

Функции

- **fn_get_team_position** - враќа на која позиција се наоѓа тимот во табелата за дадена сезона

```
CREATE OR REPLACE FUNCTION fn_get_team_position(p_team_id INT, p_season_id INT)
RETURNS INT
LANGUAGE plpgsql
AS $$
DECLARE
    v_position INT;
BEGIN
    SELECT position INTO v_position
    FROM (
        SELECT
            team_id,
            ROW_NUMBER() OVER (
                ORDER BY points DESC, goal_difference DESC, goals_for DESC
            ) AS position
        FROM vw_team_season_table
        WHERE season_id = p_season_id
    ) ranked
    WHERE team_id = p_team_id;

    IF NOT FOUND THEN
        RAISE EXCEPTION 'Team % not found in season %', p_team_id, p_season_id;
    END IF;

    RETURN v_position;
END;
$$;
```

- **fn_player_form** - враќа голови, асистенции и картони за играч во дадена сезона

```
CREATE OR REPLACE FUNCTION fn_player_form(p_player_id INT, p_season_id INT)
RETURNS TABLE(
    goals      INT,
    assists    INT,
    yellow_cards INT,
    red_cards  INT
)
LANGUAGE plpgsql
AS $$
BEGIN
    RETURN QUERY
    SELECT
        pss.goals::INT,
        pss.assists::INT,
        pss.yellow_cards::INT,
        pss.red_cards::INT
    FROM vw_player_season_stats pss
    WHERE pss.player_id = p_player_id
        AND pss.season_id = p_season_id;

    IF NOT FOUND THEN
        RAISE EXCEPTION 'Player % not found in season %', p_player_id, p_season_id;
    END IF;
END;
$$;
```

- **fn_stadium_available_seats** - враќа колку седишта се достапни за даден натпревар

```
CREATE OR REPLACE FUNCTION fn_stadium_available_seats(  
    p_match_id INT  
)  
    RETURNS INT  
    LANGUAGE plpgsql  
AS  
$$  
DECLARE  
    v_capacity      INT;  
    v_tickets_sold  INT;  
BEGIN  
    -- get stadium capacity  
    SELECT s.capacity  
    INTO v_capacity  
    FROM Match m  
        JOIN Stadium s 1..n<->1: ON s.stadium_id = m.stadium_id  
    WHERE m.match_id = p_match_id;  
  
    IF v_capacity IS NULL THEN  
        RAISE EXCEPTION 'Match with id % not found.', p_match_id;  
    END IF;  
  
    -- count tickets already issued  
    SELECT COUNT(*)  
    INTO v_tickets_sold  
    FROM Ticket t  
    WHERE t.match_id = p_match_id;  
  
    RETURN v_capacity - v_tickets_sold;  
END;  
$;
```

- **fn_team_revenue_for_season** - враќа приход од продадени билети и спонзори, како и вкупен приход

```
CREATE OR REPLACE FUNCTION fn_team_revenue_for_season(  
    p_team_id INT,  
    p_season_id INT,  
    OUT out_ticket_revenue NUMERIC,  
    OUT out_sponsor_revenue NUMERIC,  
    OUT out_total_revenue NUMERIC  
)  
LANGUAGE plpgsql  
AS $$  
BEGIN  
    -- ticket revenue  
    SELECT COALESCE(SUM(t.price), 0)  
    INTO out_ticket_revenue  
    FROM Ticket t  
    JOIN Match m 1..n<->1: ON m.match_id = t.match_id  
    WHERE m.home_team_id = p_team_id  
        AND m.season_id = p_season_id  
        AND t.is_scanned = TRUE;  
  
    -- sponsor deal values for season  
    SELECT COALESCE(SUM(sd.deal_value), 0)  
    INTO out_sponsor_revenue  
    FROM Sponsor_deal sd  
    WHERE sd.team_id = p_team_id  
        AND sd.season_id = p_season_id;  
  
    out_total_revenue := out_ticket_revenue + out_sponsor_revenue;  
END;  
$;
```

Тригери

- **trg_lineup_max_players** – не дозволува внес на повеќе од 18 играчи во составот на еден тим за даден натпревар

```
CREATE OR REPLACE FUNCTION trg_fn_lineup_max_players()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
DECLARE
    v_count INT;
BEGIN
    SELECT COUNT(*) INTO v_count
    FROM Lineup
    WHERE match_id = NEW.match_id
        AND team_id = NEW.team_id;

    IF v_count >= 18 THEN
        RAISE EXCEPTION
            'Team % already has 18 players in the lineup for match %.',
            NEW.team_id, NEW.match_id;
    END IF;

    RETURN NEW;
END;
$$;

CREATE TRIGGER trg_lineup_max_players
BEFORE INSERT ON Lineup
FOR EACH ROW
EXECUTE FUNCTION trg_fn_lineup_max_players();
```

- **trg_prevent_duplicate_lineup** – не дозволува играч да биде во составот на два различни тима на ист датум

```
CREATE OR REPLACE FUNCTION trg_fn_prevent_duplicate_lineup()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
DECLARE
    v_conflict_match INT;
BEGIN
    SELECT l.match_id INTO v_conflict_match
    FROM Lineup l
    JOIN Match m ON m.match_id = l.match_id
    JOIN Match nm ON nm.match_id = NEW.match_id
    WHERE l.player_id = NEW.player_id
        AND l.match_id <> NEW.match_id
        AND m.match_date = nm.match_date
    LIMIT 1;

    IF FOUND THEN
        RAISE EXCEPTION
            'Player % is already in the lineup for another match (%) on the same date.',
            NEW.player_id, v_conflict_match;
    END IF;

    RETURN NEW;
END;
$$;

CREATE TRIGGER trg_prevent_duplicate_lineup
BEFORE INSERT ON Lineup
FOR EACH ROW
EXECUTE FUNCTION trg_fn_prevent_duplicate_lineup();
```

- **trg_validate_contract_dates** – не дозволува внес на договор чиј краен датум е пред почетниот, и не дозволува преклопување на два активни договори за ист играч и тим

```
CREATE OR REPLACE FUNCTION trg_fn_validate_contract_dates()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
DECLARE
    v_overlap INT;
BEGIN
    IF NEW.end_date IS NOT NULL AND NEW.end_date <= NEW.start_date THEN
        RAISE EXCEPTION
            'Contract end_date (%) must be after start_date (%)',
            NEW.end_date, NEW.start_date;
    END IF;

    SELECT COUNT(*) INTO v_overlap
    FROM Player_contract
    WHERE player_id = NEW.player_id
        AND team_id = NEW.team_id
        AND contract_id <> COALESCE(NEW.contract_id, -1)
        AND (
            (NEW.end_date IS NULL)
            OR
            (end_date IS NULL OR end_date > NEW.start_date)
        )
        AND start_date < COALESCE(NEW.end_date, 'infinity'::date);

    IF v_overlap > 0 THEN
        RAISE EXCEPTION
            'Player % already has an active contract with team % that overlaps this period.',
            NEW.player_id, NEW.team_id;
    END IF;

    RETURN NEW;
END;
$$;

CREATE TRIGGER trg_validate_contract_dates
BEFORE INSERT OR UPDATE ON Player_contract
FOR EACH ROW
EXECUTE FUNCTION trg_fn_validate_contract_dates();
```

- **trg_ticket_capacity_limit** – не дозволува внес на билет доколку стадионот е веќе полн за тој натпревар

```
CREATE OR REPLACE FUNCTION fn_trg_ticket_capacity_limit()
    RETURNS TRIGGER
    LANGUAGE plpgsql
AS
$$
DECLARE
    v_available INT;
BEGIN
    v_available := fn_stadium_available_seats(p_match_id NEW.match_id);

    IF v_available <= 0 THEN
        RAISE EXCEPTION 'Cannot issue ticket: stadium is full for match % (0 seats available).',
            NEW.match_id;
    END IF;

    RETURN NEW;
END;
$$;

CREATE OR REPLACE TRIGGER trg_ticket_capacity_limit
    BEFORE INSERT
    ON Ticket
    FOR EACH ROW
EXECUTE FUNCTION fn_trg_ticket_capacity_limit();
```

- **trg_prevent_self_transfer** – не дозволува да се направи трансфер во ист тим

```
CREATE OR REPLACE FUNCTION fn_trg_prevent_self_transfer()
    RETURNS TRIGGER
    LANGUAGE plpgsql
AS
$$
DECLARE
    v_player_name VARCHAR;
    v_team_name VARCHAR;
BEGIN
    IF NEW.from_team_id = NEW.to_team_id THEN
        SELECT p.name
        INTO v_player_name
        FROM Player p
        WHERE p.player_id = NEW.player_id;

        SELECT t.name
        INTO v_team_name
        FROM Team t
        WHERE t.team_id = NEW.from_team_id;

        RAISE EXCEPTION 'Cannot transfer player "%" to the same team "%".',
            COALESCE(v_player_name, 'Unknown'),
            COALESCE(v_team_name, 'Unknown');
    END IF;

    RETURN NEW;
END;
$$;

CREATE OR REPLACE TRIGGER trg_prevent_self_transfer
    BEFORE INSERT OR UPDATE
    ON Transfer
    FOR EACH ROW
EXECUTE FUNCTION fn_trg_prevent_self_transfer();
```


- **trg_match_no_duplicate_same_day** – не дозволува да се додаде натпревар доколку некој од тимовите веќе имаат некој друг натпревар на истиот датум

```
CREATE OR REPLACE FUNCTION fn_trg_match_no_duplicate_same_day()
    RETURNS TRIGGER
    LANGUAGE plpgsql
AS
$$
DECLARE
    v_conflict_team VARCHAR;
BEGIN
    -- check if home team has a match on this date
    IF EXISTS (SELECT 1 FROM Match m WHERE m.match_id <> COALESCE(NEW.match_id, 0)
                AND m.match_date = NEW.match_date
                AND (m.home_team_id = NEW.home_team_id OR m.away_team_id = NEW.home_team_id)) THEN
        SELECT t.name INTO v_conflict_team FROM Team t
        WHERE t.team_id = NEW.home_team_id;

        RAISE EXCEPTION 'Home team "%" already has a match scheduled on %.',
            v_conflict_team, NEW.match_date;
    END IF;

    -- check if away team has a match on this date
    IF EXISTS (SELECT 1 FROM Match m WHERE m.match_id <> COALESCE(NEW.match_id, 0)
                AND m.match_date = NEW.match_date
                AND (m.home_team_id = NEW.away_team_id OR m.away_team_id = NEW.away_team_id)) THEN
        SELECT t.name INTO v_conflict_team FROM Team t
        WHERE t.team_id = NEW.away_team_id;

        RAISE EXCEPTION 'Away team "%" already has a match scheduled on %.',
            v_conflict_team, NEW.match_date;
    END IF;

    RETURN NEW;
END;
$$;

CREATE OR REPLACE TRIGGER trg_match_no_duplicate_same_day
    BEFORE INSERT OR UPDATE
    ON Match
    FOR EACH ROW
EXECUTE FUNCTION fn_trg_match_no_duplicate_same_day();
```

Процедури

- **sp_get_team_standings** - ги враќа тимовите рангирани по поени за дадена сезона

```
CREATE OR REPLACE PROCEDURE sp_get_team_standings(p_season_id INT)
LANGUAGE plpgsql
AS $$
DECLARE
    rec RECORD;
BEGIN
    IF NOT EXISTS (SELECT 1 FROM Season WHERE season_id = p_season_id) THEN
        RAISE EXCEPTION 'Season % does not exist.', p_season_id;
    END IF;

    RAISE NOTICE 'Standings for season %:', p_season_id;

    FOR rec IN
        SELECT
            ROW_NUMBER() OVER (
                ORDER BY points DESC, goal_difference DESC, goals_for DESC
            ) AS position,
            team_name,
            matches_played,
            wins,
            draws,
            losses,
            goals_for,
            goals_against,
            goal_difference,
            points
        FROM vw_team_season_table
        WHERE season_id = p_season_id
    LOOP
        RAISE NOTICE '% | % | MP:% W:% D:% L:% GD:% Pts:%',
            rec.position, rec.team_name, rec.matches_played,
            rec.wins, rec.draws, rec.losses,
            rec.goal_difference, rec.points;
    END LOOP;
END;
$$;
```

- **sp_get_top_scorers** - ги враќа најдобрите стрелци за дадена сезона со можност за ограничување на бројот на резултати

```
CREATE OR REPLACE PROCEDURE sp_get_top_scorers(p_season_id INT, p_limit INT DEFAULT 10)
LANGUAGE plpgsql
AS $$
DECLARE
    rec RECORD;
BEGIN
    IF NOT EXISTS (SELECT 1 FROM Season WHERE season_id = p_season_id) THEN
        RAISE EXCEPTION 'Season % does not exist.', p_season_id;
    END IF;

    IF p_limit < 1 THEN
        RAISE EXCEPTION 'Limit must be at least 1, got %.', p_limit;
    END IF;

    RAISE NOTICE 'Top % scorers for season %:', p_limit, p_season_id;

    FOR rec IN
        SELECT
            ROW_NUMBER() OVER (
                ORDER BY goals DESC, assists DESC
            ) AS rank,
            player_name,
            team_name,
            matches_played,
            goals,
            assists,
            yellow_cards,
            red_cards
        FROM vw_player_season_stats
        WHERE season_id = p_season_id
        ORDER BY goals DESC, assists DESC
        LIMIT p_limit
    LOOP
        RAISE NOTICE '% | % | % | G:% A:% YC:% RC:%',
            rec.rank, rec.player_name, rec.team_name,
            rec.goals, rec.assists, rec.yellow_cards, rec.red_cards;
    END LOOP;
END;
**
```

- **sp_scan_ticket** - го означува билетот како скениран доколку веќе не е и доколку натпреварот не е поминат

```
CREATE OR REPLACE PROCEDURE sp_scan_ticket(p_ticket_id INT)
    LANGUAGE plpgsql
AS
$$
DECLARE
    v_is_scanned BOOLEAN;
    v_match_date DATE;
BEGIN
    -- get ticket info
    SELECT t.is_scanned, m.match_date
    INTO v_is_scanned, v_match_date
    FROM Ticket t
        JOIN Match m 1..n<->1: ON m.match_id = t.match_id
    WHERE t.ticket_id = p_ticket_id;

    -- ticket doesn't exist
    IF v_is_scanned IS NULL THEN
        RAISE EXCEPTION 'Ticket with id % not found.', p_ticket_id;
    END IF;

    -- already scanned
    IF v_is_scanned THEN
        RAISE EXCEPTION 'Ticket % has already been scanned.', p_ticket_id;
    END IF;

    -- match is in the past (more than 1 day ago to allow match-day scanning)
    IF v_match_date < CURRENT_DATE - INTERVAL '1 day' THEN
        RAISE EXCEPTION 'Cannot scan ticket %: the match took place on % and is no longer valid.',
            p_ticket_id, v_match_date;
    END IF;

    -- all checks passed – scan the ticket
    UPDATE Ticket
    SET is_scanned = TRUE
    WHERE ticket_id = p_ticket_id;

    RAISE NOTICE 'Ticket % scanned successfully.', p_ticket_id;
END;
$;
```

- **sp_cancel_match(p_match_id)** - ги брише сите tickets, lineups, events, odds, и referee assignments за натпревар, и на крај го брише самиот натпревар.

```
CREATE OR REPLACE PROCEDURE sp_cancel_match( p_match_id INT)
    LANGUAGE plpgsql
AS
$$
DECLARE
    v_match_date    DATE;
    v_home_name     VARCHAR;
    v_away_name     VARCHAR;
BEGIN
    -- verify the match exists and get info
    SELECT m.match_date, ht.name, at.name INTO v_match_date, v_home_name, v_away_name
    FROM Match m JOIN Team ht 1.n<->1: ON ht.team_id = m.home_team_id JOIN Team at 1.n<->1: ON at.team_id = m.away_team_id
    WHERE m.match_id = p_match_id;

    IF v_match_date IS NULL THEN
        RAISE EXCEPTION 'Match with id % not found.', p_match_id;
    END IF;

    -- delete event attributes (child of Event)
    DELETE FROM Event_attribute WHERE event_id IN (SELECT event_id FROM Event WHERE match_id = p_match_id);
    -- delete events
    DELETE FROM Event WHERE match_id = p_match_id;
    -- delete lineups
    DELETE FROM Lineup WHERE match_id = p_match_id;
    -- delete tickets
    DELETE FROM Ticket WHERE match_id = p_match_id;
    -- delete referee assignments
    DELETE FROM Referee_match WHERE match_id = p_match_id;
    -- delete odds
    DELETE FROM Match_odds WHERE match_id = p_match_id;
    -- delete the match itself
    DELETE FROM Match WHERE match_id = p_match_id;

    RAISE NOTICE 'Match cancelled: % vs % on %',
        v_home_name, v_away_name, v_match_date;
END;
$;
```