COMSATS University Islamabad

Department of Computer Science

WEB TECHNOLOGIES

Assignment no. 4

Group members:

Ammar Rauf (FA23-BCS-059)

Kaab Bhinder (FA23-BCS-068)

Syeda Rubab (FA23-BCS-096)

Submitted to: Ma'am Sadia Maqbool

Class: BCS-5B **Due Date: 11-12-25**

Share It- Lender and Borrower System

Problem Statement

Many people purchase items they rarely use, such as drills, ladders, cameras, tools, books, or kitchen appliances. At the same time, someone nearby may need the same item temporarily. Currently, there is no organized, safe way for people in a community to share these items. This leads to unnecessary spending, wasted resources, and clutter in households.

Solution / What We Are Solving

ShareIt is a community-driven platform that allows people to borrow and lend items locally. Lenders can list items they are willing to share, while borrowers can search for items, request them, and pay a secure, refundable deposit. The system ensures fair usage with booking rules, pickup and return tracking, late return penalties, and dispute management. By enabling sharing within neighborhoods, ShareIt reduces unnecessary purchases, promotes sustainability, and helps people save money while making better use of resources.

1. Data Dictionary

Table-by-Table Description

1. users

The `users` table stores information about all users of the ShareIt platform, including borrowers, lenders, and admins.
It contains personal details like full name, email, phone, and address.
The `role` field distinguishes between Lender, Borrower, and Admin functionalities.
`password_hash` stores the encrypted password for secure authentication.
The `created_at` field tracks when the account was registered.
This table serves as a reference for items, bookings, wallets, and transactions.
It is the central entity connecting most other tables via foreign keys.

Field	Data Type	PK/FK	Description
user_id	SERIAL	PK	Unique identifier for each user
full_name	VARCHAR(100)	—	Full name of the user
email	VARCHAR(120)	—	Unique email for login
password_hash	TEXT	—	Encrypted password for authentication
phone	VARCHAR(20)	—	Contact number (optional)
address	VARCHAR(150)	—	User’s residential address
role	VARCHAR(20)	—	User role: Lender, Borrower, Admin
created_at	TIMESTAMP	—	Account creation timestamp (default now())

2. wallet

The `wallet` table manages the virtual wallet for each user.
It stores the balance available for deposits, refunds, or penalties.
Each wallet is linked to a user through `user_id` (1-to-1 relationship).
All monetary transactions are recorded against this wallet.
It helps in locking deposits when borrowers book items and returning them after completion.
The balance is updated automatically via transactions triggered by bookings or disputes.
This table ensures secure financial handling without storing cash details directly.

Field	Data Type	PK/FK	Description
wallet_id	SERIAL	PK	Unique wallet identifier
user_id	INT	FK → users(user_id)	User owning the wallet (1-to-1)
balance	NUMERIC(10,2)	—	Current wallet balance (default 0.00)

3. items

The `items` table holds details about all lendable items on the platform.
Each item is linked to a lender via `lender_id`.
It stores descriptive information like title, description, condition, and location.
Deposit calculation fields (`estimated_price`, `daily_deposit`) determine borrow costs.
It also contains availability rules (`min_days`, `max_days`) and active status.
Photos are stored in an array of URLs for multiple images.
This table is essential for managing inventory and item search functionalities.

Field	Data Type	PK/FK	Description
item_id	SERIAL	PK	Unique identifier for each item
lender_id	INT	FK → users(user_id)	Owner of the item
title	VARCHAR(120)	—	Item name or title
description	TEXT	—	Detailed description of the item
condition	VARCHAR(20)	—	Condition: New, Good, Used
estimated_price	NUMERIC(10,2)	—	Price estimate used for deposit calculation
min_days	INT	—	Minimum borrowing days
max_days	INT	—	Maximum borrowing days
daily_deposit	NUMERIC(10,2)	—	Lender-defined daily deposit
images	TEXT[]	—	URLs of item photos
location	VARCHAR(150)	—	City or area of the item
is_active	BOOLEAN	—	Whether the item is available for

			booking
created_at	TIMESTAMP	—	Timestamp when item was listed

4. bookings

The `bookings` table tracks each borrowing transaction between borrowers and lenders.
It records `item_id`, `borrower_id`, and `lender_id` to capture all parties.
Start and end dates define the booking duration.
`total_deposit` stores the deposit amount locked for that booking.
The `status` field tracks workflow stages: pending, accepted, rejected, or returned.
This table enables availability checks and enforces booking rules.
It is connected to transactions and disputes for financial and administrative purposes.

Field	Data Type	PK/FK	Description
booking_id	SERIAL	PK	Unique booking identifier
item_id	INT	FK → items(item_id)	Item being borrowed
borrower_id	INT	FK → users(user_id)	User borrowing the item
lender_id	INT	FK → users(user_id)	Owner of the item
start_date	DATE	—	Borrow start date
end_date	DATE	—	Borrow end date
total_deposit	NUMERIC(10,2)	—	Deposit locked for this booking
status	VARCHAR(20)	—	Booking status: Pending, Accepted, Rejected, Awaiting Pickup, Returned
created_at	TIMESTAMP	—	Booking creation timestamp

5. transactions

The `transactions` table logs all monetary activity related to users.
Each transaction references a user and optionally a booking.
It stores the `amount` and `tx_type` (Deposit, Refund, Penalty).
The `created_at` timestamp provides a historical record of all financial events.
It is linked to the wallet to update balances.
This table ensures transparency and traceability for deposits and penalties.
It allows admins and users to view transaction histories for accountability.

Field	Data Type	PK/FK	Description
tx_id	SERIAL	PK	Unique transaction identifier
user_id	INT	FK → users(user_id)	User involved in the transaction
booking_id	INT	FK → bookings(booking_id)	Optional booking reference

amount	NUMERIC(10, 2)	—	Transaction amount
tx_type	VARCHAR(20)	—	Type: Deposit, Refund, Penalty
created_at	TIMESTAMP	—	Transaction timestamp

6. disputes

The `disputes` table handles conflicts between lenders and borrowers.
It references the specific booking causing the dispute.
`description` explains the nature of the issue, and `estimated_cost` captures financial impact.
`status` indicates whether the dispute is Open, Resolved, or Rejected.
Admins review and manage disputes, updating the table accordingly.
This table protects both parties by recording disagreements formally.
It ensures that deposit deductions or penalties are applied fairly and systematically.

Field	Data Type	PK/FK	Description
dispute_id	SERIAL	PK	Unique dispute identifier
booking_id	INT	FK → bookings(booking_id)	Booking related to dispute
description	TEXT	—	Explanation of the dispute
estimated_cost	NUMERIC(10, 2)	—	Amount involved in dispute (if applicable)
status	VARCHAR(20)	—	Dispute status: Open, Resolved, Rejected

---

2. ERD Diagram

The ERD represents the ShareIt platform's database, designed to manage borrowing and lending of items in a local community. It visually illustrates tables (entities), fields (attributes), primary keys (PK), foreign keys (FK), and relationships between entities using Crow's Foot notation. The design is normalized and ready for a PostgreSQL + ORM setup (e.g., SQLAlchemy/SQLModel) for a REST API.

Entities and Attributes

1. **users**
   - Central entity representing all platform users: borrowers, lenders, and admins.
   - Fields: `user_id` (PK), `full_name`, `email`, `password_hash`, `phone`, `address`, `role`, `created_at`.

- Linked to `wallet` (1 → 1), `items` (1 → Many), and `bookings` (1 → Many as borrower/lender).

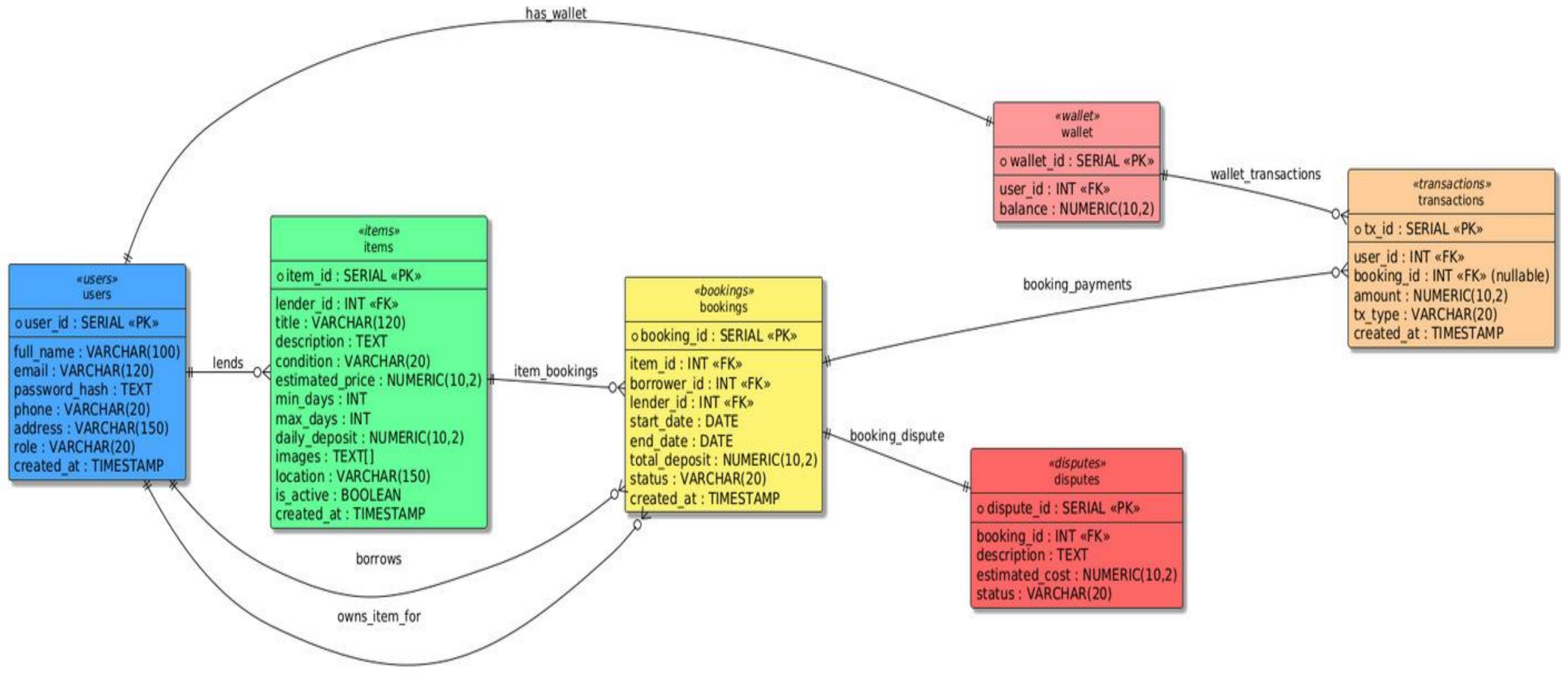
2. **wallet**
   - Represents a virtual wallet for financial transactions (deposits, refunds, penalties).
   - Fields: `wallet_id` (PK), `user_id` (FK → users.user_id), `balance`.
   - 1-to-1 relationship with `users`, 1-to-Many with `transactions`.
3. **items**
   - Contains details of all lendable items.
   - Fields: `item_id` (PK), `lender_id` (FK → users.user_id), `title`, `description`, `condition`, `estimated_price`, `min_days`, `max_days`, `daily_deposit`, `images` (array), `location`, `is_active`, `created_at`.
   - Each item can have multiple bookings (1 → Many).
4. **bookings**
   - Tracks borrowing transactions.
   - Fields: `booking_id` (PK), `item_id` (FK → items.item_id), `borrower_id` (FK → users.user_id), `lender_id` (FK → users.user_id), `start_date`, `end_date`, `total_deposit`, `status`, `created_at`.
   - Relationships:
     - Many-to-1 with `items`
     - Many-to-1 with `users` (borrower and lender)
     - 1-to-Many with `transactions`
     - 1-to-1 with `disputes`
5. **transactions**
   - Logs financial activity per user and booking.
   - Fields: `tx_id` (PK), `user_id` (FK → users.user_id), `booking_id` (FK → bookings.booking_id, nullable), `amount`, `tx_type`, `created_at`.
   - Supports multiple transactions per booking and per user.
6. **disputes**
   - Handles conflicts between lenders and borrowers.
   - Fields: `dispute_id` (PK), `booking_id` (FK → bookings.booking_id), `description`, `estimated_cost`, `status`.
   - 1-to-1 relationship with bookings ensures each booking can have at most one dispute.

Relationships and Cardinality (Crow’s Foot)

Relationship	Type	Description
users → wallet	1 → 1	Each user has exactly one wallet.
users → items	1 → Many	A user can lend multiple items.
users → bookings (borrower)	1 → Many	A user can borrow multiple items.
users → bookings (lender)	1 → Many	A user can lend multiple items that are booked.
items → bookings	1 → Many	An item can have multiple bookings over time.
bookings → transactions	1 → Many	Each booking can generate multiple financial transactions.
bookings → disputes	1 → 1	Each booking can have at most one dispute.

Design Highlights

1. **Normalization:** No redundant data; each entity stores only its own attributes.
2. **Financial tracking:** Wallet and transactions separate to maintain accurate logs.
3. **Booking integrity:** Linking both borrower and lender ensures correct reporting.
4. **Crow’s Foot notation:** Clearly shows multiplicity (1 → 1, 1 → Many) for each relationship.
5. **Scalable:** Can easily add features like ratings, categories, or multi-image support.


has_wallet
«wallet»
wallet
o wallet_id : SERIAL «PK»
user_id : INT «FK»
balance : NUMERIC(10,2)
wallet_transactions
«transactions»
transactions
o tx_id : SERIAL «PK»
user_id : INT «FK»
booking_id : INT «FK» (nullable)
amount : NUMERIC(10,2)
tx_type : VARCHAR(20)
created_at : TIMESTAMP
«users»
users
o user_id : SERIAL «PK»
full_name : VARCHAR(100)
email : VARCHAR(120)
password_hash : TEXT
phone : VARCHAR(20)
address : VARCHAR(150)
role : VARCHAR(20)
created_at : TIMESTAMP
lends
«items»
items
o item_id : SERIAL «PK»
lender_id : INT «FK»
title : VARCHAR(120)
description : TEXT
condition : VARCHAR(20)
estimated_price : NUMERIC(10,2)
min_days : INT
max_days : INT
daily_deposit : NUMERIC(10,2)
images : TEXT[]
location : VARCHAR(150)
is_active : BOOLEAN
created_at : TIMESTAMP
item_bookings
«bookings»
bookings
o booking_id : SERIAL «PK»
item_id : INT «FK»
borrower_id : INT «FK»
lender_id : INT «FK»
start_date : DATE
end_date : DATE
total_deposit : NUMERIC(10,2)
status : VARCHAR(20)
created_at : TIMESTAMP
booking_payments
booking_dispute
«disputes»
disputes
o dispute_id : SERIAL «PK»
booking_id : INT «FK»
description : TEXT
estimated_cost : NUMERIC(10,2)
status : VARCHAR(20)
borrows
owns_item_for

3. Additional Notes

3.1 Required Fields (Why They Are Required)

- `email, full_name, password_hash` in `users`
  Ensures unique identification, secure login, and proper user verification.
- `lender_id` **and** `borrower_id` in `bookings`
  Mandatory to maintain transaction integrity and to clearly identify both parties involved in a borrowing activity.
- `daily_deposit, min_days, max_days` in `items`
  Required to calculate deposit, enforce booking rules, and prevent misuse.
- `amount` **and** `tx_type` in `transactions`
  Necessary to maintain a transparent financial log and support wallet balance calculations.
- `status` **fields** (in `bookings` and `disputes`)
  Required to track progress and enforce platform rules (e.g., pending, approved, returned, resolved).

3.2 Relationship Design Justification

- **1-to-1 (users → wallet)**
  Each user has only one wallet to avoid double balances, multiple accounts misuse, or fraudulent transfers.
- **1-to-many (users → items)**
  A user can list multiple items for lending; one lender can own several products.
- **1-to-many (items → bookings)**
  An item can be booked many times over different dates, but each booking is uniquely linked to one item.
- **1-to-many (bookings → transactions)**
  A single booking may result in multiple financial events: deposit payment, late fee, refund, or penalty.
- **1-to-1 (bookings → disputes)**
  A booking can only have one open dispute at a time to avoid duplicate claims and conflict in cost estimation.

3.3 Indexing & Optimization Choices

- `email` **(users)** is indexed to allow fast login and prevent duplicates.
- **Foreign keys (**`item_id` **and** `user_id`**)** are indexed to speed up joins in `bookings`, `transactions`, and `disputes` queries.

- `status` **fields** are indexed because filtering (e.g., finding all “pending” bookings) is very common, improving dashboard/API performance.
- `created_at` **fields** help support sorting and pagination for recent activity, enhancing user experience and admin analytics.
- **Partial indexing** may be used on `is_active = true` items to speed up searches for currently available items.

3.4 Assumptions Made During Design

- Every user must be authenticated; no guest access is allowed.
- A deposit is always paid before booking approval.
- A dispute can only arise *after* an item return request is submitted.
- Only lenders can create disputes; admins resolve them.
- Images stored as an array are assumed to be URL paths from external storage (e.g., S3 or local media folder).