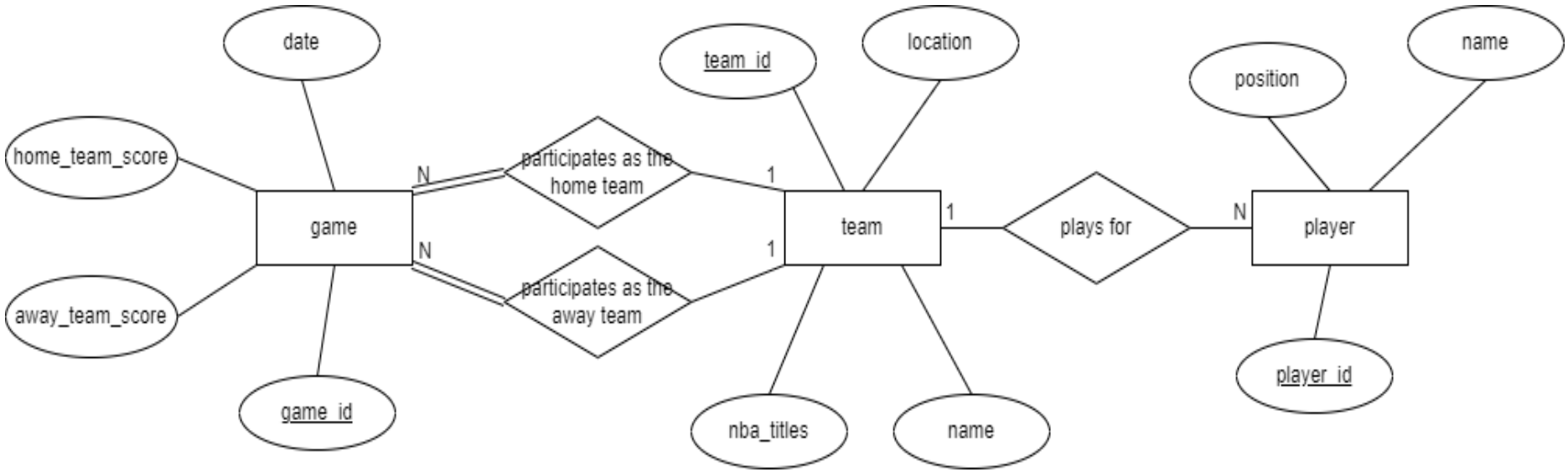
Coursework for LZSCC201 23-24

Designing My Database

Student ID: 38880059

Entity-Relationship diagram



Entity-Relationship diagram description

The diagram represents a very simplistic version of an NBA database with entities: game, team, and player. Each game has a primary key: ‘game_id’, and attributes: ‘date’ (when the game is played), ‘home_team_score’ and ‘away_team_score’ (the amount of points these teams got in the game). Team entity’s primary key is ‘team_id’, and its attributes: ‘name’, the ‘location’ where the team is based, and the number of ‘nba_titles’ that team has won. Player is identified by ‘player_id’ and it has other attributes: ‘name’ and the ‘position’ where he plays.

There are two “participates in as home team” and “participates in as the away team” similar one-to-many relationships between team and game to show that each team can play multiple games (partial participation), but each game has one home and one away team (total participation). There is also a many-to-one “plays for” relationship between player and team because teams can have multiple players (partial participation), and player has at most one team (partial participation).

Relational schema

- teams(team_id: INT, name: TEXT, location: TEXT, nba_titles: INT, PRIMARY KEY: team_id)

- players(player_id: INT, name: TEXT, position: TEXT, PRIMARY KEY: player_id, Foreign Key: team_id REFERENCING: teams, ON DELETE SET NULL)

- games(game_id: INT, date: TEXT, away_team_score: INT, home_team_score: INT, PRIMARY KEY: game_id, Foreign Key: away_team_id REFERENCING: teams, Foreign Key: home_team_id REFERENCING: teams, away_team_id CANNOT BE NULL, ON DELETE REJECT, home_team_id CANNOT BE NULL, ON DELETE REJECT)

Normal forms

General assumptions

In this DBMS I have made a few assumptions about the tables that are especially important for the normalization report:

- no team can play in more than one game in the same day
- all team names are unique
- every game's location is always the location of the home team playing in that game

Thus, the candidate keys of my tables can be determined:

- in the table 'players' the only candidate key is 'player_id'
    - other attributes can't be candidate keys since theoretically it's possible to have two different players with the same name, same team, and who are playing in the same position
- in table 'teams' the two possible candidate keys are 'team_id' and 'name'
    - while in this table we can uniquely identify rows by the combination of 'location' and 'nba_titles' entities, they do not form a composite candidate key; in the context of 'teams' table there theoretically could be two or more teams from the same location and with the same number of NBA titles.
- in table 'games' the possible candidate keys are 'game_id', a composite key that includes 'date' and 'home_team_id' and another composite key that include 'date' and 'away_team_id'

Normalization

I chose the Boyce - Codd normal form for my tables to minimize redundancy, consequently improving the efficiency of queries and reducing the possibility of insertion, deletion, or update anomalies in the database.

To achieve that, first I needed to get all my tables to the 3rd normal form, and then ensure that for every non-trivial functional dependency X → Y, X is a superkey.

First, I had to achieve the 1st normal form, hence I made sure that:

- all columns in my tables only contain atomic values;
- each table has a primary key to ensure uniqueness of rows;
- each attribute/column has a unique name within its table;
- all data stored in a column is of the same type;
- the order of rows and columns is irrelevant (Date, 2019).

To achieve the 2nd normal form, I had to identify the candidate keys and make sure that there is no non-prime attribute that is functionally dependent on any proper subset of any candidate key of the table. There are no composite keys in 'players' and 'teams' tables, which means the concern of partial dependency doesn't apply to them. In the case of 'games' table, there actually was a violation of the 2NF. For example, this table at first had a column 'location' which depended on the 'home_team_id' (which is a part of a composite candidate key) since the games are held in the home team's location. I decided to remove 'location column from 'games' table and only keep it in 'teams' table. I didn't identify any other such violations where non-primary attribute has partial dependency relationship with any candidate key.

To check whether my tables are in the 3rd normal form, I examined whether there are cases of non-prime attributes determining other non-prime attributes in my tables. No such cases were identified, which allowed me to conclude that the tables adhere to the 3rd normal form.

In the end, to ensure that my relations are in Boyce – Codd normal form, I checked whether in my tables determinants in all non-trivial functional dependencies are superkeys. I didn't identify any case where this would not be true. This led me to the conclusion that my relations are in the Boyce-Codd normal form.

SQL statements to create tables

```
CREATE TABLE players (
player_id integer PRIMARY KEY,
name text NOT NULL,
position text NOT NULL,
team_id integer,
FOREIGN KEY(team_id) REFERENCES teams(team_id) ON DELETE SET NULL
);
```

```
CREATE TABLE teams (
team_id integer PRIMARY KEY,
name text NOT NULL,
location text NOT NULL,
nba_titles integer NOT NULL
);
```

```
CREATE TABLE games (
game_id integer PRIMARY KEY,
date text NOT NULL,
home_team_id integer NOT NULL,
away_team_id integer NOT NULL,
home_team_score integer NOT NULL,
away_team_score integer NOT NULL,
FOREIGN KEY(home_team_id) REFERENCES teams(team_id),
FOREIGN KEY(away_team_id) REFERENCES teams(team_id)
);
```

Note: not explicitly adding ON DELETE REJECT/RESTRICT here because it's the default behaviour.

Queries in relational algebra

Selecting all players whose team has more than 5 NBA titles.

```
SELECT players.name AS player_name, teams.name AS team_name,
teams.nba_titles AS nba_titles
FROM players
INNER JOIN teams ON players.team_id = teams.team_id
WHERE teams.nba_titles > 5
```

$\boldsymbol{\rho}(players\_new(name \rightarrow \text{player\_name}), \boldsymbol{players})$

$\boldsymbol{\rho}(teams\_new\ (name \rightarrow \text{team\_name}), \boldsymbol{teams})$

$$\boldsymbol{\pi}_{\text{player\_name, team\_name, nba\_titles}}\left(\boldsymbol{\sigma}_{\text{nba\_titles} > 5}(\textbf{players\_new} \bowtie \textbf{teams\_new})\right)$$

Selecting all games where the player by the name of Davis Bertans has played.

```
SELECT games.date, home_teams.name AS home_team, away_teams.name AS away_team
FROM games
INNER JOIN teams home_teams ON games.home_team_id = home_teams.team_id
INNER JOIN teams away_teams ON games.away_team_id = away_teams.team_id
INNER JOIN players ON players.team_id = home_teams.team_id
OR players.team_id = away_teams.team_id
WHERE players.name = 'Davis Bertans'
```

$\boldsymbol{\rho}(home\_teams(name \rightarrow \text{home\_name}), \boldsymbol{teams})$

$\boldsymbol{\rho}(away\_teams(name \rightarrow \text{away\_name}), \boldsymbol{teams})$

$\boldsymbol{\rho}(players\_new(name \rightarrow \text{player\_name}), \boldsymbol{players})$

$$\boldsymbol{\pi}_{date,\, home\_name, away\_name}\left(\left(\boldsymbol{\sigma}_{player\_name\, =\, 'Davis\ Bertans'}\left(\boldsymbol{players\_new} \bowtie_{\boldsymbol{team\_id=home\_team\_id}} \left(\boldsymbol{games} \bowtie_{\boldsymbol{hame\_team\_id=team\_id}} (\boldsymbol{home\_teams})\right)\right)\right)\right.$$
$$\left.\cup\left(\boldsymbol{\sigma}_{player\_name\, =\, 'Davis\ Bertans'}\left(\boldsymbol{players\_new} \bowtie_{\boldsymbol{team\_id=away\_team\_id}} \left(\boldsymbol{games} \bowtie_{\boldsymbol{away\_team\_id=team\_id}} (\boldsymbol{away\_teams})\right)\right)\right)\right)$$

The above statement in relational algebra doesn't exactly represent the SQL query, however the output remains the same. The approach taken in relational algebra query is separately selecting games where the player's team played as home team, and combining that part with games, where the player's team played as the away team, using union. However, the SQL approach is first performing joins with the tables and then filtering them by the player's name taking into account both home games and away games. The difference comes from the use of 'OR' operator in SQL which cannot be directly used in conditions in relational algebra.

Queries with grouping and aggregation

Selecting all players whose team has participated in more than one game.

```
SELECT players.name, COUNT(games.game_id) AS games_played
FROM players
INNER JOIN teams on players.team_id = teams.team_id
LEFT JOIN games ON games.home_team_id = teams.team_id
OR games.away_team_id = teams.team_id
GROUP BY players.player_id
HAVING COUNT(games.game_id)>1
```

Teams sorted in descending order by the total amount of points they have accumulated in their home games.

```
SELECT teams.name, SUM(games.home_team_score) AS total_home_points
FROM teams
LEFT JOIN games ON teams.team_id = games.home_team_id
GROUP BY teams.team_id
ORDER BY SUM(games.home_team_score) DESC
```

Top 10 players sorted in descending order by the amount of points their team has accumulated in all games.

```
SELECT players.name AS player, teams.name AS team,
    SUM(games.home_team_score) + SUM(games.away_team_score) AS total_points
FROM players
INNER JOIN teams ON players.team_id = teams.team_id
LEFT JOIN games ON teams.team_id = games.home_team_id OR teams.team_id =
games.away_team_id
GROUP BY players.player_id
ORDER BY total_points DESC
LIMIT 10
```

References

Date, C.J. (2019*) Database Design and Relational Theory: Normal Forms and All That Jazz*. Second edition. [Online]. Berkeley, CA: Apress L. P. [Accessed 21st March 2024]. Available at: https://doi.org/10.1007/978-1-4842-5540-7