```python
from pymongo import MongoClient
import bson
import requests

URL = "https://blockchain.info"
LATEST_BLOCK = '/latestblock'
RAW_BLOCK = '/rawblock/'

client = MongoClient()

database = client['bitcoin-cluster']

Entity = database['Entity']
Blocks = database['Block']
Transaction = database['Transaction']
AddressEntity = database['AddressEntity']
control = database['control']
AddressChange = database['AddressChange']
count = 0


def updateLastBlock(lastBlock):
    try:
        control.update_one({"name": "lastblock"}, {"$set": {"value": lastBlock}})
        Blocks.insert(lastBlock)
    except Exception as e:
        print("Fail on update the last block ", e)
        raise e


def getTheLastBlock():
    lastBlockControl = control.find_one({
        "name": "lastblock"
    })

    if lastBlockControl is not None:
        lastBlock = lastBlockControl['value']
    else:
        control.insert({"name": "lastblock"})
        lastBlock = requests.get(URL + LATEST_BLOCK).json()
        lastBlock = requests.get(URL + RAW_BLOCK + lastBlock['hash']).json()
        updateLastBlock(lastBlock)

    return lastBlock


def populateTransactionsDatabaseWhenNecessary(lastBlock, MAX_POPULATION=1e6):
    print("doing count transactions")
    population = Transaction.count_documents({})
    print(population, " transactions!")
    while True:
        if population >= MAX_POPULATION:
            return
        try:
            print("Calling the last block...")
            actualBlock = requests.get(URL + RAW_BLOCK + lastBlock['prev_block']).json()
            Transaction.insert_many(actualBlock['tx'])
            population += len(actualBlock['tx'])
            updateLastBlock(actualBlock)
        except Exception as e:
            print("fail on insert transaction ", e)
            raise e

        lastBlock = actualBlock
        print("Actual Population: ", population)


def executeH1Clustering():
    print("Proccess transactions with h1: ")
    for transaction in Transaction.find():

        count transactions()

        addresses = get all address in transaction(transaction)

        entityToMerge = get all entity and remove address already in db(addresses)

        newEntityId = get new entity id(entityToMerge)

        update database addressEntity(addresses, entityToMerge, newEntityId)

def executeH2Clustering():
    change wallets = {}
    for transaction in Transaction.find():
        count transactions()

        addresses = get all address in transaction(transaction)

        if len(transaction['out']) > 1:
            first time = 0
            first address = None

            for output in transaction['out']:
                if output.get('addr') is not None and not (output['addr'] in addresses or output['addr'] in change wallets):
                    first address = output['addr']
                    first time += 1

            if first time == 1:
                add change wallet(addresses, first address)
                change wallets[first address] = True


def add change wallet(addresses, first address):
    if len(addresses):
        entity = AddressEntity.find one({"address": addresses[0]})
        if entity is not None:
            first time on db = AddressEntity.find one({"address": first address})
            if first time on db is None or not len(first time on db):
                AddressEntity.insert one({"address": first address, "entity": entity['entity']})

            if AddressChange.find one({"address": first address}) is None:
                AddressChange.insert one({"address": first address, "entity": entity['entity']})


def update database addressEntity(addresses, entityToMerge, newEntityId):
    # if merge, update new entityId
    if len(entityToMerge) > 1:
        AddressEntity.update many({"entity": {"$in": entityToMerge}}, {"$set": {"entity": newEntityId}})
    # insert all addresses without entity
    newAddressesEntity = [{'address': address, 'entity': newEntityId} for address in addresses]
    if len(newAddressesEntity):
        AddressEntity.insert many(newAddressesEntity)


def count transactions():
    global count
    if not count % 10000:
        print(count)
    count += 1


def get all address in transaction(transaction):
    addresses = []
    for inputAddress in transaction['inputs']:
        address = getAddress(inputAddress)
        if address is not None:
            addresses.append(address)
    addresses = list(set(addresses))
    return addresses


def get_all_entity_and_remove_address_already_in_db(addresses):
    addressesEntity = AddressEntity.find({"address": {"$in": addresses}})
    entityToMerge = []
    for addressEntity in addressesEntity:
        addresses.remove(addressEntity['address'])
        if addressEntity['entity'] not in entityToMerge:
            entityToMerge.append(addressEntity['entity'])
    return entityToMerge


def get_new_entity_id(entityToMerge):
    if len(entityToMerge) == 1:
        newEntityId = entityToMerge[0]
    else:
        newEntityId = bson.objectid.ObjectId()
    return newEntityId


def getAddress(inputAddress):
    address = None
    if inputAddress.get('prev') is not None:
        address = inputAddress['prev'].get('addr')
    if address is None and inputAddress.get('prev_out') is not None:
        address = inputAddress['prev_out'].get('addr')
    return address


def main():
    global count
    populateTransactionsDatabaseWhenNecessary(getTheLastBlock())
    count = 0
    executeH1Clustering()
    count = 0
    executeH2Clustering()

main()
```