
**National Institute of Business Management**

**School of Computing**

**Higher National Diploma in Software Engineering**

**Data Management 2**

**Melony**

**Personal Finance Management System**

**Submitted By**


| | |
|---|---|
| **COHNDSE251F-058** | **W W Thilini Piyumika** |
| **COHNDSE251F-059** | **P A R Sahas Suraweera** |


**Date of Submission: 5th of November 2025**

## Acknowledgment

I would like to thank our lecturers ,group members and supervisors for their continuous support and guidance through this project. Also thankful to the National Institute of Business Management for providing guidance and resources . We have named this application as ‘MELONY’ and provides a special tribute to Melanie Janine Kanaka, Excellent achievement of the first Asian woman to be CIMA President as she empowers the women participation in finance management.

## Abstract

Our Melony - personal finance management system was created for users to manage their personal financial activities efficiently while maintaining data transactions between local SQLite and Oracle centralized databases. Server- first technique and Local - first techniques are used for seamless and error free data transactions. SQLite support for users to continue operations offline.To prevent data duplication and loss we have used synchronization techniques . The project was built using client- server architecture and for the backend and API we have used node.js with Express .js and for the frontend we have used react .

# Table of contents

# Chapter 1. Introduction

## 1.1 Problem Definition

In the modern world personal finance management holds an important place when it comes to expenditure management . Managing finances is one of the most important parts not only for the individuals but also for every business entity. As for the individuals managing personal finance has become a crucial part in their lives because of the rising living cost and inflation .

Especially as a part of technological growth, doing payments is way easier than the past manual methods because of that as people who are living in the Sri lankans face shortage of money because they tend to make unnecessary payments most of the time. And also when it comes to liability payments such as credit cards , loan payments, sometimes people forget to make payments on time to address all these issues.

Hence the “Melony” personal finance management system is developed to support individuals to track their income, expenses , savings and liabilities in an organized manner. Our solution not only supports financial management but also manages savings, budgeting, debt management and provides data to take effective financial decisions.

## 1.2 Objectives

- Create a role base and secure personal finance management system for managing an individual’s financial activities.
- Tracking all transactions that users process via their own accounts and other third party accounts to show their income and expenses in a clear way.
- Allow users to categorize their budgets and set usage limits and reminders for better tracking of income usage.
- Motivate users to set their saving goals and achieve it.
- Generate summarized reports to get an idea of transactions.
- Preventing data loss and duplication by data synchronization with local and central databases.

# Chapter 2. System Review

## 2.1 Description of the System

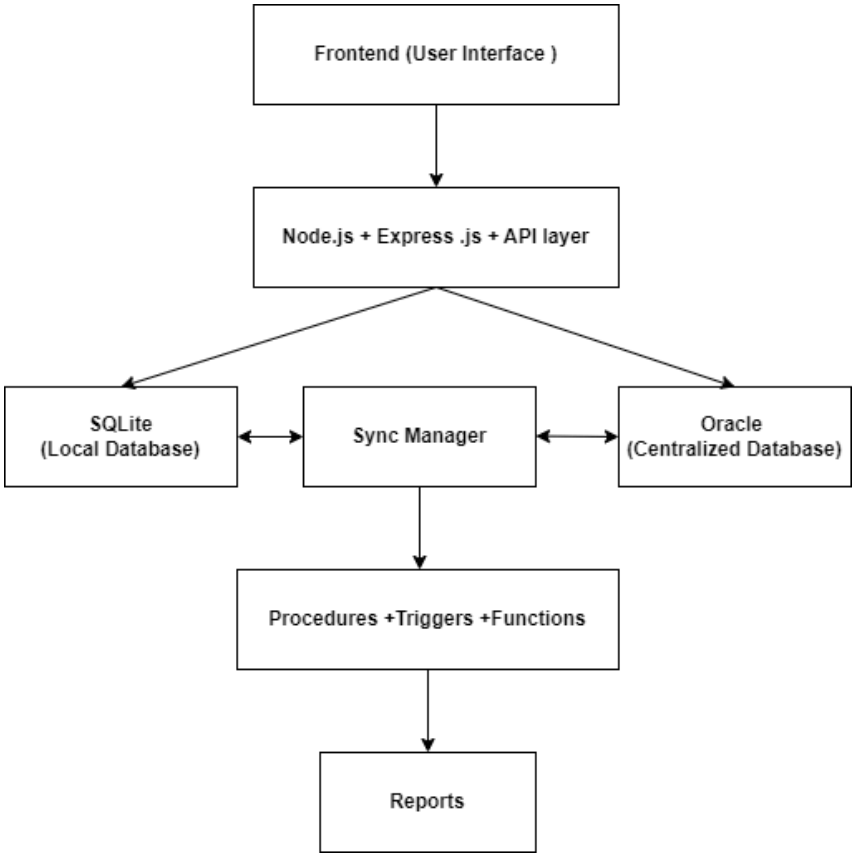
**Melony -Personal Finance Management System** is a digital solution that supports individuals to manage their income sources and other financial activities. Our system allows users to record their income ,set and manage budget, manage savings and manage bank accounts and transactions . Melony financial management system also works offline because it is maintaining synchronization between SQLite local database and centralized Oracle database.

Individual users can create an one user account using an email and after creating an user account they can create an account based on the type whether it's an asset based account or liability based account .Once the user made a transaction under any of these categories it will automatically update the balance.

**Key Features**

- Secure User management with email logins for regular users and admin users .
- Users can create an user account by including user name, email, password address and occupation.
- Users can set a budget by including maximum limit , warning limit , start date and end date , it also contains transaction categories like Healthcare,Dining, Transport,utilities and groceries .
- Users can create two different account types , assets and liabilities.
- Users can set reminders for due dates for liability payments and for budget deadlines.
- Users can track their expenditure daily basis and can analyse .
- Automated fund transfers between accounts.
- The System Support allows users to plan and manage savings systematically by goal based savings tracking.
- The system is supported to synchronize with a centralized oracle database by securing the access to offline usage via SQLite.
- The system allows users to create notes.

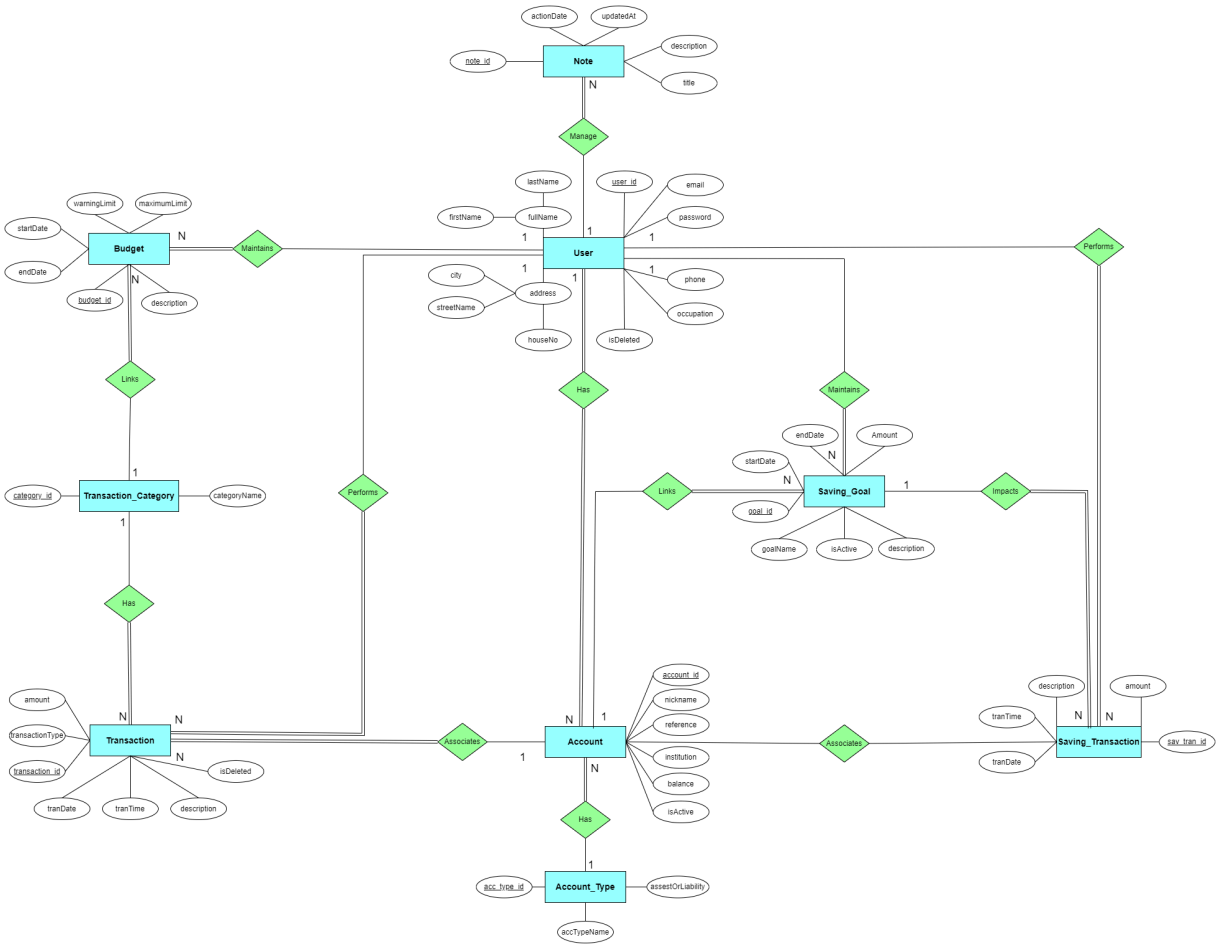
## 2.2 Architecture of the System



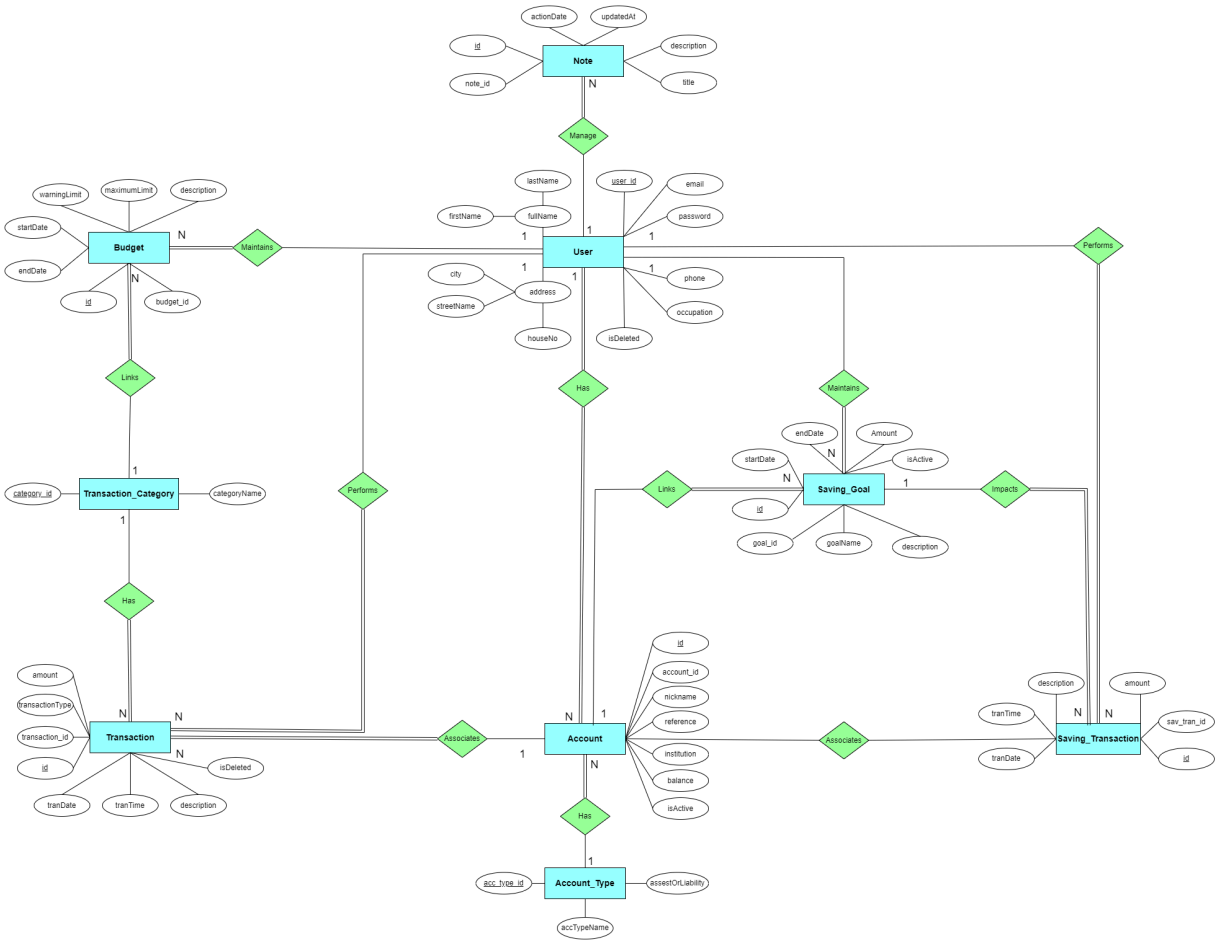
# Chapter 3 .Physical and Logical Database Design

## 3.1 Logical Design

### 3.1.1 Sqlite



### 3.1.2 Oracle


actionDate
updatedAt
id
Note
description
note_id
title
N
Manage
lastName
user_id
email
firstName
fullName
password
warningLimit
maximumLimit
description
startDate
Budget
N
Maintains
User
Performs
endDate
N
id
budget_id
city
address
phone
streetName
houseNo
occupation
isDeleted
Links
Has
Maintains
endDate
Amount
startDate
N
isActive
category_id
Transaction_Category
categoryName
Performs
Links
Saving_Goal
Impacts
id
goal_id
goalName
description
Has
amount
transactionType
transaction_id
id
Transaction
isDeleted
tranDate
tranTime
description
Associates
Account
id
account_id
nickname
reference
institution
balance
isActive
Associates
description
amount
tranTime
Saving_Transaction
sav_tran_id
tranDate
id
Has
acc_type_id
Account_Type
assestOrLiability
accTypeName

| Entity Name | Description |
|---|---|
| User | Individuals who are using the system for manage their personal financial activities |
| Budget | Category based financial limits that users can plan and control |
| Transaction Category | Financial Expenditure categories like Healthcare, utilities and Transport |
| Transaction | Handles user transactions , that users made through a assets and liability accounts |
| Account | Represent all financial accounts that user manage |
| Account Type | Financial accounts categorization such as assets or liabilities |
| Saving Goal | represent financial management movement that users can set how much they want to save to achieve their savings goals |
| Saving Transaction | Track transactions between accounts that made for achieve savings goals |
| Note | All the notes that users create for personal financial management |

## 3.2 Physical Design

### 3.2.1 SQLite

1. UserInfo

```
CREATE TABLE UserInfo (
    user_id    INTEGER PRIMARY KEY AUTOINCREMENT,
    firstName  TEXT    NOT NULL,
    lastName   TEXT    NOT NULL,
    email      TEXT    UNIQUE NOT NULL,
    password   TEXT    NOT NULL,
    occupation TEXT    NOT NULL,
    houseNO    TEXT    NOT NULL,
    streetName TEXT    NOT NULL,
    city       TEXT    NOT NULL,
    phone      TEXT    NOT NULL,
    isDeleted  TEXT    DEFAULT 'N' CHECK (isDeleted IN ('Y', 'N') )
);
```

2. Note

```
CREATE TABLE Note (
    note_id     INTEGER   PRIMARY KEY AUTOINCREMENT,
    user_id     INTEGER   NOT NULL,
    title       TEXT      NOT NULL,
    description TEXT      NOT NULL,
    actionDate  DATE,
    updatedAt   TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES UserInfo (user_id)
);
```

3. Account_Type

```
CREATE TABLE Account_Type (
    acc_type_id      INTEGER PRIMARY KEY AUTOINCREMENT,
    accTypeName      TEXT    NOT NULL,
    assetOrLiability TEXT    CHECK (assetOrLiability IN ('Asset', 'Liability') )
);
```

4. Account

```
CREATE TABLE Account (
    account_id  INTEGER PRIMARY KEY AUTOINCREMENT,
    user_id     INTEGER NOT NULL,
    acc_type_id INTEGER NOT NULL,
    nickname    TEXT,
    reference   TEXT,
    institution TEXT,
    balance     REAL    DEFAULT 0,
    isActive    TEXT    DEFAULT 'Y' CHECK (isActive IN ('Y', 'N') ),
    FOREIGN KEY (acc_type_id) REFERENCES Account_Type (acc_type_id),
    FOREIGN KEY (user_id) REFERENCES UserInfo (user_id)
);
```

5. Transaction_Category

```
CREATE TABLE Transaction_Category (
    category_id  INTEGER PRIMARY KEY AUTOINCREMENT,
    categoryName TEXT    NOT NULL
);
```

6. Transaction_Info

```
CREATE TABLE Transaction_Info (
    transaction_id  INTEGER   PRIMARY KEY AUTOINCREMENT,
    user_id         INTEGER   NOT NULL,
    account_id      INTEGER   NOT NULL,
    category_id     INTEGER,
    amount          REAL      NOT NULL,
    transactionType TEXT      CHECK (transactionType IN ('Income', 'Expense') ),
    description     TEXT,
    tranDate        DATE      DEFAULT CURRENT_DATE,
    tranTime        TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    isDeleted       TEXT      DEFAULT 'N' CHECK (isDeleted IN ('Y', 'N') ),
    FOREIGN KEY (user_id) REFERENCES UserInfo (user_id),
    FOREIGN KEY (account_id) REFERENCES Account (account_id),
    FOREIGN KEY (category_id)REFERENCES Transaction_Category (category_id)
);
```

7. Budget

```
CREATE TABLE Budget (
    budget_id    INTEGER PRIMARY KEY AUTOINCREMENT,
    user_id      INTEGER NOT NULL,
    category_id  INTEGER NOT NULL,
    startDate    DATE    NOT NULL,
    endDate      DATE    NOT NULL,
    warningLimit REAL    NOT NULL,
    maximumLimit REAL    NOT NULL,
    description  TEXT,
    FOREIGN KEY (user_id) REFERENCES UserInfo (user_id),
    FOREIGN KEY (category_id) REFERENCES Transaction_Category (category_id)
);
```

8.Saving_Goal

```
CREATE TABLE Saving_Goal (
    goal_id       INTEGER PRIMARY KEY AUTOINCREMENT,
    user_id       INTEGER NOT NULL,
    account_id    INTEGER NOT NULL,
    goalName      TEXT    NOT NULL,
    targetAmount  REAL,
    currentAmount REAL    DEFAULT 0,
    startDate     DATE    NOT NULL,
    endDate       DATE    NOT NULL,
    isActive      TEXT    DEFAULT 'Y' CHECK (isActive IN ('Y', 'N') ),
    FOREIGN KEY (user_id) REFERENCES UserInfo (user_id),
    FOREIGN KEY (account_id) REFERENCES Account (account_id)
);
```

9.Saving_Transaction

```
CREATE TABLE Saving_Transaction (
    sav_tran_id INTEGER   PRIMARY KEY AUTOINCREMENT,
    user_id               INTEGER   NOT NULL,
    goal_id               INTEGER   NOT NULL,
    account_id            INTEGER   NOT NULL,
    amount                REAL,
    description           TEXT,
    tranDate              DATE      DEFAULT CURRENT_DATE,
    tranTime              TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES UserInfo (user_id),
    FOREIGN KEY (goal_id) REFERENCES Saving_Goal (goal_id),
    FOREIGN KEY (account_id) REFERENCES Account (account_id)
);
```

3.2.2 Oracle

1. UserInfo

```
CREATE TABLE UserInfo (
    user_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    firstName VARCHAR2(50) NOT NULL,
    lastName VARCHAR2(50)NOT NULL,
    email VARCHAR2(100) UNIQUE NOT NULL,
    password VARCHAR2(255) NOT NULL,
    occupation VARCHAR2(50)NOT NULL,
    houseNO VARCHAR2(50)NOT NULL,
    streetName VARCHAR2(100)NOT NULL,
    city VARCHAR2(50)NOT NULL,
    phone VARCHAR2(20) NOT NULL,
    isDeleted CHAR(1) DEFAULT 'N' CHECK(isDeleted IN ('Y','N'))
);
```

2. Note

```
CREATE TABLE Note (
    id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    note_id NUMBER NOT NULL,
    user_id NUMBER NOT NULL,
    title VARCHAR2(100) NOT NULL,
    description VARCHAR2(255) NOT NULL,
    actionDate DATE,
    updatedAt TIMESTAMP DEFAULT SYSTIMESTAMP,
    CONSTRAINT note_unique UNIQUE (note_id, user_id),
    FOREIGN KEY (user_id) REFERENCES UserInfo(user_id)
);
```

3. Account_type

```
CREATE TABLE Account_Type (
    acc_type_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    accTypeName VARCHAR2(50) NOT NULL,
    assetOrLiability VARCHAR2(50) CHECK(assetOrLiability IN ('Asset','Liability'))
);
```

4. Account

```
CREATE TABLE Account (
    id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    account_id NUMBER NOT NULL,
    user_id NUMBER NOT NULL,
    acc_type_id NUMBER NOT NULL,
    nickname VARCHAR2(50),
    reference VARCHAR2(50),
    institution VARCHAR2(50),
    balance NUMBER(12,2) DEFAULT 0,
    isActive CHAR(1) DEFAULT 'Y' CHECK (isActive IN ('Y', 'N')),
    CONSTRAINT acc_unique UNIQUE (account_id, user_id),
    FOREIGN KEY (user_id) REFERENCES UserInfo(user_id),
    FOREIGN KEY (acc_type_id) REFERENCES Account_Type(acc_type_id)
);
```

5. Transaction_Category

```
CREATE TABLE Transaction_Category (
    category_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    categoryName VARCHAR2(100) NOT NULL
);
```

6. Transaction_info

```
CREATE TABLE Transaction_Info (
    id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    transaction_id NUMBER NOT NULL,
    user_id NUMBER NOT NULL,
    account_id NUMBER NOT NULL,
    category_id NUMBER,
    amount NUMBER(12,2) NOT NULL,
    transactionType VARCHAR2(20) CHECK (transactionType IN ('Income', 'Expense')),
    description VARCHAR2(255),
    tranDate DATE DEFAULT SYSDATE,
    tranTime TIMESTAMP DEFAULT SYSTIMESTAMP,
    isDeleted CHAR(1) DEFAULT 'N' CHECK (isDeleted IN ('Y','N')),
    CONSTRAINT trans_unique UNIQUE (transaction_id, user_id),
    FOREIGN KEY (account_id, user_id) REFERENCES Account(account_id, user_id),
    FOREIGN KEY (category_id) REFERENCES Transaction_Category(category_id)
);
```

7. Budget

```
CREATE TABLE Budget (
    id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    budget_id NUMBER NOT NULL,
    user_id NUMBER NOT NULL,
    category_id NUMBER NOT NULL,
    startDate DATE NOT NULL,
    endDate DATE NOT NULL,
    warningLimit NUMBER(12,2) NOT NULL,
    maximumLimit NUMBER(12,2) NOT NULL,
    description VARCHAR2(255),
    CONSTRAINT budget_unique UNIQUE (budget_id, user_id),
    FOREIGN KEY (user_id) REFERENCES UserInfo(user_id),
    FOREIGN KEY (category_id) REFERENCES Transaction_Category(category_id)
);
```

8. Saving_Goal

```
CREATE TABLE Saving_Goal (
    id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    goal_id NUMBER NOT NULL,
    user_id NUMBER NOT NULL,
    account_id NUMBER NOT NULL,
    goalName VARCHAR2(100) NOT NULL,
    targetAmount NUMBER(12,2),
    currentAmount NUMBER(12,2) DEFAULT 0,
    startDate DATE NOT NULL,
    endDate DATE NOT NULL,
    isActive CHAR(1) DEFAULT 'Y' CHECK(isActive IN ('Y','N')),
    CONSTRAINT goal_unique UNIQUE (goal_id, user_id),
    FOREIGN KEY (user_id) REFERENCES UserInfo(user_id),
    FOREIGN KEY (account_id, user_id) REFERENCES Account(account_id, user_id)
);
```

9. Saving_Transaction

```
CREATE TABLE Saving_Transaction (
    id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    sav_tran_id NUMBER NOT NULL,
    user_id NUMBER NOT NULL,
    goal_id NUMBER NOT NULL,
    account_id NUMBER NOT NULL,
    amount NUMBER(12,2),
    description VARCHAR2(255),
    tranDate DATE DEFAULT SYSDATE,
    tranTime TIMESTAMP DEFAULT SYSTIMESTAMP,
    CONSTRAINT saving_trans_unique UNIQUE (saving_transaction_id, user_id),
    FOREIGN KEY (user_id) REFERENCES UserInfo(user_id),
    FOREIGN KEY (goal_id, user_id) REFERENCES Saving_Goal(goal_id, user_id),
    FOREIGN KEY (account_id, user_id) REFERENCES Account(account_id, user_id)
);
```

# Chapter 4. Table Structures and Schema Overview

## 4.1 SQLite Table Structures and Overview

1. UserInfo

| Column | Data Type | Constraint | Description |
|---|---|---|---|
| User_id | INTEGER | Primary Key , Autoincrement | User ID |
| firstName | TEXT | Not Null | First name of user |
| lastName | TEXT | Not Null | Last name of user |
| email | TEXT | Unique, Not Null | Email of user |
| password | TEXT | Not Null | User password |
| occupation | TEXT | Not Null | Occupation of user |
| houseNo | TEXT | Not Null | House No of user |
| streetName | TEXT | Not Null | Street name of address |
| city | TEXT | Not Null | City of address |
| phone | TEXT | Not Null | User contact number |
| isDeletetd | TEXT | Check | User deleted or not |

2. Note

| Column Name | Data Type | Constraint | Description |
|---|---|---|---|
| note_id | INTEGER | Primary Key, Autoincrement | Unique id for note |
| user_id | INTEGER | Foreign Key, Not Null | Unique id to define user |
| title | TEXT | Not Null | Title of note |
| description | TEXT | Not Null | Description |
| actionDate | DATE | | A date of reminder |
| updatedAt | TIMESTAMP | Default Current_timestamp | Last updated time |

3. Account_Type

| Column Name | Data Type | Constraint | Description |
|---|---|---|---|
| acc_type_id | INTEGER | Primary Key, Autoincrement | Unique identification for account type |
| accTypeName | TEXT | Not Null | Account type name |
| assetOrLiability | TEXT | Check | Identity account is a liability or asset |

4. Account

| Column Name | Data Type | Constraint | Description |
|---|---|---|---|
| account_id | INTEGER | Primary Key, Autoincrement | Unique identification for account |
| user_id | TEXT | Not Null | User id of account owner |
| acc_type_id | TEXT | Not Null | Asset or liability |
| nickname | TEXT | | User entered name for the account |
| reference | TEXT | | Reference code |
| institution | TEXT | | Bank name or institution name |
| balance | REAL | Default 0 | Current balance of the account |
| isActive | TEXT | Default 'Y' Check (Isactive In ('Y', 'N') ) | To check whether account is active or the only values enter 'Y' or 'N' |

5. Transaction_Category

| Column Name | Data Type | Constraint | Description |
|---|---|---|---|
| category_id | INTEGER | Primary Key, Autoincrement | Unique id or each category |
| categoryName | TEXT | Not Null | Name of the category |

6. Transaction_Info

| Column Name | Data Type | Constraint | Description |
|---|---|---|---|
| transaction_id | INTEGER | Primary key, autoincrement | Unique id for transaction |
| user_id | INTEGER | Not null | Unique id that can identify user |
| account_id | INTEGER | Foreign key, not null | Unique id to identify account |
| category_id | INTEGER | Foreign key, not null | Unique id to identify category |
| amount | REAL | Not null | Transaction amount |
| transactionType | TEXT | Check | Transaction type such as income or expense |
| description | TEXT | | Description about transaction |
| tranDate | DATE | Default current_date | Transaction date |
| tranTime | TIMESTAMP | Default current_timestamp | Transaction time |

7. Budget

| Column Name | Data Type | Constraint | Description |
|---|---|---|---|
| budget_id | INTEGER | Primary key, autoincrement | Unique id for budget type |
| user_id | INTEGER | Foreign key, not null | Unique id to define user |
| category_id | INTEGER | Foreign key, not null | Unique id to identify category |
| startDate | DATE | Foreign key, not null | Start date of budget allocation |
| endDate | DATE | Not null | End date of budget allocation |
| warningLimit | REAL | | Warning budget amount limit |
| maximumLimit | REAL | | Maximum amount of budget allocation |
| description | TEXT | Default current_date | Description |

8. Saving_Goal

| Column Name | Data Type | Constraint | Description |
|---|---|---|---|
| goal_id | INTEGER | Primary key, autoincrement | Unique id of goal |
| user_id | INTEGER | Foreign key, not null | Unique id to define user |
| account_id | INTEGER | Foreign key, not null | Unique id to identify account |
| goalName | TEXT | Not null | Saving goal name |
| targetAmount | REAL | Not null | Savings amount |
| currentAmount | REAL | Default 0 | Current Amount |
| startDate | DATE | Not null | Start date |
| endDate | DATE | Not null | End date |
| isActive | TEXT | Check (isactive in ('y', 'n') ) | Savings goal active status |

9. Saving Transaction

| Column Name | Data Type | Constraint | Description |
|---|---|---|---|
| sav_tran_id | INTEGER | Primary Key, Autoincrement | Unique id of savings transaction |
| user_id | INTEGER | Foreign Key, Not Null | Unique id to define user |
| goal_id | INTEGER | Foreign Key, Not Null | Unique id of goal |
| account_id | INTEGER | Foreign Key, Not Null | Unique id to identify account |
| amount | REAL | | Savings amount |
| description | TEXT | | description |
| tranDate | DATE | Default Current_date | Transaction date |
| tranTime | TIMESTAMP | Default Current_timestamp | Transaction time |

## 4.2 Oracle Table Structures Overview

1. UserInfo

| Column Name | Data Type | Constraint | Description |
|---|---|---|---|
| User_id | NUMBER | Primary Key , Autoincrement | User ID |
| firstName | VARCHAR2 | Not Null | First name of user |
| lastName | VARCHAR2 | Not Null | Last name of user |
| email | VARCHAR2 | Unique , Not Null | Email of user |
| password | VARCHAR2 | Not Null | User password |
| occupation | VARCHAR2 | Not Null | Occupation of user |
| houseNo | VARCHAR2 | Not Null | House No of user |
| streetName | VARCHAR2 | Not Null | Street name of address |

2. Note

| Column Name | Data Type | Constraint | Description |
|---|---|---|---|
| id | NUMBER | Primary Key, Autoincrement | Unique id for note |
| note_id | NUMBER | Not,Null,<br>Composite Unique(User_id) | Local id for note |
| user_id | NUMBER | Foreign Key, Not Null,<br>Composite Unique(Note_id) | Unique id to define user |
| title | VARCHAR2 | Not Null | Title of note |
| description | VARCHAR2 | Not Null | Description |
| actionDate | DATE | | A date of reminder |
| updatedAt | TIMESTAMP | Default Current_timestamp | Last updated time |

3. Account_Type

| Column Name | Data Type | Constraint | Description |
|---|---|---|---|
| acc_type_id | NUMBER | Primary Key, Autoincrement | Unique identification for account type |
| accTypeName | VARCHAR2 | Not Null | Account type name |
| assetOrLiability | VARCHAR2 | Check | Identity account is a liability or asset |

4. Account

| Column Name | Data Type | Constraint | Description |
|---|---|---|---|
| id | NUMBER | Primary Key, Autoincrement | Unique identification for account |
| account_id | NUMBER | Not Null,<br>Composite Unique(User_id) | Local identification for account |
| user_id | VARCHAR2 | Foreign Key, Not Null, Composite Unique(Account_id) | User id of account owner |
| acc_type_id | VARCHAR2 | Not Null | Asset or liability |
| nickname | VARCHAR2 | | User entered name for the account |
| reference | VARCHAR2 | | Reference code |
| institution | VARCHAR2 | | Bank name or institution name |
| balance | NUMBER | Default 0 | Current balance of the account |
| isActive | VARCHAR2 | Default 'Y' Check<br>(Isactive In ('Y', 'N') ) | To check whether account is active or the only values enter 'Y' or 'N' |

5. Transaction_Category

| Column Name | Data Type | Constraint | Description |
|---|---|---|---|
| category_id | NUMBER | Primary Key, Autoincrement | Unique id or each category |
| categoryName | VARCHAR2 | Not Null | Name of the category |

6. Transaction_Info

| Column Name | Data Type | Constraint | Description |
|---|---|---|---|
| id | NUMBER | Primary Key, Autoincrement | Unique identification for account type |
| transaction_id | NUMBER | Not Null, Composite Unique(user_id) | Local id for transaction |
| user_id | NUMBER | Composite Foreign Key(account_id), Not Null, Composite Unique(transaction_id) | Unique id that can identify user |
| account_id | NUMBER | Composite Foreign Key(account_id), Not Null | Unique id to identify account |
| category_id | NUMBER | Foreign Key | Unique id to identify category |
| amount | NUMBER | Not Null | Transaction amount |
| transactionType | VARCHAR2 | Check (transactionType IN ('Income', 'Expense')) | Transaction type such as income or expense |
| description | VARCHAR2 | | Description about transaction |
| tranDate | DATE | Default current_date | Transaction date |
| tranTime | TIMESTAMP | Default current_timestamp | Transaction time |

7. Budget

| Column Name | Data Type | Constraint | Description |
|---|---|---|---|
| id | NUMBER | Primary Key, Autoincrement | Unique id for budget type |
| budget_id | NUMBER | Not Null, Composite Unique(user_id) | |
| user_id | NUMBER | Foreign Key, Not Null, Composite Unique(budget_id) | Unique id to define user |
| category_id | NUMBER | Foreign Key | Unique id to identify category |
| startDate | DATE | Foreign Key | Start date of budget allocation |
| endDate | DATE | Not Null | End date of budget allocation |
| warningLimit | NUMBER | Not Null | Warning budget amount limit |
| maximumLimit | NUMBER | | Maximum amount of budget allocation |
| description | VARCHAR2 | Default Current_date | Description |

8. Saving_Goal

| Column Name | Data Type | Constraint | Description |
|---|---|---|---|
| id | NUMBER | Primary Key, Autoincrement | Unique id of goal |
| goal_id | NUMBER | Not Null,<br>Composite Unique(user_id) | Local id of goal |
| user_id | NUMBER | Foreign Key, Composite Foreign Key(account_id), Not Null,<br>Composite Unique(goal_id) | Unique id to define user |
| account_id | NUMBER | Composite Foreign Key(user_id),<br>Not Null | Unique id to identify account |
| goalName | VARCHAR2 | Not Null | Saving goal name |
| targetAmount | NUMBER | Not Null | Savings amount |
| currentAmount | NUMBER | DEFAULT 0 | Current Amount |
| startDate | DATE | Not Null | Start date |
| endDate | DATE | Not Null | End date |
| isActive | CHAR | CHECK (isActive IN ('Y', 'N') ) | Savings goal active status |

9. Saving_Transaction

| Column Name | Data Type | Constraint | Description |
|---|---|---|---|
| id | NUMBER | Primary Key, Autoincrement | Unique id of savings transaction |
| sav_trans_id | NUMBER | Not Null,<br>Composite Unique(user_id) | Unique id of savings transaction |
| user_id | NUMBER | Foreign Key, Composite Foreign Key(goal_id), Composite Foreign Key(account_id), Not Null, Composite Unique(sav_tran_id) | Unique id to define user |
| goal_id | NUMBER | Composite Foreign Key(user_id), Not Null | Unique id of goal |
| account_id | NUMBER | Composite Foreign Key(user_id), Not Null | Unique id to identify account |
| amount | NUMBER | | Savings amount |
| description | VARCHAR2 | | description |
| tranDate | DATE | Default Current_date | Transaction date |
| tranTime | TIMESTAMP | Default Current_timestamp | Transaction time |

# Chapter 5. Oracle PL/SQL Procedures Implementation

## 5.1 Procedure for Update Account Balance After Transaction

Once the user processes a transaction related to income or expense , get the total amount of income or expense then update account balance according to the transaction amount.

```
CREATE OR REPLACE PROCEDURE update_account_balance (
    p_account_id IN NUMBER,
    p_user_id    IN NUMBER
) AS
    v_total_income  NUMBER := 0;
    v_total_expense NUMBER := 0;
    v_new_balance   NUMBER := 0;
BEGIN

    SELECT NVL(SUM(amount), 0)
    INTO v_total_income
    FROM Transaction_Info
    WHERE account_id = p_account_id
      AND user_id = p_user_id
      AND transactionType = 'Income'
      AND isDeleted = 'N';

    SELECT NVL(SUM(amount), 0)
    INTO v_total_expense
    FROM Transaction_Info
    WHERE account_id = p_account_id
      AND user_id = p_user_id
      AND transactionType = 'Expense'
      AND isDeleted = 'N';
    v_new_balance := v_total_income - v_total_expense;


    UPDATE Account
    SET balance = v_new_balance
    WHERE account_id = p_account_id
      AND user_id = p_user_id;

    COMMIT;
END;
/
```

**Trigger for Auto Update After Insertion**

This trigger is created to confirm when a new transaction is inserted , automatically update account balance.

```
CREATE OR REPLACE TRIGGER trg_update_balance_after_insert
AFTER INSERT ON Transaction_Info
FOR EACH ROW
BEGIN
    update_account_balance(:NEW.account_id, :NEW.user_id);
END;
/
```

**Trigger for Auto Update After Soft Delete**

This trigger is for when user wanted to delete a transaction then translation is mark as deleted , reduce the the transaction amount and update account balance.

```
CREATE OR REPLACE TRIGGER trg_update_balance_after_soft_delete
AFTER UPDATE OF isDeleted ON Transaction_Info
FOR EACH ROW
WHEN (NEW.isDeleted = 'Y')
BEGIN
    update_account_balance(:NEW.account_id, :NEW.user_id);
END;
/
```

## 5.2 Function for Calculate User's Net Worth

Calculate user's net worth and return total value in a text format.

```
CREATE OR REPLACE FUNCTION get_net_worth (
    p_user_id IN NUMBER
) RETURN VARCHAR2
AS
    v_assets       NUMBER := 0;
    v_liabilities  NUMBER := 0;
    v_net          NUMBER := 0;
BEGIN
    SELECT
NVL(SUM(CASE WHEN t.assetOrLiability = 'Asset' THEN a.balance ELSE 0 END), 0),
NVL(SUM(CASE WHEN t.assetOrLiability = 'Liability' THEN a.balance ELSE 0 END),0)
    INTO
        v_assets, v_liabilities
    FROM
        Account a
        JOIN Account_Type t ON a.acc_type_id = t.acc_type_id
    WHERE
        a.user_id = p_user_id
        AND a.isActive = 'Y';

    v_net := v_assets - v_liabilities;

    RETURN 'Total Assets: ' || v_assets ||
           ', Total Liabilities: ' || v_liabilities ||
           ', Net Worth: ' || v_net;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN 'No accounts found for this user.';
END;
/
```

## 5.3 Procedure for Check and Notify Due Notes

Check notes and then automatically check whether the action date is equal to date and then send a reminder.

```
CREATE OR REPLACE PROCEDURE check_due_notes
IS
BEGIN
    FOR r IN (
        SELECT n.note_id, n.user_id, n.title, n.description
        FROM Note n
        WHERE TRUNC(n.actionDate) = TRUNC(SYSDATE)
    )
    LOOP
     INSERT INTO Note_Alerts (note_id, user_id, message)
     VALUES (
            r.note_id,
            r.user_id,
            ' Reminder: "' || r.title || '" is due today - ' || r.description
        );
DBMS_OUTPUT.PUT_LINE('Reminder created for Note ' || r.note_id || ': ' || r.title);
    END LOOP;

    COMMIT;
END;
/
```

# Chapter 6. Synchronization Mechanism

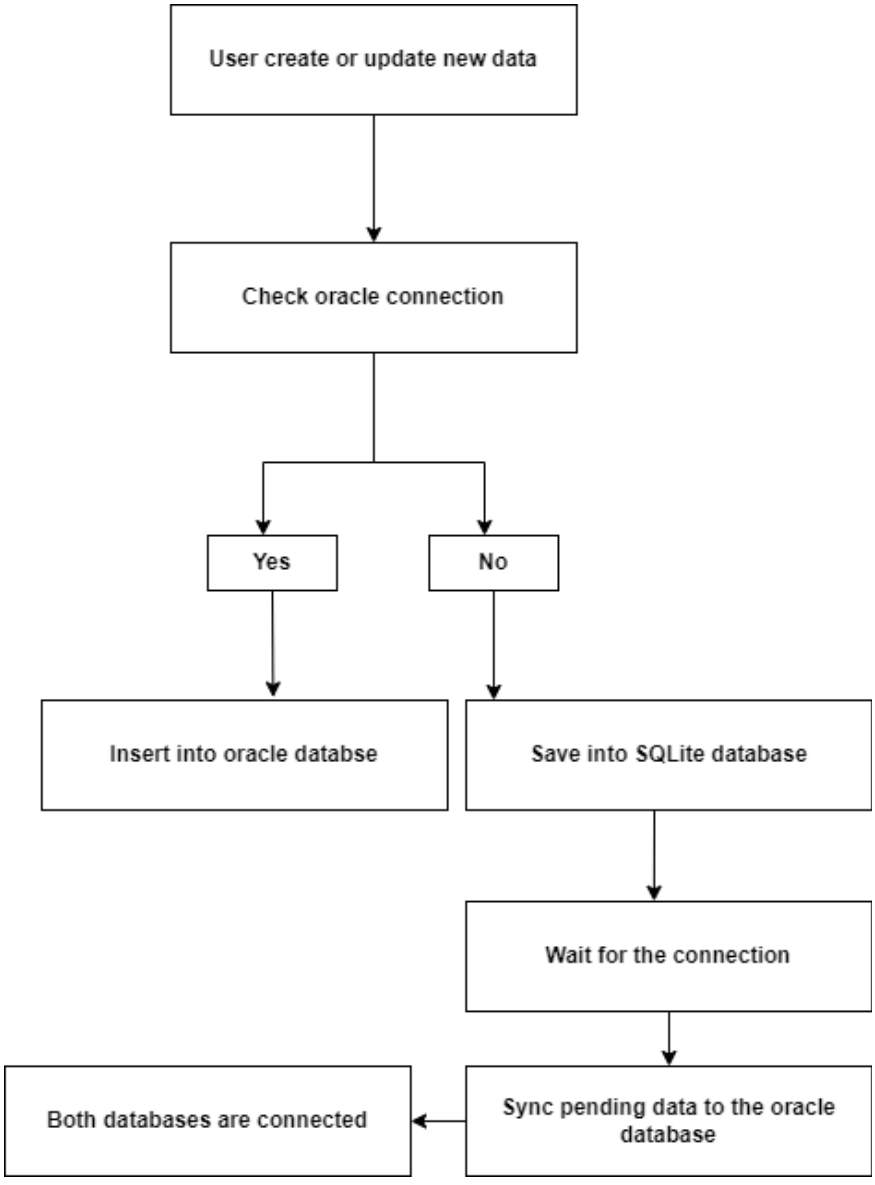
## 6.1 Concept

Synchronization is one of the main concepts in this personal finance management system , it supports smooth and consistent data flow between local SQLite database and central Oracle database. This method is crucial for data integrity and because of this users can work online and offline without any data duplication and any errors.SQLite also known as local database store and data when user is offline .When user perform insert or update operation offline data first inserted into the SQLite database , Each operation hold a isSynced flag maintaining the status which indicate whether it is uploaded to oracle or not.Then all the record data that inserted or updated are stored in a JSON file.The system execute the Sync manager script every 10 minutes when the connection establish gain or sever restarted . This sync manager script checks whether there is a pending JSON file or unsynced data . this prevents data duplication and data redundancy and multiple stance attempts.Once the oracle database connection is established each record run the relevant sql statement and data transferred by using parameter binding to prevent sql injections.Once the oracle database connection successful a message is showing and that pending JSON will be removed from the pending file. If any kind of error occurs, that pending

file does not remove the data until the oracle database connection is successfully established. This supports preventing data loss and smooth data transfer between SQLite and Oracle database.

## 6.2 Process Flow

Process flow of the synchronization is supported to protect user data successfully.When users run a specific action like insertion or update in any expenses income or in accounts records are stored in a pending sync file . Once the sync cycle starts then the sync manager updates the data in the oracle database.



# Chapter 7. Financial Reports and Analysis

## 7.1 Transaction Summary Report

A detailed summary of all transactions done by the user.

**PL/SQL programme**

```
CREATE OR REPLACE PROCEDURE generate_transaction_summary (
    p_user_id IN NUMBER
)
AS
    v_total_income  NUMBER := 0;
    v_total_expense NUMBER := 0;
    v_net_balance   NUMBER := 0;
BEGIN
    SELECT
    NVL(SUM(CASE WHEN transactionType = 'Income' THEN amount ELSE 0 END), 0),
    NVL(SUM(CASE WHEN transactionType = 'Expense' THEN amount ELSE 0 END), 0)
    INTO v_total_income, v_total_expense
    FROM Transaction_Info
    WHERE user_id = p_user_id
      AND isDeleted = 'N';

    v_net_balance := v_total_income - v_total_expense;

    DBMS_OUTPUT.PUT_LINE('===== TRANSACTION SUMMARY REPORT =====');
    DBMS_OUTPUT.PUT_LINE('User ID: ' || p_user_id);
    DBMS_OUTPUT.PUT_LINE('-----------------------------------------');
    DBMS_OUTPUT.PUT_LINE('Total Income   : ' || v_total_income);
    DBMS_OUTPUT.PUT_LINE('Total Expense  : ' || v_total_expense);
    DBMS_OUTPUT.PUT_LINE('Net Balance    : ' || v_net_balance);
    DBMS_OUTPUT.PUT_LINE('-----------------------------------------');
END;
/
```

## 7.2 Transaction Summary of Budget

This report shows money inflows and outflows by budget category.

PL/SQL programme

```
CREATE OR REPLACE PROCEDURE generate_budget_transaction_summary (
    p_user_id IN NUMBER
)AS
BEGIN
    DBMS_OUTPUT.PUT_LINE('===== TRANSACTION SUMMARY BY BUDGET =====');
    DBMS_OUTPUT.PUT_LINE('User ID: ' || p_user_id);
    DBMS_OUTPUT.PUT_LINE('----------------------------------------');
    DBMS_OUTPUT.PUT_LINE('Category | Income | Expense | Net Result');
    FOR r IN (
        SELECT  c.categoryName,
NVL(SUM(CASE WHEN t.transactionType = 'Income' THEN t.amount ELSE 0 END), 0) AS total_income,
NVL(SUM(CASE WHEN t.transactionType = 'Expense' THEN t.amount ELSE 0 END), 0) AS total_expense,
NVL(SUM(CASE WHEN t.transactionType = 'Income' THEN t.amount ELSE 0 END), 0) -
NVL(SUM(CASE WHEN t.transactionType = 'Expense' THEN t.amount ELSE 0 END), 0) AS net_amount
        FROM
            Transaction_Info t
            JOIN Transaction_Category c ON t.category_id = c.category_id
        WHERE
            t.user_id = p_user_id
            AND t.isDeleted = 'N'
        GROUP BY
            c.categoryName
        ORDER BY
            c.categoryName
    )
    LOOP
        DBMS_OUTPUT.PUT_LINE(
            RPAD(r.categoryName, 20) || ' | ' ||
            LPAD(r.total_income, 10) || ' | ' ||
            LPAD(r.total_expense, 10) || ' | ' ||
            LPAD(r.net_amount, 10)
        );
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('----------------------------------------');
END;
```

## 7.3 Saving Goals Report

This report tracks users' saving goals. This report helps them to evaluate whether they achieved it or not .This report displays User name, details about goals  set by user, progress presentation and deadline

PL/SQL programme

```
CREATE OR REPLACE PROCEDURE generate_saving_goals_report (
    p_user_id IN NUMBER
)
AS
BEGIN
    DBMS_OUTPUT.PUT_LINE('===== SAVING GOALS REPORT =====');
    DBMS_OUTPUT.PUT_LINE('User ID: ' || p_user_id);
    DBMS_OUTPUT.PUT_LINE('----------------------------------------');
    DBMS_OUTPUT.PUT_LINE('Goal Name | Target | Current | Progress | Deadline');

    FOR r IN (
SELECT
  goalName,
  targetAmount,
 currentAmount,
  ROUND((currentAmount / targetAmount) * 100, 2) AS progress_percent,
   TO_CHAR(endDate, 'YYYY-MM-DD') AS deadline
    FROM Saving_Goal
    WHERE user_id = p_user_id
        ORDER BY endDate
    )
    LOOP
        DBMS_OUTPUT.PUT_LINE(
            RPAD(r.goalName, 15) || ' | ' ||
            LPAD(r.targetAmount, 10) || ' | ' ||
            LPAD(r.currentAmount, 10) || ' | ' ||
            LPAD(r.progress_percent || '%', 8) || ' | ' ||
            r.deadline
        );
    END LOOP;

    DBMS_OUTPUT.PUT_LINE('----------------------------------------');
END;
/
```

# Chapter 8. Data Security and Privacy

## 8.1 SQLite Security

SQLite is the local database that are embedded to the local device and it is focused on device level data protection and control.All the data such as user credentials , personal information and transactions are secular stored with using Hash passwords and bcrypt and parameterized sql queries to prevent sql injections.When it’s comes to the sQLite local database it’s a local database so it ensures that app has read and write access to its files.

## 8.2 Oracle Security

Oracle is the centralized database that can implement advanced data protection ,access control and auditing .Role based access ensures that only authorized users can change and modify the data.Oracle's. Also, Oracle uses access control and Virtual Private Database policies to safely isolate user data in multiuser scenarios. All database operations are audited and logged on a regular basis to ensure accountability and uncover anomalies. Backup and recovery are controlled by Oracle Recovery Manager, which provides data integrity in the event of data corruption or loss, making Oracle the secure in the synchronization architecture.

# Chapter 9. Backup and Recovery

## 9.1 SQLite

In SQLite a local database is created to ensure that if any failure happens all the data that is stored are recoverable and consistent .SQLite is a filebase local database and all the data stored in a .db file. In SQLite, we have used file based backup , after a specific period of time SQLite database file will be copied and stored in cloud storage.

Also before running synchronization a copy of a current file will save to the cloud database to prevent data loss .As for the recovery method if a SQLite database file got corrupted , latest updated database file will replace .In synchronization JSON based file will use and because of that any  unsync data will not lose.

### 9.2 Oracle Database

In the oracle database we have used recovery manager -RMAN , RMAN command line ensures transaction logs are backed up. Also the recovery manager can restore all data using command.when synchronization happens oracle validate with unique keys and timestamps confirming no data duplication.

# Chapter 10. Migration Plan

Melony- Personal finance management system uses SQLite as a local database and oracle as a centralized database, the migration plan outlines the structure that is used to transfer data from the local database to the centralized database ensuring data integrity , security and no duplicate data entries. The main goal of the migration plan is to move all locally stored data into the Oracle database once the server connection is available.

There are three main phases in this migration: the first phase is data extraction , second phase is data transformation phase and third phase is loading phase. In the data extraction phase are there any unsynchronized records identified by the pending sync list , then check these data do not contain any duplicated data or missing values. In the data transformation phase data types of the record will be changed to oracle's schema such as REAL to NUMBER and TEXT to VARCHAR2 like that.Loading phase synchronization manager loads the data into the centralized database using SQL transaction.

In our personal finance management system we have used a server-side technique for important data such as user registration and email verification. That means when creating a user it first creates in an oracle centralized database unique constraints such as user_id and email validated by the server , once the server verified record has been copied to the local SQLite database for offline use. This method supports to avoid duplicate entries and data conflicts , for an example Let's say a user want to create an account in this system then that entered data first go to the oracle database which act as a primary validation authority , instead of store in local SQLite database , Oracle check that email is registered one or not if no then create the user at the same time oracle creates a unique global user_id for this user. If the oracle insertion is successful then system automatically synchronise the record with SQLite ,In case oracle server is disconnected then all user record goes to the pending sync file as a jason file , once oracle is connected then all the data will push to the oracle server and do the verification and again store in the SQLite.

## References

Elmasi, R., & Navathe, S. B. (2020). *Fundamentals of Database Systems* (7th ed.). Pearson Education.

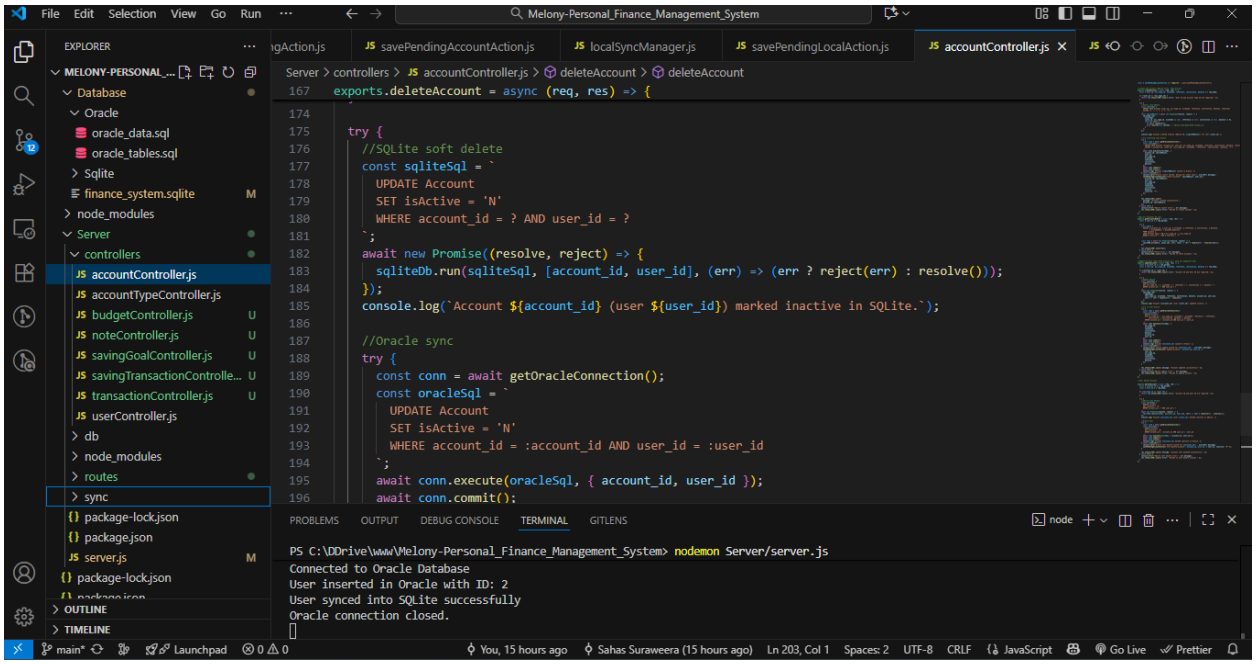
Hipp, D. R. (2024). *SQLite Documentation.* SQLite.org. https://www.sqlite.org/doc.html

Oracle Corporation. (2024). *Oracle® Database SQL Language Reference, 21c.* Oracle Help Center. https://docs.oracle.com/en/database/oracle/oracle-database/21/sqlr/

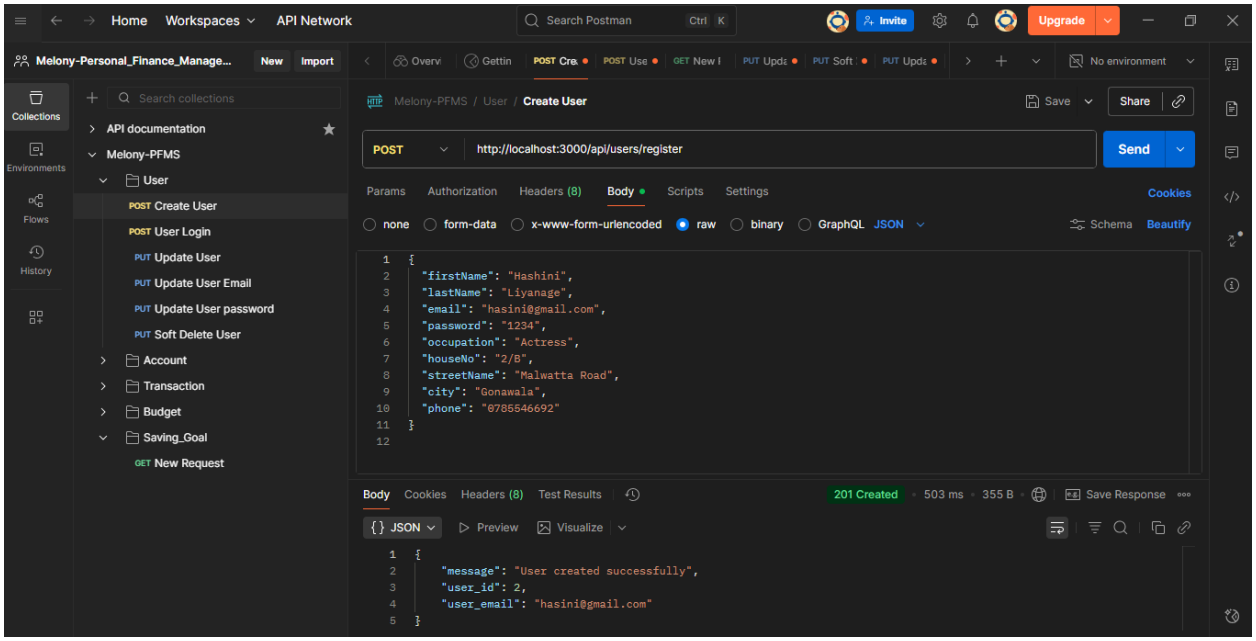
Node.js Foundation. (2024). *Node.js v20.0.0 Documentation.* https://nodejs.org/en/docs

# Appendacies

## Backend File Structure with node.js




## Reset API Check with POSTMAN



# GitHub Link

**https://github.com/SahasSuraweera/Melony-Personal_Finance_Management_System.git**

**-End of the Report-**