MỘT SỐ KỸ THUẬT TỐI ƯU HÓA TRUY VẤN DỮ LIỆU TRONG DJANGO ORM

Phạm Văn Tho [1], Phạm Khánh Bảo [1]

TÓM TẮT

Django là một Framework được phát triển bằng ngôn ngữ lập trình Python, hỗ trợ các lập trình viên phát triển ứng dụng nhanh chóng và hiệu quả. Một trong những tính năng quan trọng của Django là Django ORM, cho phép thao tác và quản lý dữ liệu thông qua mô hình hướng đối tượng mà không cần viết truy vấn SQL thủ công. Tuy nhiên, để cải thiện hiệu suất xử lý yêu cầu từ client, ứng dụng hoạt động mượt mà hơn và mang lại trải nghiệm tốt nhất cho người dùng, việc tối ưu hóa truy vấn cơ sở dữ liệu trong Django ORM là rất quan trọng. Trong bài viết này, chúng tôi sẽ trình bày một số kỹ thuật tối ưu hóa truy vấn dữ liệu sử dụng Django ORM.

Keyword: *Django Framework, Django ORM, tối ưu hóa truy vấn trong Django ORM.*

1. Đặt vấn đề

Tối ưu hóa truy vấn đến cơ sở dữ liệu (CSDL) [5] là công việc vô cùng quan trọng khi xây dựng ứng dụng. Việc tối ưu hóa truy vấn đến CSDL không chỉ giảm tác vụ tương tác đến CSDL, mà còn làm cho việc truy xuất đến CSDL nhanh hơn, tránh các sự cố không mong muốn như điểm chết, thiếu hụt tài nguyên dẫn đến những hậu quả nghiêm trọng. Thêm vào đó, tối ưu hóa truy vấn đến CSDL mang lại cho người dùng có trải nghiệm tốt hơn khi sử dụng ứng dụng.

Django ORM [4, 8] là kỹ thuật cho phép chúng ta truy vấn và quản lý dữ liệu thông qua sử dụng mô hình hướng đối tượng dựa trên nền tảng ngôn ngữ lập trình Python. Kỹ thuật này cho phép các lập trình viên tương tác đến CSDL một cách dễ dàng và thuận tiện mà không cần phải có kiến thức về ngôn ngữ SQL. Khi sử dụng kỹ thuật ORM trong Django, các lớp đối tượng được định nghĩa trong model sẽ được ánh xạ thành các bảng dữ liệu trong CSDL. Khi các đối tượng được tạo ra hoặc truy xuất, ORM sẽ tự động tạo hoặc tìm kiếm các bản ghi có trong CSDL và trả về đối tượng tương ứng. Django ORM mang lại nhiều lợi ích cho các nhà phát triển ứng dụng, thay vì viết các câu lệnh SQL phức tạp thực hiện thêm, sửa, xóa dữ liệu trong CSDL, chúng ta có thể sử dụng các phương thức đã được xây dựng sẵn trong Django ORM để thực hiện các thao tác này một cách đơn giản mà không cần phải có kỹ năng viết lệnh SQL. Việc thiết lập quan hệ giữa các bảng và truy vấn từ bảng dữ liệu này sang bảng dữ liệu khác sử dụng Django ORM thực hiện một cách dễ dàng thông qua khai báo khóa ngoại trong xây dựng lớp đối tượng trong model. Tuy nhiên, trong một số trường hợp, chúng ta cần thêm, cập nhật một lượng lớn dữ liệu, hoặc lợi dụng quan hệ giữa các bảng dữ liệu để truy vấn dữ liệu từ bảng này qua bảng khác làm tăng số lượng tương tác đến CSDL, tốc độ xử lý và hiệu năng ứng dụng giảm đáng kể. Do đó, khi phát triển ứng dụng với Django, chúng ta cần áp dụng đồng bộ các kỹ thuật tối ưu hóa truy vấn dữ liệu để ứng dụng đạt hiệu năng tốt nhất, cung

cấp cho người dùng trải nghiệm tốt nhất khi sử dụng ứng dụng.

Trong bài viết này, chúng tôi trình bày một số kỹ thuật tối ưu hóa truy vấn dữ liệu trong Django ORM. Trước tiên, chúng tôi sẽ giới thiệu tổng quan về truy vấn cơ sở dữ liệu trong Django ORM và lý do cần tối ưu hóa. Tiếp theo, bài viết sẽ trình bày các phương pháp tối ưu hóa giúp nâng cao hiệu suất xử lý dữ liệu, giảm tải cho hệ thống. Cuối cùng, chúng tôi sẽ đưa ra kết luận và một số khuyến nghị quan trọng để lập trình viên có thể sử dụng Django ORM một cách hiệu quả, tối ưu hóa truy vấn và mang lại trải nghiệm tốt hơn cho người dùng.

2. Một số kỹ thuật tối ưu hóa truy vấn dữ liệu trong Django ORM

Trong Django ORM, Queryset là bộ truy vấn lười biếng, nghĩa là nó chỉ thực hiện tương tác lên CSDL khi tập truy vấn được đánh giá bằng các phương thức như lặp, cắt, lưu vào bộ nhớ đệm và một số phương thức khác của Python như *len(), count().* Do đó, chúng ta có thể xếp chồng nhiều bộ lọc lên Queryset. Tuy nhiên, điều này làm tăng tải của hệ thống và tăng tương tác đến CSDL. Để tối ưu truy vấn đến CSDL trong Django ORM, chúng ta có thể sử dụng kết hợp nhiều kỹ thuật với nhau như đánh chỉ mục, sử dụng bộ nhớ đệm, sử dụng biểu thức F, tối ưu truy vấn đến bảng có chứa khóa ngoại, tối ưu thêm, cập nhật dữ liệu thông qua phương thức *bulk_create(), bulk_update()* để ứng dụng đạt được hiệu quả tốt nhất. Trong bài viết này, để mô phỏng các phương pháp tối ưu hóa truy vấn đến CSDL bằng Django ORM, chúng tôi giả sử ứng dụng có 3 lớp đối tượng được khai báo trong model như sau:

```
class Book(models.Model):
    name = models.CharField(max_length=200)
    date_created = models.DateField()
    price = models.FloatField(default=0)
    author = models.ManyToManyField('Author', related_name='books')
    categories = models.ForeignKey('Category',on_delete=models.CASCADE)
    def __str__(self):
        return self.name

class Author(models.Model):
    full_name = models.CharField(max_length=200)
    dob = models.DateField()
    def __str__(self):
        return self.full_name
class Category(models.Model):
    name = models.CharField(max_length=200)
    description = models.CharField(max_length=200)
    def __str__(self):
        return self.name
```

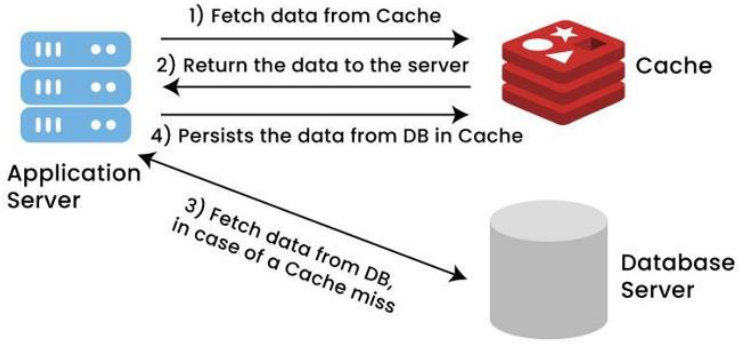
2.1. Đánh chỉ mục dữ liệu

Đánh chỉ mục [1] là kỹ thuật giúp tăng tốc độ tìm bản ghi từ CSDL. Đánh chỉ mục cho các trường sẽ giúp trình truy vấn tìm kiếm bản ghi cần tìm dễ dàng hơn mà không cần phải quét toàn bộ các bản ghi trong bảng, từ đó đạt hiệu suất tìm kiếm cao hơn. Trong Django, để đánh chỉ mục cho một trường nào đó của bảng dữ liệu, khi khai báo lớp đối tượng trong model, chúng ta thêm thuộc tính "*db_index=True*" vào thuộc tính cần đánh chỉ mục của lớp đối tượng, hoặc khai báo các thuộc tính đánh chỉ mục trong *class Meta* của lớp đối tượng.

Đánh chỉ mục các trường dữ liệu làm cho hệ quản trị CSDL xử lý nhiều, có thể làm giảm hiệu suất hệ thống. Do đó, khi lựa chọn trường đánh chỉ mục, chúng ta nên chọn bảng có số lượng lớn các bản ghi và trường thường xuyên truy vấn để đảm bảo hiệu suất truy vấn dữ liệu tốt nhất.

2.2. Sử dụng bộ nhớ đệm

Bộ nhớ đệm CSDL [6] là cách tiếp cận tốt nhất để nhận phản hồi nhanh nhất từ CSDL. Sử dụng bộ nhớ đệm CSDL sẽ làm giảm tương tác đến CSDL, giảm rủi ro và ngăn ngừa quá tải truy vấn dữ liệu.



Hình 1. *Mô hình sử dụng bộ nhớ đệm trong truy vấn dữ liệu [10]*

Django cung cấp một số thư viện để cấu hình bộ nhớ đệm như Memcached và Redis. Bộ nhớ đệm này giúp ứng dụng tránh chạy cùng một truy vấn nhiều lần để tương tác đến CSDL. Thêm nữa, khi mỗi Queryset được lấy ra và lưu vào trong bộ nhớ đệm, chúng ta có thể gọi lại mà không cần tương tác đến CSDL. Điều này làm tăng hiệu quả truy xuất dữ liệu của ứng dụng.

Ví dụ:

print([b.name for b in Book.objects.all()])

*print([b.*date_created *for e in Book.objects.all()])*

Đoạn mã trên sẽ thực hiện 2 truy vấn y hệt đến CSDL để lấy ra tất cả các đối tượng Book. Để tránh điều này, chúng ta tận dụng Queryset lưu trữ trong bộ nhớ đệm để gọi lại Queryset tương tự trước đó. Đoạn mã ví dụ trên được viết lại như sau:

```
queryset = Book.objects.all()
print([b.name for b in queryset])
print([b.date_created for b in queryset])
```

2.3. Sử dụng biểu thức F

Trong Django, biểu thức F [2, 9] được sử dụng để tham chiếu trực tiếp đến giá trị một trường nào đó của đối tượng trong model, và thực hiện một hành động lên trường tham chiếu mà không cần tìm nạp chúng từ CSDL vào bộ nhớ đệm. Trong một số trường hợp, chúng ta sử dụng biểu thức F thay cho các truy vấn tương tác đến CSDL để tăng hiệu năng hoạt động của ứng dụng.

Ví dụ: Giả sử chúng ta muốn tăng giá tất cả sách trong CSDL lên 20%, thì chúng ta sẽ làm như sau:

```
books = Book.objects.all()
for book in books:
  book.price *= 1.2
  book.save()
```

Với cách làm này, có quá nhiều tương tác đến CSDL thông qua vòng lặp *for,* làm giảm hiệu suất hệ thống ứng dụng. Do đó, để giảm tương tác đến CSDL trong trường hợp này, chúng ta có thể sử dụng biểu thức F với một truy vấn duy nhất đến CSDL để thực hiện yêu cầu. Đoạn mã trên được viết lại như sau:

```
from django.db.models import F
Book.objects.update(price=F('price') * 1.2)
```

2.4. Tối ưu truy vấn quan hệ giữa các bảng

Truy vấn dữ liệu đối với bảng có khóa ngoại [8], số lượng tương tác đến CSDL lớn, làm giảm hiệu quả truy vấn. Do đó, Django ORM cung cấp 2 phương thức để tối ưu hóa truy vấn trong trường hợp này là *select_related* và *prefetch_related*.

Hàm *select_related* hoạt động bằng cách JOIN các trường của các bảng liên quan vào một truy vấn. Vì vậy, *select_related* trả về kết quả truy vấn là các đối tượng liên quan trong cùng một truy vấn CSDL. *select_related* chỉ sử dụng cho quan hệ *one-to-one* hoặc quan hệ *one-to-many* ở bảng có khóa ngoại.

Ví dụ 1: Để lấy tất cả sách và danh mục sách trong CSDL, thông thường chúng ta thực hiện như sau:

```
queryset = Book.objects.all()
books = []
for book in queryset:
   books.append({'name': book.name, 'author': book.categories.name})
```

Kết quả phân tích truy vấn dữ liệu ở ví dụ 1 bằng công cụ Django Debug Toolbar như hình bên dưới.

Hình 2. *Kết quả phân tích truy vấn lấy tất cả sách và danh mục sách trong CSDL*

Từ kết quả phân tích trên, chúng ta thấy để lấy tất cả thông tin sách kết hợp với thông tin danh mục sách, hệ thống cần thực hiện 7 truy vấn (không bao gồm truy vấn liên quan đến quản lý session và truy vấn xác thực người dùng), trong đó có 6 truy vấn giống nhau và 5 truy vấn trùng nhau. Với dữ liệu mẫu có 6 bản ghi sách, 4 danh mục sách đã có 7 truy vấn đến CSDL. Như vậy, với trường hợp dữ liệu sách lớn thì số truy vấn đến CSDL là rất lớn. Do đó, để giảm số truy vấn đến CSDL, trong trường hợp này, bảng *book* có quan hệ *one-to-many* đến bảng *categories* qua khóa ngoại *categories* nên chúng ta sử dụng hàm *select_related* để tối ưu hóa truy vấn đến CSDL.

Trở lại ví dụ 1, chúng ta viết lại mã truy vấn dữ liệu sử dụng hàm *select_related* như sau:

```
queryset = Book.objects.select_related('categories').all()
books = []
for book in queryset:
    books.append({'name': book.name, 'author': book.categories.name})
```

Kết quả phân tích truy vấn dữ liệu ở ví dụ 1 sử dụng hàm *select_related* bằng công cụ Django Debug Toolbar như hình bên dưới.

Hình 3. *Kết quả phân tích truy vấn lấy tất cả sách và danh mục sách trong CSDL sử dụng hàm select_related*

Với kết quả này, khi sử dụng hàm *select_related* để lấy tất cả thông tin sách kết hợp với thông tin danh mục sách, hệ thống cần thực hiện 1 truy vấn (không bao gồm truy vấn liên quan đến quản lý session và truy vấn xác thực người dùng). Như vậy hàm *select_related* chỉ thực hiện một truy vấn duy nhất tương tác đến CSDL để lấy thông tin sách, đồng thời JOIN các bảng liên quan vào Queryset và lưu vào trong bộ nhớ đệm. Khi đó, để lấy thông tin liên quan đến *categories*, chúng ta chỉ cần truy vấn trực tiếp đến Queryset trong bộ nhớ đệm.

Tương tự như *select_related, prefetch_related* được sử dụng cho quan hệ *many-to-many* và *many-to-one*.

Ví dụ 2: Để lấy tất cả tác giả và sách mỗi tác giả viết trong CSDL, thông thường chúng ta thực hiện như sau:

```
queryset = Author.objects.all()
authors = []
for author in queryset:
    books = [book.name for book in author.books.all()]
    authors.append({'name': author.full_name, 'books': books})
```

Kết quả phân tích truy vấn dữ liệu ở ví dụ 2 bằng công cụ Django Debug Toolbar như hình bên dưới.

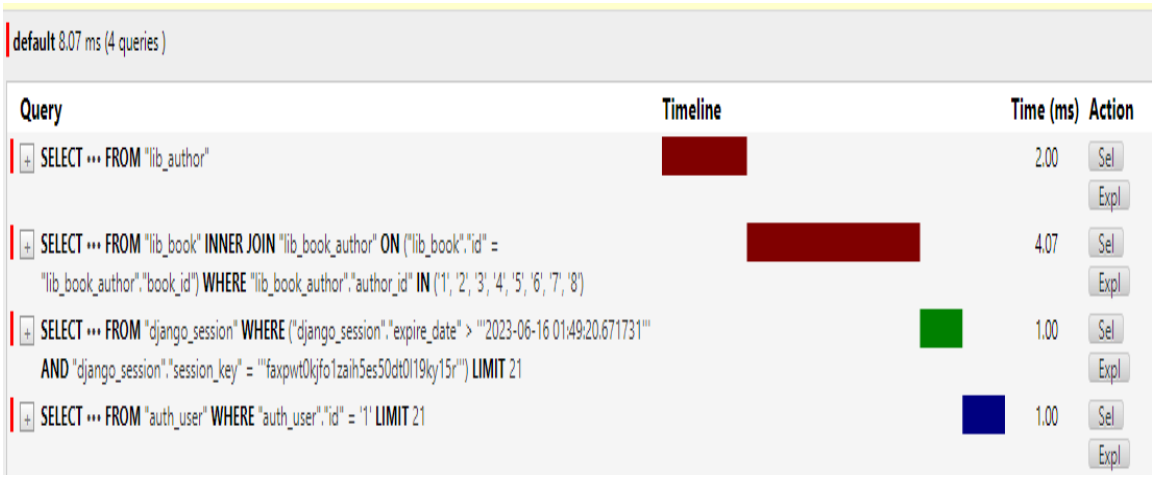
Hình 4. *Kết quả phân tích truy vấn lấy tất cả tác giả và sách tác giả viết trong CSDL*

Từ kết quả phân tích trên cho thấy hệ thống thực hiện 9 truy vấn đến CSDL để lấy tất cả thông tin tác giả và sách tác giả viết, trong đó có 8 truy vấn giống nhau với dữ liệu mẫu có 8 tác giả và 6 cuốn sách. Trường hợp dữ liệu tác giả và sách tác giả viết nhiều, số truy vấn tương tác đến CSDL là rất lớn. Do đó, trong trường hợp này để giảm số truy vấn tương tác đến CSDL, chúng ta sử dụng *prefetch_related* để tối ưu hóa truy vấn.

Trở lại ví dụ 2, để lấy tất cả tác giả và sách tác giả viết có trong CSDL, chúng ta sử dụng *prefetch_related* để tối ưu hóa truy vấn. Đoạn mã truy vấn được viết lại như sau:

```
queryset = Author.objects.prefetch_related('books').all()
authors = []
for author in queryset:
    books = [book.name for book in author.books.all()]
    authors.append({'name': author.full_name, 'books': books})
```

Kết quả phân tích truy vấn dữ liệu ở ví dụ 2 sử dụng hàm *prefetch_related* bằng công cụ Django Debug Toolbar như hình bên dưới.



Hình 5. *Kết quả phân tích truy vấn lấy tất cả tác giả và sách mỗi tác giả viết trong CSDL sử dụng hàm prefetch_related*

Từ kết quả này cho thấy, khi sử dụng hàm *prefetch_related*, để lấy tất cả thông tin tác giả và sách tác giả viết, hệ thống cần thực hiện 2 truy vấn (không bao gồm truy vấn liên quan đến quản lý session và truy vấn xác thực người dùng). Như vậy hàm *prefetch_related* chỉ thực hiện hai truy vấn tương tác đến CSDL để lấy tất cả thông tin tác giả, đồng thời JOIN các bảng liên quan vào Queryset và lưu vào trong bộ nhớ đệm. Với giải pháp này, hiệu năng xử lý hệ thống được cải thiện đáng kể, đồng thời hạn chế số lần tương tác trực tiếp đến CSDL.

2.5. Tối ưu thêm, cập nhật dữ liệu trong Django ORM

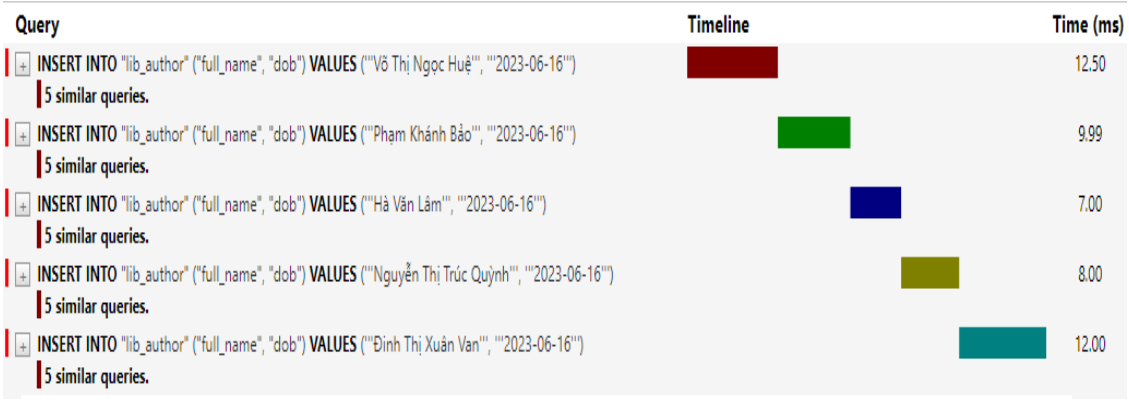
Django ORM cung cấp cho chúng ta một số phương thức thực hiện thêm và cập nhật dữ liệu một cách đơn giản [2, 7, 8]. Tuy nhiên, nếu chúng ta thực hiện các thao tác trên với nhiều bản ghi, thông qua vòng lặp thì có quá nhiều tương tác đến CSDL, làm cho hệ thống mất an toàn, giảm hiệu năng xử lý yêu cầu hệ thống. Do đó, để tối ưu hiệu năng hệ thống, chúng ta có thể sử dụng biểu thức F được trình bày trong phần 2.3, hoặc chúng ta có thể cập nhật nhiều bản ghi thực hiện trực tiếp trên bộ lọc *Queryset*.

Django ORM cung cấp hai phương thức mạnh mẽ giúp tối ưu hóa hiệu suất khi làm việc với nhiều bản ghi: *bulk_create()* và *bulk_update()*. Phương thức *bulk_create()* cho phép thêm nhiều bản ghi cùng lúc chỉ với một truy vấn duy nhất, giúp giảm đáng kể thời gian xử lý và số lần tương tác đến CSDL so với việc gọi phương thức *save()* cho từng đối tượng riêng lẻ. Trong khi đó, *bulk_update()* hỗ trợ cập nhật đồng thời nhiều bản ghi, hạn chế số lượng truy vấn SQL đến CSDL và cải thiện hiệu suất hệ thống.

Ví dụ 3: Khi muốn thêm 5 bản ghi sách vào CSDL, chúng ta có thể làm như sau:

```
Author.objects.create(full_name="Võ Thị Ngọc Huệ")
Author.objects.create(full_name="Phạm Khánh Bảo")
Author.objects.create(full_name="Hà Văn Lâm")
Author.objects.create(full_name="Nguyễn Thị Trúc Quỳnh")
Author.objects.create(full_name="Đinh Thị Xuân Van")
```

Kết quả phân tích truy vấn dữ liệu ở ví dụ 3 bằng công cụ Django Debug Toolbar như hình bên dưới.



Hình 6. *Kết quả phân tích truy vấn thêm riêng lẻ từng đối tượng vào CSDL*

Từ kết quả hình 6 cho thấy khi thêm 5 đối tượng vào CSDL, hệ thống phải thực hiện 5 lần tương tác riêng lẻ đến CSDL. Điều này có nghĩa là mỗi lần thêm một bản ghi, hệ thống sẽ gửi một truy vấn riêng biệt đến CSDL, dẫn đến việc tăng số lượng truy vấn SQL và làm giảm hiệu suất hệ thống.

Trở lại ví dụ 3, chúng ta thực hiện thêm toàn bộ 5 đối tượng vào CSDL bằng phương thức *bulk_create()* của Django ORM. Khi đó, hệ thống chỉ thực thi với một truy vấn duy nhất. Đoạn mã thêm 5 đối tượng Author ở ví dụ 3 được viết lại như sau:

```
Author.objects.bulk_create([
    Author(full_name="Võ Thị Ngọc Huệ"),
    Author(full_name="Phạm Khánh Bảo"),
    Author(full_name="Hà Văn Lâm"),
    Author(full_name="Nguyễn Thị Trúc Quỳnh"),
    Author(full_name="Đinh Thị Xuân Van")]
)
```

Kết quả phân tích truy vấn dữ liệu ở trên bằng công cụ Django Debug Toolbar như hình bên dưới.

Hình 7. *Kết quả phân tích truy vấn thêm đối tượng theo lô vào CSDL*

Từ kết quả phân tích truy vấn theo hình 7, chúng ta thấy khi sử dụng phương thức *bulk_create()* của Django ORM để thêm 5 đối tượng vào CSDL, hệ thống sử dụng một truy vấn duy nhất tương tác đến CSDL, thời gian phản hồi kết quả hết 5 ms trong khi thêm từng đối tượng riêng lẻ thời gian phản hồi hết 37,49 ms. Như vậy, hiệu suất hệ thống khi sử dụng phương thức *bulk_create()* được cải thiện đáng kể so với việc thêm

từng bản ghi riêng lẻ. Kỹ thuật này đặc biệt hữu ích khi cần xử lý số lượng lớn dữ liệu, giúp giảm độ trễ, tăng tốc xử lý và hạn chế tải cho hệ thống.

Tương tự như phương thức *bulk_create(),* Django ORM cung cấp phương thức *bulk_update()* để cập nhật hàng loạt đối tượng trong cơ sở dữ liệu chỉ với một truy vấn duy nhất, giúp tối ưu hiệu suất thay vì cập nhật từng đối tượng riêng lẻ. Khi sử dụng phương thức *save(),* mỗi lần cập nhật một bản ghi sẽ tạo ra một truy vấn SQL riêng, làm tăng số lượng tương tác đến CSDL, trong khi *bulk_update()* gom tất cả các bản ghi vào một truy vấn duy nhất, giảm tương tác đến CSDL và tối ưu tài nguyên hệ thống.

3. Kết luận

Django ORM là một công cụ quan trọng hỗ trợ lập trình viên dễ dàng tương tác với cơ sở dữ liệu thông qua đối tượng mà không cần hiểu sâu về câu lệnh SQL. Tuy nhiên, để đảm bảo tối ưu tài nguyên, an toàn truy cập, tối ưu hiệu suất xử lý hệ thống và cải thiện trải nghiệm người dùng, việc tối ưu hóa truy vấn là rất quan trọng. Điều này đòi hỏi lập trình viên phải giảm thiểu các truy vấn không cần thiết, hạn chế tương tác trực tiếp đến cơ sở dữ liệu, tối ưu tốc độ truy xuất dữ liệu và tránh tình trạng nghẽn trong quá trình xử lý. Vì vậy, khi phát triển ứng dụng với Django, việc áp dụng đồng bộ các kỹ thuật tối ưu hóa là cần thiết để hệ thống vận hành ổn định, hiệu quả và có khả năng mở rộng tốt hơn.

Trên cơ sở phân tích các phương pháp tối ưu hóa truy vấn trong Django ORM, chúng tôi đề xuất một số khuyến nghị nhằm nâng cao hiệu suất hệ thống khi phát triển ứng dụng với Django, cụ thể:

- Sử dụng *bulk_create()* và *bulk_update()* để chèn hoặc cập nhật nhiều bản ghi cùng lúc, giúp giảm số lượng truy vấn SQL, giảm tải hệ thống và tối ưu tốc độ xử lý dữ liệu.

- Thay vì lặp qua các bản ghi để kiểm tra số lượng hoặc sự tồn tại của một đối tượng trong CSDL, chúng ta sử dụng các phương thức *exists()* hoặc *count(),* đây là các phương thức của QuerySet, giúp giảm tương tác đến dữ liệu thông qua vòng lặp.

- Cấu hình và sử dụng bộ nhớ đệm để giảm tải truy vấn lặp lại vào cơ sở dữ liệu, cải thiện tốc độ và hiệu suất xử lý hệ thống.

TÀI LIỆU THAM KHẢO

[1]. António Esteves and João Fernandes, *Improving the Latency of Python-based Web Applications,* Centro ALGORITMI, School of Engineering, University of Minho, Campus de Gualtar, Braga, Portugal, 2019.

[2]. F. Bahia, Ensenada, Baja California, Mexico Daniel Rubio, *Web Application Development and Deployment with Python,* Pages 341-401, Springer Nature, 2017.

[3]. Carl Burch, Django, a web framework using Python: tutorial presentation, Journal of Computing Sciences in Colleges, Volume 25, Issue 5, 2010.

[4]. Songtao Chen, Shahed Ahmmed, Karu Lal, Chunhua Deming, *Django Web Development Framework: Powering the Modern Web,* International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE), 2023.

[5]. Silberschatz, Korth and Sudarshan, *Chapter 16: Query Optimization*, Database System Concepts, 7th Ed.

[6]. Django Software Foundation, 07-08-2024, *document describes Django 5.1*, https://docs.djangoproject.com/en/5.1/topics/cache.

[7]. Django Software Foundation, 07-08-2024, *document describes Django 5.1*, https://docs.djangoproject.com/en/5.1/topics/db/optimization.

[8]. Django Software Foundation, 07-08-2024, *document describes Django 5.1*, https://docs.djangoproject.com/en/5.1/ref/models/expressions.

[9]. Dharmesh Patel, September 13, 2022, *A Detailed View on Django ORM Optimization*, https://www.inexture.com/optimize-django-orm-queries.

[10]. Geeksforgeeks, 20-Dec-2024, *Caching – System Design Concept*, https://www.geeksforgeeks.org/caching-system-design-concept-for-beginners.

TECHNIQUES FOR OPTIMIZING DATA QUERIES IN DJANGO ORM

Pham Van Tho[1], Pham Khanh Bao[1]

ABSTRACT

Django is a framework developed by using the Python programming language, supporting programmers in developing applications quickly and efficiently. One of Django's important features is the Django ORM, which allows the manipulation and management of data through an object-oriented model without having to write manual SQL queries. However, in order to improve the performance of processing requests from the clients, to make the application run more smoothly, and to provide the best user experience, it is quite important to optimize database queries in Django ORM. In this article, we are going to present some techniques for optimizing data queries using Django ORM.

Keyword: *Django Framework, Django ORM, optimizing queries in Django ORM.*

[1]Khoa Công nghệ thông tin, Trường Đại học Phạm Văn Đồng.

Email: pvantho@pdu.edu.vn