

# Објаснување

## Функции

Материјализиран поглед - Просечен рејтинг за даден настан

- За даден event, ни враќа просечна оценка преку табелата reviews.

Функција 2 - Проверка дали корисникот смее да остави уште едно review

- За даден корисник и event, ни враќа true/false дали корисникот смее да го оцени настанот. Даден корисник, за настан кој го посетил смее да остави само едно review.

Функција 3 - Приказ на спонзори за даден настан

- За даден event, ни враќа табела со сите негови спонзори, типот на спонзорство и уплатената сума.

Функција 4 - Дали на даден билет му е истечен lock-от

- За даден билет проверува дали моменталното време е после истекувањето на катанецот.

Функција 5 - Проверува дали дадено столче е пристапно

- Не секое седиште е wheelchair-accessible.

Функција 6 - Верификува тикет

- За даден тикет проверува дали смее да се скенира.

Функција 7 - Проверува за даден тип на тикет уште колку има слободни

- Проверува според total\_allocated

Функција 8 - Број во редицата за чекање на корисникот

- Враќа кој по ред е корисникот во waitlist

## Процедури

### Процедура 1 - Додавање на спонзор на настан

- За даден настан и спонзор, проверува дали и двата ентитети постојат во базата и доколку постојат, го внесува спонзорството во табелите.

### Процедура 2 - Додавање попуст на нарачка

- За дадена нарачка (order\_cart), проверува дали истата е веќе процесирана (платена/откажана) или веќе има искористено друг промо код, и доколку правилата се запазени, ѝ додава попуст.

### Процедура 3 - Префрлање на билет на пријател (како донирање)

- За даден билет, испраќач и примач, проверува дали билетот е валиден и во сопственост на испраќачот, по што креира нова нарачка за примачот и го преврзува билетот на неговиот профил.

### Процедура 4 - Резервирање на билет

- За дадена нарачка, се проверува дали е достапно седиштето и се пополнуваат соодветните табели.

### Процедура 5 - Стари резервации ги прави expired

- Ако резервациите се reserved, ама им поминал lock (на билетот), отпушти ги седиштата што ги држи и направи ја резервацијата/билетот expired.

## Тригери

### Тригер 1 - Валидација на рејтинг за настан

- За дадено review, пред да се зачува или ажурира во базата, проверува дали оценката е помеѓу 1 и 5 и фрла грешка за да спречи внесување на невалидни вредности.

### Тригер 2 - Автоматско логирање на време при скенирање билет

- За даден билет, при промена на статусот во скениран, автоматски го запишува точното време на скенирање (scanned\_at) и не дозволува промена на статусот на билет кој е скениран, да го врати во статус нескениран.

### Тригер 3 - Заштита од overbooking на билети

- При обид за издавање нов билет (пред INSERT), динамички пресметува колку билети се веќе издадени за тој тип и забранува креирање на нов доколку е надминат максималниот лимит на капацитетот.

### Тригер 4 - Дадена нарачка ако стане cancelled, треба да биде испратено refund барање

- Доколку е платена нарачката и е cancelled, треба да се рефундира.

### Тригер 5 - Заштита од overlapping sessions во иста секција/локација

- При обид за додавање на нова сесија, не може веќе да има сесија во таа секција/локација.

### Тригер 6 - Заштита од два refund requests

- При обид за додавање на ново барање за refund, проверува дали веќе корисникот пробал да направи refund.

Код

## Код

### 1. Функции

```
-- Materialized view
-- Average rating of an event
create materialized view mv_event_avg_rating as
select
    event_id,
    avg(star_rating)::numeric(10,2) as avg_rating,
    count(*) as review_count
from review
group by event_id;

select avg_rating
from mv_event_avg_rating
where event_id = 592;
```

```
-- Function 2
-- Check if a user has already reviewed an event
create or replace function fn_has_user_reviewed(p_user_id int, p_event_id int)
returns boolean
language 'plpgsql'
as
$$
begin
    return exists (select 1
                    from review
                    where user_id = p_user_id
                      and event_id = p_event_id);
end;
$$;

select fn_has_user_reviewed(75946, 59240);
```

```
-- Function 3
-- List of sponsors for an event
create or replace function fn_event_sponsors(p_event_id int)
returns table
(
    event_title      varchar,
    sponsor_name     varchar,
    sponsor_type     varchar,
    sponsor_amount_paid numeric
)
language 'plpgsql'
as
$$
begin
    return query
```

```

        select e.title, s.name, st.type, st.sponsor_amount_paid
        from event e
            join sponsor_event se on e.event_id = se.event_id
            join sponsor s on s.sponsor_id = se.sponsor_id
            join sponsor_type_sponsor sts on sts.sponsor_id = s.sponsor_id
            join sponsor_type st on sts.sponsor_type_id =
st.sponsor_type_id
        where e.event_id = p_event_id
        order by st.sponsor_amount_paid desc;
end;
$$;

select *
from fn_event_sponsors(5000);

```

```

-- Function 4
-- Check whether a reserved ticket
-- Useful before payment or confirmation.
create or replace function fn_order_reservation_expired(
    p_ticked_id bigint
)
    returns boolean
    language 'plpgsql'
as
$$
declare
    v_expires_at timestamp;
begin
    select lock_expires_at
    into v_expires_at
    from ticket
    where ticket_id = p_ticked_id;

    return now() > v_expires_at;
end;
$$;

select *
from fn_order_reservation_expired(343);

```

```

-- Function 5
-- Check if a seat is accessible
create or replace function fn_seat_is_accessible(
    p_seat_id int
)
    returns boolean
    language 'plpgsql'
as
$$
begin
    return exists (select 1
                    from seat
                    where seat_id = p_seat_id

```

```

                                and is_accessible = true);
end;
$$;

select *
from fn_seat_is_accessible(1000);

```

```

-- Function 6
-- Verifies a ticket.
-- if it is not reserved/valid, it does not allow to be scanned again.
create or replace function fn_verify_ticket(ticket_barcode_hash varchar)
returns boolean
language 'plpgsql'
as
$$
declare
    check_ticket          boolean;
    check_ticket_status   varchar;
    ticket_scanned        boolean;
begin
    select ticket.status, ticket.is_scanned
    into check_ticket_status, ticket_scanned
    from ticket
    where ticket.barcode_hash = ticket_barcode_hash;

    if check_ticket_status is null
    then
        raise exception 'Ticket is not valid!';
    end if;

    if ticket_scanned = true
    then
        raise exception 'Ticket already scanned, cannot enter with the same
ticket twice!';
    end if;

    if check_ticket_status = 'CANCELLED'
    then
        raise exception 'Ticket is cancelled, cannot be scanned';
    end if;

    if check_ticket_status = 'REFUNDED'
    then
        raise exception 'Ticket has been refunded, cannot enter';
    end if;

    check_ticket = true;

    return check_ticket;
end;
$$;

select *
from fn_verify_ticket('d6e8deb54d4625f007131e9441012b59');

-- Function 7

```



```

-- Checks how many tickets from a type are still available for a given event
-- Used by a customer
create or replace function fn_check_ticket_type_remaining_for_event(event
integer, ticket_type_chosen integer)
    returns int
    language 'plpgsql'
as
$$
declare
    remaining_tickets int;
begin
    select tt.total_allocated - count(t.ticket_id)
    into remaining_tickets
    from ticket_type as tt
        left join ticket t on tt.ticket_type_id = t.ticket_type_id and
t.status in ('VALID', 'RESERVED')
    where tt.event_id = event
        and tt.ticket_type_id = ticket_type_chosen
    group by tt.total_allocated;

    return remaining_tickets;
end;
$$;

select *
from fn_check_ticket_type_remaining_for_event(592, 1467);

-- Function 8
-- Returns a user's entry in the waitlist for a given session
-- Used by a customer
create or replace function fn_check_waitlist_position(p_user_id bigint,
session_id integer)
    returns integer
    language 'plpgsql'
as
$$
declare
    position int;
    user_time timestamp;
begin
    select waitlist_entry.joined_at
    into user_time
    from waitlist_entry
    where user_id = p_user_id
        and session_id = event_schedule_session_id;

    select count(*)
    into position
    from waitlist_entry
    where status = 'ACTIVE'
        and joined_at < user_time;

    return position;
end;
$$;

select *

```

```
from fn_check_waitlist_position(976932, 41112);
```

## 2. Процедури

```
-- Procedure 1
-- For a given event, add a sponsor to a given sponsor type
create or replace procedure pr_add_event_sponsor(
    p_event_id int,
    p_sponsor_id int,
    p_sponsor_type_id int
)
    language 'plpgsql'
as
$$
begin
    if not exists (select 1
                   from event
                   where event_id = p_event_id) then
        raise exception 'Nastanot ne postoi';
    end if;

    if not exists (select 1
                   from sponsor
                   where sponsor_id = p_sponsor_id) then
        raise exception 'Sponzorot ne postoi';
    end if;

    if not exists (select 1
                   from sponsor_type
                   where sponsor_type_id = p_sponsor_type_id) then
        raise exception 'Tipot na sponzorstvo ne postoi';
    end if;

    insert into sponsor_event(event_id, sponsor_id)
    values (p_event_id, p_sponsor_id);

    insert into sponsor_type_sponsor(sponsor_id, sponsor_type_id)
    values (p_sponsor_id, p_sponsor_type_id);

    raise notice 'Uspesno dodaden sponzor na nastan so tip na sponzorstvo';
end;
$$;

call pr_add_event_sponsor(1, 4, 1);
select *
from sponsor_event
where event_id = 1;

-- Procedure 2
-- Add a discount to an order
create or replace procedure pr_apply_discount_to_order(p_order_id bigint,
p_discount_id int)
    language 'plpgsql'
as
$$
begin
```

```

        if exists(select 1
                    from order_cart
                        join order_status os on order_cart.status_id =
os.status_id
                    where order_id = p_order_id
                        and status_name in ('Paid', 'Partially Refunded', 'Refunded',
'Expired', 'Cancelled')) then
            raise exception 'Narackata e procesirana, ne moze da se dodeli popust na
nea';
        end if;
        if exists(select 1
                    from order_cart
                    where order_id = p_order_id
                        and discount_id is not null) then
            raise exception 'Vekje e dodaden popust';
        end if;

        update order_cart set discount_id = p_discount_id where order_id =
p_order_id;

        raise notice 'Uspesno dodaden popust';
end;
$$;

call pr_apply_discount_to_order(5398, 2);
select *
from order_cart oc
        join order_status os on oc.status_id = os.status_id
where oc.order_id = 5398
    and os.status_name in ('Open', 'Pending Payment');

select *
from order_cart oc
        join order_status os on oc.status_id = os.status_id
where oc.order_id = 5398
    and os.status_name in ('Open', 'Pending Payment');

```

```

-- Procedure 3
-- Transfers a ticket to a friend
create or replace procedure pr_transfer_ticket_to_friend(
    p_ticket_id bigint,
    p_from_user_id bigint,
    p_to_user_id bigint
)
    language 'plpgsql'
as
$$
declare
    v_ticket_status    varchar;
    v_current_owner_id int;
    v_new_order_id     bigint;
    v_discount_id      int;
begin
    if not exists
        (select 1 from user_app where user_id = p_to_user_id) then

```

```

        raise exception 'Корисникот на кој сакате да му го пратите билетот не
постои!';
    end if;

    select t.status, oc.user_id, oc.discount_id
    into v_ticket_status, v_current_owner_id, v_discount_id
    from ticket t
        join order_cart oc on t.order_id = oc.order_id
    where t.ticket_id = p_ticket_id;

    if v_ticket_status is null then
        raise exception 'Билетот не постои во системот!';
    end if;

    if v_current_owner_id != p_from_user_id then
        raise exception 'Не можете да префрлите билет кој не е ваша
сопственост!';
    end if;

    if v_ticket_status != 'VALID' then
        raise exception 'Можат да се префрлаат само валидни билети!';
    end if;

    insert into order_cart (user_id, discount_id, status_id, created_at,
total_price)
    values (p_to_user_id, v_discount_id, 3, now(), 0)
    returning order_id into v_new_order_id;

    update ticket
    set order_id = v_new_order_id
    where ticket_id = p_ticket_id;

    raise notice 'Билетот % е успешно префрлен од корисник % на корисник %',
p_ticket_id, p_from_user_id, p_to_user_id;
end;
$$;

call pr_transfer_ticket_to_friend(59, 7, 8);
select *
from ticket t
        join order_cart oc on t.order_id = oc.order_id
where t.ticket_id = 59;

```

```

-- Procedure 4
-- Create order with ticket type
-- Creates a new order and one ticket for a user, but first checks if the seat
is available.
create or replace procedure pr_create_order_with_ticket(
    p_user_id bigint,
    p_ticket_type_id int,
    p_seat_id int default null
)
    language 'plpgsql'

```

```

as
$$
declare
    v_order_id          bigint;
    v_price              numeric(12, 2);
    v_pending_status_id int;
    v_updated_seat_id   int;
begin
    select status_id
    into v_pending_status_id
    from order_status
    where status_name = 'Pending Payment';

    if v_pending_status_id is null then
        raise exception 'Order status Pending Payment not found';
    end if;

    select price
    into v_price
    from price_tier
    where ticket_type_id = p_ticket_type_id
    order by price
    limit 1;

    if v_price is null then
        raise exception 'No price found for ticket type %', p_ticket_type_id;
    end if;

    if p_seat_id is not null then
        update seat
        set is_available = false
        where seat_id = p_seat_id
        and is_available = true
        returning seat_id into v_updated_seat_id;

        if v_updated_seat_id is null then
            raise exception 'Seat % does not exist or is already unavailable',
p_seat_id;
        end if;
    end if;

    insert into order_cart(user_id,
                           status_id,
                           created_at,
                           total_price)
    values (p_user_id,
            v_pending_status_id,
            now(),
            v_price)

```

```

    returning order_id into v_order_id;

    insert into ticket(order_id,
                      ticket_type_id,
                      lock_expires_at,
                      status,
                      barcode_hash,
                      seat_id,
                      is_scanned,
                      scanned_at,
                      is_presale)
    values (v_order_id,
           p_ticket_type_id,
           now() + interval '15 minutes',
           'RESERVED',
           md5('ticket-' || v_order_id || '-' || now()::text),
           p_seat_id,
           false,
           null,
           false);
end;
$$;

call pr_create_order_with_ticket(7, 1467, 600530);
select *
from user_app
      join order_cart oc on user_app.user_id = oc.user_id
where oc.user_id = 7;

```

```

-- Procedure 5
-- Expire old reservations
-- This is useful for cleanup.
create or replace procedure pr_expire_old_reservations(
    p_ticket_id bigint
)
    language 'plpgsql'
as
$$
declare
    v_expired_status_id integer;
begin
    select status_id
    into v_expired_status_id
    from order_status
    where status_name = 'Expired';

    if v_expired_status_id is null then
        raise exception 'Order status Expired not found';
    end if;
end;

```

```

end if;

update seat s
set is_available = true
where s.seat_id in (select t.seat_id
                    from ticket t
                    where t.status = 'RESERVED'
                       and t.lock_expires_at < now()
                       and t.seat_id is not null
                       and t.ticket_id = p_ticket_id);

update ticket
set status = 'EXPIRED'
where status = 'RESERVED'
   and lock_expires_at < now()
   and ticket_id = p_ticket_id;

update order_cart oc
set status_id = v_expired_status_id
where exists (select 1
              from ticket t
              where t.order_id = oc.order_id
                 and t.ticket_id = p_ticket_id
                 and t.status = 'EXPIRED');
end;
$$;

call pr_expire_old_reservations(55);
select ticket_id, status
from ticket
where ticket_id = 55;

```

### 3. Тригери

```

-- Trigger 1
-- Check if review rating is valid
create or replace function fn_check_review_rating()
returns trigger
language 'plpgsql'

```

```

as
$$
begin
    if new.star_rating < 1 or new.star_rating > 5 then
        raise exception 'Rejtingot treba da e od 1-5';
    end if;
    return new;
end;
$$;

create or replace trigger trg_check_review_rating
    before insert or update
    on review
    for each row
execute function fn_check_review_rating();

insert into review (user_id, event_id, star_rating, review_text, created_at)
values (1, 1, -1, 'This is a test review', now());

```

```

-- Trigger 2
-- Set scanned_at when ticket is scanned
create or replace function fn_set_ticket_scanned_at()
    returns trigger
    language 'plpgsql'
as
$$
begin
    if old.is_scanned = false and new.is_scanned = true then
        new.scanned_at := now();
    end if;

    if old.is_scanned = true and new.is_scanned = false then
        raise exception 'Ne moze da se odskenira tiket';
    end if;
    return new;
end;
$$;

```

```

create or replace trigger trg_set_ticket_scanned_at
    before update
    on ticket
    for each row
execute function fn_set_ticket_scanned_at();

update ticket
set is_scanned = false
where ticket_id = 9112018;

```



```
select ticket_id, is_scanned, scanned_at
from ticket
where ticket_id = 9112018;
```

```
-- Trigger 3
-- Prevent ticket overbooking
create or replace function fn_prevent_ticket_overbooking()
    returns trigger
    language 'plpgsql'
as
$$
declare
    v_total_allocated int;
    v_current_sold    int;
begin
    select total_allocated
    into v_total_allocated
    from ticket_type
    where ticket_type_id = new.ticket_type_id;

    select count(*)
    into v_current_sold
    from ticket
    where ticket_type_id = new.ticket_type_id;

    if v_current_sold >= v_total_allocated then
        raise exception 'Ne moze da se izdade biletot, kapacitetot za ovoj tip e
poln';
    end if;

    return new;
end;
$$;

create or replace trigger trg_trigger_prevent_overbooking
    before insert
    on ticket
    for each row
execute function fn_prevent_ticket_overbooking();

select *
from event e
    join ticket_type t on t.event_id = e.event_id
where e.event_id = 180;
```

```
-- Trigger 4
-- When an order status changes to Cancelled
```

```

-- automatically insert refund requests for paid payments
create or replace function fn_create_refund_on_cancel()
    returns trigger
    language 'plpgsql'
as
$$
declare
    v_cancelled_status_id bigint;
begin
    select status_id
    into v_cancelled_status_id
    from order_status
    where status_name = 'Cancelled';

    if new.status_id = v_cancelled_status_id
        and old.status_id != v_cancelled_status_id then

        insert into refund_request(reason, requested_at, user_id, payment_id)
        select 'Automatic refund due to cancellation',
            now(),
            oc.user_id,
            p.payment_id
        from payment p
            join order_cart oc on oc.order_id = p.order_id
        where p.order_id = new.order_id;
    end if;

    return new;
end;
$$;

create trigger trg_create_refund_on_cancel
    after update
    on order_cart
    for each row
execute function fn_create_refund_on_cancel();

select *
from order_cart oc
    join order_status os on oc.status_id = os.status_id
where status_name = 'Pending Payment'
limit 10;

update order_cart oc
set status_id = 7
where oc.order_id = 72;

select *
from refund_request

```

```
where user_id = 72;
```

```
-- Trigger 5
-- Prevent overlapping sessions in same location/section
-- same section cannot host two sessions at overlapping times
create or replace function fn_prevent_session_overlap()
    returns trigger
    language 'plpgsql'
as
$$
begin
    if exists (select 1
                from event_schedule_session ess
                where ess.section_id = new.section_id
                  and ess.event_id = new.event_id
                  and ess.schedule_id != new.schedule_id
                  and (
                      ess.start_time < new.end_time and ess.end_time >
new.start_time
                  )) then
        raise exception 'Another session already exists in this section during
this time period';
    end if;
end;
$$;

create trigger trg_prevent_session_overlap
    before insert or update
    on event_schedule_session
    for each row
execute function fn_prevent_session_overlap();

select *
from event_schedule_session
limit 10;

insert into event_schedule_session (event_id, session_title, start_time,
end_time, section_id)
values (1, 'Test Session', '2024-12-22 23:00:01', '2024-12-22 23:00:02', 1);
```

```
-- Trigger 6
-- Checks if a user has already submitted a refund request for the same payment
create or replace function fn_check_refund_eligibility()
    returns trigger
    language 'plpgsql'
as
```

```

$$
declare
    processed_at_time timestamp;
    refund_check      boolean;
begin

    select p.payment_id
    into refund_check
    from payment p
        join refund_request rr on p.payment_id = rr.payment_id
    where p.payment_id = new.payment_id;

    if refund_check is not null then
        raise exception 'Refund request already placed!';
    end if;

    select payment.processed_at
    into processed_at_time
    from payment
    where payment_id = new.payment_id;

    if now() > (processed_at_time + interval '1 second') then
        raise exception 'Refund request can only be placed before window of 30
days';
    end if;
end;
$$;

create trigger trg_check_refund_eligibility
    before insert or update
    on refund_request
    for each row
execute function fn_check_refund_eligibility();

select *
from payment
limit 10;

insert into refund_request(reason, requested_at, accepted_at, user_id,
payment_id)
values ('test', now(), null, 1, 5)

```