CS-5800
Project Design Report

Console Rental System

The Oasis

Maanav Choubey Aashay Maheshwarkar

Gavin Murdock

Introduction

The created database is for a game rental system. This is intended to meet the needs of gamers who would rather rent games or consoles for a limited time than purchase them. This emphasizes the peer-to-peer game rental system, where the game is rented from one 'user' to another 'gamer' in need.

Keeping up with the latest games is difficult, and gamers may soon find themselves disinterested in what they're playing or wanting to switch to a different platform. Furthermore, gamers may want to test out newly released games before investing in one for their preferred console. In such cases, it is advantageous for the gamer to rent or try the game before purchasing it.

As this system is a peer-to-peer rental system with no intermediaries, low-cost game rentals would be possible. Furthermore, we have witnessed the craze that a new game creates when it is released. People would have to pay much less for such a game using our game rental system.

Requirement Analysis

Data Requirements

1. The Console should have console_id, which would uniquely identify each tuple in the Console table, identifying the console_title, available_count, rented_count, return_date, and rental_date.
2. The Request should have the request_id which would uniquely identify a request containing the console_id and the time in the Request table.
3. The Customer should have a customer_id which would uniquely identify a tuple in the Customer table, identifying the customer_name, customer_phone, customer_email, customer_address.
4. The Concerns table should have the case_id, which would uniquely identify the customer_id, console_id, status (resolved/pending) and the concern.
5. The Renter table should have the renter_id, which would uniquely identify the renter_name, renter_phone, renter_email, renter_address.
6. The Login table would have the usernames and passwords, which would be uniquely identifiable by the username. Furthermore the passwords would be stored in a MD5 format.

Functional Requirements:

There are a lot of software requirements specifications included in the functional requirements of the Gaming Rental System, which contains various process, namely Registration, Check out, Receipt Generation, and Database.

Administrator related requirements:
The system should let the administrator:

- View the user's accounts.
- To delete the users' accounts.
- View concerns about users.
- The administrator also views all products available for rent.
- The administrator can resolve a concern.

Customer Related Requirements

- *Customers Sign-up*: The customer should be able to sign-up to the Console rental system with a username and a password.
- *Assigning an ID to the customers*: The system should assign an unique ID to each customer.
- *Login*: The customer should be able to login with his username and password.
- *Search the desired product*: The system should allow the customer to search for specific consoles.
- *Add his/her contact details*: The system should allow the customer to add his/her phone number, email, and address.
- *Request the amount of time he/she needs that product*: The customer should be able to make a request for an available console for a period of time.
- *Raise concern:* The renter should be able to raise a concern in case of a damaged system returned.

Renter related requirements:

- *Register account:* The renter should be able to sign-up to the Console rental system.
- *Assigning an ID to the renters:* The system should assign an unique ID to each renter.
- *Login*: The renter should be able to login with his username and password.
- *Add his/her contact details*: The system should allow the renter to add his/her phone number, email, and address.
- *Upload the desired product:* The renter must be able to upload his/her consoles to the system.

- *Delete the product:* The renter should be able to delete an existing post in case he is no longer interested in renting the same.
- *Accept the request:* The renter must be able to accept the raised request regarding his/her console.
- *Raise concern:* The renter should be able to raise a concern in case of a damaged system returned.

Console related requirements:

- *Assigning an ID to the Console :*Unique ID should be assigned to a console upon entry into the system.
- *Available Quantity*: The Available quantity of the console should be updated in the system upon each successful return/checkout.
- *Rented Quantity:* The rented quantity should be updated for the console upon each successful checkout/return.
- *Date of Rental:* The date when the console was rented should be updated in the system.
- *Date of Return:* The due date when the console is expected to be returned should be updated in the system.

Conceptual Design

Entities and Attributes

Entity 1: Login

Entity stores the User ID and Passwords of all the users. To make this secure, the password would be stored in a MD5 format.

Attributes:

1. username: The username of a ‘user’ in the Console rental system. Formed once a new user signs up.
2. password: The password of a corresponding ‘user’ in the system.

Relationships:

1. Renter signs up into Login.
2. Customer signs up into Login.

Primary key:

It is defined by username attribute because it is a unique attribute which can identify the tuples of the Login Entity.

Entity 2: Renter

Entity stores the details of all the users who are willing to rent away their console.
Attributes:

1. renter_id: The user_id of a Renter in the Console rental system.
2. renter_name: The name of the Renter.
3. renter_phone: The phone number of the Renter.
4. renter_email: The email address of the Renter.
5. renter_address: The physical address of the Renter.

Relationships:

1. Renter signs up into Login.
2. Renter posts Console: Here, the Renter has the option to post a new Console in the system.
3. Renter raises Concern: Here the Renter raises a concern regarding their Console which was rented to a Customer.

Primary key:
It is defined by renter_id attribute because it is a unique attribute which can identify the tuples of the Renter Entity.

Entity 3: Customer

Entity stores the details of all the Customers who are willing to rent a console from the renting system.
Attributes:

1. customer_id: The user_id of a Customer in the Console rental system.
2. customer_name: The name of the Customer.
3. customer_phone: The phone number of the Customer. 4. customer_email: The email address of the Customer. 5. customer_address: The physical address of the Customer.

Relationships:

1. Customer signs up into Login.
2. Customer requests Console: Here, the Customer has the option to request a new Console from the system.
3. Customer raises Concern: Here the Customer raises a concern regarding a faulty Console which was rented from a Renter.

Primary key:
It is defined by Customer_id attribute because it is a unique attribute which can identify the tuples of the Customer Entity.

Entity 4: Console_Data

Entity stores the details of all the users who are willing to rent away their console.

Attributes:

1. Console_name: It is basically the name of our console (Ex: PS5, Nintendo Switch, etc.)
2. brands: The brand to which our console belongs to.
3. rented_count: The quantity of Console which has been rented.
4. Available_count: The quantity of available consoles in our system.

Relationships:

1. Renter posts Console: Here, the Renter has the option to post a new Console in the system.
2. Customer requests Console: Here the Customer requests for a specific Console from the system.

Primary key:
It is defined by combination of (console_name, brand) attribute because it is a unique attribute which can identify the tuples of the console Entity.

Entity 5: Concerns

Entity stores the details of all the Concerns between the Renters and the Customers and vice versa. Attributes:

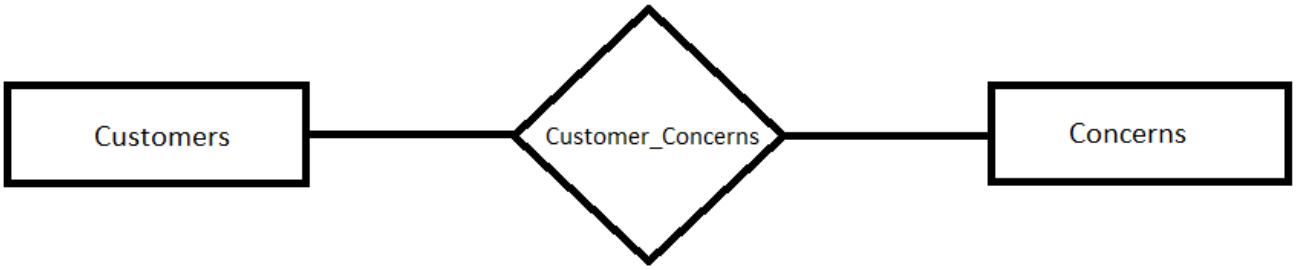
1. Console_id: The console_id for a console regarding which the concern is raised.
2. Case_id: The case_id which uniquely identifies the concern
3. Concern_status: The status of the Concern(open/closed)
4. concern: The message related with a concern
5. customer_id: The customer_id associated with the Concern.
6. Renter_id: The renter_id associated with the Concern.

Relationships:
1. Customer raises Concern: Here the Customer raises a concern regarding a faulty Console which was rented from a Renter.
2. Renter raises Concern: Here the Renter raises Concern for a specific Console which was rented out to a Customer.

Primary key:
It is defined by case_id attribute because it is a unique attribute which can identify the tuples of the Concern Entity.

Relationships

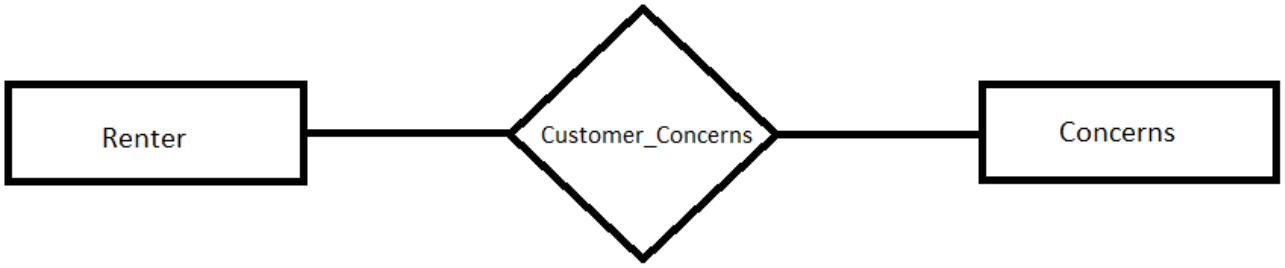
Relationship1: Customer_concerns



Relation: Shows the relation when a Customer raises a concern. Attributes:
Customer_id: Foreign key from Customer. Customer_id of the customer who has raised the concern.

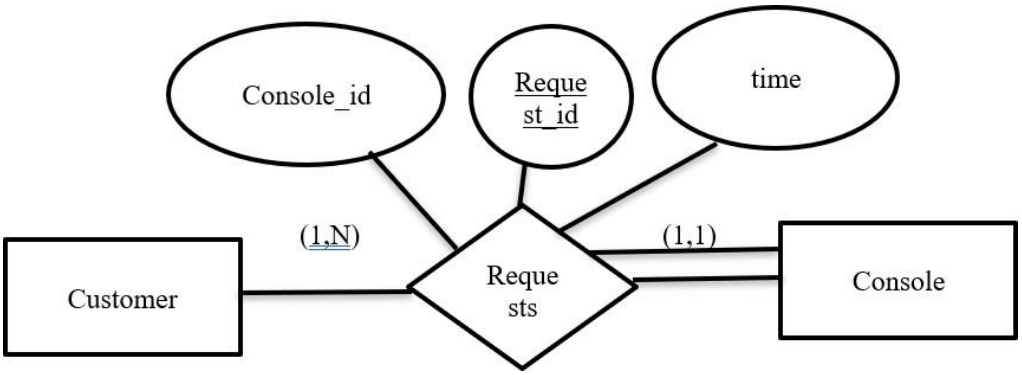
Console_id: Foreign key from Console. Console_id for which the concern has been raised.
Renter_id: Foreign key from Renter. Renter_id of the renter regarding whom the concern has been raised.
Cardinalities:
Customer has a (1,N) cardinality as, each customer can raise N number of concerns.
Concerns has a (1,1) cardinality as each concern can be raised by exactly one customer.

Relationship2: Customer_Concerns



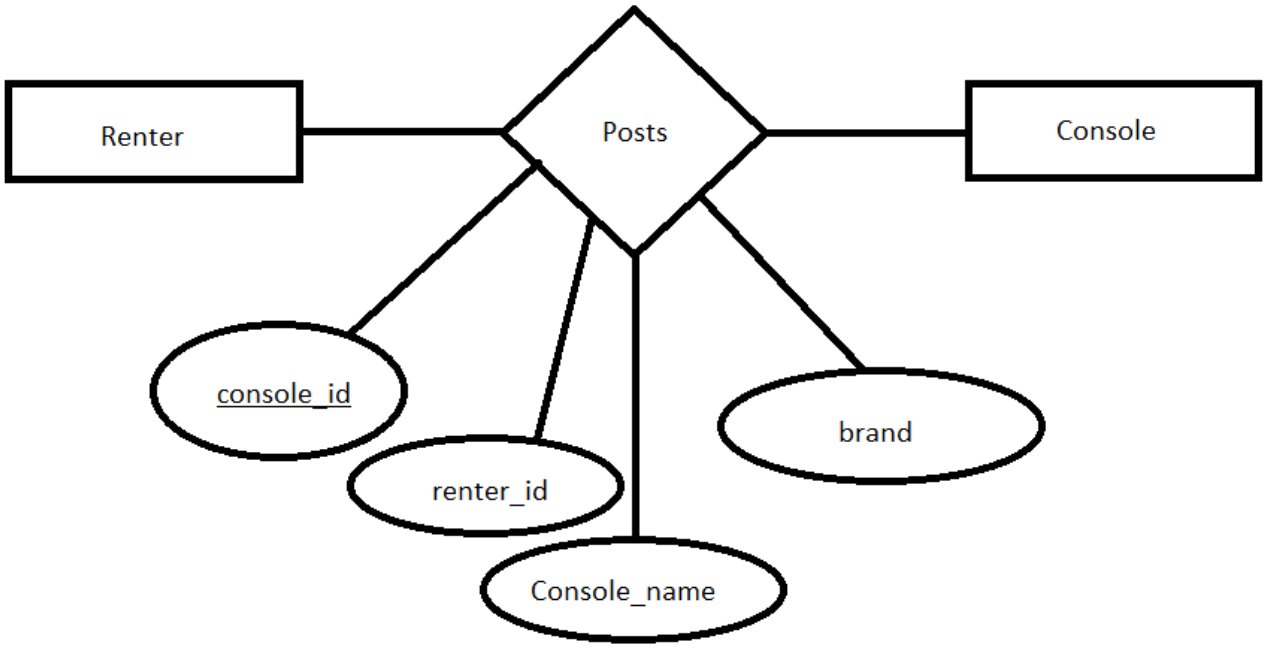
Relation: Shows the relation when a Customer raises a concern. Attributes:
Customer_id: Foreign key from Customer. Customer_id of the customer regarding whom the concern has been raised.
Console_id: Foreign key from Console. Console_id for which the concern has been raised.
Renter_id: Foreign key from Renter. Renter_id of the Renter who has raised the concern
Cardinalities:
Renter has a (1,N) cardinality as each renter can raise N number of concerns.
Concerns has a (1,1) cardinality as each concern can be raised by exactly one customer.

Relationship3: Requests



Relation: Shows the relation when a Customer requests a console. Attributes:
Customer_id: Foreign key from Customer. Customer_id of the customer who has requested the console.
Console_id: Foreign key from Console. Console_id for which the request has been raised
Request_id: Request_id of the uniquely created request
Time: The duration for which the console has been requested Cardinalities:
Customer has a (1,N) cardinality as each customer can raise N number of requests. Console has a (1,1) cardinality as each Console can be requested by exactly one customer.

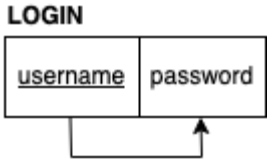
Relationship4: Posts



Relation: Shows the relation when a Renter Posts a Console for renting. Attributes:
renter_id: Foreign key from Renter. Renter_id of the Renter who has posted the console.
Console_id: Primary key for relation. Console_id for the console which has been posted.
Console_name: Name of the console from Console entity.
Brand: Console brand from console entity.
Cardinalities:
Renter has a (1,N) cardinality as, each Renter can raise N number of posts.
Console has a (1,1) cardinality as each console can be posted by exactly one customer.

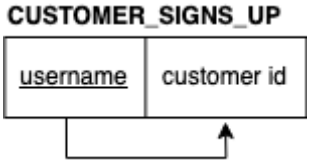
Relational Model

- Relation: LOGIN
  - Functional dependency diagram:



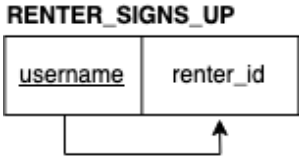
  - Username and password are single atomic values, and password is functionally dependent on the primary key, username. Therefore, Login meets the requirements of 1NF.
  - The only non-prime attribute of Login is password, and it is fully functionally dependent on the primary key, username. Therefore, Login meets the requirements of 2NF.
  - The only non-prime attribute of Login is password, and it is non-transitively dependent on the primary key, username. Therefore, Login meets the requirements of 3NF.

- Relation: CUSTOMER_SIGNS_UP
  - Functional dependency diagram:



  - Username and customer_id are single atomic values, and customer_id is functionally dependent on the primary key, username. Therefore, Customer_Signs_Up meets the requirements of 1NF.
  - The only non-prime attribute of Customer_Signs_Up is customer_id, and it is fully functionally dependent on the primary key, username. Therefore, Customer_Signs_Up meets the requirements of 2NF.
  - The only non-prime attribute of Customer_Signs_Up is customer_id, and it is non-transitively dependent on the primary key, username. Therefore, Customer_Signs_Up meets the requirements of 3NF.

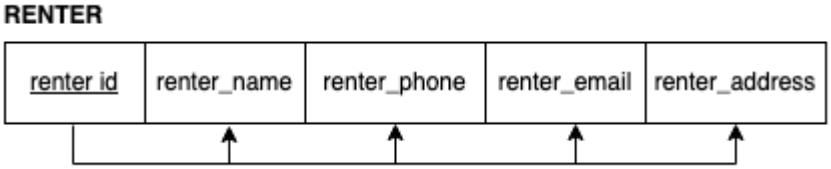
- Relation: RENTER_SIGNS_UP
  - Functional dependency diagram:



  - Username and renter_id are single atomic values, and renter_id is functionally dependent on the primary key, username. Therefore, Renter_Signs_Up meets the requirements of 1NF.
  - The only non-prime attribute of Renter_Signs_Up is renter_id, and it is fully functionally dependent on the primary key, username. Therefore, Renter_Signs_Up meets the requirements of 2NF.
  - The only non-prime attribute of Renter_Signs_Up is renter_id, and it is non-transitively dependent on the primary key, username. Therefore, Renter_Signs_Up meets the requirements of 3NF.

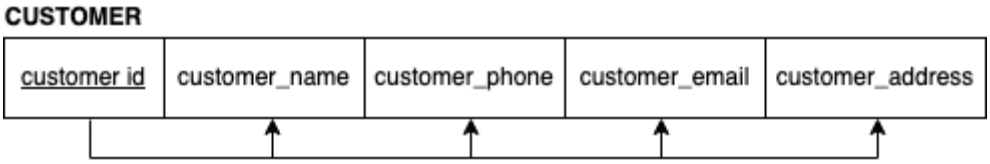
- Relation: RENTER

- Functional dependency diagram:



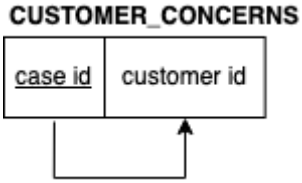
- All attributes of Renter (renter_id, renter_name, renter_phone, renter_email, and renter_address) are single atomic values and are all functionally dependent on the primary key, renter_id. Therefore, Login meets the requirements of 1NF.
- The non-prime attributes of Renter are renter_name, renter_phone, renter_email, and renter_address. They are all fully functionally dependent on the primary key, renter_id. Therefore, Renter meets the requirements of 2NF.
- None of the non-prime attributes in Renter can be determined by another set of non-prime attributes in Renter. Therefore, all the non-prime attributes of Renter are non-transitively dependent on the primary key, renter_id, and the Renter relation meets the requirements of 3NF.

- Relation: CUSTOMER
- Functional dependency diagram:



- All attributes of Customer (customer_id, customer_name, customer_phone, customer_email, and customer_address) are single atomic values and are all functionally dependent on the primary key, customer_id. Therefore, Customer meets the requirements of 1NF.
- The non-prime attributes of Customer are customer_name, customer_phone, customer_email, and customer_address. They are all fully functionally dependent on the primary key, customer_id. Therefore, Customer meets the requirements of 2NF.
- None of the non-prime attributes in Customer can be determined by another set of non-prime attributes in Customer. Therefore, all the non-prime attributes of Customer are non-transitively dependent on the primary key, customer_id, and the Customer relation meets the requirements of 3NF.

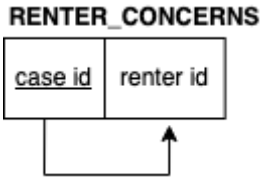
- Relation: CUSTOMER_CONCERNS
- Diagram:



- Case_id and customer_id are single atomic values, and customer_id is functionally dependent on the primary key, case_id. Therefore, Customer_Concerns meets the requirements of 1NF.
- The only non-prime attribute of Customer_Concerns is customer_id, and it is fully functionally dependent on the primary key, case_id. Therefore, Customer_Concerns meets the requirements of 2NF.

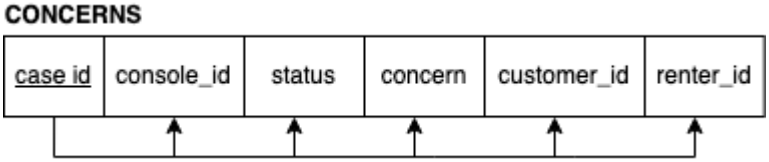
- The only non-prime attribute of Customer_Concerns is customer_id, and it is non-transitively dependent on the primary key, case_id. Therefore, Customer_Concerns meets the requirements of 3NF.

- Relation: RENTER_CONCERNS
- Diagram:



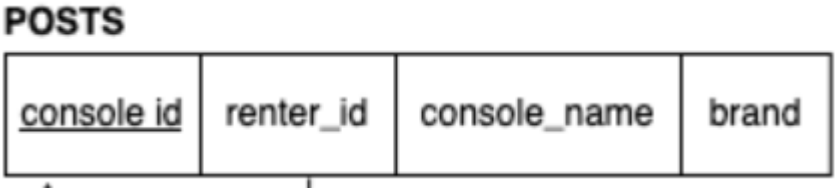
- Case_id and renter_id are single atomic values and have no functional dependencies within this relation. Therefore, Renter_Concerns meets the requirements of 1NF.
- All attributes of this relation are prime attributes and there are no functional dependencies within this relation. Therefore, Renter_Concerns meets the requirements of 2NF and 3NF.

- Relation: CONCERNS
- Functional dependency diagram:



- All attributes of Concerns (case_id, console_id, status, concern, customer_id and renter_id) are single atomic values and are all functionally dependent on the primary key, case_id. Therefore, the Concerns relation meets the requirements of 1NF.
- The non-prime attributes in Concerns are console_id, status, concern, customer_id, and renter_id. They are all fully functionally dependent on the primary key, case_id. Therefore, the Concerns relation meets the requirements of 2NF.
- None of the non-prime attributes in Concerns can be determined by another set of non-prime attributes in Concerns. Therefore, all the non-prime attributes of Concerns are non-transitively dependent on the primary key, case_id, and the Concerns relation meets the requirements of 3NF.

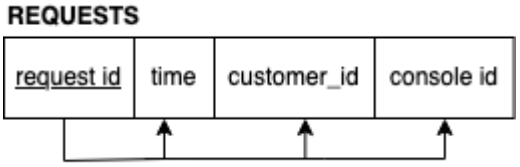
- Relation: POSTS
- Diagram:



- Renter_id, console_id, console_name, brand are single atomic values and have no functional dependencies within this relation. Therefore, the Posts relation meets the requirements of 1NF.
- Console_id is prime attribute and there are no functional dependencies within this relation. Therefore, the Posts relation meets the requirements of 2NF and 3NF.

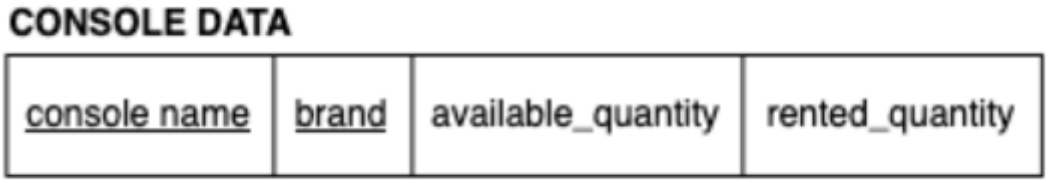
- Relation: REQUESTS

- Functional dependency diagram:



- All attributes of Requests (request_id, time, customer_id, and console_id) are single atomic values and are all functionally dependent on the primary key, request_id. Therefore, the Requests relation meets the requirements of 1NF.
- The non-prime attributes of Requests are time, customer_id, and console_id. They are all fully functionally dependent on the primary key, request_id. Therefore, the Requests relation meets the requirements of 2NF.
- None of the non-prime attributes in Requests can be determined by another set of non-prime attributes in Requests. Therefore, all the non-prime attributes of Requests are non-transitively dependent on the primary key, request_id, and the Requests relation meets the requirements of 3NF.

- Relation: CONSOLE_DATA
- Functional dependency diagram:



- All attributes of Console_DATA (console_name, available_quantity, rented_quantity, and brand) are single atomic values and are all functionally dependent on the primary key, console_name,brand. Therefore, Console_data meets the requirements of 1NF.
- The non-prime attributes of Console_data are available_quantity, rented_quantity. They are all fully functionally dependent on the primary key, console_id. Therefore, Console_data meets the requirements of 2NF.
- None of the non-prime attributes of Console_data can be determined by another set of non-prime attributes of Console_data. Therefore, all the non-prime attributes of Console_data are non-transitively dependent on the primary key, console_name,brand, and the Console_Data relation meets the requirements of 3NF.

Data Dictionary

Following are the Data Dictionary for each table:

- *Concerns*

Field	Type	Null	Key	Default	Extra
case_id	varchar(255)	NO	PRI	NULL	
console_id	varchar(255)	YES	MUL	NULL	
concern_status	varchar(255)	YES		NULL	
concern	varchar(255)	YES		NULL	
customer_id	varchar(255)	YES	MUL	NULL	
renter_id	varchar(255)	YES	MUL	NULL	

Description:

- Case_id: Unique id assigned to each concern.
- Console_id: Foreign key which has id for each console for which a concern is raised.
- Concern_status: Shows if the concern is resolved or unresolved.
- Concern: Comments by the renter or customer defining their issues.
- Customer_id: Foreign key which has id for each customer who raises a concern.
- Renter_id: Foreign key which has id for each renter who raises a concern.

- *Console_availability*

Field	Type	Null	Key	Default	Extra
console_id	varchar(255)	NO	PRI	NULL	
available_flag	tinyint(1)	YES		1	

Description:

- Console_id: Primarykey which has id for each console for which a concern is raised.
- Available_flag: Checks if a console is available or not available.

- *Console_data:*

Field	Type	Null	Key	Default	Extra
console_name	varchar(255)	YES		NULL	
brand	varchar(255)	YES		NULL	
available_count	int	YES		NULL	
rented_count	int	YES		NULL	

Description:

- Console_name: Name of our consoles (Example: Play Station 5, Nintendo Switch, etc.)
- Brand: Tells us the specific brand for every console.
- Available_count: Tells us how many consoles are available to be rented out.
- Rented_count: Tells us how many consoles are already rented.

Customer:

```
340 •  describe customer;
```

Result Grid | Filter Rows: | Export:

Field	Type	Null	Key	Default
customer_id	varchar(255)	NO	PRI	NULL
customer_name	varchar(255)	YES		NULL
customer_phone	varchar(255)	YES		NULL
customer_email	varchar(255)	YES		NULL
customer_address	varchar(255)	YES		NULL

Description:

- Customer_id: Primary key which has id for each customer.
- Customer_name: Name of customer.
- Customer_phone: Stores the phone number of our customer.
- Customer_mail: Saves the mail id for every customer.
- Customer_address: Stores the address of each customer.

- *Customer_concerns:*

```
341 •  describe customer_concerns;
342
```

Result Grid | Filter Rows: | Export:

Field	Type	Null	Key	Default	Extra
case_id	varchar(255)	NO	PRI	NULL	
customer_id	varchar(255)	YES	MUL	NULL	

Description:

- Customer_id: Foreign key which has id for each customer who raises a concern.
- Case_id: Unique id assigned to each concern.

- *Customer_signs_up:*

```
342 •  describe customer_signs_up;
```

Result Grid | Filter Rows: | Expo

Field	Type	Null	Key	Defau
username	varchar(255)	NO	PRI	NULL
customer_id	varchar(255)	YES	UNI	NULL

Description:

- Username: Unique username for every login.
- Customer_id: Has the unique id of each customer.

- *Login*:

```
343 •  describe login;
```

Result Grid | Filter Rows: | Export:

Field	Type	Null	Key	Default
username	varchar(255)	NO	PRI	NULL
password	varchar(255)	YES		NULL

- Username: Unique username of users in our database.
- Password: Password set by a user in MD5 format.

- *Posts*:

```
344 •  describe posts;
```

Result Grid | Filter Rows: | Export:

Field	Type	Null	Key	Default	Ext
console_id	varchar(255)	NO	PRI	NULL	
renter_id	varchar(255)	NO	MUL	NULL	
console_name	varchar(255)	YES		NULL	
brand	varchar(255)	YES		NULL	

- Console_id: unique id for each posted console.
- Renter_id: stores the id of renter who gets the console.
- Console_name: Name of the console which is posted.
- Brand: Consoles brand which is being rented.

- *Renter*:

```
345 •  describe renter;
```

Result Grid | Filter Rows: | Export:

Field	Type	Null	Key	Default
renter_id	varchar(255)	NO	PRI	NULL
renter_name	varchar(255)	YES		NULL
renter_phone	varchar(255)	YES		NULL
renter_email	varchar(255)	YES		NULL
renter_address	varchar(255)	YES		NULL

  - Renter_id: Unique id of each person who rents out a console.
  - Renter_name: Name of the person who rents.
  - Renter_phone: Contact number of person who rents.
  - Renter_mail: Mail id of person who rents.
  - Renter_address: Address of person who rents.

- *Renter_concerns:*

```
346 •  describe renter_concerns;
```

Result Grid | Filter Rows: | Export:

Field	Type	Null	Key	Default	E
case_id	varchar(255)	NO	PRI	NULL	
renter_id	varchar(255)	YES	MUL	NULL	

  - Case_id: unique id which is assigned to each concern raised by a renter.
  - Renter_id: Id of the renter who raises the concern.

- *Renter_signs_up:*

```
347 •  describe renter_signs_up;
```

Result Grid | Filter Rows: | Export:

Field	Type	Null	Key	Default	Extra
username	varchar(255)	NO	PRI	NULL	
renter_id	varchar(255)	YES	UNI	NULL	

  - Username: unique username of each renter who signs up.
  - Renter_id: storing the renter's id who is signing up.

- *Requests*:

```
348 •  describe requests;
```

Result Grid | Filter Rows: | Export:

Field	Type	Null	Key	Default
request_id	varchar(255)	NO	PRI	NULL
request_time	int	NO		NULL
request_status	varchar(255)	YES		NULL
console_id	varchar(255)	NO	MUL	NULL
customer_id	varchar(255)	NO	MUL	NULL

  - Request_id: ID assigned to each new request.
  - Request_time: Time at which a renter makes a request.
  - Request_status: Let’s us know if it has been accepted or not.
  - Console_id: Unique id for each console which is being rented out.
  - Customer_id: Id of the customer making a request.

Teamwork

- Maanav Choubey: Implemented Phase 1 and 3 of the Project, and phase 5 in partnership with Aashay.
- Aashay Maheshwarkar: Implemented the whole Phase 6 and contributed in Phase 5.
- Gavin Murdock: Implemented Phase 2 and 4 of the project. Also, helped in Phase 3.

On a sidenote everyone made a significant contribution in our project and work was equally didvided.

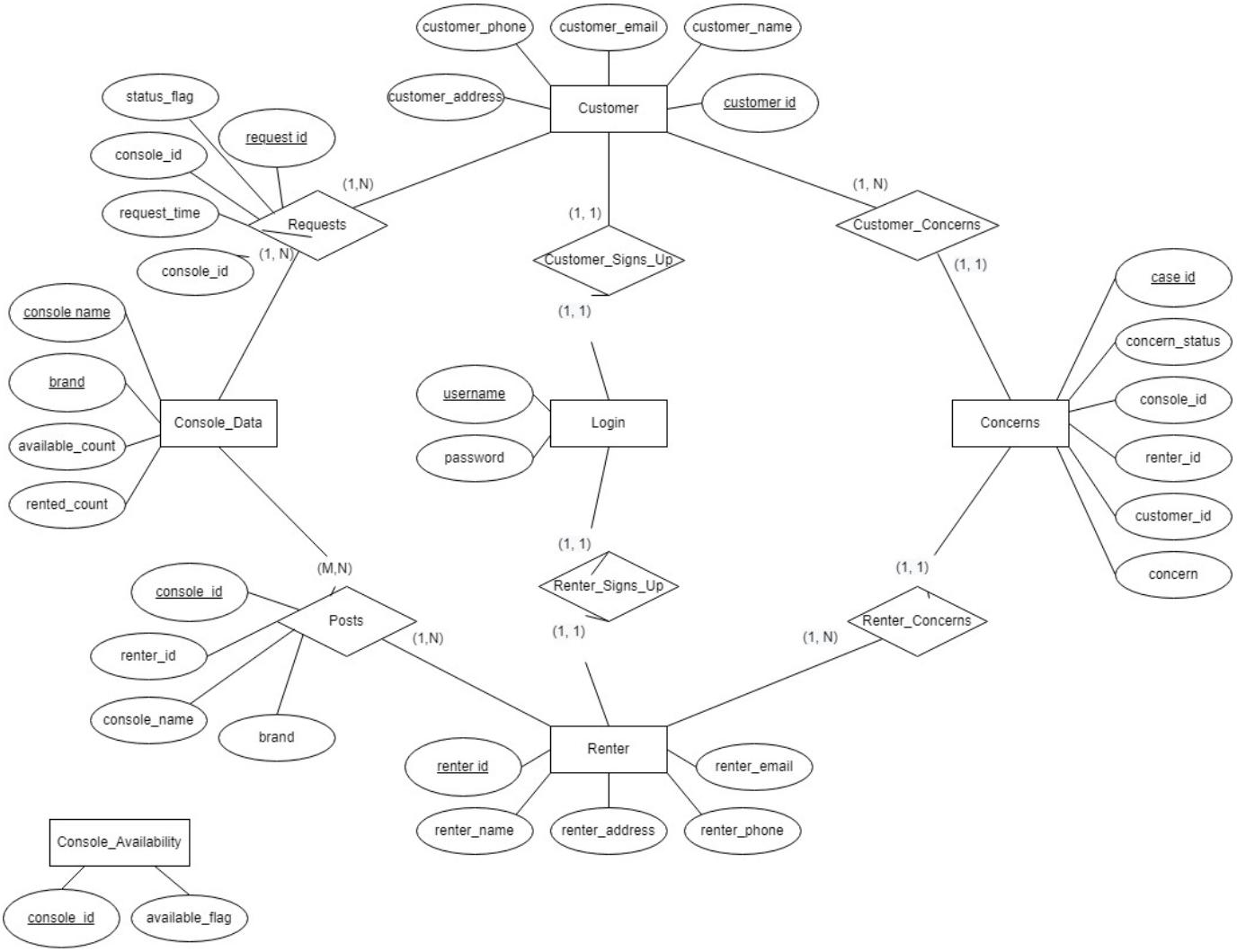
Summary

In this project, we tried to bridge the gap between regular users and people who own a gaming console but are willing to rent it for a certain amount of time. The idea for the project came from the fact that, as students, it's hard to play games when you're moving around, and it's also hard to keep track of your money when you're moving around. So, a system like this would not only help solve this problem, but it would also give people who rent these consoles a lot more options.

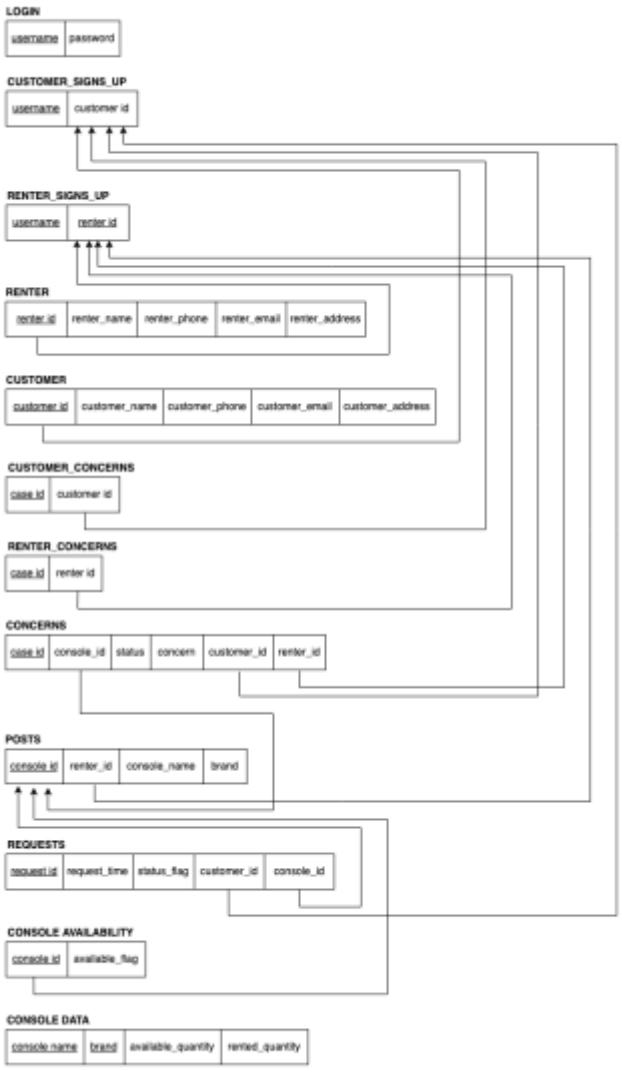
People who have never played video games before can basically rent a system for as long as they want to try out new games and features. This would make their experience better, and it might also help them figure out which consoles are best for them. Our database takes all of these things into account and keeps a list of things and a flow of information. We not only worked on the Customer and Renter parts of the project, but we also thought about how a system like this would work in a real-time app, from how to sign up to how to post a console online to how to keep track of all the consoles and even how a user might have questions or problems.

Appendix A

EER MODEL



RELATIONAL MODEL


LOGIN
username
password
CUSTOMER_SIGNS_UP
username
customer id
RENTER_SIGNS_UP
username
renter id
RENTER
renter id
renter_name
renter_phone
renter_email
renter_address
CUSTOMER
customer id
customer_name
customer_phone
customer_email
customer_address
CUSTOMER_CONCERNS
case id
customer id
RENTER_CONCERNS
case id
renter id
CONCERNS
case id
console_id
status
concern
customer_id
renter_id
POSTS
console id
renter_id
console_name
brand
REQUESTS
request id
request_time
status_flag
customer_id
console_id
CONSOLE AVAILABILITY
console id
available_flag
CONSOLE DATA
console name
brand
available_quantity
rented_quantity