Data mining and discovery

SQL Assignment

Student name : Sai Krishna Vavilli

Student ID 23022047

Introduction:

This report aims to showcase how a banking database was created using SQLite and Python. The database is designed with several tables to manage customer information, track account balances, and record transactions between accounts. Key features of the database include:

(a) Ensuring transaction amounts cannot exceed the sender's available balance.
(b) Using UUIDs to uniquely identify accounts and transactions.
(c) Following relational database principles, with proper use of foreign keys to maintain data integrity.
(d) It also automatically updates the bank balance in real-time following each transaction.
(e) The transaction ID serves as a compound key i.e (unique indentifier) for each transaction. By integrating a UUID into the transaction ID, the system guarantees that each transaction is distinct, thereby effectively preventing the creation of duplicate transactions.

Database Structure and Schema :

The database is designed with three tables: customer, balance, and transaction. Below is the schema for each table.

Customer Table (1000 rows)

customer_id : (Primary Key) Unique identifier for each customer (UUID).
first_name : (Nominal Data) Customer’s first name (Text).
last_name : (Nominal Data) Customer’s last name (Text)
email : (Nominal Data) Customer’s email address (Text, can be NULL).
Phone : (Nominal Data)Customer’s phone number (random 10 digit integer, can be NULL).
date_of_birth : (Interval Data) Customer’s date of birth (Date Time, can be NULL)
account_type : (Ordinal Data) The customer's type of account, ranked as 'silver', 'gold', or 'platinum'.

Balance Table
(1000 rows)

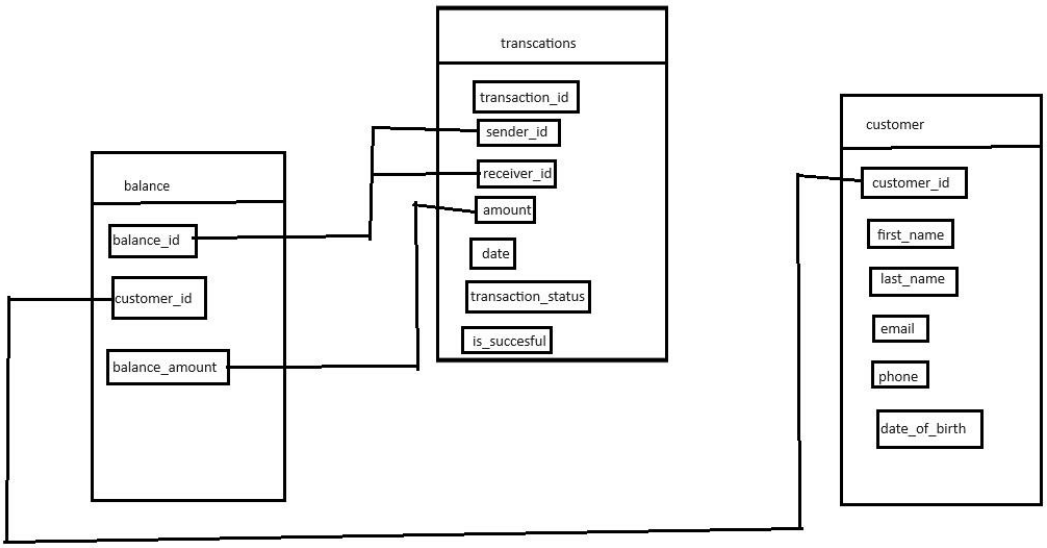
balance_id :(Primary Key): Unique identifier for each balance record (UUID).
customer_id :(Foreign Key): Links to the customer table to indicate which customer owns the balance.
Balance_amount : (Ratio Data) The balance of the customer’s account (Real Number, cannot be NULL).

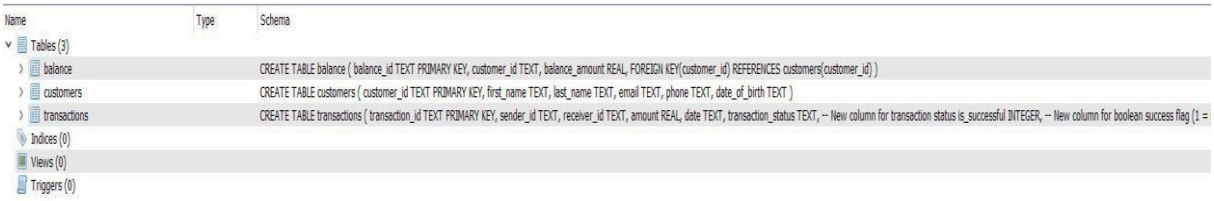
Transaction Table
(10000 rows)

transaction_id (Compund Key): Unique identifier for each transaction (UUID).
sender_id (Foreign Key): Links to the balance_id of the sender from the balance table.
receiver_id (Foreign Key): Links to the balance_id of the receiver from the balance table.
amount: (Ratio Data) The transaction amount (Real Number, cannot exceed sender's balance).
transaction_date: (Interval Data) The date and time when the transaction occurred (Date Time, stored in YYYY-MM-DD HH:MM:SS format).
transcation_status : (Nominal Data) This provides a categorical variable indicating whether the status is 'failed', 'pending', or 'completed'.

is_ successful : (Nominal Data) This column provides a Boolean value indicating whether it is successful or not.

Graphical representation of the database





Transaction Table:

We now use sender_id and receiver_id as a compound key by adding a UNIQUE constraint:

CONSTRAINT sender_receiver_unique UNIQUE (sender_id, receiver_id).This constraint ensures that no two transactions can occur between the same sender_id and receiver_id pair, preventing duplicates.

Each transaction still has a unique transaction_id (UUID), which is the primary key of the table. This allows each transaction to be uniquely identified, even if multiple transactions occur between the same sender and receiver.

The sender_ID and receiver_ID are derived from the balance_id. The balance table must be updated dynamically after each transaction to reflect the new balances for both the sender and receiver based on the transaction amount.

Process Flow:

Transaction Verification:

Before proceeding with the transaction, verify if the sender has sufficient funds. If the sender's balance is greater than or equal to the transaction amount, the transaction can continue. Otherwise, it should be flagged as unsuccessful.

Balance Update:

If the transaction is successful, update the sender’s balance by subtracting the transaction amount and update the receiver’s balance by adding the same amount.

Commit Changes:

After the balances have been updated, commit the changes to the database to ensure that the transaction is permanently recorded and the balances are accurate.

This ensures that the transaction is handled such a way that no money is created nor destroyed which is important for banking.

DATA GENERATION:

Here is how I generated the database

```
import sqlite3
import uuid
import random
from datetime import datetime, timedelta

# Function to create and populate the database
def create_banking_db():
    # Connect to SQLite database (or create it if it doesn't exist)
    conn = sqlite3.connect('banking_database.db')
    cursor = conn.cursor()

    # Create Account Types Table (Ranking System for Account Types)
    cursor.execute('''
    CREATE TABLE IF NOT EXISTS account_types (
        account_type TEXT PRIMARY KEY, -- Silver, Gold, Platinum
        rank INTEGER  -- Rank: Silver = 1, Gold = 2, Platinum = 3
    )
    ''')

    # Insert account types into the account_types table (Ordinal Data)
    cursor.executemany('''
    INSERT OR IGNORE INTO account_types (account_type, rank) VALUES (?, ?)
    ''', [
        ('Silver', 1),
        ('Gold', 2),
        ('Platinum', 3)
    ])

    # Create Customer Table with account_type (Foreign Key)
    cursor.execute('''
    CREATE TABLE IF NOT EXISTS customers (
        customer_id TEXT PRIMARY KEY,
        first_name TEXT,
        last_name TEXT,
        email TEXT,
        phone TEXT,
        date_of_birth TEXT,
```

```
    account_type TEXT,  -- Foreign Key to account_types table
    FOREIGN KEY(account_type) REFERENCES account_types(account_type)
  )
  ''')

  # Create Balance Table
  cursor.execute('''
  CREATE TABLE IF NOT EXISTS balance (
    balance_id TEXT PRIMARY KEY,
    customer_id TEXT,
    balance_amount REAL,
    FOREIGN KEY(customer_id) REFERENCES customers(customer_id)
  )
  ''')

  # Create Transaction Table with compound key (transaction_id, sender_id,
receiver_id)
  cursor.execute('''
  CREATE TABLE IF NOT EXISTS transactions (
    transaction_id TEXT,
    sender_id TEXT,
    receiver_id TEXT,
    amount REAL,
    date TEXT,
    transaction_status TEXT,
    is_successful INTEGER,
    PRIMARY KEY (transaction_id, sender_id, receiver_id),
    FOREIGN KEY(sender_id) REFERENCES balance(balance_id),
    FOREIGN KEY(receiver_id) REFERENCES balance(balance_id)
  )
  ''')

  # Generate random data for customers
  def generate_random_customer():
    first_names = ['John', 'Jane', 'Alice', 'Bob', 'Charlie', 'David', 'Emily']
    last_names = ['Smith', 'Doe', 'Johnson', 'Brown', 'Taylor', 'Anderson']
    emails = ['@gmail.com', '@yahoo.com', '@outlook.com']
    phone_numbers = ['+1234567890', '+1987654321', '+1112233445']
    date_of_birth = (datetime.today() -
timedelta(days=random.randint(18*365, 60*365))).strftime('%Y-%m-%d')
```

```
    # Assigning account_type from 'Silver', 'Gold', 'Platinum'
    account_types = ['Silver', 'Gold', 'Platinum']
    account_type = random.choice(account_types)

    customer_id = str(uuid.uuid4())
    first_name = random.choice(first_names)
    last_name = random.choice(last_names)
    email = first_name.lower() + last_name.lower() + random.choice(emails)
    phone = random.choice(phone_numbers)

    return customer_id, first_name, last_name, email, phone,
date_of_birth, account_type

  # Insert 1000 customers into the customer table
  for _ in range(1000):
    customer = generate_random_customer()
    cursor.execute('''
    INSERT INTO customers (customer_id, first_name, last_name, email,
phone, date_of_birth, account_type)
    VALUES (?, ?, ?, ?, ?, ?, ?)
    ''', customer)

  # Generate random balances for customers
  def  generate_random_balance(customer_id):
    balance_amount = random.uniform(100, 5000)  # Random balance
between 100 and 5000
    balance_id = str(uuid.uuid4())
    return balance_id, customer_id, balance_amount

  # Insert balances for customers
  for _ in range(1000):
    customer_id = str(uuid.uuid4()) # Random customer_id for balance
    balance = generate_random_balance(customer_id)
    cursor.execute('''
    INSERT INTO balance (balance_id, customer_id, balance_amount)
    VALUES (?, ?, ?)
    ''', balance)

  # Generate random transactions
```

```
    def generate_random_transaction(sender_id, receiver_id, amount, date):
        transaction_id = str(uuid.uuid4())
        # Randomly assign status and success flag
        status = random.choice(['Pending', 'Completed', 'Failed']) # Transaction
status
        is_successful = 1 if status == 'Completed' else 0  # Success flag (1 for
success, 0 for failure)
        return transaction_id, sender_id, receiver_id, amount, date, status,
is_successful

    # Insert 10,000 transactions (with realistic constraints)
    for _ in range(10000):  # Insert 10,000 transactions
        # Select a random sender and receiver from the balance table
        sender_balance = cursor.execute('SELECT balance_id FROM balance
ORDER BY RANDOM() LIMIT 1').fetchone()
        receiver_balance = cursor.execute('SELECT balance_id FROM balance
ORDER BY RANDOM() LIMIT 1').fetchone()

        if sender_balance and receiver_balance:
            sender_id = sender_balance[0]
            receiver_id = receiver_balance[0]

            # Ensure sender's balance is enough for the transaction
            sender_balance_amount = cursor.execute('SELECT balance_amount
FROM balance WHERE balance_id=?', (sender_id,)).fetchone()[0]
            amount = random.uniform(10, sender_balance_amount) # Ensure
transaction amount is less than or equal to balance
            transaction_date = datetime.today().strftime('%Y-%m-%d %H:%M:%S')

            # Generate the transaction
            transaction = generate_random_transaction(sender_id, receiver_id,
amount, transaction_date)

            # Insert the transaction into the table
            cursor.execute('''
            INSERT INTO transactions (transaction_id, sender_id, receiver_id,
amount, date, transaction_status, is_successful)
            VALUES (?, ?, ?, ?, ?, ?, ?)
            ''', transaction)
```

```
    # Commit the changes
    conn.commit()

    # Close the connection
    conn.close()

# Function to update the balance dynamically after each transaction
def update_balance_after_transaction():
    conn = sqlite3.connect('banking_database.db')
    cursor = conn.cursor()

    # Get all transactions
    cursor.execute('SELECT * FROM transactions')
    transactions = cursor.fetchall()

    for transaction in transactions:
        sender_id = transaction[1]
        receiver_id = transaction[2]
        amount = transaction[3]
        transaction_status = transaction[5]  # Get the transaction status
        is_successful = transaction[6]  # Get the success flag

        # If transaction is successful, update balances
        if is_successful == 1:
            # Update sender's balance
            cursor.execute('SELECT balance_amount FROM balance WHERE
balance_id=?', (sender_id,))
            sender_balance = cursor.fetchone()
            if sender_balance and sender_balance[0] >= amount:
                new_sender_balance = sender_balance[0] - amount
                cursor.execute('UPDATE balance SET balance_amount=? WHERE
balance_id=?', (new_sender_balance, sender_id))

                # Update receiver's balance
                cursor.execute('SELECT balance_amount FROM balance WHERE
balance_id=?', (receiver_id,))
                receiver_balance = cursor.fetchone()
                if receiver_balance:
                    new_receiver_balance = receiver_balance[0] + amount
```

```
                cursor.execute('UPDATE balance SET balance_amount=? WHERE
balance_id=?', (new_receiver_balance, receiver_id))

    conn.commit()
    conn.close()

# Create the database and populate it
create_banking_db()

# Update the balances dynamically after transactions
update_balance_after_transaction()

print("Banking database created and transactions processed successfully.")
```

Displaying null values in customer table:

SQL 1

```sql
SELECT
    COUNT(*) AS null_email_phone_count
FROM
    customer
WHERE
    email IS NULL OR phone IS NULL;
```

	null_email_phone_count
1	752

```
Execution finished without errors.
Result: 1 rows returned in 2ms
```

Output:

Customer table:

Table: customers

Filter in any column

	customer_id	first_name	last_name	email	phone	date_of_birth
	Filter	Filter	Filter	Filter	Filter	Filter
1	b590fdd2-889f-474c-865d-f657927e429e	Jane	Anderson	janeanderson@yahoo.com	+1987654321	1999-01-03
2	17b5df9e-857f-4f50-9244-7f8ef47d7c76	Emily	Taylor	emilytaylor@yahoo.com	+1987654321	1991-07-12
3	1a058b84-29a0-4c89-b3d4-8f088444c75e	Emily	Smith	emilysmith@yahoo.com	+1987654321	1971-04-05
4	f33b9eb3-dfab-4326-b283-c7d4e040d7ff	David	Brown	davidbrown@gmail.com	+1234567890	1987-12-15
5	4710eff6-9234-404b-9177-47fe66ed2c14	Jane	Taylor	janetaylor@outlook.com	+1112233445	1976-03-23
6	8f4f8f66-d1f8-40af-9e1c-3ed90b25ab7d	Bob	Taylor	bobtaylor@gmail.com	+1987654321	1977-01-19
7	561e588e-33eb-4e73-bd8a-0d76f3af741c	Charlie	Johnson	charliejohnson@gmail.com	+1112233445	2004-05-15
8	e731cea3-f489-43bf-8545-5eba9d16ce89	Emily	Doe	emilydoe@yahoo.com	+1112233445	1967-09-24
9	127d83cb-0e2b-4141-909d-c004247a0839	Bob	Doe	bobdoe@outlook.com	+1987654321	1987-09-28
10	b9ffef06-3080-4058-9f99-88651996a96d	John	Doe	johndoe@outlook.com	+1987654321	1987-10-16
11	395328e7-7fbe-4ec9-b313-fd401ba9c2c6	Bob	Anderson	bobanderson@gmail.com	+1112233445	1988-02-19
12	3ff54453-34a4-4252-8c76-ccf9e2efed62	John	Brown	johnbrown@outlook.com	+1987654321	1992-08-30
13	64d30d2f-d342-4883-a4cc-ebdc1e64427a	Jane	Brown	janebrown@gmail.com	+1112233445	1969-12-28
14	67140c4d-b4ac-49c2-ae31-d1073e9272e2	David	Anderson	davidanderson@gmail.com	+1987654321	1988-01-01
15	b85ce862-74c8-494a-b55c-238009521502	David	Brown	davidbrown@yahoo.com	+1987654321	1979-03-23
16	f3aa0c1f-7afe-48bb-9c57-469e0066d935	John	Smith	johnsmith@yahoo.com	+1234567890	1986-07-01
17	b69b1851-638b-45ad-9765-e58f1b02d854	Emily	Taylor	emilytaylor@gmail.com	+1234567890	1974-03-12
18	361a79bb-cad1-47f8-9002-09f6c9813dcf	Alice	Anderson	aliceanderson@yahoo.com	+1112233445	1976-01-13
19	c30c01fb-92a5-4993-9459-8fba0b262f01	Emily	Johnson	emilyjohnson@gmail.com	+1234567890	2005-12-31
20	f9380dc9-8c2b-47be-bf29-e55e10ebc043	Alice	Taylor	alicetaylor@yahoo.com	+1987654321	1975-04-14
21	e460ffd4-4773-4813-b515-c9571b46b730	Charlie	Anderson	charlieanderson@outlook.com	+1987654321	1998-02-04
22	d220cee5-2799-4fc3-991b-29ad2b1d85af	Alice	Smith	alicesmith@outlook.com	+1234567890	1981-06-18
23	34cdcd96-d7fe-4e91-9644-b3e632777d2f	Bob	Smith	bobsmith@gmail.com	+1987654321	1978-03-13
24	fd036278-a8ef-4d6e-a97d-eff39cec4397	Alice	Brown	alicebrown@yahoo.com	+1112233445	1985-04-01
25	87a5bea6-f8b6-4503-8867-615cce643f80	Emily	Taylor	emilytaylor@outlook.com	+1234567890	1966-02-01
26	3b61411e-9f4e-49c6-a482-600455c95d59	John	Anderson	johnanderson@outlook.com	+1234567890	2004-02-29
27	d5f33c20-cf6a-4b20-ac45-e0a90ced5a37	Alice	Smith	alicesmith@outlook.com	+1234567890	1972-05-28
28	4d8c0c49-3a03-4f6e-bec5-0c097b6217bd	Alice	Smith	alicesmith@outlook.com	+1234567890	1978-08-20
29	2e1842db-d141-4ac7-8323-55a8eb90dcf8	John	Doe	johndoe@yahoo.com	+1234567890	1968-06-10
30	90567950-7273-4de7-a60f-aad0fbef86dc	Jane	Doe	janedoe@gmail.com	+1987654321	1982-07-31

1 - 30 of 1000

Balance table:

Table: balance

Filter in any column

	balance_id	customer_id	balance_amount
	Filter	Filter	Filter
1	12df4b69-3017-4703-91ef-7c54fdba2bcb	2af8d389-4f53-4fa2-809e-672fbcb7eef0	2413.913843953334
2	57198a54-9dd1-49b9-af81-f8c4726b586c	c6f04e94-08d1-4d10-88b2-9ff5b6244939	3269.944145638316
3	6b007976-51d3-40e6-b8bb-81f097030f6f	bf5ad7e2-d195-40b7-aa9a-179f690f2efb	3589.4066416472
4	f3520ad4-286a-4bda-b0ee-34d7baea189f	584a55a1-d12f-46d8-8b59-672a2a1c2042	635.414453697011
5	eeaeaaa4-b22b-4672-9059-8a9f341cc372	d8c393bc-4fc9-4754-ae85-6a601572f4b5	1432.972915744426
6	07b5d402-6710-42d4-a46e-aab644cc97ca	082ecbbd-3b17-4d24-966f-cf32a0075716	481.826400854322
7	110000f2-0e73-432a-9ca2-9103b7c6537e	5796479a-e2d6-4a80-97c5-37490000e1c4	1539.969253329203
8	93386940-97b0-41e1-a70f-13de891a2791	f02b9922-0c3e-4afa-be6d-06fead1a3856	1847.381853234428
9	7c6d330e-c003-4a52-ae57-2609e7e83b64	80f265d7-7dab-4d62-9f3b-3db74fb3dc7e	1712.600602550004
10	5acca5e3-9fdc-4ba8-b4bd-075dfd9d6b93	9aa7cd02-3fa2-4625-b107-819bcb4ecd69	8001.694467734443
11	abfb02e6-922a-4c0b-9e5d-32ab5170cb47	a24e1935-90d5-4b4d-aa81-457436a44a3f	1028.414999700044
12	550d1770-e298-403c-ba58-e932cc0455f4	e132ea55-9845-4cc9-8aba-c3b21769d363	160.3293202365544
13	abcc6fc0-af56-4b7e-b13b-2a4be301b7de	f095625a-e9f3-4158-bd93-b778b4a8d832	3093.780921303095
14	e7613d13-00aa-4e75-a771-ac018540f94b	fafc2264-ad9d-4fa5-91c3-611cd0a88ecc	4006.490814078113
15	67e87504-79b7-4292-8209-5f5485435c05	c94e95f5-a22f-45b0-b3c0-e2b1d991e013	1287.766318129667
16	af76f27f-b230-48d8-9e65-a1cc17ca6d5e	be3d346f-fcd5-4c24-909b-b97a3f3d4888	296.9501396376447
17	81c8ddf3-1ee0-4e3f-9c12-5a5f10106e36	af957249-6a63-41df-9de3-b933ecda4a79	585.300675515224
18	0426d904-936d-46a5-8340-7206d99846d6	20f4ff6c-230c-4924-b4db-a08b8f90a205	765.534773760264
19	a5ee14ed-727c-4f47-8453-15462d512013	43cdf332-b1da-43b1-85f5-f46aa82882c3	2547.16803262474
20	18fdcb5b-51fb-42db-bec3-2bbba0a94ba6	7db56d25-7d90-4a63-b273-e73a207c0dd9	3101.19658742334
21	bbf0e5d7-5cd6-4524-8426-64ab3550c835	6bf43c81-7954-484c-9358-bad58ad2f752	5149.53677386536
22	d29aad2b-73d6-4ba9-a6c0-b1cae7e49e6f	da25c8de-d340-4aea-a08b-cc35080d8077	7356.87478901369
23	57a535f0-8aff-49a3-81a1-850d1d4a43e5	447cdf25-cf2b-4c9a-a92c-32a180c9a30d	4553.27523122512
24	c8e9b668-7f76-406f-b922-646787eab79a	65ffe098-a5ab-4bbc-8447-df262955f801	3093.36434169379
25	15f07f12-b0e6-4ac1-9398-dc8dfb225abe	52743642-3481-4590-b9b5-4b5fd807af10	1771.84019695106
26	6f3f364b-52be-4f33-9e03-ae497925c581	fe9e09e4-0ebf-4fd8-ab65-f797bfa835eb	1402.18120112393
27	387def35-ae3c-4f82-9ea8-81acf6967ac6	a3344bef-4347-4064-a506-10ca494e6c11	348.430928368036
28	03ada498-95b4-4728-9042-b7c28fc4764c	50e4f145-7ebc-4118-be1b-32f09756c9dd	2468.02051031601
29	0296d309-e4ed-4b1b-b25a-518124ff74e3	ed08d0a0-8d99-4a0f-a8d3-b9081ea9df50	2138.51978029332
30	2ef5b2d5-9e41-470f-8303-628dff9d994e	5800d2c8-7f7d-4fb0-abce-13e766f726a7	3881.03516764124

1 - 30 of 1000

Transcation table:

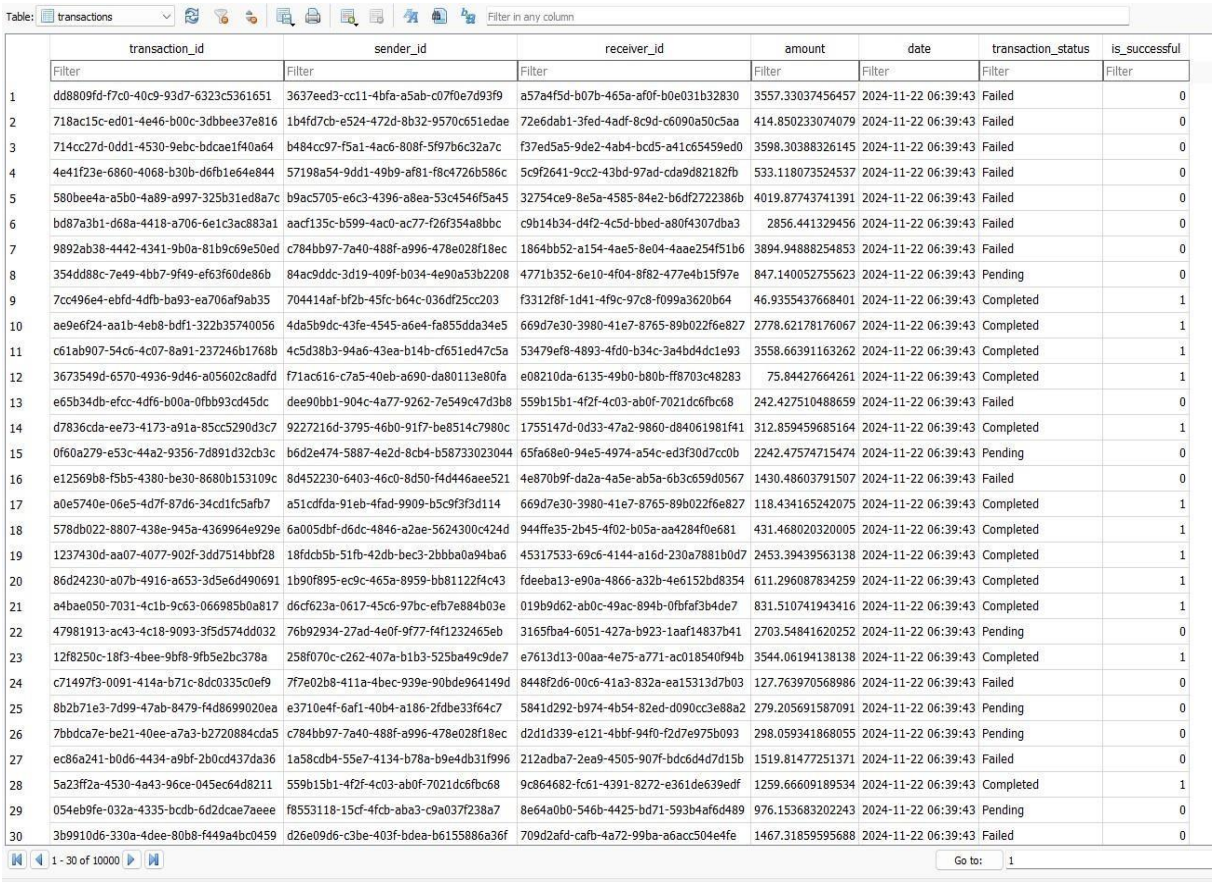


Table: transactions — Filter in any column

	transaction_id	sender_id	receiver_id	amount	date	transaction_status	is_successful
	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	dd8809fd-f7c0-40c9-93d7-6323c5361651	3637eed3-cc11-4bfa-a5ab-c07f0e7d93f9	a57a4f5d-b07b-465a-af0f-b0e031b32830	3557.33037456457	2024-11-22 06:39:43	Failed	0
2	718ac15c-ed01-4e46-b00c-3dbbee37e816	1b4fd7cb-e524-472d-8b32-9570c651edae	72e6dab1-3fed-4adf-8c9d-c6090a50c5aa	414.850233074079	2024-11-22 06:39:43	Failed	0
3	714cc27d-0dd1-4530-9ebc-bdcae1f40a64	b484cc97-f5a1-4ac6-808f-5f97b6c32a7c	f37ed5a5-9de2-4ab4-bcd5-a41c65459ed0	3598.30388326145	2024-11-22 06:39:43	Failed	0
4	4e41f23e-6860-4068-b30b-d6fb1e64e844	57198a54-9dd1-49b9-af81-f8c4726b586c	5c9f2641-9cc2-43bd-97ad-cda9d82182fb	533.118073524537	2024-11-22 06:39:43	Failed	0
5	580bee4a-a5b0-4a89-a997-325b31ed8a7c	b9ac5705-e6c3-4396-a8ea-53c4546f5a45	32754ce9-8e5a-4585-84e2-b6df2722386b	4019.87743741391	2024-11-22 06:39:43	Failed	0
6	bd87a3b1-d68a-4418-a706-6e1c3ac883a1	aacf135c-b599-4ac0-ac77-f26f354a8bbc	c9b14b34-d4f2-4c5d-bbed-a80f4307dba3	2856.441329456	2024-11-22 06:39:43	Failed	0
7	9892ab38-4442-4341-9b0a-81b9c69e50ed	c784bb97-7a40-488f-a996-478e028f18ec	1864bb52-a154-4ae5-8e04-4aae254f51b6	3894.94888254853	2024-11-22 06:39:43	Failed	0
8	354dd88c-7e49-4bb7-9f49-ef63f60de86b	84ac9ddc-3d19-409f-b034-4e90a53b2208	4771b352-6e10-4f04-8f82-477e4b15f97e	847.140052755623	2024-11-22 06:39:43	Pending	0
9	7cc496e4-ebfd-4dfb-ba93-ea706af9ab35	704414af-bf2b-45fc-b64c-036df25cc203	f3312f8f-1d41-4f9c-97c8-f099a3620b64	46.9355437668401	2024-11-22 06:39:43	Completed	1
10	ae9e6f24-aa1b-4eb8-bdf1-322b35740056	4da5b9dc-43fe-4545-a6e4-fa855dda34e5	669d7e30-3980-41e7-8765-89b022f6e827	2778.62178176067	2024-11-22 06:39:43	Completed	1
11	c61ab907-54c6-4c07-8a91-237246b1768b	4c5d38b3-94a6-43ea-b14b-cf651ed47c5a	53479ef8-4893-4fd0-b34c-3a4bd4dc1e93	3558.66391163262	2024-11-22 06:39:43	Completed	1
12	3673549d-6570-4936-9d46-a05602c8adfd	f71ac616-c7a5-40eb-a690-da80113e80fa	e08210da-6135-49b0-b80b-ff8703c48283	75.84427664261	2024-11-22 06:39:43	Completed	1
13	e65b34db-efcc-4df6-b00a-0fbb93cd45dc	dee90bb1-904c-4a77-9262-7e549c47d3b8	559b15b1-4f2f-4c03-ab0f-7021dc6fbc68	242.427510488659	2024-11-22 06:39:43	Failed	0
14	d7836cda-ee73-4173-a91a-85cc5290d3c7	9227216d-3795-46b0-91f7-be8514c7980c	1755147d-0d33-47a2-9860-d84061981f41	312.859459685164	2024-11-22 06:39:43	Completed	1
15	0f60a279-e53c-44a2-9356-7d891d32cb3c	b6d2e474-5887-4e2d-8cb4-b58733023044	65fa68e0-94e5-4974-a54c-ed3f30d7cc0b	2242.47574715474	2024-11-22 06:39:43	Pending	0
16	e12569b8-f5b5-4380-be30-8680b153109c	8d452230-6403-46c0-8d50-f4d446aee521	4e870b9f-da2a-4a5e-ab5a-6b3c659d0567	1430.48603791507	2024-11-22 06:39:43	Failed	0
17	a0e5740e-06e5-4d7f-87d6-34cd1fc5afb7	a51cdfda-91eb-4fad-9909-b5c9f3f3d114	669d7e30-3980-41e7-8765-89b022f6e827	118.434165242075	2024-11-22 06:39:43	Completed	1
18	578db022-8807-438e-945a-4369964e929e	6a005dbf-d6dc-4846-a2ae-5624300c424d	944ffe35-2b45-4f02-b05a-aa4284f0e681	431.468020320005	2024-11-22 06:39:43	Completed	1
19	1237430d-aa07-4077-902f-3dd7514bbf28	18fdcb5b-51fb-42db-bec3-2bbba0a94ba6	45317533-69c6-4144-a16d-230a7881b0d7	2453.39439563138	2024-11-22 06:39:43	Completed	1
20	86d24230-a07b-4916-a653-3d5e6d490691	1b90f895-ec9c-465a-8959-bb81122f4c43	fdeeba13-e90a-4866-a32b-4e6152bd8354	611.296087834259	2024-11-22 06:39:43	Completed	1
21	a4bae050-7031-4c1b-9c63-066985b0a817	d6cf623a-0617-45c6-97bc-efb7e884b03e	019b9d62-ab0c-49ac-894b-0fbfaf3b4de7	831.510741943416	2024-11-22 06:39:43	Completed	1
22	47981913-ac43-4c18-9093-3f5d574dd032	76b92934-27ad-4e0f-9f77-f4f1232465eb	3165fba4-6051-427a-b923-1aaf14837b41	2703.54841620252	2024-11-22 06:39:43	Pending	0
23	12f8250c-18f3-4bee-9bf8-9fb5e2bc378a	258f070c-c262-407a-b1b3-525ba49c9de7	e7613d13-00aa-4e75-a771-ac018540f94b	3544.06194138138	2024-11-22 06:39:43	Completed	1
24	c71497f3-0091-414a-b71c-8dc0335c0ef9	7f7e02b8-411a-4bec-939e-90bde964149d	8448f2d6-00c6-41a3-832a-ea15313d7b03	127.763970568986	2024-11-22 06:39:43	Failed	0
25	8b2b71e3-7d99-47ab-8479-f4d8699020ea	e3710e4f-6af1-40b4-a186-2fdbe33f64c7	5841d292-b974-4b54-82ed-d090cc3e88a2	279.205691587091	2024-11-22 06:39:43	Pending	0
26	7bbdca7e-be21-40ee-a7a3-b2720884cda5	c784bb97-7a40-488f-a996-478e028f18ec	d2d1d339-e121-4bbf-94f0-f2d7e975b093	298.059341868055	2024-11-22 06:39:43	Pending	0
27	ec86a241-b0d6-4434-a9bf-2b0cd437da36	1a58cdb4-55e7-4134-b78a-b9e4db31f996	212adba7-2ea9-4505-907f-bdc6d4d7d15b	1519.81477251371	2024-11-22 06:39:43	Failed	0
28	5a23ff2a-4530-4a43-96ce-045ec64d8211	559b15b1-4f2f-4c03-ab0f-7021dc6fbc68	9c864682-fc61-4391-8272-e361de639edf	1259.66609189534	2024-11-22 06:39:43	Completed	1
29	054eb9fe-032a-4335-bcdb-6d2dcae7aeee	f8553118-15cf-4fcb-aba3-c9a037f238a7	8e64a0b0-546b-4425-bd71-593b4af6d489	976.153683202243	2024-11-22 06:39:43	Pending	0
30	3b9910d6-330a-4dee-80b8-f449a4bc0459	d26e09d6-c3be-403f-bdea-b6155886a36f	709d2afd-cafb-4a72-99ba-a6acc504e4fe	1467.31859595688	2024-11-22 06:39:43	Failed	0

1 - 30 of 10000 — Go to: 1

Example Queries:

Get a Customer's Balance

```sql
SELECT balance_amount
FROM balance
WHERE customer_id = '2af8d389-4f53-4fa2-809e-672fbcb7eef0';
```

	balance_amount
1	2413.91384395334

```
Execution finished without errors.
Result: 1 rows returned in 2ms
At line 1:
SELECT balance_amount
FROM balance
WHERE customer_id = '2af8d389-4f53-4fa2-809e-672fbcb7eef0';
```

Get All Transactions of a Specific Customer

```sql
SELECT *
FROM transactions
WHERE sender_id = '3637eed3-cc11-4bfa-a5ab-c07f0e7d93f9'
   OR receiver_id = 'd4f3e9bc-a12b-43cd-9e7d-fb5f2a5b1f4c';
```

	transaction_id	sender_id	receiver_id	amount	date	transaction_status	is_successful
1	dd8809fd-f7c0-40c9-93d7-6323c5361651	3637eed3-cc11-4bfa-a5ab-c07f0e7d93f9	a57a4f5d-b07b-465a-af0f-b0e031b32830	3557.33037456457	2024-11-22 06:39:43	Failed	0
2	e9b21bc1-145f-4533-8286-a9053f058352	3637eed3-cc11-4bfa-a5ab-c07f0e7d93f9	7ae0a920-0bcf-42df-9bd3-a3f16c99e266	3707.03225039196	2024-11-22 06:39:44	Pending	0
3	3a5eaecd-308a-432c-817a-268b8408299d	3637eed3-cc11-4bfa-a5ab-c07f0e7d93f9	99929772-e7f4-4b2e-b738-d9e651a32f3d	2114.13693179115	2024-11-22 06:39:44	Pending	0
4	2f160465-0424-4b55-bfd5-6d2b4cab49ef	3637eed3-cc11-4bfa-a5ab-c07f0e7d93f9	e0f99276-de2c-4c83-9c01-84b0de24c61a	720.017758313702	2024-11-22 06:39:44	Completed	1
5	1c3cd88f-6893-41ef-816e-3a1ce9df80af	3637eed3-cc11-4bfa-a5ab-c07f0e7d93f9	dee90bb1-904c-4a77-9262-7e549c47d3b8	2725.58815141814	2024-11-22 06:39:45	Pending	0
6	b7d91b53-4679-41e7-8786-12f40b3d1129	3637eed3-cc11-4bfa-a5ab-c07f0e7d93f9	1886cef8-e0dc-416a-9ead-1cf244b748e0	2037.93962641784	2024-11-22 06:39:46	Failed	0

```
Execution finished without errors.
Result: 7 rows returned in 7ms
At line 1:
SELECT *
FROM transactions
WHERE sender_id = '3637eed3-cc11-4bfa-a5ab-c07f0e7d93f9'
   OR receiver_id = 'd4f3e9bc-a12b-43cd-9e7d-fb5f2a5b1f4c';
```

Get All Completed Transactions

SQL 1

```sql
SELECT *
FROM transactions
WHERE transaction_status = 'Completed';
```

	transaction_id	sender_id	receiver_id	amount	date	transaction_status	is_successful
1	7cc496e4-ebfd-4dfb-ba93-ea706af9ab35	704414af-bf2b-45fc-b64c-036df25cc203	f3312f8f-1d41-4f9c-97c8-f099a3620b64	46.9355437668401	2024-11-22 06:39:43	Completed	1
2	ae9e6f24-aa1b-4eb8-bdf1-322b35740056	4da5b9dc-43fe-4545-a6e4-fa855dda34e5	669d7e30-3980-41e7-8765-89b022f6e827	2778.62178176067	2024-11-22 06:39:43	Completed	1
3	c61ab907-54c6-4c07-8a91-237246b1768b	4c5d38b3-94a6-43ea-b14b-cf651ed47c5a	53479ef8-4893-4fd0-b34c-3a4bd4dc1e93	3558.66391163262	2024-11-22 06:39:43	Completed	1
4	3673549d-6570-4936-9d46-a05602c8adfd	f71ac616-c7a5-40eb-a690-da80113e80fa	e08210da-6135-49b0-b80b-ff8703c48283	75.84427664261	2024-11-22 06:39:43	Completed	1
5	d7836cda-ee73-4173-a91a-85cc5290d3c7	9227216d-3795-46b0-91f7-be8514c7980c	1755147d-0d33-47a2-9860-d84061981f41	312.859459685164	2024-11-22 06:39:43	Completed	1
6	a0e5740e-06e5-4d7f-87d6-34cd1fc5afb7	a51cdfda-91eb-4fad-9909-b5c9f3f3d114	669d7e30-3980-41e7-8765-89b022f6e827	118.434165242075	2024-11-22 06:39:43	Completed	1

```
Execution finished without errors.
Result: 3304 rows returned in 71ms
At line 1:
SELECT *
FROM transactions
WHERE transaction_status = 'Completed';
```

Justification for Separate Tables:

By separating the data into different tables — customer, balance, and transaction — we achieve the following:

Reduced Redundancy: Avoid storing the same customer information repeatedly in each transaction or balance record.

Efficient Data Management: Update customer details, balances, or transactions independently without disrupting the structure of other data.

Maintaining Data Integrity: Foreign keys enforce consistency, ensuring that transactions only reference valid customer and balance data.

Scalability: The database structure is scalable, allowing for future expansion (e.g., adding new account types or handling additional transaction attributes).

Security: Sensitive information can be isolated, allowing for better access control and data protection.

Conclusion

this database design provides a strong foundation for a real-world banking system, balancing efficiency, data integrity, and security while ensuring scalability for future needs. The clear separation of customer information, balances, and transactions allows for effective data management and better system performance. This structure is ideal for supporting the core operations of a banking system, ensuring it can grow and evolve.