CT80A0000 Data-Intensive Systems

Assignment 4 Documentation and explanation of design choices

Räkköläinen Aino

Option A

The Simple Platform for showing overview of Game Development projects of two companies A and B.

For this assignment, I decided to use the same idea I had in basics of database systems course where I designed a fictional relational database for game development companies to manage their game projects and related project teams consisting of employees. However, because for this assignment, there needs to be one relational database and one NoSQL database, I decided that this could be a new platform for combining the databases of two different companies. For instance, companies can have used to be different individual companies in the past but because of company acquisition, they now will need to be combined and working together. However, still each data item will be owned by a specific company because it simplifies the design a little bit.

Company A: Big Game Corporation Oy uses document-based database MongoDB for their projects, project teams and games.

Company B: Small Almost Like Indie Developers Oy has used relational database management system PostgreSQL for handling their game projects because they have had a quite small number of employees, game projects and development teams. Thus, they would have needed to have only one skilled admin to handle all of the operations in their database.

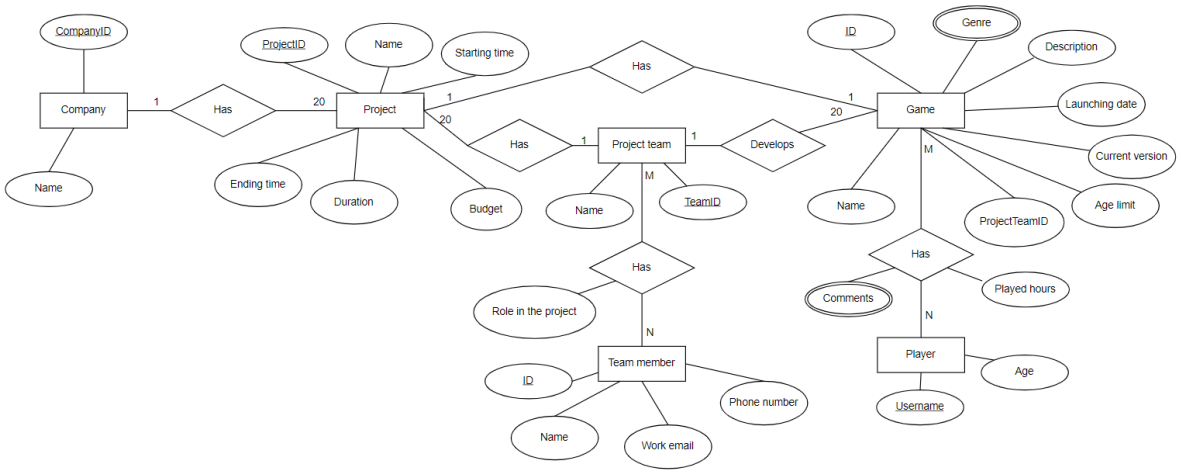

Figure 1. ER model which will be the basis for database of company B.

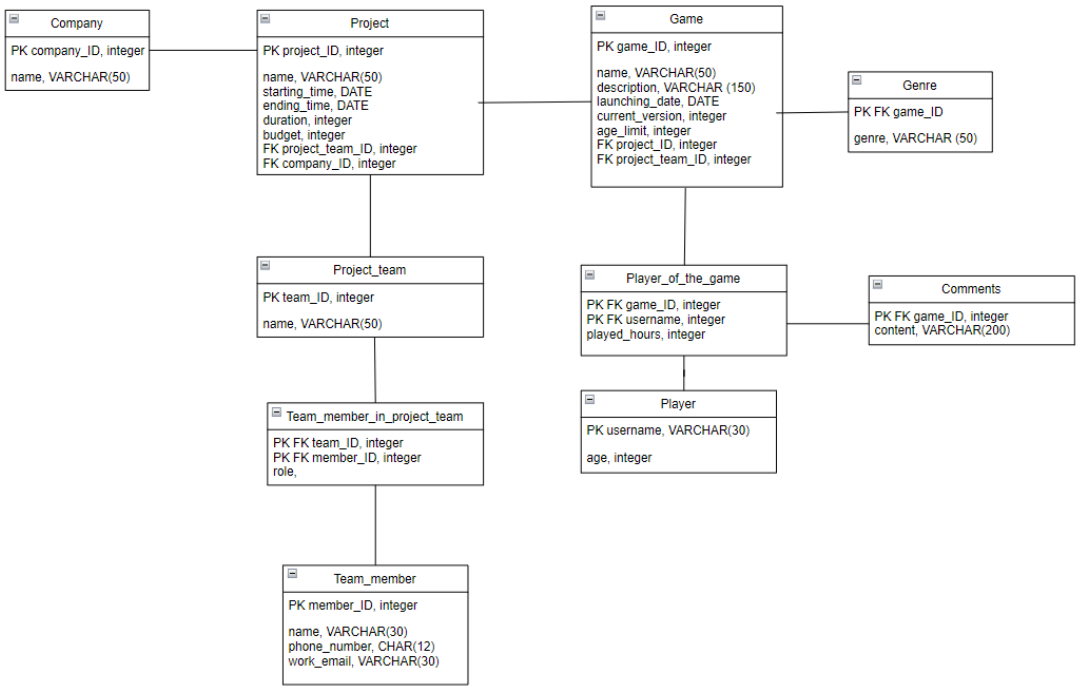

Figure 2. Relational model of previous course

Figures 1 and 2 show the data models I had designed for a previous course, and those were partially basis for this assignment as well. However, while actually implementing the design, I needed to make some exceptions and adjustments:

1) Player data or comments about game are not stored in this database because this is for internal project management only.

2) Additionally, companies will not be their own entities(tables) because there are only two companies and those have separate databases. However, the owner company will be identified with corresponding id either CA (companyA) or CB (companyB). Projects will have id based on the owner company which is mainly defined based on the home company of project manager. If project manager belongs to company A, then the project and the project team they are leading belongs to company A too if the project manager is not changed. The game will be owned by the same company as well.
3) Some adjustments will be made with field names and foreign keys. For instance, the fk_member_id in team_member_in_project table is not actually foreign key because there needs to be stored the ids of team members that are stored in company A db as well. There could probably be other ways to implement this too, but this one seemed logical way to combine project teams with members of both companies without replicating all team members to both databases. Additionally, team_member_in_project holds only the team member id and the team id because only those are needed to query the team_member and project_team tables.
4) Updating project team only updates the team's name and project manager, members are stored in a way that the newest version is the one which gets stored. It is not ideal and one future improvement could be to make it possible to update one member at a time.
5) If game or project team related to project is deleted, those are set to null in project instead of deleting the project itself.

Short conclusion of what requirement each view tries to demonstrate

a) Print out data from both databases separately **(List projects and related games' view)**
b) Print out data from both databases joining tables across databases **(List existing project teams)**
c) Insert/modify data in both databases **(Edit projects, edit games and edit project teams views (video shows only for company A but it works the same way for company B too)**

Note: Updates could also be done between databases in a sense that for project and project team, the project manager can be switched between the companies which means that place where those are stored is changed but only if there is not yet another data item with the same name. That is for avoiding having the item with same name but different attributes in multiple databases. However, this part is not fully functional due to the lack of time so that's why it was left out of demo video as well. Additionally, there is not much replication in this version of the application so the updates could be a little bit simplified.

d) Delete data from both databases **(Delete data view)**

References:

Materialize library was used for UI elements:

https://materializecss.com/getting-started.html

W3Schools was used to check some spelling in JS, HTML and CSS. For example, formatting the dates and strings etc.

https://www.w3schools.com/

MongoDB documentation for updates and deletes:

https://www.mongodb.com/docs/drivers/node/v4.0/usage-examples/updateOne/

https://www.mongodb.com/docs/drivers/node/current/fundamentals/crud/write-operations/delete/

PostgreSQL delete on cascade:

https://geshan.com.np/blog/2023/04/delete-cascade-postgres/