# DB Desgin

**Database Schema Documentation**

## Users Table

* **Purpose**: Stores information about users.

### Columns:

* + user\_id (SERIAL, PRIMARY KEY): Unique identifier for each user.
  + email (VARCHAR(255), UNIQUE): Email address of the user.
  + firebase\_id (VARCHAR(255), UNIQUE): Firebase ID for user authentication.
  + name (VARCHAR(100)): Name of the user.
  + created\_at (TIMESTAMP): Timestamp of when the user was created.

## Groups Table

* **Purpose**: Stores details about groups.

### Columns:

* + group\_id (SERIAL, PRIMARY KEY): Unique identifier for each group.
  + group\_name (VARCHAR(255)): Name of the group.
  + created\_by (INT, NOT NULL, FOREIGN KEY): References

users(user\_id); the user who created the group.

* + created\_at (TIMESTAMP): Timestamp of when the group was created.

## Group Members Table

* **Purpose**: Manages the members of each group.

### Columns:

* + group\_member\_id (SERIAL, PRIMARY KEY): Unique identifier for each group membership.
  + user\_id (INT, NOT NULL, FOREIGN KEY): References users(user\_id); the user in the group.
  + group\_id (INT, NOT NULL, FOREIGN KEY): References

groups(group\_id); the group to which the user belongs.

* + joined\_date (TIMESTAMP): Timestamp when the user joined the group.

## Payment Methods Table

* **Purpose**: Stores information about payment methods used by users.

### Columns:

* + payment\_id (SERIAL, PRIMARY KEY): Unique identifier for each payment method.
  + user\_id (INT, NOT NULL, FOREIGN KEY): References users(user\_id); the user associated with the payment method.
  + payment\_type (VARCHAR(50), NOT NULL): Type of payment method (e.g., 'UPI', 'Bank Account').
  + upi\_id (VARCHAR(100)): UPI ID, used if the payment type is 'UPI'.
  + account\_number (VARCHAR(20)): Account number, used if the payment type is 'Bank Account'.
  + ifsc\_code (VARCHAR(15)): IFSC code for bank transfers, used if the payment type is 'Bank Account'.
  + wallet\_provider (VARCHAR(50)): Optional, provider name for UPI wallet, if applicable.
  + is\_primary (BOOLEAN, DEFAULT FALSE): Indicates whether this is the primary payment method for the user.
  + created\_at (TIMESTAMP): Timestamp when the payment method was created.

## Transactions Table

* **Purpose**: Stores information about monetary transactions between users.

### Columns:

* + transaction\_id (SERIAL, PRIMARY KEY): Unique identifier for each transaction.
  + lender\_id (INT, NOT NULL, FOREIGN KEY): References

users(user\_id); the user who is lending money.

* + borrower\_id (INT, NOT NULL, FOREIGN KEY): References

users(user\_id); the user borrowing money.

* + group\_id (INT, NULL, FOREIGN KEY): References groups(group\_id); optional, for transactions within groups.
  + amount (DECIMAL(10, 2), NOT NULL): The amount of money involved in the transaction.
  + status (VARCHAR(50)): The status of the transaction (e.g., 'pending', 'successful', 'failed', 'retrying').
  + purpose (VARCHAR(255)): The purpose of the transaction (e.g., 'Loan', 'Settlement').
  + payment\_method\_id (INT, FOREIGN KEY): References payment\_methods(payment\_id); payment method used for the transaction.
  + retry\_count (INT, DEFAULT 0): Tracks the number of retry attempts for a failed transaction.
  + failure\_reason (TEXT): Describes the failure reason, if the transaction failed.

## Transaction Splits Table

* **Purpose**: Manages the breakdown of amounts owed by individual participants in a transaction.

### Columns:

* + transaction\_split\_id (SERIAL, PRIMARY KEY): Unique identifier for each split.
  + transaction\_id (INT, NOT NULL, FOREIGN KEY): References

transactions(transaction\_id); the related transaction.

* + user\_id (INT, NOT NULL, FOREIGN KEY): References users(user\_id); the user responsible for the split.
  + amount (DECIMAL(10, 2), NOT NULL): The amount each user owes in the transaction.

## Balances Table

* **Purpose**: Tracks the financial balance of each user, including amounts they owe and lent.

### Columns:

* + balance\_id (SERIAL, PRIMARY KEY): Unique identifier for each balance record.
  + user\_id (INT, NOT NULL, FOREIGN KEY): References users(user\_id); the user whose balance is tracked.
  + group\_id (INT, NULL, FOREIGN KEY): References groups(group\_id); optional, for group-specific balances.
  + owed\_amount (DECIMAL(10, 2), DEFAULT 0): Amount owed by the user.
  + lent\_amount (DECIMAL(10, 2), DEFAULT 0): Amount the user has lent.

## Requests Table

* **Purpose**: Stores requests for money between users.

### Columns:

* + request\_id (SERIAL, PRIMARY KEY): Unique identifier for each request.
  + sender\_id (INT, NOT NULL, FOREIGN KEY): References

users(user\_id); the user sending the request.

* + receiver\_id (INT, NOT NULL, FOREIGN KEY): References

users(user\_id); the user receiving the request.

* + group\_id (INT, NULL, FOREIGN KEY): References groups(group\_id); optional, for group-specific requests.
  + amount (DECIMAL(10, 2), NOT NULL): The amount requested.
  + status (VARCHAR(50), DEFAULT 'pending'): The status of the request (e.g., 'pending', 'accepted', 'rejected').
  + created\_at (TIMESTAMP): Timestamp when the request was created.

## Settlements Table

* **Purpose**: Tracks settlements of debts between users.

### Columns:

* + settlement\_id (SERIAL, PRIMARY KEY): Unique identifier for each settlement.
  + user\_id (INT, NOT NULL, FOREIGN KEY): References users(user\_id); the user making the settlement.
  + counterparty\_id (INT, NOT NULL, FOREIGN KEY): References

users(user\_id); the user being settled with.

* + group\_id (INT, NULL, FOREIGN KEY): References groups(group\_id); optional, for group-specific settlements.
  + amount (DECIMAL(10, 2), NOT NULL): The amount settled.
  + settlement\_date (TIMESTAMP): Timestamp of the settlement.

## Transaction Logs Table

* **Purpose**: Logs actions taken on transactions (e.g., initiated, successful, failed).

### Columns:

* + log\_id (SERIAL, PRIMARY KEY): Unique identifier for each log entry.
  + transaction\_id (INT, NOT NULL, FOREIGN KEY): References transactions(transaction\_id); the transaction related to the log entry.
  + action (VARCHAR(100), NOT NULL): The action taken (e.g., 'Payment Initiated', 'Payment Successful', 'Payment Failed').
  + status (VARCHAR(50)): The status of the action (e.g., 'Pending', 'Success', 'Failed').
  + details (TEXT): Additional details related to the action.
  + error\_details (TEXT): Specific error details in case of failure.
  + timestamp (TIMESTAMP): Timestamp of when the action occurred.

## Payment Notifications Table

* **Purpose**: Stores notifications for users about transaction statuses.

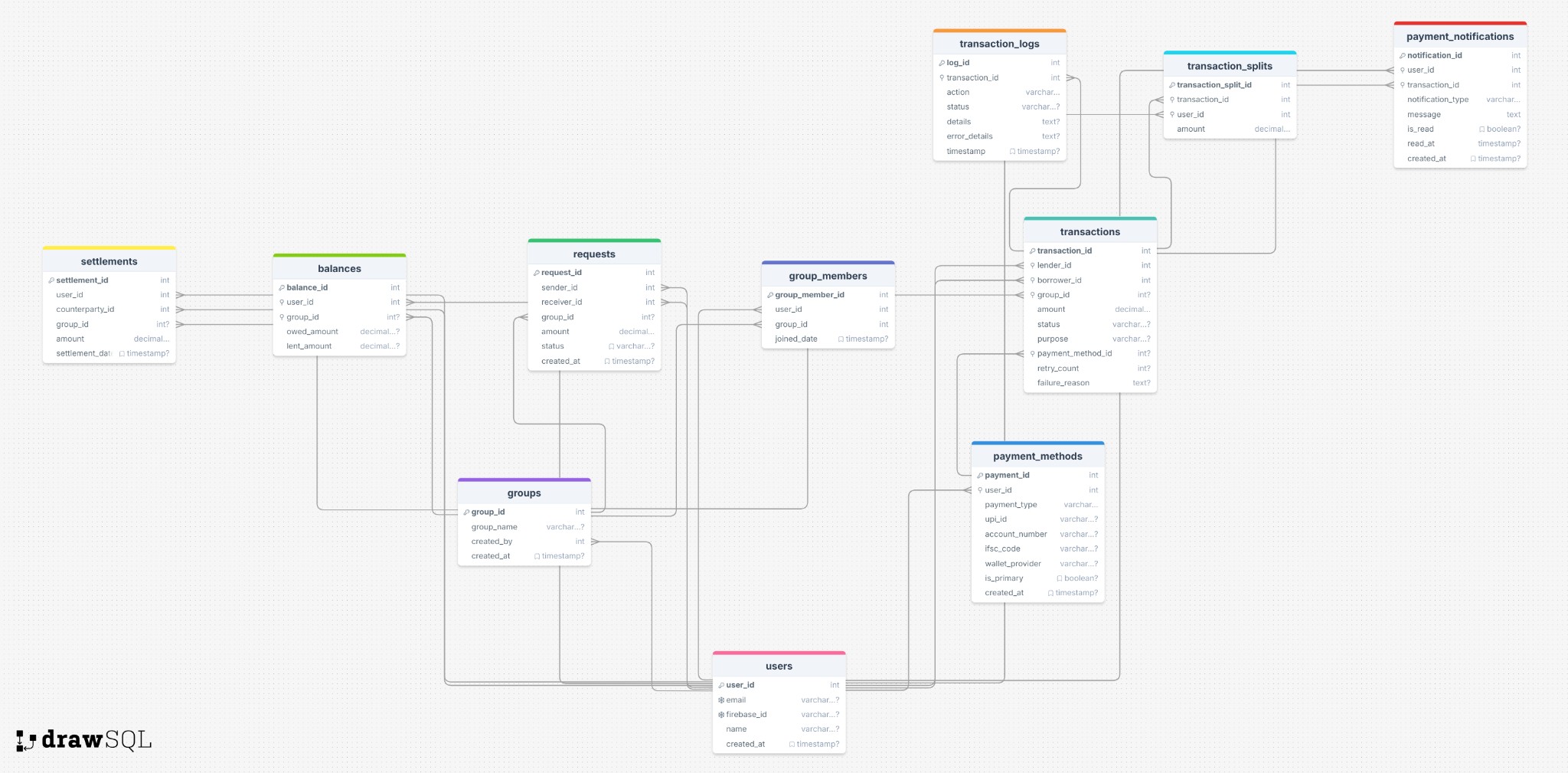
### Columns:

* + notification\_id (SERIAL, PRIMARY KEY): Unique identifier for each notification.
  + user\_id (INT, NOT NULL, FOREIGN KEY): References users(user\_id); the user to whom the notification is sent.
  + transaction\_id (INT, NOT NULL, FOREIGN KEY): References

transactions(transaction\_id); the related transaction.

* + notification\_type (VARCHAR(50), NOT NULL): The type of notification (e.g., 'Success', 'Failure', 'Pending').
  + message (TEXT, NOT NULL): The content of the notification message.
  + is\_read (BOOLEAN, DEFAULT FALSE): Indicates if the user has read the notification.
  + read\_at (TIMESTAMP): Timestamp of when the notification was read.
  + created\_at (TIMESTAMP): Timestamp when the notification was created.

# Image



Flow

## Step 1: User Creation

The process begins with creating user accounts for Abhinav, X, and Y.

### Users Table (each user has unique identifiers and contact details):

* **Abhinav**:
  + user\_id: 1
  + email: [abhinav@example.com](mailto:abhinav@example.com)
  + firebase\_id: abhinav123
  + name: Abhinav
* **X**:
  + user\_id: 2
  + email: [x@example.com](mailto:x@example.com)
  + firebase\_id: x123
  + name: X
* **Y**:
  + user\_id: 3
  + email: [y@example.com](mailto:y@example.com)
  + firebase\_id: y123
  + name: Y

### Purpose:

These users are added to the users table when they sign up through Firebase or email registration.

## Step 2: Adding Payment Methods

Now, let’s add payment methods (UPI and Bank details) for each user.

### Payment Methods Table:

* **Abhinav**:
  + user\_id: 1
  + payment\_method\_type: UPI
  + upi\_id: abhinav@upi
  + bank\_name: NULL
  + account\_number: NULL
  + ifsc\_code: NULL

### Abhinav’s Bank Details:

* + user\_id: 1
  + payment\_method\_type: Bank
  + upi\_id: NULL
  + bank\_name: "State Bank of India"
  + account\_number: "1234567890"
  + ifsc\_code: "SBIN0001234"
* **X**:
* **Y**:
* user\_id: 2
* payment\_method\_type: UPI
* upi\_id: x@upi
* bank\_name: NULL
* account\_number: NULL
* ifsc\_code: NULL
  + user\_id: 3
  + payment\_method\_type: Bank
  + upi\_id: NULL
  + bank\_name: "HDFC Bank"
  + account\_number: "9876543210"
  + ifsc\_code: "HDFC0001234"

## Step 3: Creating Groups and Adding Members

Next, we create a group where Abhinav, X, and Y will participate.

### Groups Table:

* **Group**: "Dinner with Friends"
  + group\_id: 1
  + group\_name: "Dinner with Friends"
  + created\_by: 1 (Abhinav)

### Group Members Table:

* **Abhinav**:
  + group\_member\_id: 1
  + user\_id: 1 (Abhinav)
  + group\_id: 1
* **X**:
* **Y**:
* group\_member\_id: 2
* user\_id: 2 (X)
* group\_id: 1
* group\_member\_id: 3
* user\_id: 3 (Y)
* group\_id: 1

## Step 4: Recording a Group Transaction

Abhinav decides to pay ₹3,000 for the dinner and wants to split it between the three of them.

### Transactions Table:

* **Transaction**: Abhinav pays ₹3,000 for the dinner.
  + transaction\_id: 2
  + lender\_id: 1 (Abhinav)
  + borrower\_id: NULL (Group transaction)
  + group\_id: 1 ("Dinner with Friends")
  + amount: ₹3,000
  + purpose: "Dinner"
  + status: "Completed"

### Transaction Splits Table:

* **Abhinav** (Lender):
  + transaction\_split\_id: 1
  + transaction\_id: 2
  + user\_id: 1 (Abhinav)
  + amount: ₹0 (since Abhinav paid the entire amount)
* **X** (Owes ₹1,000):
  + transaction\_split\_id: 2
  + transaction\_id: 2
  + user\_id: 2 (X)
  + amount: ₹1,000
* **Y** (Owes ₹1,000):
  + transaction\_split\_id: 3
  + transaction\_id: 2
  + user\_id: 3 (Y)
  + amount: ₹1,000

## Step 5: Balances Update

After the group transaction is created, balances are updated for each user to track how much they owe Abhinav.

### Balances Table:

* **Abhinav’s Balance**:
  + user\_id: 1 (Abhinav)
  + group\_id: 1 (Dinner group)
  + owed\_amount: ₹0
  + lent\_amount: ₹3,000 (because Abhinav paid the full ₹3,000)

### X’s Balance:

* + user\_id: 2 (X)
  + group\_id: 1 (Dinner group)
  + owed\_amount: ₹1,000 (X owes ₹1,000)
  + lent\_amount: ₹0

### Y’s Balance:

* + user\_id: 3 (Y)
  + group\_id: 1 (Dinner group)
  + owed\_amount: ₹1,000 (Y owes ₹1,000)
  + lent\_amount: ₹0

## Step 6: Settling the Transaction

At this point, X and Y decide to pay Abhinav the amounts they owe. These payments can be made through the selected payment methods: UPI or Bank.

### Settlements Table:

* **X’s Settlement**:
  + settlement\_id: 1
  + user\_id: 2 (X)
  + counterparty\_id: 1 (Abhinav)
  + group\_id: 1 (Dinner group)
  + amount: ₹1,000 (X pays ₹1,000 to Abhinav)

### Y’s Settlement:

* + settlement\_id: 2
  + user\_id: 3 (Y)
  + counterparty\_id: 1 (Abhinav)
  + group\_id: 1 (Dinner group)
  + amount: ₹1,000 (Y pays ₹1,000 to Abhinav) After settlement, the balances are updated:

### Updated Balances Table:

* **Abhinav’s Balance**:
  + owed\_amount: ₹0
  + lent\_amount: ₹0 (because the full ₹3,000 is now paid back)

### X’s Balance:

* + owed\_amount: ₹0 (X has fully repaid)
  + lent\_amount: ₹0

### Y’s Balance:

* + owed\_amount: ₹0 (Y has fully repaid)
  + lent\_amount: ₹0

## Step 7: Notifications and Transaction Logs

Notifications will be sent to users (Abhinav, X, Y) informing them of the completed transaction and successful payments.

### Transaction Logs Table:

Logs can include entries like:

* **Log**: "Dinner transaction created with ₹3,000 paid by Abhinav."
* **Log**: "X settled ₹1,000 with Abhinav."
* **Log**: "Y settled ₹1,000 with Abhinav."

### Payment Notifications Table:

* **Abhinav**: Notification of payment received from X and Y.
* **X**: Notification of successful settlement with Abhinav.
* **Y**: Notification of successful settlement with Abhinav.

## Step 8: Final Summary of the Entire Process

* **Users**: Abhinav, X, and Y create accounts and add payment methods (UPI/Bank).
* **Groups**: Abhinav creates a group "Dinner with Friends" and adds X and Y.
* **Transactions**: Abhinav pays ₹3,000 for the dinner, splitting the cost with X and Y.
* **Balances**: The balances are updated to reflect how much each user owes.
* **Settlements**: X and Y repay their share of ₹1,000 each to Abhinav.
* **Notifications and Logs**: Users are notified of the transaction and payments, and logs capture the entire flow of the transaction.