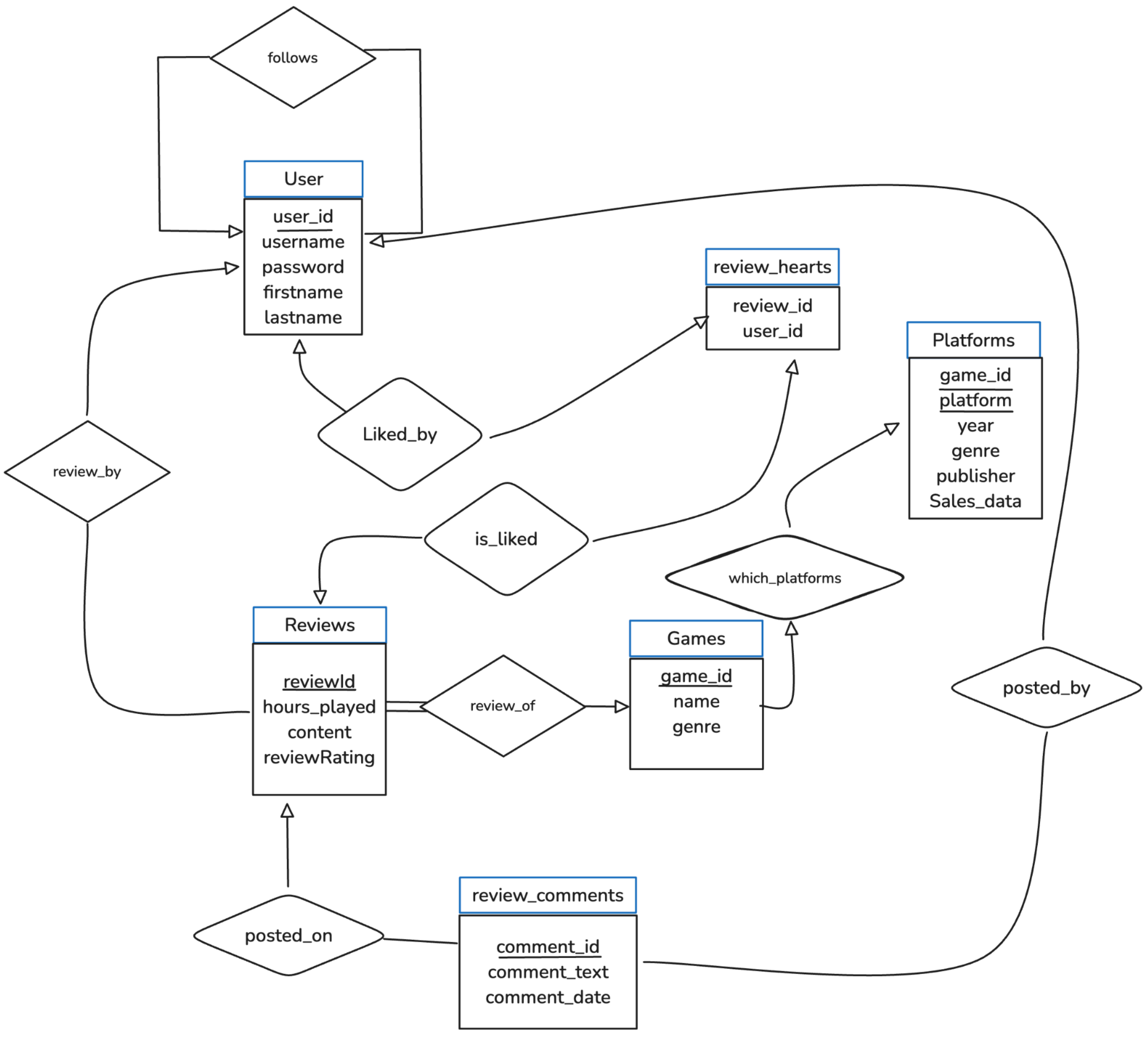
Database Design

ER Model



Resulting relations after converting the ER model to relations
Functional dependencies that you have identified
Normalization steps and final normalized relations.

Relations:
games(game_id, name, genre)
user(userId, username, password, firstName, lastName)
follows(followingId, followedId)
platforms(game_id, platform, year, publisher, na_sales, eu_sales, jp_sales, other_sales, global_sales)

review(reviewId, userId, game_id, hoursPlayed, reviewRating, content, postDate, heartsCount, commentsCount, isHearted, isBookmarked)
review_hearts(userId, reviewId)
review_comments(commentsId, reviewId, userId, commentText, commentDate)

Full table:
(game_id, game_name, game_genre, platform, year, publisher, na_sales, eu_sales, jp_sales, other_sales, global_sales, userId, username, password, firstName, lastName, followingId, followedId, reviewId, review_content, review_rating, hoursPlayed, postDate, heartsCount, commentsCount, isHearted, isBookmarked, heart_userId, commentId, commentText, commentDate)

FD:
userId → username, password, firstName, lastName
game_id → name, genre
Game_id, platform → year, publisher, na_sales, eu_sales, jp_sales, other_sales
reviewId → userId, gameId, hoursPlayed, reviewRating, content, postDate
commentsId → reviewId, userId, commentText, commentDate

3NF:

1. Split rhs

userId → username
userId → password
userId → firstName
userId → lastName
Game_id → name
Game_id → genre
Game_id, platform → year
Game_id, platform → publisher
Game_id, platform → na_sales
Game_id, platform → eu_sales
Game_id, platform → jp_sales
Game_id, platform → other_sales
reviewId → userId
reviewId → gameId
reviewId → hoursPlayed
reviewId → reviewRating
reviewId → content
reviewId → postDate
commentsId → reviewId
commentsId → userId
commentsId → commentText
commentsId → commentDate

2. Remove from LHS if redundant

Game_id, platform → year
Game_id, platform → publisher
Game_id, platform → na_sales
Game_id, platform → eu_sales
Game_id, platform → jp_sales
Game_id, platform → other_sales

- Can't get these without platform, game_id cannot determine platform. Platform is not redundant, cannot be removed.

3. Try to remove each FD and see if remaining FDs can still infer removed FD

Test comments → userId
- reviewId → userId exists. Therefore comments → userId is redundant

Result:
userId → username
userId → password
userId → firstName
userId → lastName
Game_id → name
Game_id → genre
Game_id, platform → year
Game_id, platform → publisher
Game_id, platform → na_sales
Game_id, platform → eu_sales
Game_id, platform → jp_sales
Game_id, platform → other_sales
reviewId → userId
reviewId → gameId
reviewId → hoursPlayed
reviewId → reviewRating
reviewId → content
reviewId → postDate
commentsId → reviewId
commentsId → userId (removed)
commentsId → commentText
commentsId → commentDate

4. Merge FDs with same LHS

userId → username, password, firstName, lastName
game_id → name, genre
Game_id, platform → year, publisher, na_sales, eu_sales, jp_sales, other_sales

reviewId → userId, gameId, hoursPlayed, reviewRating, content, postDate
commentsId → reviewId, commentText, commentDate

5. Merge into tables

games(game_id, name, genre)
user(userId, username, password, firstName, lastName)
follows(followingId, followedId)
platforms(game_id, platform, year, publisher, na_sales, eu_sales, jp_sales, other_sales, global_sales)
review(reviewId, userId, game_id, hoursPlayed, reviewRating, content, postDate, heartsCount, commentsCount)
review_hearts(userId, reviewId)
review_comments(commentsId, reviewId, commentText, commentDate)

We found that we needed differences from the 3NF implementation. 3NF wanted us to remove the review_comments(userId) attribute, as the userId could be found from the reviewId, but in practice, this userId under the comment was the userId of the person who posted the comment, which could be different from the person who wrote the review. As this was often the case, we decided to use the ER diagram version which kept this relation. The other relations and their attributes were to our liking.