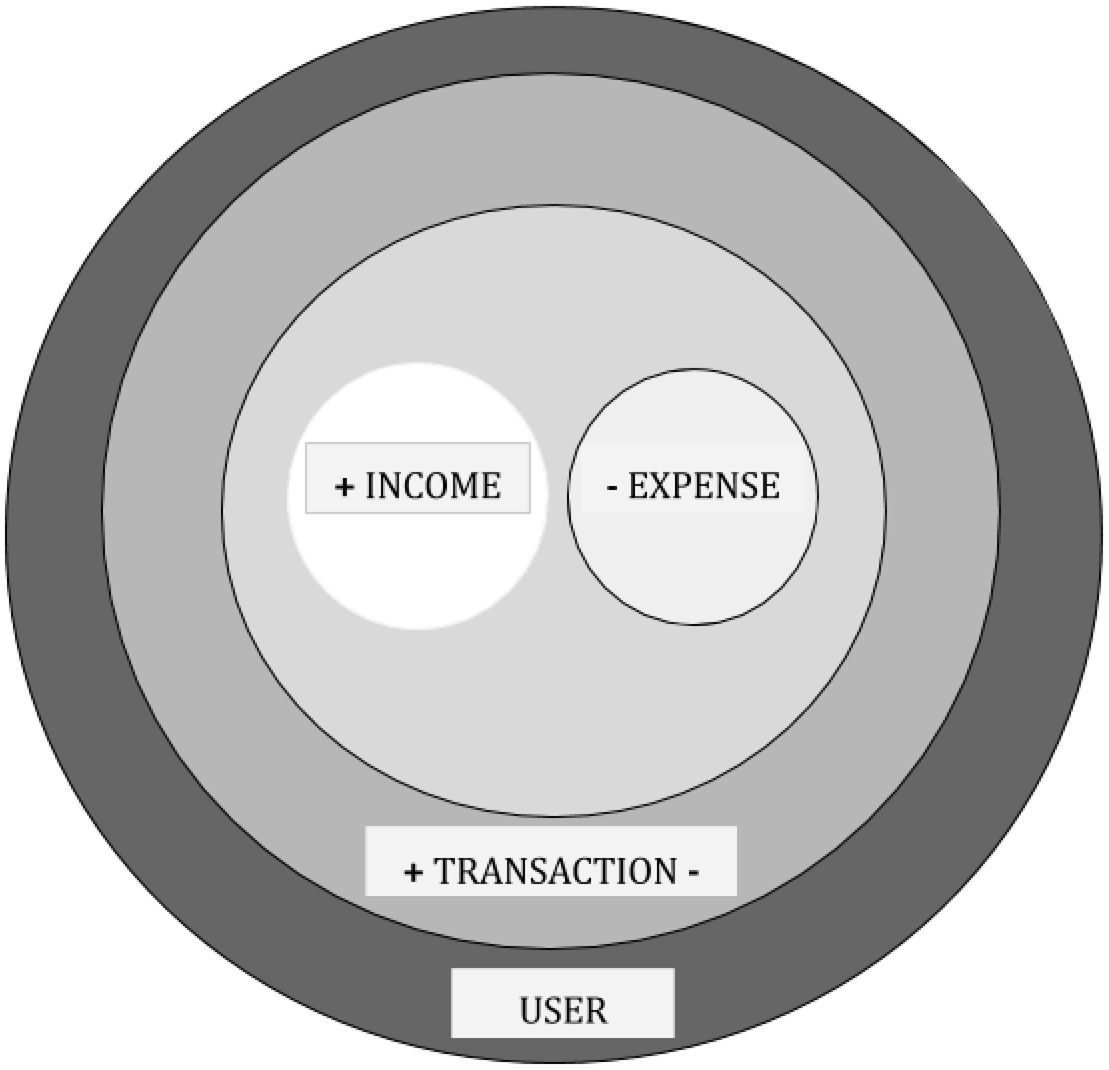
Salary Manager

To manage the transactions that involves in earning and spending the money. Kindly refer the diagram…

From Outer to Inner View:



Notables: Gist

- All the tables are made independent on each other and made in such a way that it wont hold any redundant data. This is achieved by the use of the junction or bridge tables and normalization concepts.
- Triggers, key constraints, if conditions and variables are also used.
- Log maintenance is given importance. All the user login information is stored separately. Similarly, all the table creation, updation and deletion are maintained.
- Tables are separated into blocks so that the purpose can be well identified. The blocks are
    1. Independent Table Block
        a. Address
    2. User Profile Table Block
        a. User
        b. User_Info
        c. User_Address
        d. User_Log
    3. Transaction Table Block
        a. Transaction
        b. Income_Transaction
        c. Expense_Transaction
    4. Earning Table Block
        a. Job
        b. Income
    5. Expense Table Block
        a. Expense

Scenario:

1) User Log In
    a) **Tables involded:**
        i) User
            (1) To maintain user account profiles
        ii) UserInfo
            (1) To maintain additional user information
        iii) UserAddress
            (1) To maintain different address used by user

2) Transaction Updation
    a) **Tables involved**
        i) Transaction
            (1) All transaction information will be available here
        ii) Income_transaction
            (1) All income transaction will be available here

iii) Expense_transaction
    (1) All expense transaction will be available here

b) Earning via jobs
   i) E-g: Employee working as a ‘Data Scientist’ (Job type 1) and then working as part time driver in Uber (Job type 2) that sums up the income
   ii) Tables involved:
      (1) Income
         (a) Salary earned by which job - such information avail here
      (2) Job
         (a) Job types
c) Expense
   i) Table involved:
      (1) Expense
         (a) Expense spent after earning info avail
d) Independent Tables
   i) Tables involved:
      (1) Address
         (a) This table will be the hub to store all address entries
         (b) In future, job addresses will also be using this one.

Concepts Review while designing the database:

Income Table (consider this for an example)
Job table

jobId	roleName	incomeAmount	earnedDate	note	**userId**
45	Desk Asst	3000.00	11/02/2017	Job @desk	3
46	Uber	2500.00	11/04/2017	job@ride	5
47	Web Design	1200.00	11/07/2017	job@design	3
48	Online Tutoring	1700.00	11/11/2017	lyft@ny	3
49	Desk Asst	1300.00	11/05/2017	dek@asst	5
50	Web Design	4100.00	11/21/2017	web@des	5

- **Observed**
    - 1NF achieved since atomic values alone present (i.e) only one attrib values present aptly.
    - 2NF achieved since all the entries available there depends upon the primary key jobId like jobRoleName, amount earned based on job, amount earned because of the job at a date, notes to note regarding job which depends upon job and finally, the userId who is mapped to jobId.
    - 3NF not achieved since, all the non-prime members are dependent on each other and data redundancy available which is not a good design. So, we split the tables accordingly now.

Income table

incomeId	incomeAmount	earnedDate	note	**userId**	**jobId**
63	3000.00	11/02/2017	Job @desk	3	45
64	2500.00	11/04/2017	job@ride	5	46
65	1200.00	11/07/2017	job@design	3	47
66	1700.00	11/11/2017	ont@ny	3	48
67	1300.00	11/05/2017	dek@asst	5	49
68	4100.00	11/21/2017	web@des	5	50

Job table

jobId	roleName
45	Desk Asst
46	Uber
47	Web Design
48	Online Tutoring

- 3NF though achieved, 3.5NF or BCNF can be used to further decompose this table to access the entries we desire faster. 3.5NF deals with the multi valued dependncies of primary keyed values. Here, jobId, userId and incomeId are in a single table. That can further be reduced to different tables. We can use the junction table and move the userId to **Transaction Table** (this is done because same scenario ocurred for **Expense table** as only one userId column when added to transaction table avoided the chance of including it into **Income and Expense table**).

Income table

incomeId	incomeAmount	earnedDate	note	**jobId**	**userId**
63	3000.00	11/02/2017	Job @desk	45	3
64	2500.00	11/04/2017	job@ride	46	5
65	1200.00	11/07/2017	job@design	47	3
66	1700.00	11/11/2017	ont@ny	48	3
67	1300.00	11/05/2017	dek@asst	49	5
68	4100.00	11/21/2017	web@des	50	3

IncomeTransaction table (Junction or Bridge table)

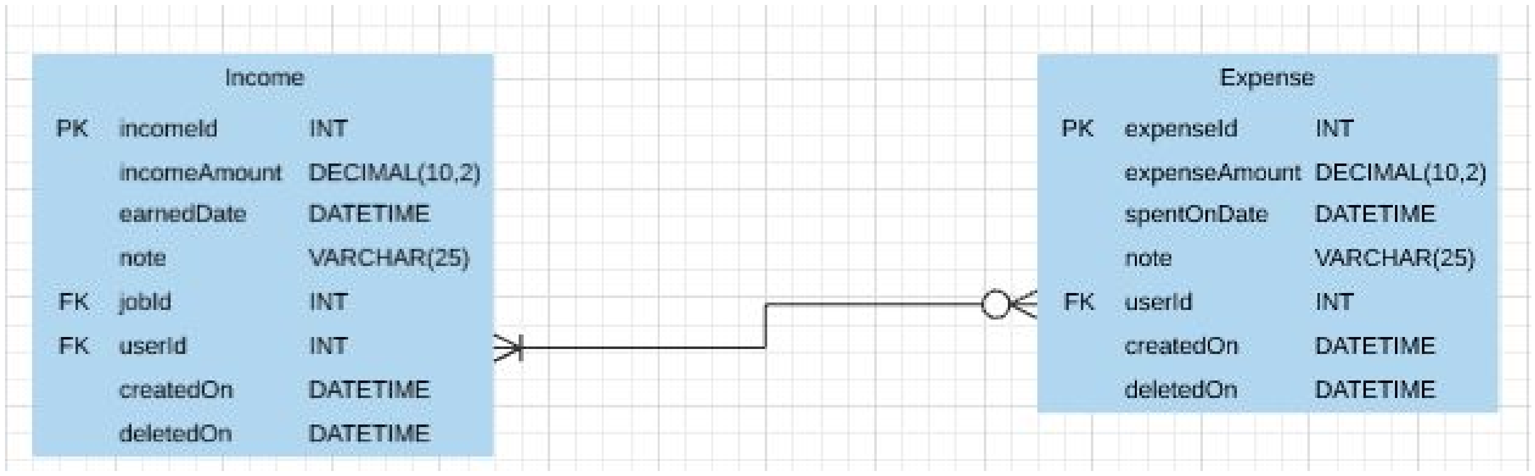
incomeTransactionId	**transactionId**	**incomeId**
52	84	63
53	85	64
54	86	65
55	87	66
56	88	67
57	89	68

Transaction table

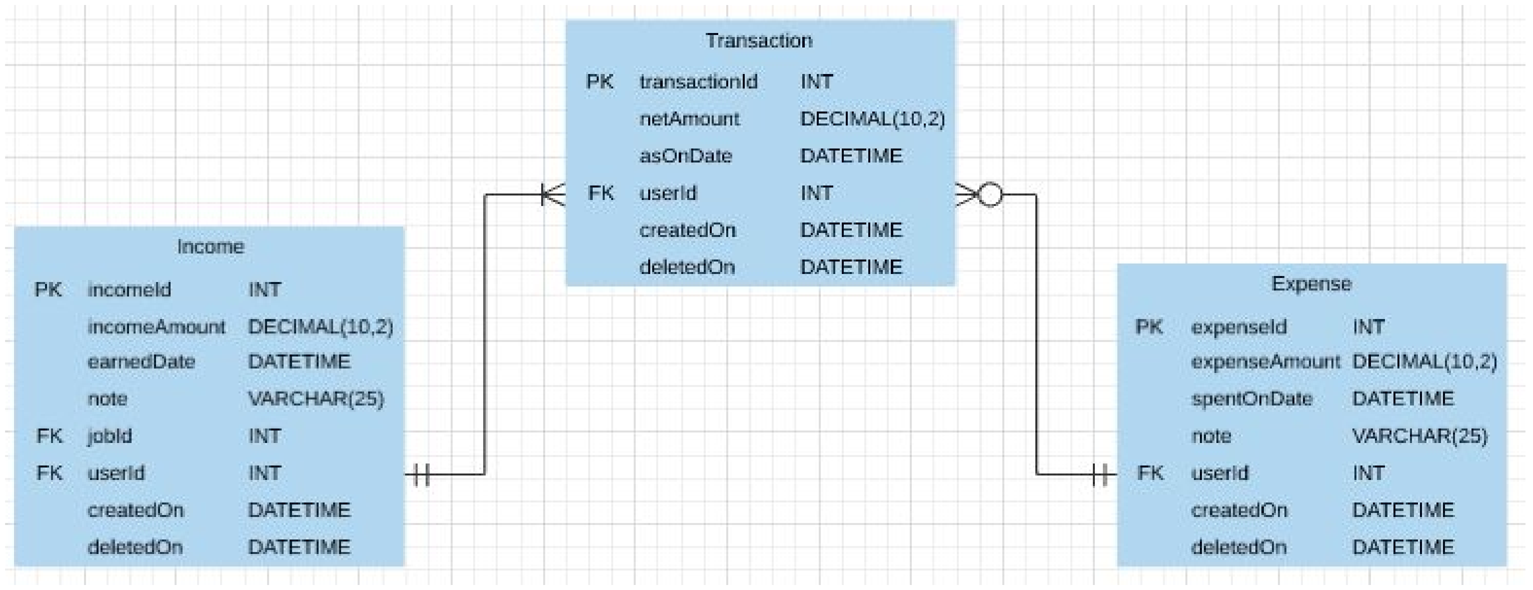
transactionId	**userId**	netAmount	asOnDate	isIncome
84	3	3000.00	11/02/2017	Y
85	5	2500.00	11/04/2017	Y
86	3	4200.00	11/07/2017	Y
87	3	5900.00	11/11/2017	Y

- Also, **Transaction table** behaves as **Bridge table** between **Job+Income** and **Expense table** since many to many relationship between job/income and expense table observed.

Before Bridge Table



After Bridge Table



Future Improvements:

- Job address addition along with location access (on click, will give from: current_location to: job address)
- Loan from others and Loan to others feature to be added
- User profile image feature
- User email password recovery feature