Paisa

Personal Budget Tracker

Subhash Gottumukkala (102203609)
Pranav Chawla (102203627)
Alok Sinha (102203580)
Jyotish Yadav (102203632)

Subgroup: 2CO14

Submitted to:
Ms. Nishu Mehta

Table of Contents

Introduction

Effective financial management is essential for students in today's economy. Paisa is a robust database-driven program created to offer complete package for money management and tracking spending effectively. This system is designed for users who are looking for a dependable and user-friendly solution to manage their day-to-day expenditure and ensure they stay on track with their financial objectives.

The main goal of the Paisa is to enable users to establish, oversee, and track different expenses, classify spending, and produce informative reports to enhance financial decision-making. The application is constructed with Oracle's PL/SQL, utilizing advanced relational database management techniques including stored procedures, functions, triggers, and cursors to improve data integrity and automate essential tasks.

Key Features:

1. **User Management:** The system provides functionalities for managing user profiles, including registration, login, and user data maintenance. This allows for personalized budget tracking tailored to individual user preferences and requirements.
2. **Budget Creation and Management:** Users can create multiple budgets based on different needs or time frames. Each budget includes details such as total budget amount, start and end dates, and a description. The system also allows for modifications and adjustments to budgets, ensuring flexibility in response to changing financial situations.
3. **Expense Tracking:** At the core of the system is the ability to record and categorize expenses against specific budgets. This feature includes the insertion of expense details such as amount, date, category, and associated budget. Users can input recurring expenses as well as one-time expenditures.
4. **Reporting and Insights:** The system offers comprehensive reporting tools that generate real-time insights into budget utilization and financial health. Users can view summaries and detailed reports, which help in understanding spending patterns and identifying potential savings opportunities.
5. **Procedures and Functions:** To facilitate complex queries and transactions, the system employs PL/SQL procedures and functions. These elements encapsulate business logic to handle tasks like calculating total expenses for a given budget, checking budget balances, and more, thereby promoting code reusability and maintenance.
6. **Triggers:** Paisa uses database triggers to enhance data quality and consistency. These triggers automatically update critical information, such as the last modified date of budgets when expenses are added or modified, and enforce business rules like preventing expenses that exceed the budget limit.
7. **Cursors:** The system utilizes PL/SQL cursors for efficiently managing data retrieval operations. Cursors are used to iterate over expense records, aggregate spending by category, and more, providing flexible and powerful data manipulation capabilities.

8. **Data Integrity and Security:** The architecture of the system is designed with a focus on security and data integrity. Features such as constraints, indexes, and normalized data models ensure that the data is not only accurate but also securely stored and processed.

Requirements:

The application will require a database management system like MySQL, framework like Flutter or React Native and tools like Android Studio or VSCode. The system will include modules for user onboarding, expense tracker, reporting.

ER Diagram

ER to Tables

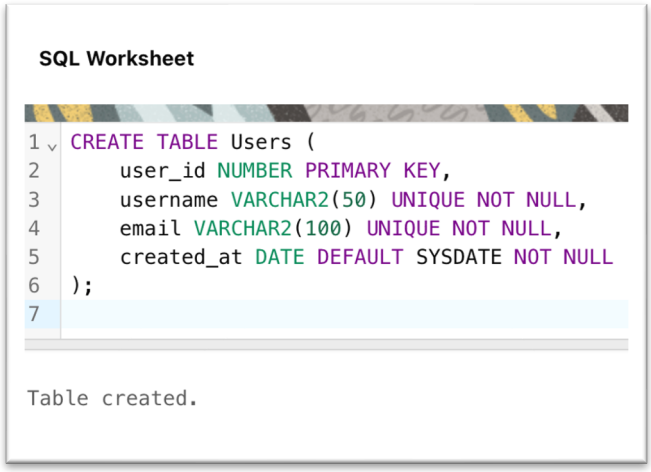
ENTITY 1	RELATIONSHIP	ENTITY 2	CARDINALITY	PARTICIPATION
Users	have	Budgets	1:N	Total-Partial
Budgets	include	Expenses	1:N	Total-Partial
Users	manage	Categories	M:N	Partial-Partial
Categories	categorize	Expenses	1:N	Total-Total

Normalization

Entities	Functional Dependencies	Normal Form
Users	user_id, username, email	3NF
Budgets	budget_id, user_id, total_budget, start_date, end_date, last_modified	3NF
Expenses	expense_id, budget_id, category_id, amount, description, date_incurred	3NF
Categories	category_id, category_name	3NF

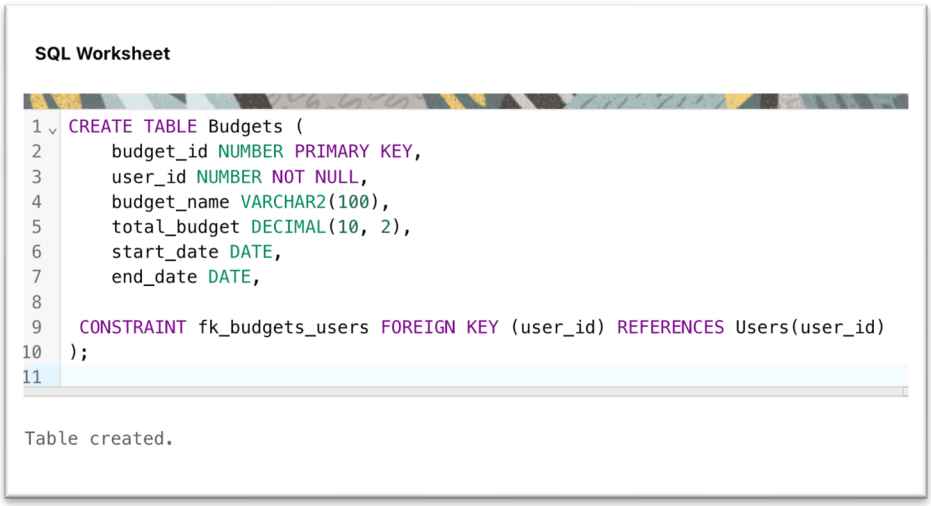
SQL

```
CREATE TABLE Users (
   user_id NUMBER PRIMARY KEY,
   username VARCHAR2(50) UNIQUE NOT NULL,
   email VARCHAR2(100) UNIQUE NOT NULL,
   created_at DATE DEFAULT SYSDATE NOT NULL
);
```

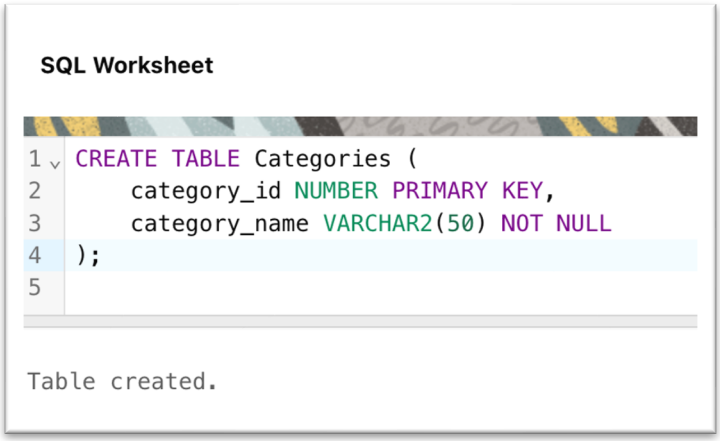



```
CREATE TABLE Budgets (
   budget_id NUMBER PRIMARY KEY,
   user_id NUMBER NOT NULL,
   budget_name VARCHAR2(100),
   total_budget DECIMAL(10, 2),
   start_date DATE,
   end_date DATE,

 CONSTRAINT fk_budgets_users FOREIGN KEY (user_id) REFERENCES Users(user_id)
);
```




```
CREATE TABLE Categories (
  category_id NUMBER PRIMARY KEY,
  category_name VARCHAR2(50) NOT NULL
);
```




```
CREATE TABLE Expenses (
  expense_id NUMBER PRIMARY KEY,
  budget_id NUMBER NOT NULL,
  category_id NUMBER NOT NULL,
  amount DECIMAL(10, 2),
  expense_date DATE DEFAULT SYSDATE NOT NULL,
  description VARCHAR2(255),
  CONSTRAINT fk_expenses_budgets FOREIGN KEY (budget_id) REFERENCES
Budgets(budget_id),
  CONSTRAINT fk_expenses_categories FOREIGN KEY (category_id) REFERENCES
Categories(category_id)
);
```

```
SQL> CREATE TABLE Expenses (
  2  expense_id NUMBER PRIMARY KEY,
  3  budget_id NUMBER NOT NULL,
  4  category_id NUMBER NOT NULL,
  5  amount DECIMAL(10, 2),
  6  expense_date DATE DEFAULT SYSDATE NOT NULL,
  7  description VARCHAR2(255),
  8  FOREIGN KEY (budget_id) REFERENCES Budgets(budget_id),
  9  FOREIGN KEY (category_id) REFERENCES Categories(category_id)
 10  );

Table created.
```

PL/SQL

Stored Procedures and Cursors:

1. Adding Users

```
SQL> CREATE OR REPLACE PROCEDURE Add_User (
  2  p_username IN Users.username%TYPE,
  3  p_email IN Users.email%TYPE
  4  )
  5  AS
  6  l_user_id Users.user_id%TYPE;
  7  BEGIN
  8  SELECT user_id_seq.NEXTVAL INTO l_user_id FROM DUAL;
  9  INSERT INTO Users (user_id, username, email)
 10  VALUES (l_user_id, p_username, p_email);
 11  COMMIT;
 12
 13  DBMS_OUTPUT.PUT_LINE('New user added with ID: ' || l_user_id);
 14  EXCEPTION
 15  WHEN OTHERS THEN
 16  DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
 17  END;
 18  /
```

2. Creating Budgets

```
SQL> CREATE OR REPLACE PROCEDURE Create_Budget (
  2  p_user_id IN Budgets.user_id%TYPE,
  3  p_budget_name IN Budgets.budget_name%TYPE,
  4  p_total_budget IN Budgets.total_budget%TYPE,
  5  p_start_date IN Budgets.start_date%TYPE,
  6  p_end_date IN Budgets.end_date%TYPE
  7  )
  8  AS
  9  l_budget_id Budgets.budget_id%TYPE;
 10  BEGIN
 11  SELECT budget_id_seq.NEXTVAL INTO l_budget_id FROM DUAL;
 12  INSERT INTO Budgets (budget_id, user_id, budget_name, total_budget, start_date, end_date)
 13  VALUES (l_budget_id, p_user_id, p_budget_name, p_total_budget, p_start_date, p_end_date);
 14  COMMIT;
 15  DBMS_OUTPUT.PUT_LINE('Budget created successfully with ID: ' || l_budget_id);
 16  EXCEPTION
 17  WHEN OTHERS THEN
 18  DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
 19  END;
 20  /
```

3. Recording an Expense/Transaction

```
SQL> CREATE OR REPLACE PROCEDURE Record_Expense (
  2  p_budget_id IN Expenses.budget_id%TYPE,
  3  p_category_id IN Expenses.category_id%TYPE,
  4  p_amount IN Expenses.amount%TYPE,
  5  p_description IN Expenses.description%TYPE
  6  )
  7  AS
  8  BEGIN
  9  INSERT INTO Expenses (expense_id, budget_id, category_id, amount, description)
 10  VALUES (expense_id_seq.NEXTVAL, p_budget_id, p_category_id, p_amount, p_description);
 11  COMMIT;
 12  DBMS_OUTPUT.PUT_LINE('Expense recorded successfully.');
 13  EXCEPTION
 14  WHEN OTHERS THEN
 15  DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
 16  END;
 17  /
```

4. Cursor for Displaying Expenses

```
SQL> CREATE OR REPLACE PROCEDURE List_Expenses_By_Budget (
  2  p_budget_id IN Expenses.budget_id%TYPE
  3  ) AS
  4  CURSOR c_expenses IS
  5  SELECT e.amount, e.description, e.expense_date, c.category_name
  6  FROM Expenses e
  7  JOIN Categories c ON e.category_id = c.category_id
  8  WHERE e.budget_id = p_budget_id
  9  ORDER BY e.expense_date DESC;
 10  l_amount Expenses.amount%TYPE;
 11  l_description Expenses.description%TYPE;
 12  l_expense_date Expenses.expense_date%TYPE;
 13  l_category_name Categories.category_name%TYPE;
 14  BEGIN
 15  OPEN c_expenses;
 16  LOOP
 17  FETCH c_expenses INTO l_amount, l_description, l_expense_date, l_category_name;
 18  EXIT WHEN c_expenses%NOTFOUND;
 19  DBMS_OUTPUT.PUT_LINE('Amount: ' || l_amount || ' Description: ' || l_description || Date: ' || l_expe
nse_date || ' Category: ' || l_category_name);
 20  END LOOP;
 21  CLOSE c_expenses;
 22  EXCEPTION
 23  WHEN OTHERS THEN
 24  DBMS_OUTPUT.PUT_LINE('Error retrieving expenses: ' || SQLERRM);
 25  CLOSE c_expenses;
 26  END;
 27  /
```

Stored Functions:

```
SQL> CREATE OR REPLACE FUNCTION Total_Expenses (
  2  p_budget_id IN Expenses.budget_id%TYPE
  3  ) RETURN DECIMAL
  4  AS
  5  l_total DECIMAL(10, 2) := 0;
  6  BEGIN
  7  SELECT SUM(amount) INTO l_total FROM Expenses WHERE budget_id = p_budget_id;
  8  RETURN l_total;
  9  EXCEPTION
 10  WHEN NO_DATA_FOUND THEN
 11  RETURN 0;
 12  WHEN OTHERS THEN
 13  RAISE_APPLICATION_ERROR(-20001, 'An error occurred calculating total expenses: ' || SQLERRM);
 14  END;
 15  /

Function created.
```

Triggers:

```
SQL> CREATE OR REPLACE TRIGGER Update_Budget_Last_Modified
  2  AFTER INSERT OR UPDATE ON Expenses
  3  FOR EACH ROW
  4  BEGIN
  5  UPDATE Budgets
  6  SET last_modified = SYSDATE
  7  WHERE budget_id = :NEW.budget_id;
  8  END;
  9  /
```

Conclusion: