Project 1 Stage 2

Database Design

Content

1 UML Diagram

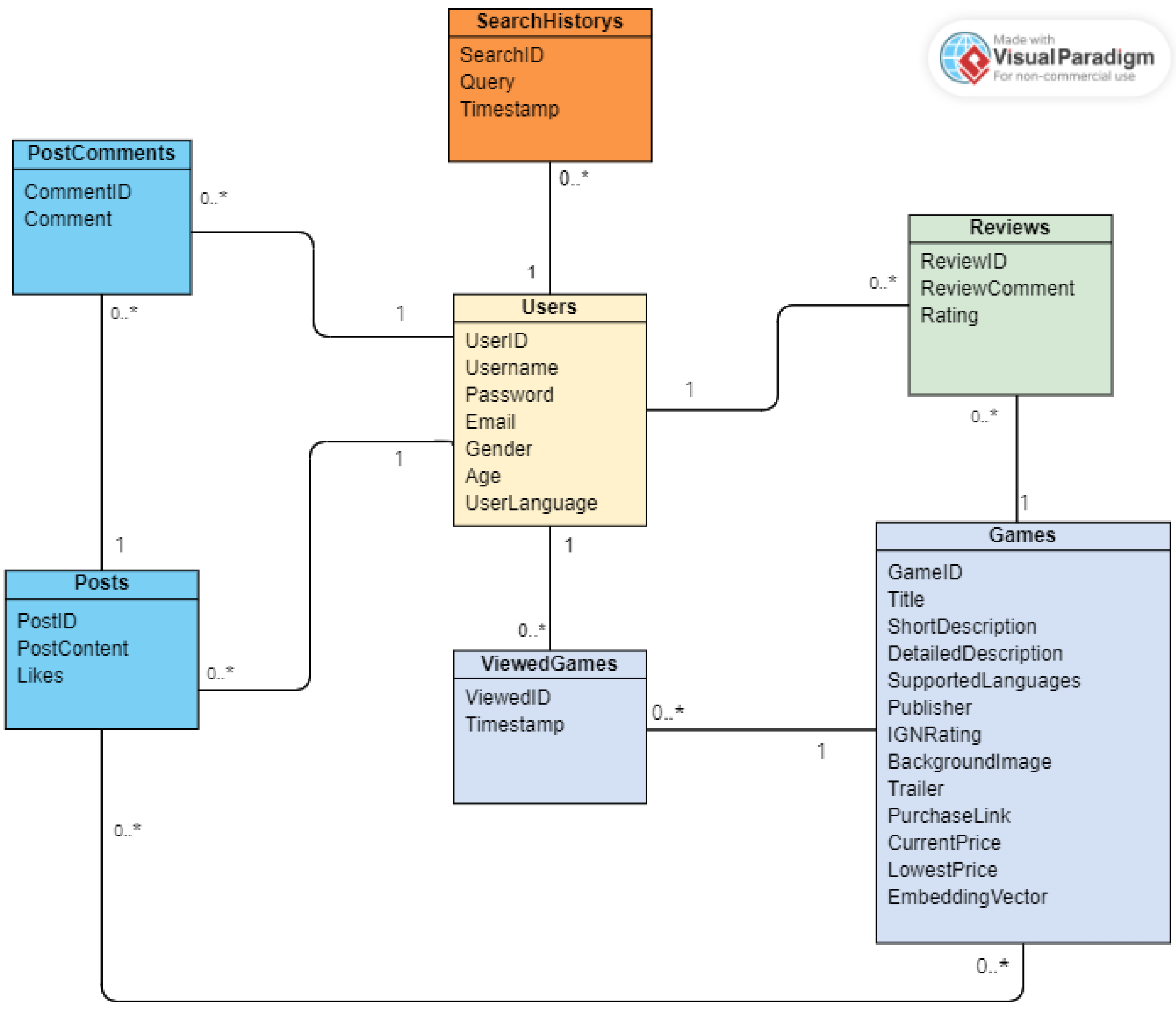
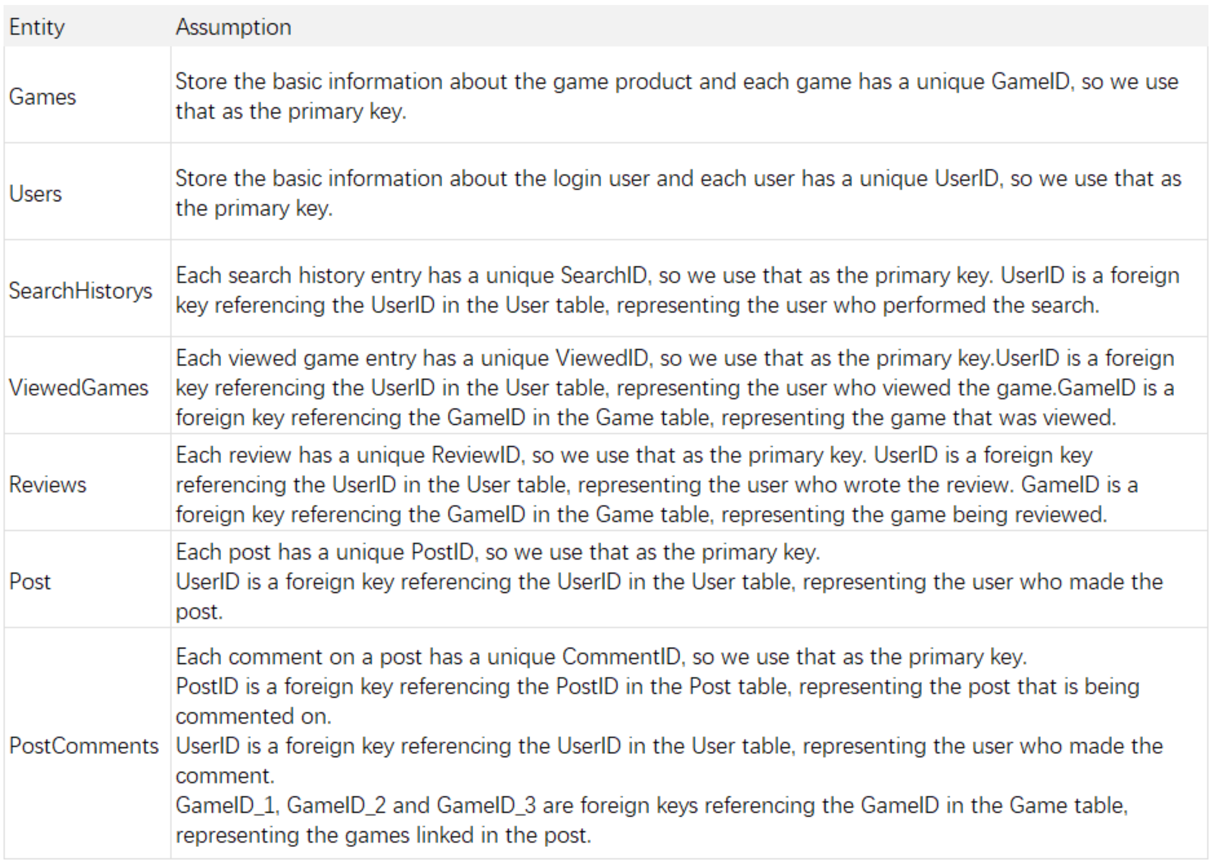


figure 1 UML Diagram

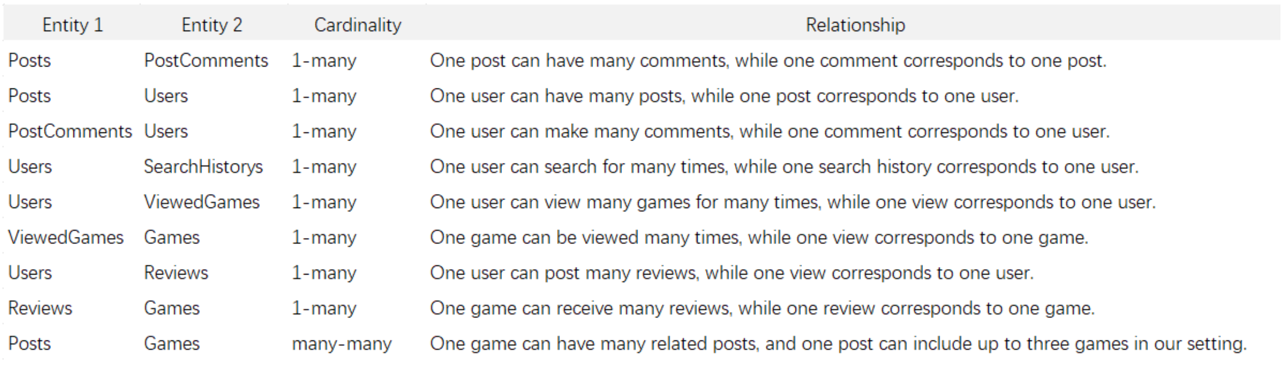
1.1 Assumption for UML Diagram



Entity	Assumption
Games	Store the basic information about the game product and each game has a unique GameID, so we use that as the primary key.
Users	Store the basic information about the login user and each user has a unique UserID, so we use that as the primary key.
SearchHistorys	Each search history entry has a unique SearchID, so we use that as the primary key. UserID is a foreign key referencing the UserID in the User table, representing the user who performed the search.
ViewedGames	Each viewed game entry has a unique ViewedID, so we use that as the primary key.UserID is a foreign key referencing the UserID in the User table, representing the user who viewed the game.GameID is a foreign key referencing the GameID in the Game table, representing the game that was viewed.
Reviews	Each review has a unique ReviewID, so we use that as the primary key. UserID is a foreign key referencing the UserID in the User table, representing the user who wrote the review. GameID is a foreign key referencing the GameID in the Game table, representing the game being reviewed.
Post	Each post has a unique PostID, so we use that as the primary key. UserID is a foreign key referencing the UserID in the User table, representing the user who made the post.
PostComments	Each comment on a post has a unique CommentID, so we use that as the primary key. PostID is a foreign key referencing the PostID in the Post table, representing the post that is being commented on. UserID is a foreign key referencing the UserID in the User table, representing the user who made the comment. GameID_1, GameID_2 and GameID_3 are foreign keys referencing the GameID in the Game table, representing the games linked in the post.

figure 2 Assumption for UML Diagram

1.2 Description for Each Relationship and Cardinality



Entity 1	Entity 2	Cardinality	Relationship
Posts	PostComments	1-many	One post can have many comments, while one comment corresponds to one post.
Posts	Users	1-many	One user can have many posts, while one post corresponds to one user.
PostComments	Users	1-many	One user can make many comments, while one comment corresponds to one user.
Users	SearchHistorys	1-many	One user can search for many times, while one search history corresponds to one user.
Users	ViewedGames	1-many	One user can view many games for many times, while one view corresponds to one user.
ViewedGames	Games	1-many	One game can be viewed many times, while one view corresponds to one game.
Users	Reviews	1-many	One user can post many reviews, while one view corresponds to one user.
Reviews	Games	1-many	One game can receive many reviews, while one review corresponds to one game.
Posts	Games	many-many	One game can have many related posts, and one post can include up to three games in our setting.

figure 3 Description for relationship and cardinality

2 Process of Normalization

2.1 Initial Data Schema

Reviews(ReviewID: INT [PK], Rating: INT, Comment: TEXT, GameID: INT [PK], Title: VARCHAR(50), ShortDescription: TEXT, DetailedDescription: TEXT, SupportedLanguages: VARCHAR(100), Publisher: VARCHAR(30), IGNRating: FLOAT, BackgroundImage: VARCHAR(100), PosterImage: VARCHAR(100), Trailer: VARCHAR(100), PurchaseLink: VARCHAR(100), InitialPrice: FLOAT, FinalPrice: FLOAT, EmbeddingVectors: TEXT)

SearchHistory (SearchID: INT [PK], Query: VARCHAR(60), Timestamp: DATETIME, UserID: INT)

ViewedGames (ViewedID: INT [PK], GameID: INT, Timestamp: DATETIME, UserID: INT, Username: VARCHAR(20), Password: VARCHAR(30), Email: VARCHAR(30), Gender: INT, Region: VARCHAR(50), Age: INT, UserLanguages: VARCHAR(100))

PostComments (CommentID: INT [PK], Comment: TEXT, PostID: INT, UserID: INT, PostContent: TEXT, PostLikes: INT)

2.2 Decomposition to BCNF

2.2.1 Reviews

Reviews (ReviewID, Rating, Comment, GameID, Title, ShortDescription, DetailedDescription, SupportedLanguages, Publisher, IGNRating, BackgroundImage, PosterImage, Trailer, PurchaseLink, InitialPrice, FinalPrice, EmbeddingVectors)

FD1: ReviewID **->** Rating, Comment, GameID, Title, ShortDescription, DetailedDescription, SupportedLanguages, Publisher, IGNRating, BackgroundImage, PosterImage, Trailer, PurchaseLink, InitialPrice, FinalPrice, EmbeddingVectors

FD2: GameID **->** Title, ShortDescription, DetailedDescription, SupportedLanguages, Publisher, IGNRating, BackgroundImage, PosterImage, Trailer, PurchaseLink, InitialPrice, FinalPrice, EmbeddingVectors

Split on FD2:

- **Reviews** (ReviewID, Rating, Comment, GameID,)
- **Games** (GameID, Title, ShortDescription, DetailedDescription, SupportedLanguages, Publisher, IGNRating, BackgroundImage, PosterImage, Trailer, PurchaseLink, InitialPrice, FinalPrice, EmbeddingVectors**)**

2.2.2 ViewedGames

ViewedGames (ViewedID, GameID, Timestamp, UserID, Username, Password, Email, Gender, Region, Age, UserLanguages)

FD1: ViewedID -> GameID, Timestamp, UserID, Username, Password, Email, Gender, Region, Age, UserLanguages

FD2: UserID -> Username, Password, Email, Gender, Region, Age, UserLanguages

Split on FD2:

- **ViewedGames** (ViewedID, GameID, Timestamp, UserID)
- **Users** (UserID, Username, Password, Email, Gender, Region, Age, UserLanguages**)**

2.2.3 PostComments

PostComments (CommentID, Comment, PostID, UserID, PostContent, PostLike)

FD1: CommentID -> Comment, PostID, UserID, PostContent, PostLike

FD2: PostID -> UserID, PostContent, PostLike

Split on FD2:

- **PostComments** (CommentID, Comment, PostID)
- **Posts (**PostID, UserID, PostContent, PostLike)

2.3 Why BCNF (Compared to 3NF)

- BCNF provides a stricter schema, which ensures more data integrity and reduces more redundancy.
- BCNF provides a schema that is easier to understand and maintain.
- In our design, there is no complex functional dependency between attributes. So using BCNF will not cause a big loss in functional dependency.

3 Relational Schema

Games (GameID: int [PK], Title: varchar[40], ShortDescription: text, DetailedDescription: text, SupportedLanguages: varchar[200], Publisher: varchar[30], IGNRating: float, BackgroundImage: varchar[100], PosterImage: varchar[100], Trailer: varchar[100], PurchaseLink: varchar[], InitialPrice: float, FinalPrice: float, EmbeddingVectors: json)

Users (UserID: int [PK], Username: varchar[20], Password: varchar[30], Email: varchar[30], Gender: int, Age: int, UserLanguage: varchar[50])

SearchHistory (SearchID: int [PK], UserID: int [FK to Users.UserID], Query: varchar[60], Timestamp: datetime)

ViewedGames (ViewedID: int [PK], UserID: int [FK to Users.UserID], GameID: int [FK to Games.GameID], Timestamp: datetime)

Reviews (ReviewID: int [PK], UserID: int [FK to Users.UserID], GameID: int [FK to Games.GameID], ReviewComment: text, Rating: int)

Posts (PostID: int [PK], UserID: int [FK to Users.UserID], PostContent: text, Likes: int)
Post_Game (PostID: int [PK] [FK to Post.PostID], GameID: int [PK] [FK to Games.GameID])

PostComments (CommentID: int [PK], PostID: int [FK to Posts.PostID], UserID: int [FK to Users.UserID], Comment: text)