

# Database “[transaction.db](#)” tables

users table sample

user_id	username	password
1	'azcarraga'	'azcarraga123'
2	'mirasol'	'mirasol123'
3	'nicolas'	'nicolas123'

transactions table sample

transaction_id	transaction_date	transaction_type	transaction_category	transaction_amount	transaction_descriptions	user_id
1	'2024-01-05'	'savings'	'Monthly Allowance'	5000.00	'Salary savings'	1
2	'2024-01-05'	'expense'	'Bills'	2500.00	'Electricity bill'	1
3	'2024-01-02'	'income'	'Salary'	25000.00	'Monthly paycheck'	1
4	'2024-01-02'	'investment'	'Stocks'	5000.00	'Initial stock investment'	1
5	'2024-03-10'	'savings'	'Monthly Allowance'	5000.00	'Salary savings'	2
6	'2024-03-07'	'expense'	'Bills'	2500.00	'Electricity bill'	2
7	'2024-03-04'	'income'	'Salary'	25000.00	'Monthly paycheck'	2
8	'2024-03-01'	'investment'	'Stocks'	5000.00	'Initial stock investment'	2
9	'2024-05-10'	'savings'	'Monthly Allowance'	5000.00	'Salary savings'	3
10	'2024-05-01'	'expense'	'Bills'	2500.00	'Electricity bill'	3
11	'2024-04-01'	'income'	'Salary'	25000.00	'Monthly paycheck'	3
12	'2024-03-10'	'investment'	'Stocks'	5000.00	'Initial stock investment'	3

# Backend (Requirements)

## 1. class **UserRepository**

---

## 2. class **UserManager**

---

## 3. dataclass **Transaction**

- Just for storing data
- Holds data about a certain transaction
- Has attributes in the decorator
- 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 database
- Uses the **Transaction** class to store data

### methods:

#### + **getAllTransactions():**

- 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():**

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

#### + **getTransactionsByCategory():**

- 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]**)

#### + **addTransaction():**

- ❖ 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():**

- 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 decorator
- 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():**

- Uses the **getAllTransactions()** 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 using the totals of each type
- Returns the **Finance** object

+ **calculateOverallBalance():**

- Takes in one parameter:
  - ❖ *overall\_finance* (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():**

- Uses the **getTransactionsByType()** 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]**) in a separate list (one for each type)
- Sort and group the transactions from the list by **year\_month** (one dictionary per type):
  - ❖ Each dictionary maps **year\_month** (a **str**) to **List[Finance]**
  - ❖ For **year\_month** 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...],
```

```

        "2024-03": [transaction1, transaction2, transaction3...],
        "2024-04": [transaction], # transaction obj containing all 0s,
        "2024-05": [transaction1, transaction2, transaction3...],
        ...
    }

```

```

    savings = {...}
    income = {...}
    investment = {...}

```

- For each year-month:
  - ❖ Calculates the total amount for each transaction type
  - ❖ Creates a **Finance** object using the totals of each type
- Appends each **Finance** object to a dict:
  - ❖ Each dictionary maps **year\_month** (a **str**) to a **Finance** object
- Returns the dictionary (a **Dict[str, Finance]**)

#### + **calculateQuarterlyFinances():**

- Uses the **getTransactionsByType()** 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]**) in a separate list (one for each type)
- Sort and group the transactions from the list by **year\_quarter** (one dictionary per type):
  - ❖ Each dictionary maps **year\_quarter** (a **str**) to **List[Finance]**
  - ❖ For **year\_quarter** 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-Q1": [transaction], # transaction obj containing all 0s
        "2024-Q2": [transaction1, transaction2, transaction3...],
        "2024-Q3": [transaction1, transaction2, transaction3...],
        "2024-Q4": [transaction], # transaction obj containing all 0s,
        "2025-Q1": [transaction1, transaction2, transaction3...],
        ...
    }
    savings = {...}
    income = {...}
    investment = {...}

```

- For each year-month:
  - ❖ Calculates the total amount for each transaction type
  - ❖ Creates a **Finance** object using the totals of each type
- Appends each **Finance** object to a dict:
  - ❖ Each dictionary maps **year\_quarter** (a **str**) to a **Finance** object
- Returns the dictionary (a **Dict[str, Finance]**)

#### + **createMonthlyGraph():**

- Takes in one parameter:

- ❖ **monthly\_finances** (a **Dict[str, Finance]**)

- Plots the monthly data into a grouped bar graph using the **<matplotlib>** library
  - ❖ With title, labels, and legend
- Returns the result (a **matplotlib.Figure**)

+ **createQuarterlyGraph():**

- Takes in one parameter:
  - ❖ **quarterly\_finances** (a **Dict[str, Finance]**)
- Plots the quarterly data into a grouped bar graph using the **<matplotlib>** library
  - ❖ With title, labels, and legend
- Returns the result (a **matplotlib.Figure**)