Secured Transaction System
README

Objectives:

- **Implement a blockchain with the following functionalities:**

1. Create at least 10 nodes as miners who are connected to each other. Each miner is connected to at least 2 users.
2. Create a wallet which contains private and public keys.
3. Use the hash of the public key as the address of each user's wallet.
4. Using digital signatures verify the sender and receiver.
5. The header of a block in the blockchain should contain: block index, timestamp, previous block hash, and merkle root.
6. The body of the block should contain the hash of each transaction.
7. Store the transactions in UTXO format (w.r.t. Bitcoin wallet).

- **Execute and run the above-described Blockchain, then store the data (transactions) in the database and ask the following queries:**

1. Genesis transaction: find the (Genesis) block hash from the transaction hash.
2. Find the addresses and amounts of the transactions.
3. Show the block information of the block with the hash address of (input the hash of the block).
4. Show the height of the most recent block stored.
5. Show the most recent block stored.
6. The average number of transactions per block in the entire Bitcoin blockchain (in your database).
7. Show a summary report of the transactions in the block with height 6 with two columns
    A. "Number of transactions": numbers of transactions with inputs.
    B. "Total input Bitcoins": total inputs' BTC of transactions with this number of inputs.

Executions Steps:

1. There are following files: **miner-1.py (port 5001), miner-2.py (port 5002), user-1.py (port 5005), user-2.py (port 5006)**
2. Install the libraries mentioned in the import statements

```python
import datetime
import hashlib
import json
from textwrap import dedent
from time import time
from uuid import uuid4
from urllib.parse import urlparse
import requests
from flask import Flask, jsonify, request
import binascii
from typing import List
import typing

# following imports are required by PKI
import Crypto
import Crypto.Random
from Crypto.Hash import SHA
from Crypto.PublicKey import RSA
from Crypto.Signature import PKCS1_v1_5
```

3. Open the terminal and run the following command to start the server and the miner code runs at the port **5001**

   python miner-1.py

```
PS D:\hemani\acads\semester 7\blockchain\assignment-1\attempt-1> python miner-1.py
 * Serving Flask app 'miner-1' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on all addresses (0.0.0.0)
   WARNING: This is a development server. Do not use it in a production deployment.
 * Running on http://127.0.0.1:5001
 * Running on http://172.31.49.237:5001 (Press CTRL+C to quit)
```

4. Run the file named **miner-2.py** in a new terminal which has a different port No. **5002**

```
PS D:\hemani\acads\semester 7\blockchain\assignment-1\attempt-1> python miner-2.py
 * Serving Flask app 'miner-2' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on all addresses (0.0.0.0)
   WARNING: This is a development server. Do not use it in a production deployment.
 * Running on http://127.0.0.1:5002
 * Running on http://172.31.49.237:5002 (Press CTRL+C to quit)
```

5. Files for all the **10 miners** to demonstrate can be generated by copying the miner-1.py and changin the port number for each of the miners. Thus, **miner-1.py, miner-2.py, miner-3.py, miner-4.py, miner-5.py, miner-6.py, miner-7.py, miner-8.py, miner-9.py, miner-10.py** can be generated
6. Similary 20 files for **user-1.py** to **user-20.py** can be generated by changing the port number for each file of the user

• Create at least 10 nodes as miners who are connected to each other. Each miner is connected to at least 2 users

Files named **miner-1.py** to **miner-10.py** are run on different ports to demonstrate 10 miners
Files named **user-1.py** to **user-20.py** are run on different ports to demonstrate 20 users
Each user sends the transactions to the miner defined in their code and the miner then broadcasts the transaction to other connected miners

Methods supported for miners:

1. Mine new blocks
2. Add New Transactions
3. Register New Miner Nodes
4. Get the current chain
5. Resolve the chains by choosing the one with longer length
6. Broadcast the transaction received from the user to other miner nodes

Methods supported for users:

1. Add new transactions (which is sent to the intended miner)
2. Get current chain

● Create a wallet which contains private and public keys.
● Use the hash of the public key as the address of each user's wallet.

A Private key and Public Key is generated for every user (wallet) and the address of the wallet is generated by the Hash of the Public key

```
# Generate a Private Key
random = Crypto.Random.new().read
self.private_key = RSA.generate(1024, random)

# Generate Public Key
self.public_key = self.private_key.publickey()

# Use hash of Public Key as address of the wallet
wallet_address = hashlib.sha256(binascii.hexlify(self.public_key.exportKey(format='DER')).
decode('ascii').encode()).hexdigest()
#a73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d72
```

● Using digital signatures verify the sender and receiver.

Following functions are used to implement the digital signature mechanism in the code

```
def encrypt(rsa_publickey,plain_text):
    cipher_text=rsa_publickey.encrypt(plain_text,32)[0]
    b64cipher=base64.b64encode(cipher_text)
    return b64cipher

def decrypt(rsa_privatekey,b64cipher):
    decoded_ciphertext = base64.b64decode(b64cipher)
    plaintext = rsa_privatekey.decrypt(decoded_ciphertext)
    return plaintext

def sign(privatekey,data):
    return base64.b64encode(str((privatekey.sign(data,''))[0]).encode())

def verify(publickey,data,sign):
    return publickey.verify(data,(int(base64.b64decode(sign)),))
```

• The header of a block in the blockchain should contain: block index, timestamp, previous block hash, and merkle root.
• The body of the block should contain the hash of each transaction.

The **MerkleTree** Class has been implemented with all the necessary steps and the block has been defined as required. Current transactions (in other words MemPool is used to store the transactions that are yet to be added to the block)

```
block = {
    'index': len(self.chain) + 1,
    'timestamp': time(),
    'transactions': self.current_transactions,
    'proof': proof,
    'previous_hash': previous_hash or self.hash(self.chain[-1]),
    'merkle_root': MerkleTree(tmp).getRootHash()
}
```

• Store the transactions in UTXO format (w.r.t. Bitcoin wallet).

The transactions in UTXO format are stored in the list named **wallet** for all the user files

```
def __init__(self, miner):
    self.current_transactions = []
    self.chain = []
    self.miner = miner
    self.wallet = []
```

Assumptions

1. The blockchain is a simple list (storing all the blocks in our blockchain), there is another list to store the transactions for each of the miners as their own copy)
2. The blockchain class is responsible for managing the chain. It will store transactions and have some helper methods for adding new blocks to the chain.
3. Each block has an index, a timestamp, a list of transactions (or in simple terms, data), a proof (the nonce value), hash of the root of the merkle tree and the hash of the previous block
4. Proof of work has been used to implement the consensus algorithm

Next Steps:

- After Execution of above steps (the blockchain)
- Implement the database
- Store the data (transactions) in the database and
- Run the required queries

Database Implementation Steps:

1. Install MySQL
2. Open the MySQL Shell and follow the below instructions to create the database

```
MySQL Shell 8.0.31

Copyright (c) 2016, 2022, Oracle and/or its affiliates.
Oracle is a registered trademark of Oracle Corporation and/or its affiliates.
Other names may be trademarks of their respective owners.

Type '\help' or '\?' for help; '\quit' to exit.
 MySQL  JS > \sql
Switching to SQL mode... Commands end with ;
 MySQL  SQL > \connect root@localhost
Creating a session to 'root@localhost'
Fetching global names for auto-completion... Press ^C to stop.
Your MySQL connection id is 20 (X protocol)
Server version: 8.0.31 MySQL Community Server - GPL
No default schema selected; type \use <schema> to set one.
 MySQL  localhost:33060+ ssl  SQL > create database blockchainDB;
Query OK, 1 row affected (0.0032 sec)
 MySQL  localhost:33060+ ssl  SQL > use blockchainDB;
Default schema set to `blockchainDB`.
Fetching global names, object names from `blockchaindb` for auto-completion... Press ^C to stop.
 MySQL  localhost:33060+ ssl  blockchaindb  SQL > show tables;
Empty set (0.0014 sec)
```

```
 MySQL  localhost:33060+ ssl  blockchaindb  SQL > create table block (index_id int, hash varchar(255), timestamp varchar(255), transactions varchar(255), pr
oof varchar(255), previous_hash varchar(255), merkle_root varchar(255));
Query OK, 0 rows affected (0.0117 sec)
 MySQL  localhost:33060+ ssl  blockchaindb  SQL > show tables;
+----------------------+
| Tables_in_blockchaindb |
+----------------------+
| block                  |
+----------------------+
1 row in set (0.0014 sec)
 MySQL  localhost:33060+ ssl  blockchaindb  SQL > |
```

Storing the transactions into the Database:

- Obtaining the transactions from the APIs written in Assignment 1 and adding those transactions into the database:

- Structure of the Block as defined in MySQL Table:

```
4addf8976d72',  ' at line 1
MySQL  localhost:33060+ ssl  blockchaindb  SQL > SHOW COLUMNS FROM block;
+----------------+--------------+------+-----+---------+-------+
| Field          | Type         | Null | Key | Default | Extra |
+----------------+--------------+------+-----+---------+-------+
| index_id       | int          | YES  |     | NULL    |       |
| hash           | varchar(255) | YES  |     | NULL    |       |
| timestamp      | varchar(255) | YES  |     | NULL    |       |
| transaction_id | varchar(255) | YES  |     | NULL    |       |
| proof          | varchar(255) | YES  |     | NULL    |       |
| previous_hash  | varchar(255) | YES  |     | NULL    |       |
| merkle_root    | varchar(255) | YES  |     | NULL    |       |
+----------------+--------------+------+-----+---------+-------+
7 rows in set (0.0030 sec)
```

Adding transaction record to the MySQL database

```
MySQL  localhost:33060+ ssl  blockchaindb  SQL > insert into block values(0, 'a73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d72', '11:57:05', 'b73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d72',  'c732721
75a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d72', 'd73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d72', 'e73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d72');
Query OK, 1 row affected (0.0045 sec)
```

Adding all the transactions to the database:

```
Query OK, 1 row affected (0.0025 sec)
MySQL  localhost:33060+ ssl  blockchaindb  SQL > SELECT * FROM block;
| index_id | hash                                                             | timestamp | transaction_id                                                   | proof                                                            | previous_hash                                                    | merkle_root                                                      |
|        0 | a73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d72 | 11:57:05  | b73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d72 | c73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d72 | d73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d72 | e73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d72 |
|        1 | a73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d73 | 11:57:10  | b73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d73 | c73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d73 | d73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d73 | e73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d73 |
|        2 | a73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d44 | 11:57:20  | b73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d44 | c73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d44 | d73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d44 | e73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d44 |
|        3 | a73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d92 | 11:57:30  | b73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d92 | c73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d92 | d73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d92 | e73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d92 |
|        4 | a73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d58 | 11:57:40  | b73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d58 | c73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d58 | d73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d58 | e73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d58 |
|        4 | a73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d58 | 11:57:40  | b73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d58 | c73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d58 | d73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d58 | e73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d58 |
|        5 | a73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d97 | 11:57:50  | b73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d97 | c73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d97 | d73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d97 | e73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d97 |
|        6 | a73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d88 | 11:58:10  | b73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d88 | c73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d88 | d73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d88 | e73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d88 |
|        7 | a73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d22 | 11:58:20  | b73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d22 | c73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d22 | d73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d22 | e73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d22 |
|        8 | a73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d65 | 11:58:30  | b73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d65 | c73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d65 | d73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d65 | e73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d65 |
|        9 | a73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976401 | 11:58:40  | b73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976401 | c73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976401 | d73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976401 | e73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976401 |
|       10 | a73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976501 | 11:58:50  | b73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976501 | c73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976501 | d73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976501 | e73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976501 |
12 rows in set (0.0104 sec)
```

Run the Queries

1. Genesis transaction: find the (Genesis) block hash from the transaction hash

```sql
SELECT hash FROM block WHERE transaction_id = 'b73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d72';
```

- Output:

```
1 row in set (0.0033 sec)
MySQL  localhost:33060+ ssl  blockchaindb  SQL > SELECT hash FROM block WHERE transaction_id = 'b73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d72';
+------------------------------------------------------------------+
| hash                                                             |
+------------------------------------------------------------------+
| a73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d72 |
+------------------------------------------------------------------+
1 row in set (0.0018 sec)
```

2. Find the addresses and amounts of the transactions

- Display Addresses

```
SELECT hash FROM block;
```

- Output:

```
1 row in set (0.0018 sec)
MySQL localhost:33060+ ssl blockchaindb SQL > SELECT hash FROM block;
+------------------------------------------------------------------+
| hash                                                             |
+------------------------------------------------------------------+
| a73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d72 |
| a73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d73 |
| a73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d44 |
| a73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d92 |
| a73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d58 |
| a73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d58 |
| a73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d97 |
| a73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d88 |
| a73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d22 |
| a73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d65 |
| a73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976401 |
| a73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976501 |
+------------------------------------------------------------------+
12 rows in set (0.0025 sec)
```

- Display the amount of transactions (Assumption: One block contains only one transaction)

```
SELECT COUNT(*) FROM block;
```

- Output:

```
MySQL localhost:33060+ ssl blockchaindb SQL > SELECT COUNT(*) FROM block;
+----------+
| COUNT(*) |
+----------+
|       12 |
+----------+
```

3. Show the block information of the block with the hash address of (input the hash of the block)

```
SELECT * FROM block WHERE hash =
'a73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d73';
```

Output:

```
1 row in set (0.0226 sec)
MySQL  localhost:33060+ ssl  blockchaindb  SQL > SELECT * FROM block WHERE hash = 'a73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d73';
| index_id | hash | timestamp | transaction_id | proof | previous_hash | merkle_root |
|        1 | a73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d73 | 11:57:10 | b73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d73 | c73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d73 | d73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d73 | e73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976d73 |
1 row in set (0.0009 sec)
```

4. Show the height of the most recent block stored

```
SELECT index_id FROM block ORDER BY index_id DESC LIMIT 1;
```

Output:

```
MySQL  localhost:33060+ ssl  blockchaindb  SQL > SELECT index_id FROM block ORDER BY index_id DESC LIMIT 1;
+----------+
| index_id |
+----------+
|       10 |
+----------+
```

Here, Height = index_id

5. Show the most recent block stored

```
SELECT * from block ORDER BY index_id DESC LIMIT 1;
```

Output:

```
ERROR: 1054: Unknown column 'id' in 'order clause'
MySQL  localhost:33060+ ssl  blockchaindb  SQL > select * from block ORDER BY index_id DESC LIMIT 1;
| index_id | hash | timestamp | transaction_id | proof | previous_hash | merkle_root |
|       10 | a73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976501 | 11:58:50 | b73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976501 | c73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976501 | d73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976501 | e73272175a4c33f4ccc4bd6d565d565a824b90e576e8dbebc2634addf8976501 |
1 row in set (0.0017 sec)
```

6. The average number of transactions per block in the entire Bitcoin blockchain (in your database)

- Added a column named "Number of Transactions"

```
1 row in set (0.0048 sec)
 MySQL  localhost:33060+ ssl  blockchaindb  SQL > alter table block add no_of_transactions int;
Query OK, 0 rows affected (0.0317 sec)

Records: 0  Duplicates: 0  Warnings: 0
 MySQL  localhost:33060+ ssl  blockchaindb  SQL > SHOW COLUMNS FROM block;
+--------------------+--------------+------+-----+---------+-------+
| Field              | Type         | Null | Key | Default | Extra |
+--------------------+--------------+------+-----+---------+-------+
| index_id           | int          | YES  |     | NULL    |       |
| hash               | varchar(255) | YES  |     | NULL    |       |
| timestamp          | varchar(255) | YES  |     | NULL    |       |
| transaction_id     | varchar(255) | YES  |     | NULL    |       |
| proof              | varchar(255) | YES  |     | NULL    |       |
| previous_hash      | varchar(255) | YES  |     | NULL    |       |
| merkle_root        | varchar(255) | YES  |     | NULL    |       |
| no_of_transactions | int          | YES  |     | NULL    |       |
+--------------------+--------------+------+-----+---------+-------+
8 rows in set (0.0023 sec)
 MySQL  localhost:33060+ ssl  blockchaindb  SQL >
```

Assumption - In my case, I have added one transaction per block. Thus all the records have "no_of_transactions" value as **1**

```
ERROR: 1054: Unknown column 'no_of_transactions' in 'field list'
 MySQL  localhost:33060+ ssl  blockchaindb  SQL > update block set no_of_transactions=1 where index_id=0;
Query OK, 1 row affected (0.0043 sec)

Rows matched: 1  Changed: 1  Warnings: 0
 MySQL  localhost:33060+ ssl  blockchaindb  SQL > update block set no_of_transactions=1 where index_id=1;
Query OK, 1 row affected (0.0038 sec)

Rows matched: 1  Changed: 1  Warnings: 0
```

- **Display the Average of the number of transactions**

```
SELECT AVG(no_of_transactions) FROM block;
```

- Output:

```
 MySQL  localhost:33060+ ssl  blockchaindb  SQL > select avg(no_of_transactions) from block;
+-------------------------+
| avg(no_of_transactions) |
+-------------------------+
|                  1.0000 |
+-------------------------+
1 row in set (0.0009 sec)
 MySQL  localhost:33060+ ssl  blockchaindb  SQL >
```

7. Show a summary report of the transactions in the block with height 6 with two columns:

A. "Number of transactions": numbers of transactions with inputs.
B. "Total input Bitcoins": total inputs' BTC of transactions with this number of inputs.

- Added a column named "Coins"

```
1 row in set (0.0009 sec)
MySQL  localhost:33060+ ssl  blockchaindb  SQL > alter table block add coins int;
Query OK, 0 rows affected (0.0917 sec)

Records: 0  Duplicates: 0  Warnings: 0
MySQL  localhost:33060+ ssl  blockchaindb  SQL >
```

- The value of coins transacted for each of the transactions can be taken from the API defined in assignment 1, update the records accordingly

```
MySQL  localhost:33060+ ssl  blockchaindb  SQL > update block set coins=30 where index_id=0;
Query OK, 1 row affected (0.0137 sec)

Rows matched: 1  Changed: 1  Warnings: 0
MySQL  localhost:33060+ ssl  blockchaindb  SQL > update block set coins=40 where index_id=1;
Query OK, 1 row affected (0.0029 sec)

Rows matched: 1  Changed: 1  Warnings: 0
MySQL  localhost:33060+ ssl  blockchaindb  SQL > update block set coins=10 where index_id=2;
Query OK, 1 row affected (0.0031 sec)

Rows matched: 1  Changed: 1  Warnings: 0
MySQL  localhost:33060+ ssl  blockchaindb  SQL > update block set coins=10 where index_id=3;
Query OK, 1 row affected (0.0027 sec)
```

Display Summary as required:

```
SELECT no_of_transactions, coins FROM block WHERE index_id=6;
```

- Output

```
MySQL  localhost:33060+ ssl  blockchaindb  SQL > SELECT no_of_transactions, coins FROM block WHERE index_id=6;
+--------------------+-------+
| no_of_transactions | coins |
+--------------------+-------+
|                  1 |    30 |
+--------------------+-------+
1 row in set (0.0014 sec)
```