Напредни бази на податоци

Фаза 3- Индекси и оптимизација на прашалници

Проект: BookNest

Вероника Иванова 231073

Ивана Апостолова 231076

Ивана Дачевска 231077

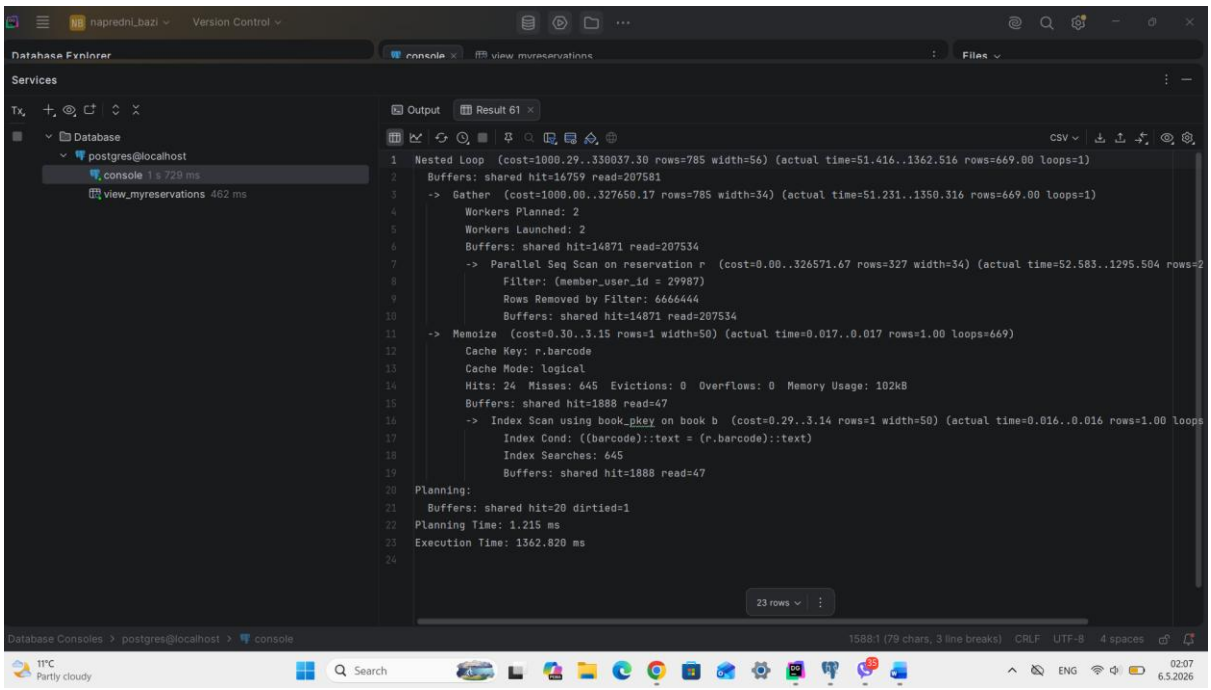
View: view_myreservations

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.



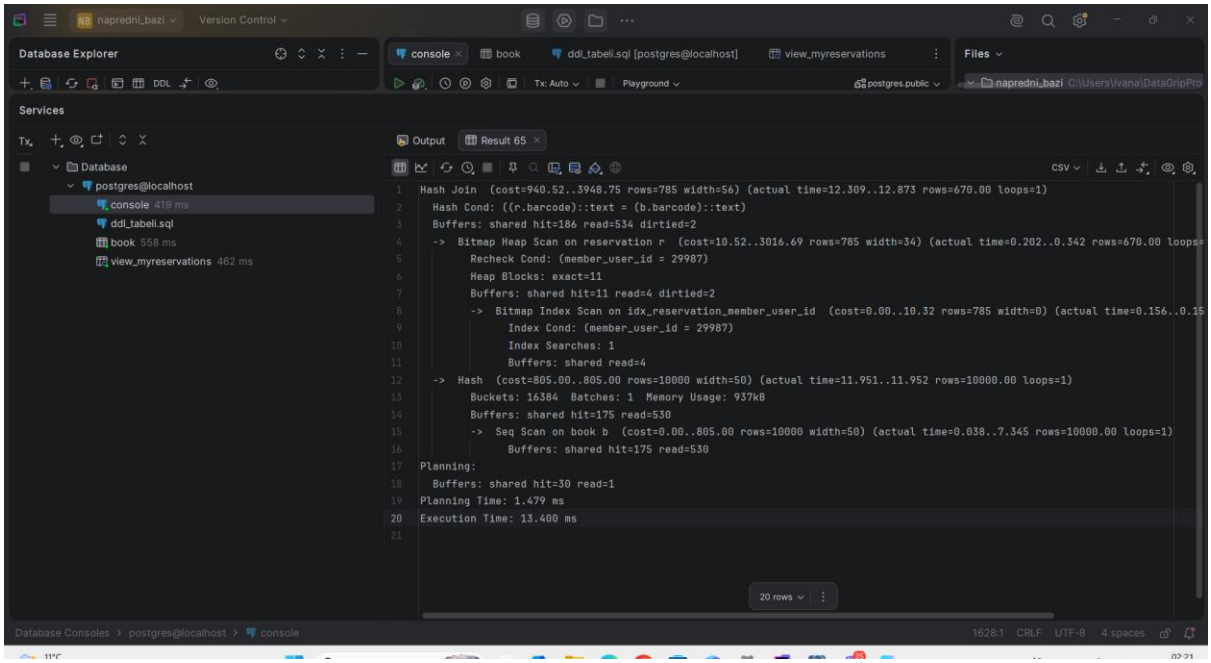
```
[2026-05-06 02:06:19] postgres.public> EXPLAIN ANALYZE
                                       SELECT *
                                       FROM view_myreservations
                                       WHERE member_user_id = 29987
[2026-05-06 02:06:21] 23 rows retrieved starting from 1 in 1 s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

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

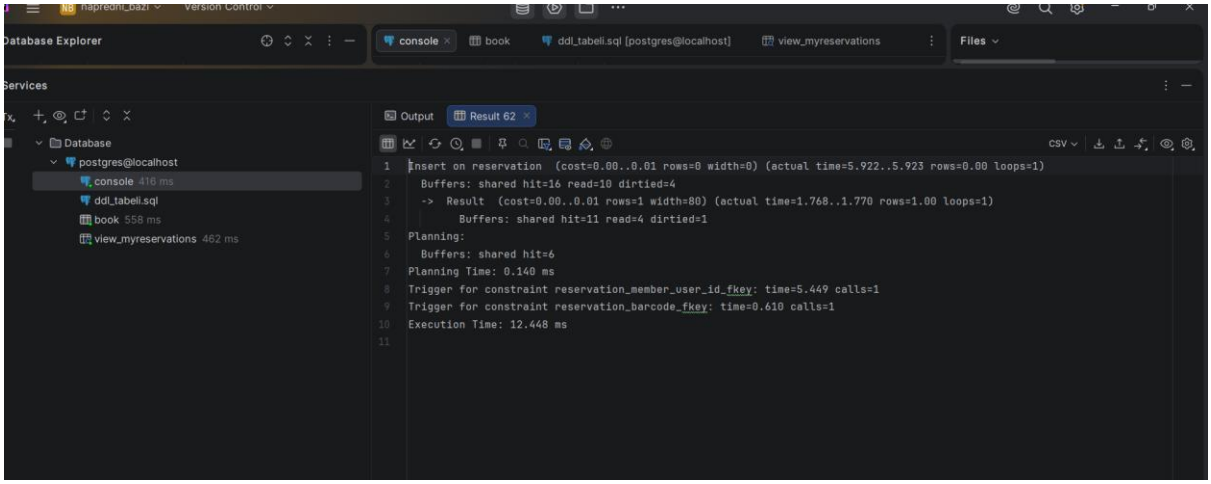
```
CREATE INDEX idx_reservation_member_user_id
    ON reservation(member_user_id);
```

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.

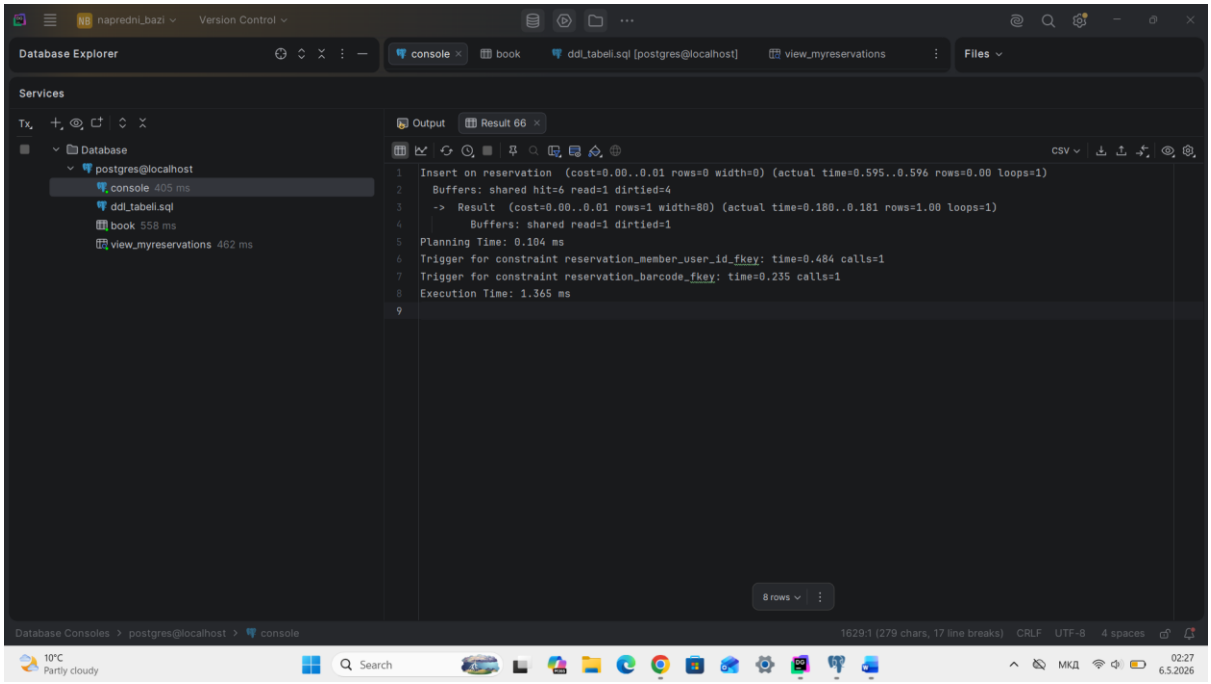


Времето на **INSERT операцијата** пред индексирање изнесува 12.448 ms, а додека по индексирањето изнесува 1.365 ms.

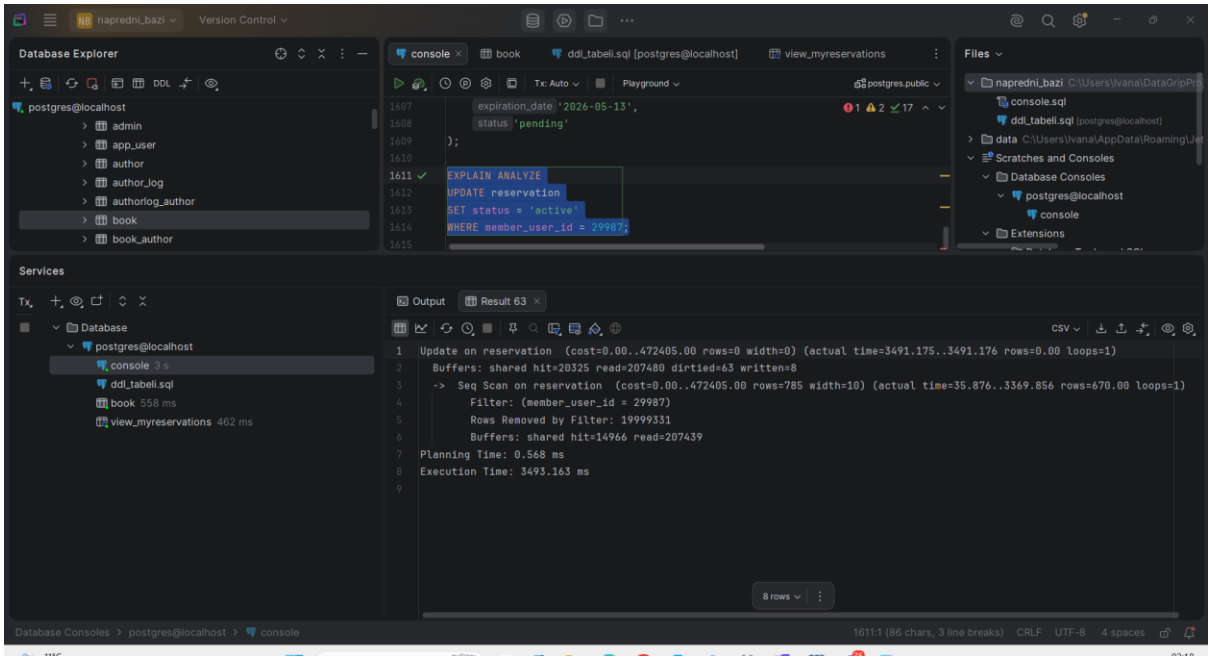


```
[2026-05-06 02:15:30] 10 rows retrieved starting from 1 in 384 ms (execution: 30 ms, fetching: 354 ms)
```

По индексирање:

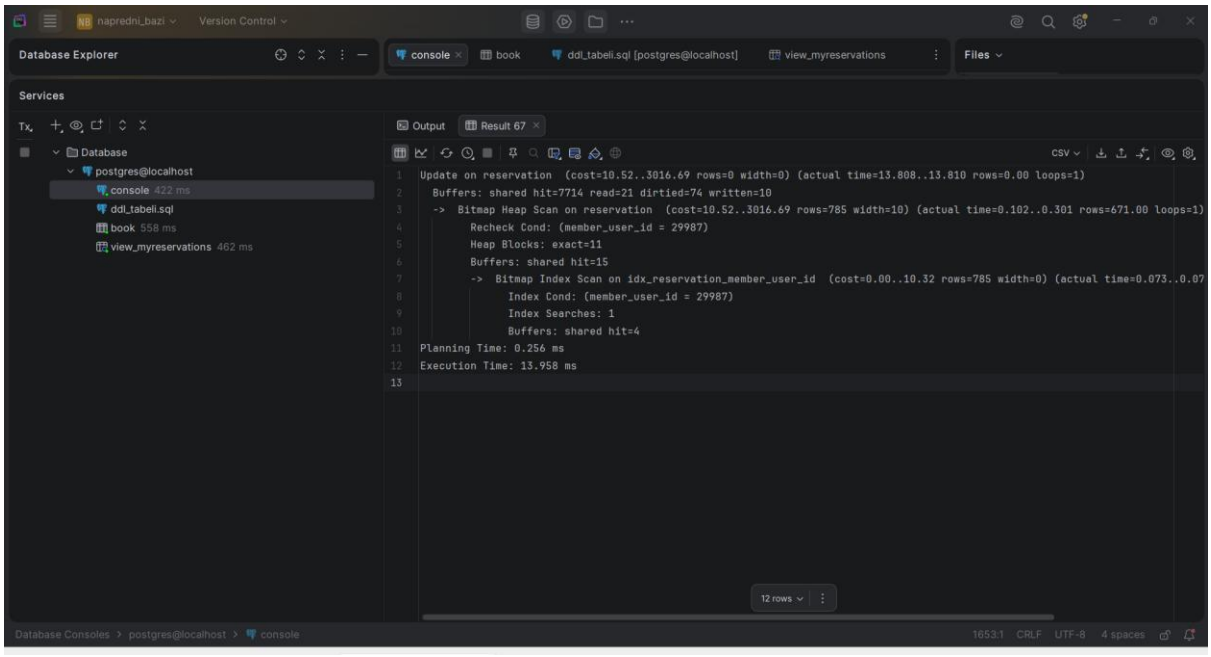


Најзначително подобрување се забележува кај **Update операциите** каде времето на извршување пред индексирање изнесува 3493.163 ms, а по индексирање се намалува на 13.958 ms.




```
                                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:



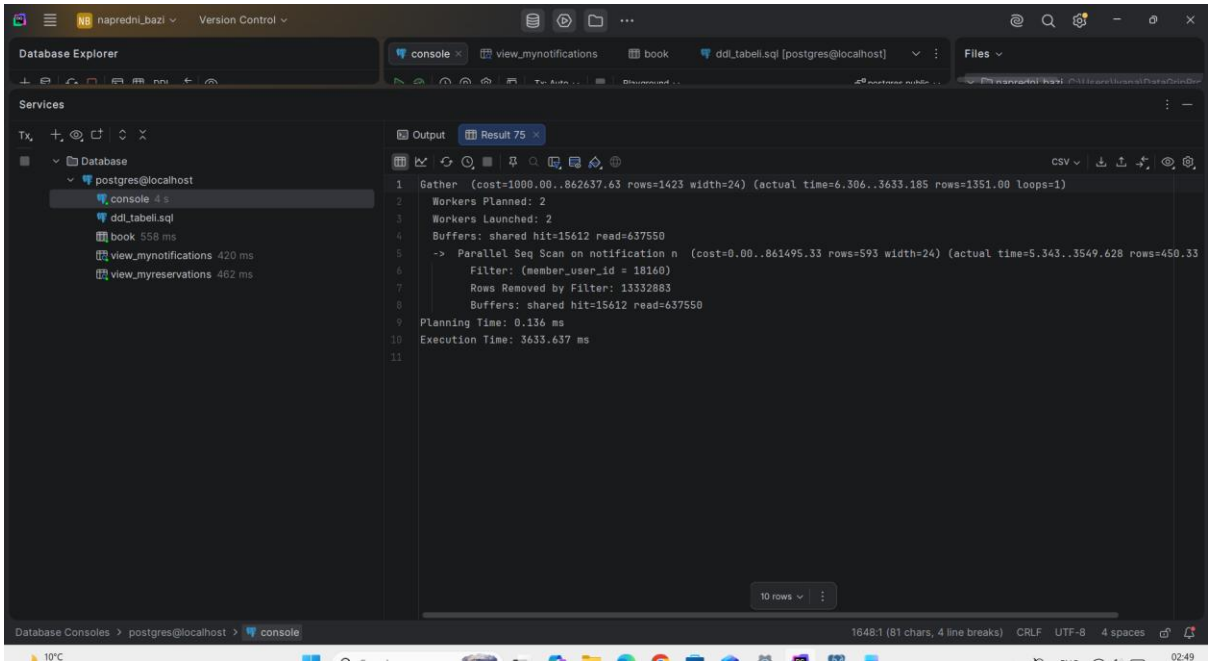
View: view_mynotifications

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)
```

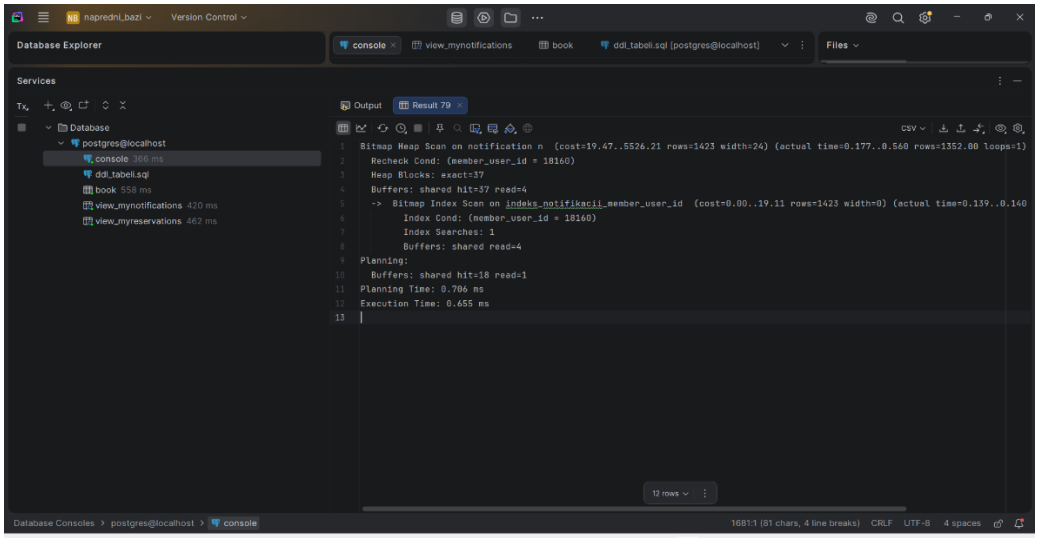


Најбавната операција беше филтрирањето по member_user_id, па затоа беше додаден следниот индекс:

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

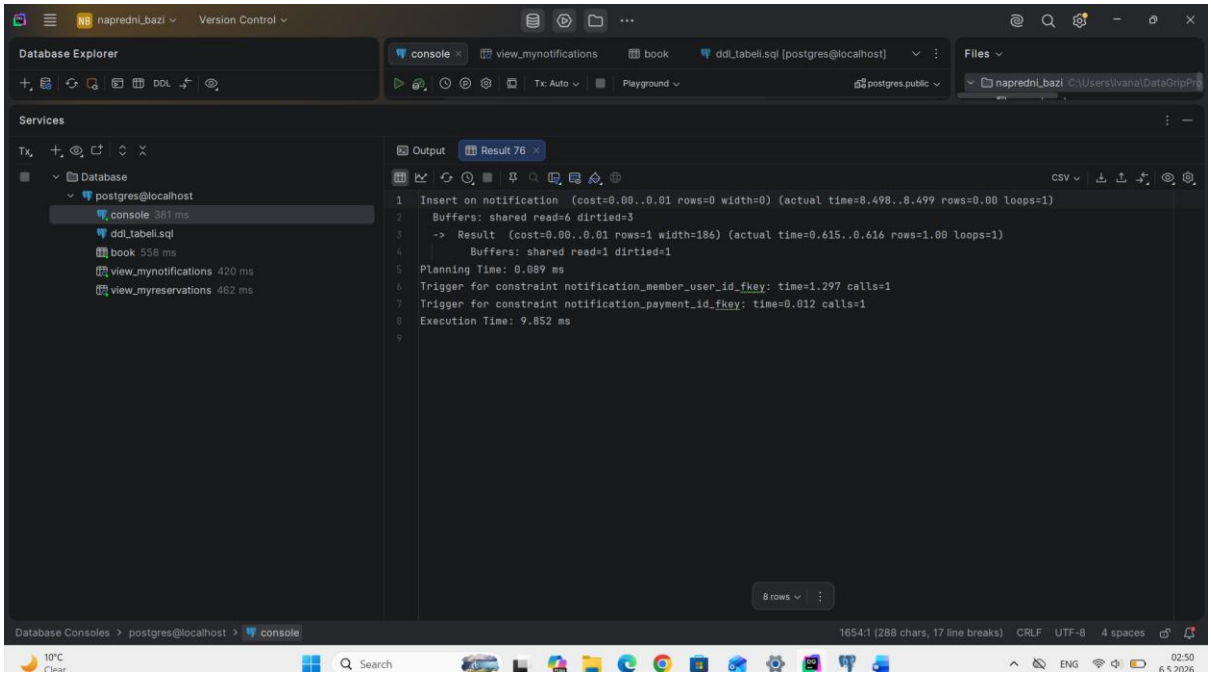
По додавањето на индексот, PostgreSQL започна да користи **Bitmap Index Scan** и **Bitmap Heap Scan** наместо **Parallel Sequential Scan**.

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.

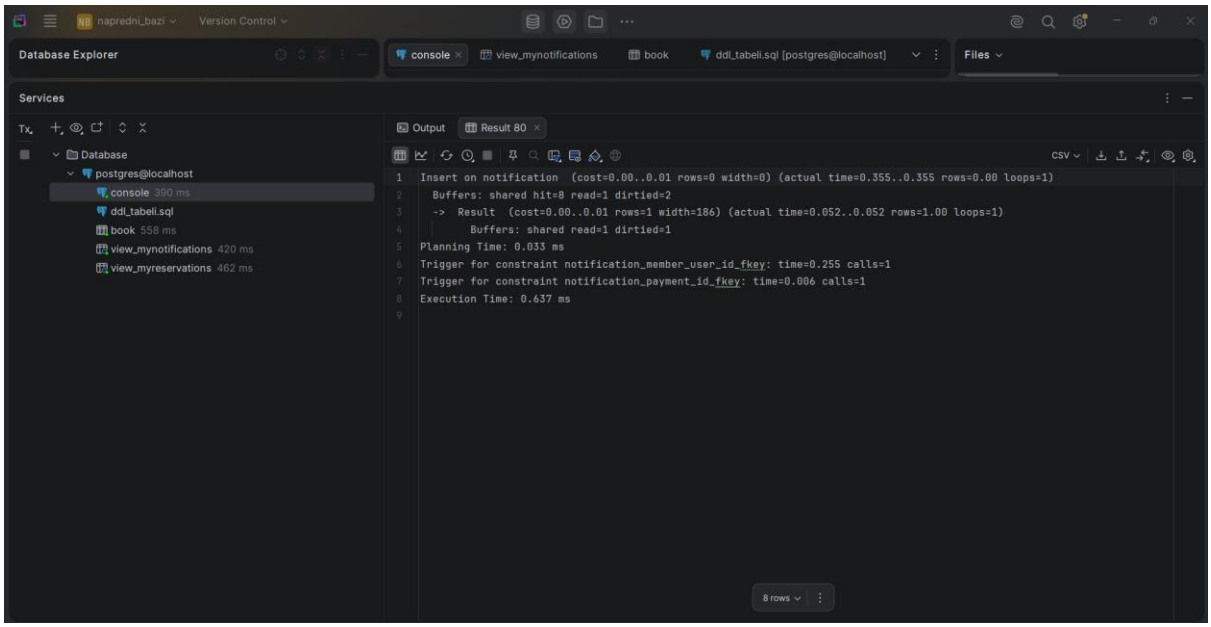



```
[2026-05-06 02:52:10] postgres.public> EXPLAIN ANALYZE
                                       SELECT *
                                       FROM view_mynotifications
                                       WHERE member_user_id = 18160
[2026-05-06 02:52:10] 12 rows retrieved starting from 1 in 344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

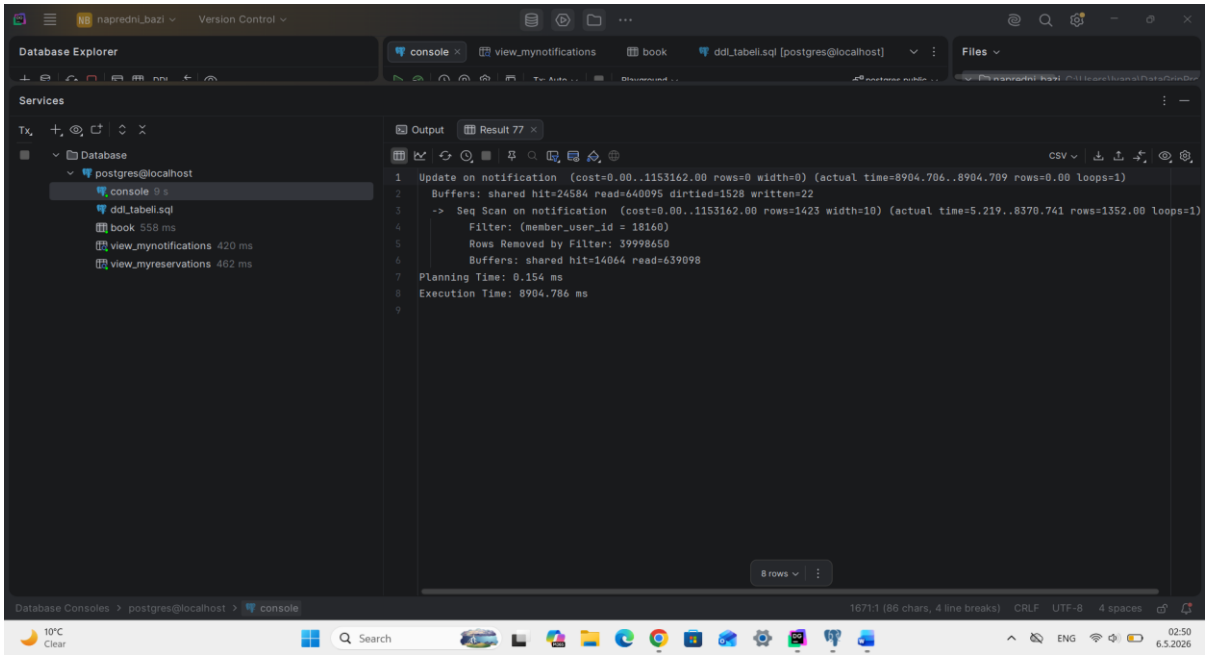
Времето на **INSERT операцијата** пред индексирање изнедува **9.852** ms, а по индексирање изнесува **0.637** ms.



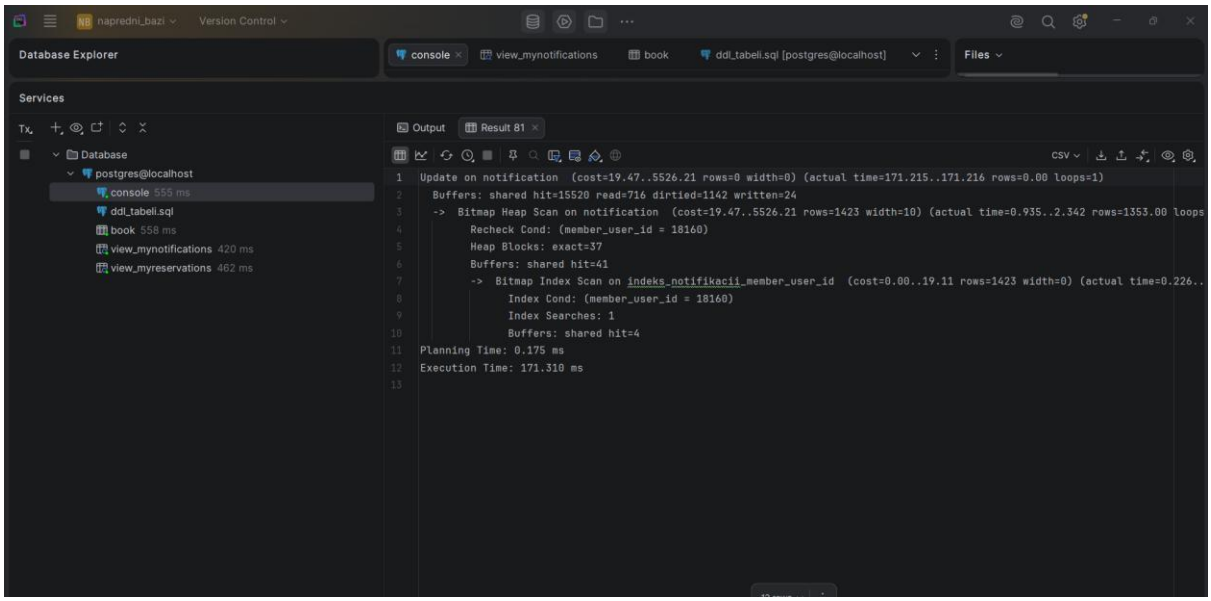
По индексирање:



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.

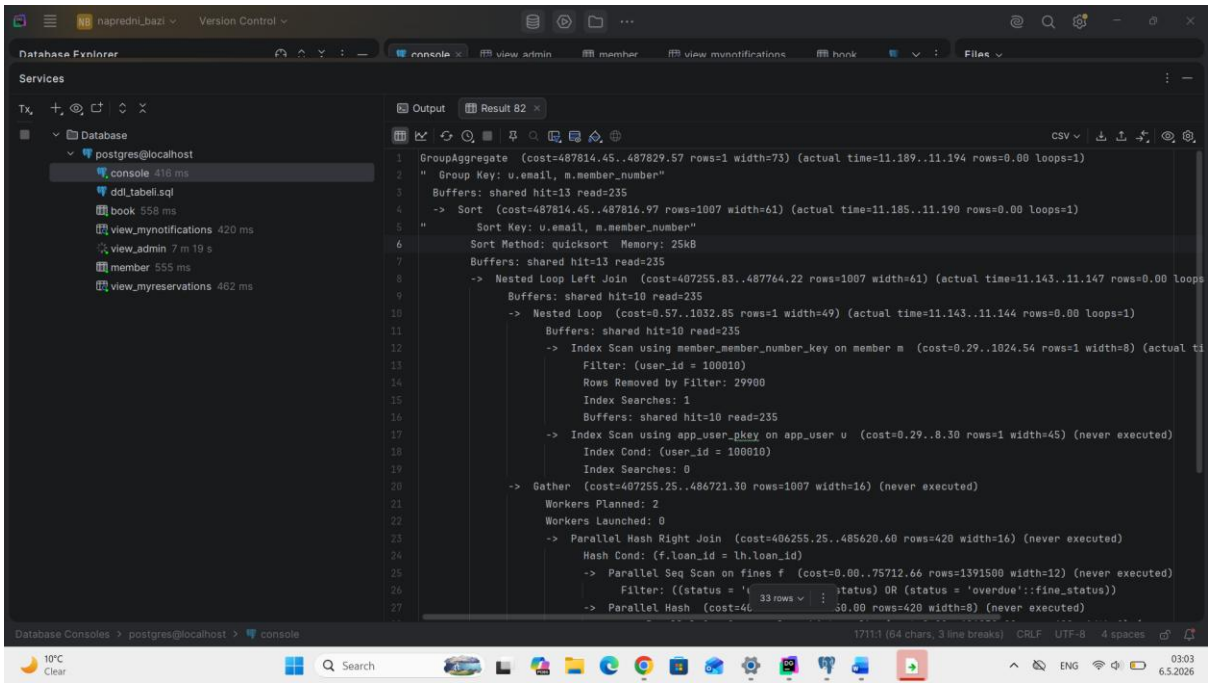
Explain Analyze покажува дека PostgreSQL користи **GroupAggregate** поради агрегатните функции **COUNT и SUM**. Најголем дел од времето се троши на агрегација и групирање на податоците, а не на пребарување.

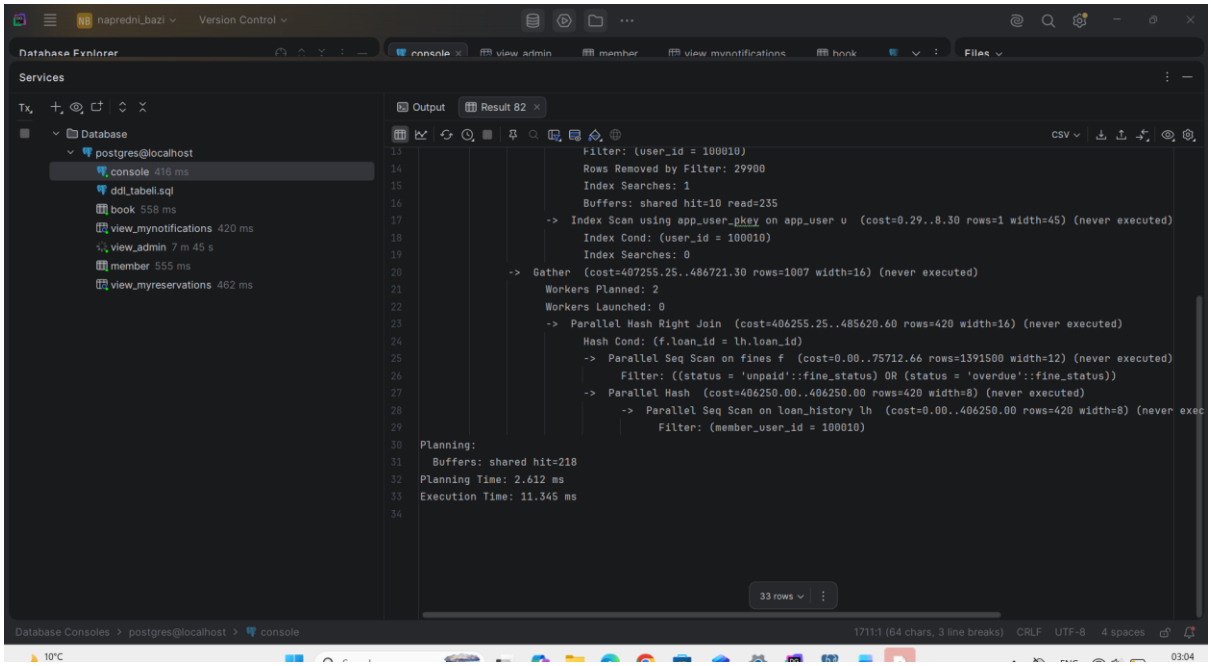
Дополнително, PostgreSQL веќе користи постоечки **B-tree индекси** преку Index Scan операции врз табелите member и app_user. Овие индекси се автоматски креирани преку **PRIMARY KEY и UNIQUE** ограничувањата.

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)
```





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**.

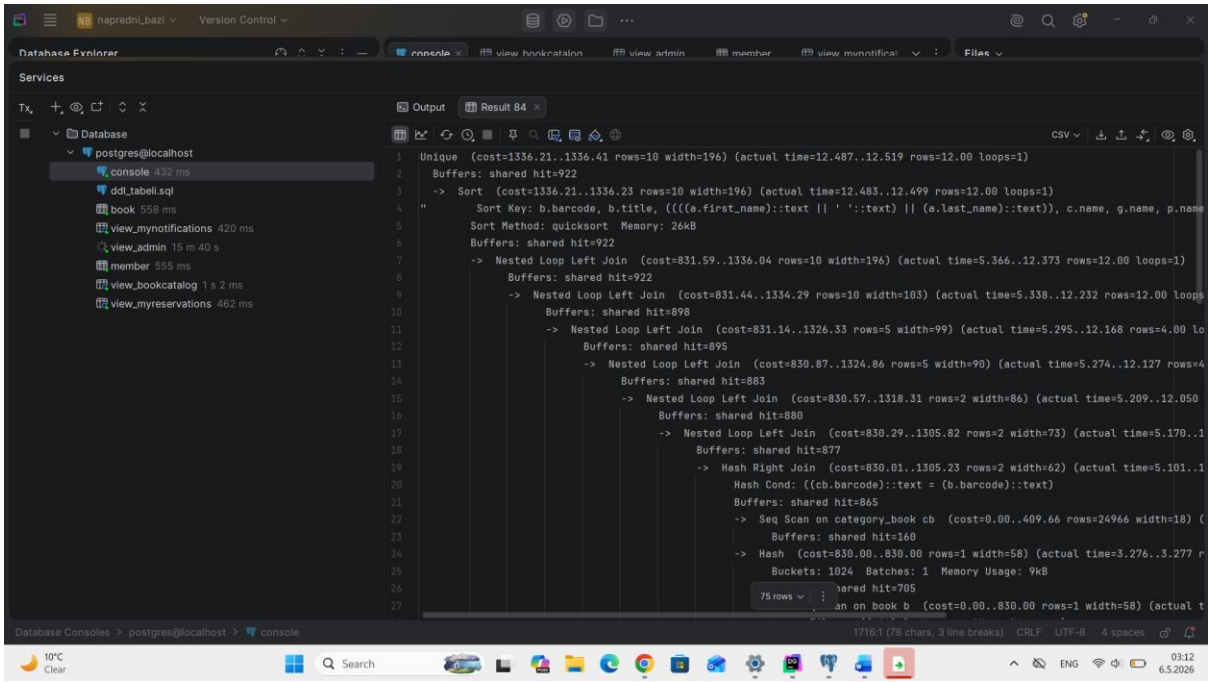
Погледот ќе се користи за пребарување и преглед на информации за книги во библиотеката. За овој поглед се важни перформансите бидејќи корисниците често ќе пребаруваат книги.

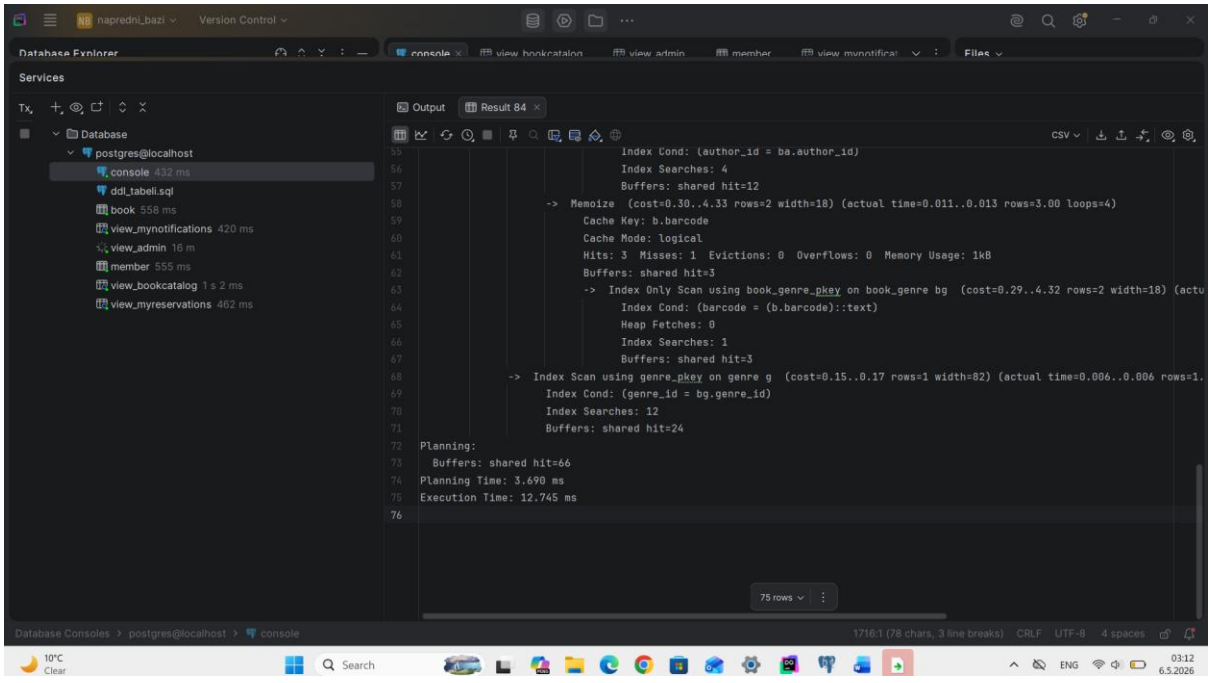
Иницијалното време за извршување на погледот изнесува **12.745** ms, што е **прифатливо** време за апликацијата.

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.






```
WHERE book_title LIKE 'Harry%
[2026-05-06 03:09:16] 75 rows retrieved starting from 1 in 388 ms (execution: 25 ms, fetching: 363 ms)
```

View: view_bookmonthly

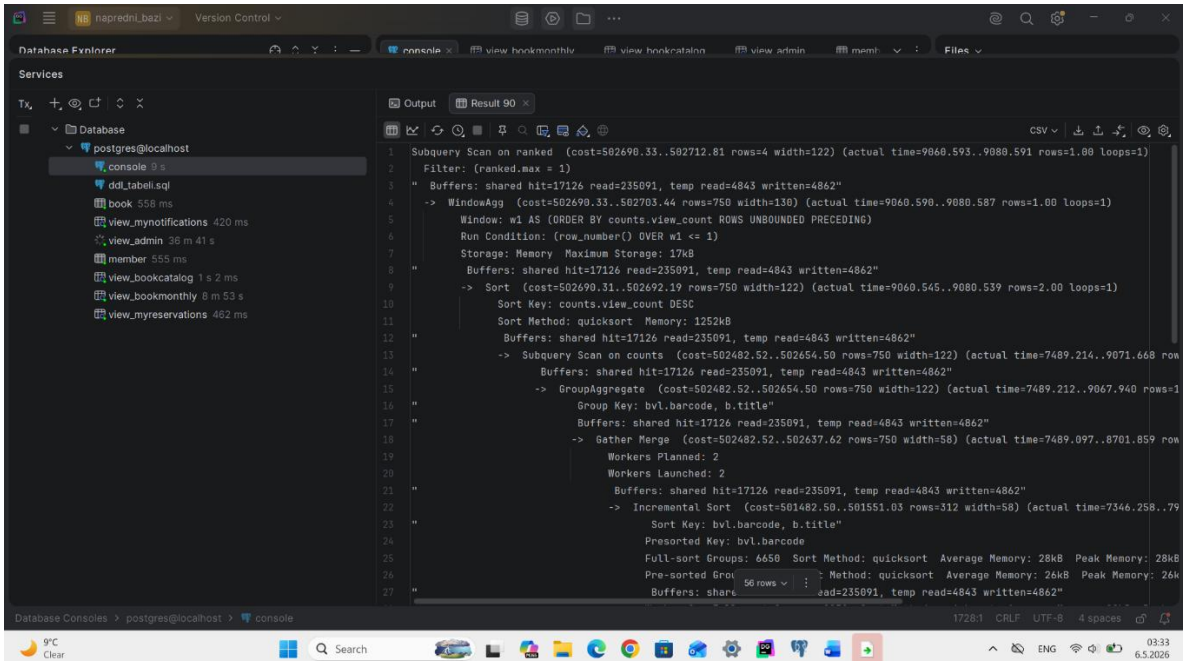
Погледот view_bookmonthly ќе се користи за прикажување на **најпрегледуваната книга** за секој месец.

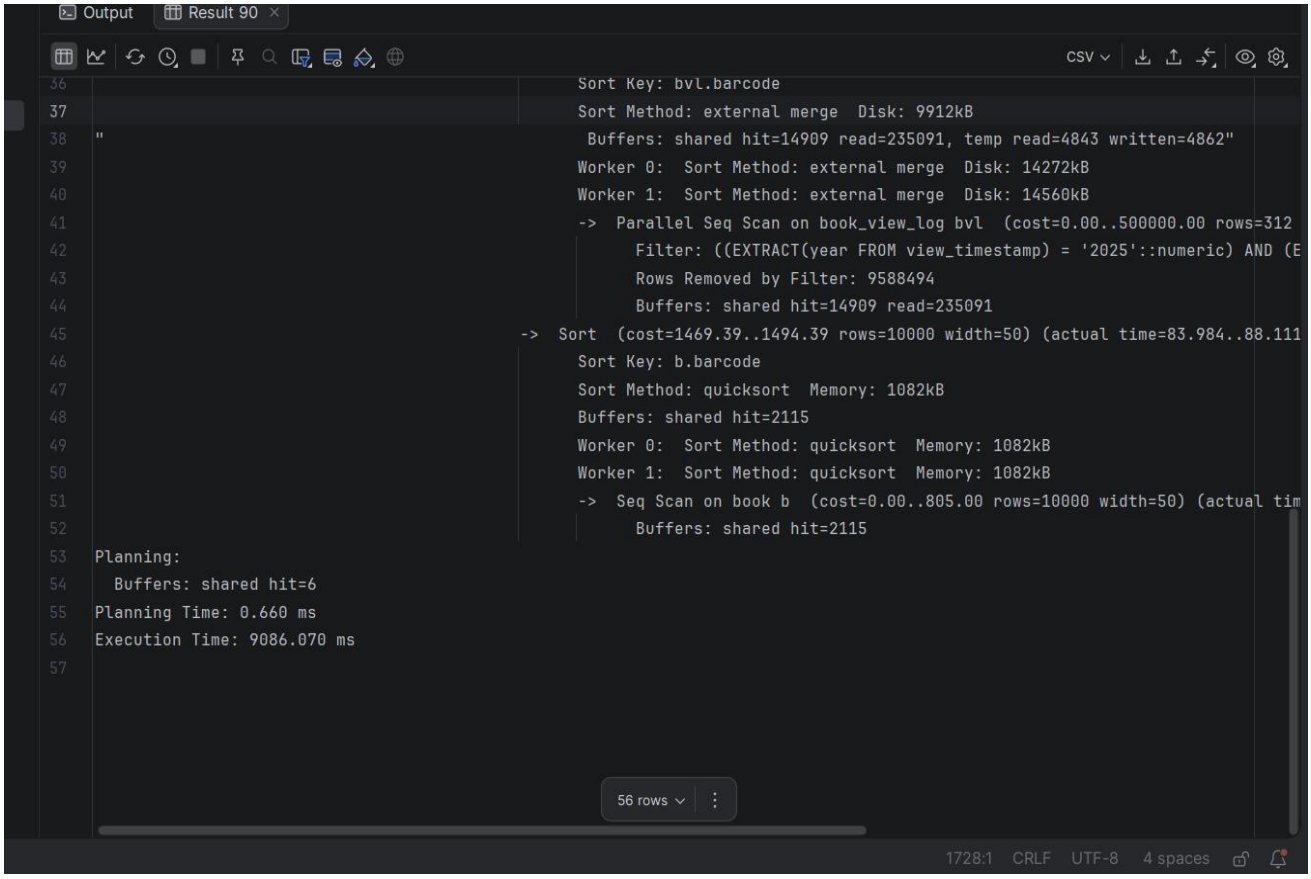
Погледот користи агрегатни функции и аналитички функции, односно **COUNT, GROUP BY и ROW_NUMBER**, со цел да се пресмета бројот на прегледи по книга и да се одреди книгата со најголем број прегледи за секој месец.

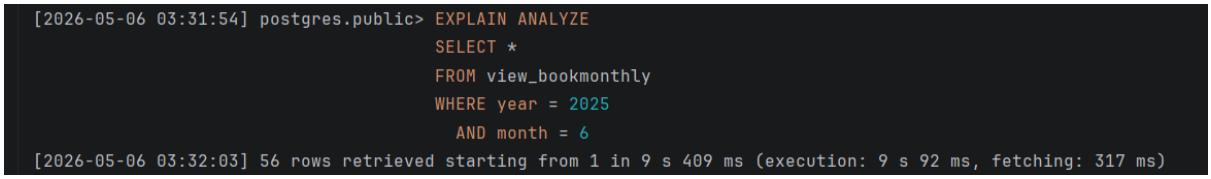
Поради природата на прашалникот и обработката на големо множество податоци од табелата book_view_log, PostgreSQL мора да изврши агрегација, сортирање и рангирање на податоците.

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)
```







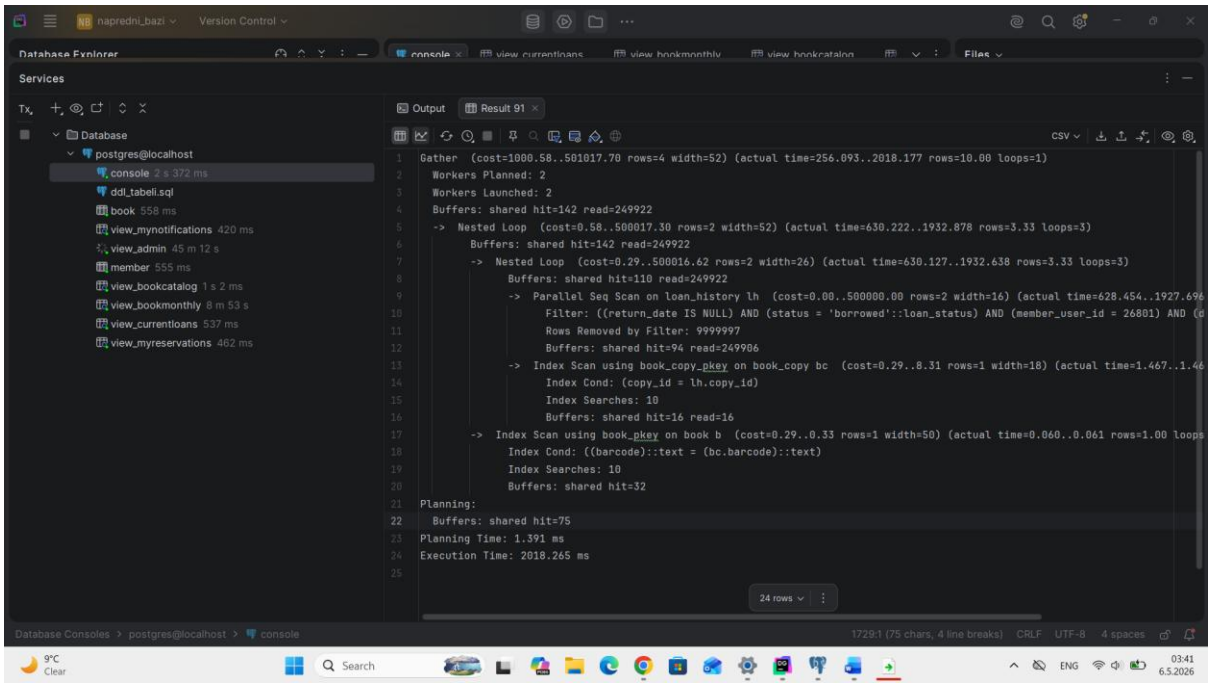
View: view_currentloans

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)
```



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 корисник. Најголем дел од времето се троши на пребарување низ целата табела loan_history.

Поради тоа беше креиран partial B-tree индекс:

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

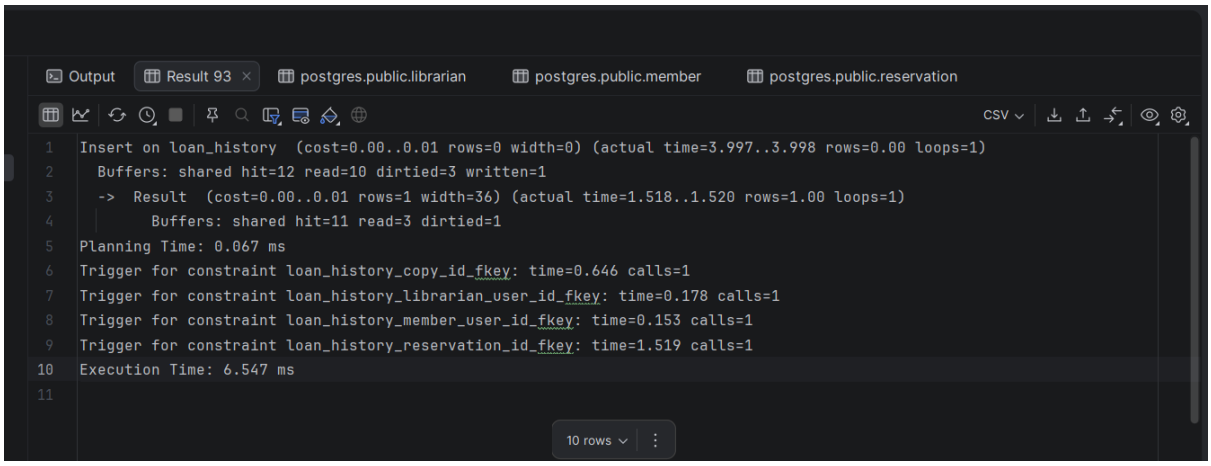
Овој индекс ги содржи само активните позајмувања, односно редовите кај кои книгите не се вратени и статусот е borrowed. Со тоа индексот е помал и поефикасен од обичен индекс врз целата табела.

По додавањето на индексот PostgreSQL започна да користи **Index Scan** врз indeks_aktivni_pozajmuvanja наместо Parallel Sequential Scan, а времето на извршување се намали на **3.842 ms**, што претставува значително подобрување и е прифатливо за апликацијата.

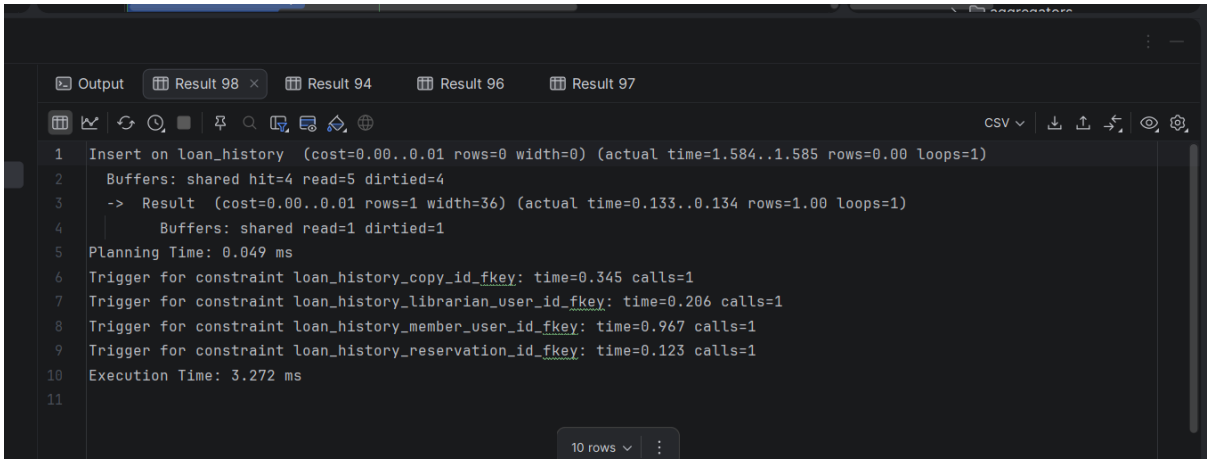
```
[2026-05-06 04:06:30] postgres.public> EXPLAIN ANALYZE
                                       SELECT *
                                       FROM view_currentloans
                                       WHERE korisnik_id = 15622
[2026-05-06 04:06:30] 20 rows retrieved starting from 1 in 344 ms (execution: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

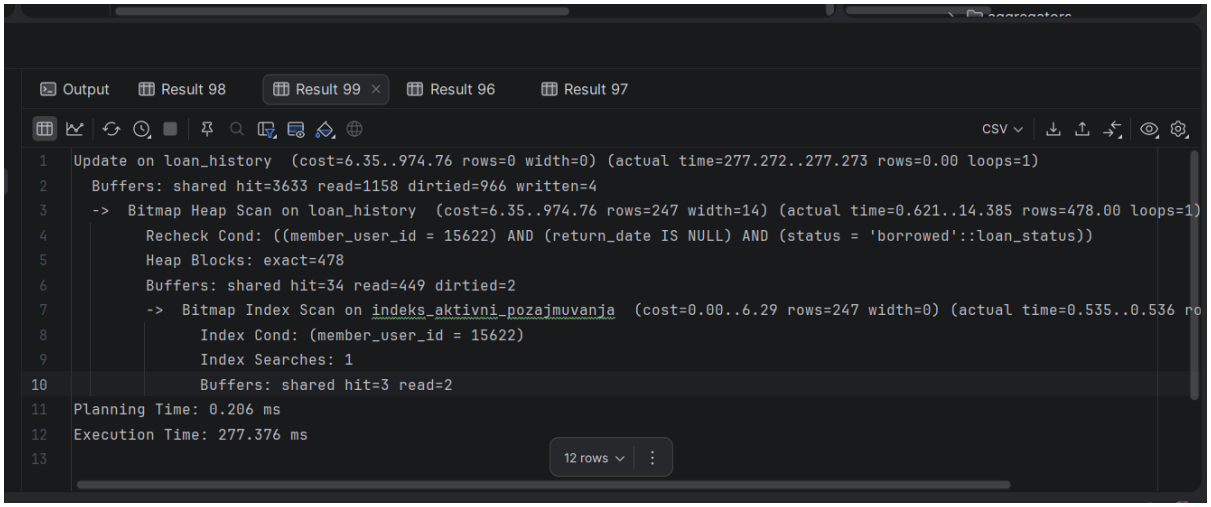


Insert операција со индекс:



Update операција без индекс:

Update операција без индекс:



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

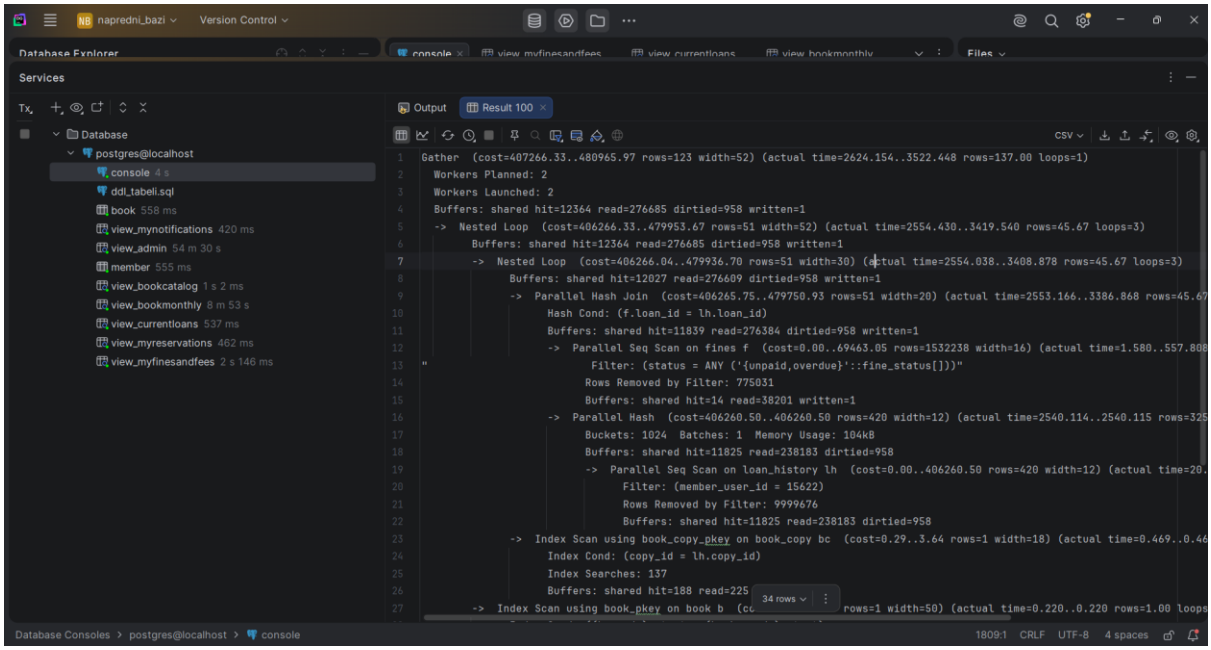
Примарен филтер за погледот view_myfinesandfees ќе биде според **korisnik_id**.

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.



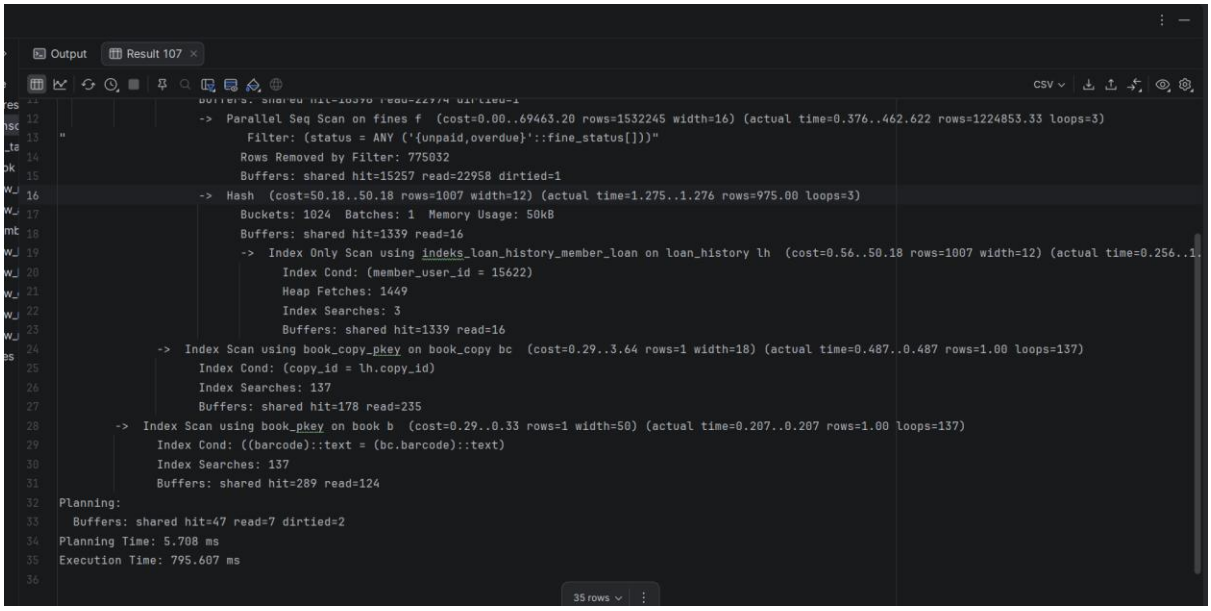
Поради тоа беше креиран следниот **B-tree индекс**:

```
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 indeks_loan_history_member_loan, а времето на извршување се намали од **3523.558** ms на

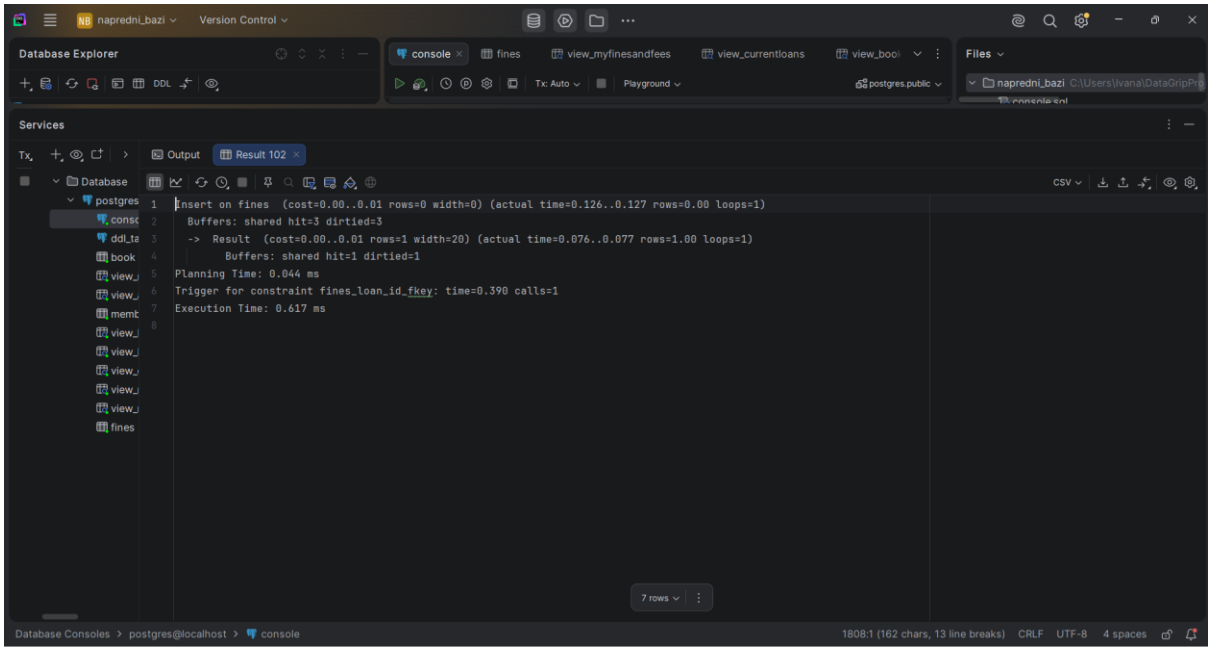
795.607 ms, што претставува значително подобрување на перформансите.



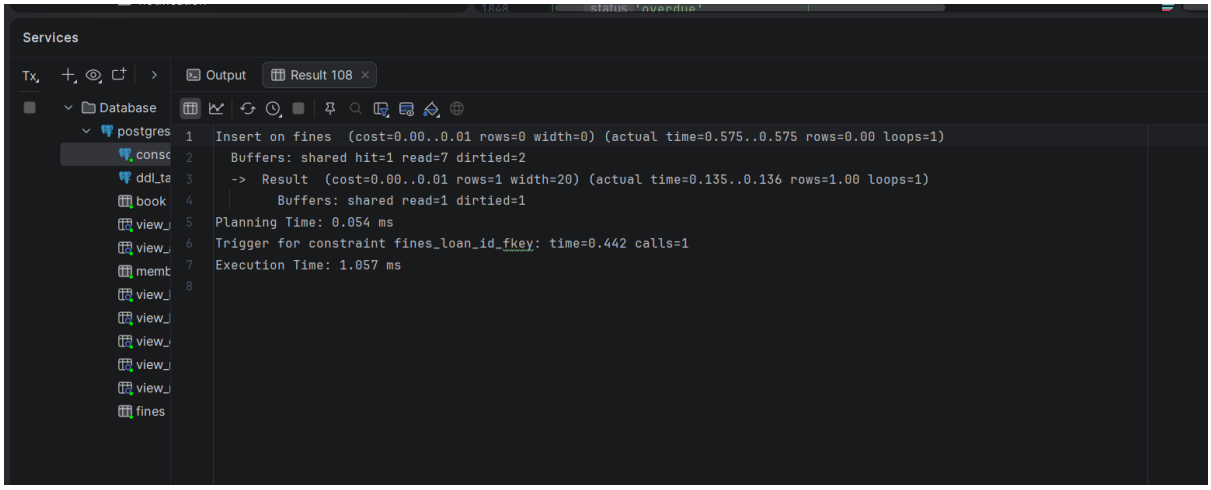
Иако PostgreSQL сè уште користи Parallel Sequential Scan врз табелата fines, времето на извршување е значително намалено поради побрзото пребарување во loan_history.

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.

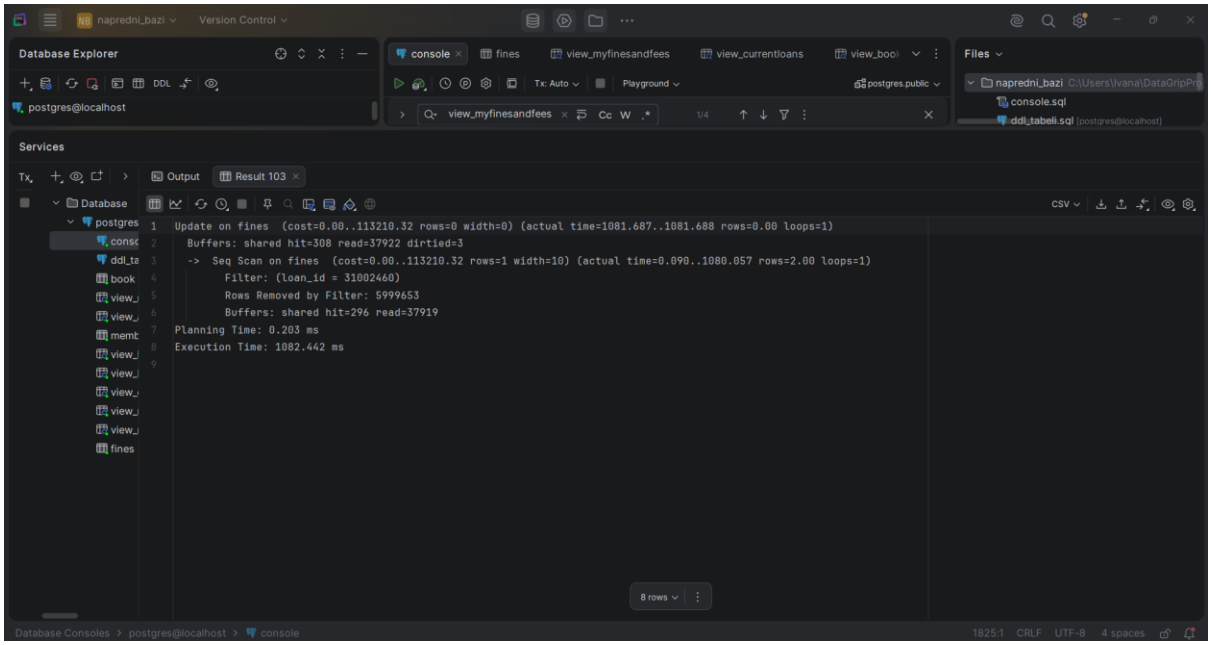
Insert операција без индекс:



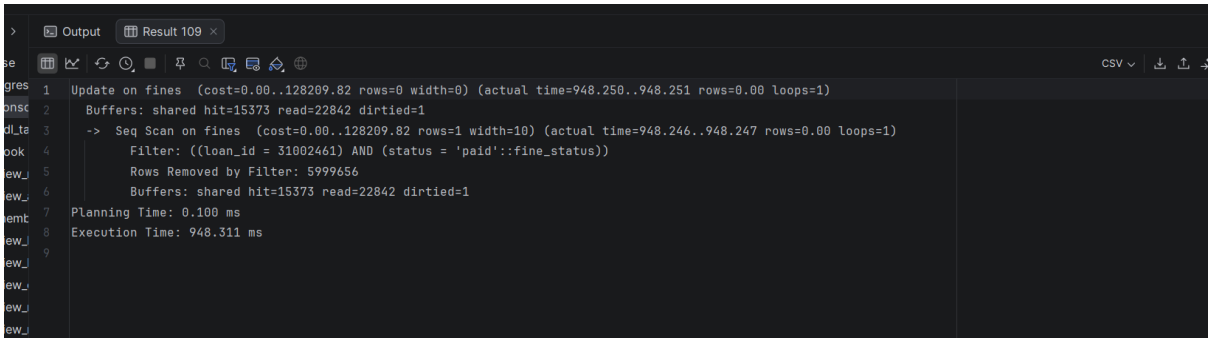
Insert операција со индекс:



Update операција без индекс:



Update операција со индекс:



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,

датум, време, локација, максимален број седишта и број на слободни седишта. Дополнително, преку CASE израз се пресметува и статус availability_status кој покажува дали настанот е AVAILABLE или FULL.

За тестирање на перформансите беше извршен следниот SELECT:

```
SELECT *
FROM view_upcomingevents
WHERE title LIKE 'Lecture%';
```

Explain Analyze покажува дека PostgreSQL користи Sequential Scan врз табелата event. При пребарувањето биле вратени 1255 редови, а 8745 редови биле отфрлени преку филтерот.

Иако не се користи индекс, времето на извршување изнесува само **2.716** ms, што е **прифатливо** за апликацијата. Бидејќи табелата event не е многу голема, PostgreSQL проценува дека Sequential Scan е поефикасен од користење индекс.

Поради добрите перформанси не беше потребна дополнителна оптимизација или додавање индекс за овој поглед.

View: view_userborrowhistory

Погледот view_userborrowhistory се користи за **приказ на историјата на позајмувања на корисниците**. Во погледот се прикажуваат информации за книгата, датумот на позајмување, рокот за враќање, статусот на позајмувањето, времетраењето на користење и бројот на денови на доцнење.

За тестирање на перформансите беше извршен следниот SELECT:

```
SELECT *
FROM view_userborrowhistory
WHERE korisnik_id = 15622;
```

Explain Analyze покажува дека PostgreSQL користи **Bitmap Index Scan** врз индексот indeks_loan_history_member_loan. Индексот е креиран врз колоните:

(member_user_id, loan_id, copy_id)

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.

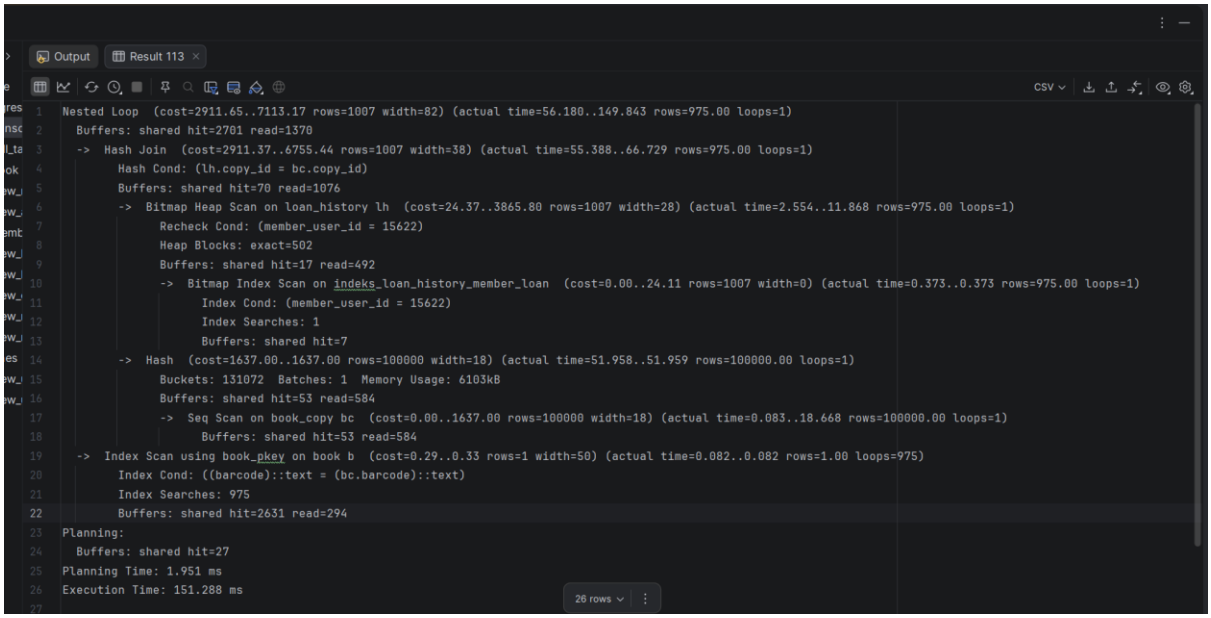
Во Explain Analyze може да се забележи:

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.



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