Database Schema

The application uses two main entities: `users` and `transactions`. Below is the schema for each entity.

1. Users Table

The `users` table stores information about the users in the system and their current account balance.

- Table Name: `users`

- Columns:

- `user\_id` (String, Primary Key): A unique identifier for each user. Indexed for faster queries.

- `amount` (Float, Non-nullable): The current balance for the user.

SQL Representation

CREATE TABLE users (

user\_id VARCHAR PRIMARY KEY, -- Unique user identifier

amount FLOAT NOT NULL -- The user's balance

);

2. Transactions Table

The `transactions` table logs each balance operation (top-up or deduction) for the users.

- Table Name: `transactions`

- Columns:

- `transaction\_id` (String, Primary Key): A unique identifier for each transaction, indexed for faster queries.

- `user\_id` (String, Foreign Key): The user associated with the transaction. It references the `user\_id` in the `users` table.

- `amount` (Float, Non-nullable): The amount for this transaction. Positive values indicate a top-up, negative values indicate a deduction.

SQL Representation

CREATE TABLE transactions (

transaction\_id VARCHAR PRIMARY KEY, -- Unique transaction identifier

user\_id VARCHAR NOT NULL, -- The user linked to the transaction

amount FLOAT NOT NULL, -- Transaction amount (positive for top-up, negative for deduction)

FOREIGN KEY (user\_id) REFERENCES users (user\_id) -- Foreign key linking to users

);

Database Setup Script Using SQLAlchemy

This section describes the setup of the database using SQLAlchemy in Python. SQLAlchemy automatically generates SQL commands based on the defined models.

1. Import Required Libraries

First, you import the necessary libraries for setting up the database and managing connections.

from sqlalchemy import create\_engine, Column, String, Float

from sqlalchemy.ext.declarative import declarative\_base

from sqlalchemy.orm import sessionmaker

import os

from dotenv import load\_dotenv

load\_dotenv()

Load database URL from environment variable

DATABASE\_URL = os.getenv('url')

Create an engine to connect to the database

engine = create\_engine(DATABASE\_URL)

SessionLocal: factory for database sessions

SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

Base class for models

Base = declarative\_base()

2. Define the User Model

The `User` class defines the structure of the `users` table in Python using SQLAlchemy ORM.

class User(Base):

\_\_tablename\_\_ = "users" Table name in the database

Columns for the table

user\_id = Column(String, primary\_key=True, index=True) Unique identifier for the user

amount = Column(Float, nullable=False) Balance for the user

3. Define the Transaction Model

The `Transaction` class defines the structure of the `transactions` table.

class Transaction(Base):

\_\_tablename\_\_ = "transactions" Table name in the database

Columns for the table

transaction\_id = Column(String, primary\_key=True, index=True) Unique transaction ID

user\_id = Column(String, nullable=False) Foreign key linking to the user's ID

amount = Column(Float, nullable=False) Amount (positive for top-up, negative for deduction)

4. Create the Database Tables

After defining the models, the tables can be created in the database using SQLAlchemy’s `Base.metadata.create\_all()` method. This will generate the SQL commands required to create the tables if they don’t already exist.

Create tables in the database if they don't exist

Base.metadata.create\_all(bind=engine)

This command will execute the necessary SQL to create the `users` and `transactions` tables based on the schema defined in the `User` and `Transaction` models.

Complete SQL Setup Script

If you prefer to directly use SQL scripts for setting up the database, here is a complete script that can be executed in any SQL-compatible database (such as PostgreSQL or MySQL):

SQL Setup Script

-- Create the users table

CREATE TABLE users (

user\_id VARCHAR(255) PRIMARY KEY, -- User's unique ID

amount FLOAT NOT NULL -- User's account balance

);

-- Create the transactions table

CREATE TABLE transactions (

transaction\_id VARCHAR(255) PRIMARY KEY, -- Unique ID for the transaction

user\_id VARCHAR(255) NOT NULL, -- ID of the user who performed the transaction

amount FLOAT NOT NULL, -- Amount of the transaction (positive for top-up, negative for deduction)

FOREIGN KEY (user\_id) REFERENCES users(user\_id) -- Foreign key linking to the user ID

);

-- Create indexes to optimize search performance (optional)

CREATE INDEX idx\_user\_id ON users(user\_id);

CREATE INDEX idx\_transaction\_id ON transactions(transaction\_id);

This script performs the following actions:

1. Creates the `users` table to store user account balances.

2. Creates the `transactions` table to store transaction logs.

3. Defines a foreign key on `transactions.user\_id` to ensure referential integrity, linking it to `users.user\_id`.

4. Creates indexes to optimize query performance when searching by `user\_id` and `transaction\_id`.

Database Interaction Example Using SQLAlchemy

Here’s an example of how your application interacts with the database using SQLAlchemy:

Top-up Example:

When a user performs a top-up:

1. The application checks if the user exists in the `users` table.

2. If the user exists, the balance is increased by the top-up amount.

3. A new transaction is recorded in the `transactions` table with a positive amount.

def topup\_user(user\_id: str, amount: float):

with get\_db\_session() as db:

user = db.query(User).filter(User.user\_id == user\_id).first()

if not user:

user = User(user\_id=user\_id, amount=amount)

db.add(user)

else:

user.amount += amount

transaction\_id = f"{user\_id}\_{datetime.now().strftime('%Y%m%d%H%M%S')}"

transaction = Transaction(transaction\_id=transaction\_id, user\_id=user\_id, amount=amount)

db.add(transaction)

db.commit()

Deduction Example:

Similarly, when a user deducts an amount:

1. The application checks if the user exists and has sufficient balance.

2. The balance is decreased by the specified amount.

3. A new transaction is recorded with a negative amount.

def deduct\_user(user\_id: str, amount: float):

with get\_db\_session() as db:

user = db.query(User).filter(User.user\_id == user\_id).first()

if not user:

raise HTTPException(status\_code=404, detail="User not found")

if user.amount < amount:

raise HTTPException(status\_code=400, detail="Insufficient balance")

user.amount -= amount

transaction\_id = f"{user\_id}\_{datetime.now().strftime('%Y%m%d%H%M%S')}"

transaction = Transaction(transaction\_id=transaction\_id, user\_id=user\_id, amount=-amount)

db.add(transaction)

db.commit()