In [1]:

```
from neo4j import GraphDatabase

class Neo4jDB:

    def close(self):
        if self.__driver is not None:
            self.__driver.close()

    def __init__(self, uri, user, pwd=''):
        self.__uri = uri
        self.__user = user
        self.__pwd = pwd
        self.__driver = None
        try:
            self.__driver = GraphDatabase.driver(self.__uri, auth=(self.__user, self.__pwd
            print("Connected to Neo4J driver ", self.__driver)
        except Exception as e:
            print("Failed to create the driver:", e)

    def query(self, query, parameters=None, db=None):
        assert self.__driver is not None, "Driver not initialized!"
        session = None
        response = None
        try:
            session = self.__driver.session(database=db) if db is not None else self.__dri
            response = list(session.run(query, parameters))
        except Exception as e:
            print("Query failed:", e)
        finally:
            if session is not None:
                session.close()
        return response

neoConn = Neo4jDB(uri="bolt://localhost:7687", user="")
```

```
Connected to Neo4J driver  <neo4j.BoltDriver object at 0x000001C28089A250>
```

In [2]:

```
import mysql.connector
from mysql.connector import Error

host = 'localhost'
schema = 'sakila'
user = 'root'
password = 'sembiran2009'

try:
    connection = mysql.connector.connect(host=host, database=schema, user=user, password=p

    if connection.is_connected():
        db_Info = connection.get_server_info()
        print("Connected to MySQL Server version ", db_Info)
        cursor = connection.cursor()
        cursor.execute("select database();")
        record = cursor.fetchone()
        print("You're connected to database: ", record)

except Error as e:
    print("Error while connecting to MySQL", e)
```

```
Connected to MySQL Server version  8.0.29
You're connected to database:  ('sakila',)
```

In [3]:

```python
import math
import mmh3
from bitarray import bitarray

class BloomFilter(object):

    '''
    Class for Bloom filter, using murmur3 hash function
    '''

    def __init__(self, items_count, fp_prob):
        '''
        items_count : int
            Number of items expected to be stored in bloom filter
        fp_prob : float
            False Positive probability in decimal
        '''
        # False possible probability in decimal
        self.fp_prob = fp_prob

        # Size of bit array to use
        self.size = self.get_size(items_count, fp_prob)

        # number of hash functions to use
        self.hash_count = self.get_hash_count(self.size, items_count)

        # Bit array of given size
        self.bit_array = bitarray(self.size)

        # initialize all bits as 0
        self.bit_array.setall(0)

    def add(self, item):
        '''
        Add an item in the filter
        '''
        digests = []
        for i in range(self.hash_count):

            # create digest for given item.
            # i work as seed to mmh3.hash() function
            # With different seed, digest created is different
            digest = mmh3.hash(item, i) % self.size
            digests.append(digest)

            # set the bit True in bit_array
            self.bit_array[digest] = True

    def check(self, item):
        '''
        Check for existence of an item in filter
        '''
        for i in range(self.hash_count):
            digest = mmh3.hash(item, i) % self.size
            if self.bit_array[digest] == False:

                # if any of bit is False then,its not present
                # in filter
                # else there is probability that it exist
                return False
        return True

    @classmethod
    def get_size(self, n, p):
        '''
        Return the size of bit array(m) to used using
```

```
        following formula
        m = -(n * lg(p)) / (lg(2)^2)
        n : int
            number of items expected to be stored in filter
        p : float
            False Positive probability in decimal
        '''
        m = -(n * math.log(p))/(math.log(2)**2)
        return int(m)

    @classmethod
    def get_hash_count(self, m, n):
        '''
        Return the hash function(k) to be used using
        following formula
        k = (m/n) * lg(2)

        m : int
            size of bit array
        n : int
            number of items expected to be stored in filter
        '''
        k = (m/n) * math.log(2)
        return int(k)
```

In [4]:

```
def check_bf(data_mysql, data_neo4j, bloomf):
    #print("================================================DATA==========================
    #print(data_mysql)
    #print("------------------------------------------------------------------------------
    #print(data_neo4j)
    #print("==============================================================================

    for item in data_mysql:
        bloomf.add(item)

    match = True
    for d in data_neo4j:
        if bloomf.check(d):
            if d not in data_mysql:
                print("'{}' is false positive (it does not actually in MySQL)!".format(d))
                match = False
            #else:
            #    print("'{}' is probably present!".format(d))
        else:
            print("'{}' is found in Neo4j but not in MySQL!".format(d))
            match = False

    return match
```

In [5]:

```
def _solve_fields(table, field_names):
    s = ''
    first = True
    for f in field_names:
        if first:
            first = False
        else:
            s += ','
        s +='{}.{}'.format(table,f)
    return s
```

In [8]:

```
def _read_csv(table, s, e):
```

```
    file = 'db.localhost/{}/{}.csv'.format(schema, table)
    #print("file: {}".format(file))
    return pd.read_csv (file, sep = s, encoding = e)
```

In [9]:

```
from pandas import DataFrame
import pandas as pd
import csv
import time

start_time = time.time()

q1 = ("SHOW TABLES FROM " + schema)
c1 = connection.cursor(dictionary=True, buffered=True)
c1.execute(q1)

table_list = c1.fetchall()
c2 = connection.cursor()
p = 0.01
df = pd.DataFrame(columns=['Table', 'Records', 'p', 'm', 'k', 'Time(mins)'])
for entry in table_list:
    st = time.time()
    _, table = entry.popitem()
    records = _read_csv(table, ';', 'utf-8')
    field_names = list(records.columns)
    fields = _solve_fields(table, field_names)

    data_mysql = []
    data_neo4j = []

    first = True
    for i, r in records.iterrows():
        s = ''.join(str(d) for d in r)
        s = s.replace('None','')
        s = s.replace('nan','')
        s = s.replace('.0','')
        data_mysql.append(s)

    q3 = "MATCH ({}:{}) RETURN {}".format(table, table.capitalize(), fields)
    #print(q3)
    result2 = neoConn.query(q3)
    for r in result2:
        s = ''.join(str(d) for d in [str(a) for a in r])
        s = s.replace('None','')
        s = s.replace('nan','')
        s = s.replace('.0','')
        data_neo4j.append(s)

    n = len(data_mysql)
    bloomf = BloomFilter(n,p)
    result3 = check_bf(data_mysql, data_neo4j, bloomf)
    if result3:
        print("====> All nodes in Neo4j and records in MySQL are matched!")
    else:
        print("====> Nodes in Neo4j and records in MySQL DO NOT matched!")

    et = time.time() - st
    mn = "{:.2f}".format(et / 60)

    df = df.append({'Table':table.capitalize(), 'Records':n, 'p':bloomf.fp_prob, 'm':bloom
    #print("Table: {}, Records: {}, Bloom Filter (m:{},p:{},k:{}), Time: {} mins".format(t

elapsed_time = time.time() - start_time
mins = "{:.2f}".format(elapsed_time / 60)
#st = time.strftime("%H:%M:%S", time.gmtime(elapsed_time))
```

```
print("Elapsed Time: {} mins".format(mins))
df
```

```
====> All nodes in Neo4j and records in MySQL are matched!
====> All nodes in Neo4j and records in MySQL are matched!
====> All nodes in Neo4j and records in MySQL are matched!
====> All nodes in Neo4j and records in MySQL are matched!
====> All nodes in Neo4j and records in MySQL are matched!
====> All nodes in Neo4j and records in MySQL are matched!
====> All nodes in Neo4j and records in MySQL are matched!
====> All nodes in Neo4j and records in MySQL are matched!
====> All nodes in Neo4j and records in MySQL are matched!
====> All nodes in Neo4j and records in MySQL are matched!
====> All nodes in Neo4j and records in MySQL are matched!
====> All nodes in Neo4j and records in MySQL are matched!
====> All nodes in Neo4j and records in MySQL are matched!
====> All nodes in Neo4j and records in MySQL are matched!
====> All nodes in Neo4j and records in MySQL are matched!
Elapsed Time: 0.39 mins
```

Out[9]:

	Table	Records	p	m	k	Time(mins)
0	Actor	200	0.01	1917	6	0.00
1	Address	603	0.01	5779	6	0.00
2	Category	16	0.01	153	6	0.00
3	City	600	0.01	5751	6	0.00
4	Country	109	0.01	1044	6	0.00
5	Customer	599	0.01	5741	6	0.00
6	Film	1000	0.01	9585	6	0.01
7	Film_actor	5462	0.01	52353	6	0.02
8	Film_category	1000	0.01	9585	6	0.00
9	Film_text	1000	0.01	9585	6	0.00
10	Inventory	4581	0.01	43909	6	0.02
11	Language	6	0.01	57	6	0.00
12	Payment	16049	0.01	153830	6	0.17
13	Rental	16044	0.01	153782	6	0.14
14	Staff	2	0.01	19	6	0.00
15	Store	2	0.01	19	6	0.00