Project: Game Reviews Database

Team: It Depends
Members: Tyler Do, Juliano Nguyen, Alex Tang, Shanelee Tran
Last updated: March 12, 2018

Table of Contents

---

1) Introduction

This document describes the Games Review Database that will be developed by Team It Depends. Each section will provide a description of the entities, diagrams, queries, tools, and schedule. The purpose of the Games Review Database is to store games, user reviews of games, purchases of games, clubs and their posts, and user profiles.

2) Scenario

Someone is considering to buy a game but don't know if it is good. They go to our website and search the game they are considering to buy in our Games Review Database. They read the rating and reviews for the game they are interested to inform their purchasing decision.

After purchasing a game, the person would like to share their experience with the game they have purchased. They go to our website, search for the game they would like to post their review for and give the game a rating and a short review.

3) Entities

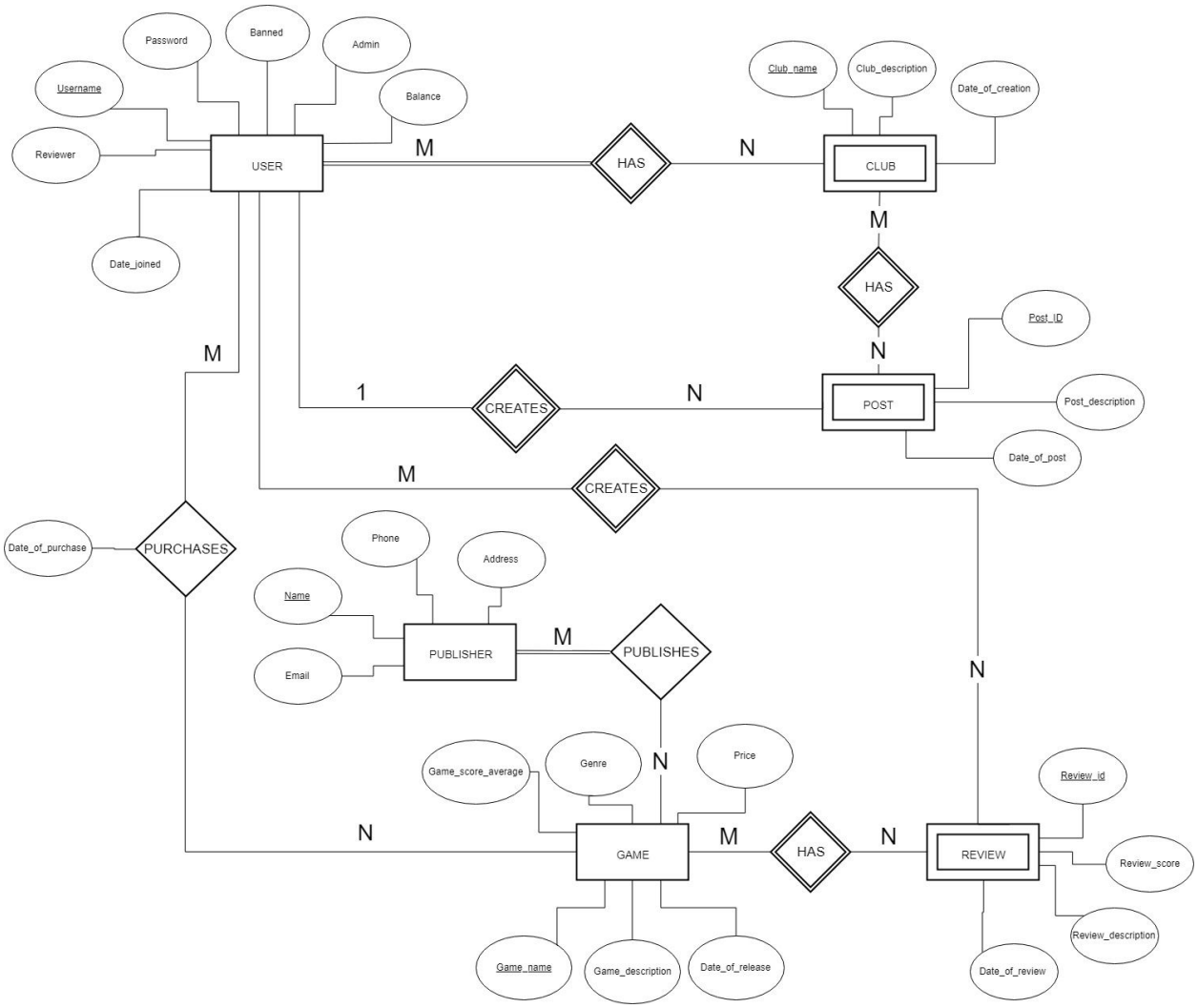
This section provides a list of the entities and their corresponding attributes for the Games Review Database.

- Game
    - Attributes
        - Game_name - string, not null
        - Game_description - string
        - Game_score_average - (0 - 100)
        - Date_of_release - ("01/01/2018")
        - Genre - string
        - Price - decimal
- Review
    - Attributes
        - Review_Id - int, not null
        - Review_description - string
        - Date_of_review - ("01/01/2018")
        - Review_score - int
- User
    - Attributes
        - Username - string, not null
        - Password - string
        - Date_Joined - ("01/01/2018")
        - Reviewer - boolean

        - Admin - boolean
        - Banned - boolean
        - Balance - decimal
- Publisher
    - Attributes
        - Pub_Name - string, not null
        - Pub_Email - string
        - Pub_Address - string
        - Pub_Phone - (XXX-XXX-XXXX)
- Club
    - Attributes
        - Club_Name - string, not null
        - Club_description- string
        - Date_of_creation - date, not null
- Post
    - Attributes
        - Post_ID - int, not null
        - Post_description - string
        - Date_of_post - (“01/01/2018”), not null

4) Entity-Relationship Diagram

Figure 1: ER Diagram for Games Review Database



Introduction

Our Games Review Database is designed to include User, Game, Publisher, Review, Club, and Post. The database contains a list of users, a list of PC games, a list of reviews for each game, a list of publishers that published games, and a list of clubs that contain users. Within a club, users are allowed to make a post for only club members to see. A User can create as many new clubs and can join as many new clubs made by other users. Each club will have an owner and a list of users in that club. A User can create a review for a specific game, which includes their personal description and rating of the game. Additionally, a user can purchase a game for themself if they have enough money in their account; this can hold a list of purchases from all users in the database. Each game will have a publisher, so users can find the publisher, and review and buy games from their interested publisher. A game will also contain their ratings and price, so users can find the best games at a good price.

Description

The Review, Club, and Post are weak entities in our system. The Publisher entity can publish many games. The User can create many reviews for games. The Game includes attributes to define each game. The Review includes reviews about specific games from different users. The User can create posts and join a club. A Club will have posts for the club members to view. Users can purchase as many games and the database will record every user's purchase.

Design Decisions

- We made Review a weak entity because it cannot exist without User and Game entity.
- We made Post and Club weak entities because they cannot exist without User.
- We made Club have an identifying relationship with Post because we want each club to have posts for the club members to view.
- We included Banned and Admin attributes to User to see which users are banned and which users have Admin privileges.
- We included personal attributes for Publisher, like phone, so users can contact them.
- We made Purchases because we wanted users to be able to purchase games.

Limitations

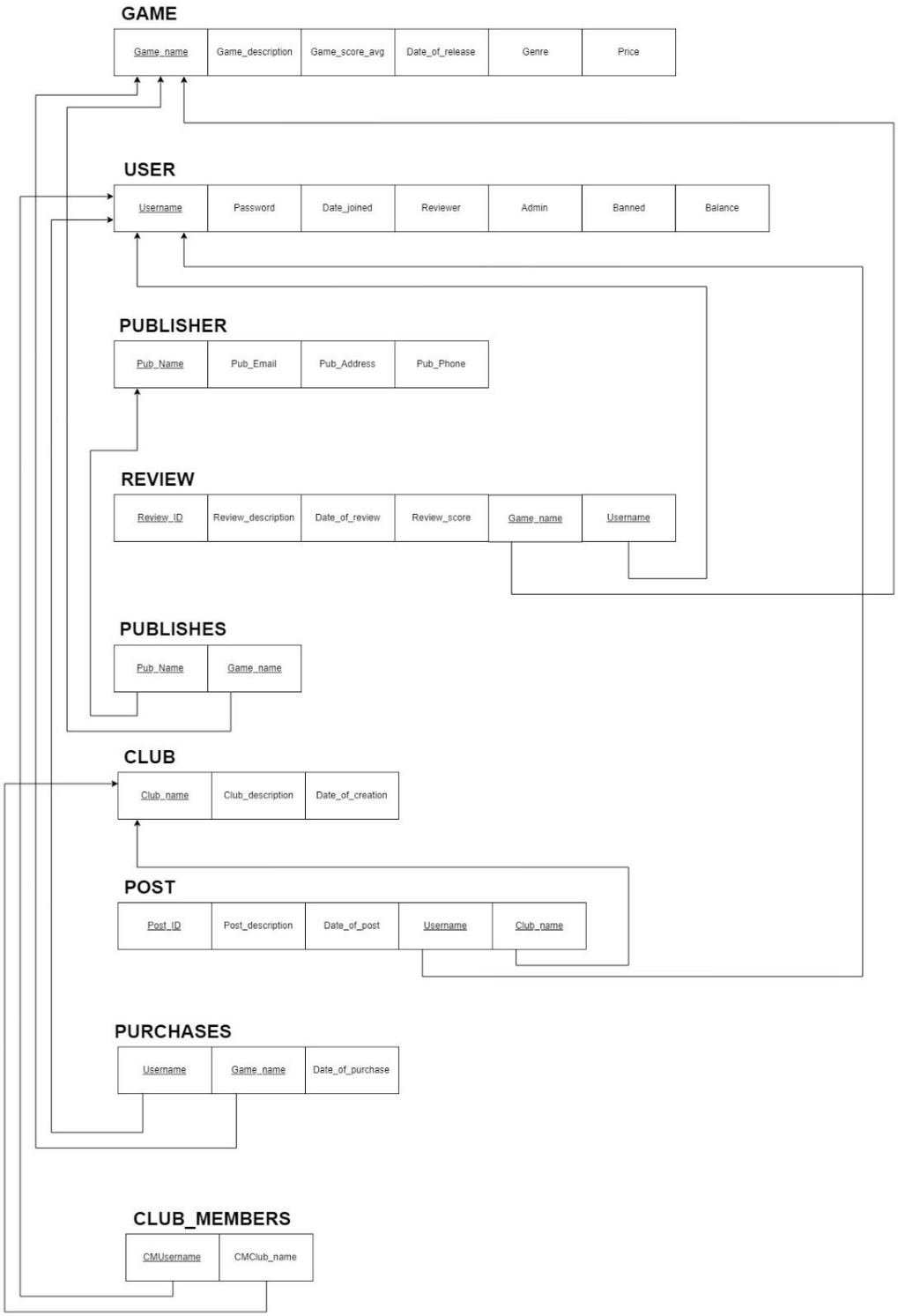
- Does not handle remake of games that have the same title
- Our design does not support games from other platforms
- Our design does not have rankings of games over a period time
- Does not have a suggested games list
- Design does not hold a purchase history for a specific user
- Design does not support refunds
- Design does not support payment transactions
- Design does not support user account creation

Assumptions

- Assume the owner of Review is Game and User
- Assume the owner of Club is User and Post
- Assume the owner of Post is User and Club
- Assume there is only 1 phone number, address, and email for Publisher
- Assume a game cannot exist without a publisher
- Assume user can't buy or create a post/review for a game that doesn't exist
- Assume that the user can't create a post without being in a club
- Assume there is no maximum number of users/clubs/games/reviews/posts

5) Relational Model

Figure 2: RM Schema for Games Review Database



Introduction

The RM for our database contains entities for Game, User, Publisher, Review, Club, Post. It also includes relationships among specific entities, such as Publishes and Purchases.

In Game, the primary key is Game_name, since we are assuming that most game titles are unique. In User, the primary key is Username because the system should not include users that have duplicate usernames. In Publisher, the primary key is Pub_name. In Review, the primary key is Review_ID since there will be multiple reviews from multiple different users; the foreign keys are Game_name and Username. In Club, the primary key is Club_ID to uniquely identify a new club, even if there are similar club names; the foreign keys are Username and Post_ID. In Post, the primary key is the Post_ID since there will be many posts among different clubs; the foreign keys are Username and Club_ID.

The relationship Publishes have the primary keys and foreign keys of Pub_name and Game_name. The relationship Purchases have the primary keys and foreign keys of Username and Game_name.

Description

Table	**Description**
Game [Entity]	Each game includes a title, a description of the game, the average score that users gave, the release date, the genre, and the price.
User [Entity]	Each user includes a username, password, the number of reviews they did, their joined date, and money balance. A user is either a reviewer, admin, or banned.
Publisher [Entity]	Each publisher includes their name, email, address, and phone number.
Review [Entity]	Each review includes an ID, a description from the user, the review date, the score that users gave, the number of views on that review. It will also contain the game name and the username that made it.
Club [Entity]	Each club includes an ID, a club name, the owner of the club, and the number of users. It will also contain a list of usernames and a list of post IDs.
Post [Entity]	Each post includes an ID, a description from the user, and the post's date. It will also

	contain the username and the club ID of where the post belongs.
Publishes [Relationship]	It includes the publisher's name and the game name.
Purchases [Relationship]	Each purchase includes the purchase date. It will also contain the username and game name.
Club_Members [Relationship]	It includes a list of users and their respective clubs they are in.

The relationship "PUBLISHES" is between the Publisher and Game entity. The relationship "PURCHASES" is between the User and Game entity. The relationship "CLUB_MEMBERS" is between the Club and User entity.

Design Decisions

- Review has Review_ID, Game_name, and Username as its primary key because as a weak entity, it gets the primary key from its owners.
- Club has Club_ID, Username, and Post_ID as its primary key because as a weak entity, it gets the primary key from its owners.
- Post has Post_ID, Username, and Club_ID as its primary key because as weak entity, it gets the primary key from its owners.
- We made Club because it adds functionality for the user, so they can create/join a club.
- We made Post because it adds functionality for the user, so they can create a message for club members to see.
- We made "PUBLISHES" because each game needs a publisher and for the user to see their contact information.
- We made "PURCHASES" because it adds another functionality for the user; thus, a user can purchase a game.
- We made "CLUB_MEMBERS" because we wanted a way to keep track of which users were in which club.

Limitations

- Same as from the Entity-Relationship Diagram section
- User and Game entities are associated with many tables.

Assumptions

- Same as from the Entity-Relationship Diagram section
- Assume a publisher's address includes only City, State

Domain Constraints

GAME

Attribute	**Domain Constraint**
Game_name	String; max of 25 characters; not null
Game_description	String; max of 300 characters
Game_score_avg	Integer; between [0 - 100]
Date_of_release	Date; YYYY-MM-DD; not null
Genre	String; max of 10 characters
Price	Decimal; $XX.00; [0 - 100.00]

REVIEW

Attribute	**Domain Constraint**
Review_id	Integer; positive; unique for each review
Review_description	String; max of 50 characters
Date_of_review	Date; YYYY-MM-DD
Review_score	Integer; positive

USER

Attribute	**Domain Constraint**
Username	String; max of 15 characters; not null
Password	String; max of 15 characters; not null
Date_joined	Date; YYYY-MM-DD
Reviewer	Boolean
Admin	Boolean
Banned	Boolean
Balance	Decimal; $XX.00

PUBLISHER

Attribute	Domain Constraint
Pub_Name	String; max of 15 characters
Pub_Email	String; max of 20 characters
Pub_Address	String; can include street, city, state
Pub_Phone	String; 10 digits; only 1 phone

CLUB

Attribute	Domain Constraint
Club_ID	Integer; positive; unique for each club
Club_name	String; max of 15 characters
Club_owner	String; max of 15 characters
No_of_users	Integer; positive

POST

Attribute	Domain Constraint
Post_ID	Integer; positive; unique for each post
Post_description	String; max of 50 characters
Date_of_post	Date; YYYY-MM-DD

Referential Integrity Constraints

- Game is related to Publishes, Review, Purchases
- User is related to Review, Club, Post, Purchases
- Publisher is related to Publishes
- Post is related to Club
- Club_Members is related to User and Club

6) Queries

This section contains a sample of possible queries in English for the database.

The sort functionality supports our application in that a user can find the game(s) they are looking for in an organized way.

- Sort the list of games in the database in alphabetical order.
- Sort the games in the database by rating
- Sort the games by review date
- Sort games by number of reviews
- Sort games by number of views

This functionality supports our application in that a user can view games of their interest/other users/posts in the database.

- List the games for a particular genre
- List the games by a specific publisher
- List the games by a specific letter (ie. games that only start with ‘A’)
- Find a user who purchased a specific game
- List posts by a specific user

7) SQL Statements

This section will contain a list of sample SQL statements that will retrieve relevant information from a table or multiple tables in our database.

SQL Statement	**Purpose**
SELECT Game_name, Review_ID, Review_description, Date_of_review FROM GAME, REVIEW WHERE Game_name = RGame_name AND Price <= 30 ORDER BY Date_of_review;	● This statement will list the games and their reviews for games that have a price less than or equal to $30.00. It will be ordered by the date of review. ● This supports the functionality of listing games and their reviews with a price constraint (e.g. find cheaper games).
SELECT Username, Game_name FROM USER, GAME, PURCHASES WHERE Username = PURUsername AND Game_name = PURGame_name AND Date_of_purchase >= '2017-01-01' AND Date_of_purchase <= '2018-02-01' ORDER BY Date_of_purchase;	● This statement will list the username and game for users that purchased between 01/01/2018 to 02/01/2018. It will be ordered by the date of purchase. ● This supports the functionality of listing user purchases of games within a specific date.
SELECT Game_name FROM GAME WHERE Game_name LIKE 'C%';	● This statement will list all the games that start with the letter 'C'. ● This supports the functionality of listing games by a specific alphabet.

SELECT Post_description, Date_of_post, PUsername FROM POST, USER WHERE PUsername = Username AND PUsername = 'gd_sojin';	● This statement will list the posts whose username is 'gd_sojin'. ● This supports our functionality in seeing posts for a specific user (e.g. inappropriate comments).
SELECT CMUsername FROM CLUB_MEMBERS, CLUB WHERE Club_name = CMClub_name AND CMClub_name = 'Auer and Sons';	● This statement will list all the users in the club named ‘Auer and Sons’. ● This supports our functionality by seeing which users are in a specific club.
SELECT CMClub_name, COUNT(*) FROM CLUB_MEMBERS, CLUB WHERE Club_name = CMClub_name GROUP BY CMClub_name HAVING COUNT(*) > 2;	● This statement will list all of the clubs that have more than 2 users. ● This supports our functionality by seeing which clubs have actively participating users.
SELECT Game_name, AVG(Review_score) FROM GAME, REVIEW, USER WHERE Game_name = RGame_name AND Username = RUsername GROUP BY Game_name HAVING AVG(Review_score) > 50;	● This statement will list all the games that will take the average score from each review of that game. It will display games that have an average greater than 50. ● This supports our functionality in seeing the average score that users have rated for a game instead of a predetermined value.
SELECT Username, Balance FROM USER WHERE Balance > 0 AND Username IN (SELECT PURUsername FROM PURCHASES WHERE Date_of_purchase >= '2018-02-01' AND Date_of_purchase <= CURDATE());	● This statement will list all users and their balance who have made a purchase within the last month and still have a balance. ● This supports our functionality in seeing which users still have a balance, but have also purchased a game in the last month.
SELECT PUsername, Post_description, Date_of_post FROM POST WHERE Date_of_post >= '2018-02-01' AND Date_of_post <= CURDATE() AND	● This statement will list all the user’s posts within the last month. This will be ordered by the date of the post. ● This supports our functionality in seeing which users have posted

PUsername IN (SELECT Username FROM USER) GROUP BY Date_of_post;	within the last month (e.g. finding inappropriate comments).
SELECT PUBPub_Name, PUBGame_name FROM PUBLISHES, PUBLISHER, GAME WHERE Game_name = PUBGame_name AND Pub_Name = PUBPub_Name AND Date_of_release >= '2018-02-01';	• This statement will list games with their respective publisher and was released greater than or equal to 02-01-2018. • This supports our functionality in seeing which publishers have recently published a game.

8) Test Data

The test data that we gathered for the games was from Steam's official website. We used the game's information, such as the price, rating score, release date, and snippets of the game's description. For publisher, we made an assumption to have address include only a street address since we had to work with Mockaroo's templates. In terms of manual test data, we had address to include only City, State.

We will use a combination of manual and the Mockaroo website to generate our test data. We included manual test data for invalid inputs in each table that has a constraint. Mockaroo will be used to generate random test data in each table. We also manually generated a few test data that contain real game information and made-up usernames, reviews, groups, and posts.

The following list contains samples of inserts into our database, where some violate our domain constraints within each table.

Insert 1: Invalid Game Insert

```
INSERT INTO GAME VALUES ('Call of Duty: World at War', 'An intense WWII combat experience', 107, '2008-11-18', 'FPS', 19.99);
```

This is an example of an invalid insert into the Game table. It has a game score value that is over the maximum limit (100).

Insert 2: Invalid User Insert

```
INSERT INTO USER VALUES ('anon123', 'pass', '2018-01-18', True, False, False, -20.00);
```

This is an example of an invalid insert into the User table. It has a negative balance for the user, which must be positive (or even zero).

Insert 3: Invalid Review Insert

```
INSERT INTO REVIEW VALUES (20, 'this game is the worst', '2018-01-28', -50, 'Thief',
'gd_sojin');
```

This is another example of an invalid insert into the Review table. It has a negative game score for the review, which must be positive (or even zero).

Insert 4: Invalid Post Insert

```
INSERT INTO POST VALUES (-11, 'konichiwa', '2018-02-28', 'gd_sojin', 1);
```

This is an example of an invalid insert into the Post table. It has a negative value for the post ID, which must be positive and unique.

Insert 5: Valid Review Insert

```
INSERT INTO REVIEW VALUES (1, 'best game ever', '2018-02-28', 90, 'Call of Duty WWII',
'gd_sojin');
```

This is an example of a valid insert into the Review table. It satisfies the constraints specified.

9) AWS Host Info

The following information was used to connect our MySQL database to our AWS server:

Host: css475.cqgnlgyii9c9.us-west-2.rds.amazonaws.com
Username: tkd8
Password: Test1234
Port: 3306

10) Tooling Assessment

Tools	**Description**
MySQL	This will be compatible when making the database into Windows Form. This will be used to create our tables, insert our sample data, and make our queries. This also can connect the database with Windows Form.
C#	All of us are familiar with the language. We will use Visual Studios to create our user interface for our database.
AWS (Amazon Web Services)	It is flexible, offers customization, and is a 12-month trial. This will host our database so

	that others can access it.
Visual Studio 2015	All of us are familiar with the software and its features. This will be used to create our user interface in Windows Form for our database.
Mockaroo	This will be used to generate random test data for our database. We will also manually input data that are not in Mockaroo.

11) Normalization

GAME

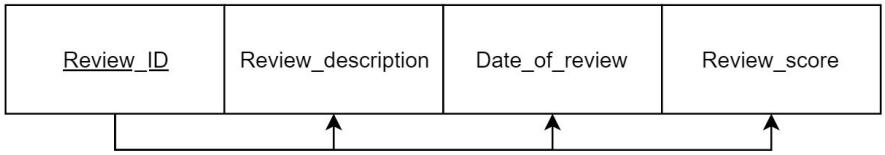
- In 3NF
    - Satisfies 1NF because all attributes are single atomic values
    - Satisfies 2NF because all non-primary key attributes are functionally dependent on Game_name
    - Satisfies 3NF because there is no transitive dependency; X is the super key in FD: {Game_name} → {Game_description, Game_score_average, Date_of_release, Genre, Price}

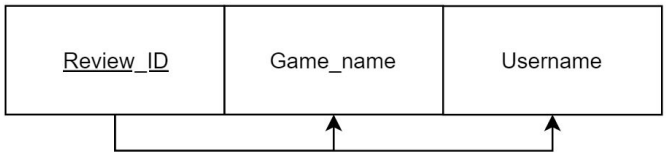
REVIEW

- In 1NF
    - Satisfies 1NF because all attributes are single atomic values
    - To normalize this to 3NF, we would have to split the tables so that the FD are:
        - {Review_ID} → {Review_description, Date_of_review, Review_score}
        - {Review_ID} → {Game_name, Username}

    We made it this way because Game_name and Username do not determine the other non-primary key attributes in the table. We think that deleting a review based on a game name would accidentally delete other review information.

REVIEW





USER

- In 3NF
    - Satisfies 1NF because all attributes are single atomic values
    - Satisfies 2NF because all non-primary keys are functionally dependent on Username
    - Satisfies 3NF because there is no transitive dependency; X is super key in the FD: {Username} → {Password, Date_joined, Reviewer, Admin, Banned, Balance}

PUBLISHER

- In 3NF
    - Satisfies 1NF because all attributes are single atomic values
    - Satisfies 2NF because all non-primary keys are functionally dependent on Pub_Name
    - Satisfies 3NF because there is no transitive dependency; X is super key in the FD: {Pub_name} → {Pub_Email, Pub_Address, Pub_Phone}

CLUB

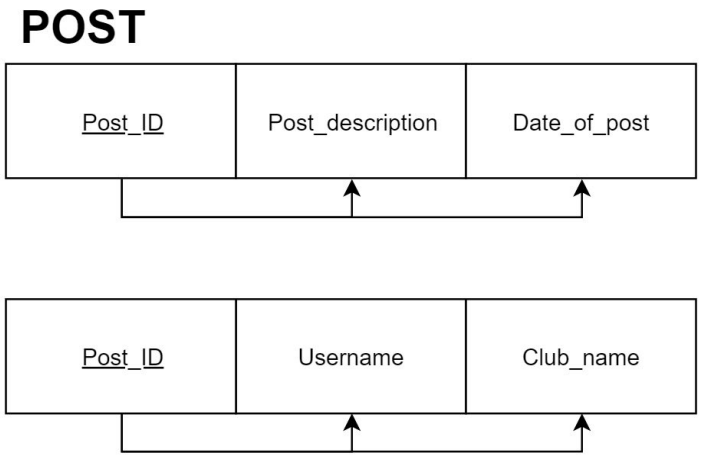
- In 3NF
    - Satisfies 1NF because all attributes are single atomic values
    - Satisfies 2NF because all non-primary key attributes are functionally dependent on Club_name
    - Satisfies 3NF because there is no transitive dependency; X is the super key in the FD: {Club_name} → {Club_description, Date_of_creation}

POST

- In 1NF
    - Satisfies 1NF because all attributes are single atomic values
    - To normalize this to 3NF, we would have to split the tables so that the FD are:

        - {Post_ID} → {Post_description, Date_of_post}
        - {Post_ID} → {Username, Club_name}

    We made it this way because Username and Club_name do not determine the other non-primary key attributes in the table. We think that deleting a post based on a username would accidentally delete other post information.



PUBLISHES

- In 1NF
    - Satisfies 1NF because all attributes are single atomic values
    - To normalize this to 3NF, we would have to make the primary key only Game_name in this table, so that the non- primary key attributes are functionally dependent on Game_name and there is no transitive dependency
        - {Game_name} → {Pub_Name}

    We made it this way because it would not make sense to have a table with both attributes as primary keys.

PURCHASES

- In 3NF
    - Satisfies 1NF because all attributes are single atomic values

- Satisfies 2NF because all non-primary key attributes are functionally dependent on both Username and Game_name
- Satisfies 3NF because there is no transitive dependency; X is super key in the FD: {Username, Game_name} → {Date_of_purchase}

CLUB_MEMBERS

- In 3NF
  - Satisfies 1NF because all attributes are single atomic values
  - Satisfies 2NF because all non-primary key attributes are functionally dependent on CMUsername
  - Satisfies 3NF because there is no transitive dependency; X is the super key in the FD: {CMUsername} → {CMClub_name}

12) Schedule

Iteration	**Description**	**Due Date**
0	Determine the requirements for the database	1/12/2018
1	Design the ER diagram and RM schema V1	1/27/2018
2	Determine developing tool(s) Refine the ER diagram and RM schema V2	2/10/2018
3	Populate the database with sample data	3/1/2018
4	Implement all of the required queries for database Create presentation poster	3/12/2018

13) Work Distribution

All of us will work on each task of the project together. During our meetings, we will be in constant communication on Discord when developing the database (including the document, diagrams, and SQL statements). If need be, we will get together in person if an aspect of the project is too complex to be done through Discord. If work was added to the project independently, the person will inform the team on what was added to the project.

14) Evaluation

Successes

The overall project was developed very well. All group members were able to frequently communicate online each week about what we needed to do. We would inform the entire group if any of us made changes to a particular aspect of the project, such as the document or diagrams. We were able to manually create invalid test data in each of our tables that have a constraint. We planned ahead and added each section to the document one by one throughout the week before the deadline. Some of the team's experience with MySQL and Visual Studios allowed members to teach one another about how to use each software.

Issues

At the beginning of the project, we had difficulty in coming up with more entities and their corresponding attributes for our database. We also had trouble figuring out how to host our database on AWS. However, we looked at online tutorials on how to setup AWS and how to connect our MySQL database to it. We had difficulty in generating our test data on Mockaroo because it was tedious and time consuming. We had to create a CSV file in Mockaroo first, then match the data and create a SQL file on the website. This is so that the foreign keys from each of our tables are linked. However, the majority of the test data from Mockaroo does not correspond to real game names. It also includes nonsense descriptions for games and reviews. We included several manual test data that included real games names and made-up usernames, reviews, clubs, and posts. We used a template of the Windows Form for our UI design and modified it to add our own features to interact with our database. It was difficult understanding the template and creating the tabs for our tables within the Windows Form.

Future Changes

Even though the project went well, there are a few changes that we believe would make it a simpler project experience. At the beginning of the project, we should have chosen relevant and important attributes for some of the entities because we included a few attributes that we think were not of much use, such as Admin in User. For a publisher's address, we should have split it into its own attributes like street, city, and state. If we had more time on the project, we would implement more features in our UI and make it more elegant.

15) References

Mockaroo. Mockaroo, LLC., n.d. Web. 8 Mar. 2018. <https://www.mockaroo.com/>.
Steam. Valve Corporation, n.d. Web. 08 Mar. 2018. <http://store.steampowered.com/>.