Application Database Design

Derek Avery and Colton Giordano

Penn State University

October 31, 2023

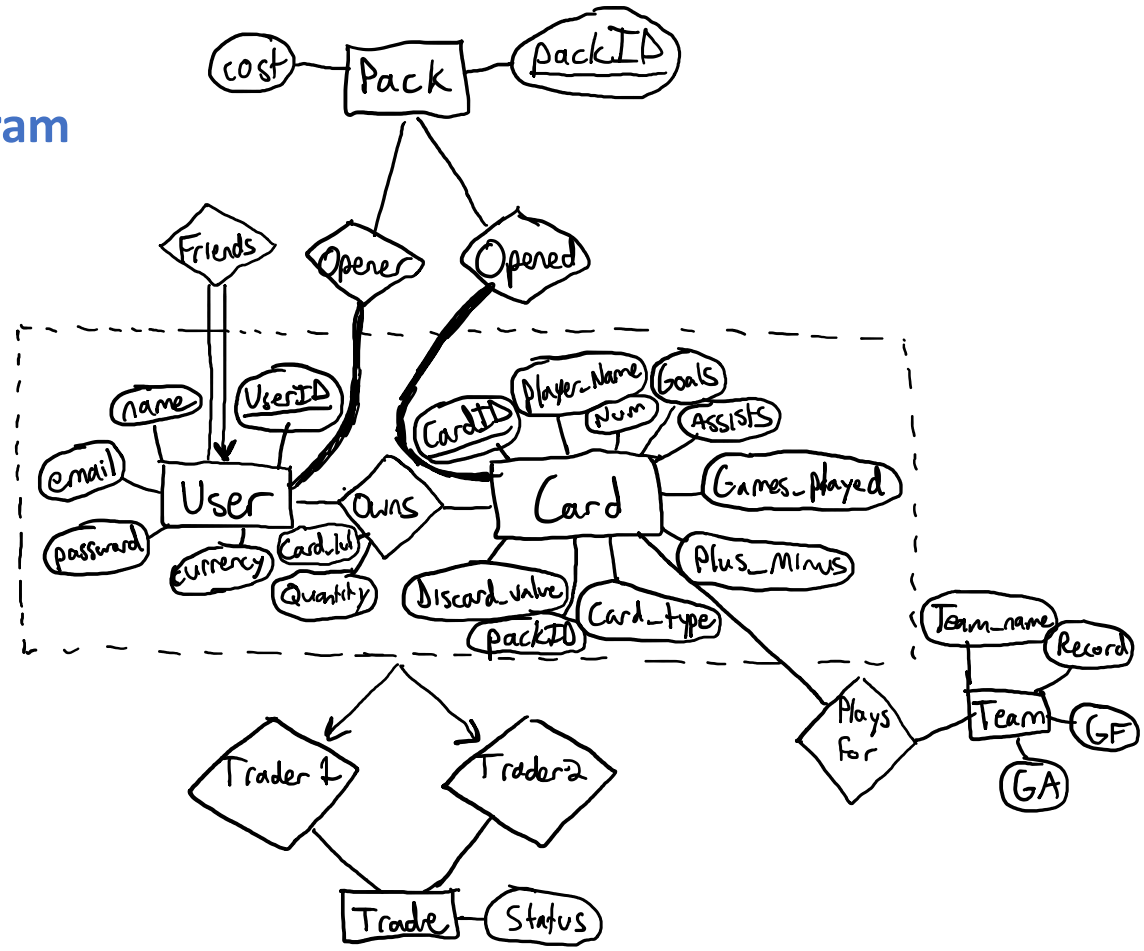
Digital Hockey Trading Card System (DHTC)

ER Modeling Procedure

- Identify Entities and Attributes
  - User
    - UserID (Unique) PK
    - Name
    - Email
    - Password
    - Currency
  - Card
    - CardID (Unique) PK
    - PackID FK-Pack
    - Player_name
    - Number
    - Goals
    - Assists
    - Games_played
    - Plus_Minus
    - Card_type
    - Discard_value
    - Open_odds
  - Pack
    - PackID (Unique) PK
    - Cost
  - Team
    - TeamName (Unique) PK
    - Record
    - Goals_for
    - Goals_against
  - Trade
    - TradeID (Unique) PK
    - Date
    - Status

- **Identify Relations**
  - **Owns (User x Card)**
    - **UserID FK-User**
    - **CardID FK-Card**
    - **Card_lvl**
    - **Quantity**
  - **Trader1 (Owns [Aggregation] x Trade)**
    - **TradeID FK-Trade**
    - **UserID FK-User**
    - **CardID FK-Card**
  - **Trader2 (Owns [Aggregation] x Trade)**
    - **TradeID FK-Trade**
    - **UserID FK-User**
    - **CardID FK-Card**
  - **Friends (User x User)**
    - **UserID1 FK-User**
    - **UserID2 FK-User**
  - **Opens (User x Pack x Card)**
    - **UserID FK-User**
    - **CardID FK-Card**
    - **PackID FK-Pack**
  - **PlaysFor (Card x Team)**
    - **Team_name FK-Team**
    - **CardID FK-Card**
- **Relationship constraints**
  - **One user trades one card or currency with another user**
  - **Each user can be friends with many other users**
  - **Each user can own many cards**
  - **Each card should play for exactly one team**
  - **Each user can open many packs**

ER Diagram


cost
Pack
packID
Friends
Opener
Opened
name
UserID
email
User
password
currency
CardID
Quantity
Owns
CardID
Player_Name
Goals
Num
Assists
Card
Games_Played
Plus_Minus
Discard_value
packID
Card_type
Trader 1
Trader 2
Trade
Status
Plays for
Team_name
Record
Team
GF
GA

DDL

```
CREATE TABLE User (
userID INT AUTO_INCREMENT,
name VARCHAR(50),
email VARCHAR(50) UNIQUE,
password VARCHAR(50),
currency INT NOT NULL,
PRIMARY KEY (userID))

CREATE TABLE Card (
cardID INT AUTO_INCREMENT UNIQUE,
packID INT,
player_name VARCHAR(50) UNIQUE,
num INT,
goals INT DEFAULT 0,
assists INT DEFAULT 0,
games_played INT DEFAULT 0,
plus_minus INT DEFAULT 0,
card_type VARCHAR(30),
discard_value INT DEFAULT 0,
PRIMARY KEY (cardID)
FOREIGN KEY (packID) REFERENCE Pack(packID)
ON DELETE RESTRICT)

CREATE TABLE Team (
team_name VARCHAR(50),
record VARCHAR(20),
gf INT DEFAULT 0,
ga INT DEFAULT 0
PRIMARY KEY (team_name))

CREATE TABLE Pack (
packID INT,
Cost INT DEFAULT 0
PRIMARY KEY (packID))

CREATE TABLE Trade (
tradeID INT AUTO_INCREMENT,
date DATETIME,
status VARCHAR(20),
PRIMARY KEY (tradeID))
```

```
CREATE TABLE Owns (
userID INT,
cardID INT,
card_lvl INT DEFAULT 1,
quantity INT DEFAULT 1,
PRIMARY KEY (userID, cardID),
FOREIGN KEY (userID) REFERENCES User(userID)
ON DELETE CASCADE,
FOREIGN KEY (cardID) REFERENCES Card(cardID)
ON DELETE RESTRICT)

CREATE TABLE Trader1(
tradeID INT,
userID INT,
cardID INT,
PRIMARY KEY (traded, userID, cardID),
FOREIGN KEY (tradeID) REFERENCES Trade(tradeID),
FOREIGN KEY (userID) REFERENCES User(userID)
ON DELETE CASCADE,
FOREIGN KEY (cardID) REFERENCES Card(cardID)
ON DELETE RESTRICT)

CREATE TABLE Trader2(
tradeID INT,
userID INT,
cardID INT,
PRIMARY KEY (traded, userID, cardID),
FOREIGN KEY (tradeID) REFERENCES Trade(tradeID),
FOREIGN KEY (userID) REFERENCES User(userID)
ON DELETE CASCADE,
FOREIGN KEY (cardID) REFERENCES Card(cardID)
ON DELETE RESTRICT)

CREATE TABLE Friends(
userID1 INT,
userID2 INT,
PRIMARY KEY (userID1, userID2),
FOREIGN KEY (userID1, userID2) REFERENCES User(userID)
ON DELETE CASCADE)
```

```
CREATE TABLE Opens(
userID INT,
cardID INT,
packID INT,
PRIMARY KEY (userID, cardID, packID)
FOREIGN KEY (userID) REFERENCES User(userID)
ON DELETE CASCADE,
FOREIGN KEY (cardID) REFERENCES Card(cardID)
ON DELETE RESTRICT,
FOREIGN KEY (packID) REFERENCES Pack(packID)
ON DELETE RESTRICT)
```

```
CREATE TABLE PlaysFor(
team_name VARCHAR(50),
cardID INT,
PRIMARY KEY (team_name, cardID)
FOREIGN KEY (team_name) REFERENCES Team(team_name),
FOREIGN KEY (cardID) REFERENCES Card(cardID))
```

Schema Refinement (Non-trivial FDs)

User

- **Anomalies**
  - **There will be no anomalies as each record is unique to a certain user. Therefore, when inserting, deleting, and updating a user, it will only happen on one record. There is no redundant information as all information is specific to each user.**
- **Functional Dependencies**
  - **{userID} -> {name, email, password, currency}**
  - **{email} -> {userID, name, password, currency}**
- **No refinement needed.**

Card

- **Anomalies**
  - **There will be no anomalies as each record is unique to a certain player. Inserting, deleting, and updating will only happen on one record. There is no redundant information as all information is tied to the specific player.**
- **Functional Dependencies**
  - **{cardID} -> {packID, player_name, num, goals, assists, games_played, plus_minus, card_type, discard_value}**
  - **{player_name} -> {cardID, packID, num, goals, assists, games_played, plus_minus, card_type, discard_value}**
- **No refinement needed.**

Team

- **Anomalies**
  - **There will be no anomalies as each record is unique to a certain team. Inserting, updating, and deleting will occur with one record. There is no redundant information as team information is tied to each specific team.**
- **Functional Dependencies**
  - **{team_name} -> {record, gf, ga}**
- **No refinement needed.**

Pack

- **Anomalies**
  - **There will be no anomalies as each record is unique to a certain pack. Inserting, updating, and deleting will occur on one record. There is no redundant information as each pack has one cost.**
- **Functional Dependencies**
  - **{team_name} -> {record, gf, ga}**
- **No refinement needed.**

Trade

- **Anomalies**
  - **There will be no anomalies as each record is unique to a certain trade. Inserting, updating, and deleting will occur on one record when a trade is requested. There is no redundant information as each trade occurs on a certain date.**
- **Functional Dependencies**
  - **{tradeID} -> {date, status}**
- **No refinement needed.**

Owns

- **Anomalies**
  - **There will be no anomalies as each record is unique to a certain user that owns a certain card. Inserting, updating, and deleting will occur when a user obtains a card or players trade cards the quantity will change. There is no redundant information as each trade occurs on a certain date.**
- **Functional Dependencies**
  - **{userID, cardID} -> {card_lvl}**
- **No refinement needed.**

Trader1

- **Anomalies**
  - **There will be no anomalies as each record is unique to a certain trader of a card. Inserting, updating, and deleting will occur on one record when a trade is requested. There is no redundant information as each trade needs a trader that trades a certain card.**
- **Functional Dependencies**
  - **No non-trivial functional dependencies / All attributes are keys**
- **No refinement needed.**

Trader2

- **Anomalies**
  - **There will be no anomalies as each record is unique to a certain trader of a card. Inserting, updating, and deleting will occur on one record when a trade is requested. There is no redundant information as each trade needs a trader that trades a certain card.**
- **Functional Dependencies**
  - **No non-trivial functional dependencies / All attributes are keys.**
- **No refinement needed.**

Friends

- **Anomalies**
  - **There will be no anomalies as each record is unique to a certain pair of users. Inserting, updating, and deleting will occur on one record when one user friends another. There is no redundant information as each user pair needs a record to relate them as friends.**
- **Functional Dependencies**
  - **No non-trivial functional dependencies / All attributes are keys.**
- **No refinement needed.**

Opens

- **Anomalies**
  - **There will be no anomalies as each record is unique to a certain user opening a card from a pack. Inserting, updating, and deleting will occur on one record when a card is opened from a pack. There is no redundant information as each pack opened needs a record of the user who opened a specific pack and received a certain card.**
- **Functional Dependencies**
  - **No non-trivial functional dependencies / All attributes are keys.**
- **No refinement needed.**

PlaysFor

- **Anomalies**
  - **There will be no anomalies as each record is unique to a card that plays for a certain team. Inserting, updating, and deleting will occur on one record to identify what team a player plays for. There is no redundant information as each card will play for one team.**
- **Functional Dependencies**
  - **No non-trivial functional dependencies / All attributes are keys.**
- **No refinement needed.**

SQL Queries for Functionalities ($ signifies variable)

- **Sign Up**
  - **Insert user into table on creation of account**

```
START TRANSACTION
INSERT INTO User (userID, name, email, password, currency)
VALUES (userID, $name, $email, $password, currency)
COMMIT
```

- **Sign In**
  - **Check user exists**

```
START TRANSACTION
SELECT *
FROM User U
WHERE U.email = $email AND U.password = $password
COMMIT
```

- **Account Manager**
  - **Make updates to account as needed**

START TRANSACTION
UPDATE User
SET name = $name, email = $email, $password = password
WHERE userID = $userID
COMMIT

- **Card Viewer**
  - **View individual cards**

START TRANSACTION
SELECT C.cardID, C.packID, C.player_name, C.num, C.goals, C.assists, C.games_played, C.plus_minus, C.card_type, C.discard_value
FROM Card C, Owns O
WHERE O.userID = $userID AND O.cardID = C.cardID AND C.cardID = $cardID
COMMIT

- **Collection Management**
  - **View your entire collection**

START TRANSACTION
SELECT C.cardID, C.packID, C.player_name, C.num, C.goals, C.assists, C.games_played, C.plus_minus, C.card_type, C.discard_value
FROM Card C, Owns O
WHERE O.userID = $userID AND O.cardID = C.cardID
COMMIT

- **Currency**
  - **Get user currency**

START TRANSACTION
SELECT U.currency
FROM User U
WHERE U.userID = $userID
COMMIT

- **Pack Opening**
  - **Update user currency and subtract cost of pack they are opening (rollback if they don't have enough)**

```
START TRANSACTION
SELECT U.currency
FROM User U
WHERE U.userID = $userID

SELECT P.Cost
FROM Pack P
WHERE P.packID = $packID

(UPDATE User
SET currency = currency – P.cost
WHERE U.userID = $userID) *Check if currency >= cost and if not ROLLBACK
COMMIT
```

  - **Get cards from a certain pack**

```
START TRANSACTION
SELECT C.cardID
FROM Pack P, User U, Card C
WHERE U.userID = $userID AND P.packID = C.packID AND P.packID = $packID
COMMIT
```

  - **Insert card into owns table when they open a pack**

```
*If they do not on this card yet
START TRANSACTION
INSERT INTO Owns (userID, cardID, card_lvl, quantity)
VALUES ($userID, $cardID, $card_lvl, $quantity)
COMMIT
```

  - **Update quantity of card that the own**

```
*If they own the card already
START TRANSACTION
UPDATE Owns as O
SET quantity = quantity + 1
WHERE O.userID = $userID AND O.cardID = $cardID AND O.card_lvl = $card_lvl
COMMIT
```

- **Friends**
  - **Get who is friends with who**

```
START TRANSACTION
SELECT Friend.userID
FROM User U, User Friend, Friends F
WHERE U.userID = F.userID1 AND Friend.userID = F.userID2
COMMIT
```

- Insert them into friends table when users become friends

START TRANSACTION
INSERT INTO Friends (userID1, userID2)
VALUES ($userID1, $userID2)
COMMIT

- ## Trading

  - When a trade is proposed keep a record of it

START TRANSACTION
INSERT INTO Trades(date, status)
VALUES ($date, $status)
COMMIT

  - After a trade has been accepted and the cards are traded

*If quantity of owned card greater than 0
START TRANSACTION
UPDATE Owns as O
SET quantity = quantity + 1
WHERE O.userID = $userID AND O.cardID = $cardID AND O.card_lvl = $card_lvl
COMMIT

*If card doesn't exist in owns
START TRANSACTION
INSERT INTO Owns(userID, cardID, card_lvl, quantity)
VALUES ($userID, $cardID, $card_lvl, $quantity)
COMMIT

- ## Fire Pit

  - Update Owns table to reflect the burning of a card

*If card quantity greater than 1
START TRANSACTION
UPDATE Owns as O
SET quantity = quantity - 1
WHERE O.userID = $userID AND O.cardID = $cardID AND O.card_lvl = $card_lvl
COMMIT

*If card quantity equal to 1
START TRANSACTION
DELETE
FROM Owns O
WHERE O.userID = $userID AND O.cardID = $cardID AND O.card_lvl = $card_lvl
COMMIT

- **Level Up**
    - **Update owns table depending on what cards you have and their levels**

***If the leveling up card quantity greater than 1**
START TRANSACTION
UPDATE Owns as O
SET quantity = quantity + 1
WHERE O.userID = $userID AND O.cardID = $cardID AND O.card_lvl = $card_lvl
COMMIT

***If leveling up card quantity greater than 1**
START TRANSACTION
INSERT INTO Owns(userID, cardID, card_lvl, quantity)
VALUES ($userID, $cardID, $card_lvl, $quantity)
COMMIT

***If the previous lower level card quantity greater than 1**
START TRANSACTION
UPDATE Owns as O
SET quantity = quantity - 1
WHERE O.userID = $userID AND O.cardID = $cardID AND O.card_lvl = $card_lvl
COMMIT

***If the previous lower level card quantity equal to 1**
START TRANSACTION
DELETE
FROM Owns O
WHERE O.userID = $userID AND O.cardID = $cardID AND O.card_lvl = $card_lvl
COMMIT