Напредни бази на податоци
Фаза 3- Индекси и оптимизација на прашалници
Проект: Smart Parking Management System (SPMS)

Симон Петров – 231203
Александар Ристов – 231252
Ангела Цветковска - 221001

-View1: Available spots per parking

1. Примарен филтер за погледот available_spots_view е според parking_id. Дополнително се користи и статусот на паркинг местото (parking_spot.status = 'free') за филтрирање на слободни паркинг места.

2. Употребата на погледот е прикажување на број на слободни паркинг места по паркинг. Овој поглед се користи во реално време за пребарување на достапни паркинзи и има директно влијание врз корисничкото искуство.

```
EXPLAIN ANALYZE
SELECT * FROM available_spots_view;
```

3. Иницијалното време за извршување на погледот изнесува **10.795 ms**, што е прифатливо за тековната големина на податоците.

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

	QUERY PLAN
1	HashAggregate (cost=465.25..468.84 rows=359 width=23) (actual time=10.665..10.723 rows=359 loops=1)
2	Group Key: p.parking_id
3	Batches: 1 Memory Usage: 61kB
4	-> Hash Join (cost=12.08..399.97 rows=13055 width=19) (actual time=0.320..8.269 rows=13055 loops=1)
5	Hash Cond: (ps.parking_id = p.parking_id)
6	-> Seq Scan on parking_spot ps (cost=0.00..353.23 rows=13055 width=8) (actual time=0.084..5.237 rows=13055 loops=1)
7	Filter: ((status)::text = 'free'::text)
8	Rows Removed by Filter: 5523
9	-> Hash (cost=7.59..7.59 rows=359 width=15) (actual time=0.212..0.212 rows=359 loops=1)
10	Buckets: 1024 Batches: 1 Memory Usage: 25kB
11	-> Seq Scan on parking p (cost=0.00..7.59 rows=359 width=15) (actual time=0.037..0.139 rows=359 loops=1)
12	Planning Time: 1.344 ms
13	Execution Time: 10.795 ms

4. Анализата на извршување (EXPLAIN ANALYZE) покажува дека се користи Sequential Scan (Seq Scan) на табелата **parking_spot**, при што се читаат сите редови и потоа се филтрираат според status = 'free'. Дополнително се извршува Hash Join со табелата **parking**, како и Hash Aggregate за групирање по **parking_id**.

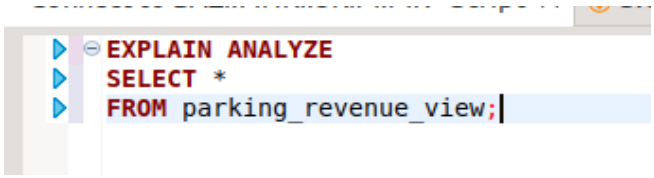
5. Во тековната имплементација не се користат индекси, бидејќи за моменталниот обем на податоци Sequential Scan е доволно ефикасен и не воведува дополнителна комплексност.

Со оглед на моменталните перформанси, погледот е ефикасен и не е потребна дополнителна оптимизација.

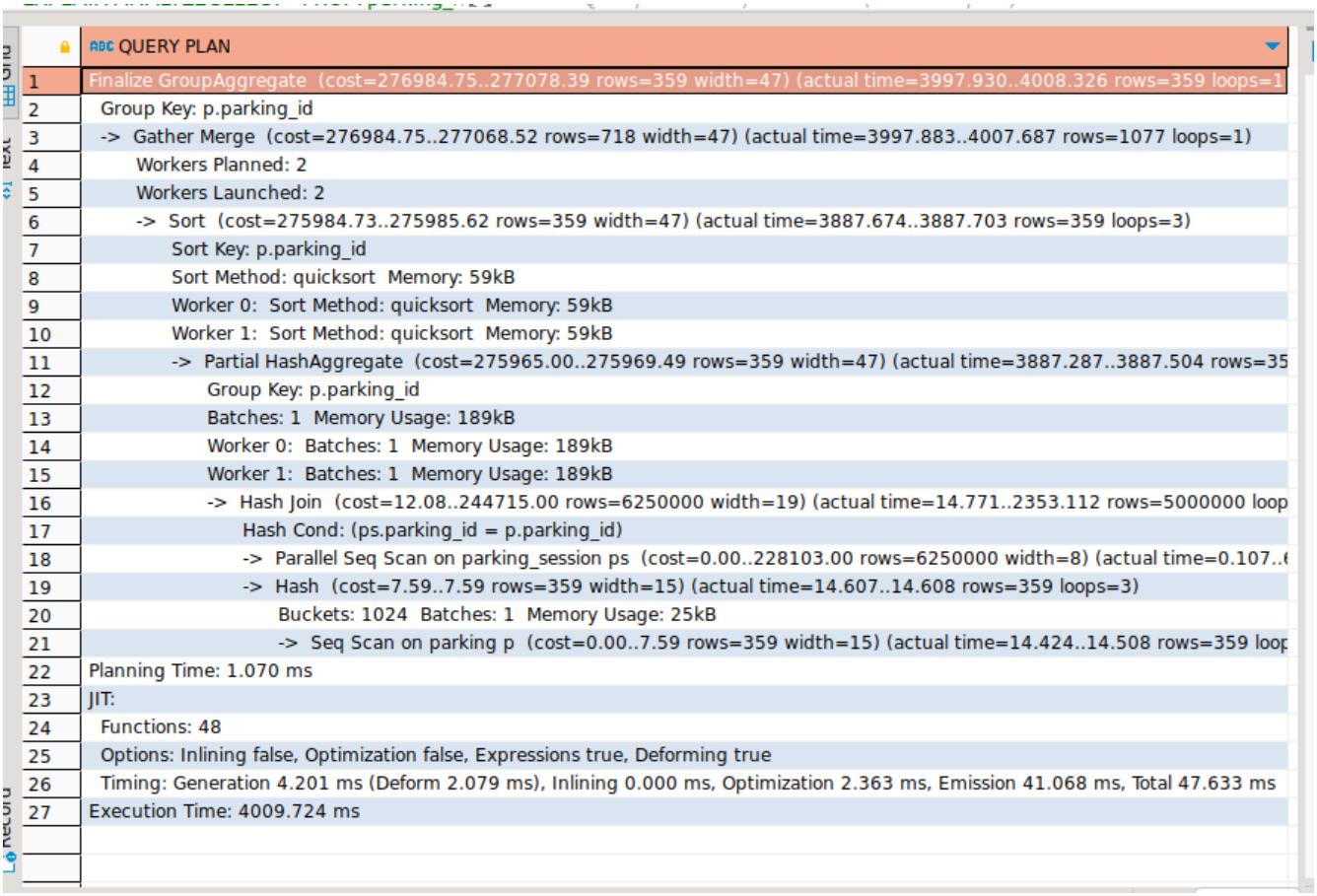
2: View: Revenue per parking

1. Примарен филтер за погледот parking_revenue_view ќе биде според parking_id. Дополнително се користи агрегатната функција SUM(ps.total_amount) за пресметка на вкупниот приход по паркинг.

2. Примарен случај на употреба е прикажување на вкупниот приход по паркинг. Овој поглед се користи во административни извештаи и анализа на заработка,



3. Иницијалното време за извршување на погледот изнесува **4009.724 ms (~4 секунди)**, што е релативно високо.



	QUERY PLAN
1	Finalize GroupAggregate (cost=276984.75..277078.39 rows=359 width=47) (actual time=3997.930..4008.326 rows=359 loops=1
2	Group Key: p.parking_id
3	-> Gather Merge (cost=276984.75..277068.52 rows=718 width=47) (actual time=3997.883..4007.687 rows=1077 loops=1)
4	Workers Planned: 2
5	Workers Launched: 2
6	-> Sort (cost=275984.73..275985.62 rows=359 width=47) (actual time=3887.674..3887.703 rows=359 loops=3)
7	Sort Key: p.parking_id
8	Sort Method: quicksort Memory: 59kB
9	Worker 0: Sort Method: quicksort Memory: 59kB
10	Worker 1: Sort Method: quicksort Memory: 59kB
11	-> Partial HashAggregate (cost=275965.00..275969.49 rows=359 width=47) (actual time=3887.287..3887.504 rows=35
12	Group Key: p.parking_id
13	Batches: 1 Memory Usage: 189kB
14	Worker 0: Batches: 1 Memory Usage: 189kB
15	Worker 1: Batches: 1 Memory Usage: 189kB
16	-> Hash Join (cost=12.08..244715.00 rows=6250000 width=19) (actual time=14.771..2353.112 rows=5000000 loop
17	Hash Cond: (ps.parking_id = p.parking_id)
18	-> Parallel Seq Scan on parking_session ps (cost=0.00..228103.00 rows=6250000 width=8) (actual time=0.107..(
19	-> Hash (cost=7.59..7.59 rows=359 width=15) (actual time=14.607..14.608 rows=359 loops=3)
20	Buckets: 1024 Batches: 1 Memory Usage: 25kB
21	-> Seq Scan on parking p (cost=0.00..7.59 rows=359 width=15) (actual time=14.424..14.508 rows=359 loop
22	Planning Time: 1.070 ms
23	JIT:
24	Functions: 48
25	Options: Inlining false, Optimization false, Expressions true, Deforming true
26	Timing: Generation 4.201 ms (Deform 2.079 ms), Inlining 0.000 ms, Optimization 2.363 ms, Emission 41.068 ms, Total 47.633 ms
27	Execution Time: 4009.724 ms

4. Анализата на извршување (EXPLAIN ANALYZE) покажува дека најбавната операција е Parallel Sequential Scan на табелата **parking_session**, при што се обработуваат голем број записи. Дополнително се извршува Hash Join со табелата **parking**, како и Group Aggregate за групирање по **parking_id**.

5. Со цел оптимизација, се додава индекс на колоната parking_id во табелата parking_session, со очекување да се подобри извршувањето на JOIN операцијата и филтрирањето на податоците.

```
CREATE INDEX idx_parking_spot_status
ON parking_spot(status);
```

6. По додавање на индексот и повторна анализа, се забележува дека времето на извршување останува приближно исто, односно околу 3908 ms, без значајно подобрување. Ова се должи на фактот дека при обработка на голем број редови оптимизаторот се уште користи Parallel Sequential Scan како поефикасна стратегија од користење на индекс.

```
EXPLAIN ANALYZE SELECT * FROM parking_
```

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

	QUERY PLAN
1	Finalize GroupAggregate (cost=276984.75..277078.39 rows=359 width=47) (actual time=3897.147..3907.474 rows=359 loops=1)
2	Group Key: p.parking_id
3	-> Gather Merge (cost=276984.75..277068.52 rows=718 width=47) (actual time=3897.101..3906.838 rows=1077 loops=1)
4	Workers Planned: 2
5	Workers Launched: 2
6	-> Sort (cost=275984.73..275985.62 rows=359 width=47) (actual time=3863.309..3863.337 rows=359 loops=3)
7	Sort Key: p.parking_id
8	Sort Method: quicksort Memory: 59kB
9	Worker 0: Sort Method: quicksort Memory: 59kB
10	Worker 1: Sort Method: quicksort Memory: 59kB
11	-> Partial HashAggregate (cost=275965.00..275969.49 rows=359 width=47) (actual time=3862.872..3863.110 rows=35
12	Group Key: p.parking_id
13	Batches: 1 Memory Usage: 189kB
14	Worker 0: Batches: 1 Memory Usage: 189kB
15	Worker 1: Batches: 1 Memory Usage: 189kB
16	-> Hash Join (cost=12.08..244715.00 rows=6250000 width=19) (actual time=14.379..2331.820 rows=5000000 loops
17	Hash Cond: (ps.parking_id = p.parking_id)
18	-> Parallel Seq Scan on parking_session ps (cost=0.00..228103.00 rows=6250000 width=8) (actual time=0.037..6
19	-> Hash (cost=7.59..7.59 rows=359 width=15) (actual time=14.289..14.290 rows=359 loops=3)
20	Buckets: 1024 Batches: 1 Memory Usage: 25kB
21	-> Seq Scan on parking p (cost=0.00..7.59 rows=359 width=15) (actual time=14.096..14.180 rows=359 loop
22	Planning Time: 0.501 ms
23	JIT:
24	Functions: 48
25	Options: Inlining false, Optimization false, Expressions true, Deforming true
26	Timing: Generation 4.230 ms (Deform 2.136 ms), Inlining 0.000 ms, Optimization 2.184 ms, Emission 40.301 ms, Total 46.715 ms
27	Execution Time: 3908.964 ms

```
INSERT INTO parking_session(user_id, vehicle_id, parking_id, tariff_id, start_time,
session_status_id, total_amount)
values (344655, 2068931, 14, 7,'2025-07-27 18:54:00', 2 ,240.00);
```

1 row(s) updated - 382ms, on 2026-05-26 at 19:42:31

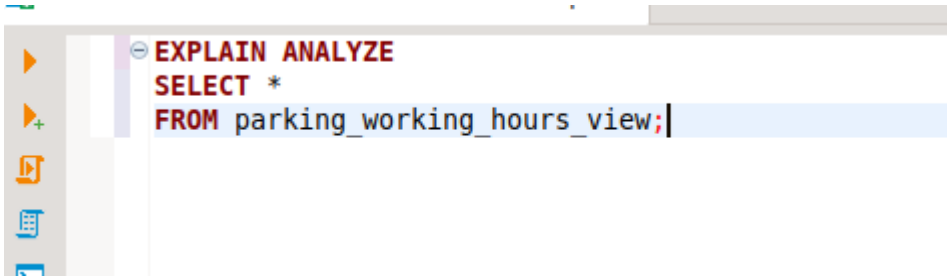
```
UPDATE parking_session SET total_amount = 300.00
WHERE session_id = 2759027;
```

1 row(s) updated - 72ms, on 2026-05-26 at 19:44:42

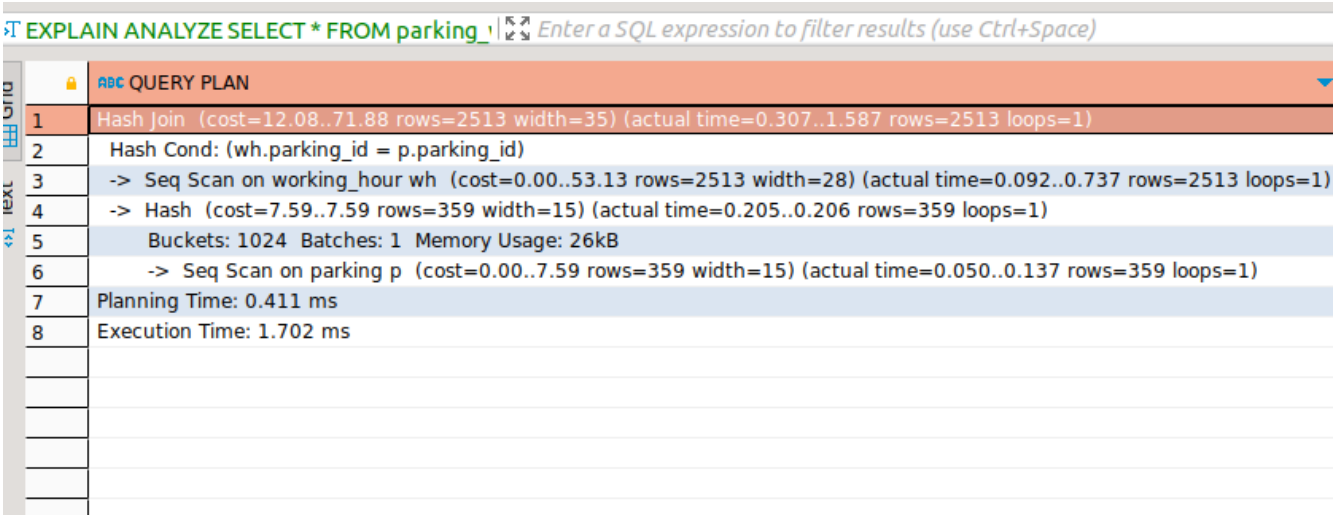
3: View: Working hours per parking

1. Примарен филтер за погледот parking_working_hours_view ќе биде според parking_id, преку кој се поврзуваат табелите parking и working_hour. Дополнително се користат колоните day_of_week, open_time и close_time за прикажување на работните часови.

2. Примарен случај на употреба е прикажување на работното време на паркинзите. Овој поглед се користи за информирање на корисниците и при процесот на резервација, па затоа неговите перформанси се важни, но не критични како кај реално-временските функционалности.



3. Иницијалното време за извршување на погледот изнесува **1.702 ms**, што е прифатливо за моменталната големина на податоците.



4. Анализата на извршување (EXPLAIN ANALYZE) покажува дека главната операција е Seq Scan на табелата **working_hour**, при што се читаат сите редови (2513 записи). Дополнително се извршува Seq Scan на табелата **parking**, како и Hash Join за поврзување на двете табели.

5. Со оглед на тоа што времето на извршување е веќе прифатливо и табелите се со мала големина, не е извршено индексирање. Бидејќи нема значајна потреба од оптимизација, погледот е погоден за користење во апликацијата

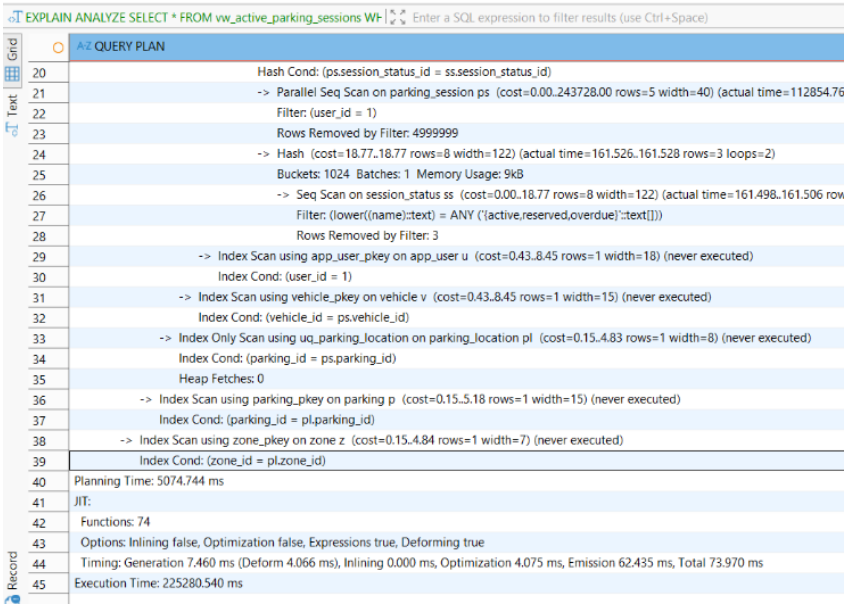
View4: Active parking sessions

1. Примарен филтер за погледот vw_active_parking_sessions е според user_id. Дополнително се користи статусот на сесијата, односно се прикажуваат само сесии со статус active, reserved или overdue.

2. Примарен случај на употреба е прикажување на моментално активната или најновата релевантна паркинг сесија за корисник. Овој поглед е важен за апликацијата бидејќи се користи при преглед на активна резервација/сесија и има директно влијание врз корисничкото искуство.

```
EXPLAIN ANALYZE
SELECT *
FROM vw_active_parking_sessions
WHERE user_id = 1;
```

3. Иницијалното време за извршување на погледот изнесува 225280.540 ms (околу 3 минути и 45 секунди). Ова време не е прифатливо за апликацијата, па затоа се пристапува кон оптимизација.



4. Анализата на извршување (EXPLAIN ANALYZE) покажува дека најбавната операција е Parallel Sequential Scan на табелата parking_session. При пребарување за user_id = 1 се читаат околу 5 милиони редови, при што 4,999,999 редови се отстрануваат со филтер. Дополнително, првичната верзија на погледот користи row_number() over (partition by user_id order by start_time desc), што предизвикува дополнително сортирање.

Почетната верзија на погледот беше:

```
CREATE OR REPLACE VIEW vw_active_parking_sessions AS
SELECT *
FROM (
    SELECT
```

```
        ps.session_id,
        u.user_id,
        u.first_name,
        u.last_name,
        v.vehicle_id,
        v.license_plate,
        p.parking_id,
        p.name AS parking_name,
        z.zone_id,
        z.name AS zone_name,
        ps.start_time,
        ps.end_time,
        ss.name AS session_status,
        ps.total_amount,
        row_number() over (
            PARTITION BY u.user_id
            ORDER BY ps.start_time DESC
        ) AS rn
    FROM parking_session ps
    JOIN app_user u ON u.user_id = ps.user_id
    JOIN vehicle v ON v.vehicle_id = ps.vehicle_id
    JOIN parking p ON p.parking_id = ps.parking_id
    JOIN parking_location pl ON pl.parking_id = p.parking_id
    JOIN zone z ON z.zone_id = pl.zone_id
    JOIN session_status ss ON ss.session_status_id = ps.session_status_id
    WHERE lower(ss.name) IN ('active', 'reserved', 'overdue')
) x
WHERE rn = 1;
```

6. Со цел оптимизација, погледот се преуредува со DISTINCT ON наместо row_number(), со цел да се намали потребата од скапа window операција и да се земе најновата сесија за секој корисник.

```
DROP VIEW vw_active_parking_sessions;

CREATE VIEW vw_active_parking_sessions AS
SELECT DISTINCT ON (u.user_id)
    ps.session_id,
    u.user_id,
    u.first_name,
    u.last_name,
    v.vehicle_id,
    v.license_plate,
    p.parking_id,
    p.name AS parking_name,
    z.zone_id,
    z.name AS zone_name,
    ps.start_time,
    ps.end_time,
    ss.name AS session_status,
    ps.total_amount
FROM parking_session ps
JOIN app_user u ON u.user_id = ps.user_id
```

```
JOIN vehicle v ON v.vehicle_id = ps.vehicle_id
JOIN parking p ON p.parking_id = ps.parking_id
JOIN parking_location pl ON pl.parking_id = p.parking_id
JOIN zone z ON z.zone_id = pl.zone_id
JOIN session_status ss ON ss.session_status_id = ps.session_status_id
WHERE lower(ss.name) IN ('active', 'reserved', 'overdue')
ORDER BY u.user_id, ps.start_time DESC;
```

7. По преуредување на погледот со DISTINCT ON, времето на извршување се намалува на 660.612 ms. Ова претставува значително подобрување во однос на иницијалните 225280.540 ms.

   Во планот сè уште се забележува Parallel Seq Scan на parking_session, но преуредувањето на прашалникот значително го намалува времето на извршување и потребата од дополнителна обработка. Погледот е многу поефикасен и е прифатлив за користење во апликацијата.

Тест прашалникот што се користеше за мерење беше:

```
EXPLAIN ANALYZE
SELECT *
FROM vw_active_parking_sessions
WHERE user_id = 1;
```

View5: User parking history

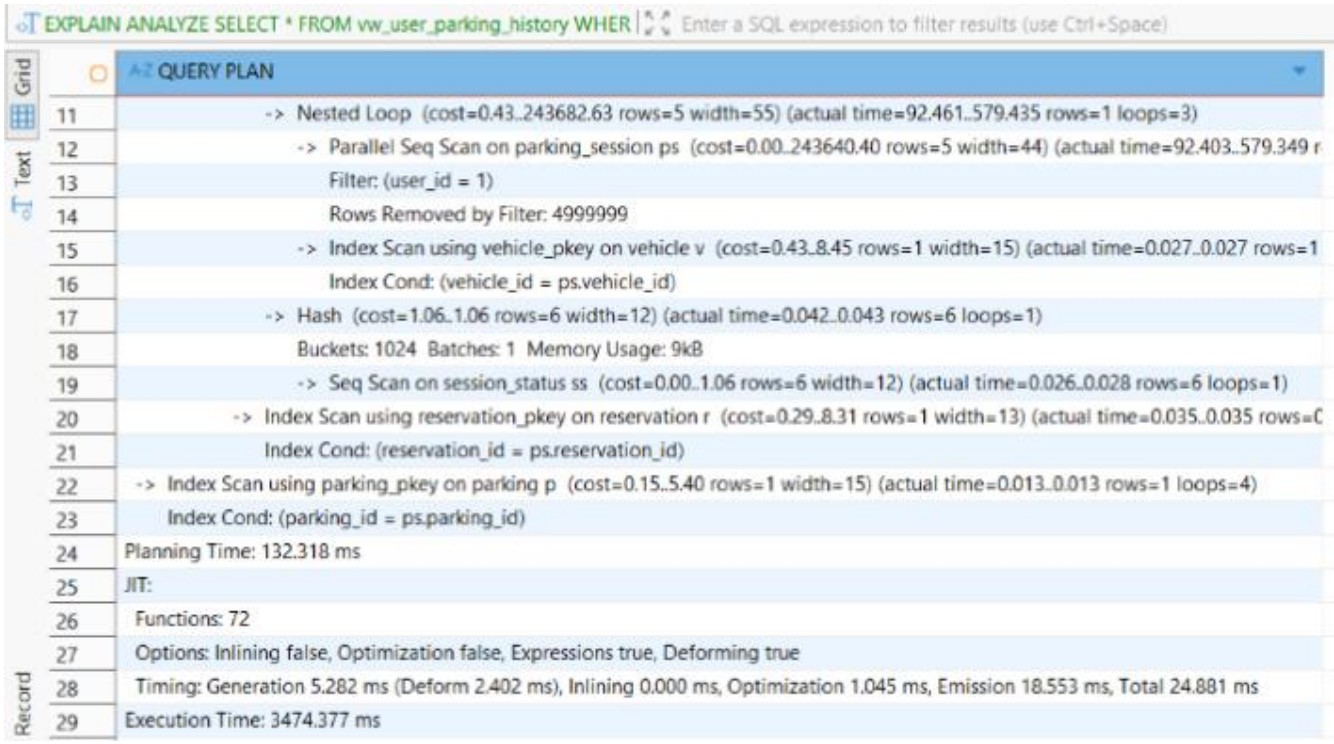
a. Примарен филтер за погледот vw_user_parking_history е според user_id. Погледот ги поврзува табелите app_user, parking_session, vehicle, parking, session_status и reservation, со цел да се добие комплетна историја на паркирање за секој корисник.

b. Примерен случај на употреба е прикажување на сите претходни паркинг сесии за конкретен корисник, вклучувајќи податоци за возилото, паркингот, статусот на сесијата и евентуалната резервација. Овој поглед е корисен за преглед на историја и анализа на користење на системот.

```
EXPLAIN ANALYZE
SELECT *
FROM vw_user_parking_history
WHERE user_id = 1;
```

c. Иницијалното време на извршување на погледот изнесува околу **670 ms**, при пребарување за конкретен user_id.

d. Анализата со EXPLAIN ANALYZE покажува дека најскапата операција е **Parallel Sequential Scan** на табелата parking_session. При пребарување за user_id = 1, системот чита голем број редови (милиони записи), при што најголем дел од нив се отстрануваат со филтерот (Rows Removed by Filter).
Дополнително, се извршуваат повеќе JOIN операции (со vehicle, parking, session_status и reservation), што дополнително го зголемува времето на извршување.



e. Со цел оптимизација, се додава индекс на табелата parking_session според user_id, бидејќи најчесто пребарувањето се врши по овој атрибут:

```
CREATE INDEX idx_parking_session_user_id
ON parking_session(user_id);

ANALYZE parking_session;
```

f. По додавање на индексот, повторно се анализира извршувањето на погледот. Времето на извршување е околу **900–1000 ms**, при што PostgreSQL сè уште користи **Parallel Sequential Scan** наместо Index Scan.

Причината за ова е што:

- бројот на резултати е мал (само неколку редови),
- но табелата е многу голема,
- а JOIN операциите ја прават проценката на planner-от таква што Sequential Scan е поефикасен во овој случај.

Време изминато во извршвување на insert и update по индексирање:

```
INSERT INTO parking_session (user_id, vehicle_id, parking_id, session_status_id,
tariff_id, start_time, end_time, total_amount)
VALUES (1, 10, 5, 1, 1, NOW(), NOW() + interval '2 hours', 0);
```

1 row(s) updated - 16ms, on 2026-05-19 at 20:51:01

```
UPDATE parking_session SET end_time = NOW() + interval '3 hours',
total_amount = 120 WHERE session_id = 1;
```

1 row(s) updated - 15ms, on 2026-05-19 at 20:51:43

Заклучок: Индексот нема значително влијание поради комплексни JOIN операции и мала селективност. Погледот е прифатлив за користење.

View6: User parking history

1. Погледот vw_parking_prices_by_zone служи за прикажување на цените за паркирање по зони, паркинзи и тарифи. Овој поглед ги поврзува табелите parking, city, parking_location, zone, price_list и tariff, со цел да се добие комплетна информација за цената на паркирање во зависност од зоната и времетраењето.

2. Погледот може да се користи за прикажување на цените на паркирање по град и зона, што е корисно за корисниците на апликацијата при избор на паркинг, како и за административни цели и анализа на ценовната политика.

```
EXPLAIN ANALYZE
SELECT *
FROM vw_parking_prices_by_zone;
```

3. Иницијалното време на извршување на погледот изнесува околу **225.970 ms**, што е прифатливо.

4. Анализата покажува дека најчесто се користат **Hash Join** операции помеѓу табелите, како и **Sequential Scan** (Seq Scan) на некои табели како parking, parking_location, zone и tariff. Ова значи дека базата чита цели табели наместо да користи индекси, што може да влијае на перформансите при поголеми количини на податоци.

View7: Reservation details

1. Примарен филтер
Примарен филтер за погледот vw_reservation_details ќе биде според user_id. Дополнително се користат информации за корисник, возило и паркинг.

2. Случај на употреба
Овој поглед се користи за прикажување на детални информации за резервации, вклучувајќи корисник, возило, паркинг, статус и временски интервал. Перформансите се важни бидејќи често се користи во апликацијата.

```
EXPLAIN ANALYZE
SELECT *
FROM vw_reservation_details
WHERE user_id = 4568;
```

3. Иницијално време на извршување
Иницијалното време за извршување на погледот изнесува 6539.461, што е релативно високо.

QUERY PLAN
Nested Loop (cost=0.86..2573.98 rows=1 width=181) (actual time=1997.244..6539.359 rows=1 loops=1)
Join Filter: (p.parking_id = r.parking_id)
Rows Removed by Join Filter: 328
-> Nested Loop (cost=0.86..2561.90 rows=1 width=144) (actual time=1725.370..6267.483 rows=1 loops=1)
-> Nested Loop (cost=0.43..2553.45 rows=1 width=117) (actual time=1382.567..5924.678 rows=1 loops=1)
-> Seq Scan on reservation r (cost=0.00..2545.00 rows=1 width=72) (actual time=652.522..5194.628 rows=1 loops=1)
Filter: (user_id = 4568)
Rows Removed by Filter: 99999
-> Index Scan using app_user_pkey on app_user au (cost=0.43..8.45 rows=1 width=49) (actual time=730.037..730.040 rows=1 loops=1)
Index Cond: (user_id = 4568)
-> Index Scan using vehicle_pkey on vehicle v (cost=0.43..8.45 rows=1 width=31) (actual time=342.792..342.792 rows=1 loops=1)
Index Cond: (vehicle_id = r.vehicle_id)
-> Seq Scan on parking p (cost=0.00..7.59 rows=359 width=41) (actual time=177.166..271.841 rows=329 loops=1)
Planning Time: 7312.257 ms
Execution Time: 6539.461 ms

4. Анализата на извршување (EXPLAIN ANALYZE) покажува дека најбавната операција е Sequential Scan на табелата reservation, при што се обработуваат голем број записи. Дополнително се извршуваат JOIN операции со табелите app_user, vehicle и parking.

5. Оптимизација
Со цел оптимизација, се додава индекс на колоната user_id во табелата reservation, со очекување да се подобри филтрирањето на податоците.

```
CREATE INDEX idx_reservation_user_id
ON reservation(user_id);
```

6. Време по оптимизација
По додавање на индексот и повторна анализа, се забележува значително подобрување на перформансите. Времето на извршување се намалува на 0.201 ms.

Време изминато во извршвување на insert и update по индексирање:

```
INSERT INTO reservation (user_id, vehicle_id, parking_id, reservation_type, reservation_status,
start_time, end_time, created_at, reservation_code)
VALUES (1, 10, 5, 'regular', 'active', NOW(), NOW() + interval '1 hour', NOW(), 'RES12345');
```

1 row(s) updated - 248ms, on 2026-05-19 at 20:35:00

```
UPDATE reservation SET reservation_status = 'cancelled', end_time = NOW(),
reservation_code = 'RES-CANCEL-001' WHERE reservation_id = 100007;
```

0 row(s) updated - 16ms, on 2026-05-19 at 20:37:54

QUERY PLAN
Nested Loop (cost=1.30..33.42 rows=1 width=181) (actual time=0.142..0.146 rows=1 loops=1)
-> Nested Loop (cost=1.15..25.21 rows=1 width=144) (actual time=0.136..0.139 rows=1 loops=1)
-> Nested Loop (cost=0.72..16.77 rows=1 width=117) (actual time=0.097..0.099 rows=1 loops=1)
-> Index Scan using idx_reservation_user_id on reservation r (cost=0.29..8.31 rows=1 width=72) (actual time=0.041..0.041 rows=1 loops=1)
Index Cond: (user_id = 4568)
-> Index Scan using app_user_pkey on app_user au (cost=0.43..8.45 rows=1 width=49) (actual time=0.051..0.052 rows=1 loops=1)
Index Cond: (user_id = 4568)
-> Index Scan using vehicle_pkey on vehicle v (cost=0.43..8.45 rows=1 width=31) (actual time=0.037..0.037 rows=1 loops=1)
Index Cond: (vehicle_id = r.vehicle_id)
-> Index Scan using parking_pkey on parking p (cost=0.15..8.17 rows=1 width=41) (actual time=0.004..0.004 rows=1 loops=1)
Index Cond: (parking_id = r.parking_id)
Planning Time: 0.726 ms
Execution Time: 0.201 ms

7. Заклучок
Со користење на индексот idx_reservation_user_id, оптимизаторот користи Index Scan наместо Sequential Scan, со што значително се намалува времето на извршување на погледот.

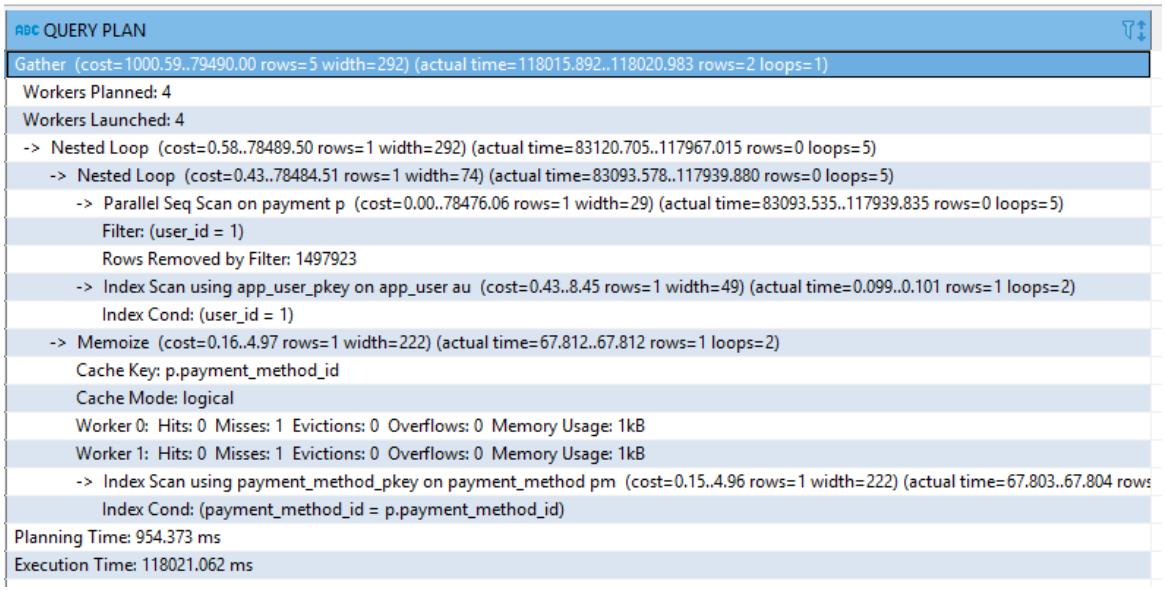
View8: Payment details

1. Примарен филтер
Примарен филтер за погледот vw_payment_details ќе биде според user_id. Дополнително се користат информации за плаќање и метод на плаќање.

2. Случај на употреба
Овој поглед се користи за прикажување на детални информации за плаќања, вклучувајќи корисник, износ, датум на плаќање и начин на плаќање. Перформансите се важни бидејќи погледот може често да се користи за историја на плаќања.

```sql
EXPLAIN ANALYZE
SELECT *
FROM vw_payment_details
WHERE user_id = 1;
```

3. Иницијално време на извршување
Иницијалното време за извршување на погледот изнесува 118021.062 ms, што е релативно високо.



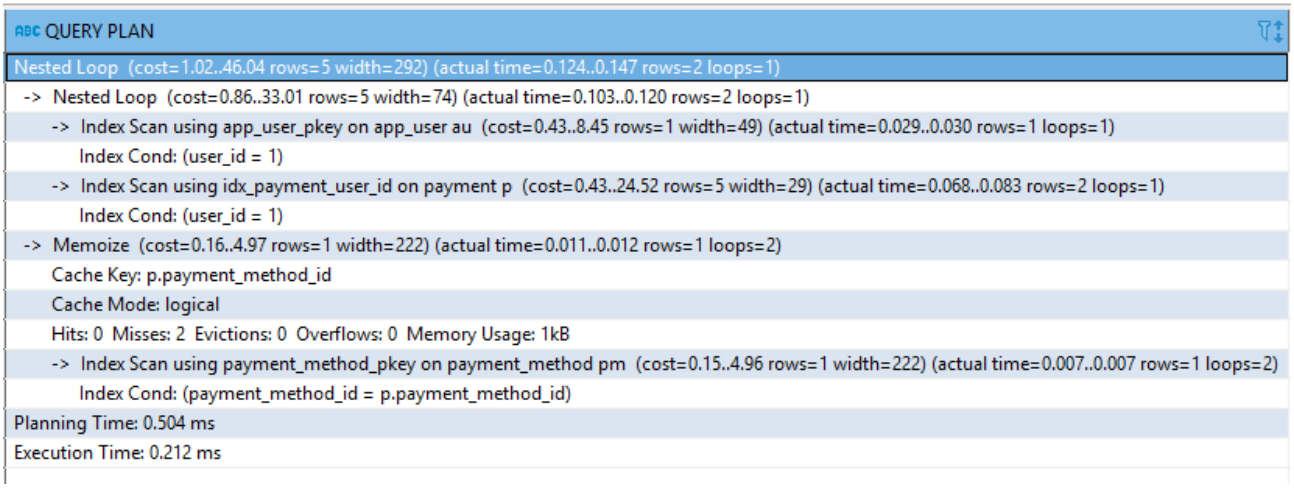
QUERY PLAN
Gather (cost=1000.59..79490.00 rows=5 width=292) (actual time=118015.892..118020.983 rows=2 loops=1)
Workers Planned: 4
Workers Launched: 4
-> Nested Loop (cost=0.58..78489.50 rows=1 width=292) (actual time=83120.705..117967.015 rows=0 loops=5)
-> Nested Loop (cost=0.43..78484.51 rows=1 width=74) (actual time=83093.578..117939.880 rows=0 loops=5)
-> Parallel Seq Scan on payment p (cost=0.00..78476.06 rows=1 width=29) (actual time=83093.535..117939.835 rows=0 loops=5)
Filter: (user_id = 1)
Rows Removed by Filter: 1497923
-> Index Scan using app_user_pkey on app_user au (cost=0.43..8.45 rows=1 width=49) (actual time=0.099..0.101 rows=1 loops=2)
Index Cond: (user_id = 1)
-> Memoize (cost=0.16..4.97 rows=1 width=222) (actual time=67.812..67.812 rows=1 loops=2)
Cache Key: p.payment_method_id
Cache Mode: logical
Worker 0: Hits: 0 Misses: 1 Evictions: 0 Overflows: 0 Memory Usage: 1kB
Worker 1: Hits: 0 Misses: 1 Evictions: 0 Overflows: 0 Memory Usage: 1kB
-> Index Scan using payment_method_pkey on payment_method pm (cost=0.15..4.96 rows=1 width=222) (actual time=67.803..67.804 rows
Index Cond: (payment_method_id = p.payment_method_id)
Planning Time: 954.373 ms
Execution Time: 118021.062 ms

4. Анализа на проблемот
Анализата на извршување (EXPLAIN ANALYZE) покажува дека најбавната операција е Parallel Sequential Scan на табелата payment, при што се обработуваат голем број записи. Дополнително се извршуваат JOIN операции со табелите app_user и payment_method.

5. Оптимизација
Со цел оптимизација, се додава индекс на колоната user_id во табелата payment, со очекување да се подобри филтрирањето на податоците.

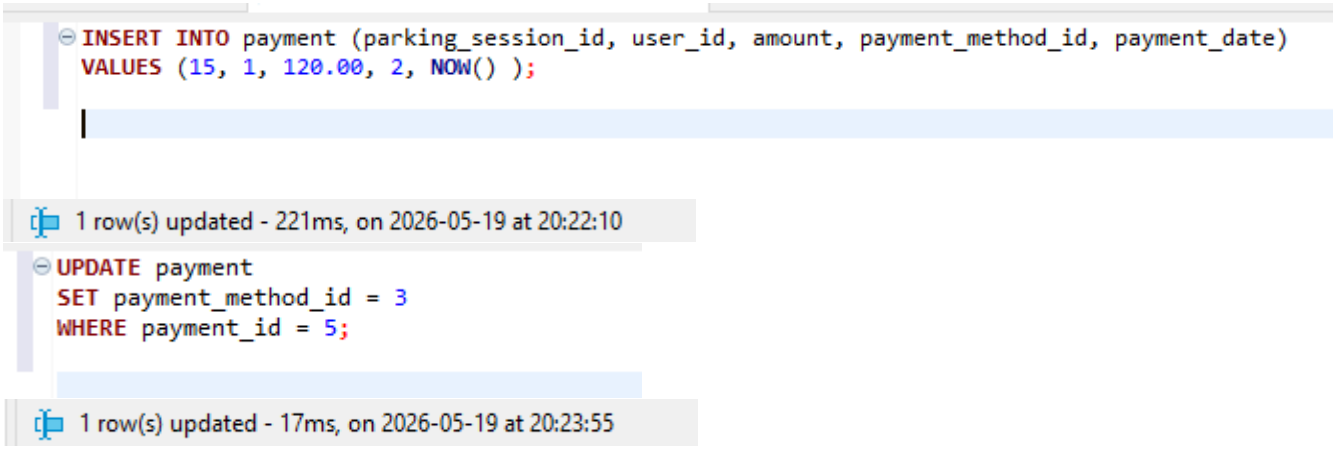
```sql
CREATE INDEX idx_payment_user_id
ON payment(user_id);
```

6. Време по оптимизација
По додавање на индексот и повторна анализа, се забележува значително подобрување на перформансите. Времето на извршување се намалува на 0.212 ms.



QUERY PLAN
Nested Loop (cost=1.02..46.04 rows=5 width=292) (actual time=0.124..0.147 rows=2 loops=1)
-> Nested Loop (cost=0.86..33.01 rows=5 width=74) (actual time=0.103..0.120 rows=2 loops=1)
-> Index Scan using app_user_pkey on app_user au (cost=0.43..8.45 rows=1 width=49) (actual time=0.029..0.030 rows=1 loops=1)
Index Cond: (user_id = 1)
-> Index Scan using idx_payment_user_id on payment p (cost=0.43..24.52 rows=5 width=29) (actual time=0.068..0.083 rows=2 loops=1)
Index Cond: (user_id = 1)
-> Memoize (cost=0.16..4.97 rows=1 width=222) (actual time=0.011..0.012 rows=1 loops=2)
Cache Key: p.payment_method_id
Cache Mode: logical
Hits: 0 Misses: 2 Evictions: 0 Overflows: 0 Memory Usage: 1kB
-> Index Scan using payment_method_pkey on payment_method pm (cost=0.15..4.96 rows=1 width=222) (actual time=0.007..0.007 rows=1 loops=2)
Index Cond: (payment_method_id = p.payment_method_id)
Planning Time: 0.504 ms
Execution Time: 0.212 ms

Време изминато во извршвување на insert и update по индексирање:



```
INSERT INTO payment (parking_session_id, user_id, amount, payment_method_id, payment_date)
VALUES (15, 1, 120.00, 2, NOW() );
```

1 row(s) updated - 221ms, on 2026-05-19 at 20:22:10

```
UPDATE payment
SET payment_method_id = 3
WHERE payment_id = 5;
```

1 row(s) updated - 17ms, on 2026-05-19 at 20:23:55

7. Заклучок
Со користење на индексот idx_payment_user_id, оптимизаторот користи Index Scan наместо Parallel Sequential Scan, со што значително се намалува времето на извршување на погледот.

View9: Penalty details

1. Примарен филтер
Примарен филтер за погледот vw_penalty_details ќе биде според user_id. Дополнително се користат информации за возило, тип на казна и статус на казна.

2. Случај на употреба
Овој поглед се користи за приказ на казни за корисници, вклучувајќи информации за возило, тип на казна, статус и опис.

```
EXPLAIN ANALYZE
SELECT *
FROM vw_penalty_details
WHERE user_id = 1;
```

3. Иницијално време на извршување
Иницијалното време за извршување на погледот изнесува 217359.106 ms, што е релативно високо.

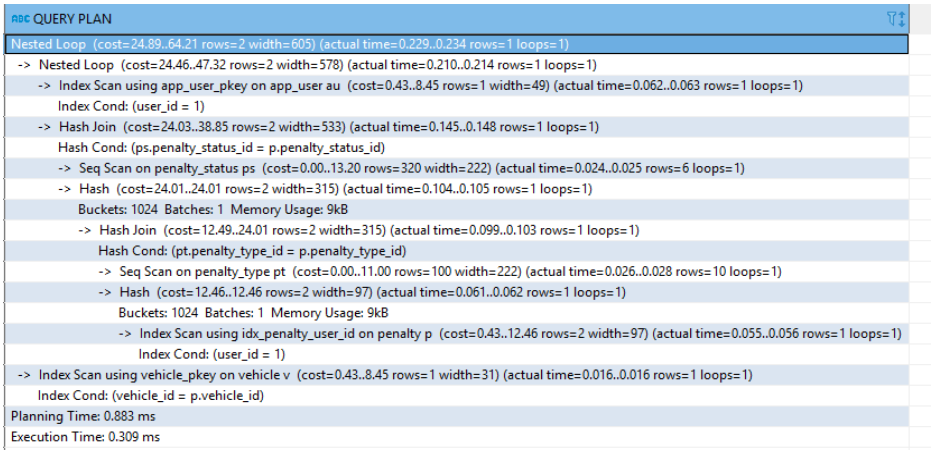
```
QUERY PLAN
Gather  (cost=1001.17..116546.70 rows=2 width=605) (actual time=217312.539..217326.262 rows=1 loops=1)
  Workers Planned: 4
  Workers Launched: 4
  ->  Nested Loop  (cost=1.17..115546.50 rows=1 width=605) (actual time=187879.901..217270.137 rows=0 loops=5)
        ->  Nested Loop  (cost=1.01..115538.32 rows=1 width=387) (actual time=187846.775..217237.010 rows=0 loops=5)
              ->  Nested Loop  (cost=0.86..115530.14 rows=1 width=169) (actual time=187785.606..217175.839 rows=0 loops=5)
                    ->  Nested Loop  (cost=0.43..115521.69 rows=1 width=142) (actual time=187785.576..217175.808 rows=0 loops=5)
                          ->  Parallel Seq Scan on penalty p  (cost=0.00..115513.23 rows=1 width=97) (actual time=187785.516..217175.747 rows=0 loops=5)
                                Filter: (user_id = 1)
                                Rows Removed by Filter: 640207
                          ->  Index Scan using app_user_pkey on app_user au  (cost=0.43..8.45 rows=1 width=49) (actual time=0.242..0.245 rows=1 loops=1)
                                Index Cond: (user_id = 1)
                    ->  Index Scan using vehicle_pkey on vehicle v  (cost=0.43..8.45 rows=1 width=31) (actual time=0.103..0.103 rows=1 loops=1)
                          Index Cond: (vehicle_id = p.vehicle_id)
              ->  Memoize  (cost=0.15..8.17 rows=1 width=222) (actual time=305.831..305.832 rows=1 loops=1)
                    Cache Key: p.penalty_type_id
                    Cache Mode: logical
                    Worker 3:  Hits: 0  Misses: 1  Evictions: 0  Overflows: 0  Memory Usage: 1kB
                    ->  Index Scan using penalty_type_pkey on penalty_type pt  (cost=0.14..8.16 rows=1 width=222) (actual time=305.810..305.811 rows=1 loops=1)
                          Index Cond: (penalty_type_id = p.penalty_type_id)
        ->  Memoize  (cost=0.16..8.18 rows=1 width=222) (actual time=165.616..165.617 rows=1 loops=1)
              Cache Key: p.penalty_status_id
              Cache Mode: logical
              Worker 3:  Hits: 0  Misses: 1  Evictions: 0  Overflows: 0  Memory Usage: 1kB
              ->  Index Scan using penalty_status_pkey on penalty_status ps  (cost=0.15..8.17 rows=1 width=222) (actual time=165.598..165.598 rows=1 loops=1)
                    Index Cond: (penalty_status_id = p.penalty_status_id)
Planning Time: 1305.833 ms
JIT:
  Functions: 160
  Options: Inlining false, Optimization false, Expressions true, Deforming true
  Timing: Generation 11.107 ms (Deform 5.021 ms), Inlining 0.000 ms, Optimization 5.738 ms, Emission 100.923 ms, Total 117.767 ms
Execution Time: 217359.106 ms
```

4. Анализа на проблемот
Анализата на извршување (EXPLAIN ANALYZE) покажува дека најбавната операција е Parallel Sequential Scan на табелата penalty, при што се обработуваат голем број записи. Дополнително се извршуваат JOIN операции со табелите app_user, vehicle, penalty_type и penalty_status.

5. Оптимизација
Со цел оптимизација, се додава индекс на колоната user_id во табелата penalty, со очекување да се подобри филтрирањето на податоците.

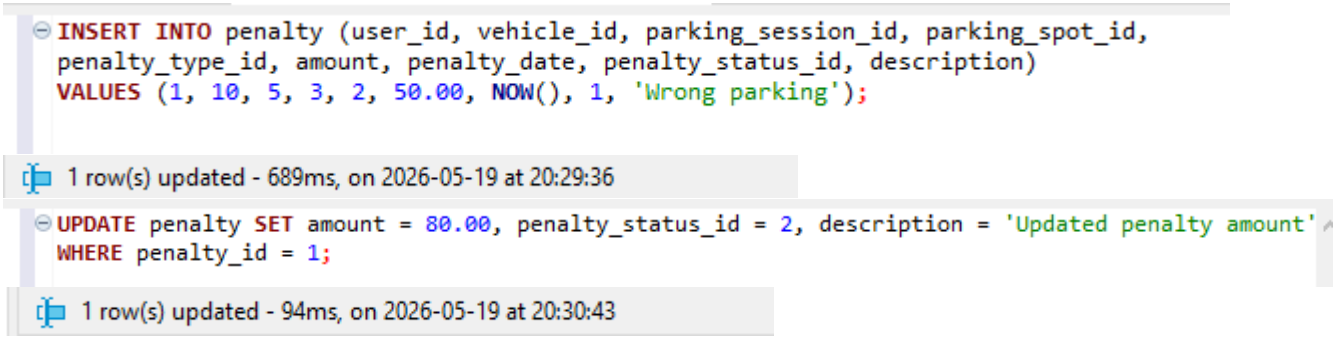
```
CREATE INDEX idx_penalty_user_id
ON penalty(user_id);
```

6. Време по оптимизација
По додавање на индексот и повторна анализа, се забележува значително подобрување на перформансите. Времето на извршување се намалува на 0.309 ms.



```
QUERY PLAN
Nested Loop  (cost=24.89..64.21 rows=2 width=605) (actual time=0.229..0.234 rows=1 loops=1)
  ->  Nested Loop  (cost=24.46..47.32 rows=2 width=578) (actual time=0.210..0.214 rows=1 loops=1)
    ->  Index Scan using app_user_pkey on app_user au  (cost=0.43..8.45 rows=1 width=49) (actual time=0.062..0.063 rows=1 loops=1)
      Index Cond: (user_id = 1)
    ->  Hash Join  (cost=24.03..38.85 rows=2 width=533) (actual time=0.145..0.148 rows=1 loops=1)
      Hash Cond: (ps.penalty_status_id = p.penalty_status_id)
      ->  Seq Scan on penalty_status ps  (cost=0.00..13.20 rows=320 width=222) (actual time=0.024..0.025 rows=6 loops=1)
      ->  Hash  (cost=24.01..24.01 rows=2 width=315) (actual time=0.104..0.105 rows=1 loops=1)
        Buckets: 1024  Batches: 1  Memory Usage: 9kB
        ->  Hash Join  (cost=12.49..24.01 rows=2 width=315) (actual time=0.099..0.103 rows=1 loops=1)
          Hash Cond: (pt.penalty_type_id = p.penalty_type_id)
          ->  Seq Scan on penalty_type pt  (cost=0.00..11.00 rows=100 width=222) (actual time=0.026..0.028 rows=10 loops=1)
          ->  Hash  (cost=12.46..12.46 rows=2 width=97) (actual time=0.061..0.062 rows=1 loops=1)
            Buckets: 1024  Batches: 1  Memory Usage: 9kB
            ->  Index Scan using idx_penalty_user_id on penalty p  (cost=0.43..12.46 rows=2 width=97) (actual time=0.055..0.056 rows=1 loops=1)
              Index Cond: (user_id = 1)
  ->  Index Scan using vehicle_pkey on vehicle v  (cost=0.43..8.45 rows=1 width=31) (actual time=0.016..0.016 rows=1 loops=1)
    Index Cond: (vehicle_id = p.vehicle_id)
Planning Time: 0.883 ms
Execution Time: 0.309 ms
```

Време изминато во извршвување на insert и update по индексирање:



```
INSERT INTO penalty (user_id, vehicle_id, parking_session_id, parking_spot_id,
penalty_type_id, amount, penalty_date, penalty_status_id, description)
VALUES (1, 10, 5, 3, 2, 50.00, NOW(), 1, 'Wrong parking');

1 row(s) updated - 689ms, on 2026-05-19 at 20:29:36

UPDATE penalty SET amount = 80.00, penalty_status_id = 2, description = 'Updated penalty amount'
WHERE penalty_id = 1;

1 row(s) updated - 94ms, on 2026-05-19 at 20:30:43
```

7. Заклучок

Со користење на индексот idx_penalty_user_id, оптимизаторот користи Index Scan наместо Parallel Sequential Scan, со што значително се намалува времето на извршување на погледот.