Appropriately

Something Your Parents Would Approve Of
Student 1 ID: rrt2850
Student 2 ID: jpm1443: Joe Meyer
Student 3 ID: ba3969 Bexzod Abduvaliev
Student 4 ID: iep8733

April 20, 2023

1 Introduction

Our approach for this project was to carefully read all the requirements for the project. Identify the key entities for the project. Determine what relationships exists between the entities that make sense for out project.

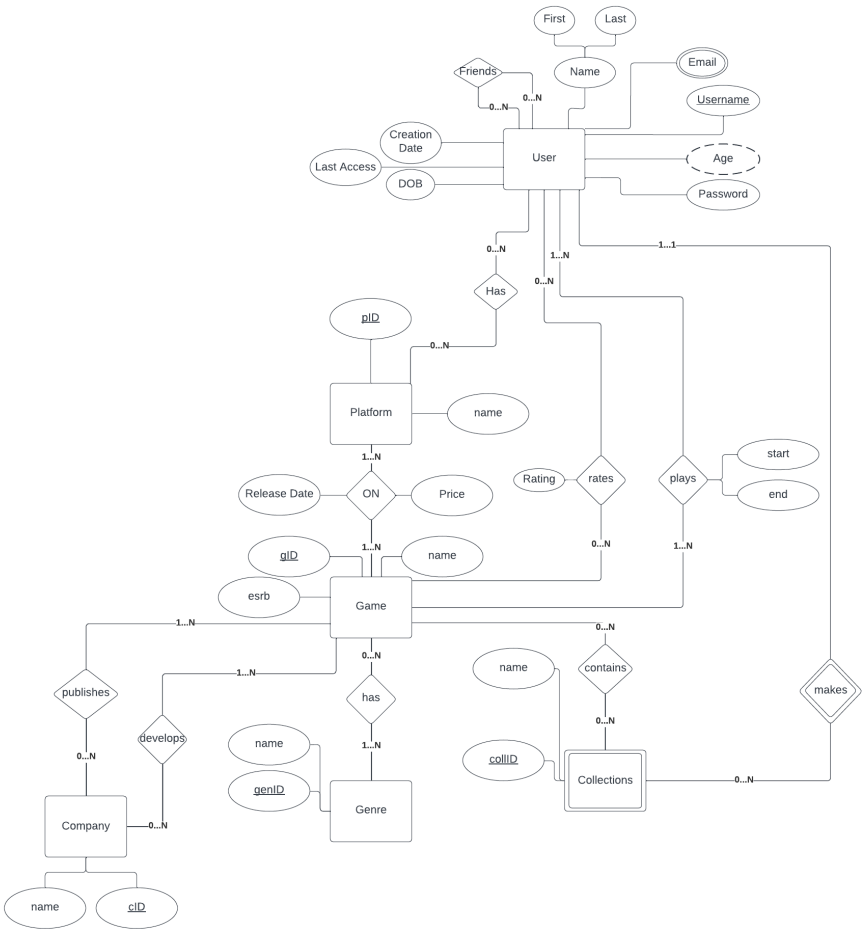
As we learned more and more through out the weeks about the principles of data management, we refined our conceptual model more and more. We started off with prototyping on the whiteboard and then later moved on to make the model in lucidcharts once we felt like we knew enough and were close to a valid conceptual model.

We believe that our primary domain will fit all the requirements laid out in the project description. It includes entities for User, Game, Company, Platform, Genre, and Collections. There are relationships between entitites likes how a User plays a Game or how a Game is developed by a Company and published by a Company to name a few. Our Database application will be a console application written in Python.

2 Design

2.1 Conceptual Model

The ER diagram contains every entity required for the database along with their attributes and relations. The cardinalities are also, of course, included. There were a couple decisions made that deserve explaining. For instance, rather than have two entities, publisher and developer, we grouped them as Company which can either publish or develop a game, which will make it easier to organize later on. Additionally, Collections is a weak entity because it relies on a user to make it.



2.2 Reduction to tables

All tables either represent an entity or a relation between entities. Whether the relations needed their own seperate table depended on the cardinality of the relationship. For instance, collections can only have one user so it made the most sense to have the username stored with the entity data rather than making a new table. Additionally, since a user can have multiple emails it was necessary for there to be a table with a username linked to each email address.

user(<u>username</u>, password, firstname, lastname, creationDate, lastAccessDate, DOB)
platform(<u>platformID</u>, name)
collections(<u>collectionID</u>, <u>*username*</u>, name)
genre(<u>genID</u>, name)
company(<u>cID</u>, name)
game(<u>gID</u>, name, esrb)

emails(<u>*username*</u>, <u>email</u>)
friends(<u>*friendUsername*</u>, <u>*frienderUsername*</u>)
hasPlatform(<u>*username*</u>, <u>*platformID*</u>)
gameOnPlatform(<u>*pID*</u>, <u>*gID*</u>, releaseDate, price)
containsGame(<u>*collID*</u>,<u>*gID*</u>, <u>*username*</u>)
userRates(<u>*username*</u>, <u>*gID*</u>, stars)
userPlays(<u>*username*</u>, <u>*gID*</u>, <u>start</u>, end)
hasGenre(<u>*genID*</u>, <u>*gID*</u>)
publishes(<u>*cID*</u>, <u>*gID*</u>)
develops(<u>*cID*</u>, <u>*gID*</u>)

2.3 Data Requirements/Constraints

- username: VARCHAR(16) NOT NULL UNIQUE
- userPassword: HASHED NOT NULL
- email: VARCHAR(255) NOT NULL
- Unique IDs: INT NOT NULL PRIMARY KEY

- Creation Date, DOB, Release Date, Last Access: DATE
- Start, End: DATETIME
- All names: VARCHAR(32)
- Price: DOUBLE(5, 2)
- esrb: CHAR(3)
- Rating: DOUBLE(3,2)

2.4 Sample instance data

username	password	firstname	lastname	creationDate	lastAccessDate	DOB
johndoe123	password123	John	Doe	2022-01-01	2022-02-15	1990-05-10
janedoe456	ilovecats	Jane	Doe	2022-02-01	2022-02-21	1995-09-03
bobsmith789	123abc	Bob	Smith	2022-02-10	2022-02-22	1985-12-25
maryjones123	password456	Mary	Jones	2022-02-14	2022-02-22	1999-01-20
samlee321	pass345	Sam	Lee	2022-02-20	2022-02-22	2002-06-15

Table 1: Example records for user table

platformID	name
1	PlayStation 5
2	Xbox Series X
3	Nintendo Switch
4	PC
5	PlayStation 4

Table 2: Example records for platform table

collectionID	username	name
1	johndoe123	Favorites
2	janedoe456	Wishlist
3	bobsmith789	Completed Games
4	maryjones123	Multiplayer Games
5	samlee321	Single Player Games

Table 3: Example records for collections table

genID	name
1	Action
2	Adventure
3	Puzzle
4	Role-playing
5	Simulation

Table 4: Example records for genre table

cID	name
1	Electronic Arts
2	Ubisoft
3	Activision
4	Nintendo
5	Square Enix

Table 5: Example records for company table

gID	name	esrb
1	Grand Theft Auto V	M
2	The Legend of Zelda: Breath of the Wild	E10+
3	Portal 2	E10+
4	Final Fantasy VII Remake	T
5	Call of Duty: Modern Warfare	M

Table 6: Example records for game table

username	email
johndoe123	john.doe123@example.com
janedoe456	jane.doe456@example.com
bobsmith789	bob.smith789@example.com
maryjones123	mary.jones123@example.com
samlee321	sam.lee321@example.com

Table 7: Example records for emails table

username1	username2
johndoe123	janedoe456
janedoe456	bobsmith789
bobsmith789	maryjones123
maryjones123	samlee321
samlee321	johndoe123

Table 8: Example records for friends table

username	platformID
johndoe123	1
janedoe456	2
bobsmith789	1
maryjones123	3
samlee321	2

Table 9: Example records for hasPlatform table

pID	gID	releaseDate	price
1	1	2022-01-15	49.99
2	3	2022-02-05	29.99
3	2	2022-02-10	59.99
1	4	2022-02-15	39.99
2	5	2022-02-20	19.99

Table 10: Example records for gameOnPlatform table

collID	gID
1	1
2	2
3	3
4	4
4	5

Table 11: Example records for containsGame table

username	gID	stars
johndoe123	1	4
janedoe456	2	3
bobsmith789	1	5
maryjones123	4	2
samlee321	5	4

Table 12: Example records for userRates table

username	gID	start	end
johndoe	123	2023-01-01 10:00:00	2023-01-01 11:00:00
sarahsmith	456	2023-02-10 18:30:00	2023-02-10 20:00:00
bobross88	789	2023-02-15 14:00:00	2023-02-15 16:00:00
janesmith	456	2023-02-11 19:00:00	2023-02-11 21:00:00
chrisjones	123	2023-02-18 16:00:00	2023-02-18 18:00:00

Table 13: Example records for userPlays table

genID	gID
1	123
2	456
1	789
3	456
2	123

Table 14: Example records for hasGenre table

cID	gID
1	123
2	456
3	789
1	456
2	123

Table 15: Example records for publishes table

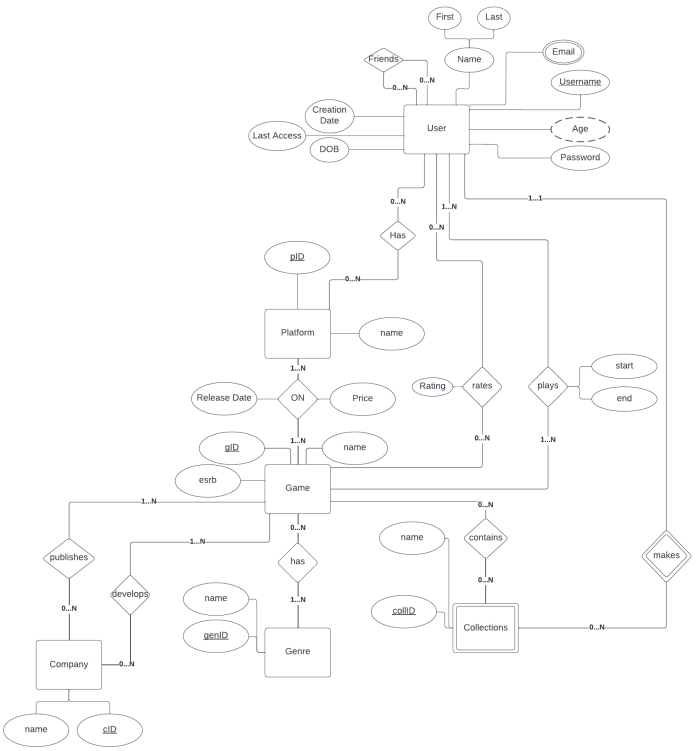
cID	gID
1	123
2	456
3	789
1	456
2	123

Table 16: Example records for develops table

3 Implementation

We utilized our reduction to tables to create the database that was then filled with data from mockaroo. To manipulate the data as a user, there is a console application run with python. The user can do everything they'd want to log and track their video games with a beautifully designed user interface.

3.1 Current Design



3.2 Current Reduction

user(<u>username</u>, password, firstname, lastname, creationDate, lastAccessDate, DOB)
platform(<u>platformID</u>, name)
collections(<u>collectionID</u>, *<u>username</u>*, name)

genre(genID, name)
company(cID, name)
game(gID, name, esrb)

emails(*username*, email)
friends(*friendUsername*, *frienderUsername*)
hasPlatform(*username*, *platformID*)
gameOnPlatform(*pID*, *gID*, releaseDate, price)
containsGame(*collID*,*gID*, *username*)
userRates(*username*, *gID*, stars)
userPlays(*username*, *gID*, start, end)
hasGenre(*genID*, *gID*)
publishes(*cID*, *gID*)
develops(*cID*, *gID*)

3.3 Creating the Tables

Our reduction to tables was relatively accurate and with a few corrections, it was easy to create the tables. For a few crucial entities, here are examples for the SQL commands to make tables:

User: create table "user"
(
username varchar(16)
constraint user_pk primary key,
password varchar(32) not null,
firstname varchar(32) not null,
lastname varchar(32) not null,
creation_date date not null,
last_access_date date not null,
dob date not null
);

Game: create table "game"
(
game_id integer
constraint game_pk primary key,
esrb varchar(3) not null,

name varchar(32) not null,
);

User Rates: create table "userRates"
(
username varchar(16)
constraint "userRates_user_username_fk" references "user",
game_id integer
constraint "userRates_game game_id_fk" references "game",
stars integer generated always as identity (maxvalue 5),
constraint "userRates_pk" primary key (username, game_id)
);

3.4 Inserting in the Program

User:
INSERT INTO p320_13."user" (username, password, firstname, lastname, creationDate, dob, lastaccessdate) VALUES ('james', 'g00dpa55word', 'james', 'james', '2023-03-24', '2001-06-02', '2023-03-24');
INSERT INTO p320_13."user" (username, password, firstname, lastname, creationDate, dob, lastaccessdate) VALUES ('greatname22', 'b3tt3rpa5sw0rd', 'john', 'peeps', '2023-03-24', '2001-8-07', '2023-03-24');

Platform:
INSERT INTO p320_13."platform" (platformID, name) VALUES ('54', 'xbox');
INSERT INTO p320_13."platform" (platformID, name) VALUES ('31', 'wii u');

Collections:
INSERT INTO p320_13."collections" (collection_id, username, name) VALUES ('22', 'james', 'horror');
INSERT INTO p320_13."collections" (collection_id, username, name) VALUES ('23', 'greatname22', 'boring games');

Genre:
INSERT INTO p320_13."genre" (genreID, name) VALUES ('54', 'horror');
INSERT INTO p320_13."genre" (genreID, name) VALUES ('55', 'adventure');

Company:
INSERT INTO p320_13."company" (companyID, name) VALUES ('54', 'microsoft');
INSERT INTO p320_13."genre" (companyID, name) VALUES ('55', 'ubisoft');

Game:
INSERT INTO p320_13."game" (gameID, name, esrb) VALUES ('54', 'satan lovers', 'M');
INSERT INTO p320_13."game" (gameID, name, esrb) VALUES ('55', 'minecraft', 'E');

E-mails:
INSERT INTO p320_13."emails" (username, email) VALUES ('james', 'james@rit.edu');
INSERT INTO p320_13."emails" (username, email) VALUES ('greatname22', 'joebiden@whitehouse.gov');

Friends:
INSERT INTO p320_13."friends" (frienderUsername, friendeeUsername) VALUES ('james', 'greatname22');
INSERT INTO p320_13."friends" (frienderUsername, friendeeUsername) VALUES ('notsecretservice14', 'james');

Has Platform:
INSERT INTO p320_13."hasPlatform" (username, platformID) VALUES ('james', '54');
INSERT INTO p320_13."hasPlatform" (username, platformID) VALUES ('greatname22', '31');

Game On Platform:
INSERT INTO p320_13."gameOnPlatform" (gameID, platformID, releaseDate, price) VALUES ('54', '31', '2023-03-24', '21.98');
INSERT INTO p320_13."gameOnPlatform" (gameID, platformID, releaseDate, price) VALUES ('55', '14', '2023-03-22', '70.98');

Contains Game:
INSERT INTO p320_13."containsGame" (gameID, collectionID, username) VALUES ('54', '31', 'james');

INSERT INTO p320_13."containsGame" (gameID, collectionID, username) VALUES ('20', '18', 'james');

User Rates:
INSERT INTO p320_13."userRate" (gameID, stars, username) VALUES ('54', '3', 'james');
INSERT INTO p320_13."userRate" (gameID, stars, username) VALUES ('55', '5', 'greatname22');

User Plays:
INSERT INTO p320_13."userRate" (gameID, username, start, end) VALUES ('54', 'james', '2023-03-24 11:58:20', '2023-03-24 13:52:31');
INSERT INTO p320_13."userRate" (gameID, username, start, end) VALUES ('54', 'greatname22', '2020-11-10 12:20:41', '2020-11-11 01:14:15');

Has Genre:
INSERT INTO p320_13."hasGenre" (genreID, gameID) VALUES ('10', '54');
INSERT INTO p320_13."hasGenre" (genreID, gameID) VALUES ('8', '54');

Publishes:
INSERT INTO p320_13."publishes" (companyID, gameID) VALUES ('10', '54');
INSERT INTO p320_13."publishes" (companyID, gameID) VALUES ('12', '22');

Develops:
INSERT INTO p320_13."develops" (companyID, gameID) VALUES ('10', '19');
INSERT INTO p320_13."develops" (companyID, gameID) VALUES ('12', '19');

3.5 Loading Data

Data was created using mockaroo where the data types could be manipulated to fit the database. If data was not correct, however, the database would reject data that violated the rules.

Example 1:

```
BULK INSERT "user"
FROM 'C:\Data\UserMockData.csv'
WITH
(
    FIELDTERMINATOR = ',',
    ROWTERMINATOR = '\n',
    FIRSTROW = 2
)
```

Example 2:

```
BULK INSERT "games"
FROM 'C:\Data\GamesMockData.csv'
WITH
(
    FIELDTERMINATOR = ',',
    ROWTERMINATOR = '\n',
    FIRSTROW = 2
)
```

Example 3:

```
BULK INSERT "collections"
FROM 'C:\Data\CollectionsMockData.csv'
WITH
(
    FIELDTERMINATOR = ',',
    ROWTERMINATOR = '\n,
    FIRSTROW = 2
)
```

Example 4:

```
BULK INSERT "genres"
FROM 'C:\Data\GenresMockData.csv'
WITH
(
    FIELDTERMINATOR = ',',
    ROWTERMINATOR = '\n',
    FIRSTROW = 2
```

```
)
```

Example 5:

```
BULK INSERT "companies"
FROM 'C:\Data\CompaniesMockData.csv'
WITH
(
    FIELDTERMINATOR = ',',
    ROWTERMINATOR = '\n',
    FIRSTROW = 2
)
```

3.6 Appendix

3.6.1 Create Account

1. Insertion statement when someone creates a new account:
   INSERT INTO p320_13."user" (username, password, firstname, lastname, creationDate, dob, lastaccessdate) VALUES ('username', 'password', 'firstname', 'last', '2023-03-24', '2001-06-02', '2023-03-24');

3.6.2 Create Collection

2. Select statement to get the last ID of collections so a new ID can be created:
   SELECT MAX(collection_id) FROM p320_13."collections"
3. Insertion statement when someone creates a new collection:
   INSERT INTO p320_13."collections (collection_id, username, name)
   VALUES ('12', 'username', 'collectionName')

3.6.3 View Collection

4. Select statement to get names of users collection:
   SELECT name FROM p320_13."collections"
   WHERE username = 'currentUser'
   ORDER BY name ASC
5. Select statement to get number of games in each collection:

SELECT COUNT('game_id') FROM p320_13."contains_game" AS G
INNER JOIN collections AS C ON G.collection_id = C.collection_id
WHERE C.name = 'collectionName' AND G.username = 'currentUser'

6. Select statement to get playing time in each collection:
SELECT EXTRACT(EPOCH FROM (SUM(C.end - C.start)))
FROM p320_13."contains_game" AS G
INNER JOIN user_plays AS C ON G.game_id = C.game_id
INNER JOIN collections AS CO ON CO.collection_id = G.collection_id
WHERE C.name = 'collectionName' AND G.username = 'currentUser'

3.6.4 Search by Game

7. Select statement for game_id when someone searches by game name:
SELECT * FROM p320_13."game"
WHERE name LIKE ('searchTerm%')
8. Select statement for game_id when someone searches by platform name:
SELECT * FROM p320_13."game_on_platform" AS G
INNER JOIN platform AS P ON G.platform_id = P.platform_id
WHERE P.name LIKE ('searchTerm%')
9. Select statement for game_id when someone searches by release date:
SELECT * FROM p320_13."game_on_platform"
WHERE DATE_PART('day', release_date::timestamp -
'searchTime'::timestamp) = 0
10. Select statement for game_id when someone searches by developer name:
SELECT * FROM p320_13."develops" AS D
INNER JOIN company AS C ON D.company_id = C.company_id
WHERE C.name LIKE ('searchTerm%')
11. Select statement for game_id when someone searches by price:
SELECT * FROM p320_13."game_on_platform"
WHERE price = (searchTerm)
12. Select statement for game_id when someone searches by genre:
SELECT * FROM p320_13."has_genre" AS H
INNER JOIN genre AS G ON H.genre_id = G.genre_id
WHERE G.name LIKE ('searchTerm%')
13. Select statement to get game name, platform names, developer names, publisher names, total time played, ESRB rating, and average star rating from the game_id that came from the prior search:
SELECT g.name, plat.name, c.name, ctwo.name,

EXTRACT(EPOCH FROM (SUM(up.end - up.start))), g.esrb, AVG(ur.stars)
FROM game AS g JOIN game_on_platform as gp ON g.game_id = gp.game_id
JOIN platform AS plat ON gp.platform_id = plat.platform_id
JOIN develops as d ON g.game_id = d.game_id
JOIN company as c ON c.company_id = d.company_id
JOIN publishes as p ON g.game_id = p.game_id
JOIN company as ctwo ON ctwo.company_id = p.company_id
JOIN user_plays as up ON up.game_id = g.game_id
JOIN user_rates as ur ON ur.game_id = g.game_id
JOIN has_genre as hg ON hg.game_id = g.game_id
JOIN genre as gen ON hg.genre_id = gen.genre_id
WHERE g.game_id IN (list of game_ids)
GROUP BY g.game_id, d.game_id, p.game_id, up.game_id, ur.game_id,
gp.game_id, gp.platform_id, plat.platform_id, c.company_id, d.company_id,
ctwo.company_id, p.company_id, g.name, gen.name
ORDER BY g.name, gp.release_date sort_by ASC;

3.6.5 Add to Collection

14. Select statement for game_id after user enters name of game to add to collection:
    SELECT game_id FROM p320_13."game"
    WHERE name IN ('gameName')
15. Select statement to determine if user has a platform the game is on:
    SELECT platform_id FROM has_platform
    WHERE username = 'currentUser'
    INTERSECT SELECT platform_id FROM game_on_platform
    WHERE game_id = gameID
16. Select statement for collection_id after user enters name of collection to be added to:
    SELECT collection_id FROM p320_13."collections"
    WHERE name IN ('collectionName') AND username = 'currentUser'
17. Insertion statement to add the selected game to the selected collection:
    INSERT INTO p320_13."collections" (collection_id, username, name)
    VALUES ('newID', 'currentUser', 'collectionName')

3.6.6 Remove from Collection

18. Select statement to get the game_id of the game the user wants to remove:
SELECT game_id FROM p320_13."game"
WHERE name='gameName'
19. Select statement to get the collection_id of the collection the user wants to remove from:
SELECT collection_id FROM p320_13."collections"
WHERE name='collectionName' AND username='currentUser'
20. Delete statement to remove row where the collection_id and the game_id match with the selected ones:
DELETE FROM p320_13."contains_game"
WHERE collection_id = collectionID AND game_id = gameID
AND username = 'currentUser'

3.6.7 Rename Collection

21. Update statement to change the name of the collection from the provided one to the new one:
UPDATE p320_13."collections" SET name = 'newName'
WHERE name IN ('oldName') AND username = 'currentUser'

3.6.8 Delete Collection

22. Select statement to get the id of the collection the user wants to delete:
SELECT collection_id FROM p320_13."collections"
WHERE name IN ('collectionName') AND username='currentUser'
23. Delete statement to remove the movies in the collection where the collection_id matches with the selected one:
DELETE FROM p320_13."contains_game"
WHERE collection_id = collectionID
24. Delete statement to remove the collection where the collection_id matches with the selected one:
DELETE FROM p320_13."collections"
WHERE collection_id = collectionID

3.6.9 Rate Game

25. Select statement to get game_id of for game user wants to rate:
    SELECT *
    FROM p320_13."game"
    WHERE name = 'Game Name';
26. Select statement to determine if user has already rated the game:
    SELECT username
    FROM (SELECT username FROM user_rates
    WHERE game_id = "gameID") AS FOO
    WHERE username = 'currentUser';
27. Insert statement to add new rating to table because the user has not already rated:
    INSERT INTO p320.13."user_rates" (username, game_id, stars)
    VALUES('currentUser', 'gameID', 'stars')
28. Update statement to update rating for the game the user has already rated:
    UPDATE p320.13."user_rates"
    SET stars = "stars"
    WHERE username IN ('currentUser') AND game_id = "gameID";

3.6.10 Play Game

29. Select statement to get games from collection to choose from:
    SELECT *
    FROM p320_13."collections" as c
    INNER JOIN p320_13."contains_game" as cg
    ON c.collection_id = cg.collection_id
    WHERE c.username = 'currentUser';
30. Select statement to get game_id of the game the user played:
    SELECT *
    FROM p320_13."game"
    WHERE game_id = 'gameId';
31. Insert statement to add the start, end time and username to the game played table:
    INSERT INTO p320_13."user_plays" (username, game_id, start, end)
    VALUES ('currentUser', 'providedGameID', '01-01-2000 12:00:00',

'01-01-2000 1:00:00')

3.6.11 Follow Friend

32. Select statement to get username where friender has already followed friendee:

SELECT friender_username FROM p320_13."friends"
WHERE friender_username = 'currentUser'
AND friendee_username = 'providedUser'

33. Insert statement to add the friends to the friend table if they were not already friends:

INSERT INTO p320_13."friends" (friender_username, friendee_username)
VALUES ('currentUser', 'providedUser')

3.6.12 Unfollow Friend

34. Select statement to get username where friender is already following friendee:

SELECT * FROM p320_13."friends"
WHERE frienderUsername = 'currentUser'
AND friendeeUsername = 'providedUser'

35. Delete statement to remove the friends in the friend table if they were friends:

DELETE FROM p320_13."user"
WHERE frienderUsername = 'currentUser'
AND friendeeUsername = 'providedUser'

3.6.13 Search for Friend

36. Select statement to get username from provided email:

SELECT username FROM p320_13."emails"
WHERE email='searchEmail'

3.6.14 Log In

37. Select statement to get password from the provided username:

SELECT password FROM p320_13."user"
WHERE username='currentUser'

38. Update statement to set user's last access time to current time:

UPDATE p320_13."user" SET lastaccessdate = 'currentDate'
WHERE username = 'currentUser'

3.6.15 Follow Count

39. Select statement to get number of followers from current user:
SELECT COUNT(friendee_username) FROM p320_13."friends"
WHERE friendee_username='currentUser'
40. Select statement to get number of following from current user:
SELECT COUNT(friender_username) FROM p320_13."friends"
WHERE friender_username='currentUser'

3.6.16 Top Ten Games

41. Select statement to get current user's top ten games by rating:
SELECT g.name FROM user_rates as U
INNER JOIN game g on g.game_id = U.game_id
WHERE U.username='currentUser'
GROUP BY U.game_id, U.stars, g.game_id
ORDER BY U.stars DESC
LIMIT 10
42. Select statement to get current user's top ten games by playtime:
SELECT g.name FROM user_rates as U
INNER JOIN game g on g.game_id = U.game_id
WHERE U.username='currentUser'
GROUP BY U.game_id, g.game_id
ORDER BY EXTRACT(EPOCH FROM (SUM(U.end-U.start))) DESC
LIMIT 10
43. Select statement to get current user's top ten games by rating then playtime:
SELECT g.name FROM user_rates as U
INNER JOIN game g on g.game_id = U.game_id
JOIN user_rates r on g.game_id = r.game_id
WHERE U.username='currentUser' AND r.username='currentUser'
GROUP BY U.game_id, g.game_id, r.stars
ORDER BY r.stars DESC,
EXTRACT(EPOCH FROM (SUM(U.end-U.start))) DESC
LIMIT 10

3.6.17 What's Popular

44. Select statement to get top 20 games in last 90 days:
```
SELECT g.name FROM user_plays as U
INNER JOIN game g on g.game_id = U.game_id
WHERE U.end::date >(current_date - 90)
AND U.start::date >(current_date - 90)
GROUP BY U.game_id, g.game_id
ORDER BY EXTRACT(EPOCH FROM (SUM(U.end-U.start))) DESC
LIMIT 20
```
45. Select statement to get top 20 games with friends:
```
SELECT g.name FROM user_plays as U
INNER JOIN game g on g.game_id = U.game_id
INNER JOIN friends f on f.friendee_username = U.username
WHERE f.friendee_username IN
    (SELECT friendee_username FROM friends
    WHERE friender_username = '" + currentUser + "')
AND U.end::date >(current_date - 90)
AND U.start::date >(current_date - 90)
GROUP BY U.game_id, g.game_id
ORDER BY EXTRACT(EPOCH FROM (SUM(U.end-U.start))) DESC
LIMIT 20
```
46. Select statement to get top 5 games of the calendar month:
```
SELECT g.name FROM user_plays as U
INNER JOIN game g on g.game_id = U.game_id
INNER JOIN game_on_platform gop on g.game_id = gop.game_id
WHERE EXTRACT(month from gop.release_date::date) =
    EXTRACT(month from current_date)
AND EXTRACT(year from gop.release_date::date) =
    EXTRACT(year from current_date)
AND EXTRACT(month from U.end::date) =
    EXTRACT(month from current_date)
AND EXTRACT(year from U.end::date) =
    EXTRACT(year from current_date)
GROUP BY U.game_id, g.game_id
ORDER BY EXTRACT(EPOCH FROM (SUM(U.end-U.start))) DESC
LIMIT 5
```

3.6.18 Recommendations

47. Select statement to get top ten games to recommend a user based off their top genres and other users top genres:

```
WITH top_genres AS (
    SELECT g.genre_id FROM user_plays up
    JOIN has_genre g ON up.game_id = g.game_id
    WHERE up.username = 'currentUser'
    GROUP BY g.genre_id
    ORDER BY COUNT(*) DESC
    LIMIT 5
)
, matching_games AS (
    SELECT hg.game_id, COUNT(*) AS matching_genres
    FROM has_genre hg
    JOIN top_genres tg ON hg.genre_id = tg.genre_id
    LEFT JOIN recommendations r ON hg.game_id = r.game_id
    AND r.username = 'currentUser'
    WHERE r.game_id IS NULL
    GROUP BY hg.game_id
    )
SELECT g.name, mg.game_id FROM matching_games mg
JOIN user_plays up ON mg.game_id = up.game_id
JOIN game g ON mg.game_id = g.game_id
WHERE mg.game_id NOT IN (SELECT game_id FROM user_plays
        WHERE username = 'currentUser')
GROUP BY mg.game_id, mg.matching_genres, g.name
ORDER BY mg.matching_genres DESC,
SUM(ABS(EXTRACT(EPOCH FROM (up."end" - up.start)))) DESC
LIMIT 10;
```

4 Data Analysis

4.1 Hypothesis

We hypothesize that the data will show that mature horror games released in October will have higher total play times than mature horror games released in other months. Halloween and October's reputation as spooky will drive

this additional interest in mature horror games.

Additionally, we believe that younger people will play more video games and make more collections than those over 18. Since they are less busy because they don't have jobs, it would make sense that they are playing more games.

4.2 Data Preprocessing

First, we cleaned up the data, which had some major outliers that included play times with the start time after the end time. Since these were clearly errors, we just removed them. We also added more play times to the database to ensure a proper sample size.

The data was extracted using complex SQL statements. To get the data for the horror games we ran this statement to get the time of games played in October:

```
SELECT g.name, EXTRACT(EPOCH FROM (SUM(U.end-U.start)))
FROM user_plays as U
INNER JOIN game g on g.game_id = U.game_id
INNER JOIN game_on_platform gop on g.game_id = gop.game_id
INNER JOIN has_genre hg on g.game_id = hg.game_id
WHERE EXTRACT(month from gop.release_date::date) = 10
AND g.esrb = 'M'
AND hg.genre_id = 5
GROUP BY U.game_id, g.game_id
ORDER BY EXTRACT(EPOCH FROM (SUM(U.end-U.start))) DESC;
```

We ran that statement again with line 6 set to "WHERE EXTRACT(month from gop.release_date::date) != 10" to get the data for mature horror games NOT released in October.

To get the data for the over/under 18 analytics we used simpler SQL statements. For the data on over 18 playtime of video games, we used:

```
SELECT EXTRACT(EPOCH FROM (AVG(U.end-U.start)))
FROM user_plays as U
INNER JOIN "user" as us ON us.username = u.username
WHERE EXTRACT(year from (AGE(current_date, us.dob))) >= 18;
```

For under 18 play time we used the same statement but with the comparison

switched to under 18.

For the data on over 18 use of collections, we used:

SELECT Count(C.collection_id) FROM collections as C
FROM collections as C
JOIN "user" u on u.username = C.username
EXTRACT(year from (AGE(current_date, u.dob))) >= 18;

For under 18 play time we used the same statement but with the comparison switched to under 18. We also counted to total number of people over 18 and under 18 to calculate averages. For the over 18 count we used:

SELECT Count(u.firstname) FROM "user" u
WHERE EXTRACT(year from (AGE(current_date, u.dob))) >= 18;

Under 18 was the same, but with the comparison switched.

4.3 Data Analytics & Visualization

Use this section to explain the process/techniques used to analyze the data, use data visualization to present the results, and explain them.

After being extracted, the data was put into excel to be turned into graphs. For the horror game hypothesis, we created these graphs:

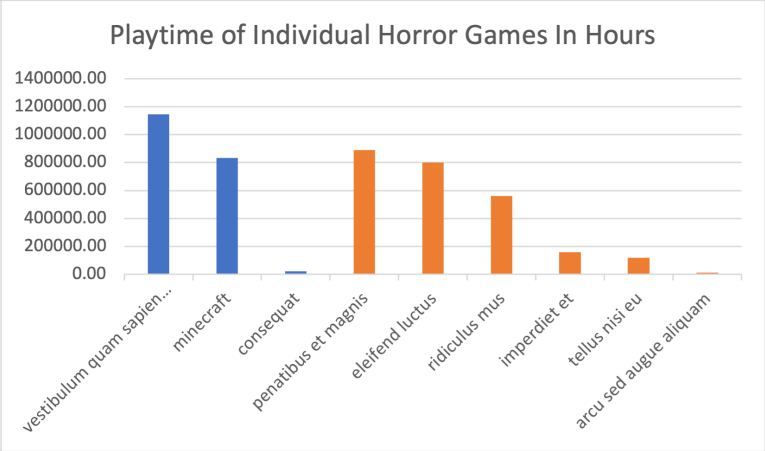


Figure 1

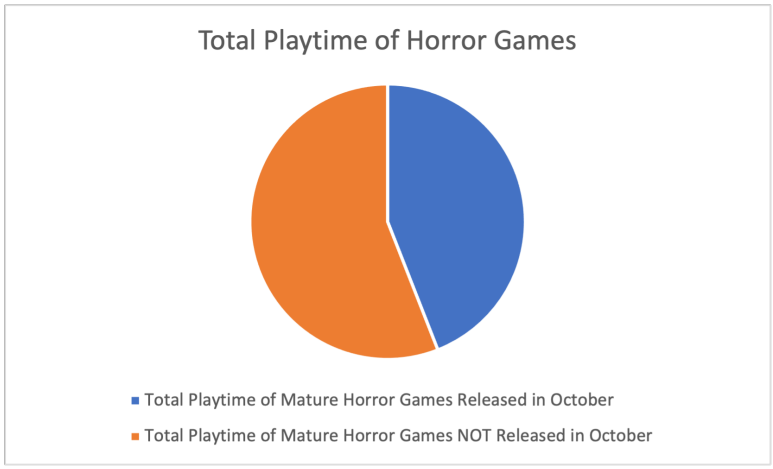


Figure 2

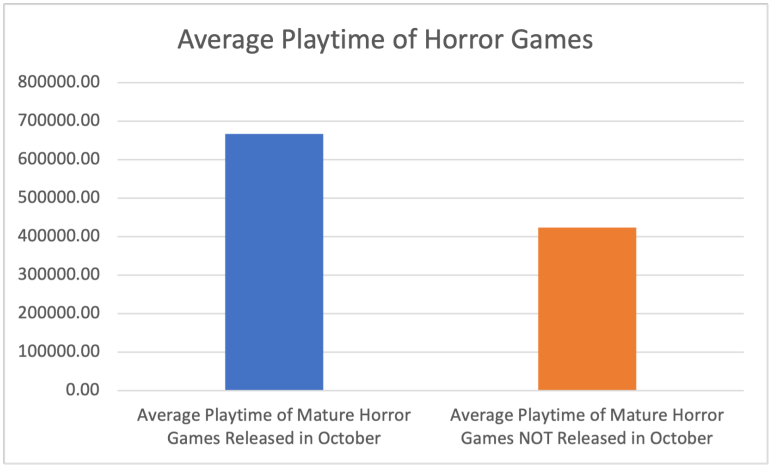


Figure 3

For the over/under 18 playtime comparison this graph was created:

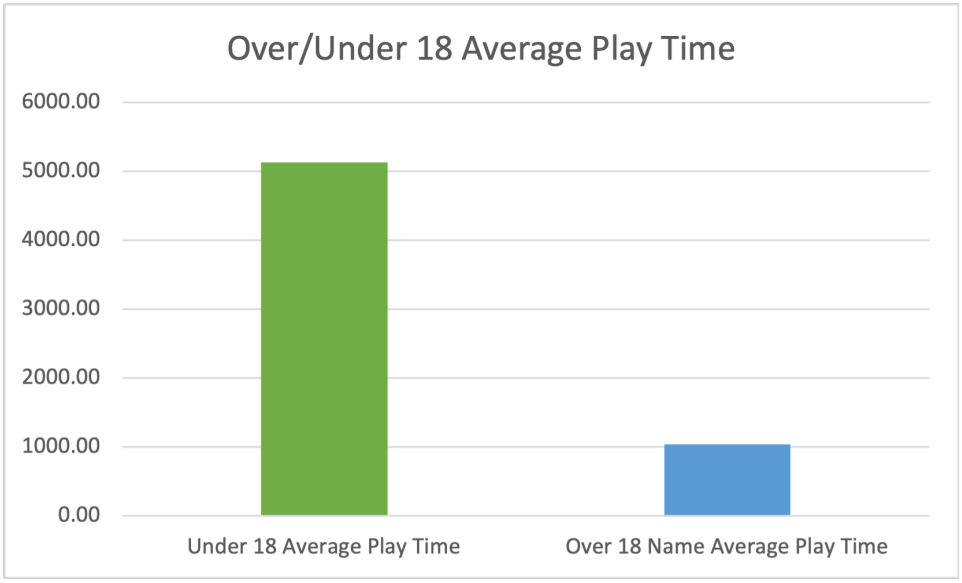


Figure 4

For the over 18 vs under 18 collections comparison these graphs were created:

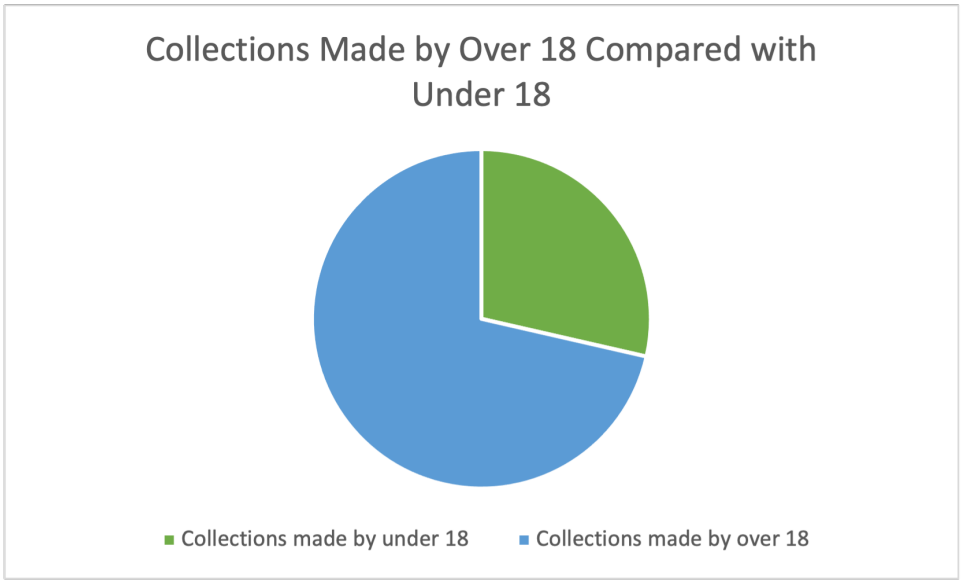


Figure 5

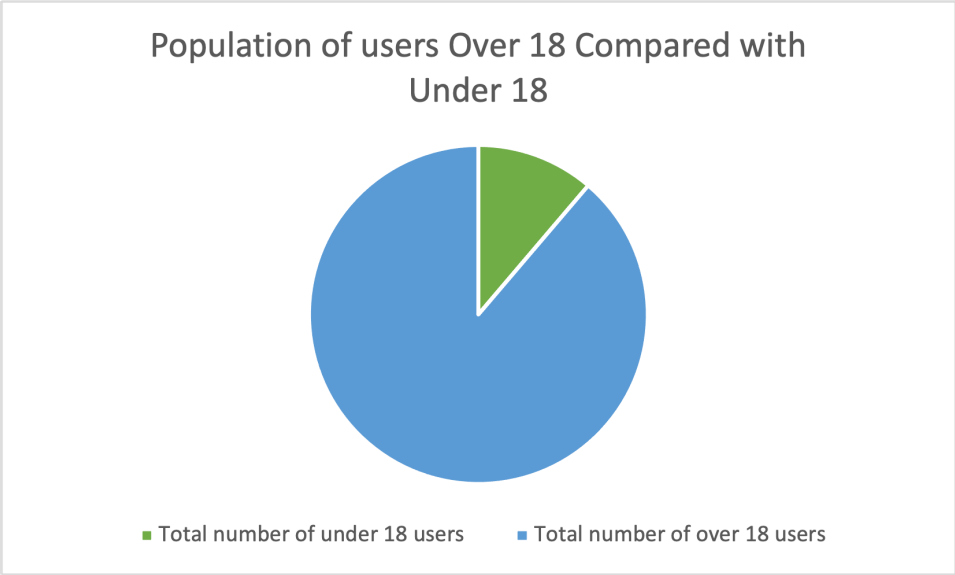


Figure 6

4.4 Conclusions

We concluded that our hypothesis about mature horror games being more popular in October than in other months was CORRECT. There were fewer horror games released in October than in the other eleven months, which makes sense, but they were played more on average. Figure 1 shows each mature horror game's total playtime, with the October releases in blue and the non-October releases in orange. Since it is hard to tell which was more popular with that graph, we used a pie chart to show the total playtime of games released in October vs not in Figure 2. Figure 2 gave us a better understanding of which category was more popular but did not give the full picture because despite having a smaller total playtime, games released in October had a higher average playing time, which was shown in Figure 3. Since Figure 3 clearly shows that mature horror games released in October average about 250k more hours than mature horror games released in other months, our hypothesis was correct. We should start to recommend to publishers and developers to release their mature horror games during October to drive more traffic to their games and our platform.

For the age analytics, we had originally thought that those under-18 would

play more video games than people over 18. After a look at the data, it turns out we were right. Figure 4 clearly shows that the average playtime for under 18-year-olds was significantly higher, over 4000 hours greater, than the average playtime for older users. Also, we learned that under-18 users make more collections than other users. Despite only making up 11% of the users, as shown by Figure 6, Figure 5 shows under-18 users make 29% of collections. Clearly, under 18 users are playing more video games and using our site more. We should sell their data to other companies and recommend them more games to keep them on the site. Would also be a good idea to ask over-18 users to quit their jobs and go professional, which would drastically increase their play times.

5 Lessons Learned

Our biggest problems were in Phase 2, when our grasp of SQL was not great and we didn't fully understand how to connect a python program with the database. The search function, for instance, was a struggle because of all the join and where clauses. We figured it out eventually, however, and by the end, we had a significantly better understanding of SQL.

Additionally, we ran into a few issues with the analytics. According to our bosses, they did not have any interest in analytics about users with first names that started with J vs users whose names started with another letter. This required we go back to the drawing board and develop new areas of research, like age. We eventually developed new graphs, which provided us with double the practice.

6 Resources

We utilized data grips to run the SQL statements. We used excel for the data anyltics and graphs. We used mockaroo for generating the data.