DataBases – Assigment 2 – SQL

Design

Tables

Default of and attribute is not nullable.

Team – similar to the business object description

TeamID(INT)>0 – Primary Key

Match - similar to the business object description

MatchID(INT)>0) – Primary Key	HomeTeamID– References Team.TeamID	AwayTeamID– References Team.TeamID	Competition (varchar(15)) = Interrnational || Domestic

Player - similar to the business object description

PlayerID(INT)>0 – Primary Key	TeamID- References Team.TeamID	Age(INT)>0	Height(INT)>0	PreferdFoot(VARCAHR(5)) =Left || Right

Stadium - similar to the business object description

StadiumID(INT)>0 – Primary Key	Capacity(INT)>0	BelongTo - References Team.TeamID **– nullable**

InStadium

MatchID – Primay Key, Reference Match.MatchID	StadiumID – Reference Stadium.StadiumID	Attendence(INT)>0

InStadium is an unique reference relation but although we saw in class this table is not needed and we can write the information in Match table, we still decided to create it for 2 reasons:

- Saving space, using only Match would enforce us to save Stadium for a Match, if a game didn't played in a Stadium we should save null, this is a waste of space
- We knew InStadium will be used in many queries, and we wanted to make it easier for us.

In this design we only save the matches we **know** played in a stadium.

Attendence – is the number of spectators in the match, if it is 0 we don't need to save the information.

ScoreIn

MatchID - Reference Match.MatchID	PlayerID – Reference Player.PlayerID	Amount(INT)>0

Primary Key(MatchID, PlayerID)

ScoreIn is a many to many relation: many players can score in a match, player can score in many matches. we created a table for this relation.

Amount – the number of goals the player scored in the match, if it is 0 we don't need to save the information.

Views

We don't know how the data access layer will be used, we can't estimate which function will be called the most, how many players will be added and then removed (same with matches, and stadiums) aka we don't know if creating a materialized view will cost more than it's value. To prevent from resources to be wasted on maintenance of views (when we add or remove data), we decided to use only non-materialized views where we see code duplication or when the query is too long and complicated.

ActiveTallTeams – A list contains all teams IDs with more than 2 players higher than 190 and the team played at least one match as home or away team. The view was created to prevent code duplication in getActiveTallTeams() and getActiveTallRichTeams(). To create the view we select the TeamID From Player where Height>190. Group by TeamID and using having to add the rules to choose the team we want(active and have more than 1 player in COUNT).

PopularHomeTeam – A list contains all teams who played as a home team at least in 1 game and in all their matches as a home team they had more than 40,000 spectators. The view was created to simplify popularTeams(). To create the view we select HomeTeamID From Match join with InStadium, group by HomeTeamID where the MIN(Attendence)>40,000.

NotHomeTeams – A list contains all the teams who didn't played as a home team in any match. The view was created to simplify popularTeams(). To create the view we extract all the team who appear in HomeTeamID from Team table.

Api implementation

CRUD API

ReturnValue addTeam(Int teamID) – teamID is inserted into Team table. If teamID>0 and not found in table.

ReturnValue addMatch(Match match) – match is inserted into Match table. All the attributes of match inserted into the suitable attribute in Match table. the tuple will be inserted only if it meets all the table creation rules.

Match getMatchProfile(Int matchID) – if there is a tuple in table Match where MatchID = matchID we return the tuple as a Match object will all it attributes.

ReturnValue deleteMatch (Match match) – if there is a tuple in Match table where MatchID = match.MatchID we delete the tuple and all the tuples who reference this MatchID in tables ScoreIn and InStadium will be deleted.

ReturnValue addPlayer (Player player) - player is inserted into Player table. All the attributes of player inserted into the suitable attribute in Player table. the tuple will be inserted only if it meets all the table creation rules.

Player getPlayerProfile (Int playerID) - if there is a tuple in table Player where PlayerID = playerID we return the tuple as a Player object will all it attributes.

ReturnValue deletePlayer(Player player) - if there is a tuple in Player table where PlayerID = player.PlayerID we delete the tuple and all the tuples who reference this PlayerID in tables ScoreIn will be deleted.

ReturnValue addStadium (Stadium stadium) - Stadium is inserted into Stadium table. All the attributes of stadium inserted into the suitable attribute in Stadium table. the tuple will be inserted only if it meets all the table creation rules.

Stadium getStadiumProfile (Int StadiumID) - - if there is a tuple in table Stadium where StadiumID = stadiumID we return the tuple as a Stadium object will all it attributes.

ReturnValue deleteStadium(Stadium stadium) - if there is a tuple in Stadium table where StdadiumID = stadium.StadiumID we delete the tuple and all the tuples who reference this StadiumID in tables InStadium will be deleted.

Basic API

ReturnValue playerScoredInMatch(Match match, Player player, int Amount) we insert into ScoreIn table a tuple with the values: MatchID = match.MatchID, PlayerID = player.PlayerID, Amount= Amount. the tuple will be inserted only if it meets all the table creation rules.

ReturnValue playerDidntScoreInMatch(Match match, Player player) – in ScoreIn table we search for a tuple where the primary key is (match.MatchID, player.PlayerID) if it exist we delete it.

ReturnValue matchInStadium(Match match, Stadium stadium, Int attendance) – we insert into InStadium a tuple with the values: MatchID = match.MatchID, StadiumID = stadium.StadiumID, Attendence = attendance. the tuple will be inserted only if it meets all the table creation rules.

ReturnValue matchNotInStadium(Match match, Stadium stadium) – in InStadium we search for a tuple with the values match.MatchID and stadium.StadiumID if it exist we delete it.

Float averageAttendanceInStadium(**Int stadiumID)** – we use the aggregation function AVG on the attribute Attendance on the table InStadium where StadiumID = stadiumID.

Int stadiumTotalGoals(**Int stadiumID)** – we use the aggregation function SUM on the attribute Amount in ScoreIn table Where MatchID is in the matches which played in the stadium ID with a nested query to get the matches which played in the stadium ID.

Bool playerIsWinner(Int playerID, Int matchID) – we select from ScoreIn table where PlayerID = playerID and Amount ≥ half of the amount of goals scored in the match, we use nested query to amount of goals scored in the match, using SUM.

List<Int> getActiveTallTeams() – we select from the view ActiveTallTeams order, and then limit by 5.

List<Int> getActiveTallRichTeams() – we select we select from the view ActiveTallTeams where the team stadium is bigger than 55,000. We use nested query in Stadium to find the teams stadium is bigger than 55,000.

List<Int> popularTeams() – we select the distinct values from the union between PopularHomeTeams and NotHomeTeams, than order.

Advenced API

List<Int> getMostAttractiveStadiums() – we use nested query in as an attribute. We select StaduimID form Stadium and for each one we calculate the Amount of Goals which scored in the stadium.

List<Int> mostGoalsForTeam(Int teamID) – we use nested query in as an attribute. We select PlayerID from Player and for each one we calculate the number of goals he scored in the table ScoreIn. We use another nested query to choose only from the players where TeamID = teamID in the table Player.

List<Int> getClosePlayers (Int playerID) –

X = all the games playerID scored in selected from the table ScoreIn,

Y = all the games PlayerID scored in selected from the table ScoreIn,

Z = number of games in the X intersect Y

We select PlayerID from Player where (Z>COUNT(X)/2) and PlayerID != playerID.