# Homework 2

NetID: xm2074, jh7948
Name: Xiao Ma, Junru He

## 1. Design a program that can generate values based on a fractal probability distribution.

System: Mac
Tool: AQuery
Code to generate stocksymbols and trade table:

```
import sys
import math
import csv
import numpy as np
from tqdm import trange
from numpy import random

min_price = 50
max_price = 500
min_quantity = 100
max_quantity = 10000
max_symbols = 100000
max_trade_rows = 10000000

def gen_price(pre):
    choices = [pre-1,pre-2, pre-3, pre-4, pre-5, pre+1, pre+2, pre+3, pre+4, pre+5]
    price = random.choice(choices)  # choose from choices randomly

    if min_price <= price <= max_price:
        return price

    return gen_price(pre)


def gen(frac, n):
    p = list(range(1,n+1))
    random.shuffle(p) # reorder
    outvec = p

    while len(p) > 1:
        p = p[:math.floor(len(p) * frac)]
        # outvec = outvec + p  # very slow....
        outvec = np.concatenate((outvec, p))

    random.shuffle(outvec)
    return outvec
```

```
def gen_trade_table(stocksymbol, n):
    print("Generate trade table...... ")
    trade_table = list()
    dic = dict() # store the current price of this stocksymbol
    for time in trange(n):
        curr = random.choice(stocksymbol)
        quantity = random.randint(min_quantity, max_quantity)

        if curr in dic.keys():
            price = gen_price(dic[curr])
        else:
            price = random.randint(min_price, max_price)

        dic[curr] = price
        trade_table.append([curr, time+1, quantity, price])
    print("Trade table generated! ")
    return trade_table


def gen_file(trade_table, filename):
    print("Writing file......")
    with open(filename, 'w') as file:
        writer = csv.writer(file)
        writer.writerow(["stocksymbol", "time", "quantity", "price"])
        for i in trange(len(trade_table)):
            writer.writerow(trade_table[i])

    print('Writing file has finished!')


if __name__ == '__main__':
    if len(sys.argv) != 2:
        print("Usage:\n python3 trade.py <output_filename.csv>.")
        exit(0)
    stocksymbol = gen(0.3, 70002)
    print('the length of stocksymbol: ', len(stocksymbol))

    trade_table = gen_trade_table(stocksymbol, max_trade_rows)
    print('the length of trade_table: ', len(trade_table))
    gen_file(trade_table, sys.argv[1])
```

**a) Find the weighted (by quantity) average price of each stock over the time.**

```
DROP TABLE IF EXISTS trade
CREATE TABLE trade(stocksymbol INT, time INT, quantity INT, price INT)
LOAD DATA INFILE "trade.csv" INTO TABLE trade FIELDS TERMINATED BY ","
select stocksymbol, sum(quantity * price) / sum(quantity) as avg_price
from trade
assuming asc stocksymbol
group by stocksymbol
into outfile "q1a.csv"
fields terminated by ","
```

**b) Find the vector of 10 trade unweighted price moving averages.**

```
select stocksymbol, avgs(10, price) as unweighted_moving
from trade
assuming asc stocksymbol
group by stocksymbol
into outfile "q1b.csv"
fields terminated by ","
```

**c) Find the vector of 10 trade weighted moving averages per stock.**

```
select stocksymbol, avgs(10, quantity * price) / avgs(10, quantity) as
weighted_moving
from trade
assuming asc stocksymbol, asc time
group by stocksymbol
into outfile "q1c.csv"
fields terminated by ","
```

**d) Find the single best buy first/sell later trade you could have done on each stock.**

```
select stocksymbol, max(price - mins(price)) as max_profit
from trade
assuming asc stocksymbol
group by stocksymbol
```

## 2. Using rules of thumb to tune.

Systems: MySQL, AQuery in Mac
Dataset: trade.csv from the first question. tradeUniform.csv generated uniformly.

**Create Table:**

```
DROP TABLE IF EXISTS trade
CREATE TABLE trade (
stocksymbol INT,
time INT,
quantity INT,
price INT
);
LOAD DATA INFILE 'trade.csv' INTO TABLE trade fields terminated by ',';

DROP TABLE IF EXISTS tradeUniform
CREATE TABLE tradeUniform (
stocksymbol INT,
time INT,
quantity INT,
price INT
);
LOAD DATA INFILE 'tradeUniform.csv'
INTO TABLE tradeUniform fields terminated by ',';
```

### Rule 1: Without redundant distinct

#### Using fractal data:

MySQL:

- Using “distinct”, the time of execution is **14.77**s.

```
select distinct time
from trade;
```

- Using “distinct”, the time of execution is **5.23**s.

```
select time
from trade;
```

AQuery2:

- Using “distinct”, the time of execution is **20.157** s

```
<sql>
select now();
select distinct time
from trade;
select now();
</sql>
<sql>
```

- Without “distinct” , the time of execution is **19.256** s

```
<sql>
select now();
select time
from trade;
select now();
</sql>
```

**Using Unifrom data:**

MySQL:

- Using “distinct”, the time of execution is **15.54**s.

```
select distinct time
from tradeUniform;
```

- Without “distinct” , the time of execution is **5.01**s.

```
select time
from trade;
```

AQuuery2:

- Using “distinct”, the time of execution is **19.12** s.

```
<sql>
select now();
select distinct time
from tradeUniform;
select now();
</sql>
```

- Without “distinct” , the time of execution is **15.165** s

```
<sql>
select now();
select time
from tradeUniform;
select now();
</sql>
```

### Rule 2: With Index

**Using fractal data:**

MySQL:

- Using “index”, the time of execution is **6.32** s.

```
create index price_stock_fractal on trade(price, stocksymbol);
select distinct stocksymbol
from trade;
```

- Without “index”, the time of execution is **6.01** s.

```
select distinct stocksymbol
from trade;
```

AQuery2:

- Using “index”, the time of execution is **0.264** s.

```
<sql>
select now();
create index price_stock_fractal on trade(price, stocksymbol);
select distinct stocksymbol
from trade;
select now();
</sql>
```

- Without “index”, the time of execution is **0.578** s.

```
<sql>
select now();
select distinct stocksymbol
from trade;
select now();
</sql>
```

**Using Unifrom data:**

MySQL:

- Using “index”, the time of execution is **9.1**s.

```
create index price_stock on trade(price, stocksymbol);
select distinct stocksymbol
from tradeUniform;
```

- Without “index”, the time of execution is **8.51**s.

```
select distinct stocksymbol
from tradeUniform;
```

AQuery2:

- Using “index”, the time of execution is **0.291**s.

```
<sql>
select now();
create index price_stock_uniform on trade(price, stocksymbol);
select distinct stocksymbol
from tradeUniform;
select now();
</sql>
```

- Without “index”, the time of execution is **0.14**s.

```
<sql>
select now();
select distinct stocksymbol
from tradeUniform;
select now();
</sql>
```

**Comment:**
We can see that for the two systems, the rules make the same imporvement. Also we can find that queires used on fractal distribution is always faster than that on uniform distribution.

## 3. Find friends.

**Code:**

```
set global local_infile = 1;
CREATE DATABASE adbdb;
use adbdb;
```

```sql
DROP TABLE IF EXISTS Likes, Friend;

CREATE TABLE Likes(person INTEGER, artist INTEGER);
CREATE TABLE Friend(person1 INTEGER, person2 INTEGER);

LOAD DATA LOCAL INFILE 'D:\HW2\p3\like.txt' INTO TABLE Likes
FIELDS TERMINATED BY ','
IGNORE 1 LINES;
LOAD DATA LOCAL INFILE 'D:\HW2\p3\friend.txt' INTO TABLE Friend
FIELDS TERMINATED BY ','
IGNORE 1 LINES;


DROP VIEW IF EXISTS tmp1, tmp2, tmp3, tmp4;

CREATE VIEW tmp1 AS
SELECT person1, person2, artist
FROM Likes
JOIN Friend
ON person2 = person;

CREATE VIEW tmp2 AS
SELECT person2, person1, artist
FROM Likes
JOIN Friend
ON person1 = person;

CREATE VIEW tmp3 AS
SELECT person1, person2, artist
FROM tmp1
WHERE not exists (
SELECT *
FROM Likes
WHERE Likes.person = person1 AND Likes.artist = tmp1.artist
);

CREATE VIEW tmp4 AS
SELECT person2, person1, artist
FROM tmp2
WHERE not exists (
SELECT *
```

```
FROM Likes
WHERE Likes.person = person2 AND Likes.artist = tmp2.artist
);

SELECT *
FROM tmp3
UNION
SELECT *
FROM tmp4;
```

**Result:**

```
+---------+---------+--------+
| person1 | person2 | artist |
+---------+---------+--------+
|       1 |     386 |    170 |
|       1 |     386 |    176 |
|       1 |     386 |    280 |
|       1 |     386 |    282 |
|       1 |     386 |    283 |
|       1 |     416 |     81 |
|       1 |     416 |     92 |
|       1 |     416 |    150 |
|       1 |     416 |    163 |
|       1 |     416 |    255 |
|       1 |     792 |     54 |
|       1 |     792 |     56 |
|       1 |     792 |    123 |
|       1 |     792 |    145 |
|       1 |     792 |    161 |
|       1 |     792 |    233 |
|       1 |     792 |    294 |
|       1 |     856 |    139 |
|       1 |     856 |    180 |
|   30000 |   26941 |    199 |
|   30000 |   27011 |     72 |
|   30000 |   27011 |    126 |
|   30000 |   27011 |    144 |
|   30000 |   27011 |    268 |
|   30000 |   27011 |    292 |
|   30000 |   27947 |    196 |
|   30000 |   27947 |    279 |
|   30000 |   27947 |    296 |
|   30000 |   27947 |    299 |
+---------+---------+--------+
7308105 rows in set (46.71 sec)
```