

1. Function check seat availability

Оваа функција проверува дали одредено седиште за конкретен настан е достапно за резервација или купување. Најпрво го пребарува статусот на билетот во табелата ticket, а доколку не постои билет за дадениот настан и седиште, фрла исклучок (TicketNotFoundException). Функцијата враќа TRUE ако билетот е со статус AVAILABLE или CANCELLED, а во сите останати случаи враќа FALSE.

```
CREATE OR REPLACE FUNCTION fn_check_seat_availability(p_event_id BIGINT, p_seat_id
BIGINT)
RETURNS BOOLEAN AS $$
DECLARE
    v_status VARCHAR;
BEGIN
    SELECT status INTO v_status
    FROM ticket
    WHERE event_id = p_event_id AND seat_id = p_seat_id;

    IF v_status IS NULL THEN
        RAISE EXCEPTION 'TicketNotFoundException: Билет за настан % и седиште % не
постои.', p_event_id, p_seat_id;
    END IF;

    IF v_status IN ('AVAILABLE', 'CANCELLED') THEN
        RETURN TRUE;
    ELSE
        RETURN FALSE;
    END IF;
END;
$$ LANGUAGE plpgsql;

select fn_check_seat_availability(1197,2261001);
```

2. Function recommend upcoming events

Оваа функција препорачува идни настани за корисникот врз основа на категоријата на последната купена карта. Најпрво ја пронаоѓа категоријата на последниот настан што корисникот го посетил, а потоа ги враќа следните 3 претстојни настани од истата категорија кои сè уште не се купени од корисникот. Резултатот содржи име на настан, датум на започнување, име на локација и категорија на настанот.

```
CREATE OR REPLACE FUNCTION fn_recommend_upcoming_events(p_customer_id BIGINT)
RETURNS TABLE (
    recommended_event_name VARCHAR,
    start_date TIMESTAMP,
    venue_name VARCHAR,
    category_name VARCHAR
) AS $$
DECLARE
    v_last_category_id BIGINT;
BEGIN
    SELECT e.category_id INTO v_last_category_id
    FROM ticket t
```

```

JOIN event e ON t.event_id = e.event_id
WHERE t.customer_id = p_customer_id
ORDER BY t.purchased_at DESC
LIMIT 1;

RETURN QUERY
SELECT
    e.name,
    e.start_datetime,
    v.venue_title,
    cat.category_name
FROM event e
JOIN venue v ON e.venue_id = v.venue_id
JOIN category cat ON e.category_id = cat.category_id
WHERE e.category_id = v_last_category_id
    AND e.status = 'PUBLISHED'
    AND e.start_datetime > CURRENT_TIMESTAMP
    AND e.event_id NOT IN (
        SELECT t.event_id FROM ticket t WHERE t.customer_id = p_customer_id
    )
ORDER BY e.start_datetime ASC
LIMIT 3;
END;
$$ LANGUAGE plpgsql;

SELECT * FROM fn_recommend_upcoming_events(5);

```

3. Function get event countdown

Оваа функција пресметува колку време останува до почетокот на одреден настан. Таа го споредува датумот на започнување на настанот со тековното време и враќа текстуална порака со преостанатото време во денови, часови и минути. Доколку настанот е веќе започнат, функцијата враќа порака дека е „ВО ТЕК“, а ако поминале повеќе од 3 часа од почетокот, враќа дека настанот е „ЗАВРШЕНО“.

```

CREATE OR REPLACE FUNCTION fn_get_event_countdown(p_event_id BIGINT)
RETURNS TEXT AS $$
DECLARE
    v_start TIMESTAMP;
    v_diff INTERVAL;
BEGIN
    SELECT start_datetime INTO v_start FROM event WHERE event_id = p_event_id;

    v_diff := v_start - CURRENT_TIMESTAMP;

    IF v_diff < INTERVAL '0 seconds' AND v_diff > INTERVAL '-3 hours' THEN
        RETURN 'НАСТАНОТ Е ВО ТЕК!';
    ELSIF v_diff <= INTERVAL '-3 hours' THEN
        RETURN 'ЗАВРШЕНО';
    ELSE
        RETURN 'ПОЧНУВА ЗА: ' ||
            EXTRACT(DAY FROM v_diff) || ' дена, ' ||
            EXTRACT(HOUR FROM v_diff) || ' часа и ' ||
            EXTRACT(MINUTE FROM v_diff) || ' минути.';
    END IF;
END;

```

```

    END IF;
END;
$$ LANGUAGE plpgsql;

SELECT fn_get_event_countdown(3);

```

4. Function add review

Оваа функција овозможува додавање рецензија за одреден настан од страна на корисник. Најпрво проверува дали настанот е завршен (COMPLETED) и дали корисникот навистина купил билет за тој настан. Доколку условите се исполнети, функцијата ја внесува рецензијата во табелата review и го враќа идентификаторот (review_id) на новокреираната рецензија.

```

CREATE OR REPLACE FUNCTION fn_add_review(
p_customer_id BIGINT,
p_event_id BIGINT,
p_rating SMALLINT,
p_comment VARCHAR(255)
) RETURNS BIGINT AS $$
DECLARE
v_review_id BIGINT;
v_event_status VARCHAR(50);
v_has_purchased BOOLEAN;
BEGIN

SELECT status INTO v_event_status
FROM event
WHERE event_id = p_event_id;

IF v_event_status != 'COMPLETED' THEN
RAISE EXCEPTION 'Reviews can only be submitted for completed events.';
END IF;

SELECT EXISTS (
SELECT 1 FROM ticket
WHERE customer_id = p_customer_id
AND event_id = p_event_id
AND status IN ('PURCHASED', 'SCANNED')
) INTO v_has_purchased;

IF NOT v_has_purchased THEN
RAISE EXCEPTION 'Only customers who purchased a ticket can leave a review.';
END IF;

INSERT INTO review (comment, rating, customer_id, event_id)
VALUES (p_comment, p_rating, p_customer_id, p_event_id)
RETURNING review_id INTO v_review_id;

RETURN v_review_id;
END;
$$ LANGUAGE plpgsql;

```

5. Procedure buy ticket

Оваа процедура овозможува купување билет од страна на корисник. Најпрво проверува дали избраниот билет е достапен (AVAILABLE). Доколку билетот е веќе купен или резервиран, фрла исклучок и процесот се прекинува. Потоа креира запис за плаќање во табелата payment, го ажурира билетот со податоци за корисникот, плаќањето и QR кодот, а статусот на билетот го поставува на PURCHASED. На крај го ажурира и записот во seat_reservation, ги зачувува промените во базата и прикажува порака за успешно купување на билетот.

```
CREATE OR REPLACE PROCEDURE pr_buy_ticket(  
  p_customer_id BIGINT,  
  p_ticket_id BIGINT,  
  p_payment_method VARCHAR(50),  
  p_amount NUMERIC(12,2)  
)  
LANGUAGE plpgsql  
AS $$  
DECLARE  
  v_current_status VARCHAR(50);  
  v_payment_id BIGINT;  
BEGIN  
  -- 1. Check if the ticket is available  
  SELECT status INTO v_current_status  
  FROM ticket  
  WHERE ticket_id = p_ticket_id;  
  
  IF v_current_status != 'AVAILABLE' THEN  
    RAISE EXCEPTION 'This ticket is already taken or unavailable.';  
  END IF;  
  
  -- 2. Create the payment record  
  INSERT INTO payment (amount, payment_date, payment_status, payment_method)  
  VALUES (p_amount, CURRENT_TIMESTAMP, 'COMPLETED', p_payment_method)  
  RETURNING payment_id INTO v_payment_id;  
  
  -- 3. Update the ticket with customer and payment info  
  UPDATE ticket  
  SET  
    customer_id = p_customer_id,  
    payment_id = v_payment_id,  
    status = 'PURCHASED',  
    purchased_at = CURRENT_TIMESTAMP;
```

```

        qr_code = 'QR-' || p_ticket_id || '-' || p_customer_id -- Generating a
dummy QR
    WHERE ticket_id = p_ticket_id;

-- 4. Update the seat reservation status
    UPDATE seat_reservation
    SET status = 'ACTIVE',
        reserved_at = CURRENT_TIMESTAMP
    WHERE ticket_id = p_ticket_id;

    COMMIT;
    RAISE NOTICE 'Ticket % successfully purchased by Customer %', p_ticket_id,
p_customer_id;
END;
$$;

```

6. Function create event

Оваа функција овозможува креирање нов настан. Најпрво проверува дали датумот на започнување е пред датумот на завршување. Доколку условот не е исполнет, фрла исклучок. Потоа внесува нов запис во табелата event со дадените податоци и го враќа идентификаторот (event_id) на новокреираниот настан.

```

CREATE OR REPLACE FUNCTION fn_create_event(
    p_name VARCHAR(255),
    p_description TEXT,
    p_start TIMESTAMP,
    p_end TIMESTAMP,
    p_status VARCHAR(50),
    p_venue_id BIGINT,
    p_category_id BIGINT
)
RETURNS BIGINT
LANGUAGE plpgsql
AS $$
DECLARE
    v_new_event_id BIGINT;
BEGIN
    -- 1. Validation: Ensure start date is before end date
    IF p_start >= p_end THEN
        RAISE EXCEPTION 'Start time must be before end time.';
    END IF;

```

```

-- 2. Insert the new event
INSERT INTO event (name, description, start_datetime, end_datetime,
status, venue_id, category_id)
VALUES (p_name, p_description, p_start, p_end, p_status, p_venue_id,
p_category_id)
RETURNING event_id INTO v_new_event_id;

-- 3. Return the ID for confirmation
RETURN v_new_event_id;
END;
$$;

```

7. Procedure cancel ticket

Оваа процедура овозможува откажување на претходно купен билет. Најпрво проверува дали билетот му припаѓа на корисникот и дали е со статус PURCHASED. Доколку условите не се исполнети, фрла исклучок. Потоа билетот повторно го прави достапен (AVAILABLE), плаќањето го означува како REFUNDED, а резервацијата на седиштето ја поставува на CANCELLED. На крај прикажува порака за успешно откажување и рефундирање на билетот.

```

CREATE OR REPLACE PROCEDURE pr_cancel_ticket(
    p_ticket_id BIGINT,
    p_customer_id BIGINT
)
LANGUAGE plpgsql
AS $$
DECLARE
    v_payment_id BIGINT;
    v_status VARCHAR(50);
BEGIN
    -- 1. Check if the ticket actually belongs to this customer and is
    purchased
    SELECT status, payment_id INTO v_status, v_payment_id
    FROM ticket
    WHERE ticket_id = p_ticket_id AND customer_id = p_customer_id;

    IF v_status IS NULL THEN
        RAISE EXCEPTION 'Ticket not found for this customer.';
    ELSIF v_status != 'PURCHASED' THEN
        RAISE EXCEPTION 'Only purchased tickets can be cancelled. Current
status: %', v_status;
    END IF;

    -- 2. Update Ticket: Remove customer/payment and set back to AVAILABLE

```

```

UPDATE ticket
SET customer_id = NULL,
    payment_id = NULL,
    status = 'AVAILABLE',
    purchased_at = NULL
WHERE ticket_id = p_ticket_id;

-- 3. Update Payment: Mark as REFUNDED
IF v_payment_id IS NOT NULL THEN
    UPDATE payment
    SET payment_status = 'REFUNDED'
    WHERE payment_id = v_payment_id;
END IF;

-- 4. Update Seat Reservation: Mark as CANCELLED
UPDATE seat_reservation
SET status = 'CANCELLED'
WHERE ticket_id = p_ticket_id AND status = 'ACTIVE';

RAISE NOTICE 'Ticket % successfully cancelled and refunded.',
p_ticket_id;
END;
$$;

```

8. Trigger Generate Tickets for Event

Оваа trigger функција автоматски креира билети при додавање нов настан. За секое седиште во локацијата на настанот внесува нов запис во табелата ticket со статус AVAILABLE. Цената и типот на билетот се одредуваат според секцијата на седиштето (VIP, BALCONY или REGULAR), а за секој билет се генерира и уникатен QR код. На крај ја враќа новододадената вредност (NEW).

```

CREATE OR REPLACE FUNCTION fn_trg_generate_tickets_for_event()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO ticket (status, ticket_price, qr_code, seat_id, event_id,
ticket_type_id)
    SELECT
        'AVAILABLE',
        CASE
            WHEN s.section_type = 'VIP' THEN 1500.00
            WHEN s.section_type = 'BALCONY' THEN 400.00
            ELSE 700.00
        END,
        md5(NEW.event_id::text || s.seat_id::text || random()::text),

```

```

        s.seat_id,
        NEW.event_id,
        (SELECT ticket_type_id FROM ticket_type WHERE type_name =
            CASE
                WHEN s.section_type = 'VIP' THEN 'VIP'
                ELSE 'Standard'
            END LIMIT 1)
    FROM seat s
    WHERE s.venue_id = NEW.venue_id;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_after_event_insert
AFTER INSERT ON event
FOR EACH ROW
EXECUTE FUNCTION fn_trg_generate_tickets_for_event();

```

9. Trigger Update event status by time

Оваа trigger функција автоматски го ажурира статусот на настанот според времето на одржување. Доколку настанот е завршен, а статусот не е COMPLETED или CANCELLED, статусот се поставува на COMPLETED. Доколку настанот е во тек и статусот е PUBLISHED, тој се менува во ONGOING. Trigger-от се активира пред секое внесување или ажурирање на запис во табелата event.

```

CREATE OR REPLACE FUNCTION fn_trg_update_event_status_by_time()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.end_datetime < CURRENT_TIMESTAMP AND NEW.status NOT IN
('COMPLETED', 'CANCELLED') THEN
        NEW.status := 'COMPLETED';
    ELIF NEW.start_datetime <= CURRENT_TIMESTAMP AND NEW.end_datetime >=
CURRENT_TIMESTAMP AND NEW.status = 'PUBLISHED' THEN
        NEW.status := 'ONGOING';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_before_event_status_check
BEFORE INSERT OR UPDATE ON event
FOR EACH ROW
EXECUTE FUNCTION fn_trg_update_event_status_by_time();

```


10. Trigger Sync Reservation on cancel

Оваа trigger функција автоматски ја синхронизира резервацијата при откажување на билет. Доколку статусот на билетот се промени од PURCHASED во AVAILABLE, во табелата seat_reservation се додава нов запис со статус CANCELLED. Trigger-от се активира по секое ажурирање на табелата ticket и обезбедува евиденција за откажаните резервации

```
CREATE OR REPLACE FUNCTION fn_trg_sync_reservation_on_cancel()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.status = 'AVAILABLE' AND OLD.status = 'PURCHASED' THEN
        INSERT INTO seat_reservation (reserved_at, status, seat_id,
ticket_id)
            VALUES (CURRENT_TIMESTAMP, 'CANCELLED', OLD.seat_id, OLD.ticket_id);
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_after_ticket_cancel_sync
AFTER UPDATE ON ticket
FOR EACH ROW
EXECUTE FUNCTION fn_trg_sync_reservation_on_cancel();
```

11. Trigger Event Broadcast Notification

Оваа trigger функција автоматски испраќа известувања до корисниците при промени на настаните. При креирање нов настан со статус PUBLISHED, се генерира известување за сите корисници дека е објавен нов настан. Доколку статусот на постоечки настан се промени во CANCELLED, се испраќа известување само до корисниците кои имаат купено билет за тој настан. Trigger-от се активира по секое внесување или ажурирање на табелата event.

```
CREATE OR REPLACE FUNCTION fn_trg_event_broadcast_notifications()
RETURNS TRIGGER AS $$
BEGIN
    IF (TG_OP = 'INSERT' AND NEW.status = 'PUBLISHED') THEN
        INSERT INTO notification (title, message, created_at, is_read,
customer_id, event_id)
            SELECT
                'New Event Alert: ' || NEW.name,
                'Hey ' || c.first_name || ', a new event has been published!
Check it out.',
                CURRENT_TIMESTAMP,
                FALSE,
```

```

        c.customer_id,
        NEW.event_id
    FROM customer c;

    ELSIF (TG_OP = 'UPDATE' AND NEW.status = 'CANCELLED' AND OLD.status !=
'CANCELLED') THEN
        INSERT INTO notification (title, message, created_at, is_read,
customer_id, event_id)
        SELECT
            'IMPORTANT: Event Cancelled - ' || NEW.name,
            'Dear ' || c.first_name || ', we regret to inform you that the
event is cancelled. Your refund is in progress.',
            CURRENT_TIMESTAMP,
            FALSE,
            t.customer_id,
            NEW.event_id
        FROM ticket t
        JOIN customer c ON t.customer_id = c.customer_id
        WHERE t.event_id = NEW.event_id AND t.customer_id IS NOT NULL;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_event_notification_broadcast
AFTER INSERT OR UPDATE ON event
FOR EACH ROW
EXECUTE FUNCTION fn_trg_event_broadcast_notifications();

```

12. Trigger Prevent Future Event Overlap

Оваа trigger функција спречува закажување на два настани во ист термин на иста локација. Пред внесување или ажурирање на настан, проверува дали веќе постои друг активен настан во истата сала чиј временски период се преклопува со новиот настан. Доколку се пронајде преклопување, фрла исклучок и не дозволува зачувување на записот. Trigger-от се активира пред секое внесување или ажурирање во табелата event.

```

CREATE OR REPLACE FUNCTION fn_trg_prevent_future_event_overlap()
RETURNS TRIGGER AS $$
BEGIN
    IF EXISTS (
        SELECT 1
        FROM event
        WHERE venue_id = NEW.venue_id
            AND event_id != NEW.event_id
    )

```

```
        AND status != 'CANCELLED'
        AND (NEW.start_datetime, NEW.end_datetime) OVERLAPS
(start_datetime, end_datetime)
    ) THEN
        RAISE EXCEPTION 'ГРЕШКА: Салата (ID: %) е веќе зафатена за друг
настан во овој период (%)! ',
            NEW.venue_id, NEW.start_datetime;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_before_event_insert_update_overlap
BEFORE INSERT OR UPDATE ON event
FOR EACH ROW
EXECUTE FUNCTION fn_trg_prevent_future_event_overlap();
```