Фаза 3: Оптимизација на прашалници и погледи

Проект: Public Transport System

Членови:

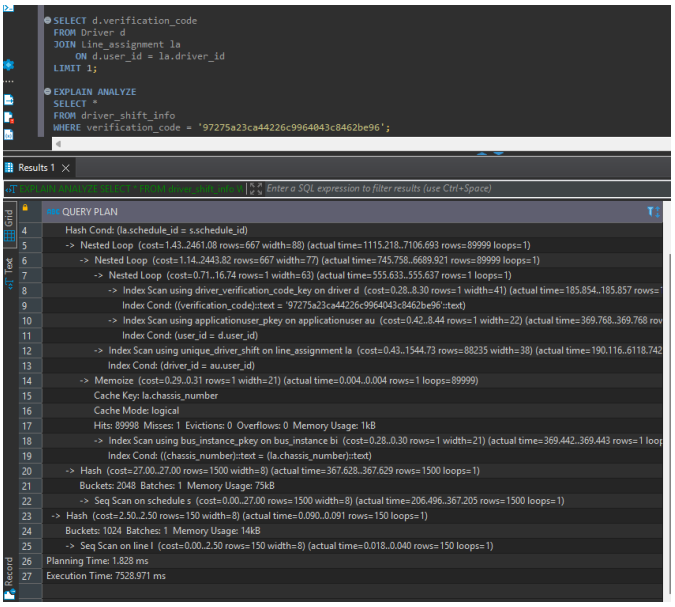
Дамјан Илиевски 231003

Викторија Георгиевска 231006

Стефан Марковиќ 231196

View 1: driver_shift_info

Погледот **driver_shift_info** се користи за приказ на информации на смените на даден возач, заедно со автобусот, распоредот и линијата на која е назначен. Примарен филтер за овој поглед е **verification_code** на возачот.



Пред оптимизацијата, времето на извршување изнесуваше **7528.971 ms**, што е над дозволената граница од 2 секунди.

Во execution plan се забележуваше дека најголем дел од времето се троши при join операциите со Line_assignment, Schedule и Bus_instance.

За оптимизација беа додадени следните индекси:

- idx_line_assignment_schedule_id на Line_assignment (schedule_id);
- idx_line_assignment_chassis_number на Line_assignment (chassis_number);
- idx_schedule_line_id на Schedule (line_id);

```sql
CREATE INDEX IF NOT EXISTS idx_line_assignment_schedule_id
ON Line_assignment(schedule_id);

CREATE INDEX IF NOT EXISTS idx_line_assignment_chassis_number
ON Line_assignment(chassis_number);

CREATE INDEX IF NOT EXISTS idx_schedule_line_id
ON Schedule(line_id);
```

По додавањето на индексите и повторното извршување на ANALYZE, времето на извршување се намали на **147.643 ms**. Ова време е под 2 секунди и е **прифатливо за апликацијата**.

```sql
EXPLAIN ANALYZE
SELECT *
FROM driver_shift_info
WHERE verification_code = '97275a23ca44226c9964043c8462be96';
```

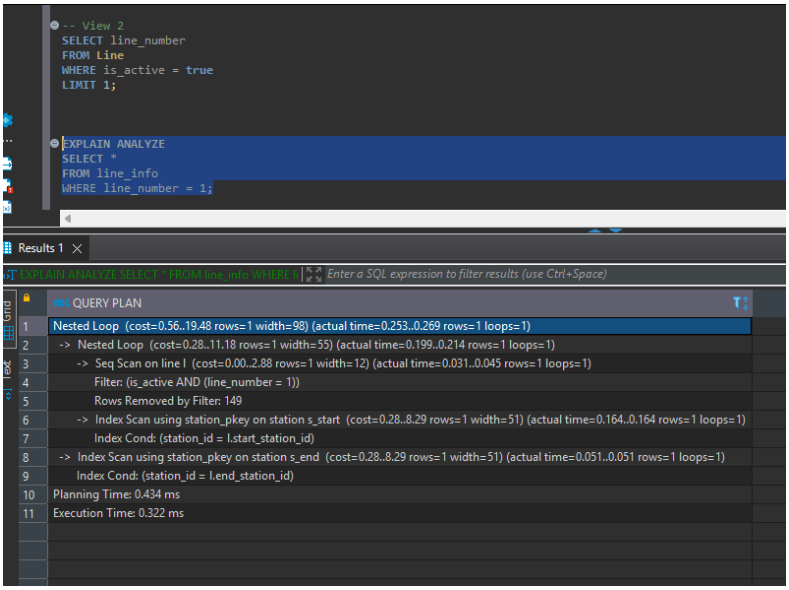
Results 1

EXPLAIN ANALYZE SELECT * FROM driver_shift_info W | Enter a SQL expression to filter results (use Ctrl+Space)

```
QUERY PLAN
  -> Nested Loop  (cost=1.14..2443.82 rows=667 width=77) (actual time=0.201..48.576 rows=89999 loops=1)
    -> Nested Loop  (cost=0.71..16.74 rows=1 width=63) (actual time=0.123..0.127 rows=1 loops=1)
      -> Index Scan using driver_verification_code_key on driver d  (cost=0.28..8.30 rows=1 width=41) (actual time=0.078..0.080 rows=1 loo
        Index Cond: ((verification_code)::text = '97275a23ca44226c9964043c8462be96'::text)
      -> Index Scan using applicationuser_pkey on applicationuser au  (cost=0.42..8.44 rows=1 width=22) (actual time=0.041..0.041 rows=1
        Index Cond: (user_id = d.user_id)
    -> Index Scan using unique_driver_shift on line_assignment la  (cost=0.43..1544.73 rows=88235 width=38) (actual time=0.076..32.413 row
      Index Cond: (driver_id = au.user_id)
  -> Memoize  (cost=0.29..0.31 rows=1 width=21) (actual time=0.000..0.000 rows=1 loops=89999)
    Cache Key: la.chassis_number
    Cache Mode: logical
    Hits: 89998  Misses: 1  Evictions: 0  Overflows: 0  Memory Usage: 1kB
    -> Index Scan using bus_instance_pkey on bus_instance bi  (cost=0.28..0.30 rows=1 width=21) (actual time=0.043..0.044 rows=1 loops=1)
      Index Cond: ((chassis_number)::text = (la.chassis_number)::text)
->  Hash  (cost=27.00..27.00 rows=1500 width=8) (actual time=0.536..0.537 rows=1500 loops=1)
  Buckets: 2048  Batches: 1  Memory Usage: 75kB
  -> Seq Scan on schedule s  (cost=0.00..27.00 rows=1500 width=8) (actual time=0.013..0.294 rows=1500 loops=1)
->  Hash  (cost=2.50..2.50 rows=150 width=8) (actual time=0.070..0.070 rows=150 loops=1)
  Buckets: 1024  Batches: 1  Memory Usage: 14kB
  -> Seq Scan on line l  (cost=0.00..2.50 rows=150 width=8) (actual time=0.021..0.043 rows=150 loops=1)
Planning Time: 2.548 ms
Execution Time: 147.643 ms
```

View2: **line_info**

Погледот **line_info** се користи за приказ на активна автобуска линија, заедно со почетната и крајната станица. Примарен филтер за овој поглед е **line_number.** Иницијалното време на извршување изнесуваше **0.322 ms**, што е значително под дозволената граница од 2 секунди.



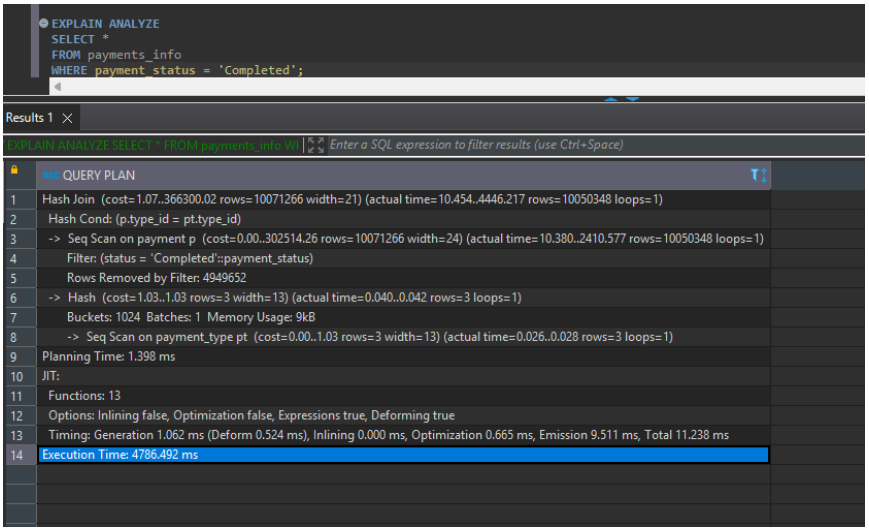
Во execution plan се забележува Seq Scan на табелата Line, но тоа не претставува проблем бидејќи табелата има мал број записи. PostgreSQL проценува дека секвенцијалното читање е поефикасно од користење индекс. Поради добрите перформанси **не беше потребно дополнително индексирање** за овој поглед.

View3: **payments_info**

Погледот **payments_info** се користи за приказ на плаќања според статус и тип на плаќање. Примарен филтер за овој поглед е **payment_status = 'Completed'**.

При извршувањето беше забележано дека query-то враќа голем број редови, па времето на извршување зависи не само од пребарувањето, туку и од количината на податоци што треба да се прикажат. Во реална апликација, овој поглед би се користел со дополнителни филтри или pagination, со цел да не се враќаат премногу записи одеднаш.

Во execution plan не се забележаа критични проблеми со join операциите или пребарувањето. Дополнителните индекси што беа тестирани не дадоа стабилно подобрување на перформансите.

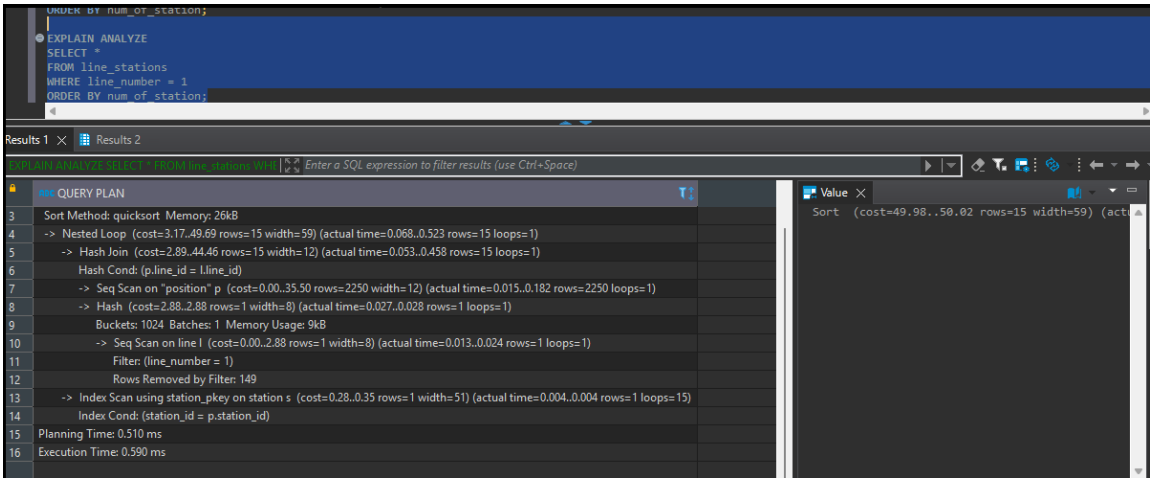


При повторени тестирања времето на извршување достигна **1487 ms**, што е значително под дозволената граница од 2 секунди. Поради тоа беше одлучено **да не се додаваат дополнителни индекси** за овој поглед.

View 4 : **line_stations**

Погледот **line_stations** се користи за приказ на сите станици низ кои поминува дадена автобуска линија. Резултатите се сортираат според **num_of_station**, бидејќи оваа колона го претставува редоследот на станиците во рамки на линијата. Примарен филтер за овој поглед е **line_number**.

Времето на извршување изнесуваше **0.590 ms**, што е значително под дозволената граница од 2 секунди. Поради добрите перформанси **не беше потребно дополнително индексирање** за овој поглед.



Дополнително беа тестирани **INSERT** и **UPDATE** операции врз табелите Station и Position, бидејќи line_stations е обичен view и не складира податоци самостојно.

```
BEGIN;

EXPLAIN ANALYZE
WITH new_station AS (
    INSERT INTO Station (ordinal_number, station_name, address)
    VALUES (
        99999,
        'TEST Station For Indexing',
        'TEST Address 1'
    )
    RETURNING station_id
),
target_line AS (
    SELECT line_id
    FROM Line
    WHERE line_number = 1
    LIMIT 1
),
next_position AS (
    SELECT
        tl.line_id,
        COALESCE(MAX(p.num_of_station), 0) + 1 AS next_num
```

Results 2

Enter a SQL expression to filter results (use Ctrl+Space)

```
QUERY PLAN
Group Key: line.line_id
-> Sort (cost=47.27..47.31 rows=15 width=8) (actual time=0.716..0.722 rows=15 loops=1)
Sort Key: line.line_id
Sort Method: quicksort Memory: 25kB
-> Hash Right Join (cost=2.89..46.98 rows=15 width=8) (actual time=0.137..0.702 rows=15 loops=1)
Hash Cond: (p.line_id = line.line_id)
-> Seq Scan on "position" p (cost=0.00..35.50 rows=2250 width=8) (actual time=0.056..0.397 rows=2250 loops=1)
-> Hash (cost=2.88..2.88 rows=1 width=4) (actual time=0.065..0.067 rows=1 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 9kB
-> Limit (cost=0.00..2.88 rows=1 width=4) (actual time=0.059..0.060 rows=1 loops=1)
-> Seq Scan on line (cost=0.00..2.88 rows=1 width=4) (actual time=0.058..0.058 rows=1 loops=1)
Filter: (line_number = 1)
Planning Time: 0.621 ms
Trigger for constraint position_station_fk on position: time=106.044 calls=1
Trigger for constraint position_line_fk on position: time=0.202 calls=1
Execution Time: 1733.457 ms
```

```
BEGIN;

EXPLAIN ANALYZE
UPDATE Station
SET address = address || ' updated'
WHERE station_id = (
    SELECT s.station_id
    FROM line_stations s
    WHERE s.line_number = 1
    ORDER BY s.num_of_station
    LIMIT 1
);

ROLLBACK;
```

Results 2

Enter a SQL expression to filter results (use Ctrl+Space)

```
QUERY PLAN
Update on station (cost=49.68..57.70 rows=0 width=0) (actual time=132.352..132.361 rows=0 loops=1)
InitPlan 1
-> Limit (cost=49.39..49.40 rows=1 width=8) (actual time=0.765..0.773 rows=1 loops=1)
-> Subquery Scan on s (cost=49.39..49.58 rows=15 width=8) (actual time=0.763..0.770 rows=1 loops=1)
-> Sort (cost=49.39..49.43 rows=15 width=1044) (actual time=0.761..0.767 rows=1 loops=1)
Sort Key: p.num_of_station
Sort Method: top-N heapsort Memory: 25kB
-> Nested Loop (cost=3.17..49.10 rows=15 width=1044) (actual time=0.146..0.748 rows=15 loops=1)
-> Hash Join (cost=2.89..44.46 rows=15 width=12) (actual time=0.101..0.615 rows=15 loops=1)
Hash Cond: (p.line_id = l.line_id)
-> Seq Scan on "position" p (cost=0.00..35.50 rows=2250 width=12) (actual time=0.048..0.340 rows=2250 loops
-> Hash (cost=2.88..2.88 rows=1 width=8) (actual time=0.036..0.037 rows=1 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 9kB
-> Seq Scan on line l (cost=0.00..2.88 rows=1 width=8) (actual time=0.019..0.030 rows=1 loops=1)
Filter: (line_number = 1)
Rows Removed by Filter: 149
-> Index Only Scan using station_pkey on station s_1 (cost=0.28..0.31 rows=1 width=4) (actual time=0.008..0.008 r
Index Cond: (station_id = p.station_id)
Heap Fetches: 0
-> Index Scan using station_pkey on station (cost=0.28..8.30 rows=1 width=522) (actual time=0.808..0.812 rows=1 loops=1)
Index Cond: (station_id = (InitPlan 1).col1)
Planning Time: 0.681 ms
Execution Time: 421.400 ms
```

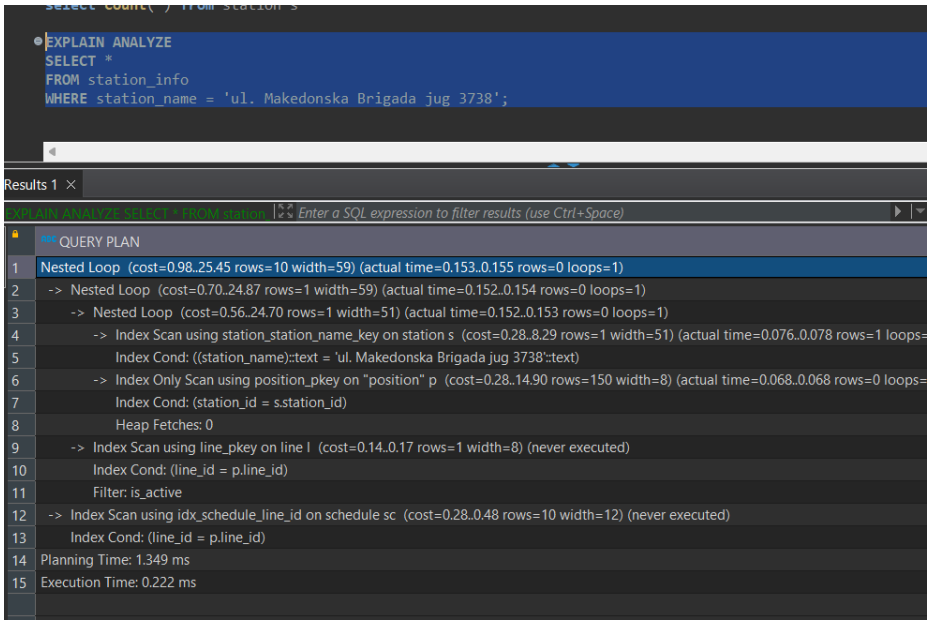
- INSERT тестот симулираше додавање нова станица и нејзино поврзување со линија преку Position. Времето на извршување изнесуваше **1733.457 ms;**
- UPDATE тестот симулираше промена на адреса на станица која припаѓа на дадена линија. Времето на извршување изнесуваше **421.400 ms**;

Сите измерени времиња се под 2 секунди, па **овој поглед е прифатлив за апликацијата.**

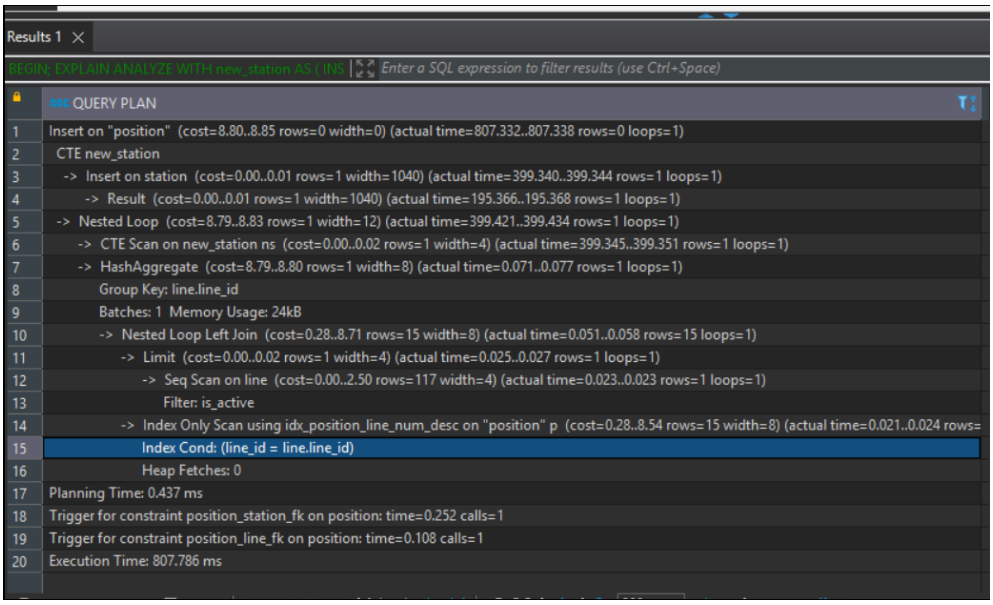
View5 : **station_info**

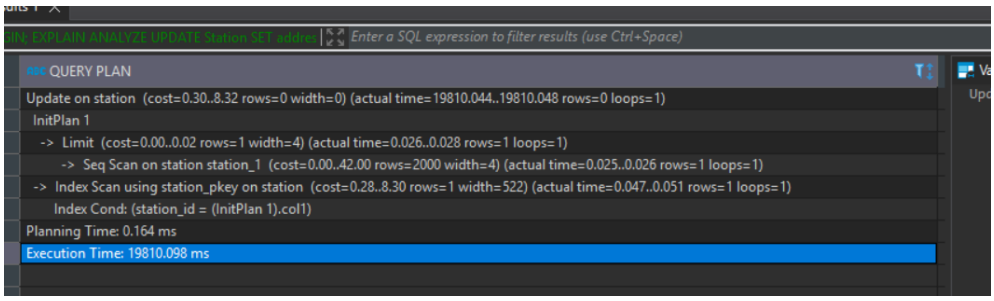
Погледот **station_info** се користи за приказ на информации за дадена станица и активните линии што минуваат низ неа. Примарен филтер за овој поглед е **station_name**.

Времето на извршување изнесуваше **145.144 ms**, што е под дозволената граница од 2 секунди. Во execution plan се гледа користење на постоечки индекси, како station_station_name_key, position_pkey, line_pkey и idx_schedule_line_id. Поради добрите перформанси **не беше потребно дополнително индексирање** за овој поглед.

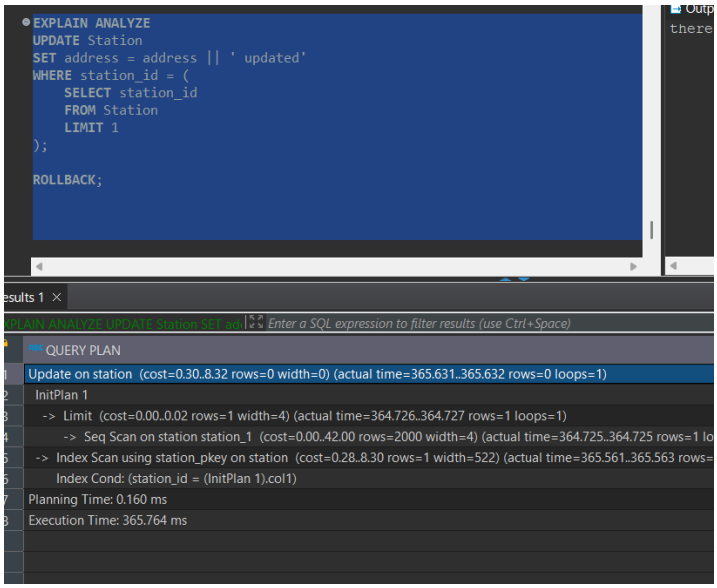


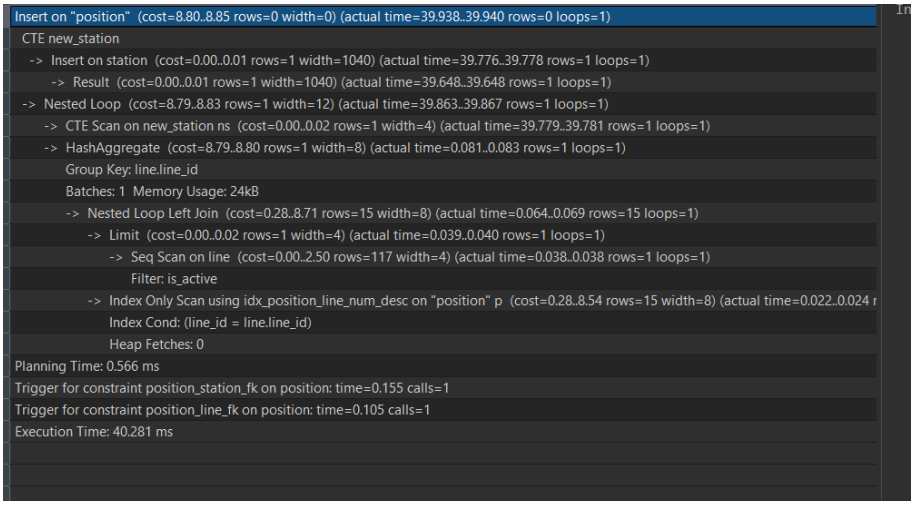
Дополнително беа тестирани **INSERT** и **UPDATE** операции врз основните табели Station и Position, бидејќи station_info е обичен view и не складира податоци самостојно.





- INSERT тестот симулира додавање нова станица и нејзино поврзување со линија преку Position. Времето на извршување изнесуваше **807.786 ms**;
- UPDATE тестот симулира промена на адреса на постоечка станица. Времето на извршување изнесуваше **19810.098 ms**;

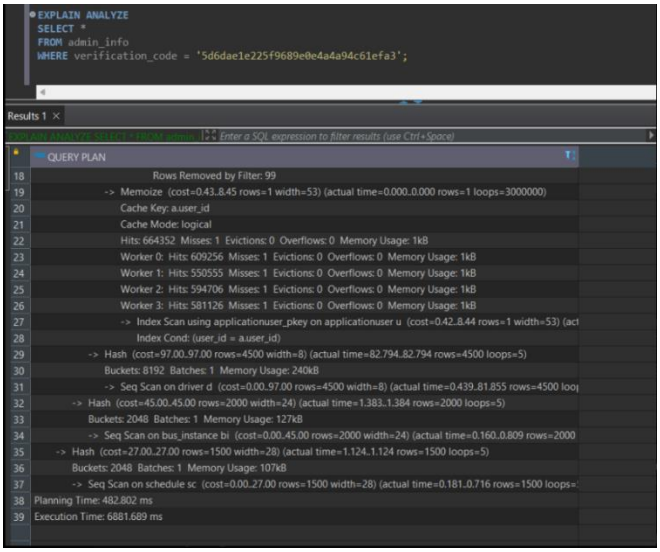




Иако UPDATE операцијата имаше повисоко време на извршување, execution plan покажува дека PostgreSQL го користи индексот station_pkey. Дополнителната проверка не покажа lock или idle transaction проблем, па високото време се третира како варијација поради shared серверската околина, а не како недостаток на индекс.

View 6: **admin_info**

Погледот **admin_info** се користи за приказ на активности поврзани со даден администратор, вклучувајќи ги возачите, автобусите и распоредите што ги администрира. Примарен филтер за овој поглед е **verification_code** на администраторот.
Пред оптимизацијата, времето на извршување изнесуваше **6881.689 ms,** што е над дозволената граница од 2 секунди.

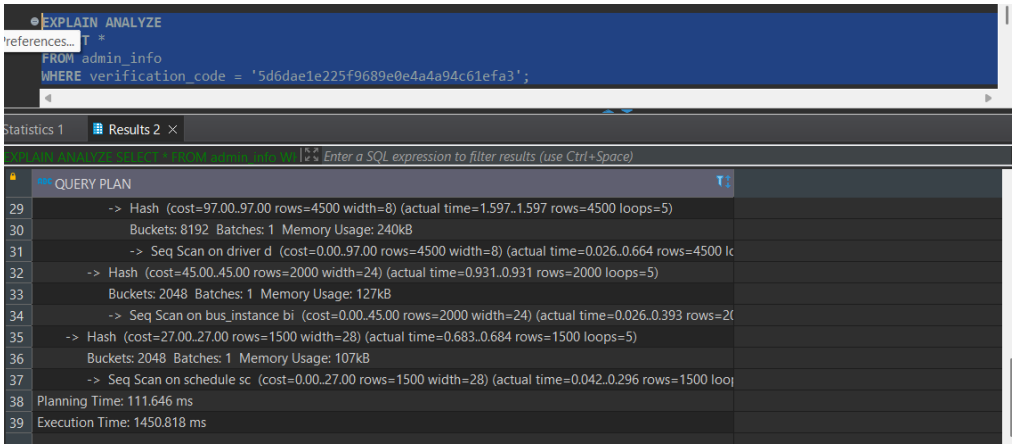


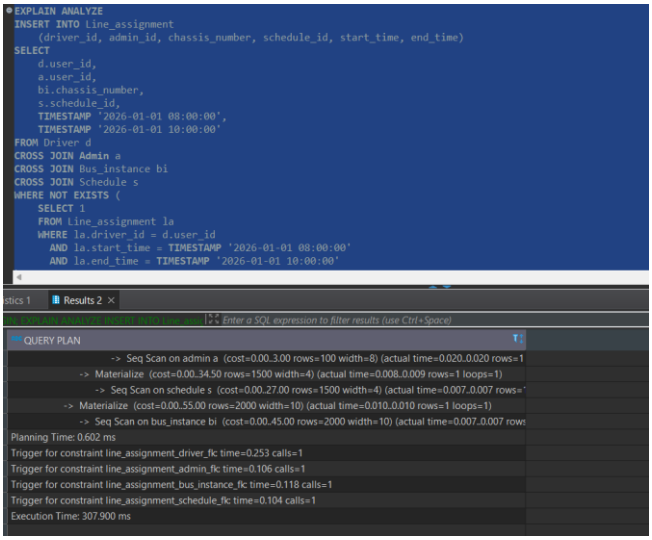
Во execution plan се забележуваше дека најголем дел од времето се троши при join операциите со табелата Line_assignment, која е најобемна и најважна во овој поглед.

За оптимизација беа додадени индекси над колоните што учествуваат во поврзувањата и филтрирањето.

```sql
CREATE INDEX IF NOT EXISTS idx_line_assignment_admin_id
ON Line_assignment(admin_id);

CREATE INDEX IF NOT EXISTS idx_line_assignment_driver_id
ON Line_assignment(driver_id);

CREATE INDEX IF NOT EXISTS idx_line_assignment_schedule_id
ON Line_assignment(schedule_id);

CREATE INDEX IF NOT EXISTS idx_line_assignment_chassis_number
ON Line_assignment(chassis_number);
```

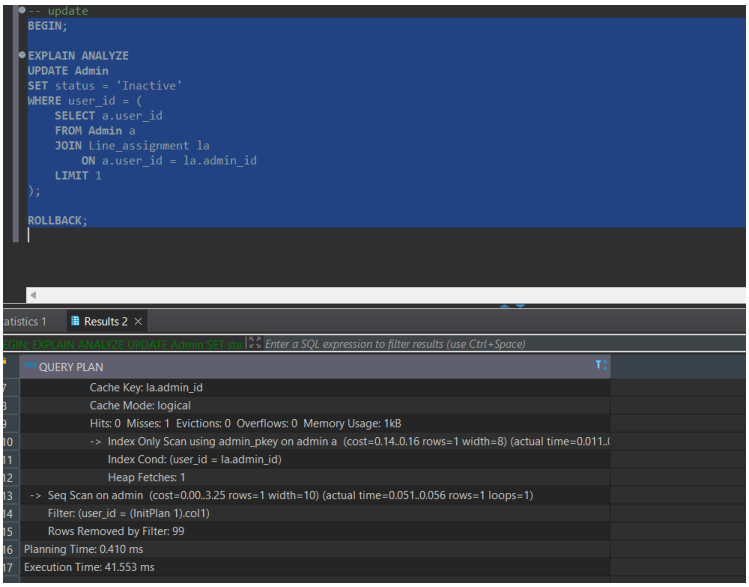
По додавањето на индексите и повторното извршување на ANALYZE, времето на извршување се намали на **1450.234 ms**. Ова време е под 2 секунди и е прифатливо за апликацијата.





Дополнително беа измерени **INSERT** и **UPDATE** операции врз основните табели што го формираат погледот.

- INSERT операцијата пред оптимизацијата изнесуваше околу 10 секунди, а по додавањето на индексите се намали на **307.9 ms**;
- UPDATE операцијата пред оптимизацијата изнесуваше околу 7 секунди, а по оптимизацијата се намали на **41.553 ms;**

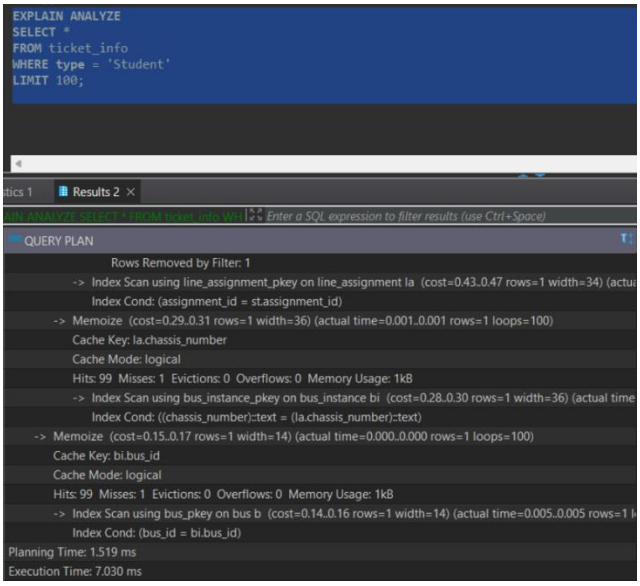


Сите времиња се под 2 секунди, **па оптимизацијата е прифатлива за апликацијата.**

View 7: **ticket_info**

Погледот **ticket_info** се користи за приказ на информации за билети според тип на корисник. Примарен филтер за овој поглед е **type = 'Student'**. Тестирањето беше направено со LIMIT 100, бидејќи во реална апликација резултатите би се прикажувале странично, а не сите одеднаш.

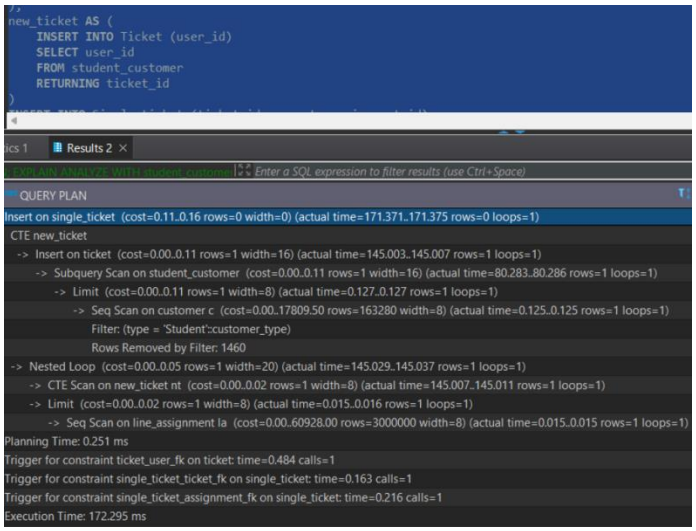
Времето на извршување на SELECT query-то изнесуваше **7.030 ms**, што е далеку под дозволената граница од 2 секунди.

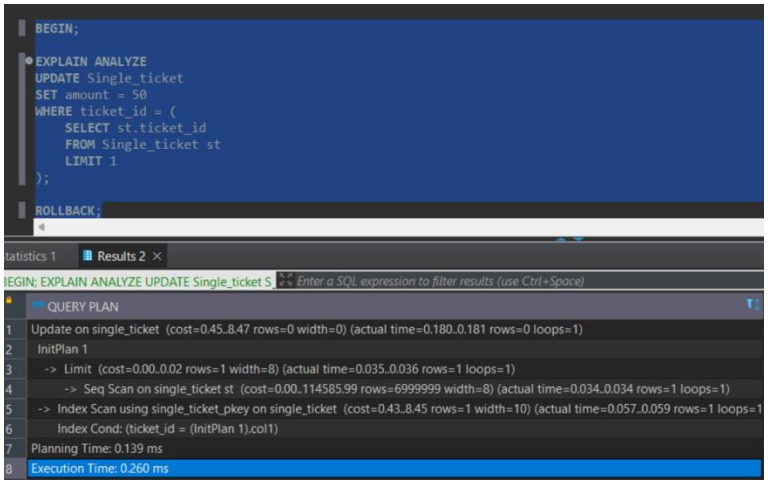


Во execution plan, се гледа дека PostgreSQL веќе користи постоечки индекси, како line_assignment_pkey, bus_instance_pkey и bus_pkey. Поради добрите перформанси **не беше потребно дополнително индексирање** за овој поглед.

Дополнително беа измерени **INSERT** и **UPDATE** операции врз основните табели Ticket и Single_ticket, бидејќи ticket_info е обичен view и не складира податоци самостојно.

- INSERT тестот симулираше креирање нов билет за корисник од тип Student и негово внесување како single ticket. Времето на извршување изнесуваше **172.295 ms**;
- UPDATE тестот симулираше ажурирање на запис во Single_ticket, при што amount останува 50 поради check constraint во табелата. Времето на извршување изнесуваше 0.260 ms;



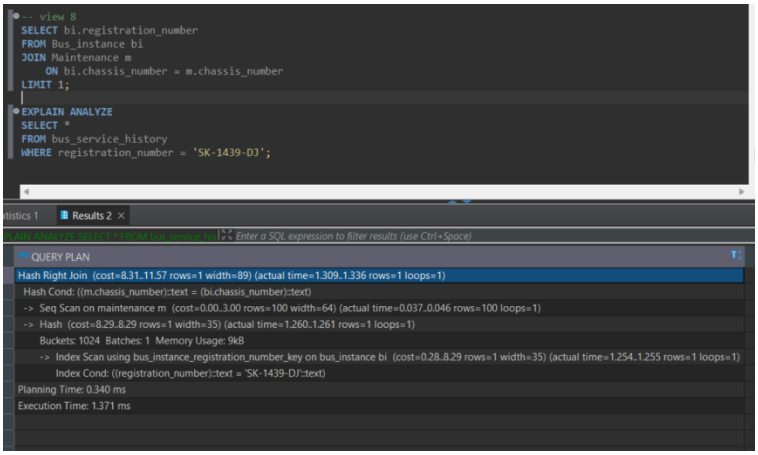


Сите времиња се под 2 секунди, **па оптимизацијата е прифатлива за апликацијата**.

View 8: **bus_service_history**

Погледот **bus_service_history** се користи за приказ на сервисната историја на даден автобус. Примарен филтер за овој поглед е **registration_number**.

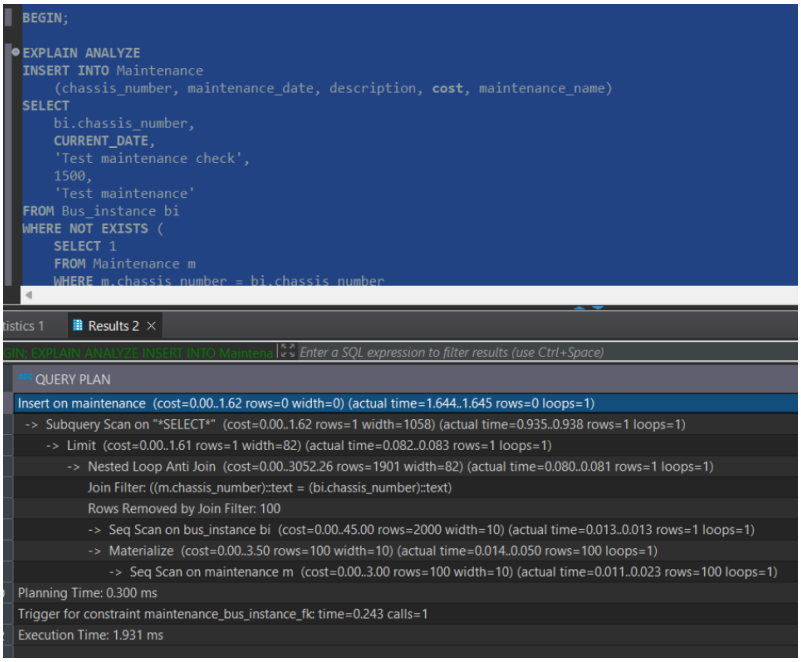
Времето на извршување изнесуваше **1.371 ms**, што е далеку под дозволената граница од 2 секунди.

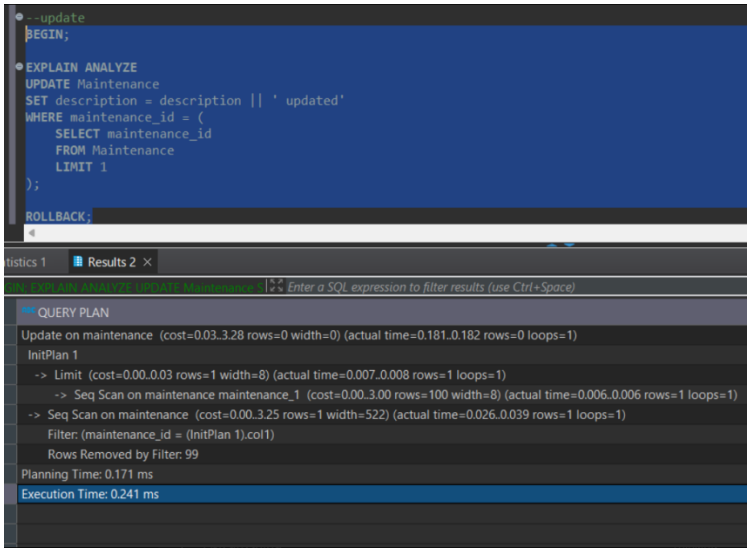


Во execution plan се гледа дека PostgreSQL го користи постоечкиот индекс bus_instance_registration_number_key, кој автоматски е креиран поради UNIQUE ограничувањето на колоната registration_number. Поради тоа **не беше потребно дополнително индексирање** за овој поглед.

Дополнително беа измерени **INSERT** и **UPDATE** операции врз табелата Maintenance, бидејќи bus_service_history е обичен view и не складира податоци самостојно.

- INSERT тестот симулираше додавање нов сервисен запис за автобус. Времето на извршување изнесуваше **1.931 ms;**
- UPDATE тестот симулираше промена на описот на сервисен запис во табелата Maintenance. Времето на извршување изнесуваше **0.241 ms;**





Сите измерени времиња се значително под 2 секунди, па за овој поглед **не беше потребно дополнително индексирање**. Погледот е прифатлив за апликацијата.

View 9: **monthly_ticket_sales**

Погледот **monthly_ticket_sales** се користи за месечен преглед на продадени билети и вкупен приход според начин на плаќање. Примарен филтер за овој поглед е **payment_month = '2025-02-01'**.

Овој поглед е најсложен од анализираните погледи, бидејќи врши агрегација врз повеќе големи табели: Payment, Customer_Payment_Ticket, Ticket, Single_ticket, Pass_ticket, Pass_type и Payment_type.

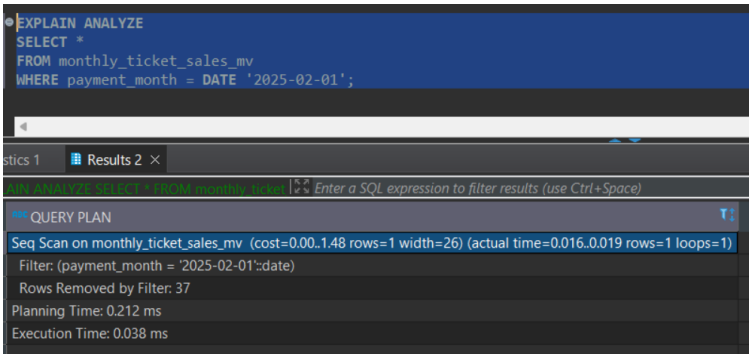
Првичното време на извршување изнесуваше **199499.209 ms**, што е далеку над дозволената граница од 2 секунди.

Во execution plan се забележува Parallel Seq Scan на табелата Payment и голем број join операции со останатите табели. Главниот проблем е што обичниот view ја пресметува агрегацијата повторно при секое повикување.

Поради тоа беше креиран materialized view monthly_ticket_sales_mv. Со materialized view резултатите од агрегацијата се претходно пресметани и зачувани.

```
CREATE MATERIALIZED VIEW monthly_ticket_sales_mv AS
SELECT *
FROM monthly_ticket_sales;
```

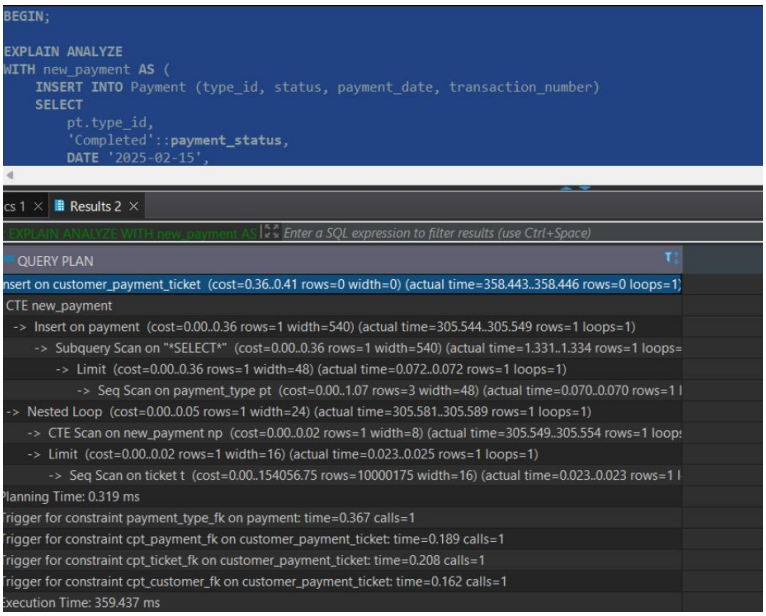
По креирањето на materialized view, времето на извршување се намали на **0.038 ms,** што е значително под дозволената граница од 2 секунди.

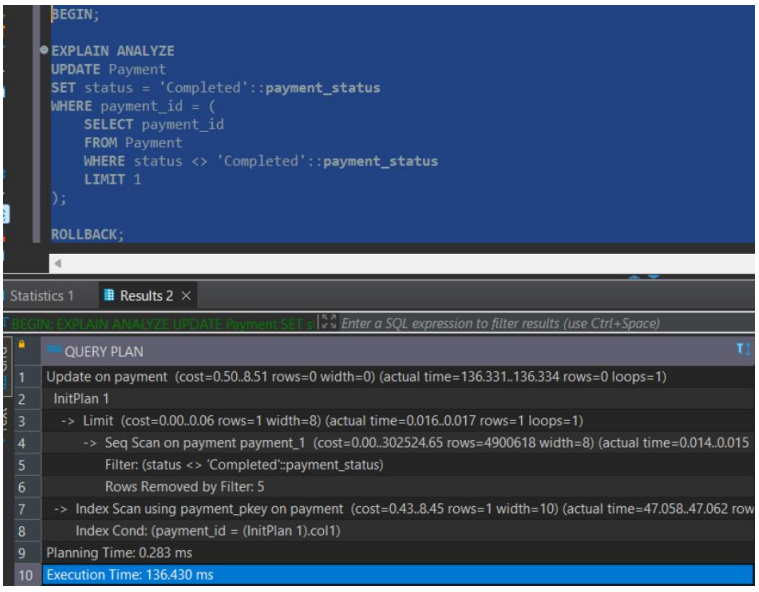


Дополнително беа измерени **INSERT** и **UPDATE** операции врз основните табели што го формираат погледот. Бидејќи monthly_ticket_sales е обичен view, а monthly_ticket_sales_mv е materialized view, податоците не се внесуваат директно во view-то, туку во табелите Payment и Customer_Payment_Ticket.

- INSERT тестот симулираше додавање ново успешно плаќање и негово поврзување со постоечки билет преку Customer_Payment_Ticket. Времето на извршување изнесуваше **359.437 ms;**

- UPDATE тестот симулираше промена на статус на плаќање во Completed. Времето на извршување изнесуваше **136.430 ms;**





Сите измерени времиња се под 2 секунди, па **овој поглед е прифатлив за апликацијата.**