DATABASE MANAGEMENT SYSTEM

Digital Wallet Tracker

Contributed By:

NAME	University
M.vaishnavi	SVKM's NMIMS,Hyd
V.Vaishnavi	SVKM's NMIMS,Hyd
B.Vaishnavi	SVKM's NMIMS,Hyd
M.sowmya	SVKM's NMIMS,Hyd

INDEX

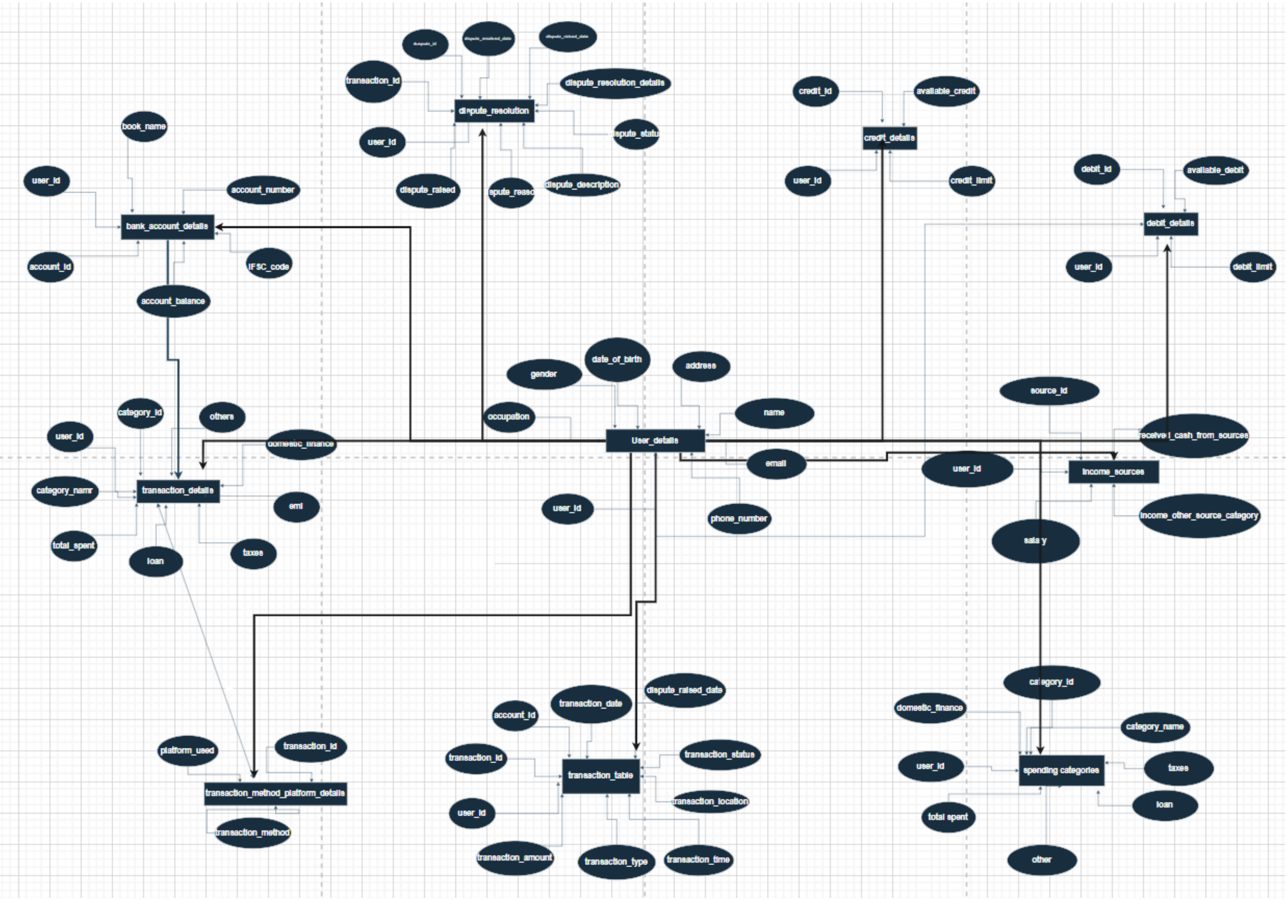
DIGITAL WALLET TRACKER

I. ABSTRACT:

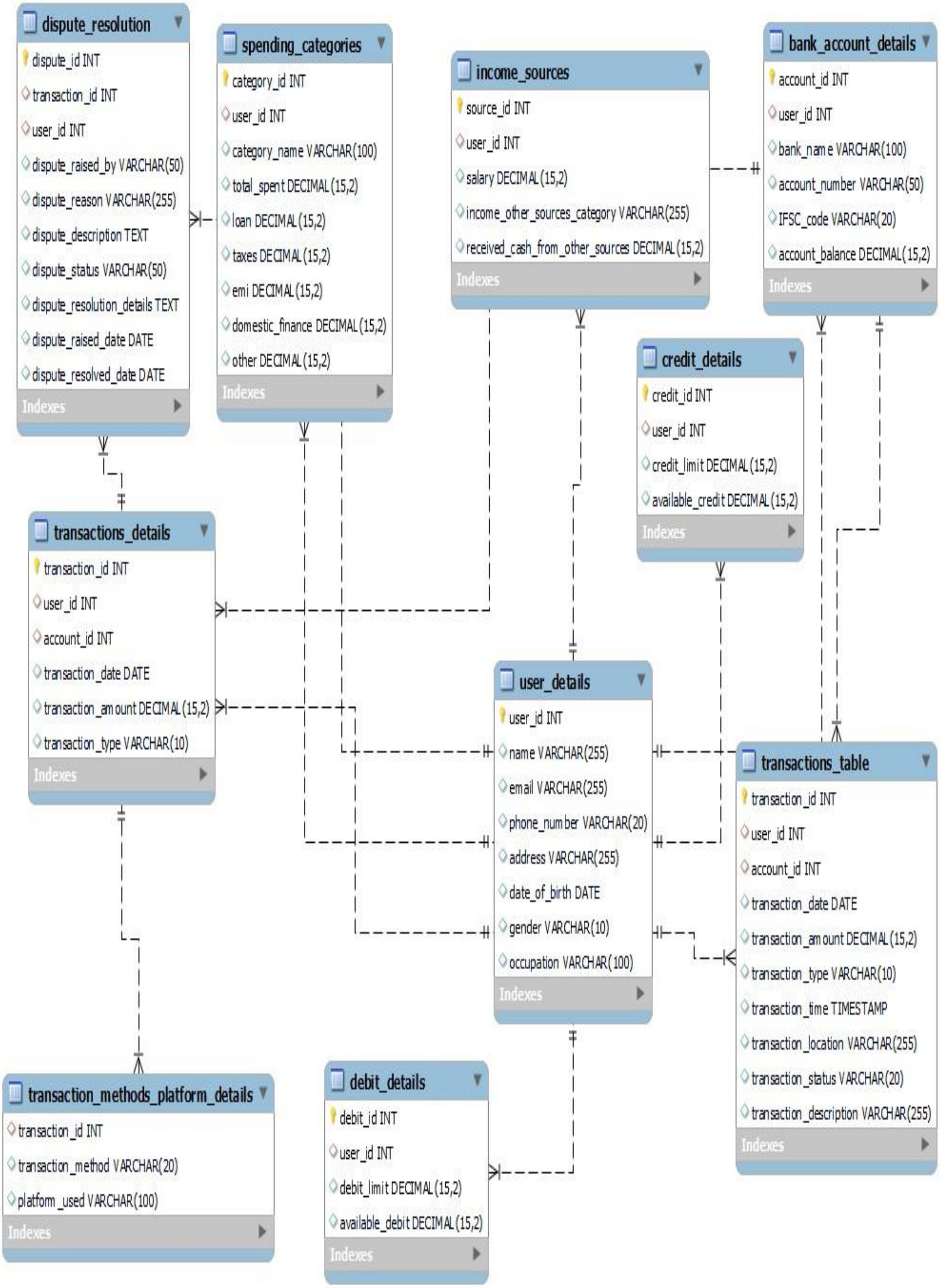
This report presents a comprehensive overview of the database schema for the Digital Wallet Tracker, a digital platform designed to monitor and manage users' financial transactions and accounts. Understanding the database schema is crucial for developers, analysts, and stakeholders involved in building and maintaining the Digital Wallet Tracker platform. The report provides detailed insights into the structure and relationships of various tables within the database, facilitating efficient data management and analysis for enhancing user experience and security.

II. INTRODUCTION:

This report presents a detailed overview of the database schema for the Unified Payments Interface (UPI) Payments System. The UPI Payments System is a digital payments infrastructure widely used in India for facilitating instant money transfers between bank accounts. Understanding the database schema is crucial for developers, analysts, and stakeholders involved in building and managing the UPI Payments System platform. This report provides insights into the structure and relationships of various tables within the database, aiding in the efficient management of user data, transactions, and dispute resolution.



III. ENTITY-RELATIONSHIP DIAGRAM (ERD):



IV. TABLE DESCRIPTION:

Detailed descriptions of each table within the database schema are provided in this section. It includes the purpose, primary key, foreign key relationships, and attributes of tables such as:

1. **User_details Table:**

Purpose: Stores basic information about users registered in the UPI Payments system.

Attributes:

`user_id` (Primary Key): Unique identifier for each user.
`name`: User's name.
`email`: User's email address.
`phone_number`: User's phone number.
`address`: User's address.
`date_of_birth`: User's date of birth.
`gender`: User's gender.
`occupation`: User's occupation.

2. **Bank_account_details Table:**

Purpose: Manages details of bank accounts associated with users.

Attributes:

`account_id` (Primary Key): Unique identifier for each bank account.
`user_id` (Foreign Key): References `user_details.user_id` to link the bank account with a user.
`bank_name`: Name of the bank.
`account_number`: Bank account number.
`IFSC_code`: IFSC (Indian Financial System Code) of the bank branch.
`account_balance`: Current balance in the bank account.

3. **transactions_details Table:**

Purpose: Records details of financial transactions conducted by users.

Attributes:

`transaction_id` (Primary Key): Unique identifier for each transaction.
`user_id` (Foreign Key): References `user_details.user_id` to identify the user associated with the transaction.
`account_id` (Foreign Key): References `bank_account_details.account_id` to link the transaction with a specific bank account.
`transaction_date`: Date of the transaction.
`transaction_amount`: Amount of the transaction.
`transaction_type`: Type of transaction (e.g., credit, debit).

4. **Credit_details Table:**

Purpose: Stores creditrelated information for users.

Attributes:

credit_id (Primary Key): Unique identifier for each credit entry.
user_id (Foreign Key): References `user_details.user_id` to link credit details with a user.
credit_limit: Maximum credit limit for the user.
available_credit: Available credit amount for the user.

5. **Debit_details Table:**

Purpose: Manages debitrelated information for users.

Attributes:

`debit_id` (Primary Key): Unique identifier for each debit entry.
`user_id` (Foreign Key): References `user_details.user_id` to link debit details with a user.
`debit_limit`: Maximum debit limit for the user.
`available_debit`: Available debit amount for the user.

6. Spending_categories Table:

Purpose: Tracks spending categories and related financial details for users.

Attributes:

`category_id` (Primary Key): Unique identifier for each spending category.
`user_id` (Foreign Key): References `user_details.user_id` to associate spending categories with a user.
`category_name`: Name of the spending category.
`total_spent`: Total amount spent in the category.
`loan`: Amount spent on loans.
`taxes`: Amount spent on taxes.
`emi`: Amount spent on EMIs.
`domestic_finance`: Amount spent on domestic financial activities.
`other`: Amount spent on other categories.

7. Income_sources Table:

Purpose: Records sources of income for users.

Attributes:

`source_id` (Primary Key): Unique identifier for each income source.
`user_id` (Foreign Key): References `user_details.user_id` to link income sources with a user.
`salary`: Monthly salary amount.
`income_other_sources_category`: Description of other income sources.
`received_cash_from_other_sources`: Amount received from other income sources.

8. Transactions_table Table:

Purpose: Stores detailed records of transactions, including additional information.

Attributes:

`transaction_id` (Primary Key): Unique identifier for each transaction.
`user_id` (Foreign Key): References `user_details.user_id` to link the transaction with a user.
`account_id` (Foreign Key): References `bank_account_details.account_id` to specify the bank account involved in the transaction.
`transaction_date`: Date of the transaction.
`transaction_amount`: Amount of the transaction.
`transaction_type`: Type of transaction (e.g., credit, debit).
`transaction_time`: Timestamp of the transaction.
`transaction_location`: Location of the transaction.
`transaction_status`: Status of the transaction (e.g., pending, completed).
`transaction_description`: Description or notes related to the transaction.

9. Dispute_resolution Table:

Purpose: Manages disputes raised by users related to transactions.

Attributes:

`dispute_id` (Primary Key): Unique identifier for each dispute.

`transaction_id` (Foreign Key): References `transactions_details.transaction_id` to link the dispute with a specific transaction.

`user_id` (Foreign Key): References `user_details.user_id` to identify the user raising the dispute.

`dispute_raised_by`: Indicates who raised the dispute (user or merchant).

`dispute_reason`: Reason for raising the dispute.

`dispute_description`: Detailed description of the dispute.

`dispute_status`: Current status of the dispute (e.g., open, resolved).

`dispute_resolution_details`: Details of how the dispute was resolved.

`dispute_raised_date`: Date when the dispute was raised.

`dispute_resolved_date`: Date when the dispute was resolved.

10. Transaction_methods_platform_details Table:

Purpose: Records the method and platform used for each transaction.

Attributes:

`transaction_id` (Foreign Key): References `transactions_details.transaction_id` to link the transaction with a specific method/platform.

`transaction_method`: Method used for the transaction (e.g., UPI, credit card).

`platform_used`: Platform used for the transaction (e.g., mobile app, website).

This comprehensive schema enables effective management and analysis of user-related data, financial transactions, credit/debit details, income sources, and dispute resolutions within the UPI Payments system. Each table and its relationships serve specific purposes to support various functionalities of the payment platform.

Relationships Between Tables

The UPI_Payments database uses various relationships between tables to model real-world financial transactions. Here's a breakdown of the key relationships:

One-to-Many:

- **user_details - bank_account_details:** A single user can have multiple bank accounts.
- **user_details - transactions_details:** A user can have a history of many transactions (debit, credit, etc.).
- **user_details - spending_categories:** A user can categorize their spending in various ways.
- **user_details - income_sources:** A user can have income from different sources.
- **user_details - transactions_table:** A user can initiate many transactions (represented by transactions_table).
- **transactions_table - dispute_resolution:** A single transaction can be linked to at most one dispute. (Optional: One-to-One)
- **user_details - credit_details (Optional):** A user can have one credit detail, assuming a single credit account. This can be Many-to-One if users can have multiple credit accounts.
- **user_details - debit_details (Optional):** A user can have one debit detail, assuming a single debit account. This can be Many-to-One if users can have multiple debit accounts.

Many-to-Many:

- **transactions_details - transaction_methods_platform_details:** A single transaction can involve multiple methods and platforms (e.g., UPI transfer using PhonePe app).

Explanation:

These relationships reflect real-life scenarios. For instance, a user (in user_details) can have several bank accounts (in bank_account_details), and each account can have numerous transactions (in transactions_details). Similarly, users can categorize their spending and have income from various sources.

The many-to-many relationship between transactions_details and transaction_methods_platform_details captures the complexity of modern transactions, where a single transaction can utilize different methods (UPI, IMPS, NEFT) and platforms (PhonePe, GPay, BHIM).

NORMALIZATION OF OUR DATABASE:

1NF (First Normal Form):

A table is in 1NF if it satisfies the following condition:

- **No Repeating Groups:** There are no cells containing multiple values. Each cell should hold a single atomic value (a basic unit of data that cannot be further broken down).

How it was satisfied initially:

- In the original schema, each cell contained a single value, like a user's name, email, or account balance. There were no repeating groups within any table.

2NF (Second Normal Form):

A table is in 2NF if it adheres to 1NF and the following additional condition:

- **No Partial Dependencies:** All non-key attributes (attributes not part of the primary key) are fully functionally dependent on the entire primary key. There should be no partial dependencies where a non-key attribute depends on only a part of the primary key.

How it was satisfied initially:

- In the original schema, any non-key attribute (e.g., transaction amount, transaction type) in a table was dependent on the entire primary key of that table (e.g., for transactions_details, the primary key was a combination of user_id, account_id, and transaction_date). This ensured no partial dependencies existed.

3NF (Third Normal Form):

A table is in 3NF if it adheres to 1NF and 2NF, and also satisfies the following condition:

- **No Transitive Dependencies:** There are no transitive dependencies, meaning no non-key attribute is dependent on another non-key attribute through the primary key. In simpler terms, if attribute A determines attribute B, and B determines attribute C, then A must directly determine C without relying on B.

How it was satisfied initially:

- In the initial schema, there weren't any transitive dependencies. For example, transaction_type in transactions_details wasn't dependent on another non-key attribute like account_balance (which could potentially depend on transaction_type). It directly depended on the primary key (user_id, account_id, and transaction_date).

Change to BCNF (Boyce-Codd Normal Form):

BCNF is a stricter form of 3NF, adding the following condition:

- **Determinant is a Candidate Key:** For every functional dependency (FD) in a table, the determinant (the set of attributes determining the value of another attribute) must be a candidate key (a minimal set of attributes uniquely identifying a record).

What we did to achieve BCNF:

1. **Merged** transaction_table **and** history_of_transactions **into** transactions_details**:** These tables had overlapping information. Combining them eliminated redundancy and ensured the determinant for newly added attributes like transaction_time, location, status, and description directly depended on the combined primary key (user_id, account_id, transaction_date).
2. **Introduced** dispute_resolution **table:** This table stores dispute details related to transactions. Separating this information from core transactions prevents redundancy and keeps the determinant for attributes in transactions_details focused on the primary key for core transaction information.

How we can say it's in BCNF:

By addressing the potential transitive dependencies and ensuring determinants are candidate keys, the modified schema avoids anomalies that could arise in 3NF. Here's why it satisfies BCNF:

- transactions_details:
  - Determinant for transaction_time, location, status, and description is the combination of user_id, account_id, and transaction_date (all part of the candidate key).
  - Determinant for user_id, account_id, and any other attribute depends on the full candidate key.
- Similar logic applies to bank_account_details, user_details, credit_details, debit_details, spending_categories, income_sources, and transaction_methods_platform_details.

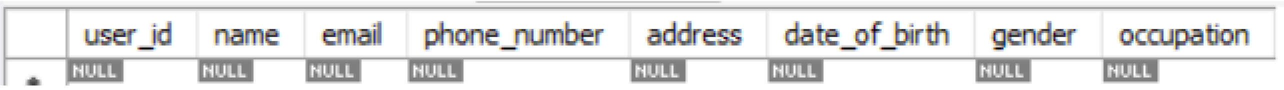
Therefore, the revised schema adheres to all the requirements of BCNF.

V. SQL SCRIPT:

The SQL script contains the complete code necessary for creating the database and its associated tables. It serves as a practical guide for implementing the database schema in the Digital Wallet Tracker platform.

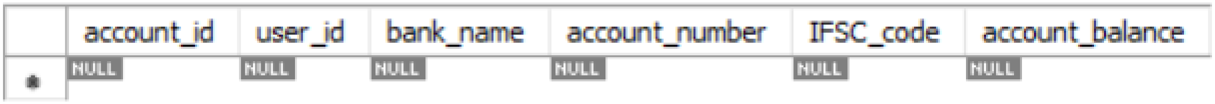
--CODE

```
create database UPI_Payments;
use UPI_Payments;
-- 1.Create user_details table
CREATE TABLE user_details (
   user_id INT PRIMARY KEY,
   name VARCHAR(255),
   email VARCHAR(255),
   phone_number VARCHAR(20),
   address VARCHAR(255),
   date_of_birth DATE,
   gender VARCHAR(10),
   occupation VARCHAR(100)
);
```



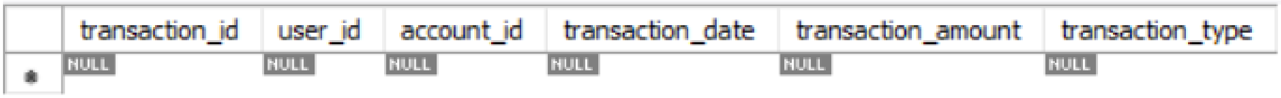
	user_id	name	email	phone_number	address	date_of_birth	gender	occupation
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

```
-- 2.Create bank_account_details table
CREATE TABLE bank_account_details (
   account_id INT PRIMARY KEY,
   user_id INT,
   bank_name VARCHAR(100),
   account_number VARCHAR(50),
   IFSC_code VARCHAR(20),
   account_balance DECIMAL(15, 2),
   FOREIGN KEY (user_id) REFERENCES user_details(user_id)
);
```



	account_id	user_id	bank_name	account_number	IFSC_code	account_balance
*	NULL	NULL	NULL	NULL	NULL	NULL

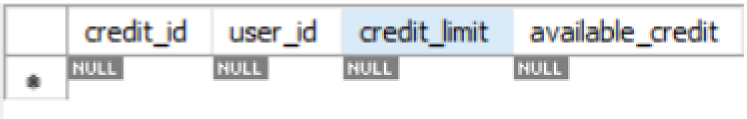
```
-- 3.Create transactions_details table
CREATE TABLE transactions_details (
   transaction_id INT PRIMARY KEY,
   user_id INT,
   account_id INT,
   transaction_date DATE,
   transaction_amount DECIMAL(15, 2),
   transaction_type VARCHAR(10),
   FOREIGN KEY (user_id) REFERENCES user_details(user_id),
   FOREIGN KEY (account_id) REFERENCES bank_account_details(account_id)
);
```



	transaction_id	user_id	account_id	transaction_date	transaction_amount	transaction_type
●	NULL	NULL	NULL	NULL	NULL	NULL

-- 4.Create credit_details table

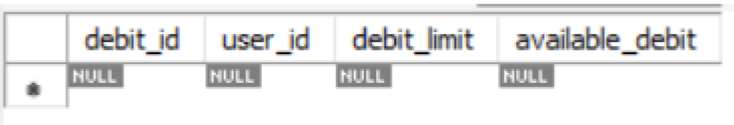
```
CREATE TABLE credit_details (
    credit_id INT PRIMARY KEY,
    user_id INT,
    credit_limit DECIMAL(15, 2),
    available_credit DECIMAL(15, 2),
    FOREIGN KEY (user_id) REFERENCES user_details(user_id)
);
```



	credit_id	user_id	credit_limit	available_credit
●	NULL	NULL	NULL	NULL

-- 5.Create debit_details table

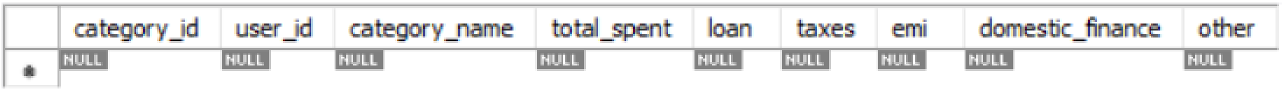
```
CREATE TABLE debit_details (
    debit_id INT PRIMARY KEY,
    user_id INT,
    debit_limit DECIMAL(15, 2),
    available_debit DECIMAL(15, 2),
    FOREIGN KEY (user_id) REFERENCES user_details(user_id)
);
```



	debit_id	user_id	debit_limit	available_debit
●	NULL	NULL	NULL	NULL

-- 6.Create spending_categories table

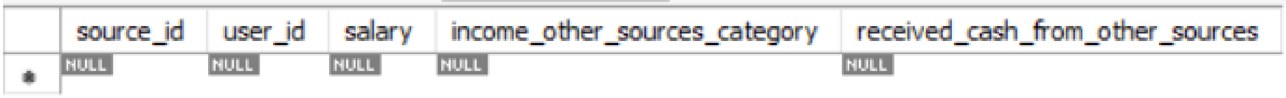
```
CREATE TABLE spending_categories (
    category_id INT PRIMARY KEY,
    user_id INT,
    category_name VARCHAR(100),
    total_spent DECIMAL(15, 2),
    loan DECIMAL(15, 2),
    taxes DECIMAL(15, 2),
    emi DECIMAL(15, 2),
    domestic_finance DECIMAL(15, 2),
    other DECIMAL(15, 2),
    FOREIGN KEY (user_id) REFERENCES user_details(user_id)
);
```



	category_id	user_id	category_name	total_spent	loan	taxes	emi	domestic_finance	other
●	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

-- 7.Create income_sources table

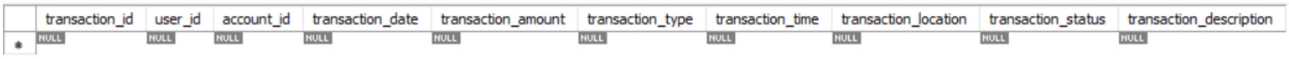
```
CREATE TABLE income_sources (
    source_id INT PRIMARY KEY,
    user_id INT,
    salary DECIMAL(15, 2),
    income_other_sources_category VARCHAR(255),
    received_cash_from_other_sources DECIMAL(15, 2),
    FOREIGN KEY (user_id) REFERENCES user_details(user_id)
);
```



	source_id	user_id	salary	income_other_sources_category	received_cash_from_other_sources
*	NULL	NULL	NULL	NULL	NULL

-- 8.Create transaction_table

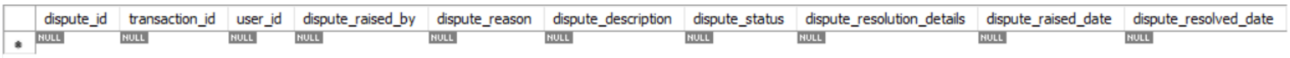
```
CREATE TABLE transactions_table (
  transaction_id INT PRIMARY KEY,
  user_id INT,
  account_id INT,
  transaction_date DATE,
  transaction_amount DECIMAL(15, 2),
  transaction_type VARCHAR(10),
  transaction_time TIMESTAMP,  -- Added from transaction_table
  transaction_location VARCHAR(255), -- Added from transaction_table
  transaction_status VARCHAR(20), -- Added from history_of_transactions
  transaction_description VARCHAR(255), -- Added from history_of_transactions
  FOREIGN KEY (user_id) REFERENCES user_details(user_id),
  FOREIGN KEY (account_id) REFERENCES bank_account_details(account_id)
);
```



	transaction_id	user_id	account_id	transaction_date	transaction_amount	transaction_type	transaction_time	transaction_location	transaction_status	transaction_description
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

-- 9.Create Dispute Resolution Table

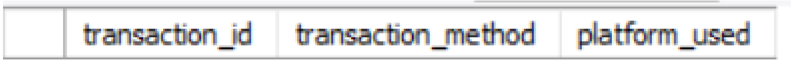
```
CREATE TABLE dispute_resolution (
  dispute_id INT PRIMARY KEY AUTO_INCREMENT,
  transaction_id INT,
  user_id INT,
  dispute_raised_by VARCHAR(50),  -- Who raised the dispute (user/merchant)
  dispute_reason VARCHAR(255),
  dispute_description TEXT,  -- Allows longer descriptions
  dispute_status VARCHAR(50),  -- Open, Pending, Resolved
  dispute_resolution_details TEXT,  -- Details of resolution
  dispute_raised_date DATE,
  dispute_resolved_date DATE,
  FOREIGN KEY (transaction_id) REFERENCES transactions_details(transaction_id),
  FOREIGN KEY (user_id) REFERENCES user_details(user_id)
);
```



	dispute_id	transaction_id	user_id	dispute_raised_by	dispute_reason	dispute_description	dispute_status	dispute_resolution_details	dispute_raised_date	dispute_resolved_date
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

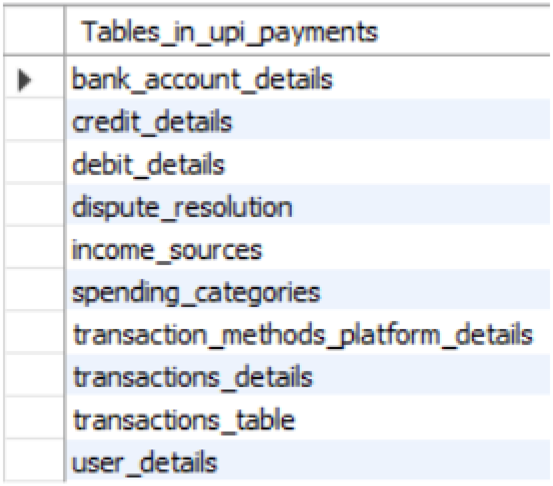
-- 10.Create transaction_methods/platform_details table

```
CREATE TABLE transaction_methods_platform_details (
    transaction_id INT,
    transaction_method VARCHAR(20),
    platform_used VARCHAR(100),
    FOREIGN KEY (transaction_id) REFERENCES transactions_details(transaction_id)
);
```



	transaction_id	transaction_method	platform_used

Show tables;



	Tables_in_upi_payments
▶	bank_account_details
	credit_details
	debit_details
	dispute_resolution
	income_sources
	spending_categories
	transaction_methods_platform_details
	transactions_details
	transactions_table
	user_details

VI. QUERIES AND OUTPUT(GUI):

1. Retrieve all transactions for a specific user:

SELECT name, email, phone_number, address, date_of_birth, gender, occupation
FROM user_details
WHERE user_id = $userId;

This query retrieves the personal details (name, email, phone number, address, date of birth, gender, and occupation) of a user specified by their user ID.

2. Transaction Count:

SELECT COUNT(*) AS number_of_transactions
FROM transactions_details
WHERE user_id = $userId;

This query counts the number of transactions for a specific user.

3.Max Expenditure:

SELECT MAX(transaction_amount) AS max_expenditure FROM transactions_details WHERE user_id = $userId;

This query retrieves the maximum transaction amount for a specific user.

4. Total Income:

SELECT SUM(salary + received_cash_from_other_sources) AS total_income FROM income_sources WHERE user_id = $userId;

This query calculates the total income for a specific user, including salary and cash received from other sources.

5. Remaining Money:

SELECT (SELECT SUM(salary + received_cash_from_other_sources) FROM income_sources WHERE user_id = $userId) - (SELECT SUM(transaction_amount) FROM transactions_details WHERE user_id = $userId) AS remaining_money;

This query calculates the remaining money for a specific user after deducting the total spent amount from the total income.

6. Credit Details:

SELECT * FROM credit_details WHERE user_id = $userId;

This query retrieves all credit details for a specific user.

7. Debit Details:

SELECT * FROM debit_details WHERE user_id = $userId;

This query retrieves all debit details for a specific user.

8. Most Used Platform:

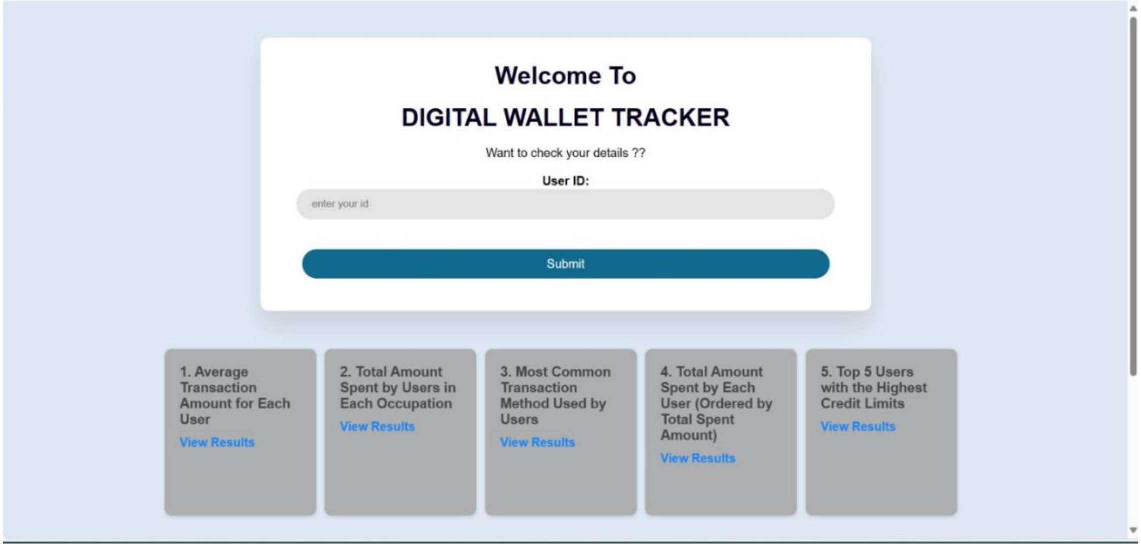
SELECT platform_used, COUNT(*) AS transaction_count FROM transaction_methods_platform_details WHERE transaction_id IN (SELECT transaction_id FROM transactions_details WHERE user_id = $userId) GROUP BY platform_used ORDER BY transaction_count DESC LIMIT 1;

This query finds the most used platform for transactions by the user with **user_id** by counting the occurrences of each **platform_used** and selecting the one with the highest count.

9. Most Common Transaction:

SELECT transaction_location, DATE(transaction_time) AS transaction_date, TIME(transaction_time) AS transaction_time, COUNT(*) AS transaction_count FROM transactions_table WHERE user_id = $userId GROUP BY transaction_location, DATE(transaction_time), TIME(transaction_time) ORDER BY transaction_count DESC LIMIT 1;

This query finds the most common transaction location, date, and time for a specific user.



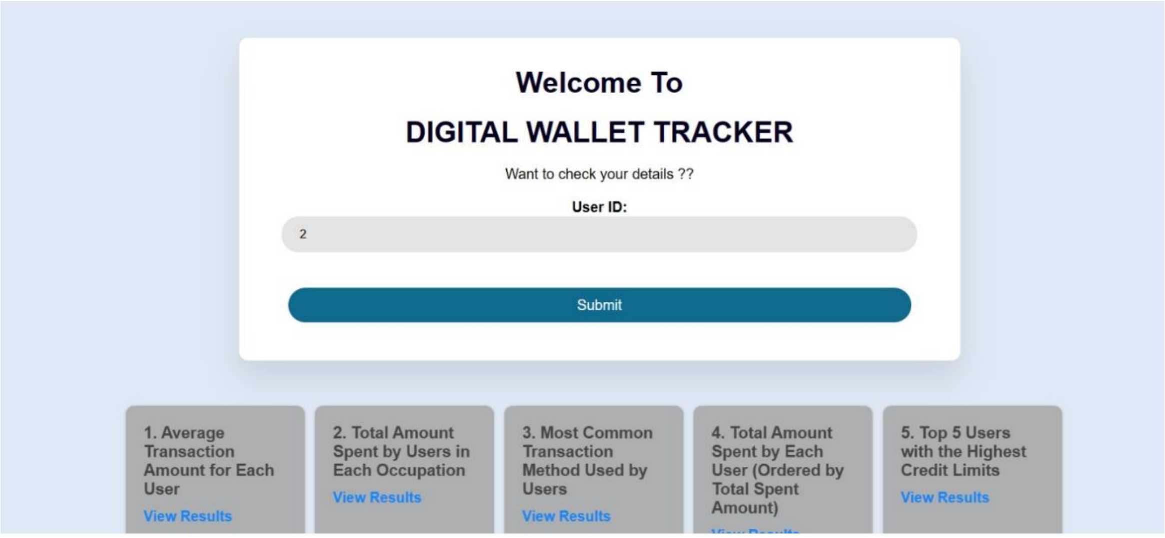
Welcome To

DIGITAL WALLET TRACKER

Want to check your details ??

User ID: enter your id

Submit
1. Average Transaction Amount for Each User
View Results
2. Total Amount Spent by Users in Each Occupation
View Results
3. Most Common Transaction Method Used by Users
View Results
4. Total Amount Spent by Each User (Ordered by Total Spent Amount)
View Results
5. Top 5 Users with the Highest Credit Limits
View Results



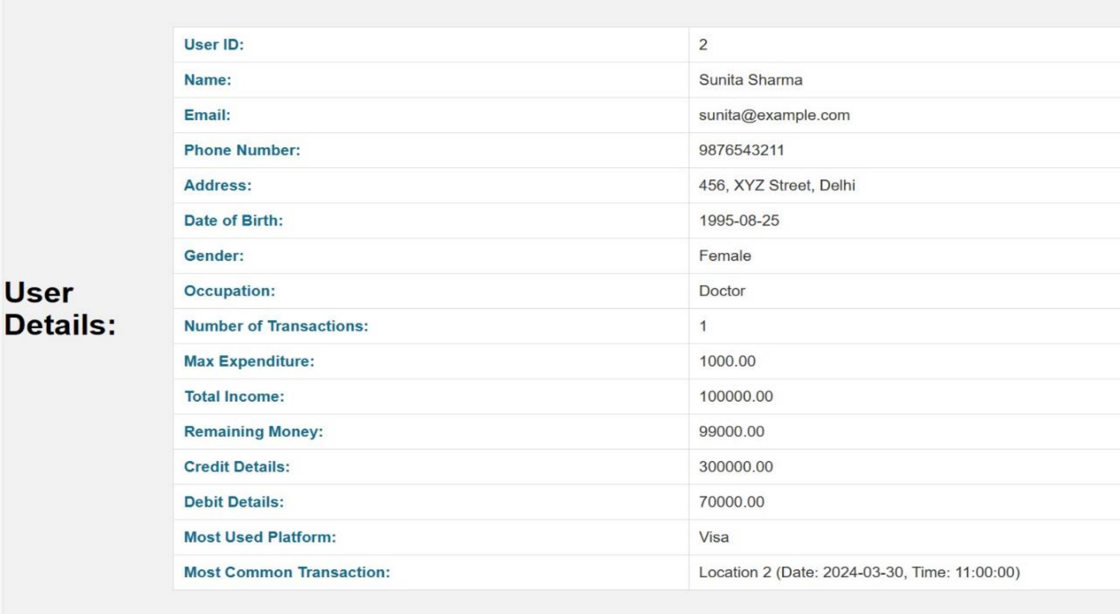
Welcome To

DIGITAL WALLET TRACKER

Want to check your details ??

User ID: 2

Submit
1. Average Transaction Amount for Each User
View Results
2. Total Amount Spent by Users in Each Occupation
View Results
3. Most Common Transaction Method Used by Users
View Results
4. Total Amount Spent by Each User (Ordered by Total Spent Amount)
5. Top 5 Users with the Highest Credit Limits
View Results



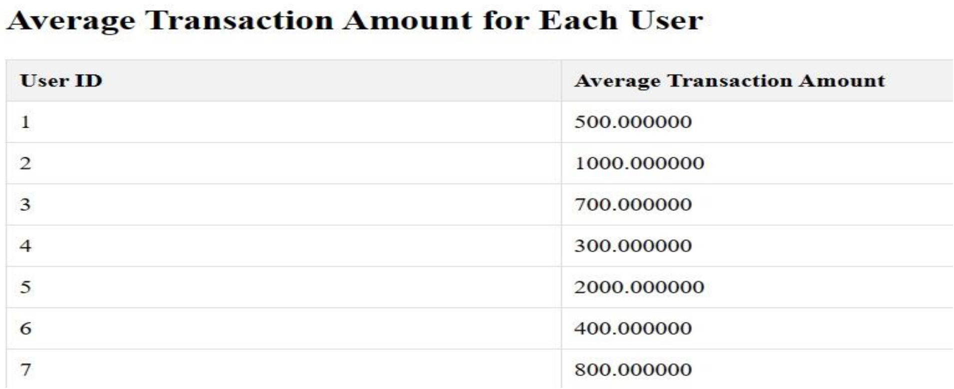
User Details:

User ID:	2
Name:	Sunita Sharma
Email:	sunita@example.com
Phone Number:	9876543211
Address:	456, XYZ Street, Delhi
Date of Birth:	1995-08-25
Gender:	Female
Occupation:	Doctor
Number of Transactions:	1
Max Expenditure:	1000.00
Total Income:	100000.00
Remaining Money:	99000.00
Credit Details:	300000.00
Debit Details:	70000.00
Most Used Platform:	Visa
Most Common Transaction:	Location 2 (Date: 2024-03-30, Time: 11:00:00)

10. Average Transaction Amount for Each User:

SELECT user_id, AVG(transaction_amount) AS avg_transaction_amount FROM transactions_table GROUP BY user_id;

This query calculates the average transaction amount for each user.



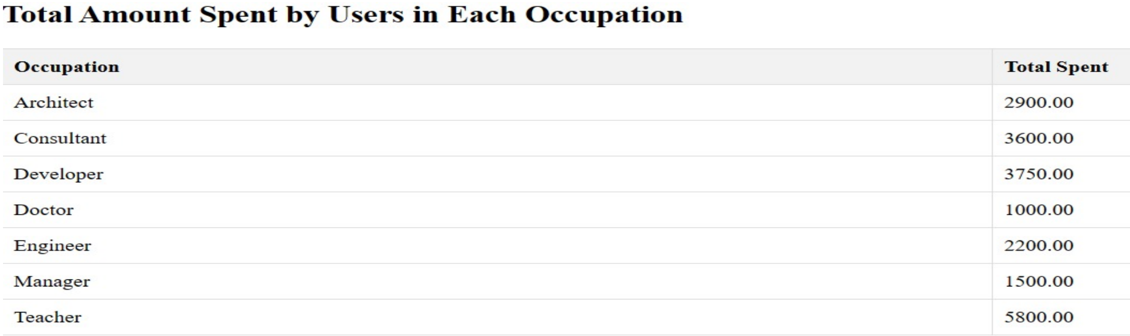
Average Transaction Amount for Each User

User ID	Average Transaction Amount
1	500.000000
2	1000.000000
3	700.000000
4	300.000000
5	2000.000000
6	400.000000
7	800.000000

11.Total Amount Spent by Users in Each Occupation:

SELECT u.occupation, SUM(t.transaction_amount) AS total_spent FROM user_details u JOIN transactions_table t ON u.user_id = t.user_id GROUP BY u.occupation;

This query calculates the total amount spent by users in each occupation.



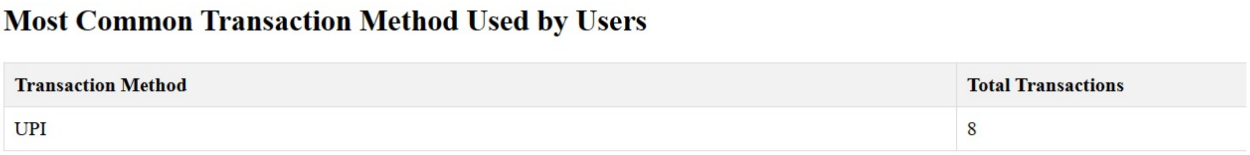
Total Amount Spent by Users in Each Occupation

Occupation	Total Spent
Architect	2900.00
Consultant	3600.00
Developer	3750.00
Doctor	1000.00
Engineer	2200.00
Manager	1500.00
Teacher	5800.00

12.Transaction Method Used by Users:

SELECT transaction_method, COUNT(*) AS total_transactions FROM transaction_methods_platform_details GROUP BY transaction_method ORDER BY total_transactions DESC LIMIT 1;

This query identifies the most common transaction method used by users by counting the occurrences of each transaction_method and selecting the one with the highest count.



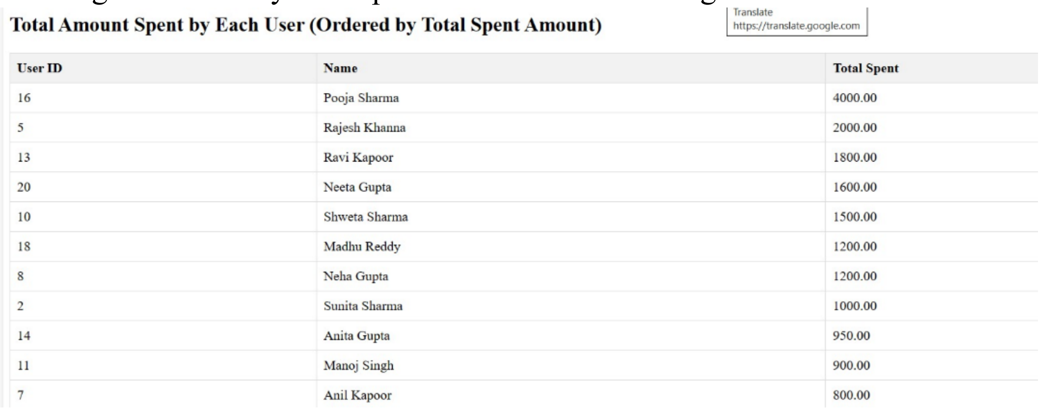
Most Common Transaction Method Used by Users

Transaction Method	Total Transactions
UPI	8

13 Total Amount Spent by Each User (Ordered by Total Spent Amount):

SELECT u.user_id, u.name, SUM(t.transaction_amount) AS total_spent FROM user_details u JOIN transactions_table t ON u.user_id = t.user_id GROUP BY u.user_id, u.name ORDER BY total_spent DESC;

This query calculates the total amount spent by each user by joining the user_details and transactions_table tables on user_id, summing up the transaction_amount for each user, and ordering the results by total spent amount in descending order.



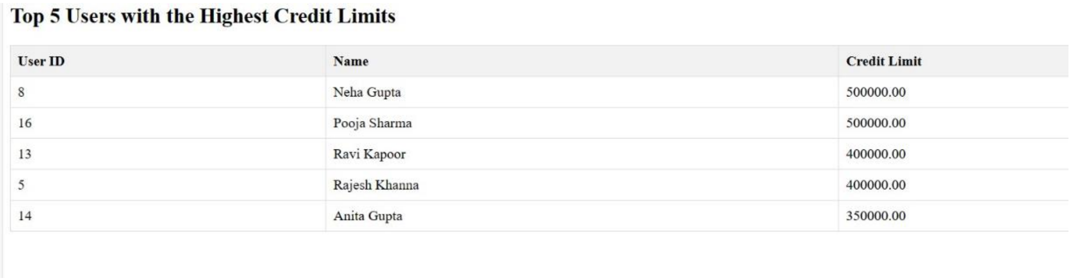
Total Amount Spent by Each User (Ordered by Total Spent Amount)

User ID	Name	Total Spent
16	Pooja Sharma	4000.00
5	Rajesh Khanna	2000.00
13	Ravi Kapoor	1800.00
20	Neeta Gupta	1600.00
10	Shweta Sharma	1500.00
18	Madhu Reddy	1200.00
8	Neha Gupta	1200.00
2	Sunita Sharma	1000.00
14	Anita Gupta	950.00
11	Manoj Singh	900.00
7	Anil Kapoor	800.00

14 Top 5 Users with the Highest Credit Limits:

SELECT u.user_id, u.name, c.credit_limit FROM user_details u JOIN credit_details c ON u.user_id = c.user_id ORDER BY c.credit_limit DESC LIMIT 5;

This query retrieves the top 5 users with the highest credit limits by joining the **user_details** and **credit_details** tables on **user_id** and ordering the results by **credit_limit** in descending order, then limiting the result to 5 rows.



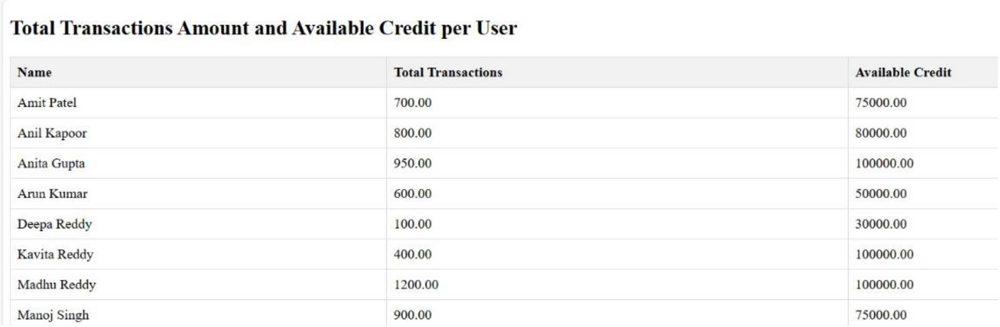
Top 5 Users with the Highest Credit Limits

User ID	Name	Credit Limit
8	Neha Gupta	500000.00
16	Pooja Sharma	500000.00
13	Ravi Kapoor	400000.00
5	Rajesh Khanna	400000.00
14	Anita Gupta	350000.00

15 Total Transactions Amount and Available Credit per User:

SELECT u.name, SUM(t.transaction_amount) AS total_transactions, c.available_credit FROM user_details u JOIN transactions_details t ON u.user_id = t.user_id JOIN credit_details c ON u.user_id = c.user_id GROUP BY u.name, c.available_credit;

This query calculates the total transactions amount and available credit for each user.



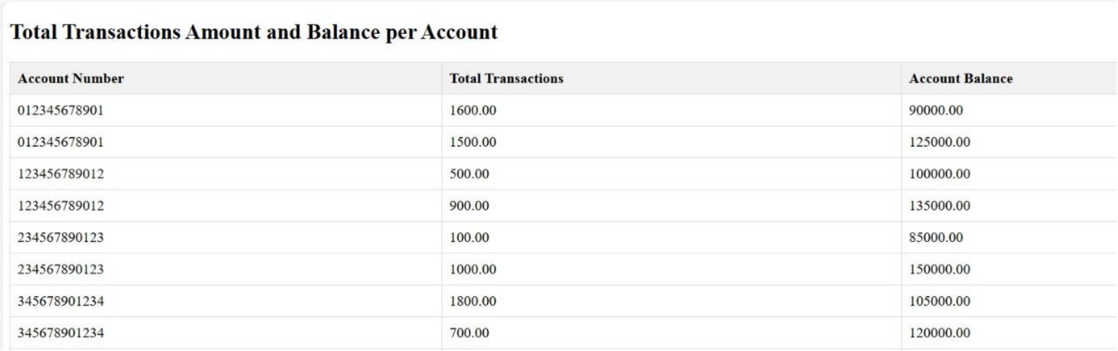
Total Transactions Amount and Available Credit per User

Name	Total Transactions	Available Credit
Amit Patel	700.00	75000.00
Anil Kapoor	800.00	80000.00
Anita Gupta	950.00	100000.00
Arun Kumar	600.00	50000.00
Deepa Reddy	100.00	30000.00
Kavita Reddy	400.00	100000.00
Madhu Reddy	1200.00	100000.00
Manoj Singh	900.00	75000.00

16 Total Transactions Amount and Balance per Account:

SELECT b.account_number, SUM(t.transaction_amount) AS total_transactions, b.account_balance FROM bank_account_details b JOIN transactions_details t ON b.account_id = t.account_id GROUP BY b.account_number, b.account_balance;

his query calculates the total transactions amount and balance per account.



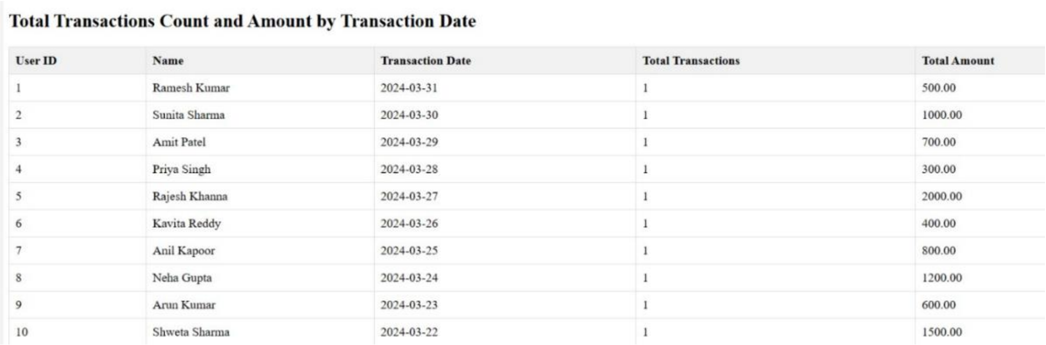
Total Transactions Amount and Balance per Account

Account Number	Total Transactions	Account Balance
012345678901	1600.00	90000.00
012345678901	1500.00	125000.00
123456789012	500.00	100000.00
123456789012	900.00	135000.00
234567890123	100.00	85000.00
234567890123	1000.00	150000.00
345678901234	1800.00	105000.00
345678901234	700.00	120000.00

17 Total Transactions Count and Amount by Transaction Date:

SELECT u.user_id, u.name, t.transaction_date, COUNT(t.transaction_id) AS total_transactions, SUM(t.transaction_amount) AS total_amount FROM transactions_table t JOIN user_details u ON t.user_id = u.user_id GROUP BY u.user_id, u.name, t.transaction_date;

This query calculates the total transactions count and amount for each user by transaction date.

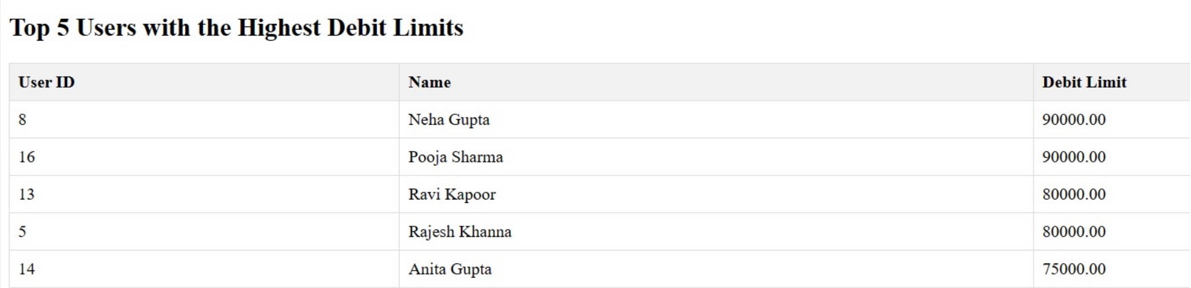


Total Transactions Count and Amount by Transaction Date

User ID	Name	Transaction Date	Total Transactions	Total Amount
1	Ramesh Kumar	2024-03-31	1	500.00
2	Sunita Sharma	2024-03-30	1	1000.00
3	Amit Patel	2024-03-29	1	700.00
4	Priya Singh	2024-03-28	1	300.00
5	Rajesh Khanna	2024-03-27	1	2000.00
6	Kavita Reddy	2024-03-26	1	400.00
7	Anil Kapoor	2024-03-25	1	800.00
8	Neha Gupta	2024-03-24	1	1200.00
9	Arun Kumar	2024-03-23	1	600.00
10	Shweta Sharma	2024-03-22	1	1500.00

18 Top 5 Users with the Highest Debit Limits:

SELECT u.user_id, u.name, d.debit_limit FROM user_details u JOIN debit_details d ON u.user_id = d.user_id ORDER BY d.debit_limit DESC LIMIT 5;



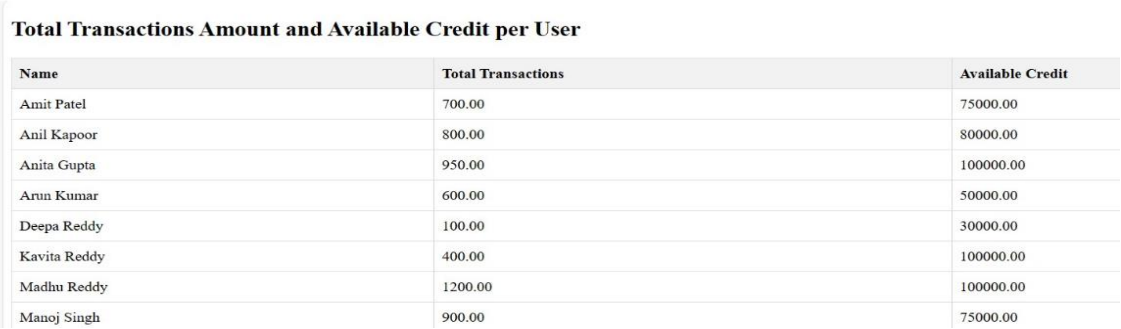
Top 5 Users with the Highest Debit Limits

User ID	Name	Debit Limit
8	Neha Gupta	90000.00
16	Pooja Sharma	90000.00
13	Ravi Kapoor	80000.00
5	Rajesh Khanna	80000.00
14	Anita Gupta	75000.00

20. Total Transactions Amount and Available Credit per User:

SELECT u.name, SUM(t.transaction_amount) AS total_transactions, c.available_credit FROM user_details u JOIN transactions_details t ON u.user_id = t.user_id JOIN credit_details c ON u.user_id = c.user_id GROUP BY u.name, c.available_credit;

This query calculates the total transactions amount and available credit for each user.



Total Transactions Amount and Available Credit per User

Name	Total Transactions	Available Credit
Amit Patel	700.00	75000.00
Anil Kapoor	800.00	80000.00
Anita Gupta	950.00	100000.00
Arun Kumar	600.00	50000.00
Deepa Reddy	100.00	30000.00
Kavita Reddy	400.00	100000.00
Madhu Reddy	1200.00	100000.00
Manoj Singh	900.00	75000.00