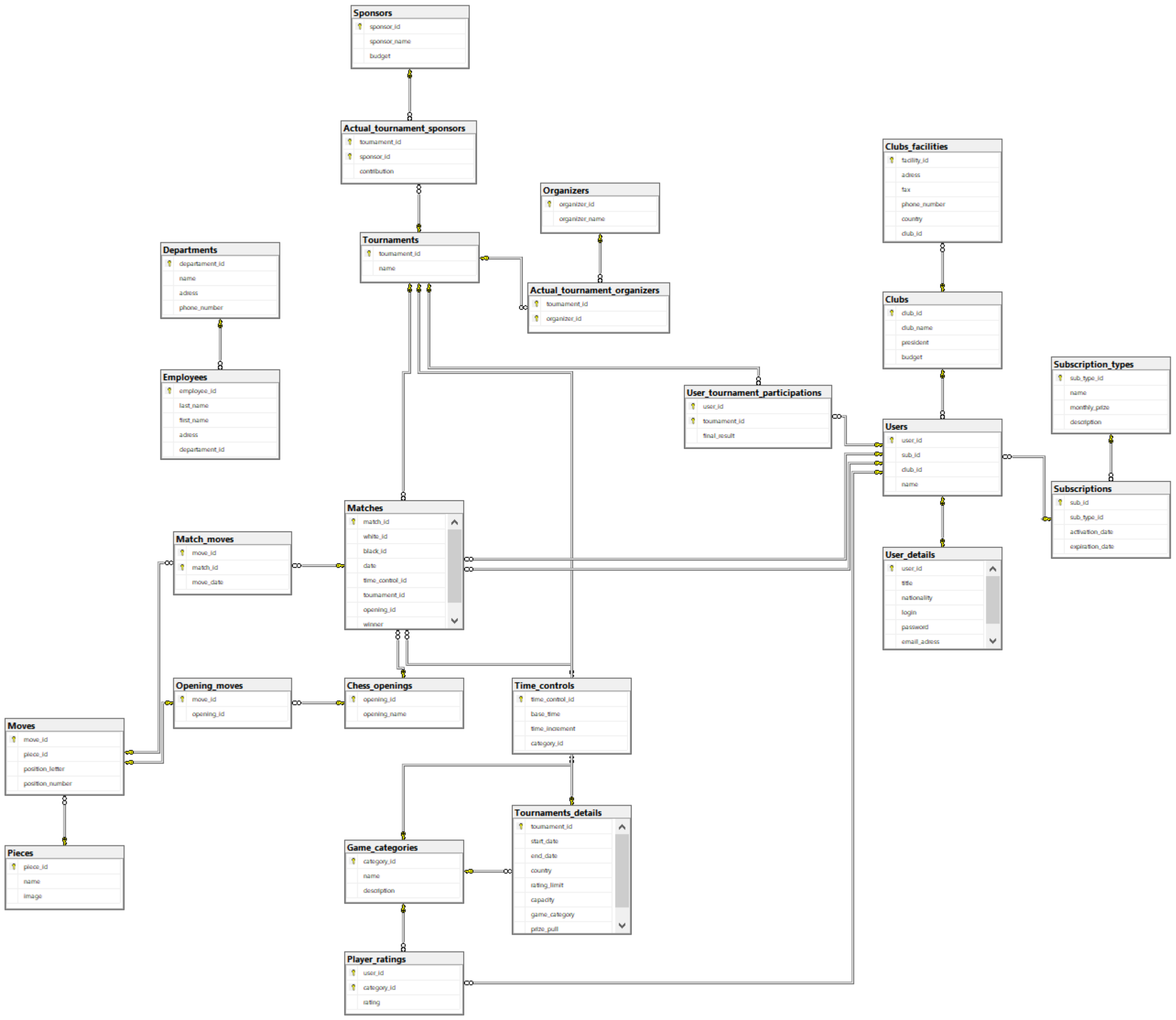
Baza Danych Serwisu Szachowego

Jan Ruszil, Izabela Ryś, Jakub Szczepanek

Spis treści

1 Diagram bazy danych



2 Tabele

2.1 Pieces

Tabela zawiera dane dotyczące bierek: identyfikator bierki (*piecie_id*), nazwę bierki (*name*), obraz (*image*).

```
CREATE TABLE Pieces (
    piece_id int  NOT NULL,
    name int  NOT NULL,
    image image  NOT NULL,
    CONSTRAINT Pieces_pk PRIMARY KEY  (piece_id),
    CONSTRAINT Piece_name_chk CHECK (name in ('Rook','Bishop','Pawn','Queen','King','Knight'))
);

ALTER TABLE Moves ADD CONSTRAINT Pieces_Moves
    FOREIGN KEY (piece_id)
    REFERENCES Pieces (piece_id);
```

2.2 Moves

Tabela zawiera dane dotyczące ruchów: identyfikator ruchu (*move_id*), identyfikator bierki(*piece_id*), dane dotyczące pola, na które przestawiono bierkę (*position_letter*, *position_number*).

```
CREATE TABLE Moves (
    move_id int  NOT NULL IDENTITY,
    piece_id int  NOT NULL,
    position_letter nvarchar(1)  NOT NULL,
    position_number int  NOT NULL,
    CONSTRAINT Moves_pk PRIMARY KEY  (move_id),
    CONSTRAINT Moves_move_correctness_chk
        CHECK (position_letter LIKE '[a-h]' AND position_number >= 1 AND position_number <= 8)
);
```

2.3 Match_moves

Tabela zawiera dane dotyczące ruchów w meczach rozgrywanych przez użytkowników: identyfikator ruchu (*move_id*), identyfikator meczu (*match_id*), data wykonania ruchu (*move_date*).

```
CREATE TABLE Match_moves (
    move_id int  NOT NULL,
    match_id int NOT NULL,
    move_date datetime  NOT NULL,
    CONSTRAINT Match_moves_pk PRIMARY KEY  (move_id,match_id)
);

ALTER TABLE Match_moves ADD CONSTRAINT Match_moves_Matches
    FOREIGN KEY (match_id)
    REFERENCES Matches (match_id);

ALTER TABLE Match_moves ADD CONSTRAINT Match_moves_Moves
    FOREIGN KEY (move_id)
    REFERENCES Moves (move_id);
```

2.4 Opening_moves

Tabela zawiera dane dotyczące ruchów w otwarciach szachowych: identyfikator ruchu (*move_id*), identyfikator otwarcia (*opening_id*).

```
CREATE TABLE Opening_moves (
    move_id int  NOT NULL,
    opening_id int  NOT NULL,
    CONSTRAINT Opening_moves_pk PRIMARY KEY  (move_id)
);

ALTER TABLE Opening_moves ADD CONSTRAINT Moves_Opening_moves
    FOREIGN KEY (move_id)
    REFERENCES Moves (move_id);

ALTER TABLE Opening_moves ADD CONSTRAINT Opening_moves_Chess_openings
    FOREIGN KEY (opening_id)
    REFERENCES Chess_openings (opening_id);
```

2.5 Chess_openings

Tabela zawiera dane dotyczące otwarć szachowych: identyfikator otwarcia (*opening_id*), nazwę otwarcia (*opening_name*).

```
CREATE TABLE Chess_openings (
    opening_id int  NOT NULL,
    opening_name nvarchar(25)  NOT NULL,
    CONSTRAINT Chess_openings_pk PRIMARY KEY  (opening_id)
);
```

2.6 Matches

Tabela zawiera dane dotyczące meczy przeprowadzanych w serwisie: identyfikator meczu (*match_id*), identyfikator gracza grającego bierkami białymi (*white_id*), identyfikator gracza grającego bierkami czarnymi (*black_id*), datę odbywania się meczu (*date*), identyfikator kontroli czasu (*time_control_id*), identyfikator otwarcia szachowego (*tournament_id*) oraz pole określające wygranego (*winner*).

```
CREATE TABLE Matches (
    match_id int  NOT NULL IDENTITY,
    white_id int  NOT NULL,
    black_id int  NOT NULL,
    date smalldatetime  NOT NULL,
    time_control_id int  NOT NULL,
    tournament_id int NULL,
    opening_id int  NULL,
    winner nvarchar(5)  NULL,
    CONSTRAINT Matches_pk PRIMARY KEY  (match_id),
    CONSTRAINT Matches_winner_chk CHECK (winner in ('white','black','draw'))
);

ALTER TABLE Matches ADD CONSTRAINT Game_types_Matches
    FOREIGN KEY (time_control_id)
    REFERENCES Time_controls (time_control_id);

ALTER TABLE Matches ADD CONSTRAINT Matches_Chess_openings
    FOREIGN KEY (opening_id)
    REFERENCES Chess_openings (opening_id);

ALTER TABLE Matches ADD CONSTRAINT Matches_Tournaments
    FOREIGN KEY (tournament_id)
    REFERENCES Tournaments (tournament_id);

ALTER TABLE Matches ADD CONSTRAINT Matches_Users_Black
    FOREIGN KEY (black_id)
    REFERENCES Users (user_id);

ALTER TABLE Matches ADD CONSTRAINT Matches_Users_White
    FOREIGN KEY (white_id)
    REFERENCES Users (user_id);
```

2.7 Game_categories

Tabela zawiera dane dotyczące kategorii gier: identyfikator kategorii (*category_id*), nazwa kategorii - Classical, Blitz, Bullet lub Rapid (*category_name*), krótki opis (*description*).

```
CREATE TABLE Game_categories (
    category_id int  NOT NULL,
    name nvarchar(9)  NOT NULL,
    description nvarchar(260)  NOT NULL,
    CONSTRAINT Game_categories_pk PRIMARY KEY  (category_id),
    CONSTRAINT Game_name_ck CHECK (name in ('Classical','Blitz','Bullet','Rapid'))
);
```

2.8 Time_controls

Tabela zawiera dane dotyczące kontroli czasu: identyfikator (*time_control_id*), podstawowy czas na grę na gracza (*base_time*), ilość czasu, o którą zwiększany jest czas na grę dla gracza po wykonaniu przez niego ruchu (*time_increment*), identyfikator kategorii (*category_id*).

```
CREATE TABLE Time_controls (
    time_control_id int  NOT NULL,
    base_time time  NOT NULL,
    time_increment time  NOT NULL,
    category_id int  NOT NULL,
    CONSTRAINT Time_controls_pk PRIMARY KEY  (time_control_id)
);

ALTER TABLE Time_controls ADD CONSTRAINT Time_controls_Game_categories
    FOREIGN KEY (category_id)
    REFERENCES Game_categories (category_id);
```

2.9 Player_ratings

Tabela zawiera dane dotyczące ratingów graczy: identyfikator użytkownika (*user_id*), identyfikator kategorii (*category_id*), rating (*rating*).

```
CREATE TABLE Player_ratings (
    user_id int  NOT NULL,
    category_id int  NOT NULL,
    rating int  NOT NULL DEFAULT 1500,
    CONSTRAINT Player_ratings_pk PRIMARY KEY  (user_id,category_id),
    CONSTRAINT Player_rating_positive CHECK (rating > 0)
);

ALTER TABLE Player_ratings ADD CONSTRAINT Game_categories_Player_ratings
    FOREIGN KEY (category_id)
    REFERENCES Game_categories (category_id);

ALTER TABLE Player_ratings ADD CONSTRAINT Player_ratings_Users
    FOREIGN KEY (user_id)
    REFERENCES Users (user_id);
```

2.10 Users

Tabela zawiera dane dotyczące użytkowników: identyfikator użytkownika (*user_id*), identyfikator subskrypcji (*sub_id*), identyfikator klubu (*club_id*), nazwa użytkownika (*name*).

```
CREATE TABLE Users (
    user_id int  NOT NULL,
    sub_id int  NULL,
    club_id int  NULL,
    name nvarchar(20) UNIQUE NOT NULL,
    CONSTRAINT Users_pk PRIMARY KEY  (user_id)
);

ALTER TABLE Users ADD CONSTRAINT Users_Clubs
    FOREIGN KEY (club_id)
    REFERENCES Clubs (club_id);

ALTER TABLE Users ADD CONSTRAINT Users_Subscriptions
    FOREIGN KEY (sub_id)
    REFERENCES Subscriptions (sub_id);
```

2.11 User_details

Tabela zawiera szczegółowe dane dotyczące użytkowników: identyfikator użytkownika (*user_id*), tytuł (*title*), narodowość (*nationality*), login (*login*), hasło (*password*), adres email (*email_address*).

```
CREATE TABLE User_details (
    user_id int  NOT NULL,
    title nvarchar(20)  NULL,
    nationality nvarchar(20)  NOT NULL,
    login nvarchar(20)  NOT NULL,
    password nvarchar(20)  NOT NULL,
    email_address nvarchar(32)  NOT NULL,
    CONSTRAINT User_details_pk PRIMARY KEY  (user_id),
    CONSTRAINT Password_length CHECK (LEN(password) >= 8)
);

ALTER TABLE User_details ADD CONSTRAINT User_details_Users
    FOREIGN KEY (user_id)
    REFERENCES Users (user_id);
```

2.12 Subscriptions

Tabela zawiera dane dotyczące subskrypcji: identyfikator subskrypcji (*sub_id*), identyfikator typu subskrypcji (*sub_type_id*), data aktywacji i wygaśnięcia (*activation_date*, *expiration_date*).

```
CREATE TABLE Subscriptions (
    sub_id int  NOT NULL IDENTITY,
    sub_type_id int  NOT NULL,
    activation_date smalldatetime  NOT NULL,
    expiration_date smalldatetime  NOT NULL,
    CONSTRAINT Subscriptions_pk PRIMARY KEY  (sub_id),
    CONSTRAINT Subscriptions_date_chk CHECK (activation_date <= expiration_date)
);

ALTER TABLE Subscriptions ADD CONSTRAINT Subscriptions_Subscription_types
    FOREIGN KEY (sub_type_id)
    REFERENCES Subscription_types (sub_type_id);
```

2.13 Subscription_types

Tabela zawiera dane dotyczące typów dostępnych subskrypcji: identyfikator typu subskrypcji (*sub_type_id*), nazwa (*name*), cena na miesiąc (*monthly_prize*), opis (*description*).

```
CREATE TABLE Subscription_types (
    sub_type_id int  NOT NULL,
    name nvarchar(20)  NOT NULL,
    monthly_prize money  NOT NULL,
    description nvarchar(100)  NOT NULL,
    CONSTRAINT Subscription_types_pk PRIMARY KEY  (sub_type_id)
);
```

2.14 Clubs

Tabela zawiera dane dotyczące klubów: identyfikator klubu (*club_id*), nazwa klubu (*club_name*), prezydent klubu (*president*) oraz budżet (*budget*).

```
CREATE TABLE Clubs (
    club_id int  NOT NULL,
    club_name nvarchar(20)  NOT NULL,
    president nvarchar(20)  NOT NULL,
    budget money  NOT NULL,
    CONSTRAINT Clubs_pk PRIMARY KEY  (club_id)
);
```

2.15 Clubs_facilities

Tabela zawiera dane dotyczące placówek klubów: identyfikator placówki (*facility_id*), adres (*address*), fax (*fax*), numer telefonu (*phone_number*), kraj (*country*), identyfikator klubu (*club_id*).

```
CREATE TABLE Clubs_facilities (
    facility_id int  NOT NULL,
    address nvarchar(25)  NOT NULL,
    fax int  NOT NULL,
    phone_number int  NOT NULL,
    country nvarchar(20)  NOT NULL,
    club_id int  NOT NULL,
    CONSTRAINT Clubs_facilities_pk PRIMARY KEY  (facility_id)
);

ALTER TABLE Clubs_facilities ADD CONSTRAINT Clubs_facilities_Clubs
    FOREIGN KEY (club_id)
    REFERENCES Clubs (club_id);
```

2.16 User_tournament_participations

Tabela zawiera dane dotyczące uczestnictwa graczy w turniejach: identyfikator uczestnika (*user_id*), identyfikator turnieju (*tournament_id*), wynik uczestnika (*final_result*).

```
CREATE TABLE User_tournament_participations (
    user_id int  NOT NULL,
    tournament_id int  NOT NULL,
    final_result int  NULL,
    CONSTRAINT User_tournament_participations_pk PRIMARY KEY  (user_id,tournament_id)
);

ALTER TABLE User_tournament_participations ADD CONSTRAINT Tournaments_User_tournament_participations
    FOREIGN KEY (tournament_id)
    REFERENCES Tournaments (tournament_id);

ALTER TABLE User_tournament_participations ADD CONSTRAINT User_tournament_participations_Users
    FOREIGN KEY (user_id)
    REFERENCES Users (user_id);
```

2.17 Tournaments

Tabela zawiera dane dotyczące turniejów: identyfikator turnieju (*tournament_id*), nazwa turnieju (*name*).

```
CREATE TABLE Tournaments (
    tournament_id int  NOT NULL,
    name nvarchar(50)  NOT NULL,
    CONSTRAINT Tournaments_pk PRIMARY KEY  (tournament_id)
);
```

2.18 Tournaments_details

Tabela zawiera szczegółowe dane dotyczące turniejów: identyfikator turnieju (*tournament_id*), czas rozpoczęcia i zakończenia turnieju (*start_date*, *end_date*), maksymalny dopuszczalny rating uczestników (*rating_limit*), maksymalna liczba uczestników (*capacity*), identyfikator kategorii gier (*game_category*), pula nagród (*prize_pool*).

```
CREATE TABLE Tournaments_details (
    tournament_id int  NOT NULL,
    start_date smalldatetime  NOT NULL,
    end_date smalldatetime  NOT NULL,
    rating_limit int  NOT NULL,
    capacity int  NOT NULL,
    game_category int  NOT NULL,
    prize_pool money NOT NULL,
    CONSTRAINT Tournaments_details_pk PRIMARY KEY  (tournament_id)
    CONSTRAINT Tournaments_details_capacity_chk CHECK (capacity > 1)
);

ALTER TABLE Tournaments_details ADD CONSTRAINT Game_categories_Tournaments_details
    FOREIGN KEY (game_category)
    REFERENCES Game_categories (category_id);

ALTER TABLE Tournaments_details ADD CONSTRAINT Tournaments_Tournaments_details
    FOREIGN KEY (tournament_id)
    REFERENCES Tournaments (tournament_id);
```

2.19 Actual_tournament_sponsors

Tabela zawiera dane dotyczące informacje o tym, jacy sponsorzy są odpowiedzialni za które turnieje: identyfikator turnieju (*tournament_id*), identyfikator sponsora (*sponsor_id*), wkład pieniężny (*contribution*).

```
CREATE TABLE Actual_tournament_sponsors (
    tournament_id int  NOT NULL,
    sponsor_id int  NOT NULL,
    contribution money NOT NULL,
    CONSTRAINT Actual_tournament_sponsors_pk PRIMARY KEY  (tournament_id,sponsor_id)
);

ALTER TABLE Actual_tournament_sponsors ADD CONSTRAINT Sponsors_Actual_tournament_sponsors
    FOREIGN KEY (sponsor_id)
    REFERENCES Sponsors (sponsor_id);

ALTER TABLE Actual_tournament_sponsors ADD CONSTRAINT Tournaments_Actual_tournament_sponsors
    FOREIGN KEY (tournament_id)
    REFERENCES Tournaments (tournament_id);
```

2.20 Sponsors

Tabela zawiera dane dotyczące sponsorów: identyfikator sponsora (*sponsor_id*), nazwę sponsora (*sponsor_name*) oraz budżet (*budget*).

```
CREATE TABLE Sponsors (
    sponsor_id int  NOT NULL IDENTITY,
    sponsor_name nvarchar(30)  NOT NULL,
    budget money NOT NULL,
    CONSTRAINT Sponsors_pk PRIMARY KEY  (sponsor_id)
);
```

2.21 Actual_tournament_organizers

Tabela zawiera dane dotyczące informacje o tym, jacy organizatorzy są odpowiedzialni za które turnieje: identyfikator turnieju (*tournament_id*), identyfikator organizatora (*organizer_id*).

```
CREATE TABLE Actual_tournament_organizers (
    tournament_id int  NOT NULL,
    organizer_id int  NOT NULL,
    CONSTRAINT Actual_tournament_organizers_pk PRIMARY KEY  (tournament_id,organizer_id)
);
```

```
ALTER TABLE Actual_tournament_organizers ADD CONSTRAINT Actual_tournament_organizers_Organizers
    FOREIGN KEY (organizer_id)
    REFERENCES Organizers (organizer_id);
```

```
ALTER TABLE Actual_tournament_organizers ADD CONSTRAINT Tournaments_Actual_tournament_organizers
    FOREIGN KEY (tournament_id)
    REFERENCES Tournaments (tournament_id);
```

2.22 Organizers

Tabela zawiera dane dotyczące organizatorów: identyfikator organizatora (*organizer_id*), nazwę organizatora (*organizer_name*).

```
CREATE TABLE Organizers (
    organizer_id int  NOT NULL IDENTITY,
    organizer_name nvarchar(30)  NOT NULL,
    CONSTRAINT Organizers_pk PRIMARY KEY  (organizer_id)
);
```

2.23 Departments

Tabela zawiera dane dotyczące działów: identyfikator działu (*department_id*), nazwę działu (*name*), adres (*address*), numer telefonu (*phone_number*).

```
CREATE TABLE Departments (
    department_id int  NOT NULL,
    name nvarchar(30)  NOT NULL,
    address nvarchar(25)  NOT NULL,
    phone_number int  NOT NULL,
    CONSTRAINT Departments_pk PRIMARY KEY  (department_id)
);
```

2.24 Employees

Tabela zawiera dane dotyczące pracowników: identyfikator pracownika (*employee_id*), imię (*first_name*), nazwisko (*last_name*), adres (*address*), identyfikator działu (*department_id*).

```
CREATE TABLE Employees (
    employee_id int  NOT NULL IDENTITY,
    last_name nvarchar(30)  NOT NULL,
    first_name nvarchar(30)  NOT NULL,
    address nvarchar(25)  NOT NULL,
    department_id int  NOT NULL,
    CONSTRAINT Employees_pk PRIMARY KEY  (employee_id)
);
```

```
ALTER TABLE Employees ADD CONSTRAINT Employees_Department
    FOREIGN KEY (departament_id)
    REFERENCES Departments (departament_id);
```

3 Widoki

3.1 BestPlayerInEachCategory

Widok zwraca najlepszego gracza dla wszystkich kategorii. Zawiera informacje o nazwie kategorii(*name*), identyfikatorze gracza (*user_id*) oraz ratingu najlepszego gracza (*rating*).

```
CREATE VIEW BestPlayerInEachCategory AS
SELECT g.name, s.user_id, s.rating FROM Game_categories g
JOIN(
SELECT user_id, category_id, rating, ROW_NUMBER() OVER(PARTITION BY(category_id) ORDER BY rating DESC) [row]
FROM Player_ratings p) s
ON g.category_id = s.category_id AND s.row = 1
```

3.2 OngoingTournaments

Widok zwraca aktualnie trwające turnieje. Zawiera informacje o nazwie turnieju (*name*), początku (*start_date*) oraz końcu (*end_date*) turnieju.

```
CREATE VIEW OngoingTournaments AS
SELECT t.name, td.start_date, td.end_date FROM Tournaments_details td
JOIN Tournaments t ON
t.tournament_id = td.tournament_id
WHERE td.start_date <= GETDATE() AND td.end_date >= GETDATE()
```

3.3 PlayersWithoutClub

Widok zwraca graczy, którzy nie są w żadnym klubie. Zawiera informacje dotyczące identyfikatora użytkownika (*user_id*) oraz jego nazwy (*name*).

```
CREATE VIEW PlayersWithoutClub AS
SELECT user_id, name FROM Users
WHERE club_id IS NULL
```

3.4 TournamentsWinners

Widok zwraca graczy, którzy kiedykolwiek wygrali jakikolwiek turniej oraz ile razy go wygrali. Zawiera informacje dotyczące identyfikatora użytkownika (*user_id*),jego nazwy (*name*) oraz liczby zwycięstw (*NumberOfWins*).

```
CREATE VIEW TournamentsWinners AS
SELECT U.user_id, U.name, s.NumberOfWins  FROM Users U
JOIN (
SELECT Utp.user_id, COUNT(*) AS NumberOfWins FROM User_tournament_participations Utp
WHERE Utp.final_result = 1
GROUP BY Utp.user_id
) AS s
ON s.user_id = U.user_id
```

3.5 OpeningsPopularity

Widok zwraca otwarcia i ich liczbę wykorzystań w meczach. Zawiera informacje o nazwie otwarcia (*opening_name*) oraz liczbę, która określa w ilu meczach to otwarcie zostało wykorzystane (*Popularity*).

```
CREATE VIEW OpeningsPopularity AS
SELECT co.opening_name, COALESCE(0,s.Popularity) AS Popularity FROM Chess_openings co
LEFT JOIN (
SELECT opening_id, COUNT(opening_id) AS Popularity FROM Matches
GROUP BY opening_id
) AS s
ON s.opening_id = co.opening_id
```

3.6 UpcomingTournaments

Widok zwraca nadchodzące w przyszłości turnieje. Zawiera informacje dotyczące nazwy turnieju (*Tournament name*), daty początku (*Start date*) oraz końca (*End date*).

```
CREATE VIEW UpcomingTournaments AS
SELECT T.name AS "Tournament name", TD.start_date AS "Start date", TD.end_date AS "End date"
FROM Tournaments_details TD
JOIN Tournaments T ON TD.tournament_id = T.tournament_id
WHERE TD.end_date >= GETDATE() AND TD.start_date >= GETDATE()
```

3.7 BasicSubscriptionUsers

Widok zwraca użytkowników z podstawową subskrypcją. Zawiera informacje dotyczące nazwy użytkownika (*name*), datę aktywacji subskrypcji (*activation_date*) oraz datę jej wygaśnięcia (*expiration_date*).

```
CREATE VIEW BasicSubscriptionUsers AS
SELECT U.name, S.activation_date, S.expiration_date
FROM Users U
JOIN Subscriptions S ON S.sub_id = U.sub_id
JOIN Subscription_types T ON S.sub_type_id = T.sub_type_id
WHERE T.name = 'Basic'
```

4 Funkcje

4.1 BestPlayersInCategory

Funkcja zwracająca najlepszego/ych graczy w danej kategorii. Przyjmuje jako parametr nazwę kategorii (*category*).

```
CREATE FUNCTION BestPlayersInCategory
( @category nvarchar(9) )
RETURNS @BestPlayers TABLE
(
    name nvarchar(20),
    title nvarchar(20),
    nationality nvarchar(20),
    rating int
)
AS
BEGIN
    IF (@category = 'blitz' OR
        @category = 'bullet' OR
        @category = 'rapid' OR
        @category = 'classical')
        INSERT INTO @BestPlayers SELECT TOP 1 WITH TIES
            u.name, ud.title, ud.nationality, p.rating FROM User_details ud
        JOIN Users u ON u.user_id = ud.user_id
        JOIN Player_ratings p ON ud.user_id = p.user_id AND
            p.category_id = (SELECT category_id FROM Game_categories gc
        WHERE gc.name = @category)
        ORDER BY rating
    ELSE
        INSERT INTO @BestPlayers VALUES
        (NULL,NULL,NULL,NULL)
    RETURN
END
```

4.2 GenerosityOfSponsors

Funkcja zwracająca tabelę sponsorów uwzględnioną o sumę pieniędzy, którą zdecydowali się przekazać na wsparcie turniejów szachowych, posortowaną malejąco względem tejże sumy.

```
CREATE FUNCTION GenerosityOfSponsors ()
RETURNS @GSponsors TABLE(
    sponsor_name nvarchar(30),
    contribution money
)
AS
```

```
BEGIN
    INSERT INTO @GSponsors
    SELECT s.sponsor_name, COALESCE(sub.contribution, 0) AS contribution FROM Sponsors AS s
    LEFT JOIN (
        SELECT sponsor_id, SUM(contribution) AS contribution FROM Actual_tournament_sponsors
        GROUP BY sponsor_id
    ) AS sub
    ON s.sponsor_id = sub.sponsor_id
    ORDER BY contribution DESC
    RETURN
END
```

4.3 RankingForGivenCategory

Funkcja zwracająca tabelę w zależności od podanej kategorii posortowaną malejąco. Przyjmuje jako parametr identyfikator kategorii (*Category_id*).

```
CREATE FUNCTION RankingForGivenCategory ( @Category_id int )
RETURNS @RankingTab TABLE
(
    Name nvarchar(20),
    Rating int
)
AS
BEGIN
    INSERT INTO @RankingTab
    SELECT U.name, P.rating FROM Player_ratings P
    JOIN Users U ON U.user_id = P.user_id
    WHERE P.category_id = @Category_id
    ORDER BY P.rating DESC
    RETURN
END
```

4.4 PlayerTournamentHistory

Funkcja zwracająca tabelę historii gier podanego użytkownika. Przyjmuje jako parametr nazwę użytkownika (*Name*).

```
CREATE FUNCTION PlayerTournamentHistory ( @Name nvarchar(20) )
RETURNS @TournamentHistoryTab TABLE
(
    TournamentName nvarchar(50),
    PlayerPosition int,
    StartDate smalldatetime,
    EndDate smalldatetime
)
AS
BEGIN
    DECLARE @Player_id int
    SELECT @Player_id = Users.user_id FROM Users
    WHERE Users.name = @Name

    INSERT INTO @TournamentHistoryTab
        SELECT T.name, UTP.final_result, TD.start_date, TD.end_date FROM User_tournament_participations UTP
        JOIN Tournaments T ON T.tournament_id = UTP.tournament_id
        JOIN Tournaments_details TD ON TD.tournament_id = UTP.tournament_id
        WHERE UTP.user_id = @Player_id
        ORDER BY TD.start_date
    RETURN
END
```

4.5 GetExpScore

Funkcja pomocnicza do wyliczania ratingu graczy.

```
CREATE FUNCTION GetExpScore
(
    @User_A_id int,
    @User_B_id int,
    @Category_id int
)
RETURNS float
AS
BEGIN
    DECLARE @Rating_A int, @Rating_B int
    SET @Rating_A = (SELECT rating FROM Player_ratings
    WHERE (user_id = @User_A_id AND category_id = @Category_id))
    SET @Rating_B = (SELECT rating FROM Player_ratings
    WHERE (user_id = @User_B_id AND category_id = @Category_id))
    DECLARE @E float
    SET @E = 1.0 / (1.0 + POWER(10, (@Rating_B - @Rating_A) / 400.0))
    RETURN @E
END
```

5 Procedury

5.1 AddNewSubscription

Procedura dodająca nową subskrypcję użytkownikowi. Przyjmuje parametry: identyfikator użytkownika (*user_id*), identyfikator typu subskrypcji (*sub_type_id*), datę aktywacji (*activation_date*) oraz wygaśnięcia subskrypcji (*expiration_date*).

```
CREATE PROCEDURE AddNewSubscription
(
    @user_id int,
    @sub_type_id int,
    @activation_date datetime,
    @expiration_date datetime
)
AS
BEGIN
    IF (EXISTS( SELECT user_id FROM Users u
        JOIN Subscriptions s ON u.sub_id = s.sub_id AND u.user_id = @user_id
    ))
    BEGIN
        RAISERROR('User cannot have two subscriptions at the one time!',-1,-1)
    END
    ELSE
    BEGIN
        DECLARE @sub_id int
        SET @sub_id = (NEXT VALUE FOR Subscription_id)
        INSERT INTO Subscriptions VALUES
        (@sub_id, @sub_type_id, @activation_date, @expiration_date)
        UPDATE Users
        SET sub_id = @sub_id
        WHERE user_id = @user_id
    END
END
```

5.2 ModifySubscription

Procedura zmieniająca subskrypcję użytkownika. Przyjmuje parametry: identyfikator użytkownika (*user_id*), identyfikator typu subskrypcji (*sub_type_id*), datę aktywacji (*activation_date*) oraz wygaśnięcia subskrypcji (*expiration_date*).

```
CREATE PROCEDURE ModifySubscription
(
    @user_id int,
    @sub_type_id int,
    @activation_date datetime,
    @expiration_date datetime
)
AS
BEGIN
    IF ( NOT EXISTS( SELECT u.user_id FROM Users u
        JOIN Subscriptions s ON u.sub_id = s.sub_id
        AND u.user_id = @user_id ))
    BEGIN
        RAISERROR('User does not have subscription!',-1,-1)
    END
    ELSE
    BEGIN
    DECLARE @sub_id int
    SET @sub_id = (SELECT s.sub_id FROM Subscriptions s JOIN Users u
        ON u.sub_id = s.sub_id AND u.user_id = @user_id)
    UPDATE Subscriptions
    SET sub_type_id = @sub_type_id, activation_date = @activation_date, expiration_date = @expiration_date
    WHERE sub_id = @sub_id
    END
END
```

5.3 JoinOrChangeClub

Procedura dodająca lub zmieniejąca klub użytkownika. Przyjmuje parametry: identyfikator użytkownika (*user_id*) oraz identyfikator klubu (*club_id*).

```
CREATE PROCEDURE join_or_change_club
(
    @user_id int,
    @club_id int
)
AS
BEGIN
    IF (NOT EXISTS(
        SELECT user_id FROM Users
        WHERE user_id = @user_id
    ))
    BEGIN
        RAISERROR('There is no such player',-1,-1)
        RETURN
    END
    IF (NOT EXISTS(
        SELECT club_id FROM Clubs
        WHERE club_id = @club_id
    ))
    BEGIN
        RAISERROR('There is no such club',-1,-1)
        RETURN
    END
    UPDATE Users
    SET club_id = @club_id
    WHERE user_id = @user_id
END
```

5.4 AddNewTournament

Procedura dodająca nowy turniej. Przyjmuje parametry: identyfikator turnieju (*Tournament_id*), nazwę turnieju (*Name*), datę startu (*Start*) oraz końca turnieju (*End*), ograniczenie na rating graczy biorących udział w turnieju (*Rating_limit*), maksymalną liczbę uczestników turnieju (*Capacity*) oraz pulę nagród pieniężnych (*Prize_pool*).

```
CREATE PROCEDURE AddNewTournament
(
    @Tournament_id int,
    @Name nvarchar(50),
    @Start smalldatetime,
    @End smalldatetime,
    @Rating_limit int,
    @Capacity int,
    @Game_category int
    @Prize_pool money
)
AS
BEGIN
    SET NOCOUNT ON;
    INSERT INTO Tournaments VALUES
    (@Tournament_id, @Name)
    INSERT INTO Tournaments_details VALUES
    (@Tournament_id, @Start, @End, @Rating_limit, @Capacity, @Game_category, @Prize_pool)
END
```

5.5 AddNewUser

Procedura dodająca nowego użytkownika do bazy danych. Przyjmuje parametry: identyfikator użytkownika (*User_id*), nazwę użytkownika(*Name*), narodowość użytkownika (*Nationality*), login użytkownika (*Login*), hasło użytkownika (*Password*) oraz adres email użytkownika (*Email*).

```
CREATE PROCEDURE AddNewUser
(
    @User_id int,
    @Name nvarchar(20),
    @Nationality nvarchar(20),
    @Login nvarchar(20),
    @Password nvarchar(20),
    @Email nvarchar(32)
)
AS
BEGIN
    SET NOCOUNT ON;
    INSERT INTO Users VALUES
    (@User_id, NULL, NULL, @Name)
    INSERT INTO User_details VALUES
    (@User_id, NULL, @Nationality, @Login, @Password, @Email)
END
```

5.6 UpdateRating

Procedura zmieniająca rating gracza (*User_id*) w podanej kategorii (*Category_id*) o podaną wielkość (*Rating_change*).

```
CREATE PROCEDURE UpdateRating
(
    @User_id int,
    @Category_id int,
    @Rating_change int
)
AS
```

```
BEGIN
    UPDATE Player_ratings
    SET rating = rating + @Rating_change
    WHERE user_id = @User_id AND category_id = @Category_id
END
```

6 Wyzwalacze

6.1 SponsorsBudgetCheck

Trigger sprawdzający czy sponsor nie wpłaca więcej na turniej niż wynosi jego budżet.

```
CREATE TRIGGER Sponsors_budget_check
ON Actual_tournament_sponsors
AFTER INSERT AS
BEGIN
    DECLARE @sponsor_id AS int
    SET @sponsor_id = (SELECT sponsor_id FROM inserted)
    DECLARE @cash_limit AS money
    SET @cash_limit = (SELECT budget FROM Sponsors WHERE sponsor_id = @sponsor_id)
    DECLARE @contribution AS money
    SET @contribution = (SELECT contribution FROM inserted)
    IF (@contribution > @cash_limit)
    BEGIN
        RAISERROR('It is not possible to exceed sponsor''s budget',-1,-1)
        ROLLBACK TRANSACTION
    END
END
```

6.2 ActualRatingCheck

Trigger sprawdzający czy rating gracza dołączającego do turnieju jest poniżej limitu.

```
CREATE TRIGGER Actual_rating_check ON User_tournament_participations AFTER INSERT AS
BEGIN
    DECLARE @User_id AS int
    SET @User_id = (SELECT user_id FROM inserted)
    DECLARE @Tournament_id AS int
    SET @Tournament_id = (SELECT tournament_id FROM inserted)
    DECLARE @Tournament_category as int
    SET @Tournament_category = (SELECT game_category FROM Tournaments_details
        WHERE tournament_id = @Tournament_id)
    DECLARE @User_rating as int
    SET @User_rating = (SELECT rating FROM Player_ratings
        WHERE user_id = @User_id AND category_id = @Tournament_category)
    DECLARE @Tournament_rating AS int
    SET @Tournament_rating = (SELECT rating_limit FROM Tournaments_details
        WHERE tournament_id = @Tournament_id)
    IF (@Tournament_rating < @User_rating)
    BEGIN
        RAISERROR('Players rating is to high',-1,-1)
        ROLLBACK TRANSACTION
    END
END
```

6.3 ActualParticipantsCountCheck

Trigger sprawdzający czy dodanie gracza nie przekroczy liczby uczestników w turnieju.

```
CREATE TRIGGER Actual_participants_count_check
ON User_tournament_participations
AFTER INSERT AS
BEGIN
    DECLARE @Tournament_id AS int
    SET @Tournament_id = (SELECT Tournament_id FROM inserted)
    DECLARE @Tournament_capacity AS int
    SET @Tournament_capacity = (SELECT capacity FROM Tournaments_details
        WHERE tournament_id = @Tournament_id)
    DECLARE @Participants_number AS int
    SET @Participants_number = (SELECT COALESCE(s.total,t.total) AS cnt
    FROM (SELECT 0 AS total) t
    LEFT JOIN (
        SELECT COUNT(*) AS total FROM User_tournament_participations
        GROUP BY tournament_id
        HAVING tournament_id = @Tournament_id
    ) s
    ON 1=1)
    IF (@Tournament_capacity < @Participants_number)
    BEGIN
        RAISERROR('Participants number over tournament''s capacity.',-1,-1)
        ROLLBACK TRANSACTION
    END
END
```

6.4 UpdatePlayersRatings

Trigger uaktualniający rating graczy po skończonym meczu.

```
CREATE TRIGGER UpdatePlayersRatings
ON Matches
AFTER UPDATE AS
BEGIN
    SET NOCOUNT ON;
    IF UPDATE(winner)
    BEGIN
        DECLARE @Winner nvarchar(5), @Category_id int, @White_id int, @Black_id int
        SET @Winner = (SELECT winner FROM inserted)
        SET @Category_id = (SELECT tk.category_id FROM inserted
                JOIN Time_controls tk ON tk.time_control_id = inserted.time_control_id)
        SET @White_id = (SELECT white_id FROM inserted)
        SET @Black_id = (SELECT black_id FROM inserted)

        DECLARE @E_White float, @E_Black float
        SET @E_White = dbo.GetExpScore(@White_id, @Black_id, @Category_id)
        SET @E_Black = 1 - @E_White
        DECLARE @Rating_change int

        IF @Winner = 'draw'
        BEGIN
            SET @Rating_change = 32 * (0.5 - @E_White)
            EXEC UpdateRating @White_id,  @Category_id, @Rating_change
            SET @Rating_change = 32 * (0.5 - @E_Black)
            EXEC UpdateRating @Black_id,  @Category_id, @Rating_change
        END
        ELSE IF @Winner = 'white'
        BEGIN
            SET @Rating_change = 32 * (1.0 - @E_White)
            EXEC UpdateRating @White_id,  @Category_id, @Rating_change
            SET @Rating_change = 32 * (0.0 - @E_Black)
            EXEC UpdateRating @Black_id,  @Category_id, @Rating_change
        END
        ELSE IF @Winner = ’black’
        BEGIN
            SET @Rating_change = 32 * (0.0 - @E_White)
            EXEC UpdateRating @White_id,  @Category_id, @Rating_change
            SET @Rating_change = 32 * (1.0 - @E_Black)
            EXEC UpdateRating @Black_id,  @Category_id, @Rating_change
        END
    END
END
```

6.5 ActualOrganizersCountCheck

Trigger sprawdzający czy liczba organizatorów nie przekracza 10-krotności maksymalnej liczby uczestników turnieju.

```
CREATE TRIGGER ActualOrganizersCountCheck
ON Actual_tournament_organizers
AFTER INSERT AS
BEGIN
    DECLARE @Tournament_id AS int
    SET @Tournament_id = (SELECT Tournament_id FROM inserted)
    DECLARE @Tournament_capacity AS int
    SET @Tournament_capacity = (SELECT capacity FROM Tournaments_details
        WHERE tournament_id = @Tournament_id)
    DECLARE @Organizers_number AS int
    SET @Organizers_number = (SELECT COALESCE(s.total,t.total) AS cnt
    FROM (SELECT 0 AS total) t
    LEFT JOIN (
        SELECT COUNT(*) AS total FROM Actual_tournament_organizers
        GROUP BY tournament_id
        HAVING tournament_id = @Tournament_id
    ) s
    ON 1=1)
    IF (@Tournament_capacity < 10*@Organizers_number)
    BEGIN
        RAISERROR(’Organizers number cannot exceed over 10 times tournament’’s capacity.’,-1,-1)
        ROLLBACK TRANSACTION
    END
END
```

7 Indeksy

Oprócz indeksów klastrowych na każdym kluczu głównym (domyślnych), indeksy nieklastrowe zostały utworzone na wszystkich kluczach obcych. Oprócz tego zostały utworzone unikalne indeksy nieklastrowe dla tabeli *matches* (identyfikator meczu wraz z datą jego rozegrania) oraz *tournament_details* (identyfikator turnieju wraz z datą jego startu).

```
CREATE NONCLUSTERED INDEX [Actual_tournament_organizers_Organizers_Index]
ON Actual_tournament_organizers
(
    organizer_id ASC
)

CREATE NONCLUSTERED INDEX [Clubs_facilities_Clubs_Index]
ON Clubs_facilities
(
    club_id ASC
)
```

```
CREATE NONCLUSTERED INDEX [Employees_Department_Index]
ON Employees
(
    departament_id ASC
)

CREATE NONCLUSTERED INDEX [Game_categories_Player_ratings_Index]
ON Player_ratings
(
    category_id ASC
)

CREATE NONCLUSTERED INDEX [Game_types_Matches_Index]
ON Matches
(
    time_control_id ASC
)

CREATE NONCLUSTERED INDEX [Match_moves_Matches_Index]
ON Match_moves
(
    match_id ASC
)

CREATE NONCLUSTERED INDEX [Match_moves_Moves_Index]
ON Match_moves
(
    move_id ASC
)

CREATE NONCLUSTERED INDEX [Matches_Chess_openings_Index]
ON Matches
(
    opening_id ASC
)

CREATE NONCLUSTERED INDEX [Matches_Tournaments_Index]
ON Matches
(
    tournament_id ASC
)

CREATE NONCLUSTERED INDEX [Matches_Users_Black_Index]
ON Matches
(
    black_id ASC
)

CREATE NONCLUSTERED INDEX [Matches_Users_White_Index]
ON Matches
(
    white_id ASC
)

CREATE NONCLUSTERED INDEX [Pieces_Moves_Index]
ON Moves
(
    piece_id ASC
)
```

```
CREATE NONCLUSTERED INDEX [Player_ratings_Users_Index]
ON Player_ratings
(
    user_id ASC
)

CREATE NONCLUSTERED INDEX [Sponsors_Actual_tournament_sponsors_Index]
ON Actual_tournament_sponsors
(
    sponsor_id ASC
)

CREATE NONCLUSTERED INDEX [Subscriptions_Subscription_types_Index]
ON Subscriptions
(
    sub_type_id ASC
)

CREATE NONCLUSTERED INDEX [Time_controls_Game_categories_Index]
ON Time_controls
(
    category_id ASC
)

CREATE NONCLUSTERED INDEX [Tournaments_Actual_tournament_organizers_Index]
ON Actual_tournament_organizers
(
    tournament_id ASC
)

CREATE NONCLUSTERED INDEX [Tournaments_Actual_tournament_sponsors_Index]
ON Actual_tournament_sponsors
(
    tournament_id ASC
)

CREATE NONCLUSTERED INDEX [Tournaments_Tournaments_details_Index]
ON Tournaments_details
(
    tournament_id ASC
)

CREATE NONCLUSTERED INDEX [Tournaments_User_tournament_participations_Index]
ON User_tournament_participations
(
    tournament_id ASC
)

CREATE NONCLUSTERED INDEX [User_details_Users_Index]
ON User_details
(
    user_id ASC
)
```

```
CREATE NONCLUSTERED INDEX [User_tournament_participations_Users_Index]
ON User_tournament_participations
(
    user_id ASC
)

CREATE NONCLUSTERED INDEX [Users_Clubs_Index]
ON Users
(
    club_id ASC
)

CREATE NONCLUSTERED INDEX [Users_Subscriptions_Index]
ON Users
(
    sub_id ASC
)

CREATE UNIQUE NONCLUSTERED INDEX [Matches_Unique_match_Index]
ON Matches
(
    match_id ASC,
    date ASC
)

CREATE UNIQUE NONCLUSTERED INDEX [Tournaments_Unique_tournament_Index]
ON Tournaments_details
(
    tournament_id ASC,
    start_date ASC
)
```

8 Role

W systemie proponujemy zdefiniowanie dwóch następujących ról - administrator oraz użytkownik.

8.1 Administrator

Posiada dostęp do wszystkich funkcji, widoków oraz procedur.

8.2 Użytkownik

Posiada dostęp do następujących widoków, funkcji oraz procedur:

– *BestPlayerInEachCategory*
– *OngoingTournaments*
– *TournamentsWinners*
– *OpeningsPopularity*
– *UpcomingTournaments*
– *JoinOrChangeClub*
– *AddNewUser*