Напредни Бази на Податоци

ФАЗА 4 - Индекси и оптимизација на погледи

Проект: Eventflow

View1 - Event sales summary

Овој view ке биде користен од страна на организаторот на настанот или од администраторот.
Целта на овој поглед е да се даде сумаризација на продажбата на билети за даден настан.

Првичниот изглед на овој поглед беше:

```
CREATE OR REPLACE VIEW public.v_event_sales_summary AS
SELECT
    e.event_id,
    e.title AS event_title,
    o.organiser_id,
    o.company_name AS organiser_name,
    COUNT(t.ticket_id) AS tickets_sold,
    COUNT(DISTINCT oc.order_id) AS total_orders,
    COALESCE(SUM(p.amount_paid), 0) AS total_revenue,
    ROUND(AVG(p.amount_paid), 2) AS average_payment_amount,
    MIN(oc.created_at) AS first_order_at,
    MAX(oc.created_at) AS last_order_at
FROM public.event e
        JOIN public.organiser o
             1..n<->1: ON o.organiser_id = e.organiser_id
        JOIN public.ticket_type tt
             1<->1..n: ON tt.event_id = e.event_id
        JOIN public.ticket t
             1<->1..n: ON t.ticket_type_id = tt.ticket_type_id
        JOIN public.order_cart oc
             1..n<->1: ON oc.order_id = t.order_id
        LEFT JOIN public.payment p
                  1<->0..n: ON p.order_id = oc.order_id
GROUP BY
    e.event_id,
    e.title,
    o.organiser_id,
    o.company_name;
```

Новиот излгед на овој поглед оптимизирано:

```
CREATE OR REPLACE VIEW public.v_event_sales_summary_2 AS
    e.event_id,
    e.title AS event_title,
    o.organiser_id,
    o.company_name AS organiser_name,

    COALESCE(s.tickets_sold, 0) AS tickets_sold,
    COALESCE(s.total_orders, 0) AS total_orders,
    COALESCE(s.total_revenue, 0) AS total_revenue,
    s.average_payment_amount,
    s.first_order_at,
    s.last_order_at

FROM public.event e
        JOIN public.organiser o
             1..n<->1: ON o.organiser_id = e.organiser_id

        LEFT JOIN LATERAL (
    SELECT
        SUM(x.ticket_count) AS tickets_sold,
        COUNT(*) AS total_orders,
        COALESCE(SUM(p.amount_paid), 0) AS total_revenue,
        ROUND(AVG(p.amount_paid), 2) AS average_payment_amount,
        MIN(oc.created_at) AS first_order_at,
        MAX(oc.created_at) AS last_order_at
    FROM (
            SELECT
                t.order_id,
                COUNT(*) AS ticket_count
            FROM public.ticket_type tt
                    JOIN public.ticket t
                         1<->1..n: ON t.ticket_type_id = tt.ticket_type_id
            WHERE tt.event_id = e.event_id
            GROUP BY t.order_id
        ) x
            JOIN public.order_cart oc
                ON oc.order_id = x.order_id
            LEFT JOIN public.payment p
                     ON p.order_id = x.order_id
```

Овој поглед беше оптимизиран со помош на рана агрегација. Сите редови кои ни се потребни се филтрирани и земени во подпрашалникот пред воопшто да учествуваат во самиот join. Ова премногу ни помага во однос на меморија бидејќи место за време на join да се работи со милионски редици се работи со илјадници.

Извршување на прашалникот и потребно време:

Operation	Params	Rows	Actual Rows	Total Cost	Actual Total Time	Startup Cost	Actual Startup Time
Select							
Nested Loops (Nested Loop)		1	1	532968.28	1830.128	532952.23	1830.125
Nested Loops (Nested Loop)		1	1	16.74	0.02	0.71	0.018
Index Scan	table: event; index: event_pkey;	1	1	8.44	0.012	0.42	0.01
Index Scan	table: organiser; index: organiser_p...	1	1	8.3	0.005	0.29	0.005
Aggregate		1	1	532951.53	1830.107	532951.52	1830.105
Nested Loops (Nested Loop)		23	24	532951.22	1830.088	303414.24	1118.359
Hash Join		23	24	532756.82	1830.019	303413.8	1118.344
Full Scan (Seq Scan)	table: payment;	100000...	10000000	203093	568.614	0	0.026
Transformation (Hash)		23	24	303413.52	1028.644	303413.52	1028.643
Access (Subquery Sca		23	24	303413.52	1028.64	303412.88	1028.636
Aggregate		23	24	303413.29	1028.638	303412.88	1028.635
Sort		23	24	303412.94	1028.632	303412.88	1028.631
Hash Join		23	24	303412.36	1028.619	27161.25	204.503
Full Scan (	table: ticket;	100000...	10000002	250001.02	700.728	0	2.161
Transformatic		3	3	27161.21	77.688	27161.21	77.687
Full Sca	table: ticket_type;	3	3	27161.21	77.679	0	1.297
Index Scan	table: order_cart; index: Order_pkey;	1	1	8.45	0.002	0.43	0.002

Property	Value
Operation	select
Planning	{"Shared Hit...
Planning Ti...	0.603
Triggers	[]
Execution ...	1830.208

Времето потребно за изршување на овој прашалник е 2.5 секунди (пишува 1.8 секунди бидејки е повеќе пати стартувано) што е секако неприфатливо.
Како што може да забележиме од планот на извршување најмногу време одземаат и најмногу редови се поминати во табелите payment и ticket и тоа е логично бидејки тоа ни се табели со 10 милиони редови.

За да го оптимизираме погледот правиме индекси на тие две табели.

Време потребно за insert во табела ticket пред индекс.

Време потребно за insert во табела payment пред индекс 19мс што е исто така одлично.

Создавањето на индексите:

```
CREATE INDEX idx_ticket_type_id ON public.ticket(ticket_type_id);
DROP INDEX idx_ticket_type_id;

CREATE INDEX idx_order_id ON payment(order_id) INCLUDE (amount_paid);
DROP INDEX idx_order_id;
```

Индексот на табелата ticket се прави на атрибутот ticket_type_id бидејки преку тој атрибут учествува во самиот join.
Додека пак на табелата payment се создава covering индекс каде покрај тоа што се става индекс на атрибутот order_id со кој учествува во join се прави INCLUDE на amount_paid. Со ова вредностите на amount_paid за секој order_id се стават во листовите на индексот што овозможува во овој случај на табелата да се прави INDEX ONLY SCAN, што е најефикасниот начин да се помине табела.
Индексите сами по себе помагаат но заедно двата во комбинација работат најдобро.

Еве го новиот план на извршување по создавањето на индексите:

Operation	Params	Rows	Actual Rows	Total Cost	Actual Total Time	Startup Cost	Actual Startup Time
Select							
Nested Loops (Nested Loop)		1	1	27590.26	58.136	27574.2	58.134
Nested Loops (Nested Loo		1	1	16.74	0.048	0.71	0.047
Index Scan	table: event; index: event_pkey;	1	1	8.44	0.02	0.42	0.02
Index Scan	table: organiser; index: organiser_pkey;	1	1	8.3	0.003	0.29	0.003
Aggregate		1	1	27573.5	58.085	27573.49	58.084
Nested Loops (Nested L		23	24	27573.2	58.074	27272.39	57.999
Nested Loops (Neste		23	24	27466.56	58.036	27271.95	57.992
Aggregate		23	24	27271.92	57.972	27271.52	57.967
Sort		23	24	27271.58	57.962	27271.52	57.961
Nested Loo		23	24	27271	57.945	0.43	0.246
Full Scan	table: ticket_type;	3	3	27161.21	57.904	0	0.224
Index Sc	table: ticket; index: idx_ticket_type_id;	8	8	36.52	0.012	0.43	0.007
Index Scan	table: order_cart; index: Order_pkey;	1	1	8.45	0.002	0.43	0.002
Index Scan (Index Or	table: payment; index: idx_order_id;	1	1	4.63	0.001	0.43	0.001

Property	Value
Operation	select
Planning	{"Shared Hit...
Planning Ti...	1.003
Triggers	[]
Execution ...	58.219

Како што може да видиме има значително подобрување во времето на извршување на погледот. Времето паѓа до 58мс. Овие индекси исто така понатаму ни помагаат во уште неколку погледи што е многу корисно за нас.

Времето за insert во табелата ticket после индекс:

Operation	Params	Rows	Actual Rows	Total Cost	Actual Total Time	Startup Cost	Actual Startup Time
Insert							
Value (Result)		1	1	0.01	0.003	0	0.003

Property	Value
Operation	insert
Planning	{"Shared Hit...
Planning Ti...	0.05
Triggers	[{"Trigger N...
Execution ...	1.075

Времето за insert во табелата payment после индекс:

Operation	Params	Rows	Actual Rows	Total Cost	Actual Total Time	Startup Cost	Actual Startup Time
Insert							
Value (Result)		1	1	0.01	0.004	0	0.004

Property	Value
Operation	insert
Planning	{"Shared Hit...
Planning Ti...	0.134
Triggers	[{"Trigger N...
Execution ...	0.441

Самиот изглед на прашалникот е:

```
select *
from postgres.public.v_event_sales_summary_2
where event_id=471582;
```

View2 - User purchase profile

Овој поглед може да биде користен од страна на корисникот да си провери сопствена статистика за неговите купувуања а исто така би можело да биде користено од страна на admin за CRM и за recommendations за корисникот.

Првичниот излгед на погледот:

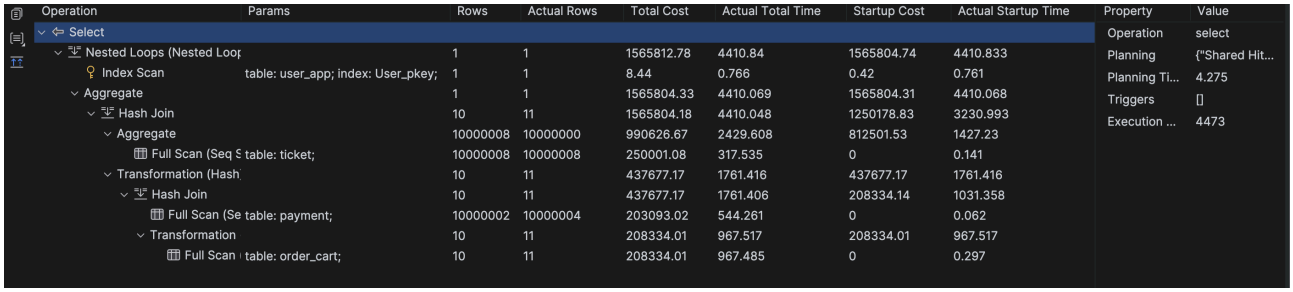
```
CREATE OR REPLACE VIEW public.v_user_purchase_profile AS
SELECT
    u.user_id,
    u.username,
    u.email,
    COUNT(DISTINCT oc.order_id) AS total_orders,
    COUNT(t.ticket_id) AS total_tickets,
    COALESCE(SUM(p.amount_paid), 0) AS total_spent,
    ROUND(AVG(p.amount_paid), 2) AS average_order_payment,
    MIN(oc.created_at) AS first_purchase_at,
    MAX(oc.created_at) AS last_purchase_at
FROM public.user_app u
        LEFT JOIN public.order_cart oc
                  1<->0..n: ON oc.user_id = u.user_id
        LEFT JOIN public.ticket t
                  1<->0..n: ON t.order_id = oc.order_id
        LEFT JOIN public.payment p
                  1<->0..n: ON p.order_id = oc.order_id
GROUP BY
    u.user_id,
    u.username,
    u.email;
```

Новиот оптимизиран изглед на погледот:

```
CREATE OR REPLACE VIEW public.v_user_purchase_profile_2 AS
SELECT
    u.user_id,
    u.username,
    u.email,

    COALESCE(s.total_orders, 0) AS total_orders,
    COALESCE(s.total_tickets, 0) AS total_tickets,
    COALESCE(s.total_spent, 0) AS total_spent,
    s.average_order_payment,
    s.first_purchase_at,
    s.last_purchase_at

FROM public.user_app u
        LEFT JOIN LATERAL (
    SELECT
        COUNT(*) AS total_orders,
        COALESCE(SUM(x.ticket_count), 0) AS total_tickets,
        COALESCE(SUM(p.amount_paid), 0) AS total_spent,
        ROUND(AVG(p.amount_paid), 2) AS average_order_payment,
        MIN(oc.created_at) AS first_purchase_at,
        MAX(oc.created_at) AS last_purchase_at
    FROM public.order_cart oc
            LEFT JOIN (
        SELECT
            t.order_id,
            COUNT(*) AS ticket_count
        FROM public.ticket t
        GROUP BY t.order_id
    ) x
                    ON x.order_id = oc.order_id
            LEFT JOIN public.payment p
                      1<->0..n: ON p.order_id = oc.order_id
    WHERE oc.user_id = u.user_id
    ) s ON true;
```

Процесот на оптимизација е исто како минатиот т.е раната агрегација и филтрирањето на редовите пред изведувањето на самиот join.

Планот за извршување и времето на извршување на овој поглед:



Operation	Params	Rows	Actual Rows	Total Cost	Actual Total Time	Startup Cost	Actual Startup Time
Select							
Nested Loops (Nested Loop		1	1	1565812.78	4410.84	1565804.74	4410.833
Index Scan	table: user_app; index: User_pkey;	1	1	8.44	0.766	0.42	0.761
Aggregate		1	1	1565804.33	4410.069	1565804.31	4410.068
Hash Join		10	11	1565804.18	4410.048	1250178.83	3230.993
Aggregate		10000008	10000000	990626.67	2429.608	812501.53	1427.23
Full Scan (Seq S	table: ticket;	10000008	10000008	250001.08	317.535	0	0.141
Transformation (Hash		10	11	437677.17	1761.416	437677.17	1761.416
Hash Join		10	11	437677.17	1761.406	208334.14	1031.358
Full Scan (Se	table: payment;	10000002	10000004	203093.02	544.261	0	0.062
Transformation		10	11	208334.01	967.517	208334.01	967.517
Full Scan	table: order_cart;	10	11	208334.01	967.485	0	0.297

Property	Value
Operation	select
Planning	{"Shared Hit...
Planning Ti...	4.275
Triggers	[]
Execution ...	4473

Како што гледаме времето на изведување е 4 секунди што никако не е прифатливо и мора да се оптмизира. Се прави full table scan на 3 табели и тоа payment, order_cart и ticket. Овие табели сите имаат по 10 милиони редови и затоа всушност изведувањето на прашалникот трае толку долго.

Среќа можеме да го користиме индексот на табелата payment што го употребивме во минатиот поглед но тој сам по себе не е доволен. Мора исто така да поставиме индекс на табелата order_cart на атрибутот user_id и на табелата ticket на атрибутот order_id. Покрај тоа што веќе имаме индекс на табелата ticket мора да направиме уште еден. Гледајки дека има индекси само на 2 колони од 10 кои ги има табелата ticket ова нема да ни прави проблем при операциите Insert, Update, Delete.

Создавањето на индексите:

```
CREATE INDEX idx_order_id ON payment(order_id) INCLUDE (amount_paid);
DROP INDEX idx_order_id;

CREATE INDEX idx_user_id ON order_cart(user_id);
DROP INDEX idx_user_id;

CREATE INDEX idx_ticket_order_id ON ticket(order_id);
DROP INDEX idx_ticket_order_id;
```

Планот на извршување и времето на извршување на погледот после индексите:

Operation	Params	Rows	Actual Rows	Total Cost	Actual Total Time	Startup Cost	Actual Startup Time
Select							
Nested Loops (Nested Lo		1	1	564795.54	1146.164	564787.5	1146.163
Index Scan	table: user_app; index: User_pkey;	1	1	8.44	0.021	0.42	0.02
Aggregate		1	1	564787.09	1146.141	564787.07	1146.14
Nested Loops (Nes		10	11	564786.94	1146.086	45.26	1.316
Merge Join		10	11	564738.32	1139.298	44.83	1.301
Sort		10	11	44.42	1.179	44.39	1.17
Index Sca	table: order_cart; index: idx_user_id;	10	11	44.23	1.161	0.43	0.266
Aggregate		10000008	9000101	439693.68	960.482	0.43	0.089
Index Sca	table: ticket; index: idx_ticket_order_id;	10000008	9000110	289693.55	332.829	0.43	0.079
Index Scan (Ind	table: payment; index: idx_order_id;	1	1	4.85	0.615	0.43	0.614

Property	Value
Operation	select
Planning	{"Shared Hit...
Planning Ti...	6.775
Triggers	[]
Execution ...	1146.239

Времето се намалува од 4 секунди на 1.1 секунда што повторно не е толку кратко време но сепак е прифатливо време особено во споредба со 4ирите секунди претходно.

Insert во order_cart пред ставањето на индексот:

```
                                    total_price
                                )
                                VALUES (
                                        125434,
                                        NULL,
                                        3,
                                        NOW(),
                                        2499.99
                                    )
[2026-05-07 23:16:32] 1 row affected in 4 ms
```

По ставањето:

```
                                    total_price
                                )
                                VALUES (
                                        125433,
                                        NULL,
                                        3,
                                        NOW(),
                                        2499.99
                                    )
[2026-05-07 23:17:48] 1 row affected in 3 ms
```

Insert во ticket пред ставањето на индексот:

```
[2026-05-07 23:19:38] nbp.public> insert into public.ticket (order_id, ticket_type_id, lock_expires_at, status, barcode_hash, seat_id, is_sca
                          VALUES (12345, 23, '2021-01-01 00:00:00', 'SOLD', '1234567890', 12345, true, '2021-01-01 00:00:00', true)
[2026-05-07 23:19:38] 1 row affected in 14 ms
```

Insert во ticket по ставањето на индексот:

```
[2026-05-07 23:20:13] nbp.public> insert into public.ticket (order_id, ticket_type_id, lock_expires_at, status, barcode_hash, seat_id, is_sca
                          VALUES (12346, 3, '2021-01-01 00:00:00', 'SOLD', '1234567820', 12345, true, '2021-01-01 00:00:00', true)
[2026-05-07 23:20:13] 1 row affected in 5 ms
```

View3 - Ticket scan statistics per event
Овој поглед би се користел од страна на организаторот на настан да се проверат статистика за билетите при влез(check-in) на настанот. Исто така може да се користи од страна на администраторот на страната.

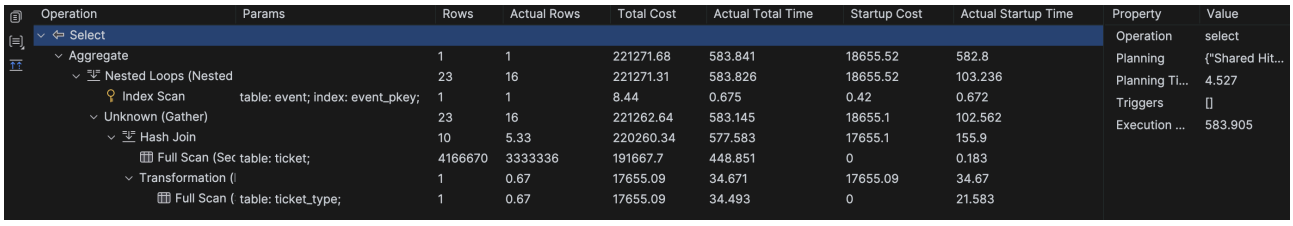
Овој поглед немаште потреба за оптимизација во однос на редолседот на извршување и извршувањето на агрегатните функции.
Излгедот на овој поглед:

```
CREATE OR REPLACE VIEW public.v_event_checkin_statistics_2 AS
SELECT
    e.event_id,
    e.title AS event_title,
    COUNT(t.ticket_id) AS total_tickets,
    COUNT(t.ticket_id) FILTER (WHERE t.is_scanned = true) AS scanned_tickets,
    COUNT(t.ticket_id) FILTER (WHERE t.is_scanned = false) AS unscanned_tickets,
    ROUND(
            100.0 * COUNT(*) FILTER (WHERE t.is_scanned = true)
                / NULLIF(COUNT(t.ticket_id), 0),
            2
    ) AS scanned_percentage,
    MIN(t.scanned_at) FILTER (WHERE t.is_scanned = true) AS first_scan_at,
    MAX(t.scanned_at) FILTER (WHERE t.is_scanned = true) AS last_scan_at
FROM public.event e
        LEFT JOIN public.ticket_type tt
                  1<->0..n: ON tt.event_id = e.event_id
        LEFT JOIN public.ticket t
                  1<->0..n: ON t.ticket_type_id = tt.ticket_type_id
GROUP BY
    e.event_id,
    e.title;
```

Прашалникот во однос на овој поглед:

```
select *
from v_event_checkin_statistics_2
where event_id=471447;
```

Планот за ивршување и времето на извршување на овој поглед без индекс:

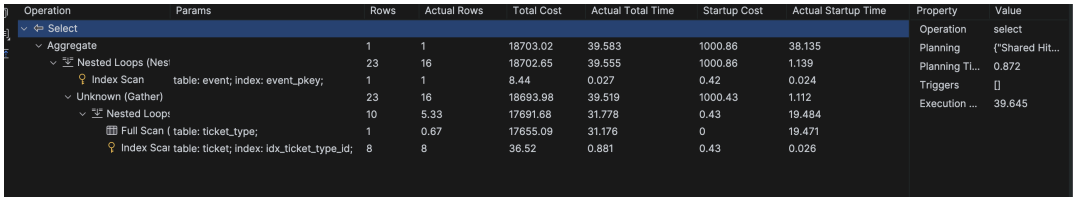


Operation	Params	Rows	Actual Rows	Total Cost	Actual Total Time	Startup Cost	Actual Startup Time
Select							
Aggregate		1	1	221271.68	583.841	18655.52	582.8
Nested Loops (Nested		23	16	221271.31	583.826	18655.52	103.236
Index Scan	table: event; index: event_pkey;	1	1	8.44	0.675	0.42	0.672
Unknown (Gather)		23	16	221262.64	583.145	18655.1	102.562
Hash Join		10	5.33	220260.34	577.583	17655.1	155.9
Full Scan (Sec	table: ticket;	4166670	3333336	191667.7	448.851	0	0.183
Transformation (		1	0.67	17655.09	34.671	17655.09	34.67
Full Scan (	table: ticket_type;	1	0.67	17655.09	34.493	0	21.583

Property	Value
Operation	select
Planning	{"Shared Hit...
Planning Ti...	4.527
Triggers	[]
Execution ...	583.905

Времето на извршување на овој поглед е 500мс што е сосема прифатливо време и нема потреба од оптимизација понатаму.
Покрај тоа што нема потреба од оптимизација ке се преупотреби индексот на табелата ticket на атрибутот ticket_type_id. Овој индекс не е наменет за овој поглед но сепак се употребува.

План за извршување со индексот на табелата ticket на атрибутот ticket_type_id.



Operation	Params	Rows	Actual Rows	Total Cost	Actual Total Time	Startup Cost	Actual Startup Time
Select							
Aggregate		1	1	18703.02	39.583	1000.86	38.135
Nested Loops (Nes		23	16	18702.65	39.555	1000.86	1.139
Index Scan	table: event; index: event_pkey;	1	1	8.44	0.027	0.42	0.024
Unknown (Gather)		23	16	18693.98	39.519	1000.43	1.112
Nested Loop:		10	5.33	17691.68	31.778	0.43	19.484
Full Scan (	table: ticket_type;	1	0.67	17655.09	31.176	0	19.471
Index Scai	table: ticket; index: idx_ticket_type_id;	8	8	36.52	0.881	0.43	0.026

Property	Value
Operation	select
Planning	{"Shared Hit...
Planning Ti...	0.872
Triggers	[]
Execution ...	39.645

Како што гледаме времето е многу подобрено но сепак ова чисто бидејки индексот е создаден за погледи каде што е многу повеќе потребен.

Времето за insert и update на табелата ticket се истите од претходно бидејки е истиот индекс од претходно.

View4 - Session occupancy and waitlist pressure
Овој поглед се користи од страна на корисниците со цел да се провери за даден настан и за дадена сесија од тој настан да се провери статистика во однос на тоа колку е исполнета.

Овој поглед исто така немаше потреба од оптимизација во однос на редоследот и агрегатните функции:

```
CREATE OR REPLACE VIEW public.v_session_demand_overview_2 AS
SELECT
    ess.schedule_id,
    ess.session_title,
    e.event_id,
    e.title AS event_title,
    l.name AS location_name,
    s.section_name,
    s.capacity AS section_capacity,
    COUNT(w.waitlist_id) AS waitlist_entries,
    COUNT(w.waitlist_id) FILTER (WHERE w.status = 'ACTIVE') AS active_waitlist_entries,
    ROUND(
            COUNT(DISTINCT w.waitlist_id)::numeric
                / NULLIF(s.capacity, 0),
            4
    ) AS waitlist_to_capacity_ratio
FROM public.event_schedule_session ess
        JOIN public.event e
             1..n<->1: ON e.event_id = ess.event_id
        JOIN public.section s
             1..n<->1: ON s.section_id = ess.section_id
        JOIN public.location l
             1..n<->1: ON l.location_id = s.location_id
        LEFT JOIN public.waitlist_entry w
                  1<->0..n: ON w.event_schedule_session_id = ess.schedule_id
GROUP BY
    ess.schedule_id,
    ess.session_title,
    e.event_id,
    e.title,
    l.name,
    s.section_name,
    s.capacity;
```

Прашалникот во однос на овој поглед:

```
select *
from v_session_demand_overview_2
where event_id=700002 AND session_title='Opening Session';
```

Планот за извршување на погледот:

Operation	Params	Rows	Actual Rows	Total Cost	Actual Total Time	Startup Cost	Actual Startup Time
Select							
Aggregate		1	1	29456.65	524.467	29456.6	524.13
Sort		1	1	29456.61	524.451	29456.6	524.115
Nested Loops (Nested Loo		1	1	29456.59	524.436	15038.01	524.088
Nested Loops (Nested l		1	1	29456.29	267.199	15037.73	266.852
Nested Loops (Neste		1	1	29447.98	267.098	15037.44	266.753
Unknown (Gather)		1	1	29439.53	266.981	15037.01	266.642
Hash Join		1	0	28439.43	248.356	14037.01	248.345
Full Scan (S	table: waitlist_entry;	416667	333333	13308.67	89.236	0	0.102
Transformation		1	0	14037	81.619	14037	81.618
Full Scan	table: event_schedule_session;	1	0	14037	77	0	46.485
Index Scan	table: event; index: event_pkey;	1	1	8.44	0.108	0.42	0.105
Index Scan	table: section; index: section_pkey;	1	1	8.31	0.095	0.29	0.095
Index Scan	table: location; index: location_pkey;	1	1	0.3	257.229	0.28	257.228

Property	Value
Operation	select
Planning Ti...	1.726
Triggers	[]
Execution ...	524.592

Како што може да видиме табелите event, section и location се прави index scan и тоа е бидејки учествуваат во join по нивниот примарен клуч. Времето на изведување на овој прашалник е 500мс што е прифатливо време.
(Ова ни беше кажано од асистентот Милан да биде проверено на факултетската база времето за да видиме дали има потреба од ставање на предлог индексот што го имавме ставено на табелата waitlist_entry на атрибутот event_schedule_session_id. Но, како што можеме да видиме времето на извршување е во мс така да нема потреба од индексот и од оптимизација)

View5 - Event rating summary
Овој поглед би можел да се користи од сите, т.е од корисниците, од организаторите на настаните а исто така и од администраторите. Целта на овој поглед е за даден настан да видиме статистика или сумаризација на неговите ratings.

Погледот немаште потреба од оптимизација во однос на редолследот на извршување и филтрирање на редовите.

```sql
CREATE OR REPLACE VIEW public.v_event_rating_summary_2 AS
SELECT
    e.event_id,
    e.title AS event_title,
    COUNT(r.review_id) AS review_count,
    COALESCE(ROUND(AVG(r.star_rating), 2), 0) AS average_rating,
    COUNT(r.review_id) FILTER (WHERE r.star_rating = 5) AS five_star_reviews,
    COUNT(r.review_id) FILTER (WHERE r.star_rating = 4) AS four_star_reviews,
    COUNT(r.review_id) FILTER (WHERE r.star_rating <= 2) AS negative_reviews,
    MAX(r.created_at) AS latest_review_at
FROM public.event e
        LEFT JOIN public.review r
                  1<->0..n: ON r.event_id = e.event_id
GROUP BY
    e.event_id,
    e.title;
```

Прашалникот во однос на овој поглед:

```sql
select *
from v_event_rating_summary_2
where event_id=471398;
```

Планот на извршување на овој поглед:

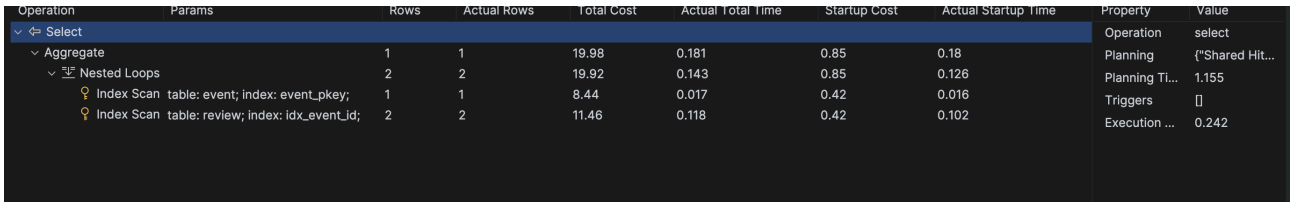
Operation	Params	Rows	Actual Rows	Total Cost	Actual Total Time	Startup Cost	Actual Startup Time
Select							
Aggregate		1	1	18122.05	91.335	1000.42	90.497
Nested Loops (N		2	2	18121.99	91.319	1000.42	0.547
Index Scan	table: event; index: event_pkey;	1	1	8.44	0.021	0.42	0.018
Unknown (Gather		2	2	18113.53	91.294	1000	0.526
Full Scan (S	table: review;	1	0.67	17113.33	86.457	0	42.043

Property	Value
Operation	select
Planning	{"Shared Hit...
Planning Ti...	2.783
Triggers	[]
Execution ...	91.378

Како што може да видиме времето на извршување на овој прашалник е сосема прифатливо и нема потреба од оптимизирање понатаму.

*Мора да се напомене дека покрај тоа што нема потреба од оптимизација на овој поглед ние имаме индекст поставено на табелата review на атрибутот event_id. Овој индекс не е наменет за овој поглед бидејќи но бидиејќи го правиме за друг поглед исто така се употребува и тука.

Планот со извршување и времето на извршување со индексот:



Operation	Params	Rows	Actual Rows	Total Cost	Actual Total Time	Startup Cost	Actual Startup Time
Select							
Aggregate		1	1	19.98	0.181	0.85	0.18
Nested Loops		2	2	19.92	0.143	0.85	0.126
Index Scan	table: event; index: event_pkey;	1	1	8.44	0.017	0.42	0.016
Index Scan	table: review; index: idx_event_id;	2	2	11.46	0.118	0.42	0.102

Property	Value
Operation	select
Planning	{"Shared Hit...
Planning Ti...	1.155
Triggers	[]
Execution ...	0.242

Од добро иде на уште подобро, но повтроно намената на индексот не е за овој поглед. Споредбата за времето на insert и update на оваа табела е поставена кај погледот за кој е наменет индексот.

View6 - Organiser performance dashboard
Овој поглед се користи од страна на администраторите на страната и погледот ни дава за даден организатор нивна статистика. Во статистиката спаѓаат вкупно организирани настани, продадени билети, итн.

Првичниот поглед:

```
CREATE OR REPLACE VIEW public.v_organiser_performance_dashboard AS
SELECT
    o.organiser_id,
    o.company_name AS organiser_name,
    COUNT(DISTINCT e.event_id) AS total_events,
    COUNT(DISTINCT t.ticket_id) AS total_tickets_sold,
    COUNT(DISTINCT oc.order_id) AS total_orders,
    COALESCE(SUM(p.amount_paid), 0) AS total_revenue,
    ROUND(AVG(r.star_rating), 2) AS average_event_rating,
    COUNT(DISTINCT r.review_id) AS total_reviews
FROM public.organiser o
        LEFT JOIN public.event e
                1<->0..n: ON e.organiser_id = o.organiser_id
        LEFT JOIN public.ticket_type tt
                1<->0..n: ON tt.event_id = e.event_id
        LEFT JOIN public.ticket t
                1<->0..n: ON t.ticket_type_id = tt.ticket_type_id
        LEFT JOIN public.order_cart oc
                1..n<->1: ON oc.order_id = t.order_id
        LEFT JOIN public.payment p
                1<->0..n: ON p.order_id = oc.order_id
        LEFT JOIN public.review r
                1<->0..n: ON r.event_id = e.event_id
GROUP BY
    o.organiser_id,
    o.company_name;
```

Погледот после оптимизација:

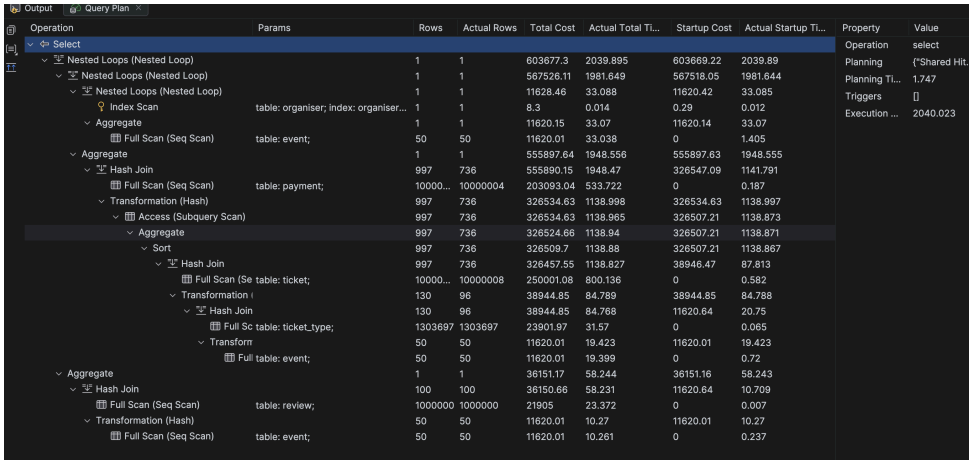
```
CREATE OR REPLACE VIEW public.v_organiser_performance_dashboard_2 AS
SELECT
    o.organiser_id,
    o.company_name AS organiser_name,

    COALESCE(ev.total_events, 0) AS total_events,
    COALESCE(sales.total_tickets_sold, 0) AS total_tickets_sold,
    COALESCE(sales.total_orders, 0) AS total_orders,
    COALESCE(sales.total_revenue, 0) AS total_revenue,
    reviews.average_event_rating,
    COALESCE(reviews.total_reviews, 0) AS total_reviews

FROM public.organiser o

        LEFT JOIN LATERAL (
    SELECT COUNT(*) AS total_events
    FROM public.event e
    WHERE e.organiser_id = o.organiser_id
    ) ev ON true

        LEFT JOIN LATERAL (
    SELECT
        COALESCE(SUM(x.ticket_count), 0) AS total_tickets_sold,
        COUNT(*) AS total_orders,
        COALESCE(SUM(p.amount_paid), 0) AS total_revenue
    FROM (
            SELECT
                t.order_id,
                COUNT(*) AS ticket_count
            FROM public.event e
                    JOIN public.ticket_type tt
                            1<->1..n: ON tt.event_id = e.event_id
                    JOIN public.ticket t
                            1<->1..n: ON t.ticket_type_id = tt.ticket_type_id
            WHERE e.organiser_id = o.organiser_id
            GROUP BY t.order_id
        ) x
            LEFT JOIN public.payment p
                    ON p.order_id = x.order_id
    ) sales ON true

        LEFT JOIN LATERAL (
    SELECT
        ROUND(AVG(r.star_rating), 2) AS average_event_rating,
        COUNT(r.review_id) AS total_reviews
    FROM public.event e
            JOIN public.review r
                    1<->1.n: ON r.event_id = e.event_id
    WHERE e.organiser_id = o.organiser_id
    ) reviews ON true;
```

Оптимизација е иста како и претходните погледи тоест агрегацијата и филтрирањето се прават пред да се извршат joins и ова ни помага со брзината, т.е. работиме со илјадници редови место со милиони.

Прашаликот поврзан со овој поглед:

```
select *
from v_organiser_performance_dashboard_2
where organiser_id=6000;
```

Планот за изведувањето на овој прашаликот и времето на изведвуање без ниту еден индекс:



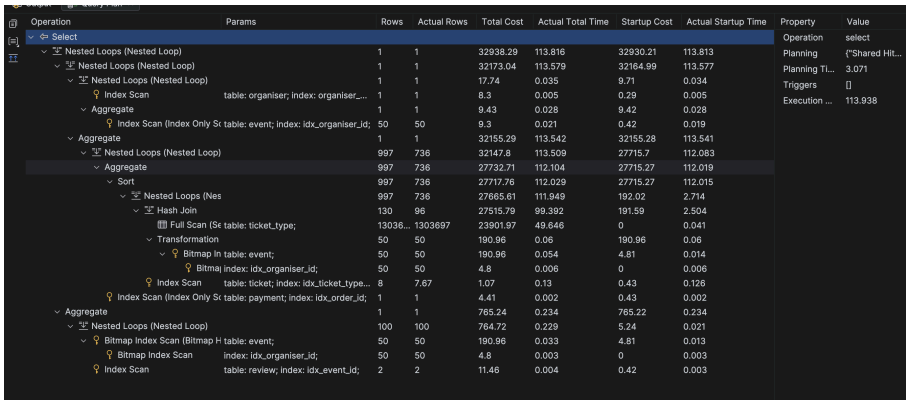
Operation	Params	Rows	Actual Rows	Total Cost	Actual Total Ti...	Startup Cost	Actual Startup Ti...
Select							
Nested Loops (Nested Loop)		1	1	603677.3	2039.895	603669.22	2039.89
Nested Loops (Nested Loop)		1	1	567526.11	1981.649	567518.05	1981.644
Nested Loops (Nested Loop)		1	1	11628.46	33.088	11620.42	33.085
Index Scan	table: organiser; index: organiser...	1	1	8.3	0.014	0.29	0.012
Aggregate		1	1	11620.15	33.07	11620.14	33.07
Full Scan (Seq Scan)	table: event;	50	50	11620.01	33.038	0	1.405
Aggregate		1	1	555897.64	1948.556	555897.63	1948.555
Hash Join		997	736	555890.15	1948.47	326547.09	1141.791
Full Scan (Seq Scan)	table: payment;	10000...	10000004	203093.04	533.722	0	0.187
Transformation (Hash)		997	736	326534.63	1138.998	326534.63	1138.997
Access (Subquery Scan)		997	736	326534.63	1138.965	326507.21	1138.873
Aggregate		997	736	326524.66	1138.94	326507.21	1138.871
Sort		997	736	326509.7	1138.88	326507.21	1138.867
Hash Join		997	736	326457.55	1138.827	38946.47	87.813
Full Scan (Se	table: ticket;	10000...	10000008	250001.08	800.136	0	0.582
Transformation		130	96	38944.85	84.789	38944.85	84.788
Hash Join		130	96	38944.85	84.768	11620.64	20.75
Full Sc	table: ticket_type;	1303697	1303697	23901.97	31.57	0	0.065
Transforn		50	50	11620.01	19.423	11620.01	19.423
Full	table: event;	50	50	11620.01	19.399	0	0.72
Aggregate		1	1	36151.17	58.244	36151.16	58.243
Hash Join		100	100	36150.66	58.231	11620.64	10.709
Full Scan (Seq Scan)	table: review;	1000000	1000000	21905	23.372	0	0.007
Transformation (Hash)		50	50	11620.01	10.27	11620.01	10.27
Full Scan (Seq Scan)	table: event;	50	50	11620.01	10.261	0	0.237

Property	Value
Operation	select
Planning	{"Shared Hit...
Planning Ti...	1.747
Triggers	[]
Execution ...	2040.023

Времето на извршување на овој поглед е 2 секунди што е едвај прифатливо но може многу подобрно. Затоа ќе извршиме оптимизација на овој поглед.

*Една работа треба да се напомене. Индексите што ги направивме на табелите payment и ticket повторно ни испаѓаат во корист и за овој поглед, но тие сами по себе не се доволни бидејќи овој е голем поглед со повеќе joins. Затоа додаваме уште два индекси. Едниот е тој што претходно го спомнавме, т.е индексот на табелата review на event_id и другиот е индекс на табелата event на атрибутот organiser_id.

Планот и времето на извршување после индексот:



Operation	Params	Rows	Actual Rows	Total Cost	Actual Total Time	Startup Cost	Actual Startup Time
Select							
Nested Loops (Nested Loop)		1	1	32938.29	113.816	32930.21	113.813
Nested Loops (Nested Loop)		1	1	32173.04	113.579	32164.99	113.577
Nested Loops (Nested Loop)		1	1	17.74	0.035	9.71	0.034
Index Scan	table: organiser; index: organiser_...	1	1	8.3	0.005	0.29	0.005
Aggregate		1	1	9.43	0.028	9.42	0.028
Index Scan (Index Only Sc	table: event; index: idx_organiser_id;	50	50	9.3	0.021	0.42	0.019
Aggregate		1	1	32155.29	113.542	32155.28	113.541
Nested Loops (Nested Loop)		997	736	32147.8	113.509	27715.7	112.083
Aggregate		997	736	27732.71	112.104	27715.27	112.019
Sort		997	736	27717.76	112.029	27715.27	112.015
Nested Loops (Nes		997	736	27665.61	111.949	192.02	2.714
Hash Join		130	96	27515.79	99.392	191.59	2.504
Full Scan (Sc	table: ticket_type;	13036...	1303697	23901.97	49.646	0	0.041
Transformation		50	50	190.96	0.06	190.96	0.06
Bitmap In	table: event;	50	50	190.96	0.054	4.81	0.014
Bitmap	index: idx_organiser_id;	50	50	4.8	0.006	0	0.006
Index Scan	table: ticket; index: idx_ticket_type...	8	7.67	1.07	0.13	0.43	0.126
Index Scan (Index Only Sc	table: payment; index: idx_order_id;	1	1	4.41	0.002	0.43	0.002
Aggregate		1	1	765.24	0.234	765.22	0.234
Nested Loops (Nested Loop)		100	100	764.72	0.229	5.24	0.021
Bitmap Index Scan (Bitmap H	table: event;	50	50	190.96	0.033	4.81	0.013
Bitmap Index Scan	index: idx_organiser_id;	50	50	4.8	0.003	0	0.003
Index Scan	table: review; index: idx_event_id;	2	2	11.46	0.004	0.42	0.003

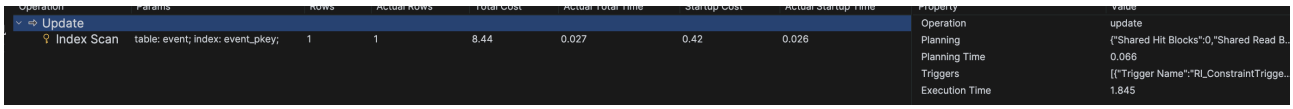
Property	Value
Operation	select
Planning	{"Shared Hit...
Planning Ti...	3.071
Triggers	[]
Execution ...	113.938

Како што може да видиме има значително подобрување во времето.

Време на извршување на insert и update на табелата event пред поставување на индекс:

```
INSERT INTO event
values ( organiser_id 1, event_id 971398, event_status_id 6,
       title 'Ohrid Cultural Night 2', start_datetime '2023-01-02 19:00:00.000000', end_datetime '2023-01-02 19:00:00.000000');
```

```
UPDATE event set organiser_id=2 where event_id=971398;
```

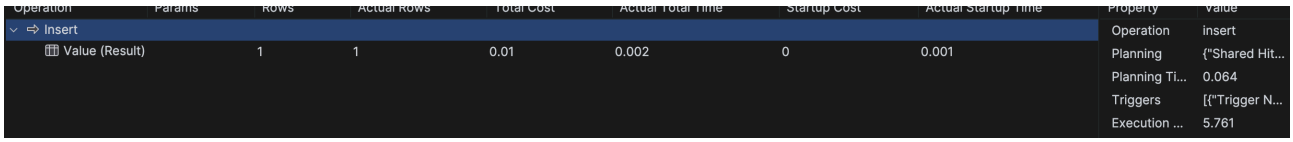


Operation	Params	Rows	Actual Rows	Total Cost	Actual Total Time	Startup Cost	Actual Startup Time
Update							
Index Scan	table: event; index: event_pkey;	1	1	8.44	0.027	0.42	0.026

Property	Value
Operation	update
Planning	{"Shared Hit Blocks":0,"Shared Read B...
Planning Time	0.066
Triggers	[{"Trigger Name":"RI_ConstraintTrigge...
Execution Time	1.845

Време на извршување на insert и update на табелата review пред индекс:

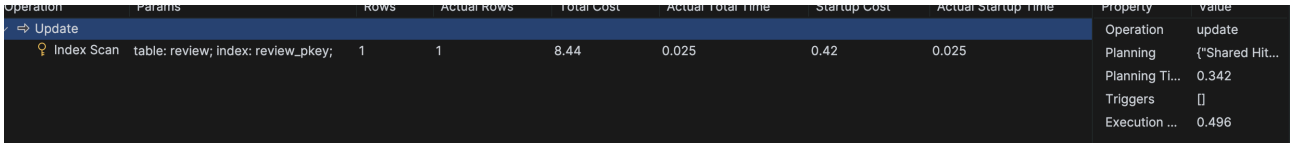
```
INSERT INTO review
values( user_id 3, event_id 471398, review_id 4000001, star_rating 4,
        review_text 'Good event, but the venue was crowded.', created_at '2023-01-02 12:00:01.000000');
```



Operation	Params	Rows	Actual Rows	Total Cost	Actual Total Time	Startup Cost	Actual Startup Time
Insert							
Value (Result)		1	1	0.01	0.002	0	0.001

Property	Value
Operation	insert
Planning	{"Shared Hit...
Planning Ti...	0.064
Triggers	[{"Trigger N...
Execution ...	5.761

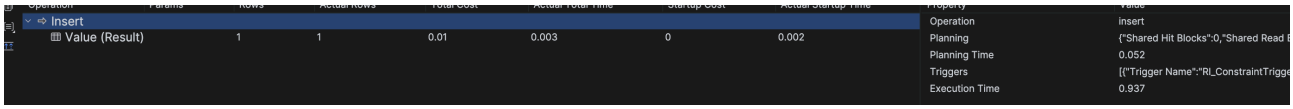
```
UPDATE review set event_id=471398 where review_id=4000001
```



Operation	Params	Rows	Actual Rows	Total Cost	Actual Total Time	Startup Cost	Actual Startup Time
Update							
Index Scan	table: review; index: review_pkey;	1	1	8.44	0.025	0.42	0.025

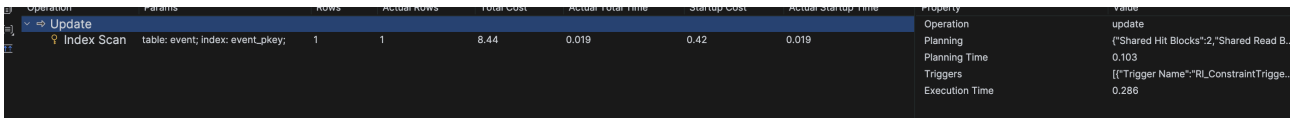
Property	Value
Operation	update
Planning	{"Shared Hit...
Planning Ti...	0.342
Triggers	[]
Execution ...	0.496

Време на извршување на insert и update после ставање на индексот:
Event:



Operation	Params	Rows	Actual Rows	Total Cost	Actual Total Time	Startup Cost	Actual Startup Time
Insert							
Value (Result)		1	1	0.01	0.003	0	0.002

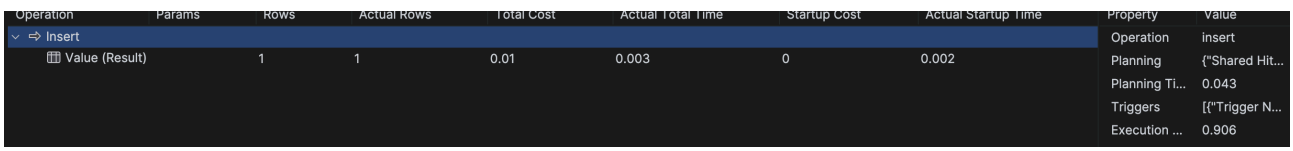
Property	Value
Operation	insert
Planning	{"Shared Hit Blocks":0,"Shared Read B...
Planning Time	0.052
Triggers	[{"Trigger Name":"RI_ConstraintTrigge...
Execution Time	0.937



Operation	Params	Rows	Actual Rows	Total Cost	Actual Total Time	Startup Cost	Actual Startup Time
Update							
Index Scan	table: event; index: event_pkey;	1	1	8.44	0.019	0.42	0.019

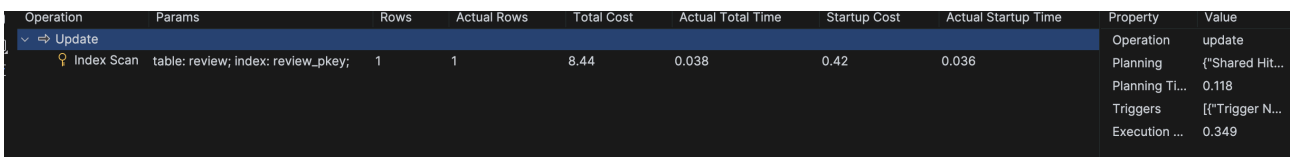
Property	Value
Operation	update
Planning	{"Shared Hit Blocks":2,"Shared Read B...
Planning Time	0.103
Triggers	[{"Trigger Name":"RI_ConstraintTrigge...
Execution Time	0.286

Review:



Operation	Params	Rows	Actual Rows	Total Cost	Actual Total Time	Startup Cost	Actual Startup Time
Insert							
Value (Result)		1	1	0.01	0.003	0	0.002

Property	Value
Operation	insert
Planning	{"Shared Hit...
Planning Ti...	0.043
Triggers	[{"Trigger N...
Execution ...	0.906



Operation	Params	Rows	Actual Rows	Total Cost	Actual Total Time	Startup Cost	Actual Startup Time
Update							
Index Scan	table: review; index: review_pkey;	1	1	8.44	0.038	0.42	0.036

Property	Value
Operation	update
Planning	{"Shared Hit...
Planning Ti...	0.118
Triggers	[{"Trigger N...
Execution ...	0.349

Креирањето на индексите:

```sql
CREATE INDEX idx_organiser_id ON event(organiser_id);
DROP INDEX idx_organiser_id;
CREATE INDEX idx_event_id ON review(event_id);
DROP INDEX idx_event_id;
```

View7 - Location utilization summary
Овој поглед би бил користен од страна на менаџери на локациите.
Преку овој полгед за дадена локација може да се провери статистика за истата.

Погледот пред да е оптимизира:

```sql
CREATE OR REPLACE VIEW public.v_location_utilization_summary AS
SELECT
    l.location_id,
    l.name AS location_name,
    lt.type_name AS location_type,
    COUNT(DISTINCT s.section_id) AS total_sections,
    COUNT(DISTINCT seat.seat_id) AS total_seats,
    COUNT(DISTINCT ess.schedule_id) AS scheduled_sessions,
    COUNT(DISTINCT e.event_id) AS hosted_events,
    COUNT(DISTINCT seat.seat_id) FILTER (WHERE seat.is_accessible = true) AS accessible_seats,
    COUNT(DISTINCT seat.seat_id) FILTER (WHERE seat.is_available = true) AS available_seats
FROM public.location l
        JOIN public.location_type lt
             1..n<->1: ON lt.type_id = l.type_id
        LEFT JOIN public.section s
                  1<->0..n: ON s.location_id = l.location_id
        LEFT JOIN public.seat seat
                  1<->0..n: ON seat.section_id = s.section_id
        LEFT JOIN public.event_schedule_session ess
                  1<->0..n: ON ess.section_id = s.section_id
        LEFT JOIN public.event e
                  1..n<->1: ON e.event_id = ess.event_id
GROUP BY
    l.location_id,
    l.name,
    lt.type_name;
```

Погледот после оптимизација:

```sql
CREATE OR REPLACE VIEW public.v_location_utilization_summary_2 AS
SELECT
    l.location_id,
    l.name AS location_name,
    lt.type_name AS location_type,

    COALESCE(sec.total_sections, 0) AS total_sections,
    COALESCE(seat_stats.total_seats, 0) AS total_seats,
    COALESCE(session_stats.scheduled_sessions, 0) AS scheduled_sessions,
    COALESCE(session_stats.hosted_events, 0) AS hosted_events,
    COALESCE(seat_stats.accessible_seats, 0) AS accessible_seats,
    COALESCE(seat_stats.available_seats, 0) AS available_seats

FROM public.location l
        JOIN public.location_type lt
             1..n<->1: ON lt.type_id = l.type_id

        LEFT JOIN LATERAL (
    SELECT COUNT(*) AS total_sections
    FROM public.section s
    WHERE s.location_id = l.location_id
    ) sec ON true

        LEFT JOIN LATERAL (
    SELECT
        COUNT(seat.seat_id) AS total_seats,
        COUNT(seat.seat_id) FILTER (WHERE seat.is_accessible = true) AS accessible_seats,
        COUNT(seat.seat_id) FILTER (WHERE seat.is_available = true) AS available_seats
    FROM public.section s
            JOIN public.seat seat
                 1<->1..n: ON seat.section_id = s.section_id
    WHERE s.location_id = l.location_id
    ) seat_stats ON true

        LEFT JOIN LATERAL (
    SELECT
        COUNT(ess.schedule_id) AS scheduled_sessions,
        COUNT(DISTINCT ess.event_id) AS hosted_events
    FROM public.section s
            JOIN public.event_schedule_session ess
                 1<->1..n: ON ess.section_id = s.section_id
    WHERE s.location_id = l.location_id
    ) session_stats ON true;
```

Оптимизацијата на погледот во однос на редолседот е исто како претходните оптимизации на погледите. Ни помага во однос на меморијата.
Прашалникот за овој поглед:

```sql
select *
from v_location_utilization_summary_2
where location_id=1000;
```

Планот за извршувањето на погледот и потребното време:

Operation	Params	Rows	Actual Rows	Total Cost	Actual Total Time	Startup Cost	Actual Startup Time
Insert							
Value (Result)		1	1	0.01	0.002	0	0.001

Property	Value
Operation	insert
Planning	{"Shared Hit...
Planning Ti...	0.065
Triggers	[{"Trigger N...
Execution ...	2.725

Output | Query Plan

Operation	Params	Rows	Actual Rows	Total Cost	Actual Total Time	Startup Cost	Actual Startup Time
Select							
Nested Loops (Nested L		1	1	211248.92	2397.121	211232.77	2397.102
Nested Loops (Neste		1	1	189523.96	2139.958	189507.83	2139.944
Nested Loops (Ne		1	1	1958.58	40.944	1942.48	40.935
Nested Loops		1	1	16.51	28.675	0.43	28.668
Index Scan	table: location; index: location_pkey;	1	1	8.3	0.06	0.28	0.054
Index Scan	table: location_type; index: location_type_pkey;	1	1	8.17	0.072	0.15	0.071
Aggregate		1	1	1942.06	12.256	1942.05	12.255
Full Scan (S	table: section;	20	20	1942	12.196	0	0.206
Aggregate		1	1	187565.36	2098.999	187565.35	2098.995
Hash Join		2000	2000	187550.35	2097.615	1942.25	8.587
Full Scan (S	table: seat;	100000...	10000000	159357	986.059	0	0.031
Transformatior		20	20	1942	8.256	1942	8.255
Full Scar	table: section;	20	20	1942	8.234	0	0.126
Aggregate		1	1	21724.95	257.144	21724.94	257.142
Sort		180	200	21724.04	257.052	21723.59	257.035
Hash Join		180	200	21716.85	256.932	1942.25	11.657
Full Scan (S	table: event_schedule_session;	900000	900000	17412	106.622	0	0.032
Transformatior		20	20	1942	8.489	1942	8.488
Full Scar	table: section;	20	20	1942	8.468	0	0.127

Property	Value
Operation	select
Planning Ti...	0.98
Triggers	[]
JIT	{"Functions"...
Execution ...	2400.017

Како што може да видиме времето е 2.5 секунди и тоа време е неприфатливо. Од планот на извршување може да приметиме дека најмногу време одземаат табелите seat и event_schedule_session, seat особено гледајки дека тоа е табела со 10 милиони редици. Овој прашалник ке биде оптимизиран со индекс на табелата seat на атрибутот section_id и исто така индекс на event_schedule_session на атрибутот section_id.

Operation	Params	Rows	Actual Rows	Total Cost	Actual Total Time	Startup Cost	Actual Startup Time
Select							
Nested Loops (Nested		1	1	13788.84	65.304	13772.7	65.29
Nested Loops (Nes		1	1	11665.28	54.461	11649.15	54.45
Nested Loops (I		1	1	1958.58	13.318	1942.48	13.311
Nested Loop		1	1	16.51	0.095	0.43	0.09
Index Sca	table: location; index: location_pkey;	1	1	8.3	0.061	0.28	0.058
Index Sca	table: location_type; index: location_type_pkey;	1	1	8.17	0.028	0.15	0.027
Aggregate		1	1	1942.06	13.218	1942.05	13.217
Full Scan	table: section;	20	20	1942	13.2	0	0.171
Aggregate		1	1	9706.68	41.134	9706.67	41.131
Nested Loop		2000	2000	9691.67	40.408	5.2	0.2
Full Scan	table: section;	20	20	1942	10.127	0	0.115
Bitmap In	table: seat;	99	100	386.49	1.464	5.2	0.065
Bitmap	index: idx_section_id;	99	100	5.18	0.037	0	0.037
Aggregate		1	1	2123.55	10.834	2123.54	10.832
Sort		180	200	2122.64	10.801	2122.19	10.789
Nested Loop		180	200	2115.45	10.712	0.42	0.161
Full Scan	table: section;	20	20	1942	9.843	0	0.116
Index Sca	table: event_schedule_session; index: idx_section_id_s...	9	10	8.58	0.039	0.42	0.036

Property	Value
Operation	select
Planning Ti...	2.128
Triggers	[]
Execution ...	65.44

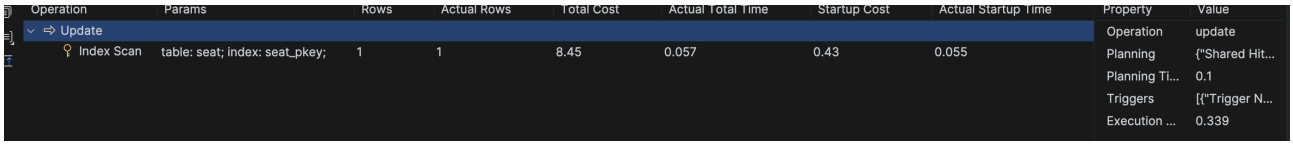
По креирањето на индекстите планот на извршување и времето се следниве:

Како што може да видиме времето е многу подобрено и сега е прифатливо. Беа пробани индексите индивидувално да се види како ке работат. Индексот на seat сам по себе исто така работеше многу добро но во комбинација со индексот на event_schedule_session работат одлично. Индексот event_schedule_session сам по себе немаше толкаво влијание на времето бидејќи покрај тоа што одзема доста време главната причина со толку долгото извшување на овој прашалник беше табелата seat.

Времето за insert и update на табелите seat и event_schedule_session пред поставување на индексот:

```
INSERT INTO seat
values ( seat_id 10000001, row_identifier 'A', section_id 2, seat_number 1, is_accessible false, is_available false);
```

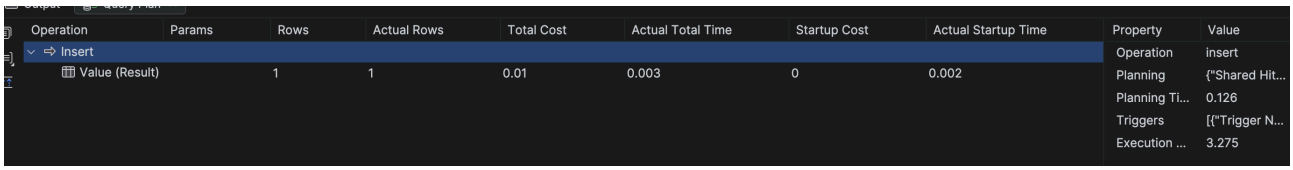
```
UPDATE seat set section_id=3 where seat_id=10000001;
```



Operation	Params	Rows	Actual Rows	Total Cost	Actual Total Time	Startup Cost	Actual Startup Time
Update							
Index Scan	table: seat; index: seat_pkey;	1	1	8.45	0.057	0.43	0.055

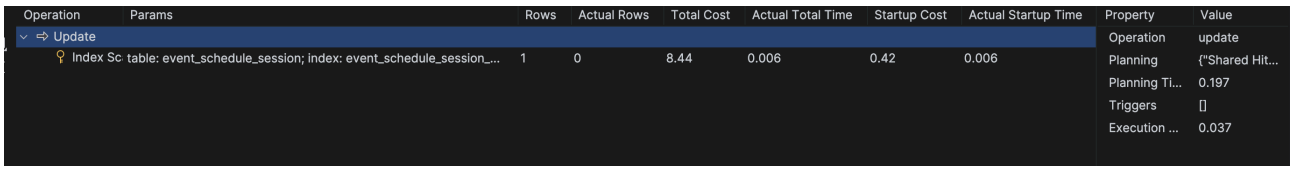
Property	Value
Operation	update
Planning	{"Shared Hit...
Planning Ti...	0.1
Triggers	[{"Trigger N...
Execution ...	0.339

```
INSERT INTO event_schedule_session
values ( event_id 500000, schedule_id 900001, session_title '' ||
                    'Main Program', start_time '2028-05-13 21:00:00.000000', end_time '' ||
                                                    '2025-05-13 22:00:00.000000', section_id 1);
```



Operation	Params	Rows	Actual Rows	Total Cost	Actual Total Time	Startup Cost	Actual Startup Time
Insert							
Value (Result)		1	1	0.01	0.003	0	0.002

Property	Value
Operation	insert
Planning	{"Shared Hit...
Planning Ti...	0.126
Triggers	[{"Trigger N...
Execution ...	3.275

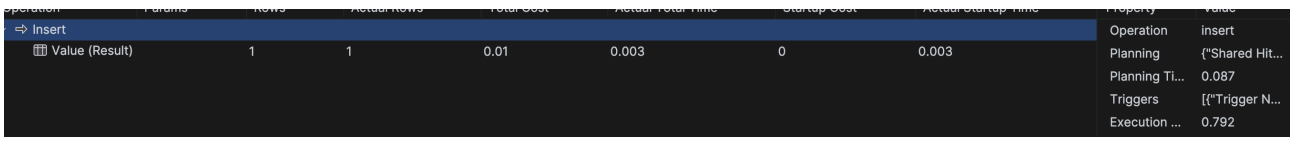


Operation	Params	Rows	Actual Rows	Total Cost	Actual Total Time	Startup Cost	Actual Startup Time
Update							
Index Sc	table: event_schedule_session; index: event_schedule_session_...	1	0	8.44	0.006	0.42	0.006

Property	Value
Operation	update
Planning	{"Shared Hit...
Planning Ti...	0.197
Triggers	[]
Execution ...	0.037

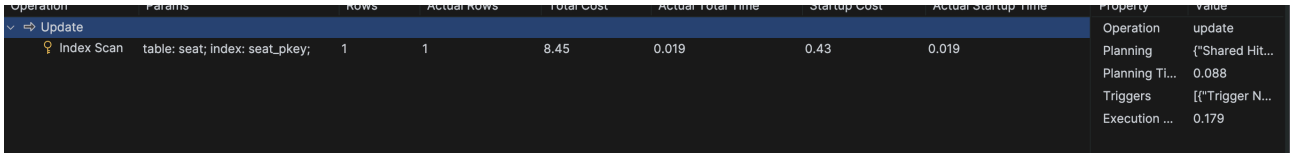
Време на insert и update операциите по поставувањето на индексите:

Seat:



Operation	Params	Rows	Actual Rows	Total Cost	Actual Total Time	Startup Cost	Actual Startup Time
Insert							
Value (Result)		1	1	0.01	0.003	0	0.003

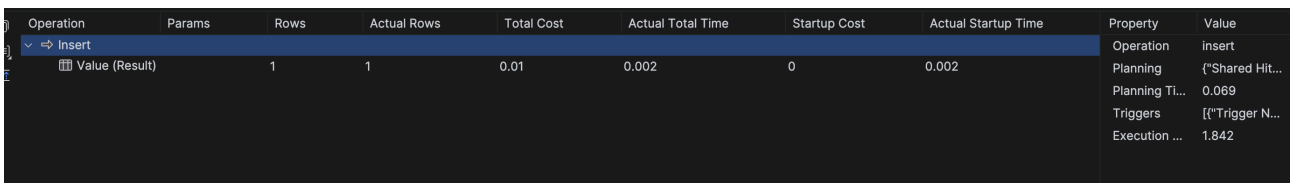
Property	Value
Operation	insert
Planning	{"Shared Hit...
Planning Ti...	0.087
Triggers	[{"Trigger N...
Execution ...	0.792



Operation	Params	Rows	Actual Rows	Total Cost	Actual Total Time	Startup Cost	Actual Startup Time
Update							
Index Scan	table: seat; index: seat_pkey;	1	1	8.45	0.019	0.43	0.019

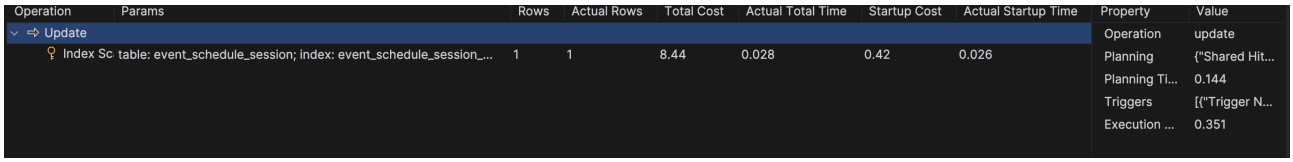
Property	Value
Operation	update
Planning	{"Shared Hit...
Planning Ti...	0.088
Triggers	[{"Trigger N...
Execution ...	0.179

Event_schedule_session:



Operation	Params	Rows	Actual Rows	Total Cost	Actual Total Time	Startup Cost	Actual Startup Time
Insert							
Value (Result)		1	1	0.01	0.002	0	0.002

Property	Value
Operation	insert
Planning	{"Shared Hit...
Planning Ti...	0.069
Triggers	[{"Trigger N...
Execution ...	1.842



Operation	Params	Rows	Actual Rows	Total Cost	Actual Total Time	Startup Cost	Actual Startup Time
Update							
Index Sc	table: event_schedule_session; index: event_schedule_session_...	1	1	8.44	0.028	0.42	0.026

Property	Value
Operation	update
Planning	{"Shared Hit...
Planning Ti...	0.144
Triggers	[{"Trigger N...
Execution ...	0.351

Креирањето на индексите:

```sql
CREATE INDEX idx_section_id ON seat(section_id);
DROP INDEX idx_section_id;

CREATE INDEX idx_section_id_session ON event_schedule_session(section_id);
DROP INDEX idx_section_id_session;
```

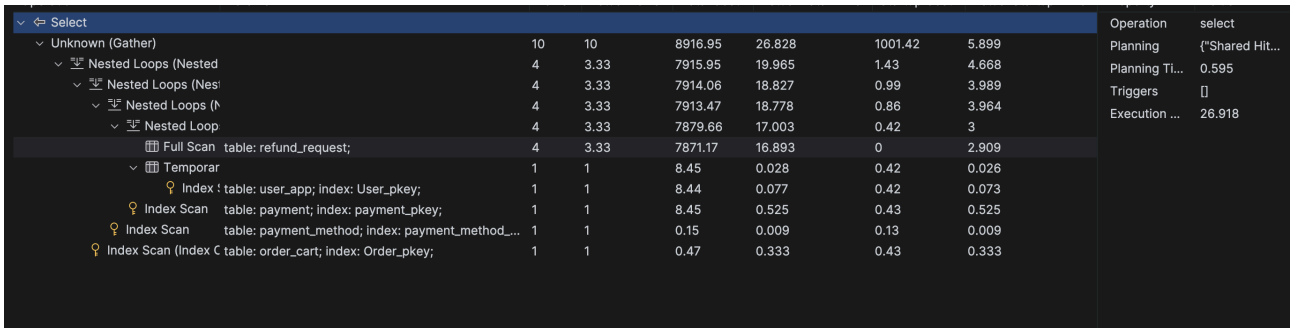
View8 - Refund analysis:

Овој поглед би се користел од страна на финансиски тим за анализа на тоа зошто всушност се праат тие refunds што би било понатаму корисно за правење на подобри понуди.

За овој поглед немаше потреба на оптимизација во однос на редолседот и извшувањето на агрегатните функции и филтрирање.

```sql
CREATE OR REPLACE VIEW public.v_refund_analysis AS
SELECT
    rr.refund_request_id,
    rr.requested_at,
    rr.accepted_at,
    CASE
        WHEN rr.accepted_at IS NOT NULL THEN 'ACCEPTED'
        ELSE 'PENDING'
        END AS refund_status,
    u.user_id,
    u.username,
    oc.order_id,
    p.payment_id,
    p.amount_paid,
    p.method_id,
    pm.method_name,
    rr.reason
FROM public.refund_request rr
         JOIN public.user_app u
              1..n<->1: ON u.user_id = rr.user_id
         JOIN public.payment p
              1..n<->1: ON p.payment_id = rr.payment_id
         JOIN public.payment_method pm
              1..n<->1: ON pm.method_id = p.method_id
         JOIN public.order_cart oc
              1..n<->1: ON oc.order_id = p.order_id;
```

Планот на извршување на прашалникот и времето на извршување е следното:

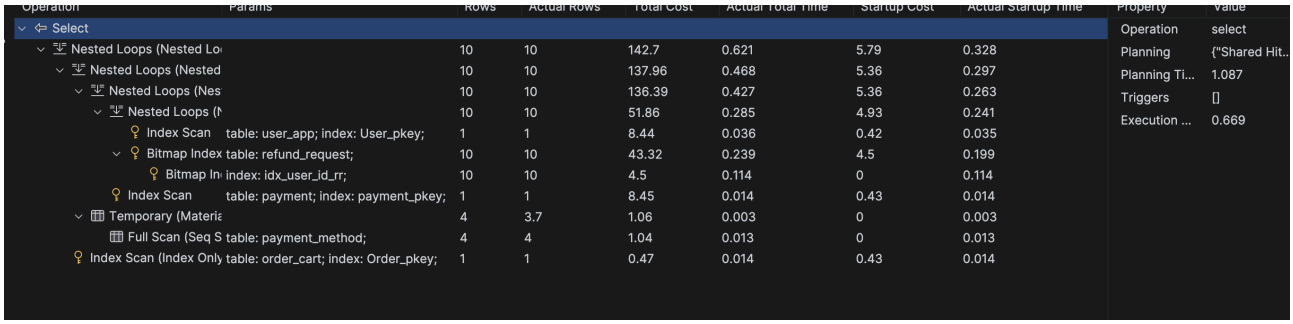


Operation	Params	Rows	Actual Rows	Total Cost	Actual Total Time	Startup Cost	Actual Startup Time
Select							
Unknown (Gather)		10	10	8916.95	26.828	1001.42	5.899
Nested Loops (Nested		4	3.33	7915.95	19.965	1.43	4.668
Nested Loops (Nes		4	3.33	7914.06	18.827	0.99	3.989
Nested Loops (N		4	3.33	7913.47	18.778	0.86	3.964
Nested Loop		4	3.33	7879.66	17.003	0.42	3
Full Scan	table: refund_request;	4	3.33	7871.17	16.893	0	2.909
Temporar		1	1	8.45	0.028	0.42	0.026
Index	table: user_app; index: User_pkey;	1	1	8.44	0.077	0.42	0.073
Index Scan	table: payment; index: payment_pkey;	1	1	8.45	0.525	0.43	0.525
Index Scan	table: payment_method; index: payment_method_...	1	1	0.15	0.009	0.13	0.009
Index Scan (Index C	table: order_cart; index: Order_pkey;	1	1	0.47	0.333	0.43	0.333

Property	Value
Operation	select
Planning	{"Shared Hit...
Planning Ti...	0.595
Triggers	[]
Execution ...	26.918

Сите табели освен refund_request кои учестуваат им се прави Index scan бидејќи учествуваат во join со нивниот примарен клуч и времето е веќе одлично. Но после обид на поставување на индекст на табелата refund_request на атрибутот user_id видовме дека има скоро 10x подобрување во времето. Затоа се решивме да го ставиме тој индекс.

Планот на извршување по поставувањето на индексот:

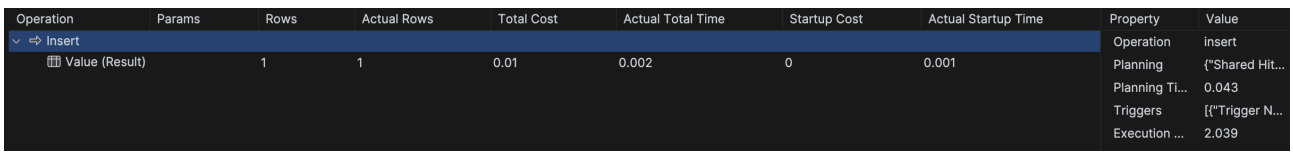


Operation	Params	Rows	Actual Rows	Total Cost	Actual Total Time	Startup Cost	Actual Startup Time
Select							
Nested Loops (Nested Lo		10	10	142.7	0.621	5.79	0.328
Nested Loops (Nested		10	10	137.96	0.468	5.36	0.297
Nested Loops (Nes		10	10	136.39	0.427	5.36	0.263
Nested Loops (N		10	10	51.86	0.285	4.93	0.241
Index Scan	table: user_app; index: User_pkey;	1	1	8.44	0.036	0.42	0.035
Bitmap Index	table: refund_request;	10	10	43.32	0.239	4.5	0.199
Bitmap In	index: idx_user_id_rr;	10	10	4.5	0.114	0	0.114
Index Scan	table: payment; index: payment_pkey;	1	1	8.45	0.014	0.43	0.014
Temporary (Materia		4	3.7	1.06	0.003	0	0.003
Full Scan (Seq S	table: payment_method;	4	4	1.04	0.013	0	0.013
Index Scan (Index Only	table: order_cart; index: Order_pkey;	1	1	0.47	0.014	0.43	0.014

Property	Value
Operation	select
Planning	{"Shared Hit...
Planning Ti...	1.087
Triggers	[]
Execution ...	0.669

Време на insert и update на табелата refund_request:

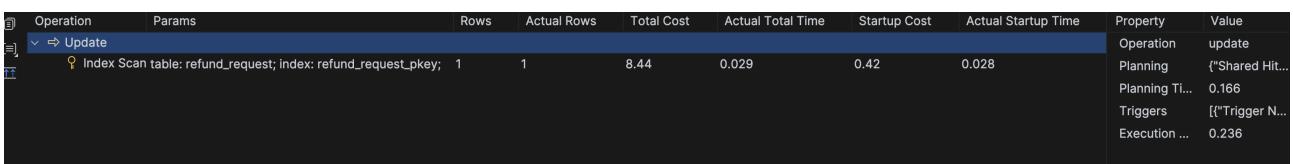
```
INSERT INTO refund_request
values ( refund_request_id 500001, reason 'User could not attend',
        requested_at '2023-04-13 10:05:02.000000', accepted_at '2023-04-15 10:05:02.000000', user_id 101, payment_id 201)
```



Operation	Params	Rows	Actual Rows	Total Cost	Actual Total Time	Startup Cost	Actual Startup Time
Insert							
Value (Result)		1	1	0.01	0.002	0	0.001

Property	Value
Operation	insert
Planning	{"Shared Hit...
Planning Ti...	0.043
Triggers	[{"Trigger N...
Execution ...	2.039

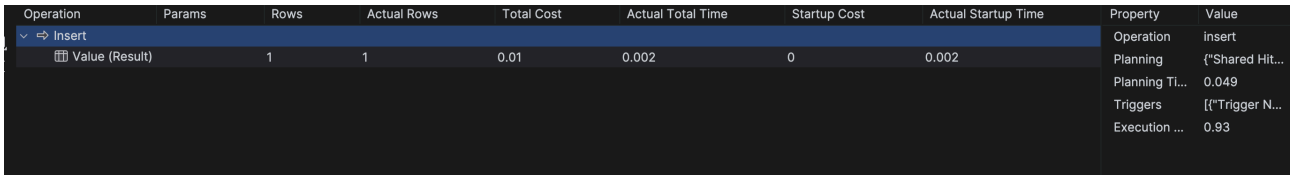
```
UPDATE refund_request set user_id=102 where refund_request_id=500001;
```



Operation	Params	Rows	Actual Rows	Total Cost	Actual Total Time	Startup Cost	Actual Startup Time
Update							
Index Scan	table: refund_request; index: refund_request_pkey;	1	1	8.44	0.029	0.42	0.028

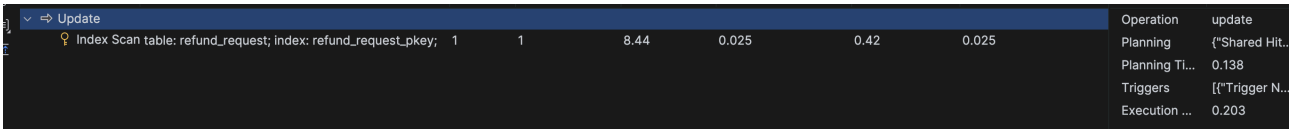
Property	Value
Operation	update
Planning	{"Shared Hit...
Planning Ti...	0.166
Triggers	[{"Trigger N...
Execution ...	0.236

Време после поставување на индексот:



Operation	Params	Rows	Actual Rows	Total Cost	Actual Total Time	Startup Cost	Actual Startup Time
Insert							
Value (Result)		1	1	0.01	0.002	0	0.002

Property	Value
Operation	insert
Planning	{"Shared Hit...
Planning Ti...	0.049
Triggers	[{"Trigger N...
Execution ...	0.93



Update						
Index Scan table: refund_request; index: refund_request_pkey;	1	1	8.44	0.025	0.42	0.025

Property	Value
Operation	update
Planning	{"Shared Hit...
Planning Ti...	0.138
Triggers	[{"Trigger N...
Execution ...	0.203

View 9 - Event_public_details
Овој поглед би бил користен на главната страна при browsing на events.
Најпрво, беше преспор за извршување, т.е. локално му требаше барем 11 минути…
Поради тоа што овој поглед содржи премногу информации, joins, пребарувања и слично, по консултации со асистентот Милан се договоривме да го претвориво во материјализиран поглед. Притоа, би бил refresh-иран овој поглед неделно/при bulk inserts. Но, бевме внимателни и при додавањето на индексите.
Прво го рефакторираме да биде поефикасен самиот поглед, слично како другите погледи (агрегација и помошни табели)

Стартиот изглед:

```
541  CREATE OR REPLACE VIEW public.v_event_public_details AS
542  SELECT
543      e.event_id,
544      e.title AS event_title,
545      e.start_datetime,
546      e.end_datetime,
547      es.status_name AS event_status,
548      o.company_name AS organiser_name,
549
550      STRING_AGG(DISTINCT c.name, ', ') AS categories,
551
552      COUNT(DISTINCT ess.schedule_id) AS session_count,
553      COUNT(DISTINCT sp.sponsor_id) AS sponsor_count,
554      COUNT(DISTINCT r.review_id) AS review_count,
555      ROUND(AVG(r.star_rating), 2) AS average_rating,
556
557      MIN(pt.price) AS lowest_ticket_price,
558      MAX(pt.price) AS highest_ticket_price,
559
560      COUNT(DISTINCT t.ticket_id) AS tickets_sold
561  FROM public.event e
562          JOIN public.event_status es
563               1..n<->1: ON es.event_status_id = e.event_status_id
564          JOIN public.organiser o
565               1..n<->1: ON o.organiser_id = e.organiser_id
566          LEFT JOIN public.event_category ec
567                    1<->0..n: ON ec.event_id = e.event_id
568          LEFT JOIN public.category c
569                    1..n<->1: ON c.id = ec.category_id
570          LEFT JOIN public.event_schedule_session ess
571                    1<->0..n: ON ess.event_id = e.event_id
572          LEFT JOIN public.sponsor_event se
573                    1<->0..n: ON se.event_id = e.event_id
574          LEFT JOIN public.sponsor sp
575                    1..n<->1: ON sp.sponsor_id = se.sponsor_id
576          LEFT JOIN public.review r
577                    1<->0..n: ON r.event_id = e.event_id
578          LEFT JOIN public.ticket_type tt
579                    1<->0..n: ON tt.event_id = e.event_id
580          LEFT JOIN public.price_tier pt
581                    1<->0..n: ON pt.ticket_type_id = tt.ticket_type_id
582          LEFT JOIN public.ticket t
583                    1<->0..n: ON t.ticket_type_id = tt.ticket_type_id
584  GROUP BY
585      e.event_id,
586      e.title,
587      e.start_datetime,
588      e.end_datetime,
589      es.status_name,
590      o.company_name;
```

Новиот:

```sql
-- 9. Event full search/details view
-- Used by public event browsing/search page.
CREATE MATERIALIZED VIEW public.mv_event_public_details AS
WITH category_stats AS (
    SELECT
        ec.event_id,
        STRING_AGG(DISTINCT c.name, ', ') AS categories
    FROM public.event_category ec
            JOIN public.category c
                 1..n<->1: ON c.id = ec.category_id
    GROUP BY ec.event_id
),
     session_stats AS (
         SELECT
             ess.event_id,
             COUNT(*) AS session_count
         FROM public.event_schedule_session ess
         GROUP BY ess.event_id
     ),
     sponsor_stats AS (
         SELECT
             se.event_id,
             COUNT(DISTINCT se.sponsor_id) AS sponsor_count
         FROM public.sponsor_event se
         GROUP BY se.event_id
     ),
     review_stats AS (
         SELECT
             r.event_id,
             COUNT(*) AS review_count,
             ROUND(AVG(r.star_rating), 2) AS average_rating
         FROM public.review r
         GROUP BY r.event_id
     ),
     price_stats AS (
         SELECT
             tt.event_id,
             MIN(pt.price) AS lowest_ticket_price,
             MAX(pt.price) AS highest_ticket_price
         FROM public.ticket_type tt
                 JOIN public.price_tier pt
                      1<->1..n: ON pt.ticket_type_id = tt.ticket_type_id
         GROUP BY tt.event_id
     ),
```

```sql
     price_stats AS (
         SELECT
             tt.event_id,
             MIN(pt.price) AS lowest_ticket_price,
             MAX(pt.price) AS highest_ticket_price
         FROM public.ticket_type tt
                 JOIN public.price_tier pt
                      1<->1..n: ON pt.ticket_type_id = tt.ticket_type_id
         GROUP BY tt.event_id
     ),
     ticket_stats AS (
         SELECT
             tt.event_id,
             COUNT(t.ticket_id) AS tickets_sold
         FROM public.ticket_type tt
                 JOIN public.ticket t
                      1<->1..n: ON t.ticket_type_id = tt.ticket_type_id
         GROUP BY tt.event_id
     )
SELECT
    e.event_id,
    e.title AS event_title,
    e.start_datetime,
    e.end_datetime,
    es.status_name AS event_status,
    o.company_name AS organiser_name,

    cs.categories,

    COALESCE(ss.session_count, 0) AS session_count,
    COALESCE(sp.sponsor_count, 0) AS sponsor_count,
    COALESCE(rs.review_count, 0) AS review_count,
    rs.average_rating,

    ps.lowest_ticket_price,
    ps.highest_ticket_price,

    COALESCE(ts.tickets_sold, 0) AS tickets_sold

FROM public.event e
         JOIN public.event_status es
              1..n<->1: ON es.event_status_id = e.event_status_id
         JOIN public.organiser o
              1..n<->1: ON o.organiser_id = e.organiser_id
         LEFT JOIN category_stats cs
                   ON cs.event_id = e.event_id
         LEFT JOIN session_stats ss
                   ON ss.event_id = e.event_id
         LEFT JOIN sponsor_stats sp
                   ON sp.event_id = e.event_id
         LEFT JOIN review_stats rs
```

```sql
         LEFT JOIN review_stats rs
                   ON rs.event_id = e.event_id
         LEFT JOIN price_stats ps
                   ON ps.event_id = e.event_id
         LEFT JOIN ticket_stats ts
                   ON ts.event_id = e.event_id;
```

Бидејќи ќе е споро самото креирање/update-ирање на овој материјализиран поглед, прво му ставивме индекс.

Индексов е опционален, ама послужува ако некогаш би барале еден специфичен настан од погледов (како што посочува и коментарот). Секако и другите индекси од горе (другите погледи) ненамерно би се искористиле и тука!

```
473     -- guarantees one row per event and makes where event_id = x fast;
474     CREATE UNIQUE INDEX idx_mv_event_public_details_event_id
475         ON public.mv_event_public_details(event_id);
```

Креирањето:

```
                                    LEFT JOIN review_stats rs
                                              ON rs.event_id = e.event_id
                                    LEFT JOIN price_stats ps
                                              ON ps.event_id = e.event_id
                                    LEFT JOIN ticket_stats ts
                                              ON ts.event_id = e.event_id
[2026-05-07 23:50:16] 500,000 rows affected in 4 s 465 ms
```

Вака би се повикувал погледот:

```
499     SELECT *
500     FROM public.mv_event_public_details;
```

Резултатот е повеќе од одличен.

Operation	Params	Rows	Actual Ro...	Total C...	Actual Total ...	Startup C...	Actual Startup ...
Select							
Full S table: mv_event_public...	5000...	500000		15752	38.664	0	0.031

Property	Value
Operation	Seq Scan
Table	mv_event_public_details
Rows	500000
Actual Rows	500000
Total Cost	15752
Actual Total Time	38.664
Startup Cost	0
Actual Startup Time	0.031

Refresh-от е исто така брз (локално):

```
REFRESH MATERIALIZED VIEW public.mv_event_public_details;
```

```
[2026-05-11 10:32:48] postgres.public> REFRESH MATERIALIZED VIEW public.mv_event_public_details
[2026-05-11 10:32:54] completed in 5 s 968 ms
```