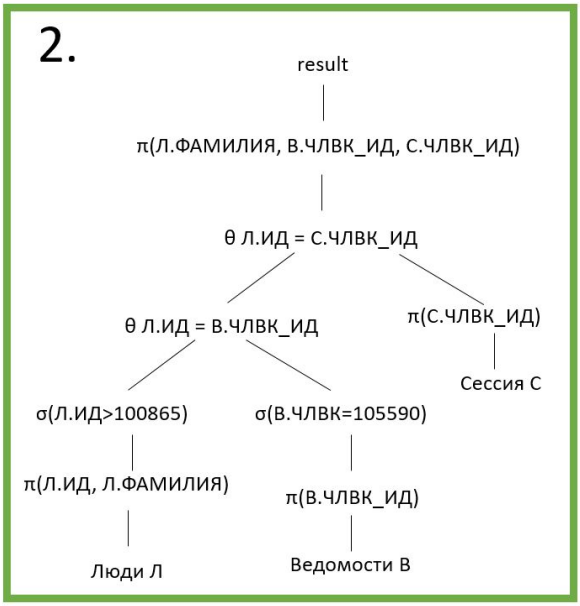
```
SELECT Л."ФАМИЛИЯ",
       В."ЧЛВК_ИД",
       С."ЧЛВК_ИД"
FROM "Н_ЛЮДИ" Л
        LEFT JOIN "Н_ВЕДОМОСТИ" В on Л."ИД" = В."ЧЛВК_ИД"
        LEFT JOIN "Н_СЕССИЯ" С on Л."ИД" = С."ЧЛВК_ИД"
WHERE Л."ИД" > 100865
 AND В."ЧЛВК_ИД" = 105590;
```

Варианты плана выполнения запросов:

1.



2.



Первый план точно самый неоптимальный, так как там все сравнения и уменьшения выборки происходят после объединений. Значит, учитывая то, что в данном случае используется Left Join и что сторона ветви имеет значение, остается только 2й план, который и является оптимальным: мы сразу перед первым left join делаем выборку и уменьшаем множество для объединения.

Индексы:

На В."ЧЛВК_ИД" имеет смысл добавить **HASH-индекс** (хеширует значение идет по этому ключу, таким образом время исполнения - амортизированная константа), так как идет прямое сравнение + поможет при объединении: Ведомости - правая таблица Left Join'а и по ней будет вестись поиск по конкретным id.

Аналогичная ситуация с С."ЧЛВК_ИД", хоть по нему не происходит выборка, но происходит объединение: это тоже правая таблица Left Join'а и по ней будет вестись поиск по конкретным id => использование HASH-индекса.

Добавление HASH-индекса на Л.ИД не имеет смысла, так как оба раза таблица ЛЮДИ является левым множеством Left Join'а и постгрес все равно полностью проитерируется по нему. На Л."ИД" имеет смысл добавить индекс **B-TREE** (про дерево писал выше), так как при выборке используется > (и не используется логическое ИЛИ).

```
CREATE INDEX ИН_Л_ИД ON "Н_ЛЮДИ" USING BTREE ("ИД");
CREATE INDEX ИН_С_ЧЛВК_ИД ON "Н_СЕССИЯ" USING HASH ("ЧЛВК_ИД");
CREATE INDEX ИН_В_ЧЛВК_ИД ON "Н_ВЕДОМОСТИ" USING HASH ("ЧЛВК_ИД");
```

Добавление этих индексов может ускорить запросы, потому что по данным атрибутам идет выборка с использованием операторов (= и >) и соединение таблиц.

EXPLAIN ANALYZE:

```
    Nested Loop  (cost=5.04..235.94 rows=448 width=24) (actual time=0.049..0.575 rows=540
loops=1)
    "  ->  Index Only Scan using ""ВЕД_ЧЛВК_FK_IFK"" on ""Н_ВЕДОМОСТИ"" ""В""
(cost=0.42..196.10 rows=64 width=4) (actual time=0.022..0.069 rows=27 loops=1)"
    "        Index Cond: (""ЧЛВК_ИД"" = 105590)"
          Heap Fetches: 27
     ->  Materialize  (cost=4.62..34.25 rows=7 width=24) (actual time=0.001..0.007 rows=20
loops=27)
           ->  Nested Loop Left Join  (cost=4.62..34.22 rows=7 width=24) (actual
time=0.023..0.050 rows=20 loops=1)
    "              Join Filter: (""Л"".""ИД"" = ""С"".""ЧЛВК_ИД"")"
```

```
"              ->  Index Scan using ""ЧЛВК_РК"" on ""Н_ЛЮДИ"" ""Л""  (cost=0.28..8.30
rows=1 width=20) (actual time=0.004..0.004 rows=1 loops=1)"
"                    Index Cond: ((""ИД"" > 100865) AND (""ИД"" = 105590))"
"              ->  Bitmap Heap Scan on ""Н_СЕССИЯ"" ""С""  (cost=4.33..25.83 rows=7
width=4) (actual time=0.013..0.028 rows=20 loops=1)"
"                    Recheck Cond: (""ЧЛВК_ИД"" = 105590)"
                   Heap Blocks: exact=9
"                    ->  Bitmap Index Scan on ""SYS_C003500_IFK""  (cost=0.00..4.33
rows=7 width=0) (actual time=0.009..0.009 rows=20 loops=1)"
"                          Index Cond: (""ЧЛВК_ИД"" = 105590)"
Planning time: 0.249 ms
Execution time: 0.791 ms
```

Выводы

Выполнив эту лабораторную работу я узнал много нового о возможностях языка SQL. В особенности об анализе и ускорении выполнения запросов.