

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.

```
SELECT d.verification_code
FROM Driver d
JOIN Line_assignment la
ON d.user_id = la.driver_id
LIMIT 1;

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

QUERY PLAN
4  Hash Cond: (la.schedule_id = s.schedule_id)
5  -> Nested Loop (cost=1.43..2461.08 rows=667 width=88) (actual time=1115.218..7106.693 rows=89999 loops=1)
6  -> Nested Loop (cost=1.14..2443.82 rows=667 width=77) (actual time=745.758..6689.921 rows=89999 loops=1)
7  -> Nested Loop (cost=0.71..16.74 rows=1 width=63) (actual time=555.633..555.637 rows=1 loops=1)
8  -> Index Scan using driver_verification_code_key on driver d (cost=0.28..8.30 rows=1 width=41) (actual time=185.854..185.857 rows=1 loops=1)
9  Index Cond: ((verification_code)=text = '97275a23ca44226c9964043c8462be96'=text)
10 -> Index Scan using applicationuser_pkey on applicationuser au (cost=0.42..8.44 rows=1 width=22) (actual time=369.768..369.768 rows=1 loops=1)
11 Index Cond: (user_id = d.user_id)
12 -> Index Scan using unique_driver_shift on line_assignment la (cost=0.43..1544.73 rows=88235 width=38) (actual time=190.116..6118.742 rows=1 loops=1)
13 Index Cond: (driver_id = au.user_id)
14 -> Memoize (cost=0.25..0.31 rows=1 width=21) (actual time=0.004..0.004 rows=1 loops=89999)
15 Cache Key: la.chassis_number
16 Cache Mode: logical
17 Hits: 89998 Misses: 1 Evictions: 0 Overflows: 0 Memory Usage: 1kB
18 -> Index Scan using bus_instance_pkey on bus_instance bi (cost=0.28..0.30 rows=1 width=21) (actual time=369.442..369.443 rows=1 loops=1)
19 Index Cond: ((chassis_number)=text = (la.chassis_number)=text)
20 -> Hash (cost=27.00..27.00 rows=1500 width=8) (actual time=367.628..367.629 rows=1500 loops=1)
21 Buckets: 2048 Batches: 1 Memory Usage: 75kB
22 -> Seq Scan on schedules s (cost=0.00..27.00 rows=1500 width=8) (actual time=206.496..367.205 rows=1500 loops=1)
23 -> Hash (cost=2.50..2.50 rows=150 width=8) (actual time=0.090..0.091 rows=150 loops=1)
24 Buckets: 1024 Batches: 1 Memory Usage: 14kB
25 -> Seq Scan on line l (cost=0.00..2.50 rows=150 width=8) (actual time=0.018..0.040 rows=150 loops=1)
26 Planning Time: 1.828 ms
27 Execution Time: 7528.971 ms
```

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);

```

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.

```

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

```

Step	Operation	Cost	Rows	Width	Actual Time	Actual Rows	Loops
6	-> Nested Loop	1.14..2443.82	rows=667	width=77	0.201..48.576	rows=89999	loops=1
7	-> Nested Loop	0.71..16.74	rows=1	width=63	0.123..0.127	rows=1	loops=1
8	-> Index Scan using driver_verification_code_key on driver d	0.28..8.30	rows=1	width=41	0.078..0.080	rows=1	loops=1
9	Index Cond: ((verification_code)::text = '97275a23ca44226c9964043c8462be96'::text)						
10	-> Index Scan using applicationuser_pkey on applicationuser au	0.42..8.44	rows=1	width=22	0.041..0.041	rows=1	loops=1
11	Index Cond: (user_id = d.user_id)						
12	-> Index Scan using unique_driver_shift on line_assignment la	0.43..1544.73	rows=88235	width=38	0.076..32.413	rows=89999	loops=1
13	Index Cond: (driver_id = au.user_id)						
14	-> Memoize	0.29..0.31	rows=1	width=21	0.000..0.000	rows=1	loops=89999
15	Cache Key: la.chassis_number						
16	Cache Mode: logical						
17	Hits: 89998 Misses: 1 Evictions: 0 Overflows: 0 Memory Usage: 1kB						
18	-> Index Scan using bus_instance_pkey on bus_instance bi	0.28..0.30	rows=1	width=21	0.043..0.044	rows=1	loops=1
19	Index Cond: ((chassis_number)::text = (la.chassis_number)::text)						
20	-> Hash	27.00..27.00	rows=1500	width=8	0.536..0.537	rows=1500	loops=1
21	Buckets: 2048 Batches: 1 Memory Usage: 75kB						
22	-> Seq Scan on schedule s	0.00..27.00	rows=1500	width=8	0.013..0.294	rows=1500	loops=1
23	-> Hash	2.50..2.50	rows=150	width=8	0.070..0.070	rows=150	loops=1
24	Buckets: 1024 Batches: 1 Memory Usage: 14kB						
25	-> Seq Scan on line l	0.00..2.50	rows=150	width=8	0.021..0.043	rows=150	loops=1
26	Planning Time: 2.548 ms						
27	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.

```

-- View 2
SELECT line_number
FROM Line
WHERE is_active = true
LIMIT 1;

EXPLAIN ANALYZE
SELECT *
FROM line_info
WHERE line_number = 1;

```

Results 1

QUERY PLAN

1	Nested Loop (cost=0.56..19.48 rows=1 width=98) (actual time=0.253..0.269 rows=1 loops=1)
2	-> Nested Loop (cost=0.28..11.18 rows=1 width=55) (actual time=0.199..0.214 rows=1 loops=1)
3	-> Seq Scan on line 1 (cost=0.00..2.88 rows=1 width=12) (actual time=0.031..0.045 rows=1 loops=1)
4	Filter: (is_active AND (line_number = 1))
5	Rows Removed by Filter: 149
6	-> Index Scan using station_pkey on station s_start (cost=0.28..8.29 rows=1 width=51) (actual time=0.164..0.164 rows=1 loops=1)
7	Index Cond: (station_id = l.start_station_id)
8	-> Index Scan using station_pkey on station s_end (cost=0.28..8.29 rows=1 width=51) (actual time=0.051..0.051 rows=1 loops=1)
9	Index Cond: (station_id = l.end_station_id)
10	Planning Time: 0.434 ms
11	Execution Time: 0.322 ms

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.

```

EXPLAIN ANALYZE
SELECT *
FROM payments_info
WHERE payment_status = 'Completed';

```

Results 1

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

QUERY PLAN

1	Hash Join (cost=1.07..366300.02 rows=10071266 width=21) (actual time=10.454..4446.217 rows=10050348 loops=1)
2	Hash Cond: (p.type_id = pt.type_id)
3	-> Seq Scan on payment p (cost=0.00..302514.26 rows=10071266 width=24) (actual time=10.380..2410.577 rows=10050348 loops=1)
4	Filter: (status = 'Completed';payment_status)
5	Rows Removed by Filter: 4949652
6	-> Hash (cost=1.03..1.03 rows=3 width=13) (actual time=0.040..0.042 rows=3 loops=1)
7	Buckets: 1024 Batches: 1 Memory Usage: 9kB
8	-> Seq Scan on payment_type pt (cost=0.00..1.03 rows=3 width=13) (actual time=0.026..0.028 rows=3 loops=1)
9	Planning Time: 1.398 ms
10	JIT:
11	Functions: 13
12	Options: Inlining false, Optimization false, Expressions true, Deforming true
13	Timing: Generation 1.062 ms (Deform 0.524 ms), Inlining 0.000 ms, Optimization 0.665 ms, Emission 9.511 ms, Total 11.238 ms
14	Execution Time: 4786.492 ms

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.

```

ORDER BY num_of_station;
EXPLAIN ANALYZE
SELECT *
FROM line_stations
WHERE line_number = 1
ORDER BY num_of_station;

```

Results 1

Results 2

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

QUERY PLAN

3	Sort Method: quicksort Memory: 26kB
4	-> Nested Loop (cost=3.17..49.69 rows=15 width=59) (actual time=0.068..0.523 rows=15 loops=1)
5	-> Hash Join (cost=2.89..44.46 rows=15 width=12) (actual time=0.053..0.458 rows=15 loops=1)
6	Hash Cond: (p.line_id = l.line_id)
7	-> Seq Scan on "position" p (cost=0.00..35.50 rows=2250 width=12) (actual time=0.015..0.182 rows=2250 loops=1)
8	-> Hash (cost=2.88..2.88 rows=1 width=8) (actual time=0.027..0.028 rows=1 loops=1)
9	Buckets: 1024 Batches: 1 Memory Usage: 9kB
10	-> Seq Scan on line l (cost=0.00..2.88 rows=1 width=8) (actual time=0.013..0.024 rows=1 loops=1)
11	Filter: (line_number = 1)
12	Rows Removed by Filter: 149
13	-> Index Scan using station_pkey on station s (cost=0.28..0.35 rows=1 width=51) (actual time=0.004..0.004 rows=1 loops=15)
14	Index Cond: (station_id = p.station_id)
15	Planning Time: 0.510 ms
16	Execution Time: 0.590 ms

Value

Sort (cost=49.98..59.02 rows=15 width=59) (act

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
  FROM target_line tl
  JOIN Position p ON tl.line_id = p.line_id
)
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

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

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 (costs=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 (costs=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=1)
-> 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 rows=1 loops=1)
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.

```

EXPLAIN ANALYZE
SELECT *
FROM station_info
WHERE station_name = 'ul. Makedonska Brigada jug 3738';

Results 1 x
QUERY PLAN
1  Nested Loop (cost=0.98..25.45 rows=10 width=59) (actual time=0.153..0.155 rows=0 loops=1)
2  -> Nested Loop (cost=0.70..24.87 rows=1 width=59) (actual time=0.152..0.154 rows=0 loops=1)
3  -> Nested Loop (cost=0.56..24.70 rows=1 width=51) (actual time=0.152..0.153 rows=0 loops=1)
4  -> Index Scan using station_station_name_key on station s (cost=0.28..8.29 rows=1 width=51) (actual time=0.076..0.078 rows=1 loops=1)
5  Index Cond: ((station_name)::text = 'ul. Makedonska Brigada jug 3738'::text)
6  -> Index Only Scan using position_pkey on "position" p (cost=0.28..14.90 rows=150 width=8) (actual time=0.068..0.068 rows=0 loops=1)
7  Index Cond: (station_id = s.station_id)
8  Heap Fetches: 0
9  -> Index Scan using line_pkey on line l (cost=0.14..0.17 rows=1 width=8) (never executed)
10 Index Cond: (line_id = p.line_id)
11 Filter: is_active
12 -> Index Scan using idx_schedule_line_id on schedule sc (cost=0.28..0.48 rows=10 width=12) (never executed)
13 Index Cond: (line_id = p.line_id)
14 Planning Time: 1.349 ms
15 Execution Time: 0.222 ms

```

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

```

Results 1 X
Enter a SQL expression to filter results (use Ctrl+Space)

QUERY PLAN
1 Insert on "position" (cost=8.80..8.85 rows=0 width=0) (actual time=807.332..807.338 rows=0 loops=1)
2 CTE new_station
3 -> Insert on station (cost=0.00..0.01 rows=1 width=1040) (actual time=399.340..399.344 rows=1 loops=1)
4 -> Result (cost=0.00..0.01 rows=1 width=1040) (actual time=195.366..195.368 rows=1 loops=1)
5 -> Nested Loop (cost=8.79..8.83 rows=1 width=12) (actual time=399.421..399.434 rows=1 loops=1)
6 -> CTE Scan on new_station ns (cost=0.00..0.02 rows=1 width=4) (actual time=399.345..399.351 rows=1 loops=1)
7 -> HashAggregate (cost=8.79..8.80 rows=1 width=8) (actual time=0.071..0.077 rows=1 loops=1)
8 Group Key: line.line_id
9 Batches: 1 Memory Usage: 24kB
10 -> Nested Loop Left Join (cost=0.28..8.71 rows=15 width=8) (actual time=0.051..0.058 rows=15 loops=1)
11 -> Limit (cost=0.00..0.02 rows=1 width=4) (actual time=0.025..0.027 rows=1 loops=1)
12 -> Seq Scan on line (cost=0.00..2.50 rows=117 width=4) (actual time=0.023..0.023 rows=1 loops=1)
13 Filter: is_active
14 -> Index Only Scan using idx_position_line_num_desc on "position" p (cost=0.28..8.54 rows=15 width=8) (actual time=0.021..0.024 rows=
15 Index Cond: (line_id = line.line_id)
16 Heap Fetches: 0
17 Planning Time: 0.437 ms
18 Trigger for constraint position_station_fk on position: time=0.252 calls=1
19 Trigger for constraint position_line_fk on position: time=0.108 calls=1
20 Execution Time: 807.786 ms

```

```

Results 1 X
Enter a SQL expression to filter results (use Ctrl+Space)

QUERY PLAN
Update on station (cost=0.30..8.32 rows=0 width=0) (actual time=19810.044..19810.048 rows=0 loops=1)
InitPlan 1
-> Limit (cost=0.00..0.02 rows=1 width=4) (actual time=0.026..0.028 rows=1 loops=1)
-> Seq Scan on station station_1 (cost=0.00..42.00 rows=2000 width=4) (actual time=0.025..0.026 rows=1 loops=1)
-> Index Scan using station_pkey on station (cost=0.28..8.30 rows=1 width=522) (actual time=0.047..0.051 rows=1 loops=1)
Index Cond: (station_id = (InitPlan 1).col1)
Planning Time: 0.164 ms
Execution Time: 19810.098 ms

```

- 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**;

```

EXPLAIN ANALYZE
UPDATE Station
SET address = address || ' updated'
WHERE station_id = (
  SELECT station_id
  FROM Station
  LIMIT 1
);
ROLLBACK;

```

```

Results 1 X
Enter a SQL expression to filter results (use Ctrl+Space)

QUERY PLAN
Update on station (cost=0.30..8.32 rows=0 width=0) (actual time=365.631..365.632 rows=0 loops=1)
InitPlan 1
-> Limit (cost=0.00..0.02 rows=1 width=4) (actual time=364.726..364.727 rows=1 loops=1)
-> Seq Scan on station station_1 (cost=0.00..42.00 rows=2000 width=4) (actual time=364.725..364.725 rows=1 loops=1)
-> Index Scan using station_pkey on station (cost=0.28..8.30 rows=1 width=522) (actual time=365.561..365.563 rows=1 loops=1)
Index Cond: (station_id = (InitPlan 1).col1)
Planning Time: 0.160 ms
Execution Time: 365.764 ms

```

```

Insert on "position" (cost=8.80..8.85 rows=0 width=0) (actual time=39.938..39.940 rows=0 loops=1)
  CTE new_station
    -> Insert on station (cost=0.00..0.01 rows=1 width=1040) (actual time=39.776..39.778 rows=1 loops=1)
    -> Result (cost=0.00..0.01 rows=1 width=1040) (actual time=39.648..39.648 rows=1 loops=1)
    -> Nested Loop (cost=8.79..8.83 rows=1 width=12) (actual time=39.863..39.867 rows=1 loops=1)
    -> CTE Scan on new_station ns (cost=0.00..0.02 rows=1 width=4) (actual time=39.779..39.781 rows=1 loops=1)
    -> HashAggregate (cost=8.79..8.80 rows=1 width=8) (actual time=0.081..0.083 rows=1 loops=1)
        Group Key: line.line_id
        Batches: 1 Memory Usage: 24kB
    -> Nested Loop Left Join (cost=0.28..8.71 rows=15 width=8) (actual time=0.064..0.069 rows=15 loops=1)
        -> Limit (cost=0.00..0.02 rows=1 width=4) (actual time=0.039..0.040 rows=1 loops=1)
        -> Seq Scan on line (cost=0.00..2.50 rows=117 width=4) (actual time=0.038..0.038 rows=1 loops=1)
            Filter: is_active
        -> Index Only Scan using idx_position_line_num_desc on "position" p (cost=0.28..8.54 rows=15 width=8) (actual time=0.022..0.024 r
            Index Cond: (line_id = line.line_id)
            Heap Fetches: 0
  Planning Time: 0.566 ms
  Trigger for constraint position_station_fk on position: time=0.155 calls=1
  Trigger for constraint position_line_fk on position: time=0.105 calls=1
  Execution Time: 40.281 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.

```

EXPLAIN ANALYZE
SELECT *
FROM admin_info
WHERE verification_code = '5d6dae1e225f9689e9e44a94c61efa3';

```

Results 1 x

QUERY PLAN

```

18      Rows Removed by Filter: 99
19      -> Memoize (cost=0.43..8.45 rows=1 width=53) (actual time=0.000..0.000 rows=1 loops=300000)
20          Cache Key: a.user_id
21          Cache Model: logical
22          Hits: 664352 Misses: 1 Evictions: 0 Overflows: 0 Memory Usage: 1kB
23          Worker 0: Hits: 609256 Misses: 1 Evictions: 0 Overflows: 0 Memory Usage: 1kB
24          Worker 1: Hits: 550555 Misses: 1 Evictions: 0 Overflows: 0 Memory Usage: 1kB
25          Worker 2: Hits: 594706 Misses: 1 Evictions: 0 Overflows: 0 Memory Usage: 1kB
26          Worker 3: Hits: 581126 Misses: 1 Evictions: 0 Overflows: 0 Memory Usage: 1kB
27      -> Index Scan using applicationuser_pkey on applicationuser u (cost=0.42..8.44 rows=1 width=53) (act
          Index Cond: (user_id = a.user_id)
28      -> Hash (cost=97.00..97.00 rows=4500 width=8) (actual time=82.794..82.794 rows=4500 loops=5)
          Buckets: 8192 Batches: 1 Memory Usage: 240kB
          -> Seq Scan on driver d (cost=0.00..97.00 rows=4500 width=8) (actual time=0.439..81.855 rows=4500 loop
          -> Hash (cost=45.00..45.00 rows=2000 width=24) (actual time=1.383..1.384 rows=2000 loops=5)
          Buckets: 2048 Batches: 1 Memory Usage: 127kB
          -> Seq Scan on bus_instance bi (cost=0.00..45.00 rows=2000 width=24) (actual time=0.160..0.809 rows=2000
          -> Hash (cost=27.00..27.00 rows=1500 width=28) (actual time=1.124..1.124 rows=1500 loops=5)
          Buckets: 2048 Batches: 1 Memory Usage: 107kB
          -> Seq Scan on schedule sc (cost=0.00..27.00 rows=1500 width=28) (actual time=0.181..0.716 rows=1500 loops=
38  Planning Time: 482.802 ms
39  Execution Time: 6881.689 ms

```

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.

```
● 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.

The screenshot shows a query execution plan for an ANALYZE operation. The query filters for a specific verification code. The plan details the cost, actual time, rows, and memory usage for each step, including hash joins and sequential scans on various tables.

Line	Operation	Cost	Actual Time	Rows	Width	Loops	Memory Usage
29	Hash	97.00.97.00	1.597.1.597	4500	8	5	240kB
30	Buckets	8192					
31	Seq Scan on driver d	0.00.97.00	0.026.0.664	4500	8		
32	Hash	45.00.45.00	0.931.0.931	2000	24	5	127kB
33	Buckets	2048					
34	Seq Scan on bus_instance bi	0.00.45.00	0.026.0.393	2000	24		
35	Hash	27.00.27.00	0.683.0.684	1500	28	5	107kB
36	Buckets	2048					
37	Seq Scan on schedule sc	0.00.27.00	0.042.0.296	1500	28		
38	Planning Time		111.646 ms				
39	Execution Time		1450.818 ms				

```

EXPLAIN ANALYZE
INSERT INTO Line_assignment
(driver_id, admin_id, chassis_number, schedule_id, start_time, end_time)
SELECT
d.user_id,
a.user_id,
bi.chassis_number,
s.schedule_id,
TIMESTAMP '2026-01-01 08:00:00',
TIMESTAMP '2026-01-01 10:00:00'
FROM Driver d
CROSS JOIN Admin a
CROSS JOIN Bus_instance bi
CROSS JOIN Schedule s
WHERE NOT EXISTS (
SELECT 1
FROM Line_assignment la
WHERE la.driver_id = d.user_id
AND la.start_time = TIMESTAMP '2026-01-01 08:00:00'
AND la.end_time = TIMESTAMP '2026-01-01 10:00:00'
)

```

Results 2 x

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

QUERY PLAN

```

-- Seq Scan on admin a (cost=0.00.3.00 rows=100 width=8) (actual time=0.020.0.020 rows=1)
-- Materialize (cost=0.00.34.50 rows=1500 width=4) (actual time=0.008.0.009 rows=1 loops=1)
-- Seq Scan on schedule s (cost=0.00.27.00 rows=1500 width=4) (actual time=0.007.0.007 rows=1)
-- Materialize (cost=0.00.55.00 rows=2000 width=10) (actual time=0.010.0.010 rows=1 loops=1)
-- Seq Scan on bus_instance bi (cost=0.00.45.00 rows=2000 width=10) (actual time=0.007.0.007 rows=1)
Planning Time: 0.602 ms
Trigger for constraint line_assignment_driver_fc time=0.253 calls=1
Trigger for constraint line_assignment_admin_fc time=0.106 calls=1
Trigger for constraint line_assignment_bus_instance_fc time=0.118 calls=1
Trigger for constraint line_assignment_schedule_fc time=0.104 calls=1
Execution Time: 307.900 ms

```

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**;

```

-- update
BEGIN;
EXPLAIN ANALYZE
UPDATE Admin
SET status = 'Inactive'
WHERE user_id = (
SELECT a.user_id
FROM Admin a
JOIN Line_assignment la
ON a.user_id = la.admin_id
LIMIT 1
);
ROLLBACK;

```

Results 2 x

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

QUERY PLAN

```

Cache Key: la.admin_id
Cache Mode: logical
Hits: 0 Misses: 1 Evictions: 0 Overflows: 0 Memory Usage: 1kB
-- Index Only Scan using admin_pkey on admin a (cost=0.14.0.16 rows=1 width=8) (actual time=0.011.0.011 rows=1)
Index Cond: (user_id = la.admin_id)
Heap Fetches: 1
-- Seq Scan on admin (cost=0.00.3.25 rows=1 width=10) (actual time=0.051.0.056 rows=1 loops=1)
Filter: (user_id = (InitPlan 1).col1)
Rows Removed by Filter: 99
Planning Time: 0.410 ms
Execution Time: 41.553 ms

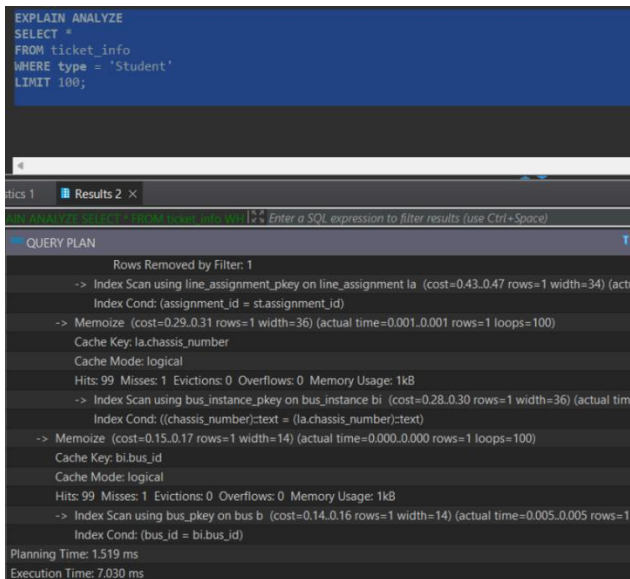
```

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.



```
EXPLAIN ANALYZE
SELECT *
FROM ticket_info
WHERE type = 'Student'
LIMIT 100;
```

Results 2 x

QUERY PLAN

Rows Removed by Filter: 1

- > Index Scan using line_assignment_pkey on line_assignment la (cost=0.43..0.47 rows=1 width=34) (actual time=0.000..0.000 rows=1 loops=1)
Index Cond: (assignment_id = st.assignment_id)
- > Memoize (cost=0.29..0.31 rows=1 width=36) (actual time=0.001..0.001 rows=1 loops=100)
Cache Key: la.chassis_number
Cache Mode: logical
Hits: 99 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=36) (actual time=0.000..0.000 rows=1 loops=1)
Index Cond: ((chassis_number)=text = (la.chassis_number)::text)
- > Memoize (cost=0.15..0.17 rows=1 width=14) (actual time=0.000..0.000 rows=1 loops=100)
Cache Key: bi.bus_id
Cache Mode: logical
Hits: 99 Misses: 1 Evictions: 0 Overflows: 0 Memory Usage: 1kB
- > Index Scan using bus_pkey on bus b (cost=0.14..0.16 rows=1 width=14) (actual time=0.005..0.005 rows=1 loops=1)
Index Cond: (bus_id = bi.bus_id)

Planning Time: 1.519 ms
Execution Time: 7.030 ms

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;

```

new_ticket AS (
  INSERT INTO Ticket (user_id)
  SELECT user_id
  FROM student_customer
  RETURNING ticket_id
)

```

Results 2

EXPLAIN ANALYZE WITH student_customer | Enter a SQL expression to filter results (use Ctrl+Space)

QUERY PLAN

Insert on single_ticket (cost=0.11..0.16 rows=0 width=0) (actual time=171.371..171.375 rows=0 loops=1)

CTE new_ticket

-> Insert on ticket (cost=0.00..0.11 rows=1 width=16) (actual time=145.003..145.007 rows=1 loops=1)

-> Subquery Scan on student_customer (cost=0.00..0.11 rows=1 width=16) (actual time=80.283..80.286 rows=1 loops=1)

-> Limit (cost=0.00..0.11 rows=1 width=8) (actual time=0.127..0.127 rows=1 loops=1)

-> Seq Scan on customer c (cost=0.00..1.7809.50 rows=163280 width=8) (actual time=0.125..0.125 rows=1 loops=1)

Filter: (type = 'Student':customer_type)

Rows Removed by Filter: 1460

-> Nested Loop (cost=0.00..0.05 rows=1 width=20) (actual time=145.029..145.037 rows=1 loops=1)

-> CTE Scan on new_ticket nt (cost=0.00..0.02 rows=1 width=8) (actual time=145.007..145.011 rows=1 loops=1)

-> Limit (cost=0.00..0.02 rows=1 width=8) (actual time=0.015..0.016 rows=1 loops=1)

-> Seq Scan on line_assignment la (cost=0.00..6.0928.00 rows=300000 width=8) (actual time=0.015..0.015 rows=1 loops=1)

Planning Time: 0.251 ms

Trigger for constraint ticket_user_fk on ticket: time=0.484 calls=1

Trigger for constraint single_ticket_ticket_fk on single_ticket: time=0.163 calls=1

Trigger for constraint single_ticket_assignment_fk on single_ticket: time=0.216 calls=1

Execution Time: 172.295 ms

```

BEGIN;
EXPLAIN ANALYZE
UPDATE Single_ticket
SET amount = 50
WHERE ticket_id = (
  SELECT st.ticket_id
  FROM Single_ticket st
  LIMIT 1
);
ROLLBACK;

```

Results 2

EXPLAIN ANALYZE UPDATE Single_ticket S | Enter a SQL expression to filter results (use Ctrl+Space)

QUERY PLAN

1 Update on single_ticket (cost=0.45..8.47 rows=0 width=0) (actual time=0.180..0.181 rows=0 loops=1)

2 InitPlan 1

3 -> Limit (cost=0.00..0.02 rows=1 width=8) (actual time=0.035..0.036 rows=1 loops=1)

4 -> Seq Scan on single_ticket st (cost=0.00..1.14585.99 rows=6999999 width=8) (actual time=0.034..0.034 rows=1 loops=1)

5 -> Index Scan using single_ticket_pkey on single_ticket (cost=0.43..8.45 rows=1 width=10) (actual time=0.057..0.059 rows=1 loops=1)

6 Index Cond: (ticket_id = (InitPlan 1).col1)

7 Planning Time: 0.139 ms

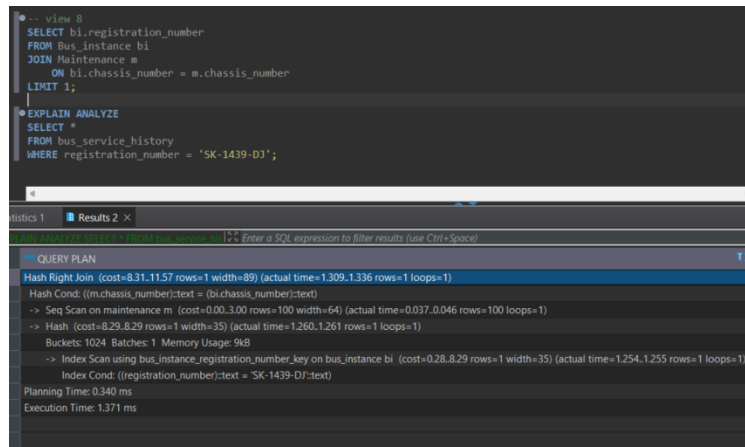
8 Execution Time: 0.260 ms

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.



```
-- view 8
SELECT bi.registration_number
FROM bus_instance bi
JOIN maintenance m
  ON bi.chassis_number = m.chassis_number
LIMIT 1;

EXPLAIN ANALYZE
SELECT *
FROM bus_service_history
WHERE registration_number = 'SK-1439-DJ';
```

QUERY PLAN

```
Hash Right Join (cost=8.31-11.57 rows=1 width=89) (actual time=1.309-1.336 rows=1 loops=1)
  Hash Cond: ((m.chassis_number)::text = (bi.chassis_number)::text)
  -> Seq Scan on maintenance m (cost=0.00-3.00 rows=100 width=64) (actual time=0.037-0.046 rows=100 loops=1)
  -> Hash (cost=8.29-8.29 rows=1 width=35) (actual time=1.260-1.261 rows=1 loops=1)
        Buckets: 1024 Batches: 1 Memory Usage: 9kB
  -> Index Scan using bus_instance_registration_number_key on bus_instance bi (cost=0.28-8.29 rows=1 width=35) (actual time=1.254-1.255 rows=1 loops=1)
        Index Cond: ((registration_number)::text = 'SK-1439-DJ)::text)
Planning Time: 0.340 ms
Execution Time: 1.371 ms
```

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**;

```

BEGIN;
EXPLAIN ANALYZE
INSERT INTO Maintenance
(chassis_number, maintenance_date, description, cost, maintenance_name)
SELECT
bi.chassis_number,
CURRENT_DATE,
'Test maintenance check',
1500,
'Test maintenance'
FROM Bus_instance bi
WHERE NOT EXISTS (
SELECT 1
FROM Maintenance m
WHERE m.chassis_number = bi.chassis_number
);

```

Statistics 1 Results 2 X

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

QUERY PLAN

Insert on maintenance (cost=0.00.1.62 rows=0 width=0) (actual time=1.644.1.645 rows=0 loops=1)

- > Subquery Scan on "SELECT*" (cost=0.00.1.62 rows=1 width=1058) (actual time=0.935.0.938 rows=1 loops=1)
 - > Limit (cost=0.00.1.61 rows=1 width=82) (actual time=0.082.0.083 rows=1 loops=1)
 - > Nested Loop Anti Join (cost=0.00.3052.26 rows=1901 width=82) (actual time=0.080.0.081 rows=1 loops=1)
 - Join Filter: ((m.chassis_number)::text = (bi.chassis_number)::text)
 - Rows Removed by Join Filter: 100
 - > Seq Scan on bus_instance bi (cost=0.00.45.00 rows=2000 width=10) (actual time=0.013.0.013 rows=1 loops=1)
 - > Materialize (cost=0.00.3.50 rows=100 width=10) (actual time=0.014.0.050 rows=100 loops=1)
 - > Seq Scan on maintenance m (cost=0.00.3.00 rows=100 width=10) (actual time=0.011.0.023 rows=100 loops=1)

Planning Time: 0.300 ms
Trigger for constraint maintenance_bus_instance_fk: time=0.243 calls=1
Execution Time: 1.931 ms

```

--update
BEGIN;
EXPLAIN ANALYZE
UPDATE Maintenance
SET description = description || ' updated'
WHERE maintenance_id = (
SELECT maintenance_id
FROM Maintenance
LIMIT 1
);
ROLLBACK;

```

Statistics 1 Results 2 X

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

QUERY PLAN

Update on maintenance (cost=0.03.3.28 rows=0 width=0) (actual time=0.181.0.182 rows=0 loops=1)

- InitPlan 1
 - > Limit (cost=0.00.0.03 rows=1 width=8) (actual time=0.007.0.008 rows=1 loops=1)
 - > Seq Scan on maintenance maintenance_1 (cost=0.00.3.00 rows=100 width=8) (actual time=0.006.0.006 rows=1 loops=1)
 - > Seq Scan on maintenance (cost=0.00.3.25 rows=1 width=522) (actual time=0.026.0.039 rows=1 loops=1)
 - Filter: (maintenance_id = (InitPlan 1).col1)
 - Rows Removed by Filter: 99

Planning Time: 0.171 ms
Execution Time: 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.

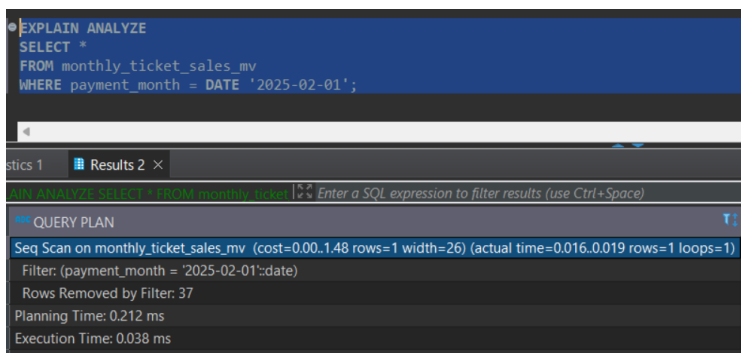
Првичното време на извршување изнесуваше **199499.209 ms**, што е далеку над дозволената граница од 2 секунди.

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.



```
EXPLAIN ANALYZE
SELECT *
FROM monthly_ticket_sales_mv
WHERE payment_month = DATE '2025-02-01';
```

Results 2 x

QUERY PLAN

Seq Scan on monthly_ticket_sales_mv (cost=0.00..1.48 rows=1 width=26) (actual time=0.016..0.019 rows=1 loops=1)

Filter: (payment_month = '2025-02-01'::date)

Rows Removed by Filter: 37

Planning Time: 0.212 ms

Execution Time: 0.038 ms

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**;

```

BEGIN;
EXPLAIN ANALYZE
WITH new_payment AS (
  INSERT INTO Payment (type_id, status, payment_date, transaction_number)
  SELECT
    pt.type_id,
    'Completed'::payment_status,
    DATE '2025-02-15',
)

```

cs 1 x Results 2 x

EXPLAIN ANALYZE WITH new_payment AS (| % Enter a SQL expression to filter results (use Ctrl+Space)

QUERY PLAN

Insert on customer_payment_ticket	(cost=0.36..0.41 rows=0 width=0) (actual time=358.443..358.446 rows=0 loops=1)
CTE new_payment	
-> Insert on payment	(cost=0.00..0.36 rows=1 width=540) (actual time=305.544..305.549 rows=1 loops=1)
-> Subquery Scan on "SELECT"	(cost=0.00..0.36 rows=1 width=540) (actual time=1.331..1.334 rows=1 loops=1)
-> Limit	(cost=0.00..0.36 rows=1 width=48) (actual time=0.072..0.072 rows=1 loops=1)
-> Seq Scan on payment_type pt	(cost=0.00..1.07 rows=3 width=48) (actual time=0.070..0.070 rows=1 loops=1)
-> Nested Loop	(cost=0.00..0.05 rows=1 width=24) (actual time=305.581..305.589 rows=1 loops=1)
-> CTE Scan on new_payment np	(cost=0.00..0.02 rows=1 width=8) (actual time=305.549..305.554 rows=1 loops=1)
-> Limit	(cost=0.00..0.02 rows=1 width=16) (actual time=0.023..0.025 rows=1 loops=1)
-> Seq Scan on ticket t	(cost=0.00..154056.75 rows=10000175 width=16) (actual time=0.023..0.023 rows=1 loops=1)

Planning Time: 0.319 ms
 Trigger for constraint payment_type_fk on payment: time=0.367 calls=1
 Trigger for constraint cpt_payment_fk on customer_payment_ticket: time=0.189 calls=1
 Trigger for constraint cpt_ticket_fk on customer_payment_ticket: time=0.208 calls=1
 Trigger for constraint cpt_customer_fk on customer_payment_ticket: time=0.162 calls=1
 Execution Time: 359.437 ms

```

BEGIN;
EXPLAIN ANALYZE
UPDATE Payment
SET status = 'Completed'::payment_status
WHERE payment_id = (
  SELECT payment_id
  FROM Payment
  WHERE status <> 'Completed'::payment_status
  LIMIT 1
);
ROLLBACK;

```

Statistics 1 Results 2 x

EXPLAIN ANALYZE UPDATE Payment SET s | % Enter a SQL expression to filter results (use Ctrl+Space)

QUERY PLAN

1	Update on payment	(cost=0.50..8.51 rows=0 width=0) (actual time=136.331..136.334 rows=0 loops=1)
2	InitPlan 1	
3	-> Limit	(cost=0.00..0.06 rows=1 width=8) (actual time=0.016..0.017 rows=1 loops=1)
4	-> Seq Scan on payment payment_1	(cost=0.00..302524.65 rows=4900618 width=8) (actual time=0.014..0.015 rows=5 loops=1)
5	Filter: (status <> 'Completed'::payment_status)	
6	Rows Removed by Filter: 5	
7	-> Index Scan using payment_pkey on payment	(cost=0.43..8.45 rows=1 width=10) (actual time=47.058..47.062 rows=1 loops=1)
8	Index Cond: (payment_id = (InitPlan 1).col1)	
9	Planning Time: 0.283 ms	
10	Execution Time: 136.430 ms	

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