Advanced Databases

Phase 4- Functions, Procedures and Triggers Documentation

Project: Balkan geotourism

Bora Alili 231504

Esra Bekiri 231521

Marko Smilevski 231557

Fisnik Mamuti 231508

FUNCTIONS

1. **Function: get_total_participant_cost(p_participant_id)**

```sql
CREATE OR REPLACE FUNCTION get_total_participant_cost(
    p_participant_id INT
)
    RETURNS DECIMAL(12, 2)
AS
$$
DECLARE
    total_flight       DECIMAL(12, 2) := 0;
    total_bus          DECIMAL(12, 2) := 0;
    total_hotel        DECIMAL(12, 2) := 0;
    total_cost         DECIMAL(12, 2);
    participant_exists INT;
    participant_status VARCHAR(20);
BEGIN

    -- Check if participant exists
    SELECT COUNT(*), MAX(status)
    INTO participant_exists, participant_status
    FROM trip_participants
    WHERE id = p_participant_id;

    IF participant_exists = 0 THEN
        RAISE EXCEPTION
            'Participant with ID % does not exist',
            p_participant_id;
    END IF;

    -- Prevent calculating costs for cancelled participants
    IF participant_status = 'cancelled' THEN
        RAISE EXCEPTION
            'Cannot calculate cost for cancelled participant %',
            p_participant_id;
    END IF;
```

```sql
    -- Flight booking total
    SELECT COALESCE(SUM(price), 0)
    INTO total_flight
    FROM flight_bookings
    WHERE trip_participant_id = p_participant_id
      AND price >= 0;

    -- Bus booking total
    SELECT COALESCE(SUM(price), 0)
    INTO total_bus
    FROM bus_bookings
    WHERE trip_participant_id = p_participant_id
      AND price >= 0;

    -- Hotel booking total
    SELECT COALESCE(SUM(hr.price), 0)
    INTO total_hotel
    FROM trip_participants_hotel_rooms tphr
             JOIN hotel_rooms hr
                  1..n<->1: ON hr.id = tphr.hotel_room_id
    WHERE tphr.trip_participant_id = p_participant_id
      AND hr.price >= 0;

    total_cost :=
            total_flight +
            total_bus +
            total_hotel;

    RETURN ROUND(total_cost, 2);

END;
```

Description:
This function calculates the total amount spent by a participant by summing the costs of flight bookings, bus bookings, and hotel room reservations. Before performing the calculation, it validates that the participant exists and has not been cancelled.

Business Logic / Use:
This function implements the cost calculation logic of the travel management system. It can be used to generate participant invoices, payment summaries, and financial reports while ensuring that only valid participant data is processed.

2. **Function: get_trip_execution_score(p_trip_execution_id)**

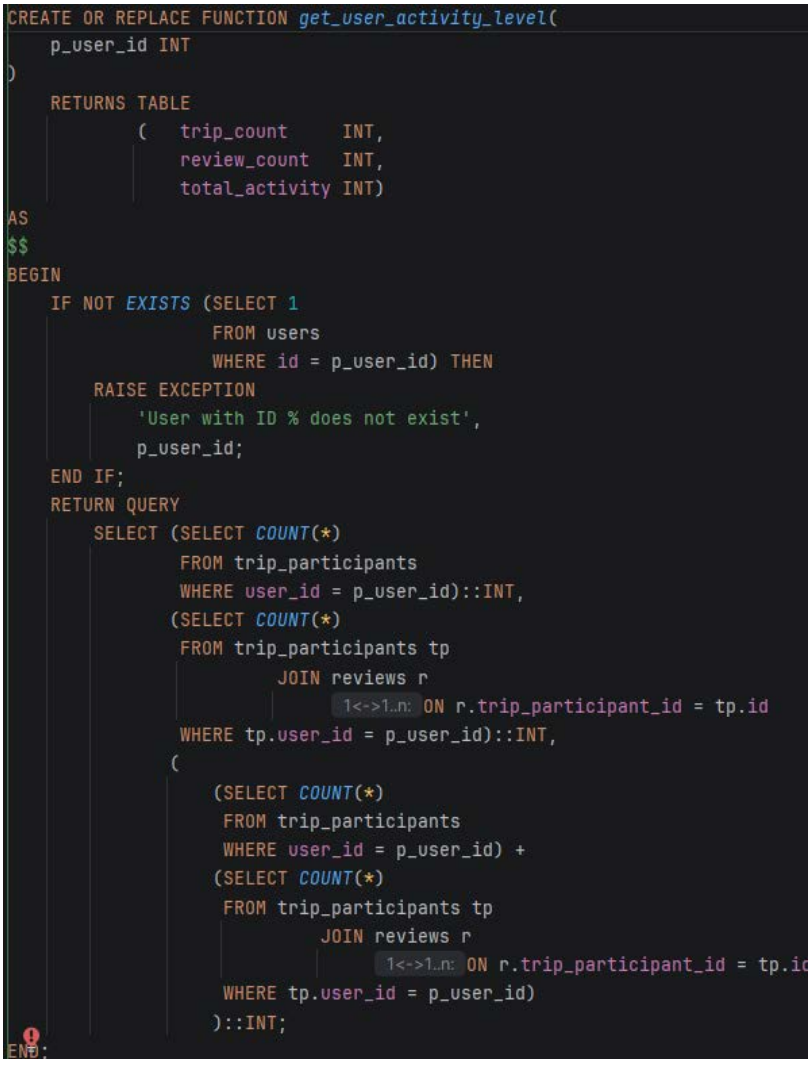
```sql
CREATE OR REPLACE FUNCTION get_trip_execution_score(
    p_trip_execution_id INT
)
    RETURNS NUMERIC
AS
$$
DECLARE
    participant_count INT;
    location_count    INT;
    avg_rating        NUMERIC;
    score             NUMERIC;
BEGIN
    -- Validate trip execution existence
    IF NOT EXISTS (SELECT 1
                   FROM trip_executions
                   WHERE id = p_trip_execution_id) THEN
        RAISE EXCEPTION
            'Trip execution with ID % does not exist',
            p_trip_execution_id;
    END IF;
    SELECT tev.participant_count,
           tev.visited_locations,
           tev.average_rating
    INTO
        participant_count,
        location_count,
        avg_rating
    FROM trip_execution_stats_view tev
    WHERE tev.trip_execution_id = p_trip_execution_id;
    score :=
            COALESCE(participant_count, 0) * 2 +
            COALESCE(location_count, 0) * 1.5 +
            COALESCE(avg_rating, 0) * 10;
    RETURN ROUND(score, 2);
END;
```

Description:
This function calculates an execution score for a trip based on the number of participants,

visited locations, and the average review rating. It verifies that the specified trip execution exists before calculating the score.

Business Logic / Use:
The function implements trip performance evaluation by combining multiple performance indicators into a single score. This score can be used to compare trips, analyze their success, and support decision-making.

3. **Function: get_user_activity_level(p_user_id)**



```
CREATE OR REPLACE FUNCTION get_user_activity_level(
    p_user_id INT
)
    RETURNS TABLE
            (   trip_count     INT,
                review_count   INT,
                total_activity INT)
AS
$$
BEGIN
    IF NOT EXISTS (SELECT 1
                   FROM users
                   WHERE id = p_user_id) THEN
        RAISE EXCEPTION
            'User with ID % does not exist',
            p_user_id;
    END IF;
    RETURN QUERY
        SELECT (SELECT COUNT(*)
                FROM trip_participants
                WHERE user_id = p_user_id)::INT,
               (SELECT COUNT(*)
                FROM trip_participants tp
                         JOIN reviews r
                              1<->1..n: ON r.trip_participant_id = tp.id
                WHERE tp.user_id = p_user_id)::INT,
               (
                   (SELECT COUNT(*)
                    FROM trip_participants
                    WHERE user_id = p_user_id) +
                   (SELECT COUNT(*)
                    FROM trip_participants tp
                             JOIN reviews r
                                  1<->1..n: ON r.trip_participant_id = tp.id
                    WHERE tp.user_id = p_user_id)
                   )::INT;
END;
```

Description:
This function returns a table containing the total number of trips, reviews, and overall activity for a specific user. It validates that the user exists before retrieving the information.

Business Logic / Use:
The function implements user activity analysis by measuring participation and engagement

within the application. The returned information can support loyalty programs, user statistics, and activity reporting.

4. **Function: trip_duration_days(p_trip_execution_id)**

```sql
CREATE OR REPLACE FUNCTION trip_duration_days(
    p_trip_execution_id INT
)
    RETURNS INT
AS
$$
DECLARE
    days INT;
BEGIN

    -- Validate trip existence
    IF NOT EXISTS (SELECT 1
                   FROM trip_executions
                   WHERE id = p_trip_execution_id) THEN
        RAISE EXCEPTION
            'Trip execution with ID % does not exist',
            p_trip_execution_id;
    END IF;

    SELECT (end_date - start_date)
    INTO days
    FROM trip_executions
    WHERE id = p_trip_execution_id;

    IF days < 0 THEN
        RAISE EXCEPTION
            'Invalid trip dates';
    END IF;

    RETURN COALESCE(days, 0);

END;
```

Description:
This function calculates the duration of a trip by subtracting the start date from the end date. It validates both the existence of the trip execution and the correctness of the travel dates.

Business Logic / Use:
The function implements trip duration calculation, which can be used for scheduling, pricing, reporting, and itinerary planning while ensuring valid trip data.

PROCEDURES

1. **Procedure: confirm_trip_participants()**

```
CREATE OR REPLACE PROCEDURE confirm_trip_participants()
AS
$$
BEGIN

    UPDATE trip_participants
    SET status = 'confirmed'
    WHERE status = 'pending'

      -- Passport must exist
      AND passport_number IS NOT NULL
      -- Remove empty spaces
      AND LENGTH(TRIM(passport_number)) BETWEEN 6 AND 12
      -- Prevent fake placeholder values
      AND UPPER(TRIM(passport_number)) NOT IN (
                            'N/A', 'UNKNOWN', 'NONE', '-'
        )
      -- Passport must contain only letters and numbers
      AND passport_number ~ '^[A-Za-z0-9]+$'
      -- Passport must contain at least 2 numbers
      AND passport_number ~ '[0-9].*[0-9]'
      -- Passport must contain at least 1 letter
      AND passport_number ~ '[A-Za-z]';
    -- Notify if no rows updated
    IF NOT FOUND THEN
        RAISE NOTICE
            'No participants eligible for confirmation';
    END IF;
END;
```

Description:

This procedure automatically changes the status of participants from **pending** to **confirmed** after validating their passport information. Only participants with correctly formatted and complete passport numbers are confirmed.

Business Logic / Use:

This procedure automates the participant confirmation process and ensures that only eligible travelers are approved for trips. It reduces manual verification while maintaining accurate participant records.

2. **Procedure: update_user_role(p_user_id, p_role_id)**

```sql
CREATE OR REPLACE PROCEDURE update_user_role(
    p_user_id INT,
    p_role_id INT DEFAULT NULL
)
AS
$$
DECLARE
    trip_count  INT;
    user_exists INT;
BEGIN

    -- Validate user existence
    SELECT COUNT(*)
    INTO user_exists
    FROM users
    WHERE id = p_user_id;

    IF user_exists = 0 THEN
        RAISE EXCEPTION
            'User with ID % does not exist',
            p_user_id;
    END IF;

    -- Count trips
    SELECT COUNT(*)
    INTO trip_count
    FROM trip_participants
    WHERE user_id = p_user_id
      AND status = 'confirmed';

    -- Remove old roles
    DELETE
    FROM user_role_map
    WHERE user_id = p_user_id;

    -- Assign custom role if provided
    IF p_role_id IS NOT NULL THEN

        INSERT INTO user_role_map(user_id, role_id)
        VALUES ( user_id p_user_id,  role_id p_role_id);

    ELSE

        -- Automatic role assignment
        IF trip_count >= 5 THEN

            INSERT INTO user_role_map(user_id, role_id)
            VALUES ( user_id p_user_id,  role_id 2);

        ELSE

            INSERT INTO user_role_map(user_id, role_id)
            VALUES ( user_id p_user_id,  role_id 1);

        END IF;

    END IF;

END;
```

Description:

This procedure updates a user's role either automatically according to the number of confirmed trips or manually when a specific role is provided. Existing role assignments are removed before assigning the new role.

Business Logic / Use:

The procedure implements dynamic role management within the application. It supports automatic user promotion based on activity while allowing administrators to assign custom roles when necessary.

3. **Procedure: assign_hotel_room(p_trip_participant_id INT, p_hotel_room_id INT, p_check_in DATE, p_check_out DATE)**

```
CREATE OR REPLACE PROCEDURE assign_hotel_room(
    p_trip_participant_id INT,
    p_hotel_room_id INT,
    p_check_in DATE,
    p_check_out DATE
)
    LANGUAGE plpgsql
AS
$$
BEGIN

    -- Validate participant + status
    IF NOT EXISTS (SELECT 1
                   FROM trip_participants
                   WHERE id = p_trip_participant_id
                     AND status = 'confirmed') THEN
        RAISE EXCEPTION
            'Participant does not exist or is not confirmed';
    END IF;

    -- Validate room
    IF NOT EXISTS (SELECT 1
                   FROM hotel_rooms
                   WHERE id = p_hotel_room_id) THEN
        RAISE EXCEPTION
            'Hotel room does not exist';
    END IF;
```

```
    -- Prevent duplicate active booking
    IF EXISTS (SELECT 1
               FROM trip_participants_hotel_rooms
               WHERE trip_participant_id = p_trip_participant_id
                 AND hotel_room_id = p_hotel_room_id
                 AND check_out >= CURRENT_DATE) THEN
        RAISE EXCEPTION
            'Participant already has an active booking for this room';
    END IF;

    INSERT INTO trip_participants_hotel_rooms(trip_participant_id,
                                              hotel_room_id,
                                              check_in,
                                              check_out)
    VALUES ( trip_participant_id p_trip_participant_id,
             hotel_room_id p_hotel_room_id,
             check_in p_check_in,
             check_out p_check_out);

END;
$$;
```

Description:

This procedure assigns a hotel room to a confirmed participant after validating the participant, the room, and ensuring that no duplicate active reservation exists. If all validations pass, the reservation is created.

Business Logic / Use:
The procedure centralizes the hotel room assignment process and guarantees that only eligible participants receive room reservations. It also prevents duplicate bookings and maintains reservation consistency.

4. **Procedure: cancel_expired_trip_reservations()**

```
CREATE OR REPLACE PROCEDURE cancel_expired_trip_reservations()
AS
$$
BEGIN

    UPDATE trip_participants tp
    SET status = 'cancelled'
    FROM trip_executions te
    WHERE tp.trip_execution_id = te.id
      AND tp.status = 'pending'

      -- trip already finished (NOT just started)
      AND te.end_date < CURRENT_DATE

      -- invalid passport still blocks confirmation
      AND (
        tp.passport_number IS NULL
            OR LENGTH(TRIM(tp.passport_number)) < 6
        );

    IF NOT FOUND THEN
        RAISE NOTICE
            'No expired reservations found';
    ELSE
        RAISE NOTICE
            'Expired reservations successfully cancelled';
    END IF;

END;
```

Description:
This procedure automatically cancels pending participant reservations for trips that have already ended when the participant still has invalid passport information. It reports whether any reservations were cancelled during execution.

Business Logic / Use:
The procedure implements automatic reservation cleanup by removing outdated pending bookings that can no longer be confirmed. This helps maintain accurate reservation records and reduces unnecessary administrative work.

TRIGGERS

1. **Trigger: trg_check_room_capacity (check_room_capacity())**

```
CREATE OR REPLACE FUNCTION check_room_capacity()
    RETURNS TRIGGER
AS
$$
DECLARE
    room_capacity INT;
    current_count INT;
BEGIN
    -- Validate room exists + capacity
    SELECT capacity
    INTO room_capacity
    FROM hotel_rooms
    WHERE id = NEW.hotel_room_id;
    IF room_capacity IS NULL THEN
        RAISE EXCEPTION
            'Hotel room with ID % does not exist',
            NEW.hotel_room_id;
    END IF;
    IF room_capacity <= 0 THEN
        RAISE EXCEPTION
            'Invalid capacity for room ID %',
            NEW.hotel_room_id;
    END IF;
    SELECT COUNT(*)
    INTO current_count
    FROM trip_participants_hotel_rooms
    WHERE hotel_room_id = NEW.hotel_room_id
      AND NEW.check_in < check_out
      AND NEW.check_out > check_in;
    IF current_count >= room_capacity THEN
        RAISE EXCEPTION
            'Room capacity exceeded for room ID % in selected dates',
            NEW.hotel_room_id;
    END IF;
    RETURN NEW;
END;
```

Description:

This trigger validates hotel room availability before a reservation is inserted by checking the room capacity and counting overlapping reservations. If the room is already full for the selected dates, the insertion is rejected.

Business Logic / Use:

The trigger implements hotel capacity control and prevents room overbooking. It ensures that accommodation reservations always respect the maximum capacity defined for each room.

2. **Trigger: trg_auto_close_trip (auto_close_trip_execution())**

```
CREATE OR REPLACE FUNCTION auto_close_trip_execution()
    RETURNS TRIGGER
AS
$$
BEGIN

    -- Validate dates
    IF NEW.end_date < NEW.start_date THEN
        RAISE EXCEPTION
            'End date cannot be before start date';
    END IF;

    -- Notify if already ended
    IF NEW.end_date < CURRENT_DATE THEN
        RAISE NOTICE
            'Trip execution % has already ended',
            NEW.id;
    END IF;

    RETURN NEW;

END;
$$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS trg_auto_close_trip
    ON trip_executions;

CREATE TRIGGER trg_auto_close_trip
    BEFORE UPDATE
    ON trip_executions
    FOR EACH ROW
EXECUTE FUNCTION auto_close_trip_execution();
```

Description:

This trigger validates trip dates before updating a trip execution. It prevents invalid date ranges and generates a notification when a trip has already ended.

Business Logic / Use:

The trigger enforces scheduling rules by ensuring logical travel dates and assisting administrators in monitoring completed trips.

3. **Trigger: trg_validate_participant_status (validate_participant_status())**

```sql
CREATE OR REPLACE FUNCTION validate_participant_status()
    RETURNS TRIGGER
AS
$$
BEGIN

    IF NEW.status NOT IN (
                          'pending',
                          'confirmed',
                          'cancelled'
        ) THEN
        RAISE EXCEPTION
            'Invalid participant status: %',
            NEW.status;
    END IF;

    -- Prevent changing cancelled participants
    IF OLD.status = 'cancelled'
        AND NEW.status <> 'cancelled' THEN

        RAISE EXCEPTION
            'Cancelled participant status cannot be changed';

    END IF;

    RETURN NEW;

END;
$$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS trg_validate_participant_status
    ON trip_participants;

CREATE TRIGGER trg_validate_participant_status
    BEFORE UPDATE
    ON trip_participants
    FOR EACH ROW
EXECUTE FUNCTION validate_participant_status();
```

Description:

This trigger validates participant status updates by allowing only predefined status values and preventing cancelled participants from becoming active again. Invalid updates are rejected.

Business Logic / Use:

The trigger implements participant lifecycle management by enforcing valid status transitions and protecting the consistency of reservation records.

4. **Trigger: trg_validate_hotel_dates (validate_hotel_dates())**

```sql
CREATE OR REPLACE FUNCTION validate_hotel_dates()
    RETURNS TRIGGER
AS
$$
BEGIN

    -- Check date order
    IF NEW.check_out <= NEW.check_in THEN
        RAISE EXCEPTION
            'Check-out date must be after check-in date';
    END IF;

    -- Prevent past reservations
    IF NEW.check_in < CURRENT_DATE THEN
        RAISE EXCEPTION
            'Check-in date cannot be in the past';
    END IF;

    -- Limit unrealistic long stays
    IF (NEW.check_out - NEW.check_in) > 60 THEN
        RAISE EXCEPTION
            'Hotel stay cannot exceed 60 days';
    END IF;

    RETURN NEW;

END;
$$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS trg_validate_hotel_dates
    ON trip_participants_hotel_rooms;

CREATE TRIGGER trg_validate_hotel_dates
    BEFORE INSERT OR UPDATE
    ON trip_participants_hotel_rooms
    FOR EACH ROW
EXECUTE FUNCTION validate_hotel_dates();
```

Description:

This trigger validates hotel reservation dates before insertions and updates. It checks that the check-out date is later than the check-in date, prevents reservations in the past, and limits the maximum stay to sixty days.

Business Logic / Use:

The trigger enforces hotel booking rules by ensuring that all accommodation reservations contain realistic and valid date ranges. It helps maintain reliable reservation data throughout the application.