

Напредни бази на податоци

Фаза 3- Индекси и оптимизација на прашалници

Проект: BookNest

Вероника Иванова 231073

Ивана Апостолова 231076

Ивана Дачевска 231077

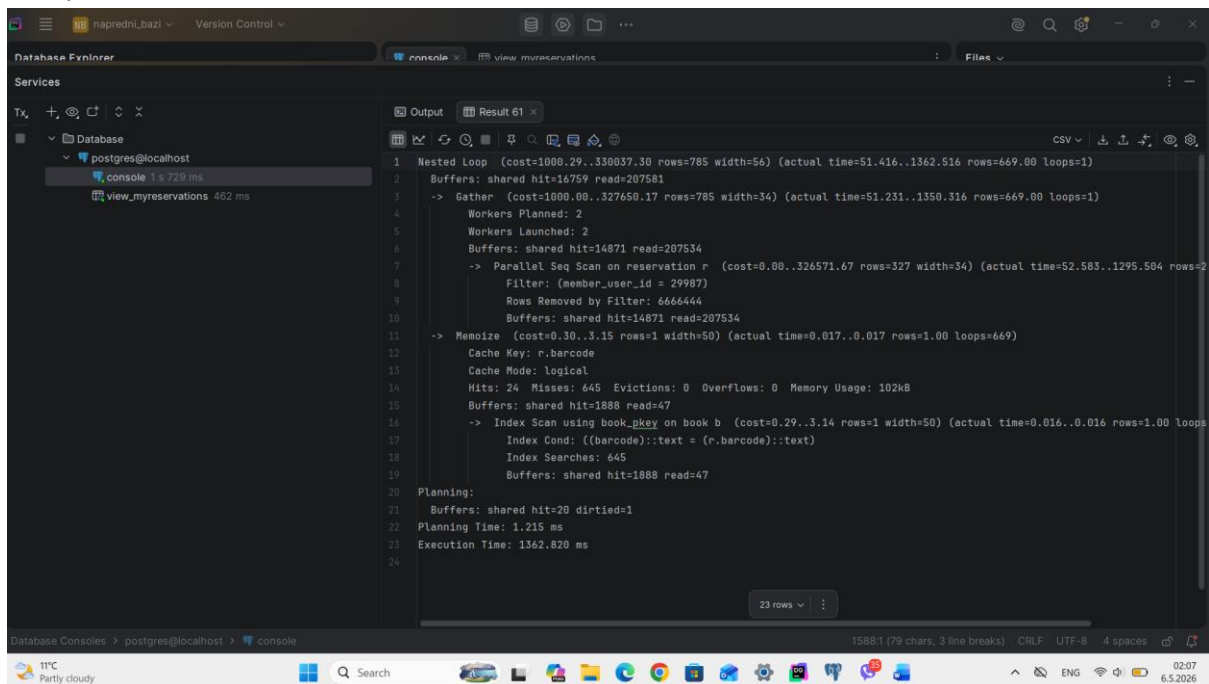
View: view_myreservations

Примарен филтер за погледот view_myreservations ќе биде според **member_user_id**, а дополнително ќе може да се пребарува и според **status** на резервацијата.

Погледот ќе се користи од членовите на библиотеката за преглед на нивните активни, истечени или откажани резервации. За овој поглед ни се важни перформансите, бидејќи често ќе се користи од страна на корисниците.

Иницијалното време за извршување на погледот изнесуваше **1362.820 ms** (1 s 705 ms во конзола). Ова време **не е прифатливо** за апликацијата, па затоа беше направена оптимизација.

Explain Analyze покажа дека PostgreSQL користи **Parallel Sequential Scan** врз табелата reservation, при што се пребарува голем број редови за да се пронајдат резервациите на конкретен член.



```
1  Nested Loop (cost=1000.29..330037.30 rows=785 width=56) (actual time=51.416..1362.516 rows=669.00 loops=1)
2    Buffers: shared hit=16759 read=207581
3    -> Gather (cost=1000.00..327650.17 rows=785 width=34) (actual time=51.231..1350.316 rows=669.00 loops=1)
4      Workers Planned: 2
5      Workers Launched: 2
6      Buffers: shared hit=14871 read=207534
7      -> Parallel Seq Scan on reservation r (cost=0.00..326571.67 rows=327 width=34) (actual time=52.583..1295.504 rows=2
8        Filter: (member_user_id = 29987)
9        Rows Removed by Filter: 6666444
10       Buffers: shared hit=14871 read=207534
11     -> Memoize (cost=0.30..3.15 rows=1 width=50) (actual time=0.017..0.017 rows=1.00 loops=669)
12       Cache Key: r.barcode
13       Cache Mode: logical
14       Hits: 24 Misses: 645 Evictions: 0 Overflows: 0 Memory Usage: 102kB
15       Buffers: shared hit=1888 read=47
16     -> Index Scan using book_pkey on book b (cost=0.29..3.14 rows=1 width=50) (actual time=0.016..0.016 rows=1.00 loops
17       Index Cond: ((barcode)::text = (r.barcode)::text)
18       Index Searches: 645
19       Buffers: shared hit=1888 read=47
20   Planning:
21     Buffers: shared hit=20 dirtied=1
22   Planning Time: 1.215 ms
23   Execution Time: 1362.820 ms
```

```
[2026-05-06 02:06:19] postgres.public> EXPLAIN ANALYZE
SELECT *
FROM view_myreservations
WHERE member_user_id = 29987
[2026-05-06 02:06:21] 23 rows retrieved starting from 1 in 1 s 705 ms (execution: 1 s 371 ms, fetching: 334 ms)
```

Најбавната операција беше филтрирањето по `member_user_id`, па затоа беше додаден следниот индекс:

```
CREATE INDEX idx_reservation_member_user_id
ON reservation(member_user_id);
```

По додавањето на индексот, PostgreSQL започна да користи **Bitmap Index Scan** и **Bitmap Heap Scan** наместо **Parallel Sequential Scan**.

Времето на извршување на **SELECT** прашалникот се намали од 1362.820 ms на 13.400 ms, што претставува **значително подобрување** и е прифатливо за потребите на апликацијата.

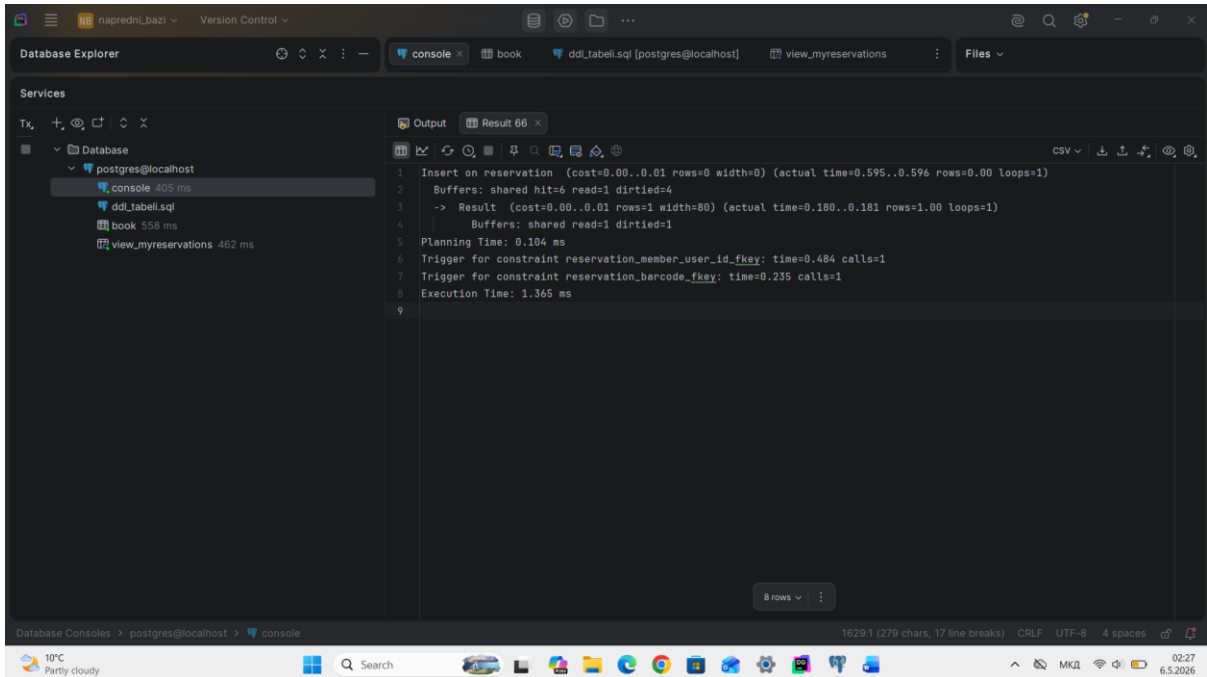
```
1 Hash Join (cost=940.52..3948.75 rows=785 width=56) (actual time=12.389..12.873 rows=670.00 loops=1)
2 Hash Cond: ((r.barcode)::text = (b.barcode)::text)
3 Buffers: shared hit=186 read=534 dirtied=2
4 -> Bitmap Heap Scan on reservation r (cost=10.52..3016.69 rows=785 width=34) (actual time=0.202..0.342 rows=670.00 loops=1)
5 Recheck Cond: (member_user_id = 29987)
6 Heap Blocks: exact=11
7 Buffers: shared hit=11 read=4 dirtied=2
8 -> Bitmap Index Scan on idx_reservation_member_user_id (cost=0.00..10.32 rows=785 width=0) (actual time=0.156..0.156 rows=670.00 loops=1)
9 Index Cond: (member_user_id = 29987)
10 Index Searches: 1
11 Buffers: shared read=4
12 -> Hash (cost=805.00..805.00 rows=10000 width=50) (actual time=11.951..11.952 rows=10000.00 loops=1)
13 Buckets: 16384 Batches: 1 Memory Usage: 937kB
14 Buffers: shared hit=175 read=530
15 -> Seq Scan on book b (cost=0.00..805.00 rows=10000 width=50) (actual time=0.038..0.7345 rows=10000.00 loops=1)
16 Buffers: shared hit=175 read=530
17 Planning:
18 Buffers: shared hit=30 read=1
19 Planning Time: 1.479 ms
20 Execution Time: 13.400 ms
21
```

Времето на **INSERT** операцијата пред индексирање изнесува 12.448 ms, а додека по индексирањето изнесува 1.365 ms.

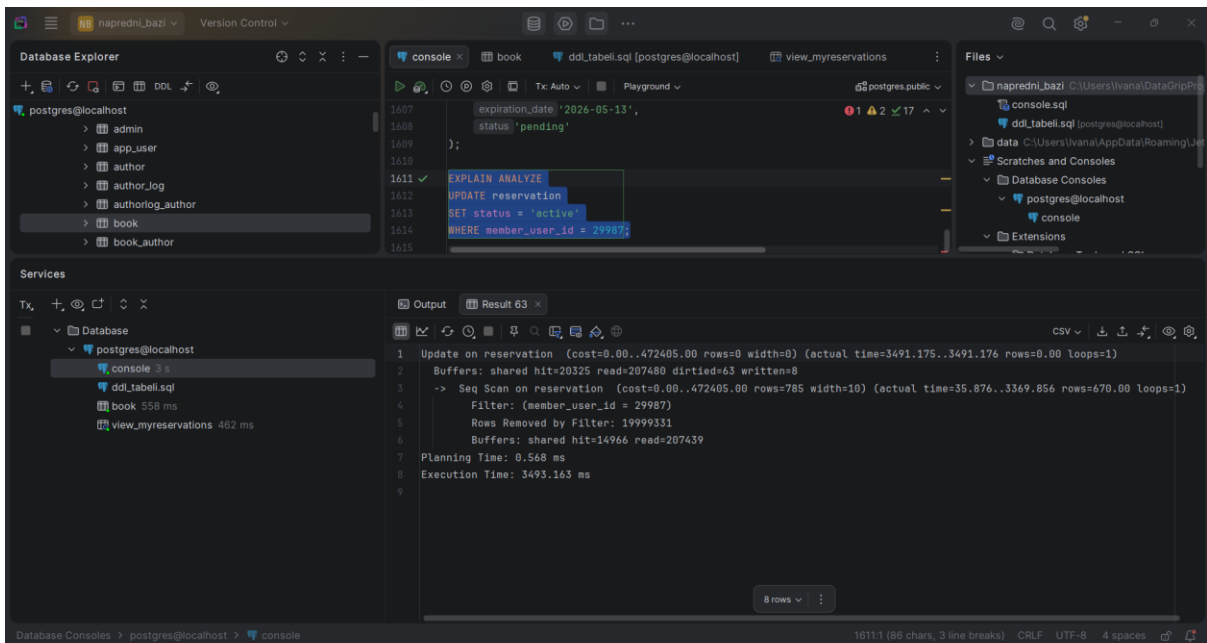
```
1 Insert on reservation (cost=0.00..0.01 rows=0 width=0) (actual time=5.922..5.923 rows=0.00 loops=1)
2 Buffers: shared hit=16 read=10 dirtied=4
3 -> Result (cost=0.00..0.01 rows=1 width=0) (actual time=1.768..1.770 rows=1.00 loops=1)
4 Buffers: shared hit=11 read=4 dirtied=1
5 Planning:
6 Buffers: shared hit=6
7 Planning Time: 0.140 ms
8 Trigger for constraint reservation_member_user_id_fkey: time=5.449 calls=1
9 Trigger for constraint reservation_barcode_fkey: time=0.610 calls=1
10 Execution Time: 12.448 ms
11
```

```
[2026-05-06 02:15:30] 10 rows retrieved starting from 1 in 384 ms (execution: 30 ms, fetching: 354 ms)
```

По индексирање:

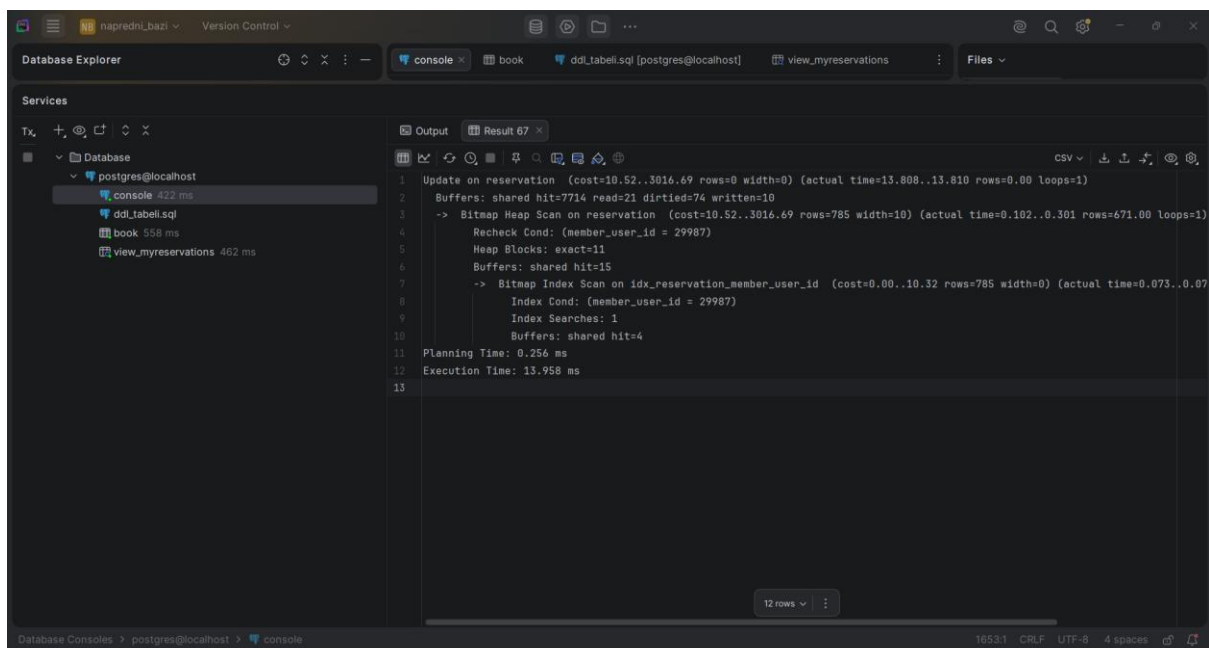


Најзначајно подобрување се забележува кај Update операциите каде времето на извршување пред индексирање изнесува 3493.163 ms, а по индексирање се намалува на 13.958 ms.



```
WHERE member_user_id = 29987
[2026-05-06 02:17:54] 8 rows retrieved starting from 1 in 3 s 867 ms (execution: 3 s 502 ms, fetching: 365 ms)
```

По индексирање:



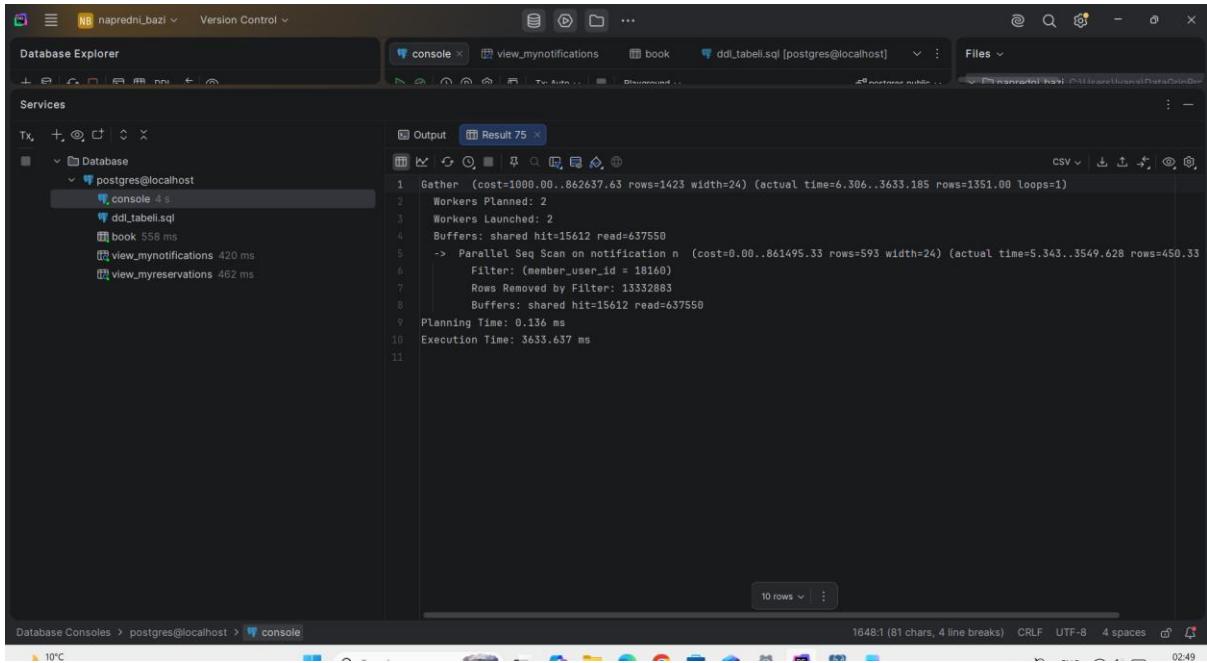
View: view_mynotifications

Примарен филтер за погледот view_mynotifications ќе биде според member_user_id.

Погледот ќе се користи од членовите на библиотеката за преглед на нивните известувања. За овој поглед ни се важни перформансите, бидејќи корисниците често ќе ги отвораат своите нотификации. Иницијалното време за извршување на погледот изнесуваше **3633.637 ms** (3 s 969 ms во конзола). Ова време **не е прифатливо** за апликацијата, па затоа беше направена оптимизација.

Explain Analyze покажа дека PostgreSQL користи **Parallel Sequential Scan** врз табелата notification, при што се пребарува голем број редови за да се пронајдат нотификациите на конкретен член.

```
[2026-05-06 02:48:20] postgres.public> EXPLAIN ANALYZE
SELECT *
FROM view_mynotifications
WHERE member_user_id = 18160
[2026-05-06 02:48:24] 10 rows retrieved starting from 1 in 3 s 969 ms (execution: 3 s 642 ms, fetching: 327 ms)
```

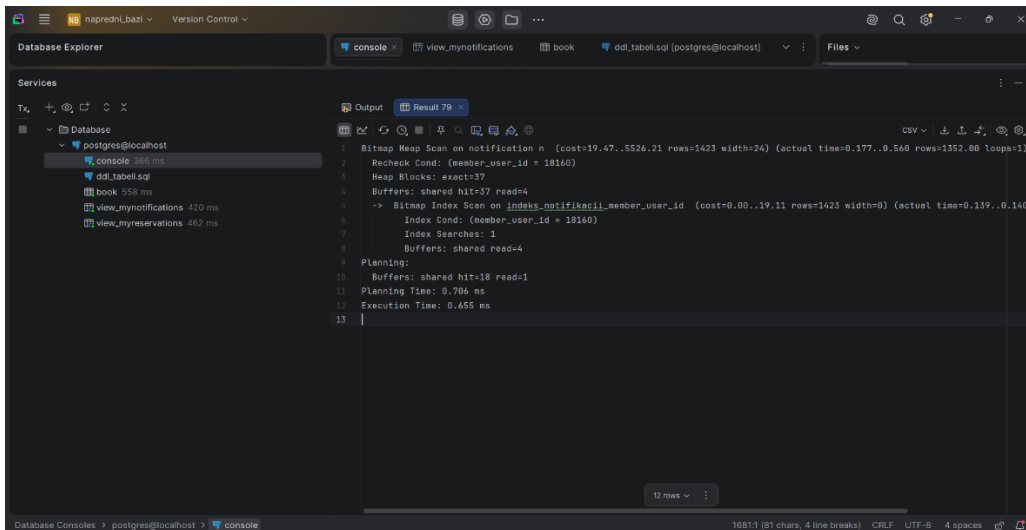


Најбавната операција беше филтрирањето по `member_user_id`, па затоа беше додаден следниот индекс:

```
CREATE INDEX indeks_notifikacii_member_user_id
ON notification(member_user_id);
```

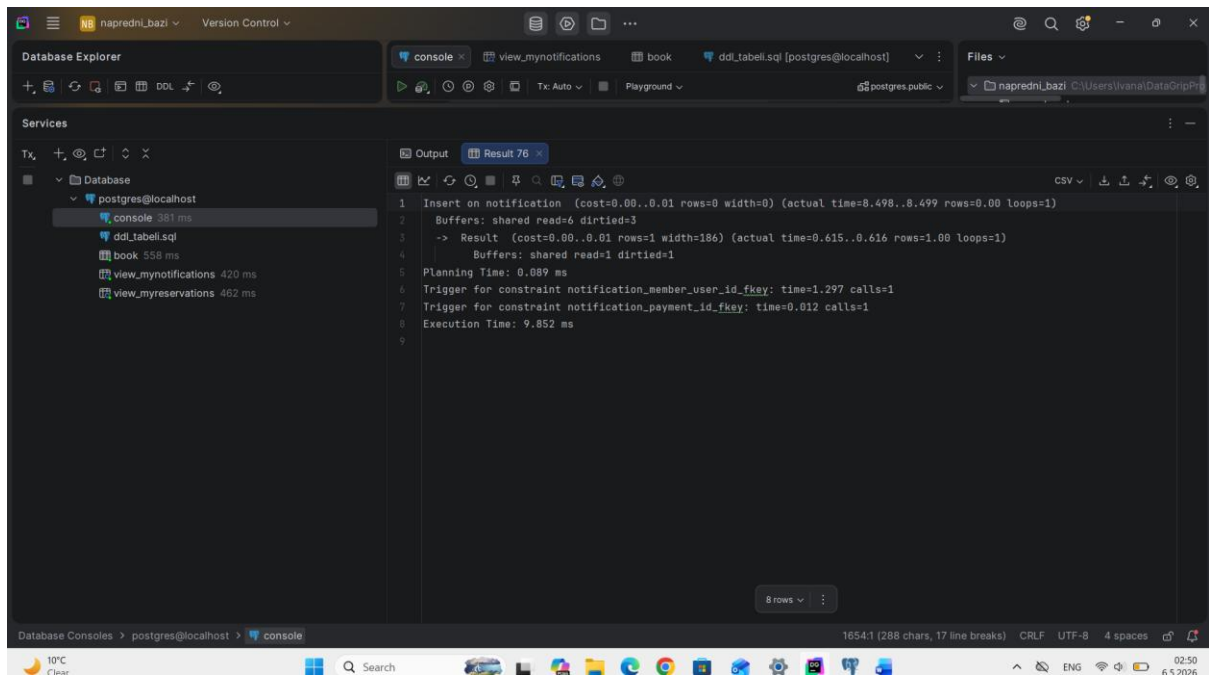
По додавањето на индексот, PostgreSQL започна да користи **Bitmap Index Scan** и **Bitmap Heap Scan** наместо **Parallel Sequential Scan**.

Времето на извршување на **SELECT** прашалникот се намали од 3633.637 ms на 0.655 ms, што претставува значително подобрување и е прифатливо за потребите на апликацијата.



```
[2026-05-06 02:52:10] postgres.public> EXPLAIN ANALYZE
SELECT *
FROM view_mynotifications
WHERE member_user_id = 18160
[2026-05-06 02:52:10] 12 rows retrieved starting from 1 in 344 ms (execution: 9 ms, fetching: 335 ms)
```

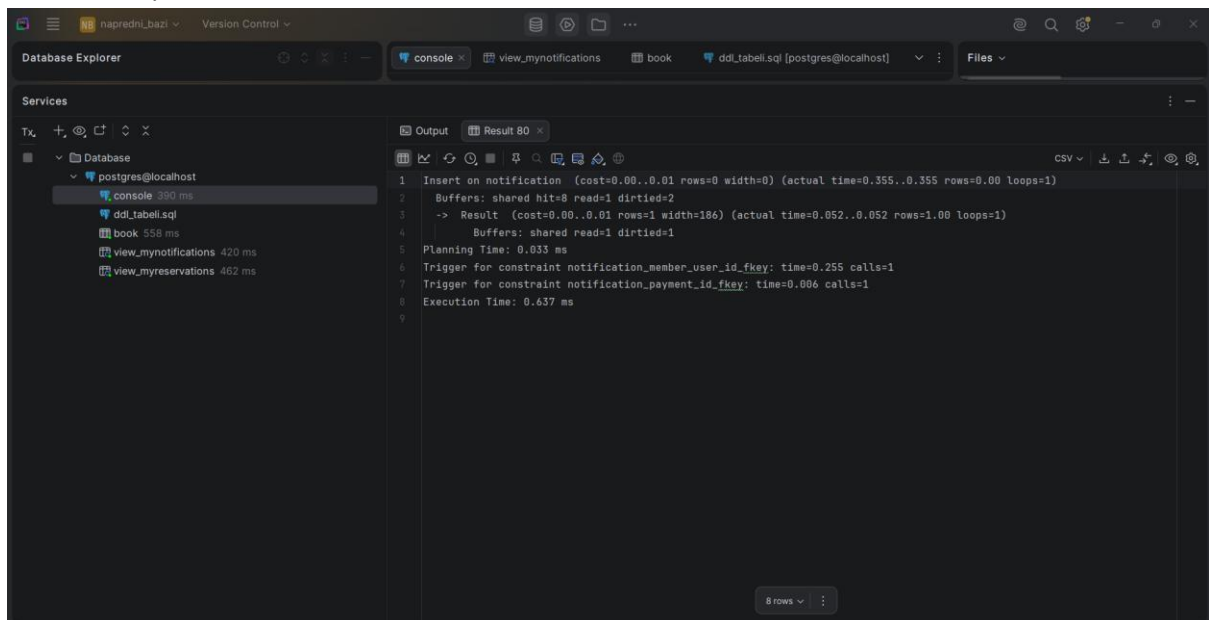
Времето на **INSERT** операцијата пред индексирање изнесува **9.852 ms**, а по индексирање изнесува **0.637 ms**.



The screenshot shows a PostgreSQL console window with the following output for an INSERT statement:

```
1 Insert on notification (cost=0.00..0.01 rows=0 width=0) (actual time=8.498..8.499 rows=0 loops=1)
2 Buffers: shared read=6 dirtied=3
3 -> Result (cost=0.00..0.01 rows=1 width=186) (actual time=0.615..0.616 rows=1 loops=1)
4 Buffers: shared read=1 dirtied=1
5 Planning Time: 0.089 ms
6 Trigger for constraint notification_member_user_id_fkey: time=1.297 calls=1
7 Trigger for constraint notification_payment_id_fkey: time=0.012 calls=1
8 Execution Time: 9.852 ms
9
```

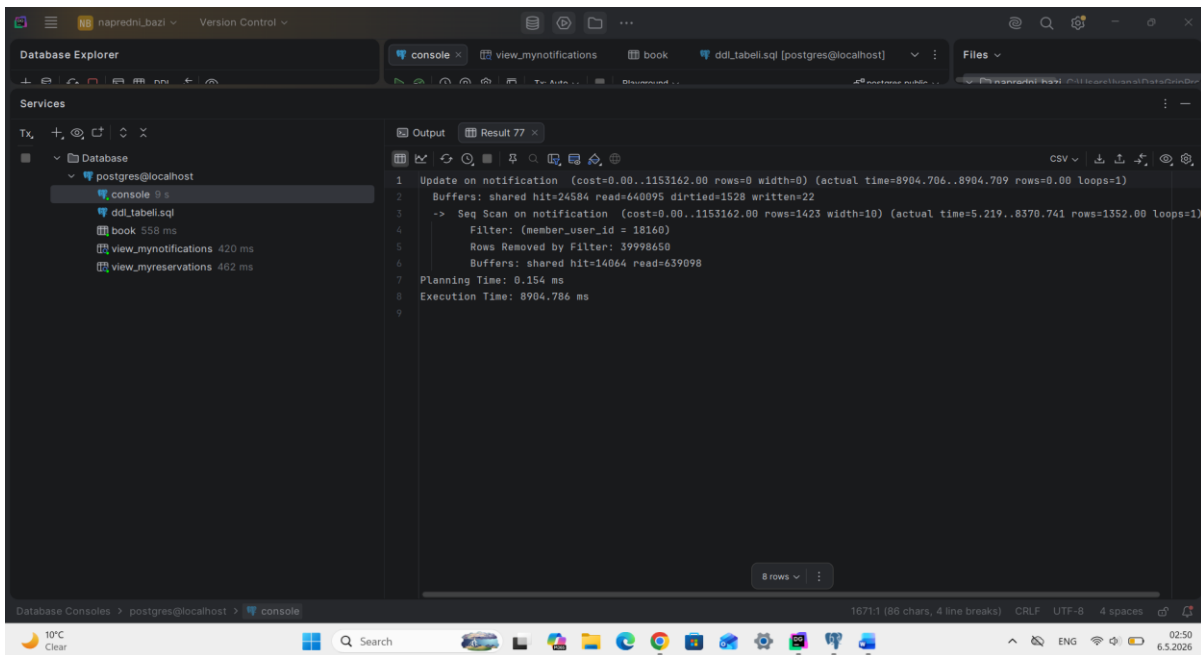
По индексирање:



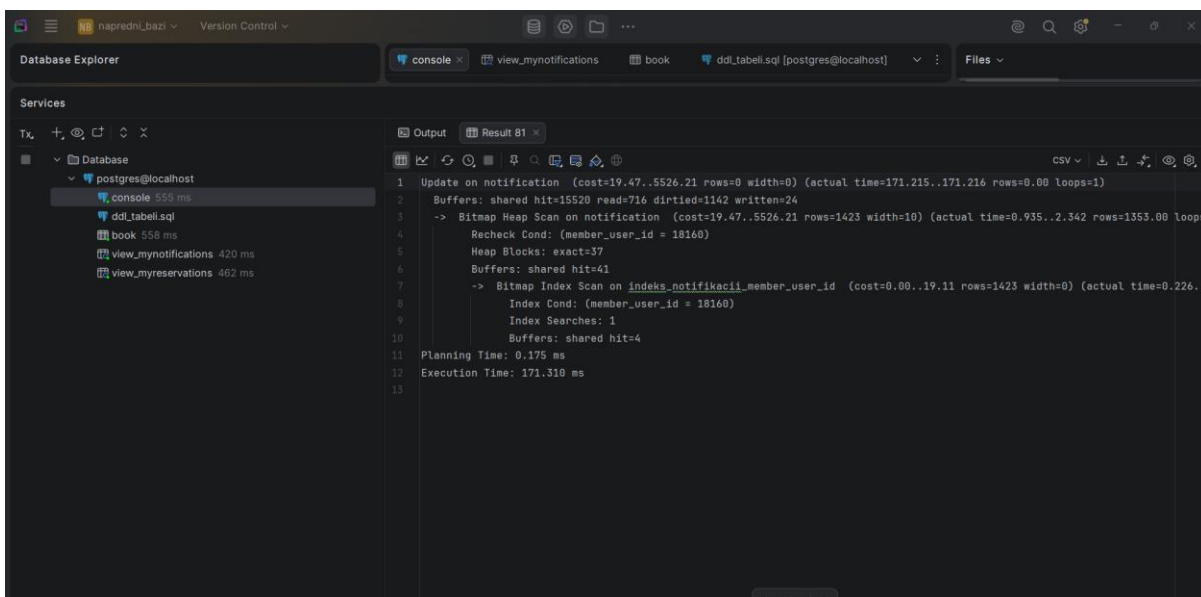
The screenshot shows a PostgreSQL console window with the following output for an UPDATE statement:

```
1 Insert on notification (cost=0.00..0.01 rows=0 width=0) (actual time=0.355..0.355 rows=0 loops=1)
2 Buffers: shared hit=8 read=1 dirtied=2
3 -> Result (cost=0.00..0.01 rows=1 width=186) (actual time=0.052..0.052 rows=1 loops=1)
4 Buffers: shared read=1 dirtied=1
5 Planning Time: 0.033 ms
6 Trigger for constraint notification_member_user_id_fkey: time=0.255 calls=1
7 Trigger for constraint notification_payment_id_fkey: time=0.006 calls=1
8 Execution Time: 0.637 ms
9
```

Времето на Update операцијата пред индексирање изнесува 8904.706 ms, а после индексирање се намалува на 171.310 ms.



По индексирање:



View: view_admin

Примарен филтер за погледот view_admin ќе биде според **user_id** на членот.

Погледот ќе се користи од администраторите за преглед на вкупниот број позајмувања, неплатени казни и вкупниот долг на членовите. За овој поглед се важни перформансите бидејќи често ќе се користи за административен преглед на податоците.

Иницијалното време за извршување на погледот изнесува **11.345 ms**, што е **прифатливо** време за апликацијата.

Explain Analyze покажува дека PostgreSQL користи **GroupAggregate** поради агрегатните функции **COUNT** и **SUM**. Најголем дел од времето се троши на агрегација и групирање на податоците, а не на пребарување.

Дополнително, PostgreSQL веќе користи постоечки **B-tree** индекси преку Index Scan операции врз табелите member и app_user. Овие индекси се автоматски креирани преку **PRIMARY KEY** и **UNIQUE** ограничувањата.

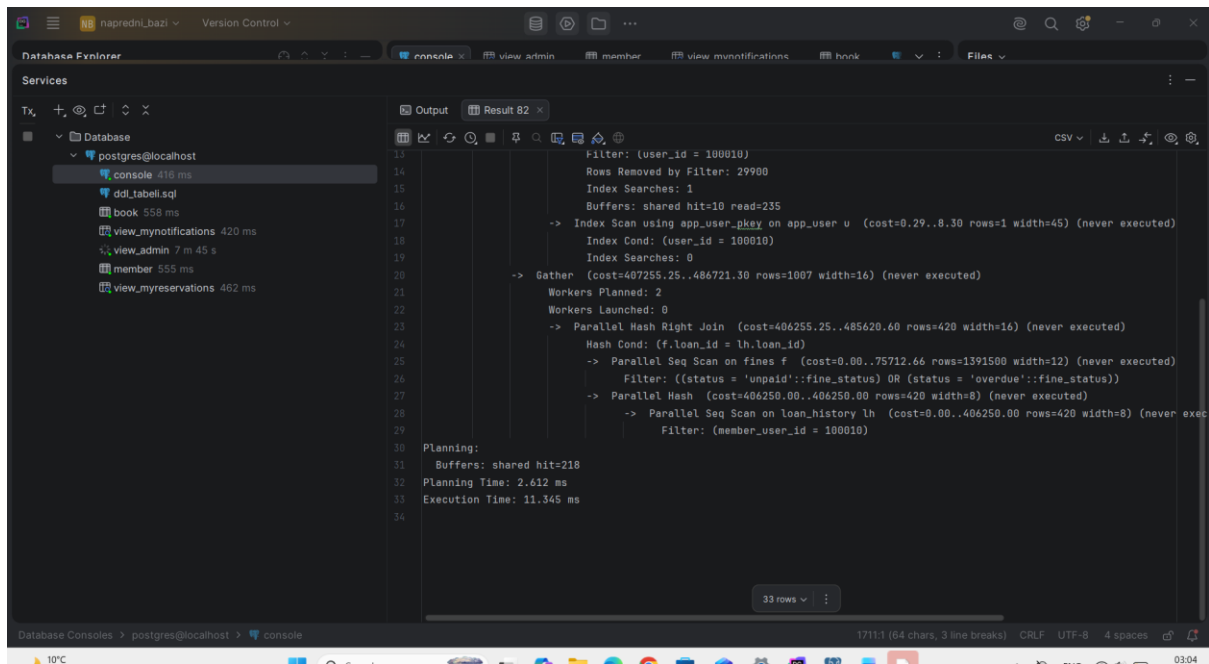
Бидејќи времето на извршување е прифатливо и веќе **се користат постоечки индекси**, **нема потреба од дополнителна оптимизација** или додавање нови индекси.

Поради употребата на агрегатни функции врз големи множества на податоци, дополнителните индекси би имале ограничен ефект врз перформансите, а би создале дополнително оптоварување кај insert и update операциите.

```
[2026-05-06 02:58:57] postgres.public> EXPLAIN ANALYZE
SELECT *
FROM view_admin
WHERE user_id = 100010
[2026-05-06 02:58:58] 33 rows retrieved starting from 1 in 391 ms (execution: 23 ms, fetching: 368 ms)
```

The screenshot shows a database console window with the following execution plan details:

- 1 GroupAggregate (cost=487814.45..487829.57 rows=1 width=73) (actual time=11.189..11.194 rows=0.00 loops=1)
 - 2 " Group Key: u.email, m.member_number"
 - 3 Buffers: shared hit=13 read=235
 - 4 -> Sort (cost=487814.45..487816.97 rows=1007 width=61) (actual time=11.185..11.190 rows=0.00 loops=1)
 - 5 " Sort Key: u.email, m.member_number"
 - 6 Sort Method: quicksort Memory: 25KB
 - 7 Buffers: shared hit=13 read=235
 - 8 -> Nested Loop Left Join (cost=407255.83..487764.22 rows=1007 width=61) (actual time=11.143..11.147 rows=0.00 loops=1)
 - 9 Buffers: shared hit=10 read=235
 - 10 -> Nested Loop (cost=0.57..1032.85 rows=1 width=49) (actual time=11.143..11.144 rows=0.00 loops=1)
 - 11 Buffers: shared hit=10 read=235
 - 12 -> Index Scan using member_member_number_key on member m (cost=0.29..1024.54 rows=1 width=8) (actual time=11.143..11.144 rows=0.00 loops=1)
 - 13 Filter: (user_id = 100010)
 - 14 Rows Removed by Filter: 29900
 - 15 Index Searches: 1
 - 16 Buffers: shared hit=10 read=235
 - 17 -> Index Scan using app_user_pkey on app_user u (cost=0.29..8.30 rows=1 width=45) (never executed)
 - 18 Index Cond: (user_id = 100010)
 - 19 Index Searches: 0
 - 20 -> Gather (cost=407255.25..486721.30 rows=1007 width=16) (never executed)
 - 21 Workers Planned: 2
 - 22 Workers Launched: 0
 - 23 -> Parallel Hash Right Join (cost=406255.25..485620.60 rows=420 width=16) (never executed)
 - 24 Hash Cond: (f.loan_id = lh.loan_id)
 - 25 -> Parallel Seq Scan on fines f (cost=0.00..75712.66 rows=1391500 width=12) (never executed)
 - 26 Filter: ((status = 'overdue'::text) OR (status = 'overdue'::text))
 - 27 -> Parallel Hash (cost=406255.25..485620.60 rows=420 width=8) (never executed)



View: view_bookcatalog

Примарен филтер за погледот view_bookcatalog ќе биде според **насловот на книгата**, а **дополнително** ќе може да се пребарува и според **автор, категорија и жанр**.

Погледот ќе се користи за пребарување и преглед на информации за книги во библиотеката. За овој поглед се важни перформансите бидејќи корисниците често ќе пребаруваат книги.

Иницијалното време за извршување на погледот изнесува **12.745 ms**, што е **прифатливо** време за апликацијата.

Explain Analyze покажува дека PostgreSQL користи **Unique, Sort** и повеќе **Nested Loop** операции **поради DISTINCT и големиот број LEFT JOIN** поврзувања меѓу табелите.

Дополнително, PostgreSQL користи **Sequential Scan** врз табелата book при пребарување со LIKE 'Harry%'. Иако не се користи дополнителен индекс врз колоната title, времето на извршување е доволно мало и прифатливо за апликацијата.

Поради тоа нема потреба од дополнителна оптимизација или додавање нови индекси, бидејќи перформансите на погледот се задоволителни.

Database Explorer

Services

Database

- postgres@localhost
 - console 432 ms
 - ddl_tabeli.sql
 - book 558 ms
 - view_mynotifications 420 ms
 - view_admin 15 m 40 s
 - member 555 ms
 - view_bookcatalog 1 s 2 ms
 - view_myreservations 462 ms

Output

```

1 Unique (cost=1336.21..1336.41 rows=10 width=196) (actual time=12.487..12.519 rows=12.00 loops=1)
2   Buffers: shared hit=922
3   -> Sort (cost=1336.21..1336.23 rows=10 width=196) (actual time=12.483..12.499 rows=12.00 loops=1)
4     Sort Key: b.barcode, b.title, (((e.first_name)::text || ' '::text) || (e.last_name)::text), c.name, g.name, p.name
5     Sort Method: quicksort Memory: 26kB
6     Buffers: shared hit=922
7     -> Nested Loop Left Join (cost=831.59..1336.04 rows=10 width=196) (actual time=5.366..12.373 rows=12.00 loops=1)
8       Buffers: shared hit=922
9       -> Nested Loop Left Join (cost=831.44..1334.29 rows=10 width=103) (actual time=5.338..12.232 rows=12.00 loops=1)
10         Buffers: shared hit=898
11         -> Nested Loop Left Join (cost=831.14..1326.33 rows=5 width=99) (actual time=5.295..12.168 rows=4.00 loops=1)
12           Buffers: shared hit=895
13           -> Nested Loop Left Join (cost=830.87..1324.86 rows=5 width=90) (actual time=5.274..12.127 rows=4.00 loops=1)
14             Buffers: shared hit=883
15             -> Nested Loop Left Join (cost=830.57..1318.31 rows=2 width=86) (actual time=5.209..12.050 rows=2.00 loops=1)
16               Buffers: shared hit=880
17               -> Nested Loop Left Join (cost=830.29..1305.82 rows=2 width=73) (actual time=5.170..12.011 rows=2.00 loops=1)
18                 Buffers: shared hit=877
19                 -> Hash Right Join (cost=830.01..1305.23 rows=2 width=62) (actual time=5.101..12.011 rows=2.00 loops=1)
20                   Hash Cond: ((cb.barcode)::text = (b.barcode)::text)
21                   Buffers: shared hit=865
22                   -> Seq Scan on category_book cb (cost=0.00..409.66 rows=24966 width=10) (actual time=0.000..12.000 rows=2.00 loops=1)
23                     Buffers: shared hit=160
24                   -> Hash (cost=830.00..830.00 rows=1 width=58) (actual time=3.276..3.277 rows=1.00 loops=1)
25                     Buckets: 1024 Batches: 1 Memory Usage: 9kB
26                     -> Seq Scan on book b (cost=0.00..830.00 rows=1 width=58) (actual time=0.000..3.276 rows=1.00 loops=1)
27

```

Database Consoles > postgres@localhost > console 1716:1 (78 chars, 3 line breaks) CRLF UTF-8 4 spaces

10°C Clear

Database Explorer

Services

Database

- postgres@localhost
 - console 432 ms
 - ddl_tabeli.sql
 - book 558 ms
 - view_mynotifications 420 ms
 - view_admin 10 m
 - member 555 ms
 - view_bookcatalog 1 s 2 ms
 - view_myreservations 462 ms

Output

```

55   Index Cond: (author_id = ba.author_id)
56   Index Searches: 4
57   Buffers: shared hit=12
58   -> Memoize (cost=0.30..4.33 rows=2 width=18) (actual time=0.011..0.013 rows=3.00 loops=4)
59     Cache Key: b.barcode
60     Cache Mode: logical
61     Hits: 3 Misses: 1 Evictions: 0 Overflows: 0 Memory Usage: 1kB
62     Buffers: shared hit=3
63     -> Index Only Scan using book_genre_pkey on book_genre bg (cost=0.29..4.32 rows=2 width=18) (actual time=0.000..0.000 rows=2.00 loops=1)
64       Index Cond: (barcode = (b.barcode)::text)
65       Heap Fetches: 0
66       Index Searches: 1
67       Buffers: shared hit=3
68     -> Index Scan using genre_pkey on genre g (cost=0.15..0.17 rows=1 width=82) (actual time=0.006..0.006 rows=1.00 loops=1)
69       Index Cond: (genre_id = bg.genre_id)
70       Index Searches: 12
71       Buffers: shared hit=24
72   Planning:
73     Buffers: shared hit=66
74     Planning Time: 3.690 ms
75     Execution Time: 12.745 ms
76

```

Database Consoles > postgres@localhost > console 1716:1 (78 chars, 3 line breaks) CRLF UTF-8 4 spaces

10°C Clear

WHERE book_circle LIKE '101%'

[2026-05-06 03:09:16] 75 rows retrieved starting from 1 in 388 ms (execution: 25 ms, fetching: 363 ms)

View: view_bookmonthly

Погледот `view_bookmonthly` ќе се користи за прикажување на **најпрегледуваната книга** за секој месец.

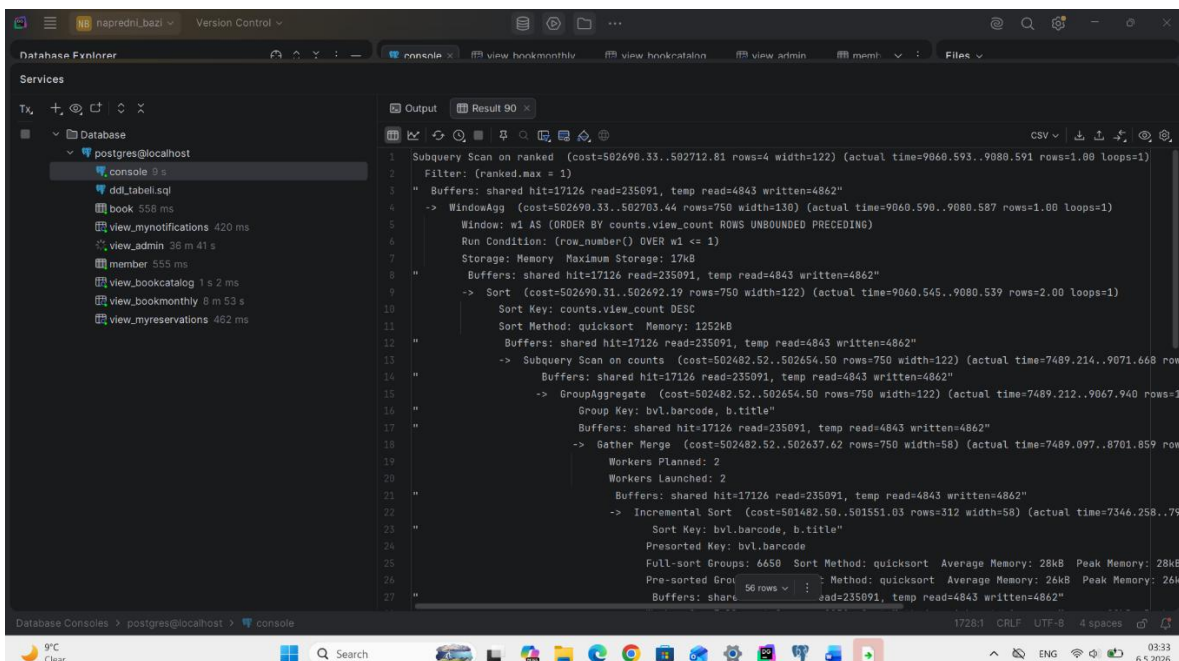
Погледот користи агрегатни функции и аналитички функции, односно **COUNT**, **GROUP BY** и **ROW_NUMBER**, со цел да се пресмета бројот на прегледи по книга и да се одреди книгата со најголем број прегледи за секој месец.

Поради природата на прашалникот и обработката на големо множество податоци од табелата `book_view_log`, PostgreSQL мора да изврши агрегација, сортирање и рангирање на податоците.

Кај вакви аналитички прашалници, дополнителните индекси често имаат ограничен ефект врз перформансите, бидејќи најголем дел од времето се троши на пресметување на агрегатните функции и групирање на резултатите, а не само на пребарување на редови.

```
[2026-05-06 03:29:03] postgres.public> EXPLAIN ANALYZE
SELECT *
FROM view_bookmonthly
WHERE year = 2024
AND month = 4

[2026-05-06 03:29:09] 53 rows retrieved starting from 1 in 5 s 484 ms (execution: 5 s 152 ms, fetching: 332 ms)
```



```
Output Result 90 x
Sort Key: bvl.barcode
Sort Method: external merge Disk: 9912kB
Buffers: shared hit=14909 read=235091, temp read=4843 written=4862"
Worker 0: Sort Method: external merge Disk: 14272kB
Worker 1: Sort Method: external merge Disk: 14560kB
-> Parallel Seq Scan on book_view_log bvl (cost=0.00..500000.00 rows=312
Filter: ((EXTRACT(year FROM view_timestamp) = '2025'::numeric) AND (E
Rows Removed by Filter: 9588494
Buffers: shared hit=14909 read=235091
-> Sort (cost=1469.39..1494.39 rows=10000 width=50) (actual time=83.984..88.111
Sort Key: b.barcode
Sort Method: quicksort Memory: 1082kB
Buffers: shared hit=2115
Worker 0: Sort Method: quicksort Memory: 1082kB
Worker 1: Sort Method: quicksort Memory: 1082kB
-> Seq Scan on book b (cost=0.00..805.00 rows=10000 width=50) (actual tim
Buffers: shared hit=2115
53 Planning:
54 Buffers: shared hit=6
55 Planning Time: 0.660 ms
56 Execution Time: 9086.070 ms
57
56 rows v ...
1728:1 CRLF UTF-8 4 spaces
```

```
[2026-05-06 03:31:54] postgres.public> EXPLAIN ANALYZE
SELECT *
FROM view_bookmonthly
WHERE year = 2025
AND month = 6
[2026-05-06 03:32:03] 56 rows retrieved starting from 1 in 9 s 409 ms (execution: 9 s 92 ms, fetching: 317 ms)
```

View: view_currentloans

Примарен филтер за погледот view_currentloans ќе биде според korisnik_id.

Погледот ќе се користи од членовите на библиотеката за преглед на нивните тековни позајмувања, датумот на позајмување, рокот за враќање и преостанатите денови до враќање на книгите. За овој поглед се важни перформансите бидејќи често ќе се користи од корисниците на системот.

Иницијалното време за извршување на погледот изнесуваше **2018.265 ms** (2s 350ms во конзола), што **не е прифатливо** време за апликацијата.

```

[2026-05-06 03:40:31] postgres.public> EXPLAIN ANALYZE
SELECT *
FROM view_currentloans
WHERE korisnik_id = 26801
[2026-05-06 03:40:33] 24 rows retrieved starting from 1 in 2 s 350 ms (execution: 2 s 29 ms, fetching: 321 ms)

```

```

1 Gather (cost=1000.58..501017.70 rows=4 width=52) (actual time=256.093..2018.177 rows=10.00 loops=1)
2   Workers Planned: 2
3   Workers Launched: 2
4   Buffers: shared hit=142 read=249922
5   -> Nested Loop (cost=0.58..500017.30 rows=2 width=52) (actual time=630.222..1932.878 rows=3.33 loops=3)
6     Buffers: shared hit=142 read=249922
7     -> Nested Loop (cost=0.29..500016.62 rows=2 width=26) (actual time=630.127..1932.638 rows=3.33 loops=3)
8       Buffers: shared hit=110 read=249922
9       -> Parallel Seq Scan on loan_history lh (cost=0.00..500000.00 rows=2 width=16) (actual time=628.454..1927.698
10         Filter: ((return_date IS NULL) AND (status = 'borrowed'::loan_status) AND (member_user_id = 26801) AND (d
11         Rows Removed by Filter: 9999997
12         Buffers: shared hit=94 read=249906
13         -> Index Scan using book_copy_pkey on book_copy bc (cost=0.29..8.31 rows=1 width=18) (actual time=1.467..1.46
14           Index Cond: (copy_id = lh.copy_id)
15           Index Searches: 10
16           Buffers: shared hit=16 read=16
17         -> Index Scan using book_pkey on book b (cost=0.29..0.33 rows=1 width=50) (actual time=0.060..0.061 rows=1.00 loops
18           Index Cond: ((barcode)::text = (bc.barcode)::text)
19           Index Searches: 10
20           Buffers: shared hit=2
21 Planning:
22   Buffers: shared hit=75
23 Planning Time: 1.391 ms
24 Execution Time: 2018.265 ms
25

```

Explain Analyze покажува дека PostgreSQL користи **Parallel Sequential Scan** врз табелата `loan_history`, при што се пребаруваат милиони редови за да се пронајдат активните позајмувања на конкретен korisnik. Najgolem del od времето se troshi na prebaruвање niz celata tabela `loan_history`.

Poradi toa беше kreiran partial B-tree indeks:

```

CREATE INDEX indeks_aktivni_pozajmuvanja
ON loan_history(member_user_id, due_date, copy_id)
WHERE return_date IS NULL
AND status = 'borrowed';

```

Ovoj indeks gi sodrzi samo aktivnite pozajmuvaња, odnosno redovite kaj кои knigite ne se vrateni i statusot e borrowed. So toa indeksot e pomal i poeffikasen od obichen indeks vrz celata tabela.

Po dodavaњeto na indeksot PostgreSQL započna da koristi **Index Scan** vrz `indeks_aktivni_pozajmuvanja` namesto Parallel Sequential Scan, a времето na izvrsuвање se namali na **3.842 ms**, što pretstavuva značitelno podobruвање i e priфatljivo za aplikacijata.

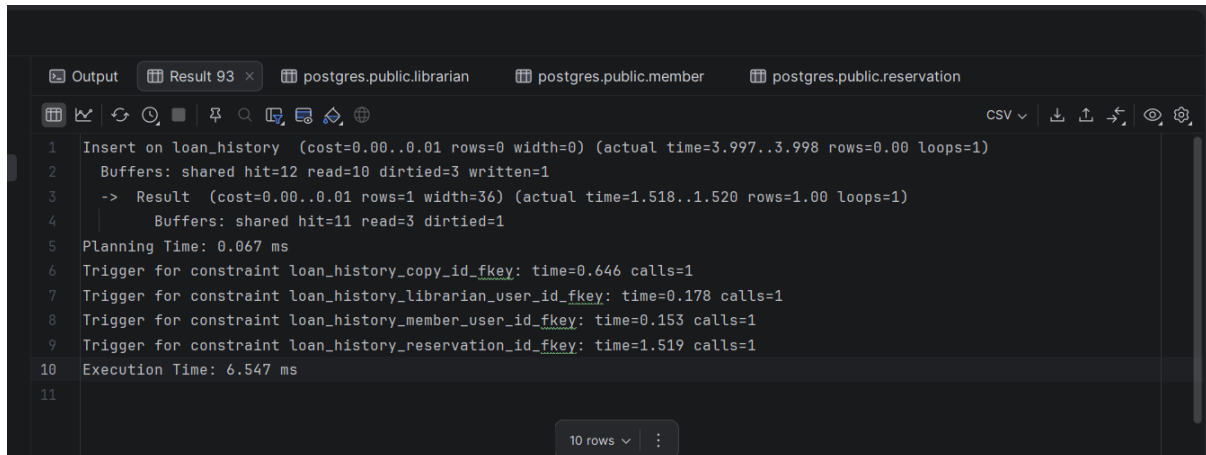
```

[2026-05-06 04:06:30] postgres.public> EXPLAIN ANALYZE
SELECT *
FROM view_currentloans
WHERE korisnik_id = 15622
[2026-05-06 04:06:30] 20 rows retrieved starting from 1 in 344 ms (execution: 12 ms, fetching: 332 ms)

```

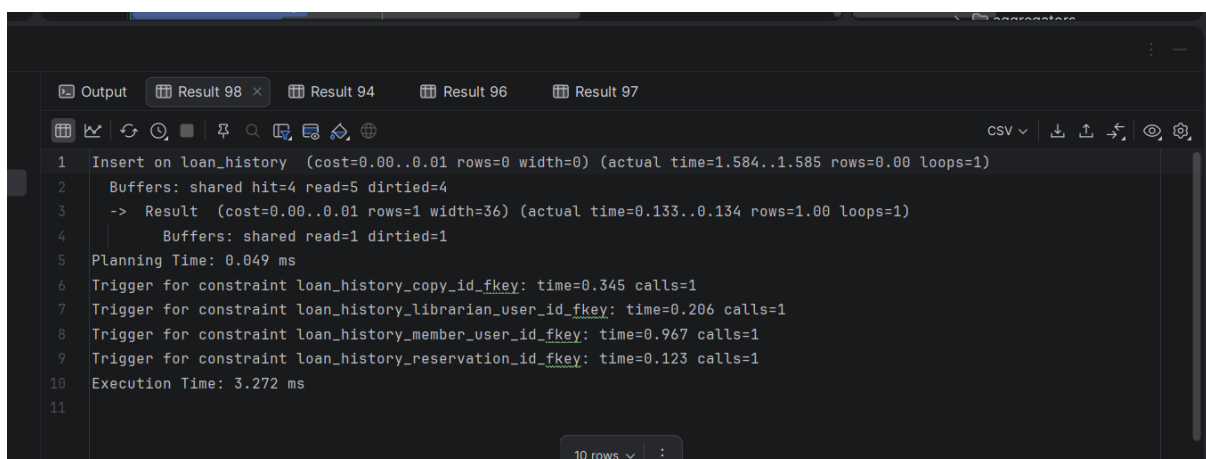
Дополнително беа тестирани и **INSERT** и **UPDATE** операции врз табелата `loan_history`. INSERT операцијата се изврши за **3.272 ms**, додека UPDATE операцијата се изврши за **277.376 ms**. Explain Analyze покажува дека PostgreSQL користи Bitmap Index Scan врз индексот `indeks_aktivni_pozajmuvanja` при извршување на UPDATE операцијата.

Insert операција без индекс:



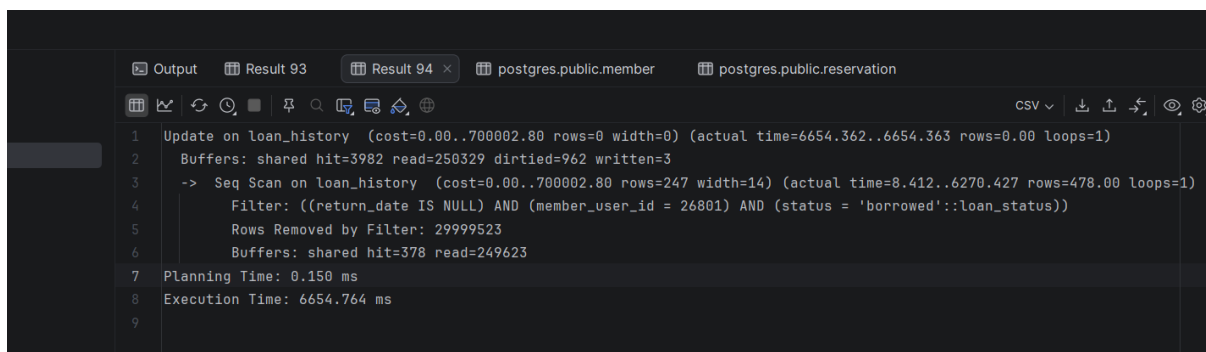
```
Output Result 93 postgres.public.librarian postgres.public.member postgres.public.reservation
1 Insert on loan_history (cost=0.00..0.01 rows=0 width=0) (actual time=3.997..3.998 rows=0.00 loops=1)
2 Buffers: shared hit=12 read=10 dirtied=3 written=1
3 -> Result (cost=0.00..0.01 rows=1 width=36) (actual time=1.518..1.520 rows=1.00 loops=1)
4 Buffers: shared hit=11 read=3 dirtied=1
5 Planning Time: 0.067 ms
6 Trigger for constraint loan_history_copy_id_fkey: time=0.646 calls=1
7 Trigger for constraint loan_history_librarian_user_id_fkey: time=0.178 calls=1
8 Trigger for constraint loan_history_member_user_id_fkey: time=0.153 calls=1
9 Trigger for constraint loan_history_reservation_id_fkey: time=1.519 calls=1
10 Execution Time: 6.547 ms
11
```

Insert операција со индекс:



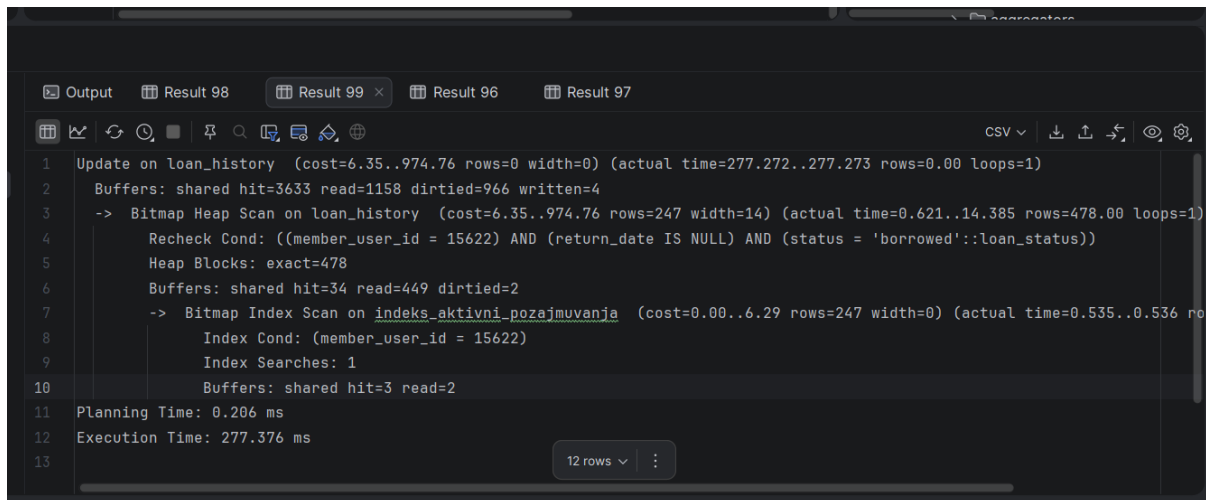
```
Output Result 98 Result 94 Result 96 Result 97
1 Insert on loan_history (cost=0.00..0.01 rows=0 width=0) (actual time=1.584..1.585 rows=0.00 loops=1)
2 Buffers: shared hit=4 read=5 dirtied=4
3 -> Result (cost=0.00..0.01 rows=1 width=36) (actual time=0.133..0.134 rows=1.00 loops=1)
4 Buffers: shared read=1 dirtied=1
5 Planning Time: 0.049 ms
6 Trigger for constraint loan_history_copy_id_fkey: time=0.345 calls=1
7 Trigger for constraint loan_history_librarian_user_id_fkey: time=0.206 calls=1
8 Trigger for constraint loan_history_member_user_id_fkey: time=0.967 calls=1
9 Trigger for constraint loan_history_reservation_id_fkey: time=0.123 calls=1
10 Execution Time: 3.272 ms
11
```

Update операција без индекс:



```
Output Result 93 Result 94 postgres.public.member postgres.public.reservation
1 Update on loan_history (cost=0.00..700002.80 rows=0 width=0) (actual time=6654.362..6654.363 rows=0.00 loops=1)
2 Buffers: shared hit=3982 read=250329 dirtied=962 written=3
3 -> Seq Scan on loan_history (cost=0.00..700002.80 rows=247 width=14) (actual time=8.412..6270.427 rows=478.00 loops=1)
4 Filter: ((return_date IS NULL) AND (member_user_id = 26801) AND (status = 'borrowed'::loan_status))
5 Rows Removed by Filter: 29999523
6 Buffers: shared hit=378 read=249623
7 Planning Time: 0.150 ms
8 Execution Time: 6654.764 ms
9
```

Update операција без индекс:



```
1 Update on loan_history (cost=6.35.974.76 rows=0 width=0) (actual time=277.272..277.273 rows=0.00 loops=1)
2   Buffers: shared hit=3633 read=1158 dirtied=966 written=4
3   -> Bitmap Heap Scan on loan_history (cost=6.35.974.76 rows=247 width=14) (actual time=0.621..14.385 rows=478.00 loops=1)
4     Recheck Cond: ((member_user_id = 15622) AND (return_date IS NULL) AND (status = 'borrowed'::loan_status))
5     Heap Blocks: exact=478
6     Buffers: shared hit=34 read=449 dirtied=2
7     -> Bitmap Index Scan on indeks_aktivni_pozajmuvanja (cost=0.00.6.29 rows=247 width=0) (actual time=0.535..0.536 rows=478.00 loops=1)
8       Index Cond: (member_user_id = 15622)
9       Index Searches: 1
10      Buffers: shared hit=3 read=2
11 Planning Time: 0.206 ms
12 Execution Time: 277.376 ms
```

Иако индексот внесува мало дополнително оптоварување кај write операциите поради неговото ажурирање, тоа е прифатливо во споредба со значителното подобрување на SELECT прашалниците и намалувањето на времето на пребарување од неколку секунди на само неколку милисекунди.

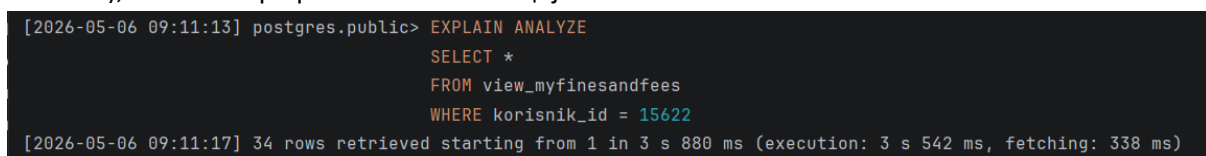
7.view_myfinesandfees

View: view_myfinesandfees

Примарен филтер за погледот view_myfinesandfees ќе биде според korisnik_id.

Погледот ќе се користи од членовите на библиотеката за преглед на нивните неплатени и задоцнети казни. За овој поглед се важни перформансите бидејќи корисниците често ќе ги проверуваат своите долгови и рокови за плаќање.

Иницијалното време за извршување на погледот изнесуваше **3523.558 ms**(3 s 880 ms во конзола), што не е прифатливо за апликацијата.



```
[2026-05-06 09:11:13] postgres.public> EXPLAIN ANALYZE
SELECT *
FROM view_myfinesandfees
WHERE korisnik_id = 15622
[2026-05-06 09:11:17] 34 rows retrieved starting from 1 in 3 s 880 ms (execution: 3 s 542 ms, fetching: 338 ms)
```

Explain Analyze покажува дека PostgreSQL користи **Parallel Sequential Scan** врз табелите fines и loan_history, при што се пребарува голем број редови за да се пронајдат казните за конкретен корисник. Најголем дел од времето се троши на филтрирање и поврзување на податоците преку loan_id.

```

1 Gather (cost=407266.33..480965.97 rows=123 width=52) (actual time=2624.154..3522.448 rows=137.00 loops=1)
2   Workers Planned: 2
3   Workers Launched: 2
4   Buffers: shared hit=12364 read=276685 dirtied=958 written=1
5   -> Nested Loop (cost=406266.33..479953.67 rows=51 width=52) (actual time=2554.430..3419.540 rows=45.67 loops=3)
6     Buffers: shared hit=12364 read=276685 dirtied=958 written=1
7     -> Nested Loop (cost=486266.04..479936.70 rows=51 width=30) (actual time=2554.038..3408.878 rows=45.67 loops=3)
8       Buffers: shared hit=12027 read=276609 dirtied=958 written=1
9       -> Parallel Hash Join (cost=406265.75..479750.93 rows=51 width=20) (actual time=2553.166..3386.868 rows=45.67 loops=3)
10         Hash Cond: (f.loan_id = lh.loan_id)
11         Buffers: shared hit=11839 read=276384 dirtied=958 written=1
12         -> Parallel Seq Scan on fines f (cost=0.00..69463.05 rows=1532238 width=16) (actual time=1.580..557.808 rows=45.67 loops=3)
13           Filter: (status = ANY ('{unpaid,overdue} '::fine_status[]))
14           Rows Removed by Filter: 775031
15           Buffers: shared hit=14 read=38201 written=1
16         -> Parallel Hash (cost=406260.50..406260.50 rows=420 width=12) (actual time=2540.114..2540.115 rows=325.00 loops=3)
17           Buckets: 1024 Batches: 1 Memory Usage: 104kB
18           Buffers: shared hit=11825 read=238183 dirtied=958
19         -> Parallel Seq Scan on loan_history lh (cost=0.00..406260.50 rows=420 width=12) (actual time=20.902..2539.154 rows=325.00 loops=3)
20           Filter: (member_user_id = 15622)
21           Rows Removed by Filter: 9999676
22           Buffers: shared hit=11825 read=238183 dirtied=958
23         -> Index Scan using book_copy_pkey on book_copy bc (cost=0.29..3.64 rows=1 width=18) (actual time=0.469..0.469 rows=1.00 loops=137)
24           Index Cond: (copy_id = lh.copy_id)
25           Index Searches: 137
26           Buffers: shared hit=188 read=225
27         -> Index Scan using book_pkey on book b (cost=0.29..0.33 rows=1 width=50) (actual time=0.220..0.220 rows=1.00 loops=137)
28           Index Cond: ((barcode)::text = (bc.barcode)::text)
29           Index Searches: 137
30           Buffers: shared hit=337 read=76
31 Planning:
32   Buffers: shared hit=500
33 Planning Time: 4.602 ms
34 Execution Time: 3523.558 ms

```

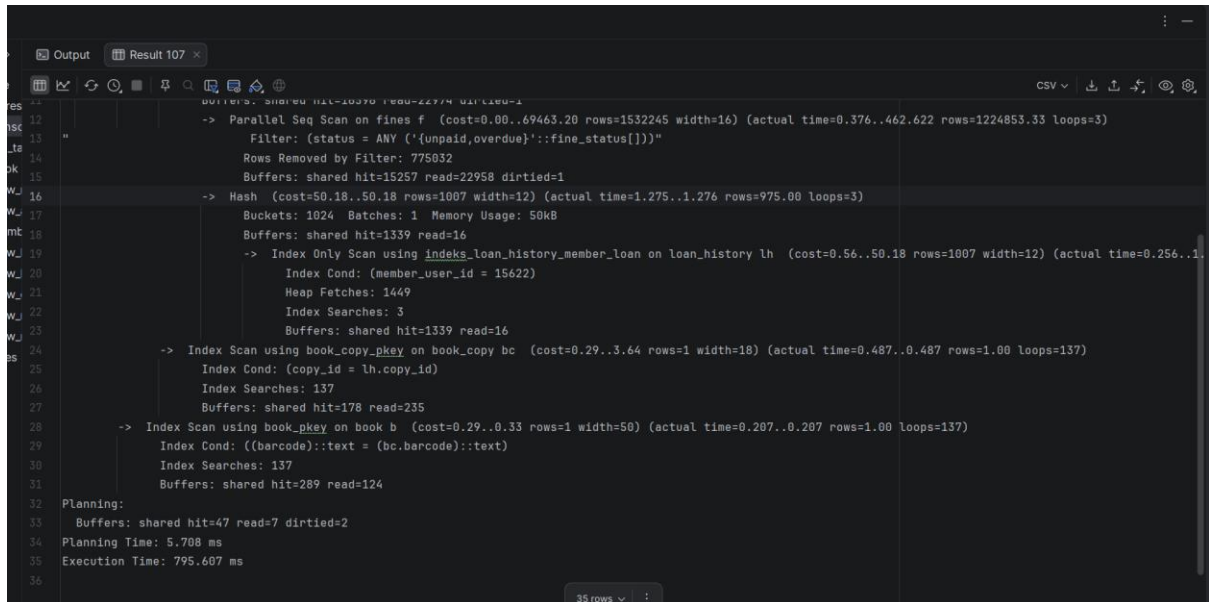
Поради тоа беше креиран следниот **B-tree индекс**:

```
CREATE INDEX indeks_loan_history_member_loan
ON loan_history(member_user_id, loan_id, copy_id);
```

Овој индекс овозможува побрзо пронаоѓање на позајмувањата за конкретен корисник и поефикасно поврзување со табелата fines.

По додавањето на индексот PostgreSQL започна да користи **Index Only Scan** врз индекс_loan_history_member_loan, а времето на извршување се намали од **3523.558 ms** на

795.607 ms, што претставува значително подобрување на перформансите.

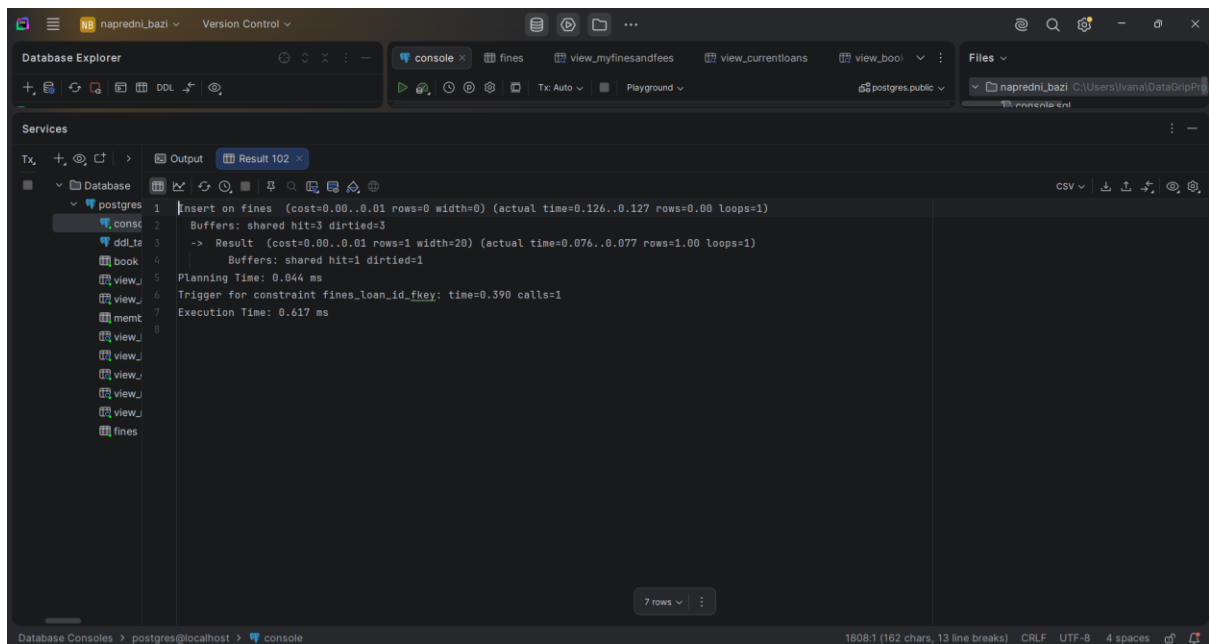


```
11 Buffers: shared hit=10070 read=22974 dirtied=1
12
13 -> Parallel Seq Scan on fines f (cost=0.00..69463.20 rows=1532245 width=16) (actual time=0.376..462.622 rows=1224853.33 loops=3)
14   Filter: (status = ANY ('{unpaid,overdue}'::fine_status[]))
15   Rows Removed by Filter: 775032
16   Buffers: shared hit=15257 read=22958 dirtied=1
17
18 -> Hash (cost=50.18..50.18 rows=1007 width=12) (actual time=1.275..1.276 rows=975.00 loops=3)
19   Buckets: 1024 Batches: 1 Memory Usage: 50kB
20   Buffers: shared hit=1339 read=16
21
22 -> Index Only Scan using indeks_loan_history_member_loan on loan_history lh (cost=0.56..50.18 rows=1007 width=12) (actual time=0.256..1.111 rows=975.00 loops=3)
23   Index Cond: (member_user_id = 15622)
24   Heap Fetches: 1449
25   Index Searches: 3
26   Buffers: shared hit=1339 read=16
27
28 -> Index Scan using book_copy_pkey on book_copy bc (cost=0.29..3.64 rows=1 width=18) (actual time=0.487..0.487 rows=1.00 loops=137)
29   Index Cond: (copy_id = lh.copy_id)
30   Index Searches: 137
31   Buffers: shared hit=178 read=235
32
33 -> Index Scan using book_pkey on book b (cost=0.29..0.33 rows=1 width=50) (actual time=0.207..0.207 rows=1.00 loops=137)
34   Index Cond: ((barcode)::text = (bc.barcode)::text)
35   Index Searches: 137
36   Buffers: shared hit=289 read=124
37
38 Planning:
39   Buffers: shared hit=47 read=7 dirtied=2
40 Planning Time: 5.708 ms
41 Execution Time: 795.607 ms
```

Иако PostgreSQL сè уште користи Parallel Sequential Scan врз табелата fines, времето на извршување е значително намалено поради побрзото пребарување во loan_history.

Дополнително беа тестирани и INSERT и UPDATE операции врз табелата fines. Индексот внесува мало дополнително оптоварување кај write операциите поради неговото ажурирање, но тоа е прифатливо во споредба со значителното подобрување на SELECT прашалниците.

Insert операција без индекс:



```
1 Insert on fines (cost=0.00..0.01 rows=0 width=0) (actual time=0.126..0.127 rows=0.00 loops=1)
2   Buffers: shared hit=3 dirtied=3
3
4 -> Result (cost=0.00..0.01 rows=1 width=20) (actual time=0.076..0.077 rows=1.00 loops=1)
5   Buffers: shared hit=1 dirtied=1
6
7 Planning Time: 0.044 ms
8 Trigger for constraint fines_loan_id_fkey: time=0.390 calls=1
9 Execution Time: 0.617 ms
```

Insert операција со индекс:

```
Services
Tx, +, -, > Output Result 108 x
Database
postgres
  consc
  ddl_ta
  book
  view_
  view_
  memt
  view_
  view_
  view_
  view_
  view_
  view_
  fines
1 Insert on fines (cost=0.00..0.01 rows=0 width=0) (actual time=0.575..0.575 rows=0.00 loops=1)
2 Buffers: shared hit=1 read=7 dirtied=2
3 -> Result (cost=0.00..0.01 rows=1 width=20) (actual time=0.135..0.136 rows=1.00 loops=1)
4 Buffers: shared read=1 dirtied=1
5 Planning Time: 0.054 ms
6 Trigger for constraint fines_loan_id_fkey: time=0.442 calls=1
7 Execution Time: 1.057 ms
8
```

Update операција без индекс:

```
Database Explorer
napredni_bazi Version Control
Database Explorer console fines view_myfinesandfees view_currentloans view_book Files
postgres@localhost console postgres.public napredni_bazi C:\Users\Ivana\DataGrpPr console.sql ddl_tabeli.sql postgres@localhost
Services
Tx, +, -, > Output Result 103 x
Database
postgres
  consc
  ddl_ta
  book
  view_
  memt
  view_
  view_
  view_
  view_
  view_
  view_
  fines
1 Update on fines (cost=0.00..113210.32 rows=0 width=0) (actual time=1081.687..1081.688 rows=0.00 loops=1)
2 Buffers: shared hit=308 read=37922 dirtied=3
3 -> Seq Scan on fines (cost=0.00..113210.32 rows=1 width=10) (actual time=0.090..1080.057 rows=2.00 loops=1)
4 Filter: (loan_id = 31002460)
5 Rows Removed by Filter: 5999653
6 Buffers: shared hit=296 read=37919
7 Planning Time: 0.203 ms
8 Execution Time: 1082.442 ms
9
```

Update операција со индекс:

```
Output Result 109 x
Database
postgres
  consc
  ddl_ta
  book
  view_
  memt
  view_
  view_
  view_
  view_
  view_
  fines
1 Update on fines (cost=0.00..128209.82 rows=0 width=0) (actual time=948.250..948.251 rows=0.00 loops=1)
2 Buffers: shared hit=15373 read=22842 dirtied=1
3 -> Seq Scan on fines (cost=0.00..128209.82 rows=1 width=10) (actual time=948.246..948.247 rows=0.00 loops=1)
4 Filter: ((loan_id = 31002461) AND (status = 'paid')::fine_status)
5 Rows Removed by Filter: 5999656
6 Buffers: shared hit=15373 read=22842 dirtied=1
7 Planning Time: 0.100 ms
8 Execution Time: 948.311 ms
9
```

View: view_upcomingevents

Погледот `view_upcomingevents` се користи за приказ на **сите претстојни настани** во библиотеката. Во погледот се прикажуваат основните информации за настанот како наслов,

При пребарувањето PostgreSQL директно ги пронаоѓа сите записи за конкретниот корисник преку `member_user_id`, наместо да ја пребарува целата табела `loan_history`. На овој начин значително се намалува бројот на редови кои треба да се обработат.

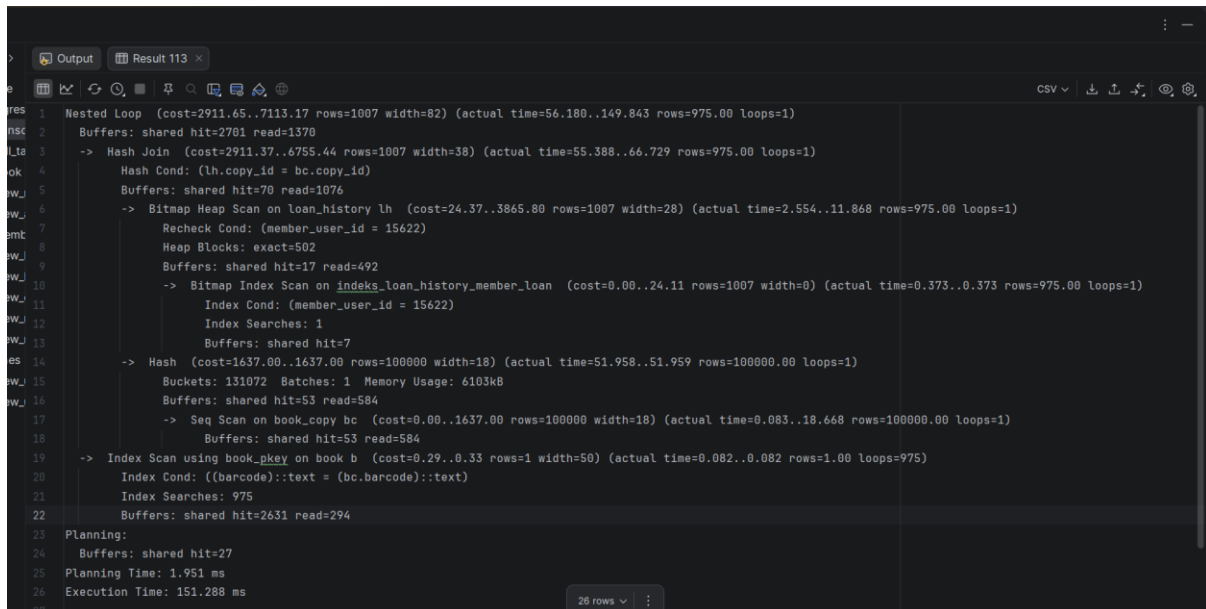
Во Explain Analyze може да се забележи:

**Bitmap Index Scan on indeks_loan_history_member_loan
Index Cond: (member_user_id = 15622)**

што покажува дека PostgreSQL **успешно го користи индексот** за филтрирање на податоците.

По пронаоѓањето на потребните записи PostgreSQL користи **Bitmap Heap Scan** за читање на соодветните блокови од табелата, а потоа **Hash Join** со табелата `book_copy` и **Index Scan** врз табелата `book` преку примарниот клуч.

Иако во извршувањето постои **Sequential Scan** врз табелата `book_copy`, времето на извршување изнесува **151.288 ms**, што е прифатливо за апликацијата.



```
1 Nested Loop (cost=2911.65..7113.17 rows=1007 width=82) (actual time=56.180..149.843 rows=975.00 loops=1)
2   Buffers: shared hit=2701 read=1370
3   -> Hash Join (cost=2911.37..6755.44 rows=1007 width=38) (actual time=55.388..66.729 rows=975.00 loops=1)
4     Hash Cond: (lh.copy_id = bc.copy_id)
5     Buffers: shared hit=70 read=1076
6     -> Bitmap Heap Scan on loan_history lh (cost=24.37..3865.80 rows=1007 width=28) (actual time=2.554..11.868 rows=975.00 loops=1)
7       Recheck Cond: (member_user_id = 15622)
8       Heap Blocks: exact=502
9       Buffers: shared hit=17 read=492
10      -> Bitmap Index Scan on indeks_loan_history_member_loan (cost=0.00..24.11 rows=1007 width=0) (actual time=0.373..0.373 rows=975.00 loops=1)
11        Index Cond: (member_user_id = 15622)
12        Index Searches: 1
13        Buffers: shared hit=7
14      -> Hash (cost=1637.00..1637.00 rows=100000 width=18) (actual time=51.958..51.959 rows=100000.00 loops=1)
15        Buckets: 131072 Batches: 1 Memory Usage: 6103kB
16        Buffers: shared hit=53 read=584
17      -> Seq Scan on book_copy bc (cost=0.00..1637.00 rows=100000 width=18) (actual time=0.083..18.668 rows=100000.00 loops=1)
18        Buffers: shared hit=53 read=584
19    -> Index Scan using book_pkey on book b (cost=0.29..0.33 rows=1 width=50) (actual time=0.082..0.082 rows=1.00 loops=975)
20      Index Cond: ((barcode)::text = (bc.barcode)::text)
21      Index Searches: 975
22      Buffers: shared hit=2631 read=294
23 Planning:
24   Buffers: shared hit=27
25 Planning Time: 1.951 ms
26 Execution Time: 151.288 ms
```

Постоечкиот индекс овозможува значително побрзо пребарување на историјата на позајмувања според корисник и не беше потребна дополнителна оптимизација.