# CMPS 350 Web Development Fundamentals
# 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.

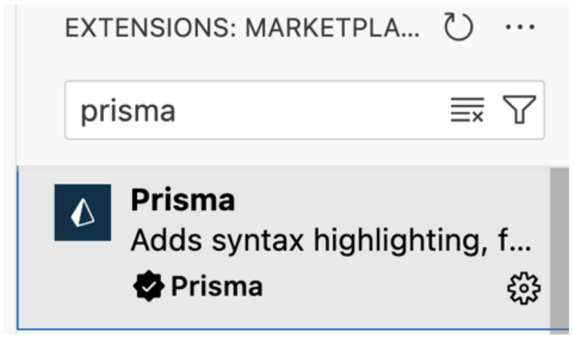
1. Install the Prisma packages using:

```
npm install

npm install prisma --save-dev

npm install @prisma/client
```

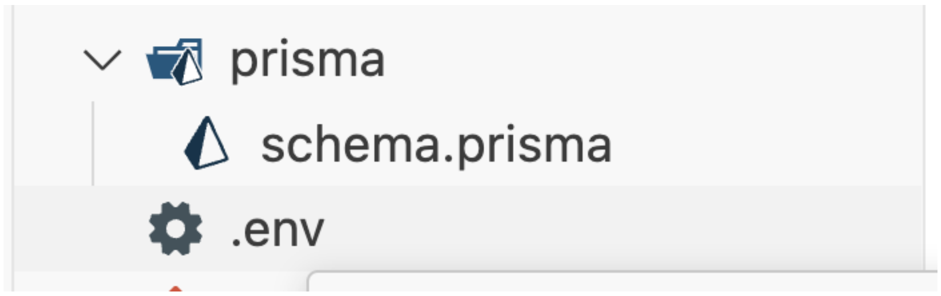
2. Install **Prisma VS Code Extension**



3. Set up Prisma with this command:

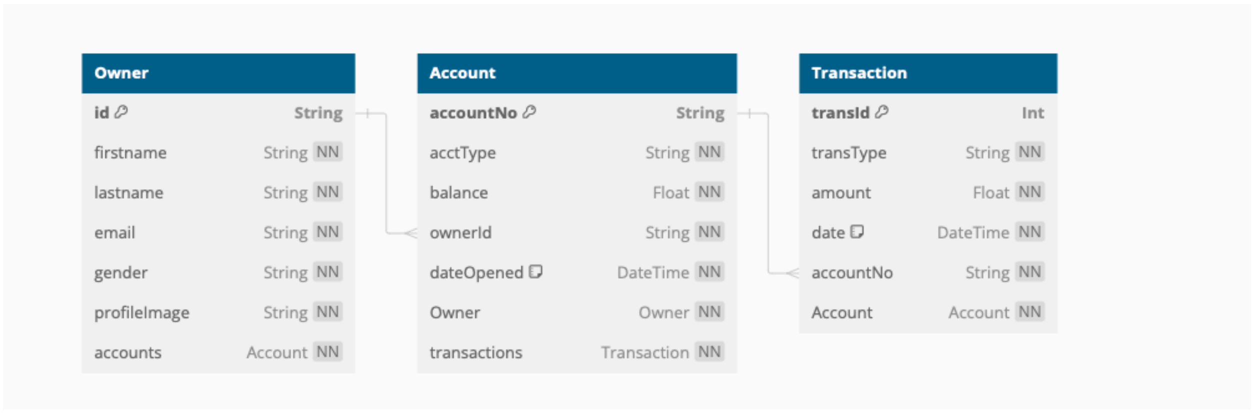
```
npx prisma init --datasource-provider sqlite
```

This command creates a new `prisma` directory with `schema.prisma` file and configures SQLite as your database [DATABASE_URL="file:./dev.db"]



## Creating the Data Model

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



| Owner | |
|---|---|
| id | String |
| firstname | String NN |
| lastname | String NN |
| email | String NN |
| gender | String NN |
| profileImage | String NN |
| accounts | Account NN |

| Account | |
|---|---|
| accountNo | String |
| acctType | String NN |
| balance | Float NN |
| ownerId | String NN |
| dateOpened | DateTime NN |
| Owner | Owner NN |
| transactions | Transaction NN |

| Transaction | |
|---|---|
| transId | Int |
| transType | String NN |
| amount | Float NN |
| date | DateTime NN |
| accountNo | String NN |
| Account | Account NN |

*Figure 1 Banking Entities Diagram*

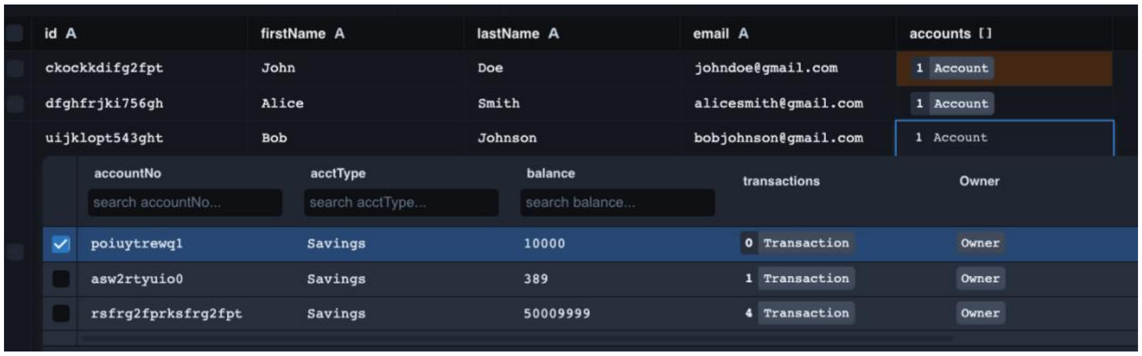
- **id** is the primary key for owner and should be auto-assigned using cuid() as you did for the account model.
- **transId** is the primary key for Trans and should be auto-assigned.
- **accountNo** is the primary key for Account and it should be auto-assigned using cuid()
- You should define a one-to-many 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`



| id A | firstName A | lastName A | email A | accounts [] |
|---|---|---|---|---|
| ckockkdifg2fpt | John | Doe | johndoe@gmail.com | 1 Account |
| dfghfrjki756gh | Alice | Smith | alicesmith@gmail.com | 1 Account |
| uijklopt543ght | Bob | Johnson | bobjohnson@gmail.com | 1 Account |

| accountNo | acctType | balance | transactions | Owner |
|---|---|---|---|---|
| search accountNo... | search acctType... | search balance... | | |
| poiuytrewq1 | Savings | 10000 | 0 Transaction | Owner |
| asw2rtyuio0 | Savings | 389 | 1 Transaction | Owner |
| rsfrg2fprksfrg2fpt | Savings | 50009999 | 4 Transaction | Owner |

4. Lets add some initial data to the database using our json files found inside the data folder **node prisma/seed.js**
5. Add this code to the schema if you want to generate the UML diagram of the database

```
generator dbml {
  provider = "prisma-dbml-generator"
}
```

6. Run **npx prisma generate**
7. Use this link to visualize https://dbdiagram.io/d

## 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:

```
import { PrismaClient } from "@prisma/client";

const prisma = new PrismaClient();
```

   a. **addAccount(account):** adds a new account
   b. **getAccounts(type):** returns a list of accounts filtered by account type if specified
   c. **getAccount(accountNo):** gets an account by account number
   d. **updateAccount(account):** updates an existing account
   e. **deleteAccount(accountNo):** deletes an account by account number
   f. **addTransaction**(transaction, **accountNo**): add either deposit or withdrawal transaction. It calls updateAccount method internally to update the balance.
   g. **getTrans(acctNo, fromDate, toDate):** get transactions for a particular account for date range
   h. **getAvgBalance():** returns average account balance by account type
   i. **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.