Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Інститут комп'ютених наук та інформаційних технологій

Кафедра «Системи штучного інтелекту»

Лабораторна робота №13

з курсу "Організація баз даних та знань"

Виконала:

студентка групи КН-208

Ріжко Марія

Перевірила:

Якимишин Х.М.

Львів 2020 р.

Мета роботи:

Навчитися аналізувати роботу СУБД та оптимізовувати виконання складних запитів на вибірку даних. Виконати аналіз складних запитів за допомогою директиви EXPLAIN, модифікувати найповільніші запити з метою їх пришвидшення.

За допомогою команди:

```
select ps.indexrelid, ps.relname, ps.indexrelname, pi.indisunique, pi.indisprimary, pi.indcollation
from pg_stat_user_indexes ps left join pg_index pi on ps.indexrelid = pi.indexrelid;
```

Визначимо, які індекси у нас є.

	indexrelid	relname	indexrelname	indisunique	indisprimary	indcollation
1	16401	user	user_pk	☑	☑	0
2	16403	user	user_username_key	☑	☐	100
3	16405	user	user_email_key	☑	☐	100
4	16416	playlist	playlist_pk	☑	☑	0
5	16424	song	song_pk	☑	☑	0
6	16435	genre	genre_pk	☑	☑	0
7	16446	playlist_type	playlist_type_pk	☑	☑	0

Знайдемо усі пісні тривалістю менше 120 секунд і проаналізуємо запит:

```
explain analyze
select *
from song
where duration < 120;
```

Аналіз:

	QUERY PLAN
1	Seq Scan on song (cost=0.00..401.00 rows=2016 width=27) (actual time=0.017..3.678 rows=2016 loops=1)
2	Filter: (duration < 120)
3	Rows Removed by Filter: 18144
4	Planning Time: 0.062 ms
5	Execution Time: 3.790 ms

Запит виконався за 3.790мс. Спробуємо покращити результат, створивши індекс для тривалості таблиці пісень.

```
create index song_duration_idx on song(duration);
```

Перевіримо чи він утворився:

	indexrelid	relname	indexrelname	indisunique	indisprimary	indcollation
1	16401	user	user_pk	☑	☑	0
2	16403	user	user_username_key	☑	☐	100
3	16405	user	user_email_key	☑	☐	100
4	16416	playlist	playlist_pk	☑	☑	0
5	16424	song	song_pk	☑	☑	0
6	16435	genre	genre_pk	☑	☑	0
7	16446	playlist_type	playlist_type_pk	☑	☑	0
8	16568	song	song_duration_idx	☐	☐	0

Запустимо і проаналізуємо запит ще раз.

	QUERY PLAN
1	Bitmap Heap Scan on song (cost=39.91..214.11 rows=2016 width=27) (actual time=0.166..0.495 rows=2016 loops=1)
2	Recheck Cond: (duration < 120)
3	Heap Blocks: exact=149
4	-> Bitmap Index Scan on song_duration_idx (cost=0.00..39.41 rows=2016 width=0) (actual time=0.146..0.146 rows=2016 loops=1)
5	Index Cond: (duration < 120)
6	Planning Time: 0.152 ms
7	Execution Time: 0.601 ms

Як видно з аналізу пошук використовує створений індекс, завдяки цьому час виконання зменшився у 6 разів.

Проте postgresql не завжди використовує створені нами індекси, він оцінює час виконання і вибирає оптимальніший, тому при пошуку пісень тривалістю більше 120 с (це більшість пісень в таблиці) викликається вбудований індекс.

	QUERY PLAN
1	Seq Scan on song (cost=0.00..401.00 rows=17136 width=27) (actual time=0.020..2.955 rows=17136 loops=1)
2	Filter: (duration > 120)
3	Rows Removed by Filter: 3024
4	Planning Time: 0.125 ms
5	Execution Time: 3.456 ms

Проаналізуємо ще один запит.

```
explain analyze
select *
from song s left join playlist_song ps on s.id = ps.song_id
    left join song_genre sg on s.id = sg.song_id
    left join genre g on sg.genre_id = g.id
    left join playlist p on ps.playlist_id = p.id
    left join "user" u on p.user_id = u.id
where s.duration < 120 and artist = 'artist8';
```

Отримуємо інформацію як виконується запит

```
QUERY PLAN
Hash Left Join  (cost=464.29..504.53 rows=151 width=250) (actual time=5.395..6.628 rows=508 loops=1)
  Hash Cond: (p.user_id = u.id)
  ->  Hash Left Join  (cost=462.95..502.69 rows=151 width=209) (actual time=5.340..6.398 rows=508 loops=1)
        Hash Cond: (ps.playlist_id = p.id)
        ->  Hash Left Join  (cost=458.11..497.45 rows=151 width=165) (actual time=5.253..6.149 rows=508 loops=1)
              Hash Cond: (s.id = sg.song_id)
              ->  Hash Right Join  (cost=453.29..490.91 rows=151 width=35) (actual time=5.055..5.788 rows=508 loops=1)
                    Hash Cond: (ps.song_id = s.id)
                    ->  Seq Scan on playlist_song ps  (cost=0.00..31.88 rows=2188 width=8) (actual time=0.039..0.268 rows=2188 loops=1)
                    ->  Hash  (cost=451.40..451.40 rows=151 width=27) (actual time=4.970..4.971 rows=504 loops=1)
                          Buckets: 1024  Batches: 1  Memory Usage: 38kB
                          ->  Seq Scan on song s  (cost=0.00..451.40 rows=151 width=27) (actual time=0.054..4.781 rows=504 loops=1)
                                Filter: ((duration < 120) AND ((artist)::text = 'artist8'::text))
                                Rows Removed by Filter: 19656
              ->  Hash  (cost=3.58..3.58 rows=100 width=130) (actual time=0.168..0.168 rows=100 loops=1)
                    Buckets: 1024  Batches: 1  Memory Usage: 13kB
                    ->  Hash Left Join  (cost=1.18..3.58 rows=100 width=130) (actual time=0.064..0.121 rows=100 loops=1)
                          Hash Cond: (sg.genre_id = g.id)
                          ->  Seq Scan on song_genre sg  (cost=0.00..2.00 rows=100 width=8) (actual time=0.021..0.030 rows=100 loops=1)
                          ->  Hash  (cost=1.08..1.08 rows=8 width=122) (actual time=0.024..0.024 rows=8 loops=1)
                                Buckets: 1024  Batches: 1  Memory Usage: 9kB
                                ->  Seq Scan on genre g  (cost=0.00..1.08 rows=8 width=122) (actual time=0.013..0.016 rows=8 loops=1)
        ->  Hash  (cost=3.26..3.26 rows=126 width=44) (actual time=0.067..0.067 rows=126 loops=1)
              Buckets: 1024  Batches: 1  Memory Usage: 18kB
              ->  Seq Scan on playlist p  (cost=0.00..3.26 rows=126 width=44) (actual time=0.024..0.036 rows=126 loops=1)
  ->  Hash  (cost=1.15..1.15 rows=15 width=41) (actual time=0.035..0.035 rows=15 loops=1)
        Buckets: 1024  Batches: 1  Memory Usage: 10kB
        ->  Seq Scan on "user" u  (cost=0.00..1.15 rows=15 width=41) (actual time=0.018..0.021 rows=15 loops=1)
Planning Time: 1.215 ms
Execution Time: 6.838 ms
```

Execution Time: 6.838 ms

Спробуємо покращити цей результат замінивши left join на inner join.

```
explain analyze
select *
from song s inner join playlist_song ps on s.id = ps.song_id
    inner join song_genre sg on s.id = sg.song_id
    inner join genre g on sg.genre_id = g.id
    inner join playlist p on ps.playlist_id = p.id
    inner join "user" u on p.user_id = u.id
where s.duration < 120 and artist = 'artist8';
```

Результат виконання

	QUERY PLAN
1	Nested Loop (cost=11.73..52.15 rows=1 width=250) (actual time=0.295..0.638 rows=2 loops=1)
2	-> Nested Loop (cost=11.60..51.88 rows=1 width=209) (actual time=0.278..0.620 rows=2 loops=1)
3	-> Nested Loop (cost=11.45..51.72 rows=1 width=165) (actual time=0.271..0.611 rows=2 loops=1)
4	-> Hash Join (cost=11.32..51.42 rows=1 width=43) (actual time=0.261..0.600 rows=2 loops=1)
5	Hash Cond: (ps.song_id = s.id)
6	-> Seq Scan on playlist_song ps (cost=0.00..31.88 rows=2188 width=8) (actual time=0.031..0.231 rows=2188 loops=1)
7	-> Hash (cost=11.31..11.31 rows=1 width=35) (actual time=0.196..0.197 rows=1 loops=1)
8	Buckets: 1024 Batches: 1 Memory Usage: 9kB
9	-> Merge Join (cost=5.61..11.31 rows=1 width=35) (actual time=0.171..0.188 rows=1 loops=1)
10	Merge Cond: (s.id = sg.song_id)
11	-> Index Scan using song_pk on song s (cost=0.29..783.49 rows=151 width=27) (actual time=0.045..0.058 row
12	Filter: ((duration < 120) AND ((artist)::text = 'artist8'::text))
13	Rows Removed by Filter: 75
14	-> Sort (cost=5.32..5.57 rows=100 width=8) (actual time=0.104..0.109 rows=100 loops=1)
15	Sort Key: sg.song_id
16	Sort Method: quicksort Memory: 29kB
17	-> Seq Scan on song_genre sg (cost=0.00..2.00 rows=100 width=8) (actual time=0.036..0.046 rows=100
18	-> Index Scan using genre_pk on genre g (cost=0.13..0.27 rows=1 width=122) (actual time=0.004..0.004 rows=1 loops=2)
19	Index Cond: (id = sg.genre_id)
20	-> Index Scan using playlist_pk on playlist p (cost=0.14..0.17 rows=1 width=44) (actual time=0.003..0.003 rows=1 loops=2)
21	Index Cond: (id = ps.playlist_id)
22	-> Index Scan using user_pk on "user" u (cost=0.14..0.25 rows=1 width=41) (actual time=0.007..0.007 rows=1 loops=2)
23	Index Cond: (id = p.user_id)
24	Planning Time: 1.093 ms
25	Execution Time: 0.724 ms

Execution Time: 0.724 ms

Час виконання скоротився у 9 разів.

Висновок

На лабораторній роботі я навчилась використовувати індекси.