Sure, let's define the entities, attributes, and relationships based on the provided information.

Entities:

Transaction

Smart Contract

Address

Attributes for Transaction:

Txhash

Method

Blockno

DateTime (UTC)

From

From\_Nametag

To

To\_Nametag

Value

Txn Fee

Attributes for Smart Contract:

Txhash

Method

Blockno

DateTime (UTC)

From

From\_Nametag

To

To\_Nametag

Value

Txn Fee

Attributes for Address:

Unique Identifier

Relationships:

Transaction to Address: "Sent" or "Received" with the property being the amount transferred.

Transaction to Smart Contract: "Interaction" with the property being the type of interaction.

Smart Contract to Address: "Sent" or "Received" with the property being the amount transferred.

Transaction to Transaction: "Related" with properties specifying the type of relation.

Now, let's represent these entities and relationships as a graph in Python. We'll use the NetworkX library to create and visualize the graph. Let's start by importing the necessary libraries and creating the graph structure.

import networkx as nx

import matplotlib.pyplot as plt

# Create a directed graph

G = nx.DiGraph()

# Add nodes for entities

G.add\_node("Transaction", color='blue')

G.add\_node("Smart Contract", color='green')

G.add\_node("Address", color='red')

# Define attributes for each entity

transaction\_attributes = [

"Txhash", "Method", "Blockno", "DateTime (UTC)",

"From", "From\_Nametag", "To", "To\_Nametag",

"Value", "Txn Fee"

]

smart\_contract\_attributes = [

"Txhash", "Method", "Blockno", "DateTime (UTC)",

"From", "From\_Nametag", "To", "To\_Nametag",

"Value", "Txn Fee"

]

address\_attributes = ["Unique Identifier"]

# Add attributes to nodes

for attr in transaction\_attributes:

G.nodes["Transaction"][attr] = None

for attr in smart\_contract\_attributes:

G.nodes["Smart Contract"][attr] = None

for attr in address\_attributes:

G.nodes["Address"][attr] = None

# Define relationships

relationships = [

("Transaction", "Address", {"label": "Sent/Received", "property": "Amount Transferred"}),

("Transaction", "Smart Contract", {"label": "Interaction", "property": "Type of Interaction"}),

("Smart Contract", "Address", {"label": "Sent/Received", "property": "Amount Transferred"}),

("Transaction", "Transaction", {"label": "Related", "property": "Type of Relation"})

]

# Add relationships to the graph

for source, target, data in relationships:

G.add\_edge(source, target, \*\*data)

# Draw the graph

pos = nx.spring\_layout(G, seed=42)

nx.draw(G, pos, with\_labels=True, node\_color=[G.nodes[n]['color'] for n in G.nodes], node\_size=2000, font\_size=10)

edge\_labels = {(source, target): data['label'] for source, target, data in G.edges(data=True)}

nx.draw\_networkx\_edge\_labels(G, pos, edge\_labels=edge\_labels, font\_color='black')

# Show the graph

plt.show()

import networkx as nx

import matplotlib.pyplot as plt

# Sample data (replace with actual blockchain data)

transactions = [

{"Txhash": "tx1", "From": "addr1", "To": "addr2", "Value": 5.0},

{"Txhash": "tx2", "From": "addr2", "To": "contract1", "Value": 2.0},

# Add more transactions...

]

smart\_contracts = [

{"Txhash": "contract1", "From": "addr3", "To": "addr4", "Value": 10.0},

# Add more smart contracts...

]

addresses = [

{"Unique Identifier": "addr1"},

{"Unique Identifier": "addr2"},

{"Unique Identifier": "addr3"},

{"Unique Identifier": "addr4"},

# Add more addresses...

]

# Initialize an empty graph

G = nx.DiGraph()

# Add nodes for entities

for entity\_type, entity\_list in [("Transaction", transactions), ("Smart Contract", smart\_contracts), ("Address", addresses)]:

for entity in entity\_list:

G.add\_node(entity["Txhash"], type=entity\_type, \*\*entity)

# Define relationships

relationships = [

{"source": "Transaction", "target": "Address", "label": "Sent/Received", "property": "Amount Transferred"},

{"source": "Transaction", "target": "Smart Contract", "label": "Interaction", "property": "Type of Interaction"},

{"source": "Smart Contract", "target": "Address", "label": "Sent/Received", "property": "Amount Transferred"},

# Add more relationships...

]

# Add edges between nodes based on relationships

for rel in relationships:

source\_nodes = [node for node, data in G.nodes(data=True) if data['type'] == rel["source"]]

target\_nodes = [node for node, data in G.nodes(data=True) if data['type'] == rel["target"]]

for source\_node in source\_nodes:

for target\_node in target\_nodes:

G.add\_edge(source\_node, target\_node, label=rel["label"], property=rel["property"])

# Visualize the graph

pos = nx.spring\_layout(G, seed=42)

nx.draw(G, pos, with\_labels=True, node\_size=2000, font\_size=10)

edge\_labels = {(u, v): d['label'] for u, v, d in G.edges(data=True)}

nx.draw\_networkx\_edge\_labels(G, pos, edge\_labels=edge\_labels, font\_color='black')

# Show the graph

plt.show()

latest code for heterogenous code:

import pandas as pd

import networkx as nx

import matplotlib.pyplot as plt

# Load transaction data from CSV (replace 'your\_transaction\_data.csv' with actual file path)

transaction\_data = pd.read\_csv('your\_transaction\_data.csv')

# Create a directed graph

G = nx.DiGraph()

# Add nodes for entities

G.add\_node("Transaction", color='blue')

G.add\_node("Smart Contract", color='green')

G.add\_node("Address", color='red')

# Define attributes for each entity

transaction\_attributes = [

"Txhash", "Method", "Blockno", "DateTime (UTC)",

"From", "From\_Nametag", "To", "To\_Nametag",

"Value", "Txn Fee"

]

smart\_contract\_attributes = [

"Txhash", "Method", "Blockno", "DateTime (UTC)",

"From", "From\_Nametag", "To", "To\_Nametag",

"Value", "Txn Fee"

]

address\_attributes = ["Unique Identifier"]

# Add attributes to nodes

for attr in transaction\_attributes:

G.nodes["Transaction"][attr] = None

for attr in smart\_contract\_attributes:

G.nodes["Smart Contract"][attr] = None

for attr in address\_attributes:

G.nodes["Address"][attr] = None

# Define relationships

relationships = [

("Transaction", "Address", {"label": "Sent/Received", "property": "Amount Transferred"}),

("Transaction", "Smart Contract", {"label": "Interaction", "property": "Type of Interaction"}),

("Smart Contract", "Address", {"label": "Sent/Received", "property": "Amount Transferred"}),

("Transaction", "Transaction", {"label": "Related", "property": "Type of Relation"})

]

# Add relationships to the graph

for source, target, data in relationships:

G.add\_edge(source, target, \*\*data)

# Add nodes and edges based on transaction data

for \_, row in transaction\_data.iterrows():

# Add transaction node

G.add\_node(row["Txhash"], type="Transaction")

# Add sender address node and edge

G.add\_node(row["From"], type="Address", \*\*{"Unique Identifier": row["From"]})

G.add\_edge(row["From"], row["Txhash"], label="Sent", property="Amount Transferred")

# Add receiver address node and edge

G.add\_node(row["To"], type="Address", \*\*{"Unique Identifier": row["To"]})

G.add\_edge(row["Txhash"], row["To"], label="Received", property="Amount Transferred")

# Draw the graph

pos = nx.spring\_layout(G, seed=42)

nx.draw(G, pos, with\_labels=True, node\_color=[G.nodes[n]['color'] for n in G.nodes], node\_size=2000, font\_size=10)

edge\_labels = {(source, target): data['label'] for source, target, data in G.edges(data=True)}

nx.draw\_networkx\_edge\_labels(G, pos, edge\_labels=edge\_labels, font\_color='black')

# Show the graph

plt.show()