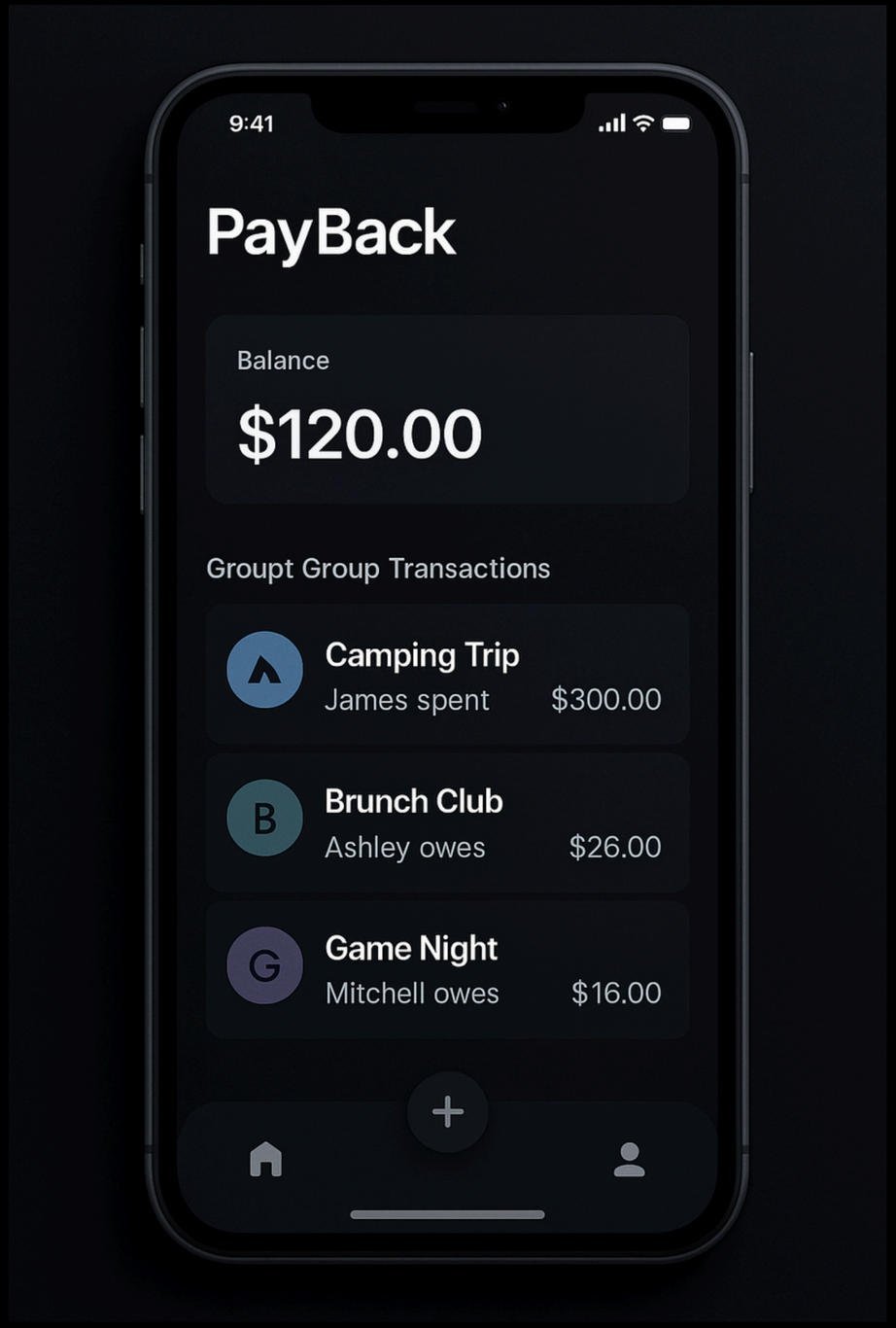
PayBack

A database-backed expense settling app

CS3200

Shahid, Malia, Dylan Mikulka, Ansh Aggarwal

TABLE OF CONTENTS



PROBLEM

Managing group expenses is challenging because it's...

- Hard to track who paid what
- Confusing to calculate who owes whom
- Easy to make mistakes
- Requires many transactions to settle up

GOAL

- Make tracking group expenses easy
- Split costs fairly among members
- Use an optimization algorithm to reduce the number of payments needed

POSSIBLE USES

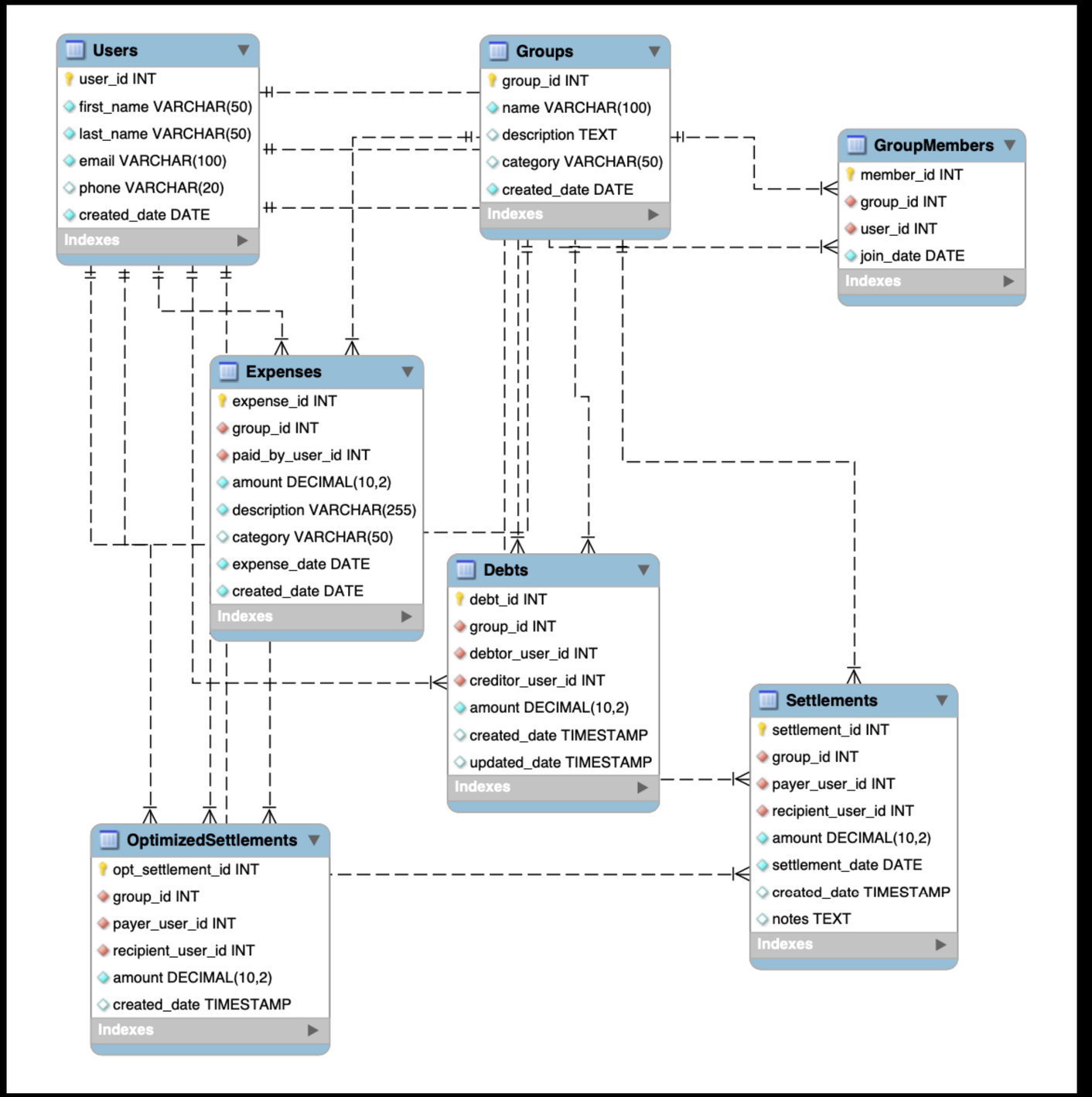
- **Dinner with Friends**
  - One person pays, app splits the cost
  - No more Venmo chaos

- **Group Travel (Spring Break, Road Trips)**
  - Dylan pays for flights, Malia books the Airbnb, Shahid rents the car
  - Without optimization: 6+ separate payments
  - With PayBack: Only 3 payments needed

- **Housing / Roommates**
  - Monthly rent, utilities, groceries, supplies
  - Clear history of all expenses
  - Easy to see who owes whom at any time

DATABASE SCHEMA

- 7 tables storing users, groups, expenses, and debts
- Optimizes settlement paths to minimize transactions
- Automatic debt calculation after each expense



DATA SOURCES

users.csv
- Contains all users in the system
- Includes user IDs, names, contact info, and account creation dates
- Used to identify who participates in each group and who pays for expenses

groups.csv
- List of all groups (e.g., Miami Trip, Boston Foodies, Apartment Bills)
- Includes the group ID, group name, creator, and creation date
- Used to link expenses and members to the correct group

group_members.csv
- Maps users to the groups they belong to
- Includes user ID, group ID, and join date
- Used to determine how many people share each expense and calculate fair splits

4. expenses.csv
- Contains all recorded expenses across all groups
- Includes payer ID, group ID, amount, date, and description
- Used to compute individual contributions, totals per group, and settlement balances

SQL QUERYS

Top Spender per Group

```sql
SELECT
    g.name AS group_name,
    CONCAT(u.first_name, ' ', u.last_name) AS top_spender,
    SUM(e.amount) AS total_paid
FROM Expenses e
JOIN `Groups` g ON e.group_id = g.group_id
JOIN Users u ON e.paid_by_user_id = u.user_id
GROUP BY g.group_id, g.name, u.user_id, u.first_name, u.last_name
HAVING SUM(e.amount) = (
    SELECT MAX(group_total)
    FROM (
        SELECT e2.group_id, e2.paid_by_user_id, SUM(e2.amount) AS group_total
        FROM Expenses e2
        WHERE e2.group_id = g.group_id
        GROUP BY e2.group_id, e2.paid_by_user_id
    ) AS subquery
)
ORDER BY total_paid DESC;
```

Net Balance Calculator

```sql
SELECT
    CONCAT(u.first_name, ' ', u.last_name) AS user_name,
    COALESCE(owed_to.total, 0) AS total_owed_to_them,
    COALESCE(they_owe.total, 0) AS total_they_owe,
    COALESCE(owed_to.total, 0) - COALESCE(they_owe.total, 0) AS net_balance,
    CASE
        WHEN COALESCE(owed_to.total, 0) - COALESCE(they_owe.total, 0) > 0
        THEN 'Net Creditor'
        ELSE 'Net Debtor'
    END AS status
FROM Users u
LEFT JOIN (
    SELECT creditor_user_id, SUM(amount) AS total
    FROM Debts
    GROUP BY creditor_user_id
) owed_to ON u.user_id = owed_to.creditor_user_id
LEFT JOIN (
    SELECT debtor_user_id, SUM(amount) AS total
    FROM Debts
    GROUP BY debtor_user_id
) they_owe ON u.user_id = they_owe.debtor_user_id
WHERE COALESCE(owed_to.total, 0) > 0 OR COALESCE(they_owe.total, 0) > 0
ORDER BY net_balance DESC;
```

OPTIMIZED SETTLEMENTS

Our Solution:

- Calculate each person's NET balance (what they're owed minus what they owe)
- Match biggest debtors with biggest creditors
- Generate the minimum transactions needed

The Problem:

- In a group of 4 people, expenses create a web of debts
- Person A owes B, B owes C, C owes D, D owes A...
- Without optimization: 6+ separate payments needed

Without Optimization	With Optimizations
Ansh pays Dylan $315	Ansh pays Dylan $444
Ansh pays Malia $186	Shahid pays Dylan $353
Ansh pays Shahid $74	Ansh pays Malia $186
Shahid pays Dylan $241	All Done!
Shahid pays Malia $112	
Malia pays Dylan $129	
6 transactions	3 transactions

– THANK YOU FOR WATCHING –