

CMPS 350 Web Development Fundamentals – Spring 2025

Lab 11 – Data Management using Prisma and SQLite Database

Objective

You will practice:

- Modelling Data using Prisma Schema Language
- Query (read/write) the database using Prisma Client

This Lab is based on Lab 10 Banking App. You are required to change the repositories to use a database instead of JSON files.

Project Setup

Download **Lab11-Data Management** from the GitHub Repo and copy it to your repository. Open the **BankingApp** in VS Code and complete the tasks below.

1. Install the Prisma packages using:

```
npm install prisma --save-dev
npm install @prisma/client
```

Also install **Prisma VS Code Extension**

2. Set up Prisma with this command:

```
npx prisma init --datasource-provider sqlite
```

This command creates a new **prisma** directory with **schema.prisma** file and configures SQLite as your database

Creating the Data Model

1. Define a Data Model in **schema.prisma** to model the following Banking App entities:

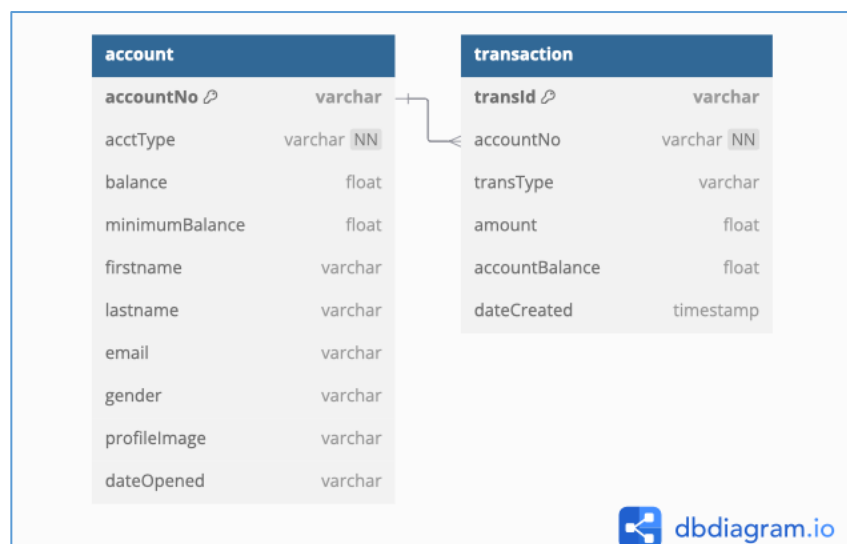
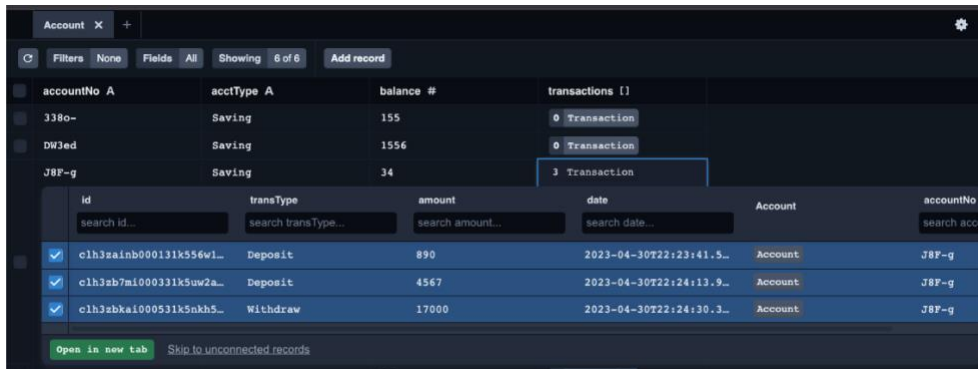


Figure 1 Banking Entities Diagram

- **transId** is the primary key for Trans and should be auto-assigned.
 - **acctNo** is the primary key for Account and it should be auto-assigned using [cuid\(\)](#)
 - You should define a one-to-many relationship between the two models as shown in figure 1.
2. Export the models to your database by using the following Prisma command
`npx prisma migrate dev --name init`
 Anytime you make changes to the models, you need to `npx prisma migrate dev`
 3. To view your database, run the following command `npx prisma studio`



Querying the database with Prisma client

1. Change **accounts-repo.js** repository functions to use **Prisma client**. **As you make progress** test each function using a console app, Postman or Mocha:
 - a. **addAccount(account)**: adds a new account
 - b. **initDB()**: read data/accounts.json and insert them in the database using createMany Prisma client function
 - c. **getAccounts(type)**: returns a list of accounts filtered by account type if specified
 - d. **getAccount(acctNo)**: gets an account by account number
 - e. **updateAccount(account)**: updates an existing account
 - f. **deleteAccount(acctNo)**: deletes an account by account number
 - g. **addTransaction(transaction, acctNo)**: add either deposit or withdrawal transaction. It calls updateAccount method internally to update the balance.
 - h. **getTrans(acctNo, fromDate, toDate)**: get transactions for a particular account for date range
 - i. **getAvgBalance()**: returns average account balance by account type
 - j. **getTransSum(fromDate, toDate)**: returns the sum of debit and sum of credit transactions completed during a date range
2. Test the app using the user interface you implemented in Lab 10.