

**CS 353 – Spring 2018**

**Database Management Systems**

**Design Report**

Social Gaming Marketplace

**Group 24**

Ali Atlı 21302442 Section 3

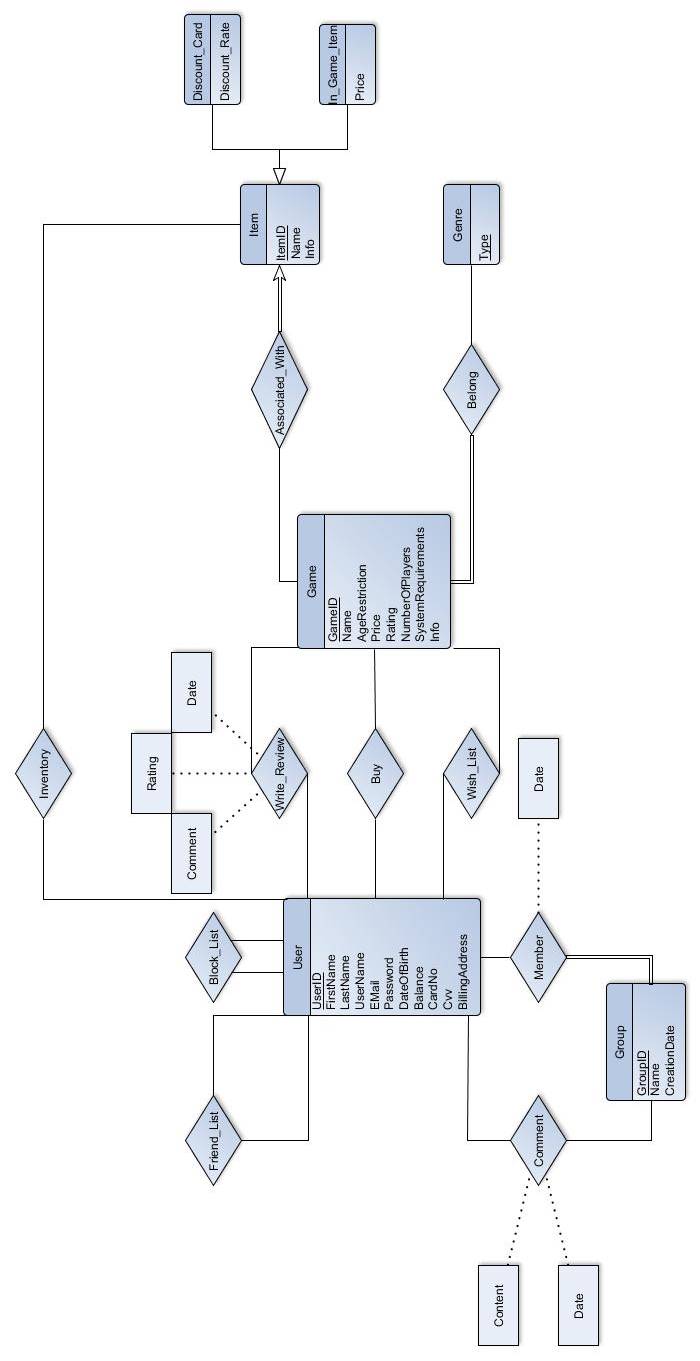
Mehmet Taha Çetin 21400281 Section 1

Ulaş İş 21401179 Section 1

Çağatay Küpeli 21402290 Section 1

**Table of Contents**

1. Revised ER Model2
   1. Updated ER Diagram2
   2. Changes Made to the Model3
2. Relational Schemas6
   1. User7
   2. Friend List9
   3. Block List10
   4. Game11
   5. Write Review13
   6. Buy15
   7. Wish List17
   8. Item19
   9. Inventory21
   10. Discount Item23
   11. In Game Item25
   12. Genre27
   13. Belong28
   14. Group29
   15. Member30
   16. Comment32
3. Functional Components34
   1. Use Cases/Scenarios34
   2. Algorithm37
      1. Average Rating Algorithm37
      2. Age Restriction Algorithm37
   3. Data Structures37
4. User Interface Design and Corresponding SQL Statements40
   1. Register/Login Screen40
   2. Store Screen42
   3. Store Screen for One Particular Genre44
   4. Profile Screen46
   5. Manage Information Screen50
   6. Community Screen52
   7. Search Screen53
   8. Group Screen55
   9. Game Screen57
   10. Inventory Screen60
5. Advance Database Components62
   1. Views62
      1. User-Age View62
      2. Group Comments View62
      3. Inventory View62
      4. Password View63
      5. Follower View63
      6. Bought Games View63
   2. Stored Procedures64
   3. Reports64
      1. Most Popular Games64
      2. Total Value of Inventory64
      3. Top Rated Games65
   4. Triggers65
   5. Constraints65
6. **Revised ER Model**



* 1. **Updated ER Diagram**

Figure 1: Updated ER Diagram

* 1. **Changes Made to the Model**

The following changes are establish in our entity-relation model to build a well-structured database system. These changes are made with respect to the feedback which is given by the teaching assistant and decisions we took as a team.

* We removed "Achievement" entity due the fact that it was hard to demonstrate in demo.
* We renamed "Player" entity to "User" in order to prevent naming conventions in our report.
* We added a missing self-referencing many-to-many relation which is called "Block\_List" to the "User" entity.
* We added missing attributes to "User". "FirstName", "LastName" and "Cvv".
* We renamed some attributes in "User" for naming convention. System will use "UserName" instead of "nickname" and "BillingAddress" instead of "Address".
* We removed some attributes in "User". "nation" and "points" are removed. "zip\_code" is added to "BillingAddress".
* We renamed "friends\_wish" relation to "Friend\_List" due to the naming convention.
* We removed "invite" relation due to the fact that it is nothing to the with database system.
* We reorganized/corrected "review" relation between "User" and "Game". The following entities and relations are removed or changed.
  + We removed "write" relation and its attributes.
  + We transform "Review" entity into a relation and named as "Write Review". Then, we added one more attribute which is called "Date". Therefore we renamed "score" to "Rating" due to the fact that the new name was better at demonstrating the purpose of the attribute.
  + We removed "has" relation.
* We removed "message" relation because we decided that we have no interest in storing conversation between two users.
* We removed "play" relation because it was planned to use for "Achievement" relation and when we removed it.
* We renamed "register" relation to "Buy" which was more suitable for the content it was planned to use for. We also removed its attribute which was "since" because it was planned to use for "Achievement" feature.
* Due to the fact that we removed Achievement, "earn" and "contains" relations are also scrapped.
* We reorganized/corrected inventory relation between "User" and "Item”. The following entities and relations are removed or changed.
  + We removed "posses" relation.
  + We transform "Inventory" entity into a relation. Then, we removed its attribute "item\_id" because it was wrong.
  + We removed "include" relation.
* We transformed "game\_genre" entity into a strong entity. For some reason, it was weak entity in the proposal which was wrong. We renamed "game\_genre" entity to "Belong" which is more suitable for naming convention.
* We removed some attributes in "Game". "target\_os", "capacity" are removed and instead we added "SystemRequirements" which is combination of both attributes.
* We added a new attribute called "Info" to store the irrelevant information about that particular game.
* We renamed "Guild" to "Group" which was better to demonstrate the idea we have in our minds. We also added one more attribute to it which is called "CreationDate".
* We removed "left\_date" attribute in "Member" relation due the fact that it was against the idea of database. We also renamed "since" to "Date" due to naming convention.
* We added a missing relation between "User" and "Group" which is called "Comment". It will store the date which was written in group's blog page. We also added an attribute to it which is called "Date" to order the comments which are entered in groups.
* We reorganized/corrected "item" entity and added its missing sub-entities.
  + We renamed "game\_item" relation to "Associated\_With" which is more suitable for naming convention.
  + We added missing sub-entities which are "Discount\_Card" and "In\_Game\_Item".
  + We added a new attribute called "Info" to "Item" to store the information about that particular item.

1. **Relational Schemas**

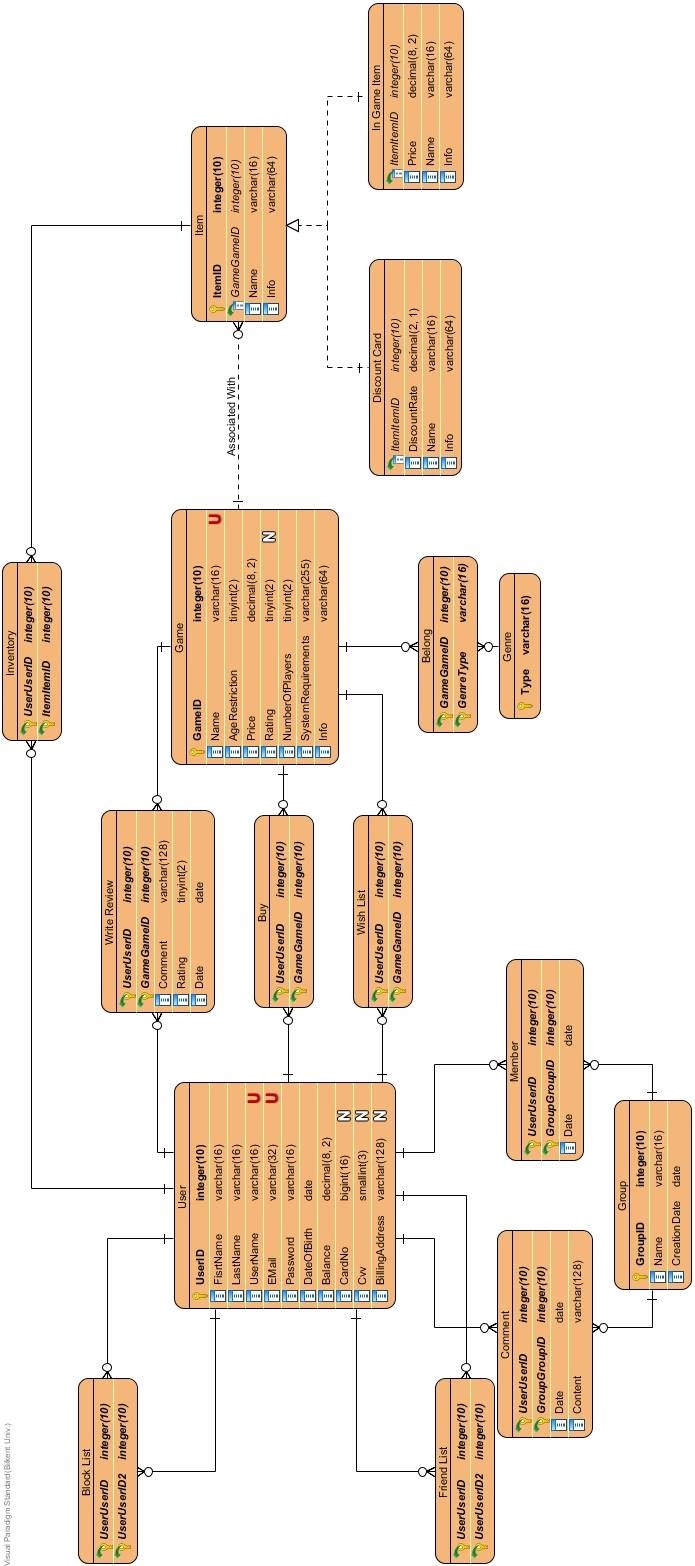


Figure 3: Physical Data Model

* 1. **User**

**Relational Model**

User (UserID, FirstName, LastName, UserName, Email, Password, DateOfBirth, Balance, CardNo, Cvv, BilingAddress)

**Functional Dependencies**

UserID - > FirstName, LastName, UserName, Email, Password, DateOfBirth, Balance, CardNo, Cvv, BilingAddress

UserName -> UserID, FirstName, LastName, Email, Password, DateOfBirth, Balance, CardNo, Cvv, BilingAddress

Email -> UserID, FirstName, LastName, UserName, Password, DateOfBirth, Balance, CardNo, Cvv, BilingAddress

**Keys**

Candidate Keys -> UserID, UserName, Email

Primary Key -> UserID

Foreign Key-> none

**Table Definition**

CREATE TABLE IF NOT EXISTS `mydb`.`User` (

`UserID` INT NOT NULL,

`FirstName` VARCHAR(16) NOT NULL,

`LastName` VARCHAR(16) NOT NULL,

`UserName` VARCHAR(16) NOT NULL UNIQUE,

`Email` VARCHAR(32) NOT NULL UNIQUE,

`DateOfBirth` DATE NULL,

`Balance` NUMERIC(8,2) NOT NULL,

‘CardNo’ BIGINT NOT NULL,

‘Cvv’ SMALLINT NOT NULL,

‘BillingAddress` VARCHAR(128) NOT NULL,

PRIMARY KEY (`UserID`))

ENGINE = InnoDB;

* 1. **Friend List**

**Relational Model**

FriendList (UserUserID, UserUserID2)

**Functional Dependencies**

None

**Keys**

Candidate Keys -> UserUserID, UserUserID2

Primary Key -> UserUserID, UserUserID2

Foreign Key-> UserUserID references User (UserID), UserUserID2 references User (UserID)

**Table Definition**

CREATE TABLE IF NOT EXISTS `mydb`.`FriendList ` (

`UserUserID` INT NOT NULL,

` UserUserID2` INT NOT NULL,

PRIMARY KEY (`UserUserID`, `UserUserID2`)

CONSTRAINT `UserUserID`

FOREIGN KEY (`UserUserID`)

REFERENCES `mydb`.`User` (`UserID`)

CONSTRAINT `UserUserID2`

FOREIGN KEY (`UserUserID2`)

REFERENCES `mydb`.`User` (`UserID`)

ENGINE = InnoDB;

* 1. **Block List**

**Relational Model**

BlockList (UserUserID, UserUserID2)

**Functional Dependencies**

None

**Keys**

Candidate Keys -> UserUserID, UserUserID2

Primary Key -> UserUserID, UserUserID2

Foreign Key-> UserUserID references User (UserID), UserUserID2 references User (UserID)

**Table Definition**

CREATE TABLE IF NOT EXISTS `mydb`.`BlockList ` (

`UserUserID` INT NOT NULL,

`UserUserID2` INT NOT NULL,

PRIMARY KEY (`UserUserID`, `UserUserID2`)

CONSTRAINT `UserUserID`

FOREIGN KEY (`UserUserID`)

REFERENCES `mydb`.`User` (`UserID`)

CONSTRAINT `UserUserID2`

FOREIGN KEY (`UserUserID2`)

REFERENCES `mydb`.`User` (`UserID`)

ENGINE = InnoDB;

* 1. **Game**

**Relational Model**

Game (GameID, Name, AgeRestriction, Price, Rating, NumberOfPlayers, SystemRequirements, Info)

**Functional Dependencies**

GameID -> Name, AgeRestriction, Price, Rating, NumberOfPlayers, SystemRequirements, Info

Name -> GameID, AgeRestriction, Price, Rating, NumberOfPlayers, SystemRequirements, Info

**Keys**

Candidate Keys -> GameID, Name

Primary Key -> GameID

Foreign Key-> none

**Table Definition**

CREATE TABLE IF NOT EXISTS `mydb`.`Game` (

`GameID` INT NOT NULL,

`Name` VARCHAR(16) NOT NULL,

`AgeRestriction` TINYINT NOT NULL,

`Price` NUMERIC(8,2) NOT NULL,

`Rating` TINYINT NULL,

`NumberOfPlayers` TINYINT NOT NULL,

`SystemRequirements` VARCHAR(256) NOT NULL,

‘Info` VARCHAR(64) NOT NULL,

PRIMARY KEY (`GameID`))

ENGINE = InnoDB;

* 1. **Write Review**

**Relational Model**

Write\_Review (UserUserID, GameGameID, Comment, Rating, Date)

**Functional Dependencies**

UserUserID, GameGameID -> Comment, Rating

**Keys**

Candidate Keys -> UserUserID, GameGameID

Primary Key -> UserUserID, GameGameID

Foreign Key-> UserUserID references User (UserID), GameGameID references Game (GameID)

**Table Definition**

CREATE TABLE IF NOT EXISTS `mydb`.`Write\_Review` (

`UserUserID` INT NOT NULL,

`GameGameID` INT NOT NULL,

`Comment` VARCHAR (128) NOT NULL,

‘Rating` TINYINT NOT NULL,

`Date` DATE NOT NULL,

PRIMARY KEY (`UserUserID`, `GameGameID`)

CONSTRAINT `UserUserID`

FOREIGN KEY (`UserUserID`)

REFERENCES `mydb`.`User` (`UserID`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `GameGameID`

FOREIGN KEY (GameGame`)

REFERENCES `mydb`.`Game` (`GameID`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB;

* 1. **Buy**

**Relational Model**

Buy (UserUserID, GameGameID)

**Functional Dependencies**

None

**Keys**

Candidate Keys -> UserUserID, GameGameID

Primary Key -> UserUserID, GameGameID

Foreign Key-> UserUserID references User (UserID), GameGameID references Game (GameID)

**Table Definition**

CREATE TABLE IF NOT EXISTS `mydb`.`Buy ` (

`UserUserID` INT NOT NULL,

`GameGameID` INT NOT NULL,

PRIMARY KEY (`UserUserID`, `GameGameID`)

CONSTRAINT `UserUserID`

FOREIGN KEY (`UserUserID`)

REFERENCES `mydb`.`User` (`UserID`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `GameGameID`

FOREIGN KEY (GameGame`)

REFERENCES `mydb`.`Game` (`GameID`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB;

* 1. **Wish List**

**Relational Model**

WhisList (UserUserID, GameGameID)

**Functional Dependencies**

None

**Keys**

Candidate Keys -> UserUserID, GameGameID

Primary Key -> UserUserID, GameGameID

Foreign Key-> UserUserID references User (UserID), GameGameID references Game (GameID)

**Table Definition**

CREATE TABLE IF NOT EXISTS `mydb`.`WishList ` (

`UserUserID` INT NOT NULL,

`GameGameID` INT NOT NULL,

PRIMARY KEY (`UserUserID`, `GameGameID`)

CONSTRAINT `UserUserID`

FOREIGN KEY (`UserUserID`)

REFERENCES `mydb`.`User` (`UserID`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `GameGameID`

FOREIGN KEY (GameGame`)

REFERENCES `mydb`.`Game` (`GameID`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB;

* 1. **Item**

**Relational Model**

Item (ItemID, Name, Info, GameGameID)

**Functional Dependencies**

ItemID -> Name, Info, GameGameID

**Keys**

Candidate Keys -> ItemID, Name

Primary Key -> ItemID

Foreign Key-> GameGameID (references Game(GameID))

**Table Definition**

CREATE TABLE IF NOT EXISTS `mydb`.`Item` (

`ItemID` INT NOT NULL,

`GameGameID` INT NOT NULL,

`Name` VARCHAR (16) NOT NULL,

‘Info` VARCHAR (64) NOT NULL,

PRIMARY KEY (`ItemID`))

CONSTRAINT ` GameGameID `

FOREIGN KEY (`GameGameID`)

REFERENCES `mydb`.`Game` (`GameID`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB;

* 1. **Inventory**

**Relational Model**

Inventory (ItemItemID, UserUserID)

**Functional Dependencies**

None

**Keys**

Candidate Keys -> UserUserID, ItemItemID

Primary Key -> UserUserID, ItemItemID

Foreign Key-> UserUserID references User (UserID), ItemItemID references Item (ItemID)

**Table Definition**

CREATE TABLE IF NOT EXISTS `mydb`.`Inventory ` (

`UserUserID` INT NOT NULL,

`ItemItemID` INT NOT NULL,

PRIMARY KEY (`UserUserID`, `ItemItemID`)

CONSTRAINT `UserUserID`

FOREIGN KEY (`UserUserID`)

REFERENCES `mydb`.`User` (`UserID`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT ` ItemItemID `

FOREIGN KEY (ItemItemID `)

REFERENCES `mydb`.`Game` (`GameID`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB;

* 1. **Discount Card**

**Relational Model**

Discount\_Card (ItemID, DiscountRate, Name, Info)

**Functional Dependencies**

ItemID -> DiscountRate, Name, Info

**Keys**

Candidate Keys -> ItemID

Primary Key -> ItemID

Foreign Key-> ItemID (references Item(ItemID))

**Table Definition**

CREATE TABLE IF NOT EXISTS `mydb`.`Discount\_Card` (

`ItemID` INT NOT NULL,

`DiscountRate` NUMERIC (2,1) NOT NULL,

`Name` VARCHAR (16) NOT NULL,

`Info` VARCHAR (64) NOT NULL,

PRIMARY KEY (`ItemID`))

CONSTRAINT `ItemID`

FOREIGN KEY (`ItemID`)

REFERENCES `mydb`.`Item` (`ItemID`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB;

* 1. **In Game Item**

**Relational Model**

In\_Game\_Item (ItemID, Price, Name, Info)

**Functional Dependencies**

ItemID -> Price, Name, Info

**Keys**

Candidate Keys -> ItemID

Primary Key -> ItemID

Foreign Key-> ItemID (references Item(ItemID))

**Table Definition**

CREATE TABLE IF NOT EXISTS `mydb`.`In\_Game\_Item` (

`ItemID` INT NOT NULL,

`Price` NUMERIC (8,2) NOT NULL,

`Name` VARCHAR (16) NOT NULL,

`Info` VARCHAR (64) NOT NULL,

PRIMARY KEY (`ItemID`))

CONSTRAINT `ItemID`

FOREIGN KEY (`ItemID`)

REFERENCES `mydb`.`Item` (`ItemID`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB;

* 1. **Genre**

**Relational Model**

Genre (Type)

**Functional Dependencies**

None

**Keys**

Candidate Keys -> Type,

Primary Key -> Type

Foreign Key-> none

**Table Definition**

CREATE TABLE IF NOT EXISTS `mydb`.`Genre` (

`Type` VARCHAR (16) NOT NULL,

PRIMARY KEY (`Type`))

ENGINE = InnoDB;

* 1. **Belong**

**Relational Model**

Belong (GameGameID, GenreType)

**Functional Dependencies**

None

**Keys**

Candidate Keys -> GameGameID, GenreType

Primary Key -> GameGameID, GenreType

Foreign Key-> GameGameID references Game (GameID), GenreType references Genre (Type)

**Table Definition**

CREATE TABLE IF NOT EXISTS `mydb`.`Belong` (

`GameGameID` INT NOT NULL,

`GenreType` VARCHAR (16) NOT NULL,

PRIMARY KEY (`GameGameID`, ` GenreType`)

CONSTRAINT ` GameGameID`

FOREIGN KEY (`GameGameID`)

REFERENCES `mydb`.`Game` (`GameID`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT ` GenreType`

FOREIGN KEY (`GenreType`)

REFERENCES `mydb`.`Genre` (`Type`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB;

* 1. **Group**

**Relational Model**

Group (GroupID, Name, CreationDate)

**Functional Dependencies**

GroupID -> Name, CreationDate

**Keys**

Candidate Keys -> GroupID

Primary Key -> GroupID

Foreign Key-> none

**Table Definition**

CREATE TABLE IF NOT EXISTS `mydb`.`Group` (

`GroupID` INT NOT NULL,

`Name` VARCHAR(16) NOT NULL,

`CreationDate` DATE NOT NULL,

PRIMARY KEY (`GroupID`))

ENGINE = InnoDB;

* 1. **Member**

**Relational Model**

Member (UserUserID, GroupGroupID, Date)

**Functional Dependencies**

UserUserID, GroupGroupID -> Date

**Keys**

Candidate Keys -> UserUserID, GroupGroupID

Primary Key -> UserUserID, GroupGroupID

Foreign Key-> UserUserID references User (UserID), GroupGroupID references Group (GroupID)

**Table Definition**

CREATE TABLE IF NOT EXISTS `mydb`.`Member` (

`UserUserID` INT NOT NULL,

`GroupGroupID` INT NOT NULL,

`Date` DATE NOT NULL,

PRIMARY KEY (`UserUserID`, `GroupGroupID`)

CONSTRAINT `UserUserID`

FOREIGN KEY (`UserUserID`)

REFERENCES `mydb`.`User` (`UserID`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `GroupGroupID`

FOREIGN KEY (`GroupGroupID`)

REFERENCES `mydb`.`Group` (`GroupID`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB;

* 1. **Comment**

**Relational Model**

Comment (GroupGroupID, UserUserID, Date)

**Functional Dependencies**

GroupGroupID, UserUserID -> Date

**Keys**

Candidate Keys -> UserIUserID, GroupGroupID

Primary Key -> UserUserID, GroupGroupID

Foreign Key-> UserUserID references User (UserID), GroupGroupID references Group (GroupID)

**Table Definition**

CREATE TABLE IF NOT EXISTS `mydb`.`Comment` (

`UserUserID` INT NOT NULL,

`GroupGroupID` INT NOT NULL,

`Date` DATE NOT NULL,

PRIMARY KEY (`UserUserID`, `GroupGroupID`)

CONSTRAINT `UserUserID`

FOREIGN KEY (`UserUserID`)

REFERENCES `mydb`.`User` (`UserID`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `GroupGroupID`

FOREIGN KEY (`GroupGroupID `)

REFERENCES `mydb`.`Group` (`GroupID`)

ON DELETE CASCADE

ON UPDATE CASCADE,)

ENGINE = InnoDB;

1. **Functional Components**
   1. **Use Cases/Scenarios**

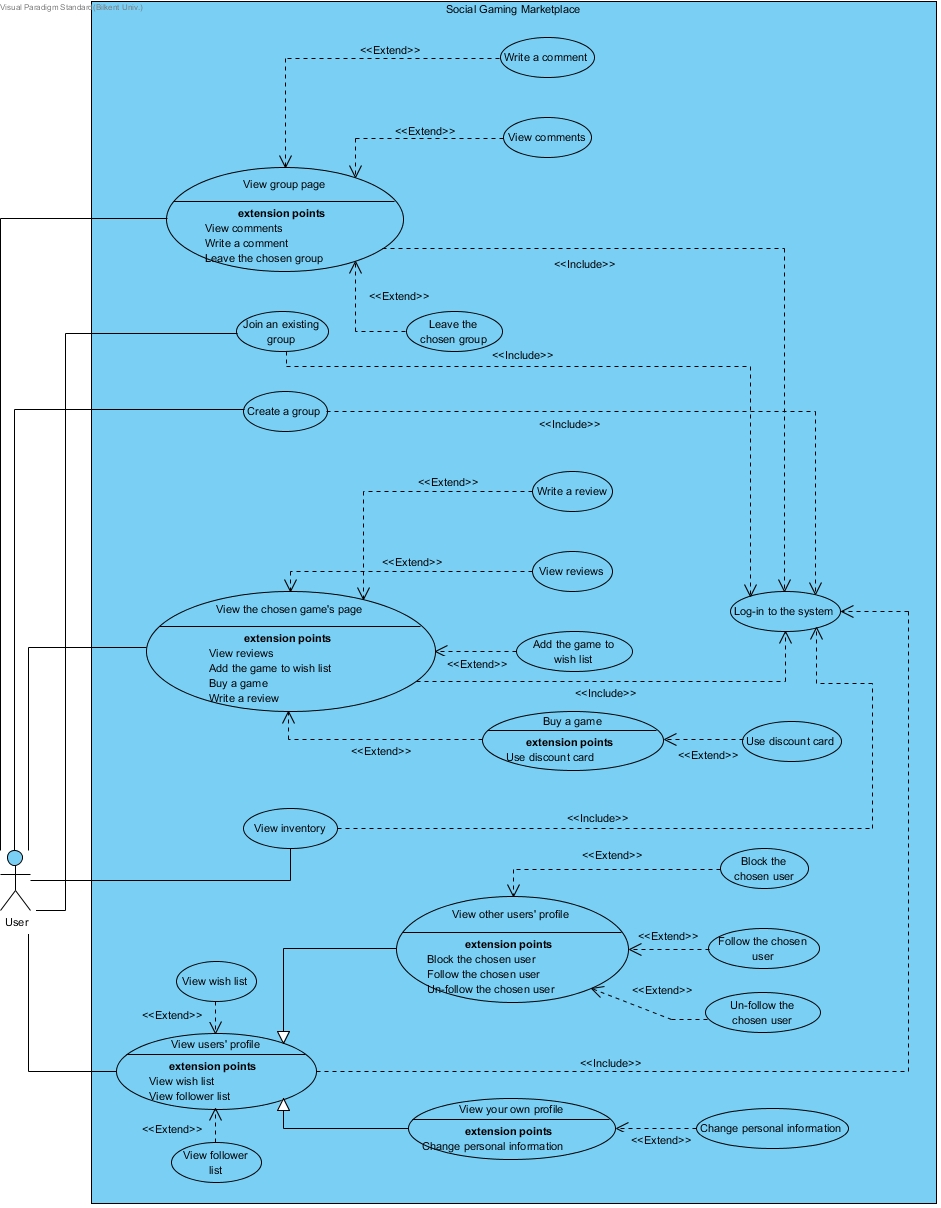


Figure 3: Use Cases/Scenarios

Social Gaming Marketplace is a gaming platform that allow users to buy and download digital games without leaving their home. It is also a social networking service where users create their own communities and discuss about any topic they want with their friends. Therefore there is only one actor, which will be called user, in Social Gaming Marketplace.

The following text will describe what user can do in Social Gaming Marketplace.

* **Log-in to the system:** Each user must register and login by using his/her own username/e-mail address and password in order to use other features in the system. There are no operation can be performed without log-in to the system.
* **Create a group:** Each user can create his/her own groups to build a community for discussing games and other stuff.
* **View group page:** Each user can search existing groups from the list each group sorted to view any group's home page.
* **Join an existing group:** Each user are allowed to join any group by using a button in that particular group's home page.
* **View comments:** Each group has a section that specifically reserved for discussion. In order to see the content user must join that particular group.
* **Write comment:** Each user that is member to a group can write comments to that group's discussion segment.
* **Leave the chosen group:** Each user are allowed to leave any group they are member of. The join button will replace itself with leave button when a user become member of that particular group.
* **View wish list:** Each user can see any user's wish list in his/her profile.
* **View follow list:** Each user can see any user's follow list in his/her profile.
* **View other user's profile:** Each user can see other user's profile unless they are block by that particular user. This operation can be done by using search feature in Social Gaming Marketplace.
* **Follow the chosen user:** Each user can follow other users in order to communicate with them and see details in his/her profile by using a button in his/her profile page. There is no taking permission feature in Social Gaming Marketplace. When one person presses follow button, both users will follow each other automatically.
* **Unfollow the chosen user:** Each user can unfollow other users in order to prevent communicate and prevent them to see details in his/her profile by using a button in his/her profile page which will replace follow button. When one person presses unfollow button, both users will unfollow each other automatically.
* **Block the chosen user:** Each user can block other users by using a button in his/her profile which prevent all the communication between those two users. Purpose of this feature is to prevent unwanted followers.
* **View your own profile:** Each user have a unique profile, the interface is similar to other users, but with more uniqueness. The data shown is unique and can managed by the owner.
* **Change personal information:** Each user can manage his/her information in setting, by using a button in his/her own profile.
* **View inventory:** Each user has access to his/her own inventory to view the items in it.
* **View the chosen game's page:** Each user have access to any game's page if his/her age is suitable for that particular game.
* **Write review:** Each user can write review for a game they own and rate it.
* **Add the game to wish list:** Each user can add any game to their wish list if his/her age is suitable for accessing that game's page.
* **Buy game:** Each user can buy any game if his/her age is suitable for accessing that game's page.
* **Use discount card:** Each user can use discount card to get a discount for the game the card is associated with.
  1. **Algorithms**
     1. **Average Rating Algorithm**

In Social Gaming Marketplace, each user can write down a review about a game they own. Each review requires a comment and a rating which is scaled from 1 to 5. When other users view the game's home page, they should be able to see all the reviews with their rating, date of entering and an average rating.

"Write\_Review" relation is dedicated to this purpose. It will store a comment, a rating and an entering date. Thus the average will be calculated just by taking average of this table.

* + 1. **Age Restriction Algorithm**

Some games in Social Gaming Marketplace is not suitable for all audience. In these case, the system will prevent those user to enter that particular game's page. The age will be calculated by subtracting date of birth from today's date.

"User" entity has an attribute called "DateOfBirth" which is required to register the system. This attribute will be used for restricting the content.

* 1. **Data Structures**

The relational schemas we have designed utilizes Numeric Types, String Types and Date.

* Numeric types are used for keeping numeric data types such as identity number, currencies and ratings. We have used type of numeric types which are INTEGER, TINYINT, DECIMAL, BIGINT, and SMALLINT. The reason behind variety of numeric types is to use memory space efficiently. For example, we used TINYINT for Rating because it only takes values between 1 and 5.
* String types are needed to store any attributes that is combination of characters such as names, addresses, and information. We only used VARCHAR. The reasoning behind this design choice is that our system not efficient enough to utilize CHARs perfectly.
* Date types are used for storing time values such as the date of the creation of group, date of birth and date of the entry/comment. We are using Date only for this purpose.

1. **User Interface Design and Corresponding SQL Statements**
   1. **Register/Login Screen**

Figure 4: Register/Login Mockup



**Inputs:** @email\_or\_username, @password

**Process:** User enters his/her e-mail address or username with matching password to login to the system.

**SQL Statements:**

SELECT UserID

FROM User

WHERE (UserName = @email\_or\_username OR EMail = @email\_or\_username) AND Password = @password;

**Inputs:** @first\_name, @last\_name, @username, @email, @password, @password\_repeat, @date\_of\_birth

**Process:** User enters his/her personal information to register the system.

**SQL Statements:**

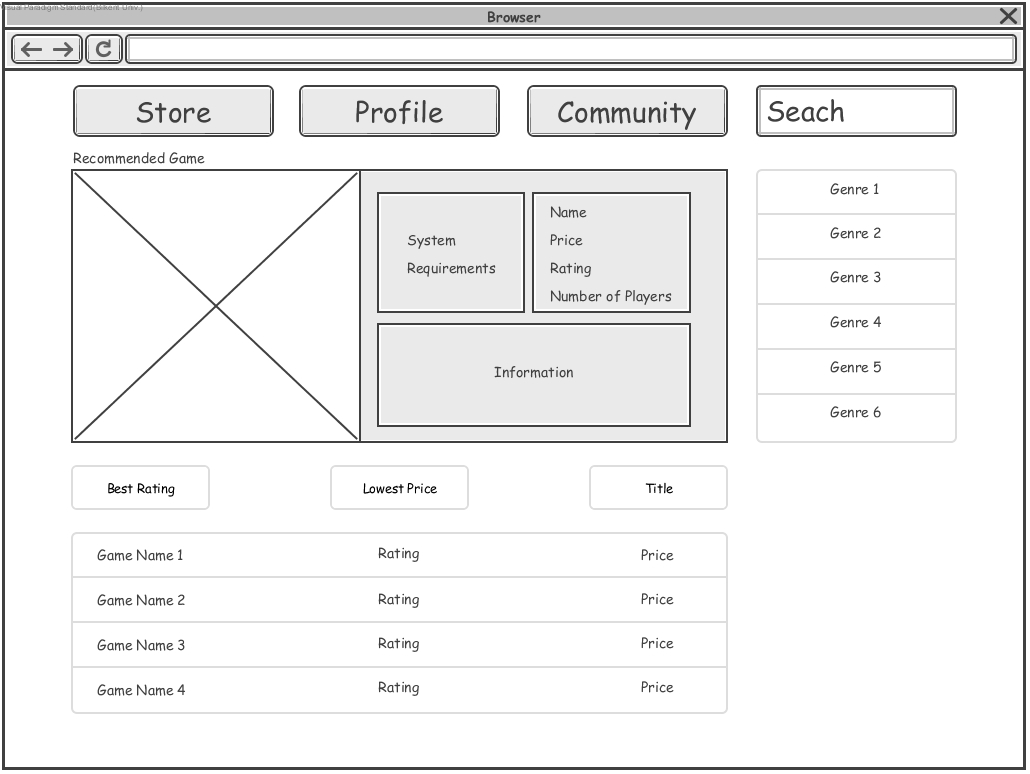
INSERT INTO User (FirstName, LastName, UserName, EMail, Password, Balance)

VALUES (@first\_name, @last\_name, @username, @email, @password, 0)

WHERE @password = @password\_repeat;

* 1. **Store Screen**

Figure 5: Store Mockup



**Inputs:** @query\_content

**Views:** age\_restriction is a view which can be find under 5.1.1.

Aim of age\_restriction is preventing user to access games that are appropriate for them.

**Process:** The store page is built to display games according to the segments which user can choose.

**SQL Statements:**

SELECT GameID, Name, Price, Rating

CASE

WHEN @query\_content = best\_rating

END

FROM age\_restriction

ORDER BY Rating DESC;

SELECT GameID, Name, Price, Rating

CASE

WHEN @query\_content = lowest\_price

END

FROM age\_restriction

ORDER BY price;

SELECT GameID, Name, Price, Rating

CASE

WHEN @query\_content = title

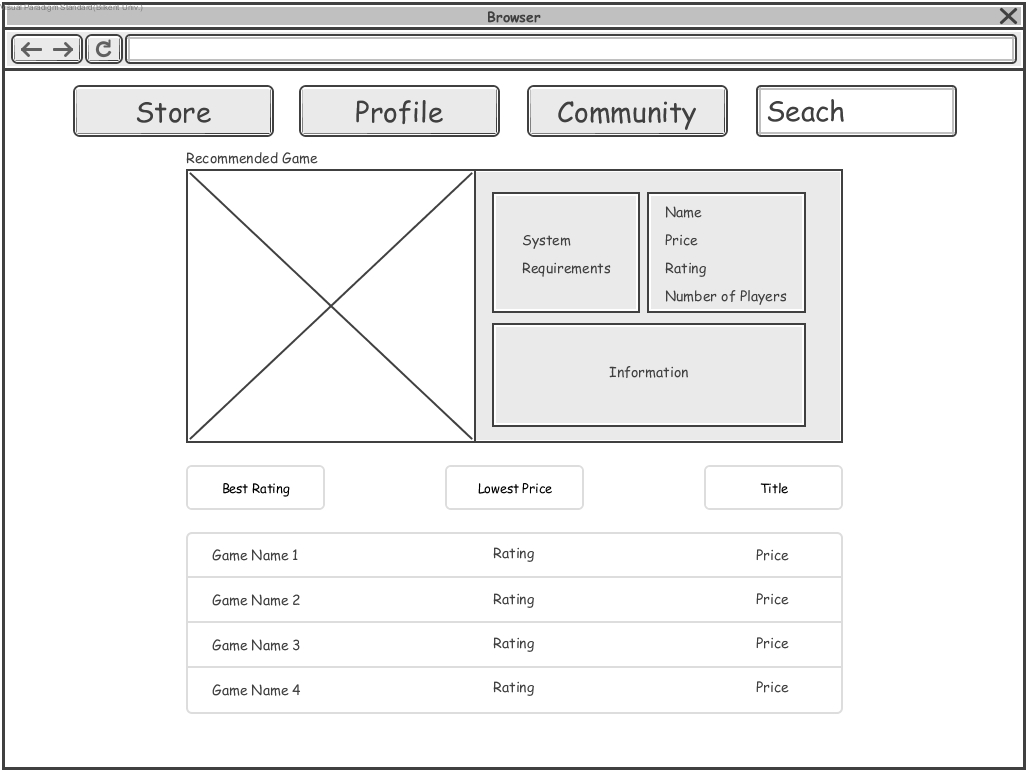
END

FROM age\_restriction

ORDER BY Name;

* 1. **Store Screen for One Particular Genre**

Figure 6: Store for One Genre Mockup



**Inputs:** @query\_content, @genre

**Views:** age\_restriction is a view which can be find under 5.1.1.

Aim of age\_restriction is preventing user to access games that are appropriate for them.

**Process:** The store page for one particular genre is built to display games according to the segments which user can choose for that particular genre.

**SQL Statements:**

SELECT AR.GameID, AR.Name, AR.Price, AR.Rating

CASE

WHEN @query\_content = best\_rating

END

FROM age\_restriction AR, Belong B

WHERE AR.GameID = B.GameGameID AND B.GenreType = @genre

ORDER BY Rating DESC;

SELECT AR.GameID, AR.Name, AR.Price, AR.Rating

CASE

WHEN @query\_content = lowest\_price

END

FROM age\_restriction AR, Belong B

WHERE AR.GameID = B.GameGameID AND B.GenreType = @genre

ORDER BY AR.Rating DESC;

SELECT AR.GameID, AR.Name, AR.Price, AR.Rating

CASE

WHEN @query\_content = title

END

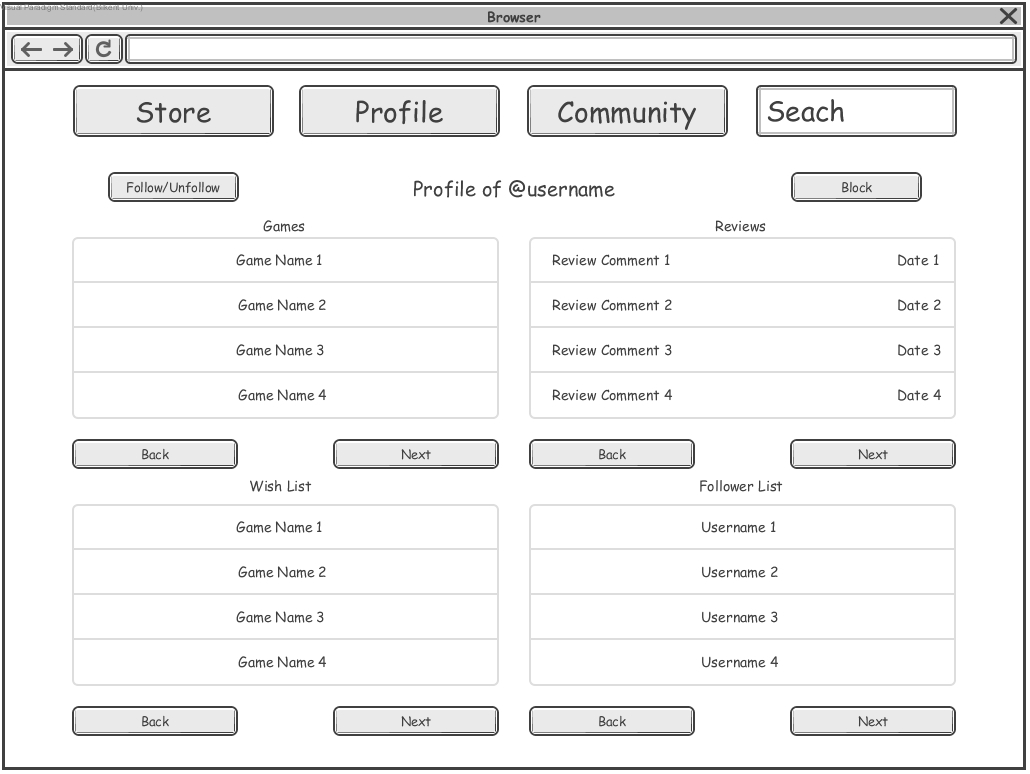
FROM age\_restriction AR, Belong B

WHERE AR.GameID = B.GameGameID AND B.GenreType = @genre

ORDER BY AR.Name;

* 1. **Profile Screen**

Figure 7: Profile Page from Other Users Perspective Mockup



**Inputs:** @user\_id, @follower\_id

**Process:** User can views other user's profile. They should be able to see their games, reviews, wish list and followers unless they are blocked.

**SQL Statements:**

SELECT G.GameID, G.Name

FROM Game G, Buy B

WHERE B.UserUserID = @user\_id AND B.GameGameID = G.GameID AND @follower\_id NOT IN ((SELECT DISTINCT UserUserID2 FROM Block\_List BL WHERE BL.UserUserID = @user\_id) UNION (SELECT DISTINCT UserUserID FROM Block\_List BL WHERE BL.UserUserID2 = @user\_id);

SELECT G.GameID, G.Name, WR.Comment, WR.Rating

FROM Game G, Write\_Review WR

WHERE WR.UserUserID = @user\_id AND WR.GameGameID = G.GameID AND @follower\_id NOT IN ((SELECT DISTINCT UserUserID2 FROM Block\_List BL WHERE BL.UserUserID = @user\_id) UNION (SELECT DISTINCT UserUserID FROM Block\_List BL WHERE BL.UserUserID2 = @user\_id);

SELECT G.GameID, G.Name

FROM Game G, Wish\_List WL

WHERE WL.UserUserID = @user\_id AND WL.GameGameID = G.GameID AND @follower\_id NOT IN ((SELECT DISTINCT UserUserID2 FROM Block\_List BL WHERE BL.UserUserID = @user\_id) UNION (SELECT DISTINCT UserUserID FROM Block\_List BL WHERE BL.UserUserID2 = @user\_id);

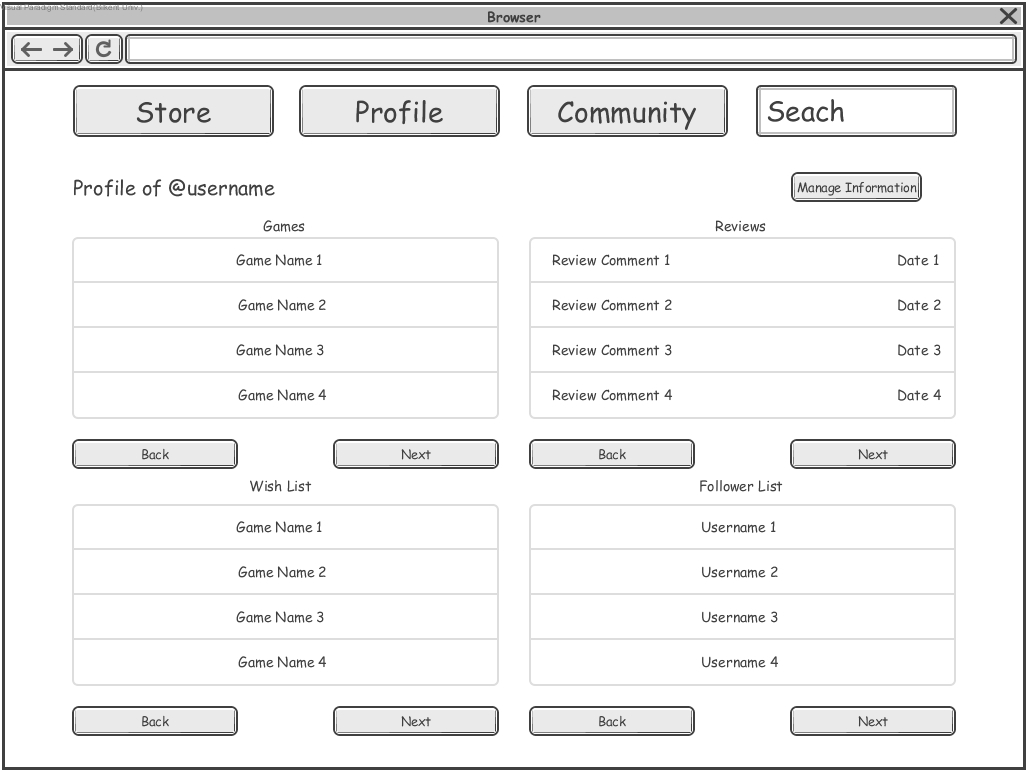
SELECT U.UserID, U.UserName

FROM User U, Friend\_List FL

WHERE FL.UserUserID = @user\_id AND (U.UserUserID = FL.UserUserID2 OR U.UserUserID = FL.UserUserID) AND @follower\_id NOT IN ((SELECT DISTINCT UserUserID2 FROM Block\_List BL WHERE BL.UserUserID = @user\_id) UNION (SELECT DISTINCT UserUserID FROM Block\_List BL WHERE BL.UserUserID2 = @user\_id);

**Input:** @user\_id

Figure 8: Profile Page from His/her Own Perspective Mockup



**Process:** User can views his/her own profile. He/she should be able to see his/her games, reviews, wish list and followers.

**SQL Statements:**

SELECT G.GameID, G.Name

FROM Game G, Buy B

WHERE B.UserUserID = @user\_id AND B.GameGameID = G. GameID

SELECT G.GameID, G.Name, WR.Comment, WR. Rating

FROM Game G, WriteReview WR

WHERE WR.UserUserID =@ user\_id AND WR.GameGameID = G.GameID

SELECT G.GameID, G.Game

FROM Game G, WishList WL

WHERE WL.UserUserID = @user\_id AND WL.GameGameID = G.GameID

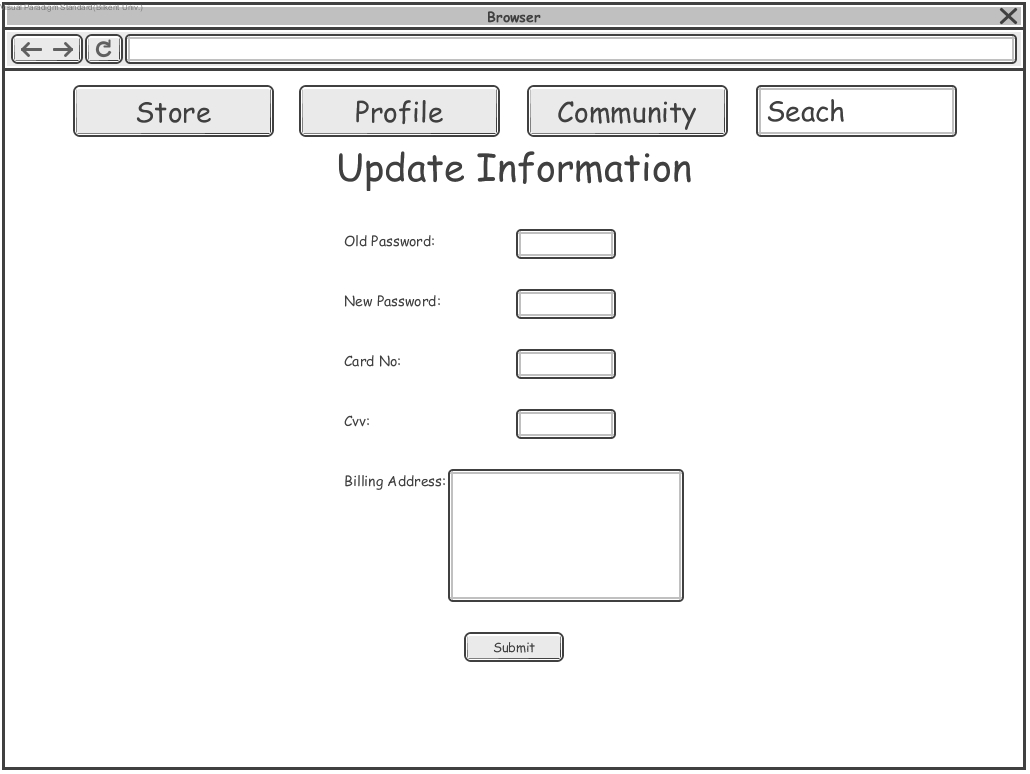
SELECT U.UserID, U.UserName

FROM FriendList FL, User U

WHERE U.UserID = FL.UserUserID AND FL.UserUserID = @user\_id

* 1. **Manage Information Screen**

Figure 7: Manage Information Mockup



**Inputs:** @user\_id, @old\_password, @new\_password, @card\_no, @cvv, @billing\_address

**Process:** Manage information page allow user to update his/her informations such as password, card number, cvv, and billing address.

**SQL Statements:**

UPDATE User

SET Password = @new\_password

WHERE UserID = @user\_id AND Password = @old\_password AND @old\_password <> @new\_password;

UPDATE User

SET CardNo = @card\_no

WHERE UserID = @user\_id;

UPDATE User

SET Cvv = @cvv

WHERE UserID = @user\_id**;**

UPDATE User

SET BillingAddress = @billing\_address

WHERE UserID = @user\_id**;**

* 1. **Community Screen**



Figure 8: Community Mockup

**Input:** @user\_id

**Process:** List all the groups that user is member of.

SELECT G.GroupID, G.Name, G.creationDate

FROM Group G

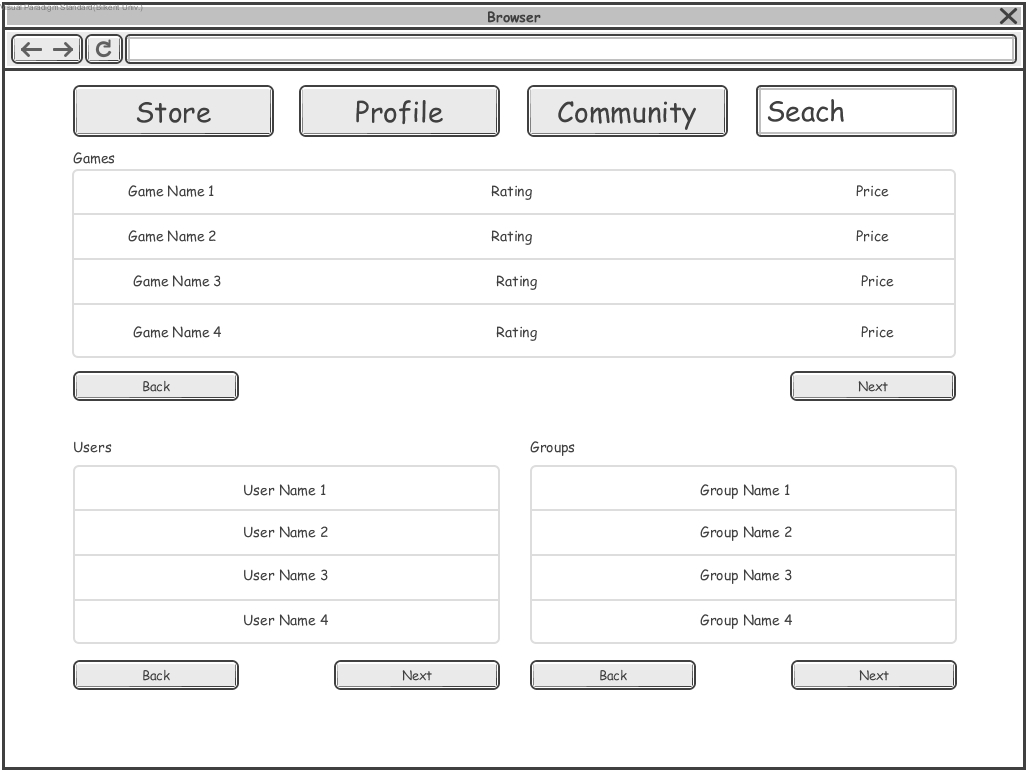
SELECT G.GroupID, G.Name, G.creationDate

FROM Group G, Member M

WHERE G.GroupID = M. GroupGroupID AND M.UserUserID = @user\_id

* 1. **Search Screen**

Figure 9: Search Mockup



**Input:** @keyword, @user\_id

**Process:** Search all games, users that are not in user block list and also the groups contains given keyword

**SQL Statements:**

SELECT G.GameID, G.Name, G.Rating, G.Price

FROM Game G

WHERE G.Name LIKE '%' + @keyword + '%';

SELECT U.UserID, U.UserName

FROM User U

WHERE U.UserName LIKE '%' + @keyword + '%' AND @user-id NOT IN (SELECT UserUserID2 FROM BlockList WHERE U.UserID = UserUserID);

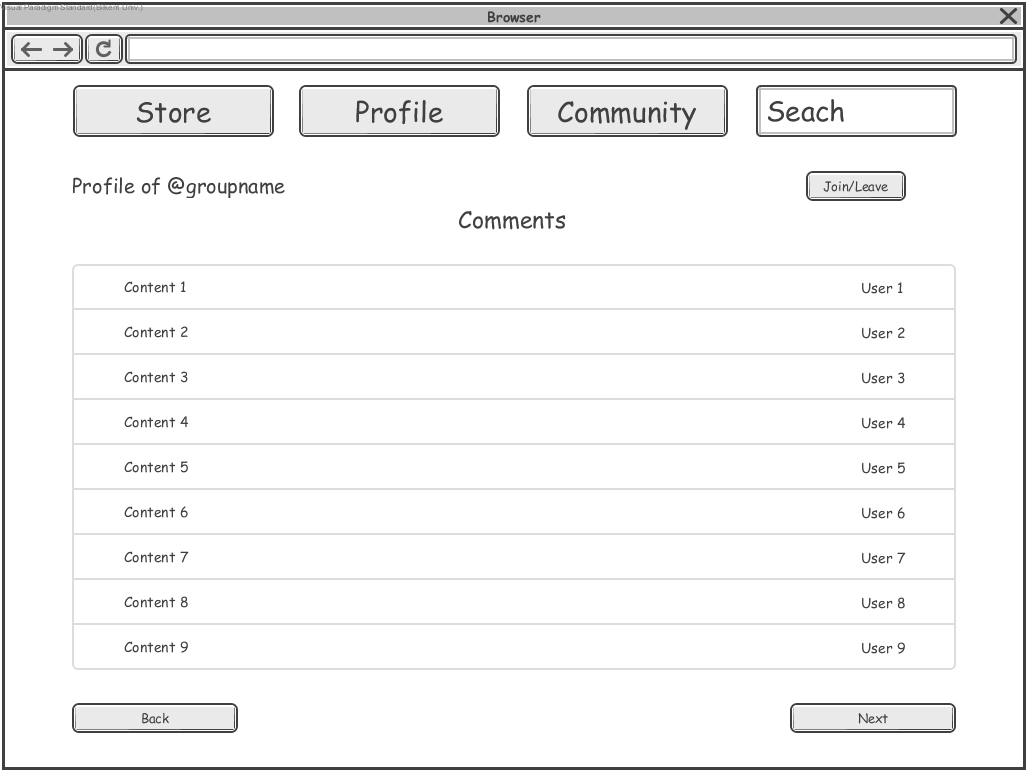
SELECT G.GroupID, G.Name

FROM Group G

WHERE G.Name LIKE '%' + @keyword + '%';

* 1. **Group Screen**

Figure 10: Group Mockup



**Inputs:** @user\_id, @group\_id

**Process:** User can see the comments of groups they are member of.

**SQL Statements:**

SELECT U.UserName, C.Content

FROM User U, Comment C, Group G

WHERE G.GroupID = @group\_id AND G.GroupID = C.GroupGroupID AND U.UserID = C.UserUserID AND @user\_id IN (SELECT M.UserUserID FROM Member M WHERE M.GroupGroupID = @group\_id)

**Inputs:** @user\_id, @group\_id

**Process:** User can join or leave groups.

**SQL Statements:**

INSERT INTO Member

VALUES (@used\_id, @group\_id, CONVERT(DATE, GETDATE ()))

WHERE @user\_id NOT IN (SELECT UserUserID FROM Member M WHERE M.GroupGroupID = @group\_id);

DELETE FROM Member

WHERE UserUserID = @user\_id AND GroupGroupID = @group\_id AND @user\_id IN (SELECT UserUserID FROM Member M WHERE M.GroupGroupID = @group\_id);

* 1. **Game Screen**

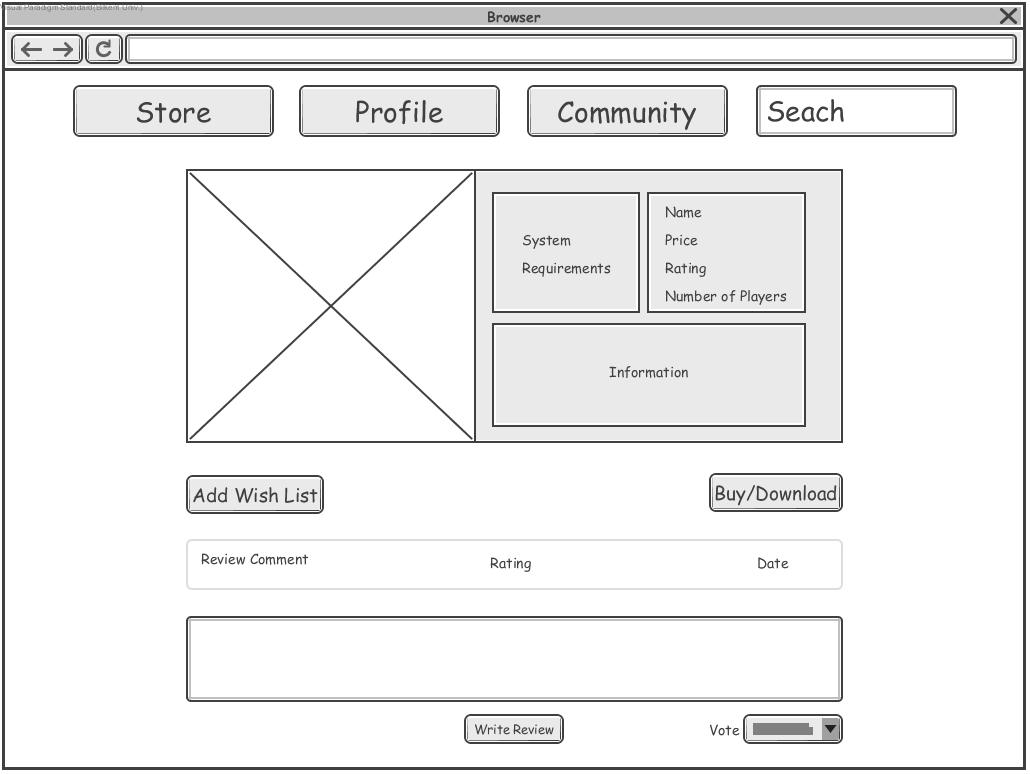


Figure 11: Game Screen For User Who Owns it Mockup

**Inputs:** @user\_id, @game\_id, @comment, @rating

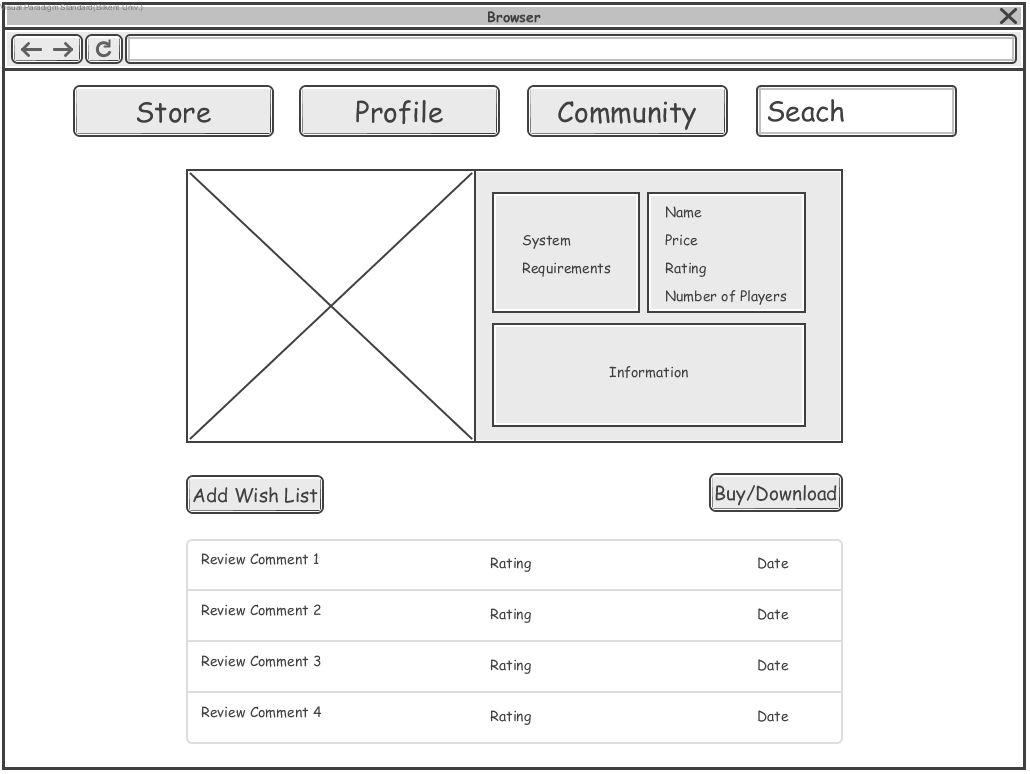
**Process:** User should be able to write review the game they own.

**SQL Statements:**

INSERT INTO Write\_Review

VALUES (@user\_id, @game\_id, @comment, @rating, CONVERT (DATE, GETDATE ())

WHERE @user\_id IN (SELECT UserUserID FROM Buy WHERE GameGameID = @game\_id)



**Inputs:** @user\_id, @game\_id

Figure 12: Game Screen Mockup

**Process:** Users display game screen.

**SQL Statements:**

SELECT G.GameID, G.Name, G.Price, G.Rating, G.Info

FROM Game G, Buy B

WHERE G.GameID = @game\_id AND B.UserUserID = @user\_id AND G.GameID = B.GameGameID;

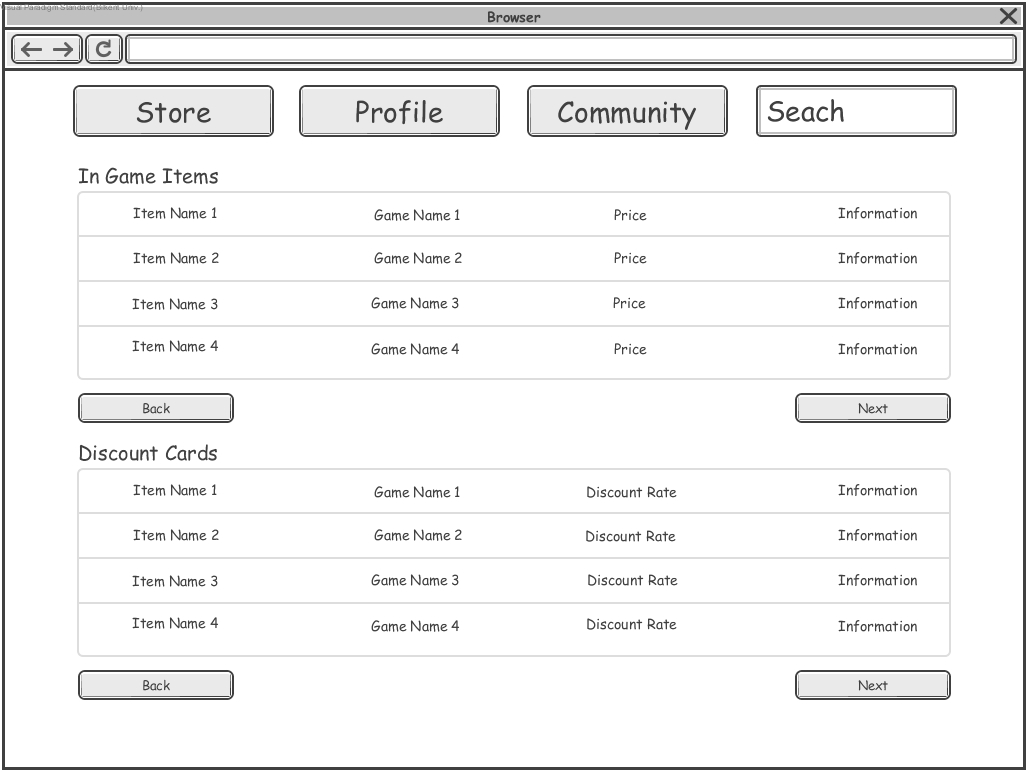
SELECT WR.Comment, WR.Rating, WR.Rate

FROM Game G, Write\_Review WR

WHERE G.GameID = @game\_id AND WR.UserUserID = @user\_id AND G.GameID = WR.GameGameID;

* 1. **Inventory Screen**

Figure 12: Inventory Mockup



**Input:** @user\_id

**Process:** List in game items and discount card separately in the user's inventory.

SELECT IGI.ItemItemID , IGI.Price

FROM In\_Game\_Item IGI, Invetory I

WHERE I.ItemItemID = IGI.ItemItemID AND I.UserUserID = @user\_id

SELECT DC.ItemItemID, DC.Discount\_Rate, DC.ItemID, DC.Name, DC.Info

FROM Discount\_Card DC, Inventory I

WHERE I.ItemItemID = DC.ItemItemID AND I.UserUserID = @user\_id

1. **Advance Database Components**
   1. **Views**
      1. **User-Age View**

Games should be appropriate for the age of each user.

CREATE VIEW age\_restriction AS

SELECT G.\*

FROM User U, Game G

WHERE G.AgeRestriction <= (SELECT (CONVERT (INTEGER, CONVERT (CHAR (4), CONVERT (DATE, GETDATE ()), 112))) – CONVERT (INTEGER, CONVERT (CHAR (4), U.DateOfBirth))))

* + 1. **Group Comments View**

Comments of each group should be visible to only members of that group. @user\_id is the UserID of the user that is viewing the group.

CREATE VIEW group\_comments AS

SELECT \*

FROM Comment C

WHERE C.GroupID IN (SELECT G.GroupID FROM Group G, Member M WHERE G.GroupID = M.GroupID AND M.UserID = @user\_id)

* + 1. **Inventory View**

Inventory of a user should be only and only visible to that user. @user\_id is the UserID of the user the inventory belongs to.

CREATE VIEW user\_inventory AS

SELECT IT.ItemID, IT.Name, IT.Info

FROM Inventory I, Item IT

WHERE I.UserID = @user\_id

* + 1. **Password View**

Password should be visible to only the user himself/herself. @user\_id is the UserID of the user the inventory belongs to.

CREATE VIEW user\_password AS

SELECT UserID, Password

FROM User

WHERE UserID = @user\_id

* + 1. **Follower View**

Follower list should be only visible to the followers of that particular user. @user\_id is the UserID of the user that is viewing the profile of the user and @follower\_id is the UserID of the user that own the profile.

CREATE VIEW user\_follower AS

((SELECT DISTINCT F.UserUserID2

FROM Friend\_List F

WHERE EXIST (SELECT \* FROM Friend\_List F, User U WHERE (F.UserUserID = @user\_id AND F.UserUserID2 = @follower\_id) OR (F.UserUserID2 = @follower\_id AND F.UserUserID = @user\_id)) AND F.UserUserID = @follower\_id)

UNION

(SELECT DISTINCT F.UserUserID1

FROM Friend\_List F

WHERE EXIST (SELECT \* FROM Friend\_List F, User U WHERE (F.UserUserID = @user\_id AND F.UserUserID2 = @follower\_id) OR (F.UserUserID2 = @follower\_id AND F.UserUserID = @user\_id)) AND F.UserUserID2 = @follower\_id))

* + 1. **Bought Games View**

Display all games that is bought by a particular user. This view is needed because each user can play/download or make a review to the games they own. @user\_id is the UserID of the user that is currently logged in.

CREATE VIEW game\_bought AS

SELECT G.GameID, G.Name

FROM Game G, Buy B

WHERE G.GameID = B.GameGameID AND B.UserID = @User\_id

* 1. **Stored Procedures**

We plan to use stored procedures when user buys a game. When a user buys a game, the balance of the player will be updated. Moreover, a discount card will be awarded to the user and added to his/her inventory. These can be done by using stored procedures.

Another stored procedure will be used for logging in to the system. Since login process is the same for all users, we can create a stored procedure for logging in to the system which will check the validity of user-name and password, and this procedure can be executed whenever the user tries to login.

* 1. **Reports**
     1. **Most Popular Games**

In the front/store page of Social Gaming Marketplace, games will be displayed with respect to different parameters and one of them is most popular games.

SELECT GameID, count(\*) as count\_game

FROM Buy

GROUP BY GameID

ORDER BY count\_game DESC

* + 1. **Total Value of Inventory**

Total value of the items in user's inventory can be found by using the following SQL query. We might put this information to user's profile to show off.

SELECT UserID, IFNULL(sum(IGI.price), 0) as total\_price

FROM Inventory I, In\_Game\_Item IGI

WHERE I.ItemID = IGI.ItemID

GROUP BY UserID

* + 1. **Top Rated Games**

In the front/store page of Social Gaming Marketplace, games will be displayed with respect to different parameters and one of them is top rated games.

SELECT G.GameID, G.Name, G.Rating

FROM Game G, Write\_Review WR

WHERE G.GameID = WR.GameGameID

GROUP BY G.GameID

ORDER BY G.Rating DESC

* 1. **Triggers**
* When a user writes a review and rates the game, the average rating of the game will be re-calculated and will get updated. For instance if user gave the game a rating higher than its current rating, the new rating of the game is expected to be higher and vice-versa.
* When a user buys a game with his/her balance, his/her account balance gets updated and the price of the game is subtracted from the balance.
* When all members of a group leaves the group, the group should be deleted from the database.
* When a user buys a game in his/her wish list that game is removed from his/her wish list.
  1. **Constraints**
* We scrapped the idea of e-mail validation system because it is hard to implement, which will take time from developing database and it is not related to the database which this project is about.
* Users can only buy/view the games that is appropriate for their age.
* Social Gaming Marketplace allows two ways of transaction. System can either take the money from balance of the account or via credit card. Account owner can choose either he/she will use discount card at this point.
* Users can’t buy games whose price is higher than their current balance if they are buying it from their balance, not from their credit card.
* Rating of a game cannot be smaller than 1 and greater than 5.