# Bài Tập Buổi 4

# Phần 1: Viết chương trình Python sinh 5 triệu bản ghi (record) ngẫu nhiên và insert vào DB Mysql. Bản ghi có dạng: ( MSV, HovaTen, Lop, HocPhi, GioiTinh, NgayNop, HocBong, KyHoc )

# Mục Tiêu : Học về Random Data và Đa luồng trong Python

## 1.Hàm Random

- Thư viện random trong Python cung cấp một loạt các hàm để tạo ra các số ngẫu nhiên, cũng như các lựa chọn ngẫu nhiên từ các chuỗi hoặc danh sách. Đây là một trong những thư viện cơ bản và hữu ích nhất trong Python, đặc biệt khi cần thực hiện các thao tác liên quan đến xác suất, mô phỏng và thử nghiệm.
- Thư viện random trong Python cung cấp nhiều công cụ mạnh mẽ để làm việc với số ngẫu nhiên và ngẫu nhiên hóa các phần tử trong danh sách. Các hàm của nó rất dễ sử dụng và có thể ứng dụng trong nhiều lĩnh vực khác nhau như xác suất, thống kê, trò chơi, và mô phỏng. Như trong ví dụ này là chúng ta sử dụng hàm trong việc tạo ngẫu nhiên các Data để chèn vào MySQL

Các thư viện để thực hiện đề bài trên mà chúng ta dùng đó là :

```
from random import random
import mysql.connector
import random
from faker import Faker
import time  # Thêm thư viện time để đo thời gian
```

Tất nhiên để chèn dữ liệu vào MySQL thì chúng ta phải có các thao tác cơ bản để tạo kết nối giữa mysql và python :

```
connectmysql = mysql.connector.connect(
    host="localhost",
    user ="root",
    password="Phuc00000@",
    database="dachsachsv",
)
print(connectmysql)

# Tạo con trỏ để thao tác với MySQL
mycursor = connectmysql.cursor()

# Tạo bảng nếu chưa tồn tại
mycursor.execute("""
    CREATE TABLE IF NOT EXISTS thongtinsv    (        masv VARCHAR(20) PRIMARY KEY,        hoten VARCHAR(50),        lop VARCHAR(50),
gioitinh VARCHAR(10),        hocphi FLOAT,        hocbong FLOAT,        ngaynop DATE,        kyhoc VARCHAR(10)    )""")
```

**Đây là phần tạo dữ liệu bằng cách sử dụng hàm random :**

```
existing_msvs = set()
def random_thongtin():
    global existing_msvs
    while True:
        masv = f"MSV_{random.randint(1000000, 9999999)}"
        if masv not in existing_msvs:
            existing_msvs.add(masv)
            hoten = faker.name()  # Tạo tên sinh viên
            lop = random.choice(['CNTT1', 'CNTT2', 'CNTT3', 'HTTMDT', 'DL1', 'DL2', 'QTANM1', 'QTANM2', 'KTMT', 'KTPM'])  # Chọn lớp ngẫu nhiên
            gioitinh = random.choice(['Nam', 'Nữ'])  # Chọn giới tính ngẫu nhiên
            hocphi = round(random.uniform(100000, 200000), 2)  # Tạo học phí ngẫu nhiên
            hocbong = 100000 if random.random() < 0.10 else 0  # Tạo 10% sinh viên có học bổng
            ngaynop = random.choice(['2024-01-15', '2024-02-01'])
```

```
        kyhoc = '2024-1'
        return (masv, hoten, lop, gioitinh, hocphi, hocbong, ngaynop, kyhoc)
```

Như chúng ta đã thấy việc tạo ra các thông tin ngẫu nhiên bằng cách sử dụng hàm random rất đơn giản. Sau khi tạo thông tin và phần còn lại là chèn 5 triệu bản ghi đã tạo vào trong MySQL với mỗi lần chèn là 10 nghìn bản ghi

```
# Hàm chèn dữ liệu vào bảng MySQL
def insert_records_to_db(record_list):
    try:
        sql = """INSERT IGNORE INTO thongtinsv (masv, hoten, lop, gioitinh, hocphi, hocbong, ngaynop, kyhoc)
        VALUES (%s, %s, %s, %s, %s, %s, %s, %s)"""
        mycursor.executemany(sql, record_list)
        connectmysql.commit()
        print(f"{len(record_list)} thông tin đã được chèn vào thành công")
    except Exception as e:
        print(f"Lỗi: {e}")
        connectmysql.rollback()
batch_size = 10000
total_records = 5000000
# Tắt các khóa chỉ mục tạm thời để tăng tốc độ chèn
mycursor.execute("ALTER TABLE thongtinsv DISABLE KEYS")
try:
    for i in range(0, total_records, batch_size):
        records = []
        for _ in range(batch_size):
            record = random_thongtin()
            records.append(record)

        # Đo thời gian chèn mỗi batch
         start_time = time.time()
        insert_records_to_db(records)  vào DB
        elapsed_time = time.time() - start_time
        print(f"Batch từ {i} đến {i + batch_size} đã được xử lý trong {elapsed_time:.2f} giây")
```

```
finally:
    # Kích hoạt lại các khóa sau khi chèn xong
    mycursor.execute("ALTER TABLE thongtinsv ENABLE KEYS")
    mycursor.close()
    connectmysql.close()
```

# 1.1 Thuật toán *Mersenne Twister* của Python

- Để tạo ra một số ngẫu nhiên trong khoảng từ (1-n) thì hàm Mersenne Twister bắt đầu bằng cách sinh ra một số ngẫu nhiên dạng số thực, thường nằm trong khoảng từ 0≤x<1, kết quả là một số ngẫu nhiên có độ phân phối đều
- Tiếp theo nó sẽ chuyển số thực thành số nguyên trong khoảng ta cần bằng công thức

```
 x=N/2^23
integer=floor(x×(b−a+1))+a

- x là số thực ngẫu nhiên trong khoảng [0, 1).
- aaa là giới hạn dưới (trong ví dụ của bạn là 1).
- bbb là giới hạn trên (trong ví dụ của bạn là 100).
```

## Thuật toán Mersenne Twister để sinh ra số thực trong khoảng 0≤x<1 dựa trên các bước sau

Thực chất không trực tiếp sinh ra số thực x trong khoảng 0≤x<1 từ đầu. Thay vào đó, nó sinh ra các số nguyên 32-bit ngẫu nhiên, sau đó những số nguyên này được chuyển đổi thành số thực nằm trong khoảng [0, 1)

### 1. Khởi tạo thật toán bằng seed

- Seed là một số nguyên dùng để khởi tạo bộ sinh số ngẫu nhiên. Và Seed có thể do mình khởi tạo, nếu bạn dùng cùng một Seed thì bạn sẽ thu được cùng môt dãy số nguyên
- Seed cũng có thể do Python tự động khởi tạo ra dựa trên giá trị thời gian (giây hoặc micro giây)

## 2. Tạo ma trận

- Ma trận trong **Mersenne Twister** có kích thước là 624 phần tử và mỗi phần tử là số nguyên 32-bit
- Khi sinh ra một số ngẫu nhiên thì thuật toán sẽ sử dụng số ngẫu nhiên đó để sinh ra các phần tử tiếp theo cho đế khi đủ 624 phần tử

## 3. Xoắn

- **Mersenne Twister** sẽ lấy một số phần tử từ ma trận trạng thái để sinh số ngẫu nhiên bằng thật toán của nó
- Sau khi xoắn được thực hiện thuật toán sẽ sinh ra số ngẫu nhiên từ ma trận, sau đó áp dụng một số phép toán bitwise để tăng tính ngẫu nhiên. Kết quả cuối cùng là số nguyên 32-bit

## 4. Áp dụng vào code

1.==masv = f"MSV_{random.randint(1000000, 9999999)}" ==

- Như đoạn code này sử dụng hàm random.randint để sinh mã sinh viên ngẫu nhiên, đây là hàm tạo ra số nguyên và đâu tiên trước khi sinh ra số nguyên ngẫu nhiên này thì nó cũng phải sinh ra một số ngẫu nhiên dạng số thực, nằm trong khoảng từ 0≤x<1, và sử dụng các công thức để tính ra số ngẫu nhiên
- **Công thức cho số nguyên**: Tạo ra số nguyên trong khoảng `[a, b]` lấy cả a và b

```
integer=floor(x×(b−a+1))+a


- x là số thực ngẫu nhiên trong khoảng [0, 1).
- aaa là giới hạn dưới (trong ví dụ của bạn là 1).
- bbb là giới hạn trên (trong ví dụ của bạn là 100).
```

2.lop = random.choice(['CNTT1', 'CNTT2', 'CNTT3', 'HTTMDT', 'DL1', 'DL2', 'QTANM1', 'QTANM2', 'KTMT', 'KTPM'])

- đây là một danh sách chúng ta hãy tạm gọi là len(danh sách) có 10 phần tử trong khoảng (0 đến len(danh sách) -1)

- để phát sinh số ngẫu nhiên Mersenne Twister biến đổi chuỗi thành trạng thái [0,9] và thực hiện các thuật toán để chọn ra các vị trí trong danh sách
  3.hocphi = round(random.uniform(100000, 200000), 2)
- Trong trường hợp này công thức tính ra số thực sẽ khác so với số nguyên
- **Công thức cho số thực**: Tạo ra số thực trong khoảng `[a, b)` lấy a và không lấy b

```
x=a+(b−a)×random(0,1)
```

# 1.2 thuật toán Linear Congruential Generator

LCG rất đơn giản, rất trực quan và dễ hiểu, sử dụng chỉ một hàm:

```
Xn+1=(aXn+c) mod m
Trong đó:

- m, 0<mm, 0<m: Mô đun, thường sẽ là một số đủ lớn, ví dụ 232,231−1,248,264232,231−1,248,264
- a, 0<a<ma, 0<a<m: Hằng số nhân _mutiplier_
- c, 0≤c<mc, 0≤c<m: Hằng số cộng thêm _increment_
- X0, 0≤X0<mX0, 0≤X0<m: seed, giá trị khởi tạo
```

Một số quy tắc cho việc chọn các tham số:

- c phải lớn hơn 0: Điều này giúp đảm bảo rằng dãy số có sự phân bố tốt hơn.
- a và m nên là số nguyên dương.
- a−1 phải chia hết cho tất cả các ước số của m: Điều này giúp đạt được chu kỳ tối đa.
- Nếu m là chẵn, thì a−1 cũng phải chia hết cho 4.
  Dãy số ngẫu nhiên có thể có chu kỳ tối đa là m. Nếu tham số được chọn đúng, LCG có thể tạo ra chu kỳ bằng mmm, nghĩa là tất cả các giá trị từ 0 đến m−1 đều có thể xuất hiện trước khi bắt đầu lặp lại.

# 1.3 Thuật toán Multiply with Carry

Multiply with Carry là thuật toán sinh số ngẫu nhiên giả để tạo ra một chuỗi số ngẫu nhiên

## Các bước của thuật toán

1. **Khởi tạo**: Bắt đầu với một giá trị hạt giống sss và một giá trị carry c.
2. **Nhân**: Ở mỗi lần lặp, nhân giá trị hạt giống hiện tại với một hằng số A.
3. **Cộng Carry**: Thêm giá trị carry từ bước trước vào kết quả của phép nhân.
4. **Cập nhật Carry**: Giá trị carry mới được tính từ kết quả (thường bằng cách lấy các bit cao hơn).
5. **Cập nhật Hạt giống**: Giá trị hạt giống mới trở thành các bit thấp hơn của kết quả.
6. **Xuất**: Số ngẫu nhiên giả được tạo ra là giá trị hạt giống mới

## Công thức

Nếu sns_nsn là hạt giống hiện tại và cnc_ncn là giá trị carry hiện tại, thì:

1. Tính toán:

   Tn=A×Sn+Cn
2. Cập nhật:
   - Giá trị carry mới: Cn+1 = [Tn/2^m] (m là số bit của carry)
   - Giá trị hạt giống mới: Sn+1 = Tn mod 2^m
   Trong MWC chúng ta sẽ có một giá trị **r**, gọi là *lag* của MWC. Và cũng giống như LCG, chúng ta cũng sẽ có mutiplier và mô đun, nhưng sẽ không còn *increment*, mà thay vào đó là một giá trị *carry*. Công thức sẽ như sau:

```
Xn=(AXn−r + Cn−1)  mod  b, n≥r
```

Trong đó, cũng giống như trên, **a** sẽ là mutiplier, và ở đây **b** sẽ là mô đun, thường là 232232. Điểm khác biệt là giá trị carry **c**, giá trị này sẽ được dùng để tính toán giá trị **x** tiếp theo. Công thức của **c** là:

```
cn=⌊axn−r+cn−1b⌋, n≥r
```

người ta thường chọn giá trị của a sao cho ab−1 là *Safe Prime*, tức ab−1 và ab/2−1 đều là nguyên tố, khi đó chu kì của MWC sẽ là ab/2−1

# 2.Đa luồng trong python

### 2.1 Luồng (thread) là gì ? Sự khác nhau giữa thread và process

- Thread là một đơn vị cơ bản trong CPU. Một luồng sẽ chia sẻ với các luồng khác trong cùng process về thông tin data, các dữ liệu của mình. Việc tạo ra thread giúp cho các chương trình có thể chạy được nhiều công việc cùng một lúc
- Process là quá trình hoạt động của một ứng dụng. Tiến trình (process)chứa đựng thông tin tài nguyên, trạng thái thực hiện của chương trình
- Thread là một bước điều hành bên trong một process. Luồng (thread) là một khối các câu lệnh (instructions) độc lập trong một tiến trình và có thể được lập lịch bởi hệ điều hành. Hay nói một cách đơn giản, Thread là các hàm hay thủ tục chạy độc lập đối với chương trình chính. Một process dĩ nhiên có thể chứa nhiều thread bên trong nó. Điểm quan trọng nhất cần chú ý là một thread có thể làm bất cứ nhiệm vụ gì một process có thể làm.

### 2.2 Đa luồng (Multithreading) là gì

- Một chương trình đa luồng chứa hai hoặc nhiều phần mà có thể chạy đồng thời và mỗi phần có thể xử lý tác vụ khác nhau tại cùng một thời điểm, để sử dụng tốt nhất các nguồn có sẵn, đặc biệt khi máy tính của bạn có nhiều CPU.
- Python cung cấp thread Module và threading Module để bạn có thể bắt đầu một thread mới cũng như một số tác vụ khác trong khi lập trình đa luồng. Mỗi một Thread đều có vòng đời chung là bắt đầu, chạy và kết thúc. Một Thread có thể bị ngắt (interrupt), hoặc tạm thời bị dừng (sleeping) trong khi các Thread khác đang chạy – được gọi là yielding.

Trong chương trình chúng ta chạy có thể sử dụng đa luồng để rút ngắn thời gian chèn dữ liệu vào trong MySQL, xử lý đồng thời nhiều tác vụ độc lập, Quản lý tài nguyên tốt hơn,...

Để dùng đa luồng cho chương trình chúng ta phải khởi tạo thư viện đa luồng, sau đó thực hiện các câu lệnh đa luồng

```
# Thư viện để sử dụng đa luồng
from concurrent.futures import ThreadPoolExecutor
# Sử dụng ThreadPoolExecutor để quản lý đa luồng
with ThreadPoolExecutor(max_workers=4) as executor:
    for i in range(0, total_records, batch_size):
        records = []  # Khởi tạo danh sách bản ghi mới
        for _ in range(batch_size):
            record = random_thongtin()
            records.append(record)

        # Đo thời gian chèn mỗi batch
        start_time = time.time()
        future = executor.submit(insert_records_to_db, records)
        result = future.result()  # Đợi kết quả của mỗi batch
        elapsed_time = time.time() - start_time
        print(f"Batch từ {i} đến {i + batch_size} đã được xử lý trong {elapsed_time:.2f} giây")
        time.sleep(1)  # Nghỉ một chút giữa các batch để giảm tải
```

## So sánh giữa việc dùng đa luồng và thủ công

### 1.Hiệu năng

- Đa luồng giúp chúng ta chia nhỏ công việc thành nhiều luồng để chạy song song tận dụng tốt hơn tài nguyên hệ thống. Có thể giúp **tăng tốc** quá trình chèn dữ liệu, đặc biệt là khi bạn có các batch dữ liệu lớn và muốn xử lý chúng đồng thời.
- Các làm thủ công là mọi việc sẽ được làm tuần tự. Tốc độ chèn sẽ bị giới hạn bởi tốc độ của từng thao tác tuần tự, dẫn đến việc tổng thời gian thực thi lâu hơn so với phương pháp đa luồng.

### Đây là thời gian khi dùng chạy thủ công

```
C:\Users\Admin\PycharmProjects\pythonProject1\.venv\Scripts\python
<mysql.connector.connection_cext.CMySQLConnection object at 0x0000
10000 thông tin đã được chèn vào thành công
Batch từ 0 đến 10000 đã được xử lý trong 7.08 giây
10000 thông tin đã được chèn vào thành công
Batch từ 10000 đến 20000 đã được xử lý trong 8.76 giây
10000 thông tin đã được chèn vào thành công
Batch từ 20000 đến 30000 đã được xử lý trong 10.08 giây
10000 thông tin đã được chèn vào thành công
Batch từ 30000 đến 40000 đã được xử lý trong 8.67 giây
10000 thông tin đã được chèn vào thành công
Batch từ 40000 đến 50000 đã được xử lý trong 9.21 giây
10000 thông tin đã được chèn vào thành công
Batch từ 50000 đến 60000 đã được xử lý trong 9.41 giây
10000 thông tin đã được chèn vào thành công
Batch từ 60000 đến 70000 đã được xử lý trong 10.41 giây
10000 thông tin đã được chèn vào thành công
Batch từ 70000 đến 80000 đã được xử lý trong 10.48 giây
```

**Đây là thời gian khi sử dụng đa luồng**

```
C:\Users\Admin\PycharmProjects\pythonProject1\.venv\Scripts\python.exe C:
10000 thông tin đã được chèn vào thành công
Batch từ 0 đến 10000 đã được xử lý trong 5.48 giây
10000 thông tin đã được chèn vào thành công
Batch từ 10000 đến 20000 đã được xử lý trong 6.44 giây
10000 thông tin đã được chèn vào thành công
Batch từ 20000 đến 30000 đã được xử lý trong 5.11 giây
10000 thông tin đã được chèn vào thành công
Batch từ 30000 đến 40000 đã được xử lý trong 5.99 giây
10000 thông tin đã được chèn vào thành công
Batch từ 40000 đến 50000 đã được xử lý trong 6.72 giây
10000 thông tin đã được chèn vào thành công
Batch từ 50000 đến 60000 đã được xử lý trong 4.44 giây
10000 thông tin đã được chèn vào thành công
Batch từ 60000 đến 70000 đã được xử lý trong 5.83 giây
10000 thông tin đã được chèn vào thành công
Batch từ 70000 đến 80000 đã được xử lý trong 5.11 giây
```

Tất nhiên là với số lượng ban ghi lớn thế này thì chúng ta không thể đòi hỏi thời gian nhanh hơn được nhưng chúng ta có thể thấy được tốc độ khi dùng đa luồng là nhanh hơn

- **Đa luồng** : thời gian (4-6) giây
- **Thủ công** : thời gian (7-10) giây

# Phần 2 :

- Viết Function lấy danh sách top10 học sinh có học phí cao nhất.
- Viết Function lấy ra doanh thu của nhà trường (doanh thu = học phí - học bổng).
- Viết Function lấy ra số lượng học sinh theo giới tính.
- Viết Function vẽ bar chart các số liệu trên.
- Tối ưu Database để có thể tìm kiếm nhanh nhất.

# Mục Tiêu : Học cách xử lý, tối ưu Database, Học về visualize Data.

# 1. Query thông thường theo yêu cầu của đề bài

### Viết Function lấy danh sách top10 học sinh có học phí cao nhất.

- Để thực hiện câu này chúng ta sẽ thực hiện tạo một hàm để truy vấn dữ liệu trong chương MySQL mà ta đã chèn trước đó

```
def top10_hocphi_cao():
    query = """
        SELECT masv, hoten, hocphi        FROM thongtinsv        ORDER BY hocphi DESC        LIMIT 10"""
    try:
        mycursor.execute(query)
        top_sinhvien = mycursor.fetchall()
        return top_sinhvien
    except mysql.connector.Error as err:
        print(f"lỗi:{err}")
        return []
# Gọi hàm và in ra top 10 học phí cao nhất
top_sinhvien = top10_hocphi_cao()
print("Danh sách 10 sinh viên có học phí cao nhất : ")
for i in top_sinhvien:
    print(f"MSV: {i[0]},họ và tên: {i[1]},học phí: {i[2]:,.2f}")
```

### Viết Function lấy ra doanh thu của nhà trường.

```
def tong_doanh_thu():
    query = """
            SELECT SUM(hocphi - hocbong) AS doanhthu
            FROM thongtinsv     """
        try:
                mycursor.execute(query)
                ketqua = mycursor.fetchone()
                doanhthu = ketqua[0]
```

```
            return doanhthu
        except mysql.connector.Error as err:
            print(f"Lỗi: {err}")
            return 0
# Gọi hàm và in ra doanh thu
doanhthu = tong_doanh_thu()
print(f"Tổng doanh thu của nhà trường là: {doanhthu:,.2f} VNĐ")
```

**Viết Function lấy ra số lượng học sinh theo giới tính.**

```
def dem_sv_theo_gioitinh():
    query = """
        SELECT gioitinh, COUNT(*) AS count        FROM thongtinsv        WHERE gioitinh IN ('Nam', 'Nữ')        GROUP BY gioitinh    """    try:
        mycursor.execute(query)
        ketqua = mycursor.fetchall()
        # Khởi tạo số lượng sinh viên nam và nữ
        dem_nam = 0
        dem_nu = 0
        # Lặp qua kết quả để xác định số lượng sinh viên nam và nữ
        for i in ketqua:
            if i[0] == 'Nam':
                dem_nam = i[1]  # Số lượng sinh viên nam
            elif i[0] == 'Nữ':
                dem_nu = i[1]  # Số lượng sinh viên nữ

        return dem_nam, dem_nu
    except mysql.connector.Error as err:
        print(f"Lỗi: {err}")
        return 0, 0  # Trả về 0 nếu có lỗi xảy ra

# Gọi hàm và in ra số lượng sinh viên nam và nữ
dem_nam,dem_nu = dem_sv_theo_gioitinh()
```

```
print(f"Số lượng sinh viên nam: {dem_nam}")
print(f"Số lượng sinh viên nữ: {dem_nu}")
```

Tuy nhiên khi thực hiện truy vấn như này chúng ta mới nhận ra rằng việc này quá lâu. Vì thế mới cần đến tối ưu Database để có thể tìm kiếm nhanh nhất

## 2.Tối ưu Database

- **Tối ưu hóa cơ sở dữ liệu (Database Optimization)** là quá trình cải thiện hiệu suất của cơ sở dữ liệu nhằm giảm thiểu thời gian truy xuất, tối ưu dung lượng lưu trữ, và đảm bảo rằng các truy vấn được thực hiện một cách nhanh chóng và hiệu quả.
- **Sử dụng chỉ mục (Indexes)**: Chỉ mục giúp tăng tốc độ truy xuất dữ liệu, đặc biệt là khi tìm kiếm, sắp xếp hoặc lọc dữ liệu. Tuy nhiên, việc tạo quá nhiều chỉ mục có thể làm chậm quá trình ghi dữ liệu (INSERT, UPDATE, DELETE).
- Index là 1 file riêng biệt được lưu trữ ở máy chủ và chỉ chứa những Fields mà bạn muốn nó chứa. Nếu bạn tạo 1 Index cho Field user_id (mã số người dùng), MySQL sẽ dễ dàng tìm ra được mã số 1 cách nhanh chóng. Ví dụ như quyển sách, khi cần tìm 1 thông tin, ta thường lật ngay tới phần “Mục Lục” và tìm từ đó để tăng tốc độ tìm. Và việc tạo ra Index này sẽ làm bạn thấy Database của bạn chạy nhanh 1 cách khác thường.

Sử dụng Index để tối ưu hóa Database, thự hiện truy vấn và so sánh thời gian khi không dùng Index
**Viết Function lấy ra doanh thu của nhà trường có sử dụng Index**
Ở phần này chúng ta phải tạo index cho từng cột học bổng, học phí để có thể truy vấn ra cột doanh thu một các nhanh chóng
Chúng ta cũng hãy kiểm tra index đã tồn tại hay chưa

```
def them_index_cho_hocphi_hocbong(mycursor):
    # Kiểm tra chỉ mục cho hocphi
    query_check_hocphi_index = """
        SHOW INDEX FROM thongtinsv WHERE Key_name = 'idx_hocphi';    """
    mycursor.execute(query_check_hocphi_index)
    index_hocphi_ton_tai = mycursor.fetchone()
    # Thêm chỉ mục cho hocphi nếu chưa tồn tại
    if not index_hocphi_ton_tai:
        query_index_hocphi = """
```

```
            ALTER TABLE thongtinsv
            ADD INDEX idx_hocphi (hocphi);        """
        try:
            mycursor.execute(query_index_hocphi)
            connectmysql.commit()
            print("Đã thêm INDEX cho cột 'hocphi'.")
        except mysql.connector.Error as err:
            print(f"Lỗi khi thêm chỉ mục cho hocphi: {err}")

    # Kiểm tra chỉ mục cho hocbong
    query_check_hocbong_index = """
        SHOW INDEX FROM thongtinsv WHERE Key_name = 'idx_hocbong';    """
    mycursor.execute(query_check_hocbong_index)
    index_hocbong_ton_tai = mycursor.fetchone()

    # Thêm chỉ mục cho hocbong nếu chưa tồn tại
    if not index_hocbong_ton_tai:
        query_index_hocbong = """
            ALTER TABLE thongtinsv
             ADD INDEX idx_hocbong (hocbong);        """
         try:
            mycursor.execute(query_index_hocbong)
            connectmysql.commit()
            print("Đã thêm INDEX cho cột 'hocbong'.")
        except mysql.connector.Error as err:
            print(f"Lỗi khi thêm chỉ mục cho hocbong: {err}")
```

**Viết Function đếm số lượng sinh viên theo giới tính có sử dụng index**

```
def them_index_cho_gioitinh(mycursor):
    query_check_index = """
        SHOW INDEX FROM thongtinsv WHERE Key_name = 'idx_gioitinh';    """
```

```
    mycursor.execute(query_check_index)
    index_ton_tai = mycursor.fetchone()
    if not index_ton_tai:  # Nếu chỉ mục chưa tồn tại
        query_index = """
            ALTER TABLE thongtinsv
            ADD INDEX idx_gioitinh (gioitinh);        """
             try:
            mycursor.execute(query_index)
            connectmysql.commit()
            print("Đã thêm INDEX cho cột 'gioitinh'.")
        except mysql.connector.Error as err:
            print(f"Lỗi khi thêm chỉ mục: {err}")
    else:
        print("INDEX 'idx_gioitinh' đã tồn tại.")
```

Sau khi tạo ra các index thì chúng ta thấy được việc thực hiện truy vấn sẽ hiện thị ra kết quả với thời gian nhanh hơn đáng kể

## So sánh giữa tối ưu hóa Database và chưa tối ưu

- **Tối ưu hóa database**: **Hiệu năng cao hơn** do các truy vấn được thực hiện nhanh chóng, giảm thiểu thời gian truy xuất dữ liệu. Cấu trúc bảng, chỉ mục, và các mối quan hệ giữa dữ liệu được thiết kế hợp lý để tối đa hóa tốc độ truy vấn, đặc biệt với các tập dữ liệu lớn. Cải thiện các truy vấn phức tạp và báo cáo bằng cách giảm thời gian xử lý.
- **Chưa tối ưu hóa**: **Hiệu năng thấp hơn**, đặc biệt khi số lượng dữ liệu tăng lên. Các truy vấn có thể trở nên rất chậm hoặc không phản hồi kịp thời. Các truy vấn có thể phải duyệt qua nhiều bản ghi không cần thiết do thiếu chỉ mục hoặc cấu trúc bảng không phù hợp. Tốc độ xử lý các tác vụ như insert, update, và delete cũng bị ảnh hưởng khi bảng không có chỉ mục hoặc thiếu các biện pháp tối ưu hóa.

## 3.Vẽ Bar Char cho các số liệu đã truy vấn ở trên

- **Visualize Data** (Trực quan hóa dữ liệu) là quá trình biến các dữ liệu phức tạp, số liệu, hoặc thông tin trừu tượng thành các biểu đồ, đồ thị, hoặc hình ảnh dễ hiểu. Mục tiêu của việc trực quan hóa dữ liệu là giúp con người dễ dàng nắm bắt, phân tích và giải thích thông tin một cách trực quan hơn so với việc chỉ đọc số liệu thô.

- Việc dùng Visualize Data giúp chúng ta dễ hiểu hơn, so sánh dữ liệu một cách dễ dàng hơn
- **Biểu đồ cột (Bar Chart)**: Giúp so sánh các nhóm dữ liệu với nhau.

Để có thể vẽ biểu đồ chúng ta hãy sử dụng thư viện

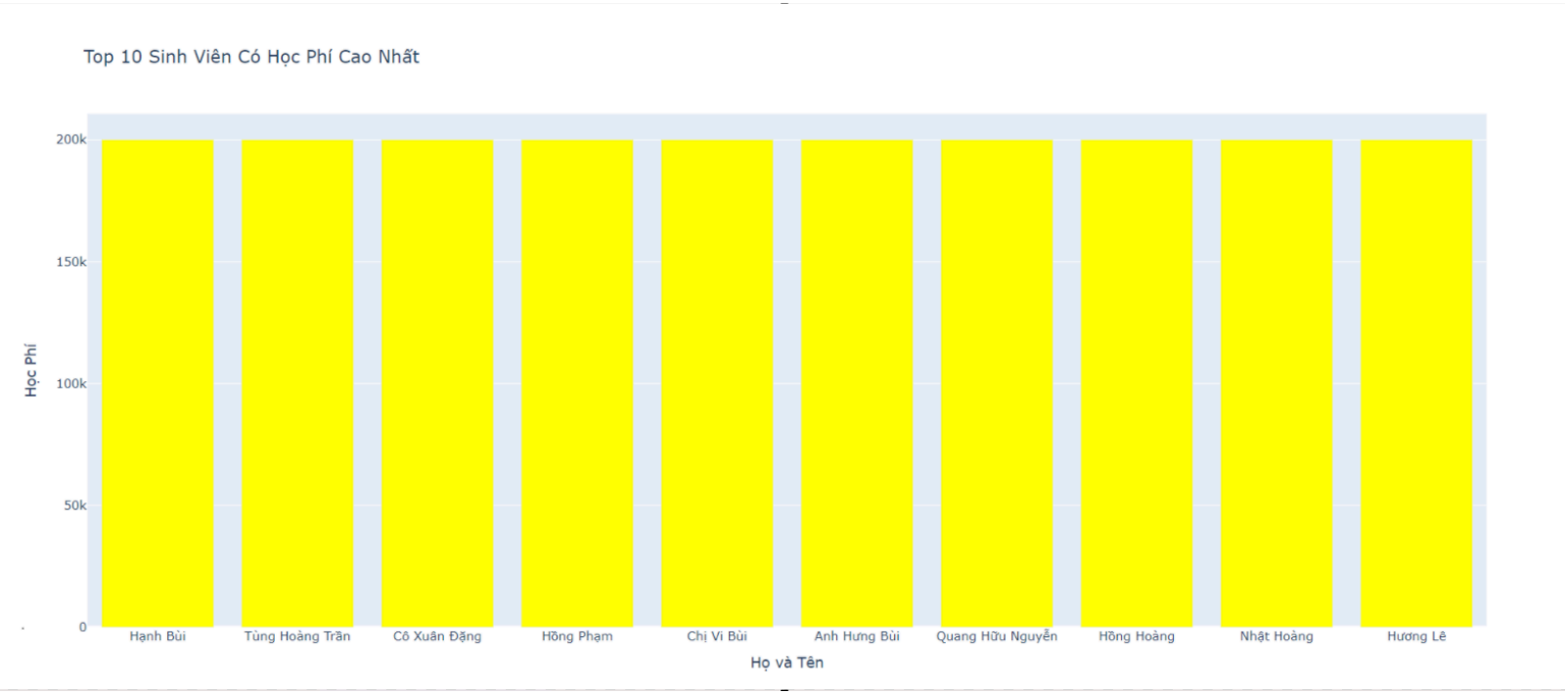
```
import plotly.graph_objects as go
```

### Vẽ biểu đồ thanh cho top 10 sinh viên có học phí cao nhất

```
# Tạo danh sách cho biểu đồ
msv_list = [sinh_vien[0] for sinh_vien in top_sinhvien]  # Danh sách mã sinh viên
hoten_list = [sinh_vien[1] for sinh_vien in top_sinhvien]  # Danh sách họ tên
hocphi_list = [sinh_vien[2] for sinh_vien in top_sinhvien]  # Danh sách học phí

# Vẽ biểu đồ thanh cho top 10 sinh viên có học phí cao nhất
fig = go.Figure(data=go.Bar(
    x=hoten_list,  # Sử dụng họ tên làm trục x
    y=hocphi_list,  # Sử dụng học phí làm trục y
    marker_color='yellow'
))

# Cài đặt tiêu đề và nhãn cho biểu đồ
fig.update_layout(
    title='Top 10 Sinh Viên Có Học Phí Cao Nhất',
    xaxis_title='Họ và Tên',
    yaxis_title='Học Phí',
)
# Hiển thị biểu đồ
fig.show()
```




**Vẽ biểu đồ so sánh Tính tổng doanh thu, tổng học phí và tổng học bổng**

```
fig = go.Figure(data=[
    go.Bar(name='Doanh thu', x=['Doanh thu'], y=[doanhthu], text=[doanhthu], textposition='auto',marker_color='lightblue'),
    go.Bar(name='Học phí', x=['Học phí'], y=[tong_hoc_phi], text=[tong_hoc_phi], textposition='auto',marker_color='lightgreen'),
    go.Bar(name='Học bổng', x=['Học bổng'], y=[tong_hoc_bong], text=[tong_hoc_bong], textposition='auto',marker_color='lightcoral')])

# Cài đặt tiêu đề và nhãn cho biểu đồ
fig.update_layout(
    title='So Sánh Doanh Thu, Học Phí và Học Bổng Của Nhà Trường',    xaxis_title='Danh Mục',    yaxis_title='Giá Trị (VNĐ)',    barmode='group'  # Thiết lập chế độ nhóm cho các thanh)

# Hiển thị biểu đồ
fig.show()
```


## Vẽ biểu đồ giới Tính

```
# Tạo biểu đồ về giới tính
fig = go.Figure(data=[
    go.Bar(
        name='Số lượng nam',
        x=['Số lượng nam'],
        y=[dem_nam],
        text=[dem_nam],  # Thêm số lượng nam làm nhãn
        textposition='auto',  # Đặt vị trí nhãn tự động
        marker_color='lightgreen'
    ),
    go.Bar(
        name='Số lượng nữ',
        x=['Số lượng nữ'],
```

```
        y=[dem_nu],
        text=[dem_nu],  # Thêm số lượng nữ làm nhãn
        textposition='auto',  # Đặt vị trí nhãn tự động
        marker_color='lightcoral'
    )
])
# Cập nhật tiêu đề và nhãn cho biểu đồ
fig.update_layout(
    title='So Sánh số lượng sinh viên Nam và sinh viên Nữ',
    xaxis_title='Danh Mục',
    yaxis_title='Số lượng sinh viên',  # Cập nhật nhãn y cho rõ ràng hơn
    barmode='group'
)
# Hiển thị biểu đồ
fig.show()
```