Forgot

Db schema config

1. `users` Table

- Stores user details.
- `user_id` is the primary key, and each user is identified uniquely by either `email` or `firebase_id`.
- This table will interact with all other tables through `user_id`.

2. `groups` Table

- Stores information about groups, including who created the group.
- `group_id` is the primary key.
- `created_by` is a foreign key that references `users(user_id)` (creator of the group).
- Groups are an important part of your system and allow grouping of transactions and balances.

3. `group_members` Table

- Manages the relationship between users and groups.
- `group_member_id` is the primary key.
- `user_id` references `users(user_id)`.
- `group_id` references `groups(group_id)`.
- The `joined_date` tracks when a user joins the group.
- This table will be essential for managing group participation and group-specific interactions.

4. `transactions` Table

- Manages the actual lending and borrowing activities between users.
- `transaction_id` is the primary key.
- `lender_id` and `borrower_id` reference `users(user_id)`.
- `group_id` references `groups(group_id)` and is optional, allowing transactions to be related to a group.
- The `status` field can be used to track the state of a transaction (pending, completed, etc.), and `purpose` can describe the reason for the transaction (loan, settlement, etc.).

5. `transaction_splits` Table

- Used to break down transactions for splitting amounts among multiple participants (in case of group transactions).
- `transaction_split_id` is the primary key.
- `transaction_id` references `transactions(transaction_id)`.
- `user_id` references `users(user_id)` and stores the individual amounts each user owes.
- This table helps in splitting amounts for group expenses.

6. `balances` Table

- Tracks the owed and lent amounts for each user, for both solo transactions and group-related transactions.
- `balance_id` is the primary key.
- `user_id` references `users(user_id)`.
- `group_id` is optional and links balances to a specific group.
- `owed_amount` and `lent_amount` allow you to keep track of what a user owes and has lent.

7. `requests` Table

- Manages money requests between users, such as one user requesting money from another.
- `request_id` is the primary key.
- `sender_id` references `users(user_id)` (who sends the request).
- `receiver_id` references `users(user_id)` (who receives the request).
- `group_id` is optional and links the request to a specific group.
- `status` indicates whether the request is pending, accepted, or rejected.

8. `settlements` Table

- Used to track settlement of debts between users.
- `settlement_id` is the primary key.
- `user_id` references `users(user_id)` (who is making the settlement).
- `counterparty_id` references `users(user_id)` (who is being settled with).
- `group_id` is optional and links settlements to a group.
- `amount` represents the amount settled.
- This table tracks when debts between users are paid off.

Indexes

The indexes you've added are useful for improving query performance:

- `users`: Indexes for `email` and `firebase_id` ensure that these fields are unique and optimized for searches.
- `group_members`, `transactions`, `transaction_splits`, `balances`: Indexes on `user_id`, `group_id`, `transaction_id` allow fast lookups, especially when dealing with user-group relationships and transactions.

How it all fits together:

1. **User Interaction**:
    - Users are added to the system, and they can create groups or join existing ones.
2. **Group Transactions**:
    - Users within a group can make transactions (e.g., pay for dinner).
    - Transactions can be split among group members.
3. **Balance Management**:
    - The system will track who owes whom and how much.
    - Balances are updated after transactions and settlements.
4. **Requests**:
    - One user can request money from another for a specific transaction or debt.
5. **Settlements**:
    - Users can settle debts by making payments, and these settlements are recorded.

Scenario 1: Solo Transaction

Use Case:

Abhinav lends ₹500 to X for a train ticket.

Step 1: Add Users

- **`users` table**:
    - `user_id: 1, name: Abhinav`
    - `user_id: 2, name: X`

Step 2: Record the Transaction

- Abhinav lends ₹500 to X.
- **`transactions` table**:
    - `transaction_id: 1, lender_id: 1 (Abhinav), borrower_id: 2 (X), group_id: NULL, amount: ₹500, status: 'pending', purpose: 'Train ticket'`

Step 3: Update Balances

- Abhinav's lent amount is updated, and X's owed amount is updated.
- **`balances` table**:
    - For Abhinav:
        - `balance_id: 1, user_id: 1, group_id: NULL, owed_amount: ₹0, lent_amount: ₹500`
    - For X:
        - `balance_id: 2, user_id: 2, group_id: NULL, owed_amount: ₹500, lent_amount: ₹0`

Step 4: Settlement

- X repays ₹500 to Abhinav.
- **`settlements` table**:
    - `settlement_id: 1, user_id: 2 (X), counterparty_id: 1 (Abhinav), group_id: NULL, amount: ₹500, settlement_date: current_date`
- Update the balances:
    - Abhinav: `lent_amount` reduced to ₹0.
    - X: `owed_amount` reduced to ₹0.
- **`balances` table**:
    - For Abhinav:

        - `balance_id: 1, user_id: 1, group_id: NULL, owed_amount: ₹0, lent_amount: ₹0`
    - For X:
        - `balance_id: 2, user_id: 2, group_id: NULL, owed_amount: ₹0, lent_amount: ₹0`

–

Scenario 2: Group Transaction

Use Case:

Abhinav, X, and YZ go out for dinner, and the total bill is ₹3,000. Abhinav pays the bill, and they split it equally (₹1,000 each).

Step 1: Create a Group

- Abhinav creates a group called **"Dinner with Friends"**.
- **`groups` table**:
    - `group_id: 1, group_name: "Dinner with Friends", created_by: 1 (Abhinav)`

Step 2: Add Members to the Group

- Abhinav adds X and YZ to the group.
- **`group_members` table**:
    - `group_member_id: 1, user_id: 1 (Abhinav), group_id: 1, joined_date: current_date`
    - `group_member_id: 2, user_id: 2 (X), group_id: 1, joined_date: current_date`
    - `group_member_id: 3, user_id: 3 (YZ), group_id: 1, joined_date: current_date`

Step 3: Record the Group Transaction

- Abhinav pays ₹3,000 for dinner.
- **`transactions` table**:
    - `transaction_id: 2, lender_id: 1 (Abhinav), borrower_id: NULL, group_id: 1 (Dinner with Friends), amount: ₹3,000, status: 'completed', purpose: 'Dinner'`
- Split the transaction equally (₹1,000 each for X and YZ).
- **`transaction_splits` table**:
    - `transaction_split_id: 1, transaction_id: 2, user_id: 2 (X), amount: ₹1,000`
    - `transaction_split_id: 2, transaction_id: 2, user_id: 3 (YZ), amount: ₹1,000`

Step 4: Update Balances

- Abhinav has lent ₹3,000.
- X and YZ each owe ₹1,000.
- **`balances` table**:
    - For Abhinav:
        - `balance_id: 3, user_id: 1, group_id: 1, owed_amount: ₹0, lent_amount: ₹3,000`
    - For X:
        - `balance_id: 4, user_id: 2, group_id: 1, owed_amount: ₹1,000, lent_amount: ₹0`
    - For YZ:
        - `balance_id: 5, user_id: 3, group_id: 1, owed_amount: ₹1,000, lent_amount: ₹0`

Step 5: Settlement

- X and YZ settle their dues with Abhinav.

- **`settlements` table**:

    - `settlement_id: 2, user_id: 2 (X), counterparty_id: 1 (Abhinav), group_id: 1, amount: ₹1,000, settlement_date: current_date`
    - `settlement_id: 3, user_id: 3 (YZ), counterparty_id: 1 (Abhinav), group_id: 1, amount: ₹1,000, settlement_date: current_date`
- Update the balances:

    - Abhinav's `lent_amount` reduced to ₹0.
    - X's and YZ's `owed_amount` reduced to ₹0.
- **`balances` table**:

    - For Abhinav:
        - `balance_id: 3, user_id: 1, group_id: 1, owed_amount: ₹0, lent_amount: ₹0`
    - For X:
        - `balance_id: 4, user_id: 2, group_id: 1, owed_amount: ₹0, lent_amount: ₹0`
    - For YZ:
        - `balance_id: 5, user_id: 3, group_id: 1, owed_amount: ₹0, lent_amount: ₹0`

Summary:

- **Solo Transaction**: Abhinav lends ₹500 to X, and X repays it.
- **Group Transaction**: Abhinav pays ₹3,000 for a group dinner, and the amount is split and settled between X, YZ, and Abhinav.