Database Design Document

Game Database Management System

Course: CMPSC 431W
Team Members: Yufeng Zhang, Ivy Qi
Date: 10/31/23
Version: 2.0

CONTENTS

ER Modeling Procedure

ER modeling procedure is used to understand the process of designing the ER diagram.

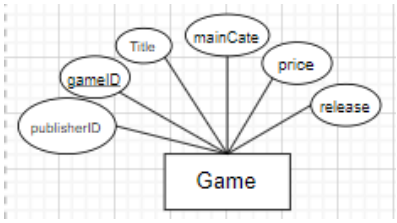
Entity

First step of the ER diagram starts with identifying the entities. Entities are pieces of data that are stored in the database. Here is an example of the Game entity, which will store all the game information, in the ER diagram as shown in Picture 1. Also, we have other entities, such as wishlist, User, Community, Dashboard, Category, Achievement, and Publisher.

Picture 1. A rectangle box around word Game represents a entity

Attributes

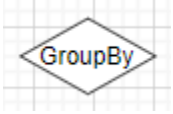
As entities are ready, we need to add attributes, which are the information or data specifically in the entity. In the ER diagram, we connect the attributes to the entity with a line and represent attributes with oval. For example the Game entity has six attributes, for example, gameID, Title, mainCata, price, release, and publisherID. Also, inside the oval, some term will be underlined as the primary key for the entity. Primary key is the key or item that uniquely identifies the entity. For example, the Game entity has unique gameIDs to define the games in the entity.



Picture 2. Game entity with six attributes

Relationship

Then we need to make connections between the entities as relationships. The relationships are identified as diamond shapes shown in Picture 3 and connect two entities. Also, based on the relationship between the entities, the ER diagram will have different lines or arrows to connect them as different types of relationship.

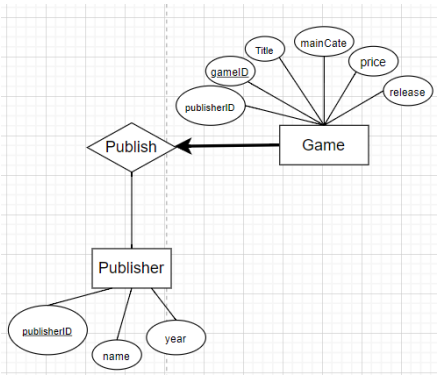


Picture 2. A relationship called GroupBy

Single-value Constraints

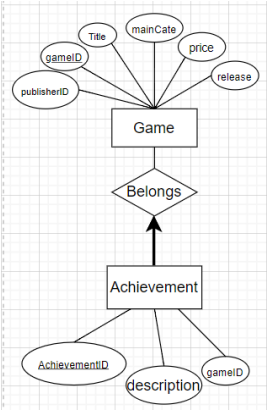
Single-value constraints is a relationship in which each entity must appear at exactly one time in the relationship and is indicated in bold arrow. There are three single-value constraints in our ER diagram as shown below.

First of all, the Game entity with the Publisher entity has a relationship called Publish, meaning that each game is published by a publisher. Each game is published by exactly one publisher, which satisfies the idea of single-value constraints, so the Game entity has a bolded line to the Publish relationship as shown in Picture 3.



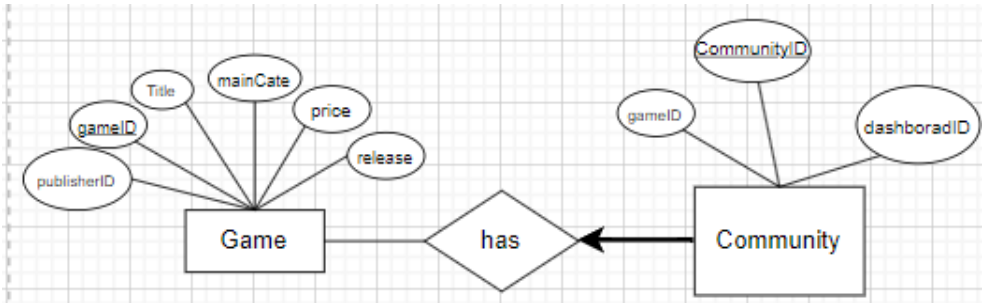
Picture 3. Game entity connect with Publisher entity with a relation called Publish in bolded arrow

We connect the Achievement entity to the Game entity with a relation called Belongs, meaning that each game has some achievements. Since, each achievement belongs to exactly one game, which satisfies the idea of single-value constraints, so the arrow from Achievement to Belongs is bolded as shown in Picture 4.



Picture 4. Achievement entity connect with Game entity with a relation called Belongs in bolded arrow

Then, connect the Community entity to the Game entity with a relation called Has, meaning that a game has some communities for the player to explore. Since Community entity belongs to exactly one game, which satisfies the idea of single-value constraints, it is indicated with a bolded arrow as shown in Picture 5.

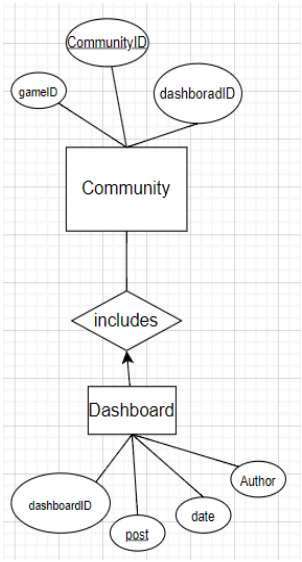


Picture 5. Community entity connects to Game entity with a relationship called Has with a bolded arrow

One to Many

One to many is a relationship that one on the left can associate with many on the right, which is indicated as the right entity has an arrow point into the relationship in the ER diagram. There is one one to many relationship in our ER diagram.

Connect the Community entity to the Dashboard entity with a relation called Includes, meaning that each community includes multiple dashboards that have different contents for the users to explore. Since one community can have more than one dashboard, we used one to many relationships to connect the two entities.

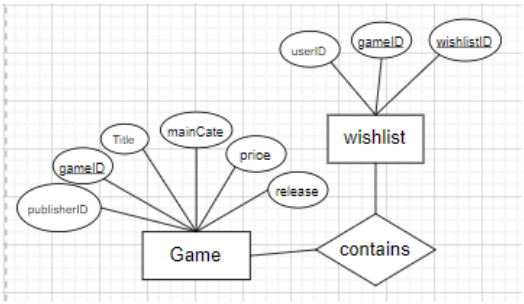


Picture 6. Community entity connects to Dashboard entity with a relationship called Includes with an arrow on the side of Dashboard to include as one to many relationships.

Many to Many

Many to many is a relationship that many on the left can associate with many on the right, which is indicated as straight lines on both sides of the relationship in the ER diagram. There is one many to many in our ER diagram.

We connect Game entity and Wishlist entity with a relation called Contains, meaning wishlist contains games for users to store their wanted games. This is a many to many relationship, since multiple games belongs to multiple wishlists.

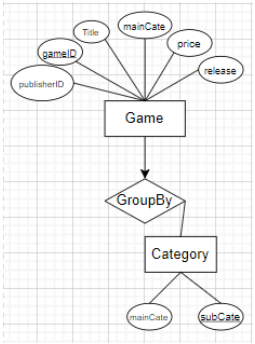


Picture 7. Game entity connects to wishlist entity with a relationship called Contains with straight lines..

Many to One

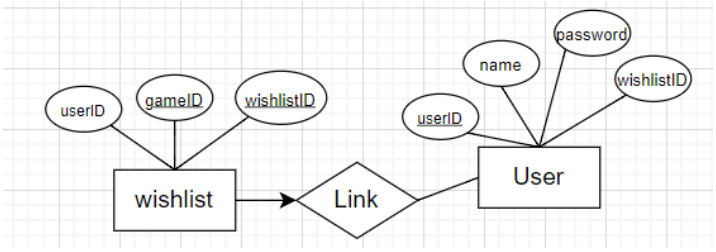
Many to one is opposite to one to many. It is a relationship that many on the left can associate with one on the right and is indicated as the left entity has an arrow point into the relationship in the ER diagram. There are two many to one relationships in our ER diagram.

We connect Game entity and Category entity with a relation called GroupBy meaning that the games are also grouped by the subcategory, and subcategories are under a specific main category. This is a many to one relation between the two entities, since many games can be one subcategory.



Picture 8. Game entity connects to Category entity with a relationship called GroupBy with a arrow on the side of Game to GroupBy as many to one relationship.

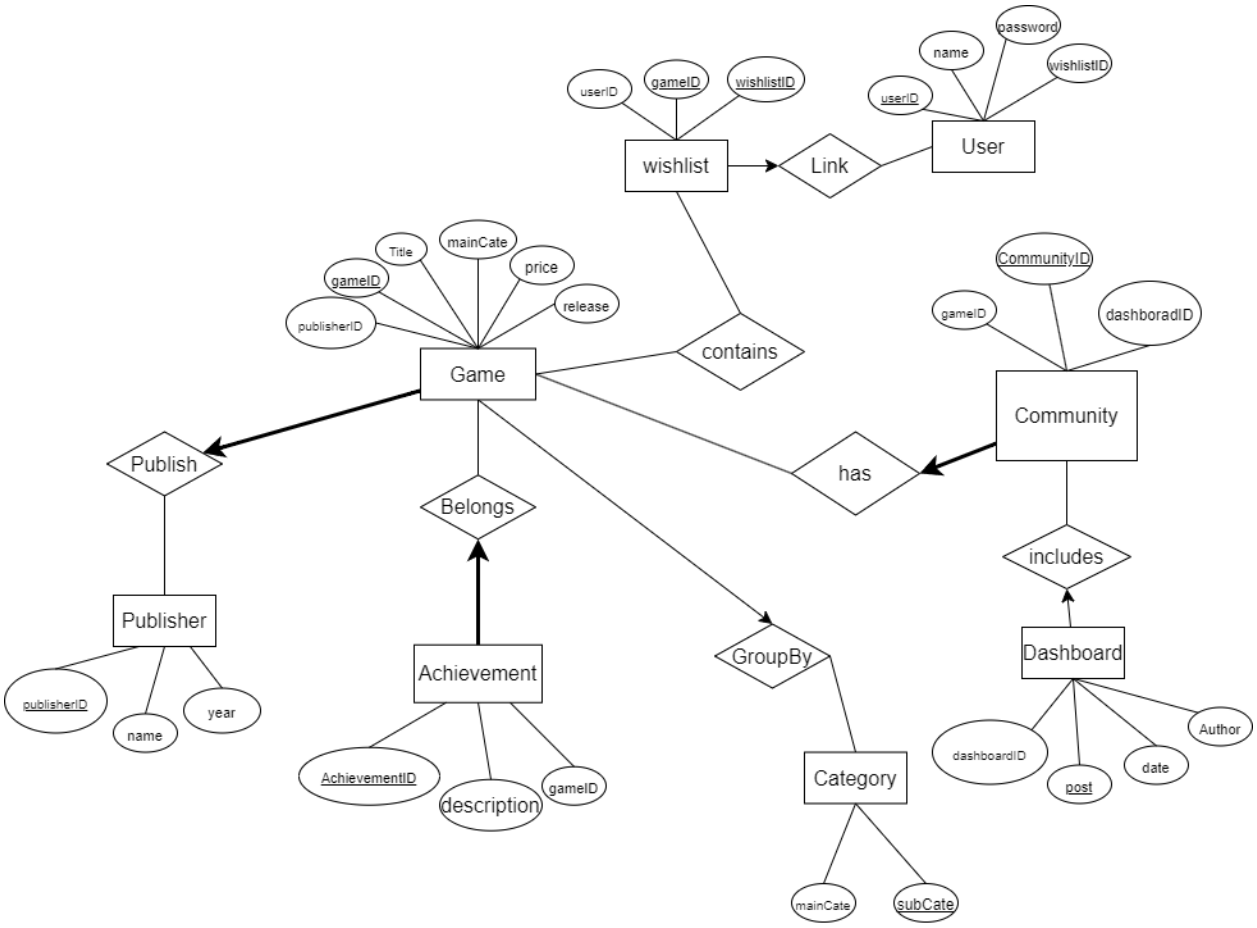
We connect Wishlist entity and User entity with a relation called Link meaning that users have wish lists that contain their wanted games. This is a many to one relation between the two entities, since many wish lists are linked to one user.



Picture 9. Wishlist entity connects to User entity with a relationship called Link with an arrow from wishlist entity to Link relationship

ER Diagram

After the ER modeling procedure, group everything together and we get below as our whole ER Diagram that contains all the entities and their relations (Picture 10):



Picture 10. The final ER diagram

Relational Tables

This step is translating the ER diagram to relational tables in DDL.

```
-- User table
CREATE TABLE User (
  userID INT PRIMARY KEY,
  name VARCHAR(255),
  password VARCHAR(255),
  wishlistID INT
);

-- Game table
CREATE TABLE Game (
  gameID INT PRIMARY KEY,
  Title VARCHAR(255),
  mainCate VARCHAR(255),
  price DECIMAL(10,2),
  release DATE
  publisherID INT,
  FOREIGN KEY (publisherID) REFERENCES Publisher(publisherID)
);

-- Publisher table
CREATE TABLE Publisher (
  publisherID INT PRIMARY KEY,
  name VARCHAR(255),
  year DATE
);

-- Achievement table
CREATE TABLE Achievement (
  achievementID INT PRIMARY KEY,
  gameID INT,
  FOREIGN KEY (gameID) REFERENCES Game(gameID)
);

-- Category table
CREATE TABLE Category (
  mainCate VARCHAR(255) PRIMARY KEY,
  subCate VARCHAR(255)
```

```
);

-- Community table
CREATE TABLE Community (
  communityID INT PRIMARY KEY,
  gameID INT,
  dashboardID INT,
  FOREIGN KEY (gameID) REFERENCES Game(gameID)
);

-- Dashboard table
CREATE TABLE Dashboard (
  dashboardID INT,
  post TEXT PRIMARY KEY,
  Author VARCHAR(255),
  date DATE
);

-- Wishlist table
CREATE TABLE Wishlist (
  wishlistID INT PRIMARY KEY,
  userID INT,
  gameID INT PRIMARY KEY,
  FOREIGN KEY (userID) REFERENCES User(userID),
  FOREIGN KEY (gameID) REFERENCES Game(gameID)
);

------------------ Relationships ------------------

-- Link relationship table
CREATE TABLE Link (
  userID INT,
  gameID INT,
  wishlistID INT,
  FOREIGN KEY (userID) REFERENCES User(userID),
  FOREIGN KEY (gameID) REFERENCES Game(gameID),
  FOREIGN KEY (wishlistID) REFERENCES Wishlist(wishlistID),
  PRIMARY KEY (userID, gameID, wishlistID)
);

-- Belongs relationship table
CREATE TABLE Belongs(
```

```
  achievementID INT,
  gameID INT,
  FOREIGN KEY (achievementID) REFERENCES Achievement(achievementID),
  FOREIGN KEY (gameID) REFERENCES Game(gameID),
  PRIMARY KEY (achievementID, gameID)
);

-- GroupBy relationship table
CREATE TABLE GroupBy (
  gameID INT,
  subCate VARCHAR(255),
  FOREIGN KEY (gameID) REFERENCES Game(gameID),
  FOREIGN KEY (subCate) REFERENCES Category(subCate),
  PRIMARY KEY (gameID, subCate)
);

-- has relationship table
CREATE TABLE has (
  gameID INT,
  CommunityID INT,
  FOREIGN KEY (gameID) REFERENCES Game(gameID),
  FOREIGN KEY (CommunityID) REFERENCES Community(CommunityID),
  PRIMARY KEY (gameID, CommunityID)
);

-- includes relationship table
CREATE TABLE includes (
  CommunityID INT,
  post TEXT,
  FOREIGN KEY (CommunityID) REFERENCES Community(CommunityID),
  FOREIGN KEY (post) REFERENCES Dashboard(post),
  PRIMARY KEY (CommunityID, post)
);

-- contains relationship table
CREATE TABLE contains (
  gameID INT,
  wishlistID INT,
  FOREIGN KEY (gameID) REFERENCES Game(gameID),
  FOREIGN KEY (wishlistID) REFERENCES wishlist(wishlistID),
  PRIMARY KEY (gameID, wishlistID)
);
```

Schema Refinement

Schema refinement is to remove the redundancy in the relational schemas, and at this step, we will check if an entity relation to another entity is in BCNF. In another word, to check if there are bad function dependencies that cannot determine everything.

Entities:

- **Game**
  - Game entity does not have a bad function dependency, which is in BCNF, since every key can determine everything, for example, gameID → Title, mainCate, price, release, publisherID is good function dependency since gameID is a primary key that can determine everything else. Also, Title → maniCata, price, release, publisherID is another good function dependency. GameID, Title as the candidate key can also determine everything as the combination of the two attributes can uniquely identify the Game entity without redundancy.

- **Publisher**
  - Publisher entity does not have a bad function dependency, which is in BCNF, since every key can determine everything. For instance, considering publisherID → name, year, since publisherID is a primary key, it will uniquely identify the entity without redundancy. In addition, name → year can be a good function dependency, since every publisher has a unique name and that can identify everything in the entity.

- **User**
  - User entity is in BCNF as not having bad function dependency. As evidence, Name → password, wishlistID are good function dependencies as names are unique when combined together, which can identify everything. Name, wishlistID → password can also be a good function dependency since these keys are all uniquely identified. Nevertheless, userID → name, password wishlistID is always a good function dependency because the userID is a primary key.

- **Community**
  - Community entity is in BCNF since for all FDs CommunityID->gameID, dashboardID, dashboardID->CommunityID, gameID, CommunityID and dashboardID are super keys that uniquely identify other attributes.

- **Achievement**
  - Achievement entity is in BCNF since all combinations of two in the three attributes can uniquely determine any of the other attributes in the relation.
- **Category**
  - Category entity is in BCNF since subCate -> mainCate is the only FD and subCate is super key.
- **Dashboard**
  - Dashboard entity is not in BCNF as dashboardID → Author has redundancy in the entity. Applying the schema refinement of decomposition, we can separate the table to one as post, Author and date, and another as post, dashboardID, which is shown in table 1, 2, and 3.

dashboardID	post	date	Author
001	post1	10/24/23	John
001	post2	10/26/23	John
002	post3	10/26/23	Mike
002	post4	10/27/23	john

Table 1. An example of Dashboard entity

Author	date	post
John	10/24/23	001
John	10/26/23	002
Mike	10/26/23	003
John	10/27/23	004

Table 2. Separated dashboard entity to author, post, and date

post	DashboardID
001	001
002	001
003	002
004	002

Table 3. Separated dashboard entity to post and dashboardID

  - **The new DDL relational table according to Dashboard:**

```
-- Dashboard table
CREATE TABLE DashboardAuthor (
  post TEXT PRIMARY KEY,
  Author VARCHAR(255),
  date DATE
);

-- Dashboard table
CREATE TABLE DashboardOthers (
  dashboardID INT,
  post TEXT PRIMARY KEY,
);
-- includes relationship table
CREATE TABLE includes (
  CommunityID INT,
  post TEXT,
  FOREIGN KEY (CommunityID) REFERENCES Community(CommunityID),
```

```
  FOREIGN KEY (post) REFERENCES DashboardAuthor(post),
  PRIMARY KEY (CommunityID, post)
);
```

- **Wishlist**
  - Wishlist entity is not in BCNF as it has a bad function dependency of wishlistID → userID since one user has a specific wishlistID, so we can apply schema refinement as decomposition. We need to break the table to two as one for userID and wishlistID and another for wishlistID and gameID as shown in Table 4, 5, and 6. Moreover, the wishlist entity has good function dependency as wishlistID, gameID → userID since wishlistID and gameID can uniquely determine everything in the entity.

WishlistID	gameID	userID
123	Mario	John
123	Zelda	John
456	Mario	Mike
789	Zelda	John

Table 4. An example of wishlist might look like but gameID is not using numbers for clarification in this case

WishlistID	userID
123	John
456	Mike
789	John

Table 5, Wishlist entity divided into wishlistID and userID as a result of schema refinement of decomposition

WishlistID	gameID
123	Mario
123	Zelda
456	Mario
789	Zelda

Table 6. Wishlist entity divided into wishlistID and gameID but gameID used name instead for clarification.

  - **The new DDL relational table according to Dashboard:**

```
-- Wishlist table
CREATE TABLE WishlistUser (
  wishlistID INT PRIMARY KEY,
  userID INT,
  FOREIGN KEY (userID) REFERENCES User(userID),
 );

-- Wishlist table
CREATE TABLE WishlistGame (
  wishlistID INT PRIMARY KEY,
  gameID INT PRIMARY KEY,
  FOREIGN KEY (gameID) REFERENCES Game(gameID)
);
-- contains relationship table
CREATE TABLE contains (
  gameID INT,
  wishlistID INT,
  FOREIGN KEY (gameID) REFERENCES Game(gameID),
```

```
  FOREIGN KEY (wishlistID) REFERENCES WishslistGame(wishlistID),
  PRIMARY KEY (gameID, wishlistID)
);

-- Link relationship table
CREATE TABLE Link (
  userID INT,
  gameID INT,
  wishlistID INT,
  FOREIGN KEY (userID) REFERENCES User(userID),
  FOREIGN KEY (gameID) REFERENCES Game(gameID),
  FOREIGN KEY (wishlistID) REFERENCES WishlistUser(wishlistID),
  PRIMARY KEY (userID, gameID, wishlistID)
);
```

SQL Queries

This step is to write the supporting SQL queries for applications that can be used in this database.

Using ? in the SQL queries to replace the unknown or necessary information that satisfies the left variable of the equation that can be specified by the user or others based on the situation in future usage.

SQL

- List all achievements of a specific game

```
SELECT AchievementID, description
FROM Game, Achievement
WHERE gameID = ?
```

- View all games released by a specified publisher.

```
SELECT gameID, Title, mainCate, price, release
FROM Game
WHERE publisherID = ?
```

- View all games under a specified main category.

```
SELECT gameID, Title, mainCate, price, release
FROM Game
WHERE mainCate = ?
```

- View all games under a specified sub-category.

```
SELECT g.gameID, g.Title, g.mainCat, g.price, g.release
FROM Game g
        JOIN GroupBy gb ON g.gameID = gb.gameID
WHERE gb.subCate = ?
```

- View all sub-categories under a specified main category.

```
SELECT subCat
FROM Category
WHERE mainCat = ?;
```

- View the community for a specified game.

```
SELECT communityID
FROM Community
WHERE gameID = ?;
```

- View all dashboards of a specified game's community.

```
SELECT d.dashboardID, d.post, d.Author, d.date
FROM Game g
        JOIN Community c ON g.gameID = c.gameID
        JOIN includes i ON c.CommunityID = i.CommunityID
        JOIN DashboardAuthor d ON i.post = d.post
WHERE g.gameID = ?;
```

- View all posts on a specified dashboard.

```
SELECT d.post
FROM DashboarOthers
WHERE dashboradID = ?
```

- View all wishlists of a specified user.

```
SELECT wishlistID
FROM WishlistUser w
WHERE userID = ?
```

- View all games under a specified user's wishlist ID.

```
SELECT gameID
FROM User u, WishlistGame w
WHERE u.wishlistID = w.wishlistID
```

- Find the top 20 oldest games.

```
SELECT gameID, Title, mainCate, price, release, publisherID
FROM Game
ORDER BY release DESC
Limit 20
```

- Find the top 20 most expensive games.

```
SELECT gameID, Title, mainCate, price, release, publisherID
```

```
FROM Game
ORDER BY price DESC
Limit 20
```

- Transfer all games from one wishlist to another wishlist for a user.

```
START TRANSACTION
        INSERT INTO WishlistGame (?, gameID)
        // the ? here is indicating the specific wish list
        // that user want to transfer all the games into
                SELECT gameID
                FROM WishlistGame
                WHERE wishlistID = ?
        DELETE WishlistGame
                WHERE wishlistID = ?
COMMIT
```