Database “transaction.db” tables

users table sample

```
sqlite> SELECT * FROM users;
```

user_id	username	password	created_at	updated_at
1	azcarraga	azcarraga123	2023-05-27 14:23:45	2025-06-02 12:10:30
2	mirasol	mirasol123	2023-04-27 09:00:45	2025-06-02 12:10:30
3	nicolas	nicolas123	2023-11-01 12:00:00	2025-06-02 12:10:30
4	jone doe	jonedoe123	2025-06-02 20:13:12	2025-06-02 20:13:12
6	eger	eger	2025-06-02 23:45:52	2025-06-02 23:45:52
7	jonh toopick	jt123	2025-06-03 01:20:52	2025-06-03 01:20:52
8	dododo	dadada	2025-06-03 09:07:22	2025-06-03 09:07:22

transactions table sample

```
sqlite> SELECT * FROM transactions LIMIT 40;
```

transaction_id	transaction_date	transaction_type	transaction_category	transaction_amount	transaction_description	user_id	created_at	updated_at
1	2025-01-01	savings	Monthly Allowance	25.0	test from edit #25	1	2025-06-01 12:57:57	2025-06-01 12:57:57
2	2025-01-01	expense	Bills	99.0		1	2025-06-01 12:57:57	2025-06-10 23:12:38
3	2025-01-01	expense	Bills	2.0	test #2	1	2025-06-01 12:57:57	2025-06-01 12:57:57
4	2025-01-01	expense	Bills	3.0	test #3	1	2025-06-01 12:57:57	2025-06-01 12:57:57
5	2025-01-01	expense	Bills	4.0	test #4	1	2025-06-01 12:57:57	2025-06-01 12:57:57
6	2025-01-01	expense	Bills	5.0	test #5	1	2025-06-01 12:57:57	2025-06-01 12:57:57
7	2025-01-01	expense	Bills	6.0	test #6	1	2025-06-01 12:57:57	2025-06-01 12:57:57
8	2025-01-01	savings	Monthly Allowance	26.0	test from edit #26	1	2025-06-01 12:57:57	2025-06-01 12:57:57
9	2025-01-01	savings	Monthly Allowance	27.0	test from edit #27	1	2025-06-01 12:57:57	2025-06-01 12:57:57
10	2025-01-01	investment	Stocks	20.0	test from edit #20	1	2025-06-01 12:57:57	2025-06-01 12:57:57
11	2025-02-13	expense	Bills	8.0	test #8	1	2025-06-01 12:57:57	2025-06-01 12:57:57
12	2025-01-14	expense	Bills	9.0	test #9	1	2025-06-01 12:57:57	2025-06-01 12:57:57
13	2025-01-01	expense	Bills	10.0	test #10	1	2025-06-01 12:57:57	2025-06-01 12:57:57
14	2025-01-01	savings	Monthly Allowance	28.0	test from edit #28	1	2025-06-01 12:57:57	2025-06-01 12:57:57
15	2025-01-01	expense	Bills	11.0	test #11	1	2025-06-01 12:57:57	2025-06-01 12:57:57
16	2025-01-01	expense	Bills	11.0	test #11	1	2025-06-01 12:57:57	2025-06-01 12:57:57
17	2025-01-01	savings	Monthly Allowance	29.0	test from edit #29	1	2025-06-01 12:57:57	2025-06-01 12:57:57
18	2025-01-01	savings	Monthly Allowance	30.0	test #30	1	2025-06-01 12:57:57	2025-06-01 12:57:57
19	2025-01-01	expense	Bills	12.0	test #12	1	2025-06-01 12:57:57	2025-06-01 12:57:57
20	2025-01-01	expense	Bills	13.0	test #13	1	2025-06-01 12:57:57	2025-06-01 12:57:57
21	2025-03-06	expense	Wellness	14.0	test #14	1	2025-06-01 12:57:57	2025-06-01 12:57:57
22	2025-01-01	expense	Bills	15.0	test #15	1	2025-06-01 12:57:57	2025-06-01 16:41:02
23	2024-12-30	investment	Crypto	6100.0	Cross-asset crypto token	3	2025-06-01 12:57:57	2025-06-01 12:57:57
24	2025-01-01	expense	Bills	1119.0	kkkk	2	2025-06-01 12:57:57	2025-06-11 23:50:33
25	2024-12-29	savings	Monthly Allowance	1600.0	Year-end allowance	3	2025-06-01 12:57:57	2025-06-01 12:57:57
26	2024-12-29	investment	Crypto	6200.0	Updated L2 coin	2	2025-06-01 12:57:57	2025-06-01 12:57:57
27	2024-12-28	expense	Wellness	2000.0	Spa	1	2025-06-01 12:57:57	2025-06-01 12:57:57
28	2025-01-01	investment	Stocks	21.0	test from edit #21	1	2025-06-01 12:57:57	2025-06-01 12:57:57
29	2024-12-27	expense	Shopping	2750.0	Clothes	3	2025-06-01 12:57:57	2025-06-01 12:57:57
30	2024-12-27	savings	Monthly Allowance	1800.0	Last for year	2	2025-06-01 12:57:57	2025-06-01 12:57:57
31	2024-12-27	investment	Stocks	8800.0	Food innovation stock	3	2025-06-01 12:57:57	2025-06-01 12:57:57
32	2024-12-26	expense	Education	3200.0	Books	2	2025-06-01 12:57:57	2025-06-01 12:57:57
33	2024-12-26	savings	Change	75.0	Avoided spending	3	2025-06-01 12:57:57	2025-06-01 12:57:57
34	2024-12-26	investment	Stocks	9000.0	Streaming platform equity	2	2025-06-01 12:57:57	2025-06-01 12:57:57
35	2024-12-25	expense	Education	3700.0	Course	1	2025-06-01 12:57:57	2025-06-01 12:57:57
36	2025-01-01	savings	Monthly Allowance	31.0	test #31	1	2025-06-01 12:57:57	2025-06-01 12:57:57
37	2025-01-01	investment	Stocks	22.0	test from edit #22	1	2025-06-01 12:57:57	2025-06-01 12:57:57
38	2024-12-24	expense	Healthcare	950.0	Medicine	3	2025-06-01 12:57:57	2025-06-01 12:57:57
39	2024-12-24	savings	Change	65.0	Didn't ride jeep	2	2025-06-01 12:57:57	2025-06-01 12:57:57
40	2024-12-24	investment	Real Estate	10900.0	Suburban apartment buy	3	2025-06-01 12:57:57	2025-06-01 12:57:57

Backend

---

1. dataclass **Account**
    - Just for storing data
    - Holds data about the user account
    - Has attributes in the constructor
    - Don't have methods

    attributes
    - *username*: **str**
    - *password*: **str**

---

2. class **UserRepository**
    - The class that directly interacts with the database
    - Can read, write, update, and delete to the **transactions table**
    - Uses the **Account** class to store data

    attributes:
    - **connection** (an **sqlite3.Connection** object)
    - **cursor** (an **sqlite3.Cursor** object)

    methods:

    \+ **initializeDatabase(): # for UserRepository**
        - Invoked in the constructor
        - Creates a database if it does not already exist

    \+ **addAccount(): # for login form**
        - Takes in one parameter:
            - **account** (an **Account** object)
        - Add **account** data to the **{users}** table of the database when:
            - **account.username** doesn't exist in the database
            - **account.username** and **account.password** are not empty
        - Return status if **was_added** (a **bool**) or not

    \+ **getAccountID(): # for login form**
        - Takes in one parameter:
            - **account** (an **Account** object)
        - Searches for a row in the **{users}** table of the database where:
            - **account.username** matches **username** column
            - **account.password** matches **password** column
        - Stores the **user_id** column of the row to **user_id**
        - Returns **user_id** (an **int**)

---

3. dataclass **Transaction**
    - Just for storing data

- Holds data about a certain transaction
- Has attributes in the constructor
- Don't have methods

attributes:
- *t_id*: **int**
- *t_date*: **str**
- *t_type*: **str**
- *t_category*: **str**
- *t_amount*: **float**
- *t_description*: **str**

---

4. class TransactionRepository

- The class that directly interacts with the database
- Can read, write, update, and delete to the **transactions table**
- Uses the **Transaction** class to store data

attributes:
- **connection** (an **sqlite3.Connection** object)
- **cursor** (an **sqlite3.Cursor** object)

methods:

+ **getAllTransactions(): # for profile page # history page**
  - Takes in one parameter:
    - **current_user_id** (an **int**)
  - Searches for rows in the **{transactions}** table of the database where:
    - **current_user_id** matches **user_id** column
  - Creates a **Transaction** object for each matching row
  - Appends all **Transaction** objects to a List
  - Returns the list (a **List[Transaction])**

+ **getTransactionsByType(): # for profile page # for edit page # for history page**
  - Takes in two parameters:
    - **current_user_id** (an **int**)
    - **t_type** (a **str**)
  - Searches for rows in the **{transactions}** table of the database where:
    - **current_user_id** matches **user_id** column
    - **t_type** matches **transactoin_type** column
  - Creates a **Transaction** object for each matching row
  - Appends all **Transaction** objects to a List
  - Returns the list (a **List[Transaction])**

+ **getTransactionsByCategory(): # for history page**
  - Takes in two parameters:
    - **current_user_id** (an **int**)
    - **t_category** (a **str**)
  - Searches for rows in the **{transactions}** table of the database where:

- ❖ **current_user_id** matches **user_id** column
- ❖ **t_category** matches **transaction_category** column

- Creates a **Transaction** object for each matching row
- Appends all **Transaction** objects to a List
- Returns the list (a **List[Transaction])**

+ **getRecentTransactions(): # for home page**
  - Takes in two parameters:
    - ❖ **current_user_id** (an **int**)
    - ❖ **t_count** (an **int**)
  - Searches for rows in the **{transactions}** table of the database where:
    - ❖ **current_user_id** matches **user_id** column
  - Retrieves a specific number of recently inserted rows indicated by **t_count**
  - Creates a **Transaction** object for each row
  - Appends all **Transaction** objects to a List
  - Returns the list (a **List[Transaction])**

+ **addTransaction(): # for add page**
  - ❖ Takes in two parameters:
    - ❖ **current_user_id** (an **int**)
    - ❖ **transaction** (a **Transaction** object)
  - ❖ Converts the **transaction** object into a tuple
  - ❖ Inserts the tuple into the **{transactions}** table of the database:
  - ❖ Sets the **user_id** column to **current_user_id**

+ **modifyTransaction(): # for edit page**
  - Takes in three parameters:
    - ❖ **current_user_id** (an **int**)
    - ❖ **t_id** (an **int**)
    - ❖ **transaction** (a **Transaction** object)
  - Converts the **transaction** object into a tuple
  - Updates the row in the **{transactions}** table of the database using the tuple where:
    - ❖ **current_user_id** matches **user_id** column
    - ❖ **t_id** matches **transaction_id** column

+ **deleteTransaction():**
  - Takes in two parameters:
    - ❖ **current_user_id** (an **int**)
    - ❖ **t_id** (an **int**)
  - Deletes the row in the **{transactions}** table of the database where:
    - ❖ **current_user_id** matches **user_id** column
    - ❖ **t_id** matches **transaction_id** column

---

5. dataclass **Finance**

- Just for storing data
- Holds data about a user's finances
- Has attributes in the constructor
- Don't have methods

attributes:

- *total_income*: **float**
- *total_expenses*: **float**
- *total_savings:* **float**
- *total_investment*: **float**

---

6. class **TransactionManager**

- Handles the app's logic
- Initializes **TransactionRepository**
- Calculates data
- Creates graph
- Uses the **Transaction** and **Finance** classes to store data

methods:

+ **calculateOverallFinance(): # for profile page**
  - Takes in one parameter:
    - **user_id** (an **int**)
  - Uses the **getAllTransactions(user_id)** method of the **TransactionRepository** object to get all of the user's transactions
  - Calculate the total amount of each transaction type separately
  - Creates a **Finance** object containing the totals of each type
  - Returns the **Finance** object

+ **calculateOverallBalance(): # for profile page # for home page**
  - Takes in one parameter:
    - **user_id** (an **int**)
  - Uses **calculateOverallFinance(user_id)** to get **overall_fiance** (a **Finance** object)
  - Uses **total_expenses** and **total_income** of the **overall_finance** object
  - Subtracts **total_expenses** from **total_income**
  - Returns the result (a **float**)

+ **calculateMonthlyFinances(): # for home page**
  - Takes in one parameter:
    - **user_id** (an **int**)
  - Uses the **getTransactionsByType(user_id)** method of the **TransactionRepository** object to get all of the user's transactions for each type:
    - "expense", "savings", "income", "investment"
  - Stores each result (a **List[Transactions]**) separately (one for each type)
  - Sort and group the transactions from the list by **year_month** (one dictionary for each type):
    - Each dictionary maps **year_month** (a **str**) to **List[Finance]**

- ❖ For **year_month** gap that doesn't have transactions for a given type, map the **year_month** to a list containing a dummy **Transaction** object where all attributes are set to 0.0 (a **float**)
- ❖ Ex:

```
expenses = {
        "2024-01":[transaction], # transaction obj containing all 0s
        "2024-02":[transaction1, transaction2, transaction3…],
        "2024-04":[transaction], # transaction obj containing all 0s,
        "2024-05":[transaction1, transaction2, transaction3…],
        …
        "2025-01":[transaction1, transaction2, transaction3…],
        …
}
savings = {…}
income = {...}
investment = {...}
```

- For each year-month:
  - ❖ Calculates the total amount of each transaction type
  - ❖ Creates a **Finance** object containing the totals of each transaction type
- Appends each **Finance** object to a dict:
  - ❖ The dictionary maps **year_month** (a **str**) to the **Finance** object containing the totals for a specific month
  - ❖ Ex:

```
monthly_finances = {
        "2024-01":total_finance
        "2024-02":total_finance,
        "2024-03":total_finance,
        "2024-04":total_finance,
        …
        "2025-01":total_finance,
        …
}
```

- Returns the dictionary (a **Dict[str, Finance]**)

+ **calculateQuarterlyFinances(): # for home page**
  - Takes in one parameter:
    - ❖ **user_id** (an **int**)
  - Uses the **getTransactionsByType(user_id)** method of the **TransactionRepository** object to get all of the user's transactions for each type:
    - ❖ "expense", "savings", "income", "investment"
  - Stores each result (a **List[Transactions]**) separately (one for each type)
  - Sort and group the transactions from the list by **year_quarter** (one dictionary for each type):
    - ❖ Each dictionary maps **year_quarter** (a **str**) to **List[Finance]**

- ❖ For **year_quarter** gap that doesn't have transactions for a given type, map the **year_quarter** to a list containing a dummy **Transaction** object where all attributes are set to 0.0 (a **float**)
- ❖ Ex:

  expenses = {

  "2024-Q1":[transaction], # transaction obj containing all 0s

  "2024-Q2":[transaction1, transaction2, transaction3…],

  "2024-Q4":[transaction], # transaction obj containing all 0s,

  "2025-Q1":[transaction1, transaction2, transaction3…],

  …

  "2025-Q4":[transaction1, transaction2, transaction3…],

  …

  }

  savings = {…}

  income = {...}

  investment = {...}

- For each year-quarter:
  - ❖ Calculates the total amount of each transaction type
  - ❖ Creates a **Finance** object containing the totals of each transaction type
- Appends each **Finance** object to a dict:
  - ❖ The dictionary maps **year_quarter** (a **str**) to the **Finance** object containing the totals for a specific quarter
  - ❖ Ex:

    quarterly_finances = {

    "2024-Q1":total_finance

    "2024-Q2":total_finance,

    "2024-Q3":total_finance,

    "2024-Q4":total_finance,

    …

    "2025-Q4":total_finance,

    …

    }

- Returns the dictionary (a **Dict[str, Finance]**)

+ **createMonthlyGraph(): # for home page**
  - Takes in 3 parameters:
    - ❖ **user_id** (an **int**)
    - ❖ **width_in** (a **float**)
    - ❖ **height_in** (a **float**)
    - ❖ **dpi** (a **float**)
    - ❖ **title_size** (an **int**)
    - ❖ **label_size** (an **int**)

- Uses **calculateMonthlyFinances()** to get **monthly_finances** (a **Dict[str, Finance]**)
- Plots the monthly income and expenses into separate line graphs using the **{matplotlib}** library
    - ❖ With dimensions **width_in** and **height_in**, and **dpi**
    - ❖ With y-axis (**amount**) and x-axis (**month**)
    - ❖ Uses **title_size** for the graph title and **label_size** for the graph labels
    - ❖ Uses **width_in**, **height_in**, and **dpi** for the graph dimensions
- Returns the results (a **Tuple[matplotlib.figure.Figure, matplotlib.figure.Figure]**)

+ **createQuarterlyGraph(): # for home page**
    - Takes in 3 parameters:
        - ❖ **user_id** (an **int**)
        - ❖ **width_in** (a **float**)
        - ❖ **height_in** (a **float**)
        - ❖ **dpi** (a **float**)
        - ❖ **title_size** (an **int**)
        - ❖ **label_size** (an **int**)
    - Uses **calculateQuarterlyFinances()** to get **quarterly_finances** (a **Dict[str, Finance]**)
    - Plots the quarterly data into a grouped line graph using the **{matplotlib}** library
        - ❖ With dimensions **width_in** and **height_in**, and **dpi**
        - ❖ With y-axis (**amount**) and x-axis (**quarters**)
        - ❖ Uses **title_size** for the graph title and **label_size** for the graph labels
        - ❖ Uses **width_in**, **height_in**, and **dpi** for the graph dimensions
    - Returns the result (a **matplotlib.figure.Figure**)