Otimizando suas queries no Django ORM

grupy.rn

Quem nunca teve aquela query que demorava mais do que deveria?!

Em sistemas que exigem uma alta performance, uma query mal feita pode trazer muitos problemas

Latência alta

Alta utilização de recursos

Latência alta

Alta utilização de recursos

Latência alta

Bloqueios e deadlocks

Alta utilização de recursos

Latência alta

Bloqueios e deadlocks

Escalabilidade comprometida

Alta utilização de recursos

Latência alta

Bloqueios e deadlocks

Escalabilidade comprometida

Custo elevado

Alta utilização de recursos

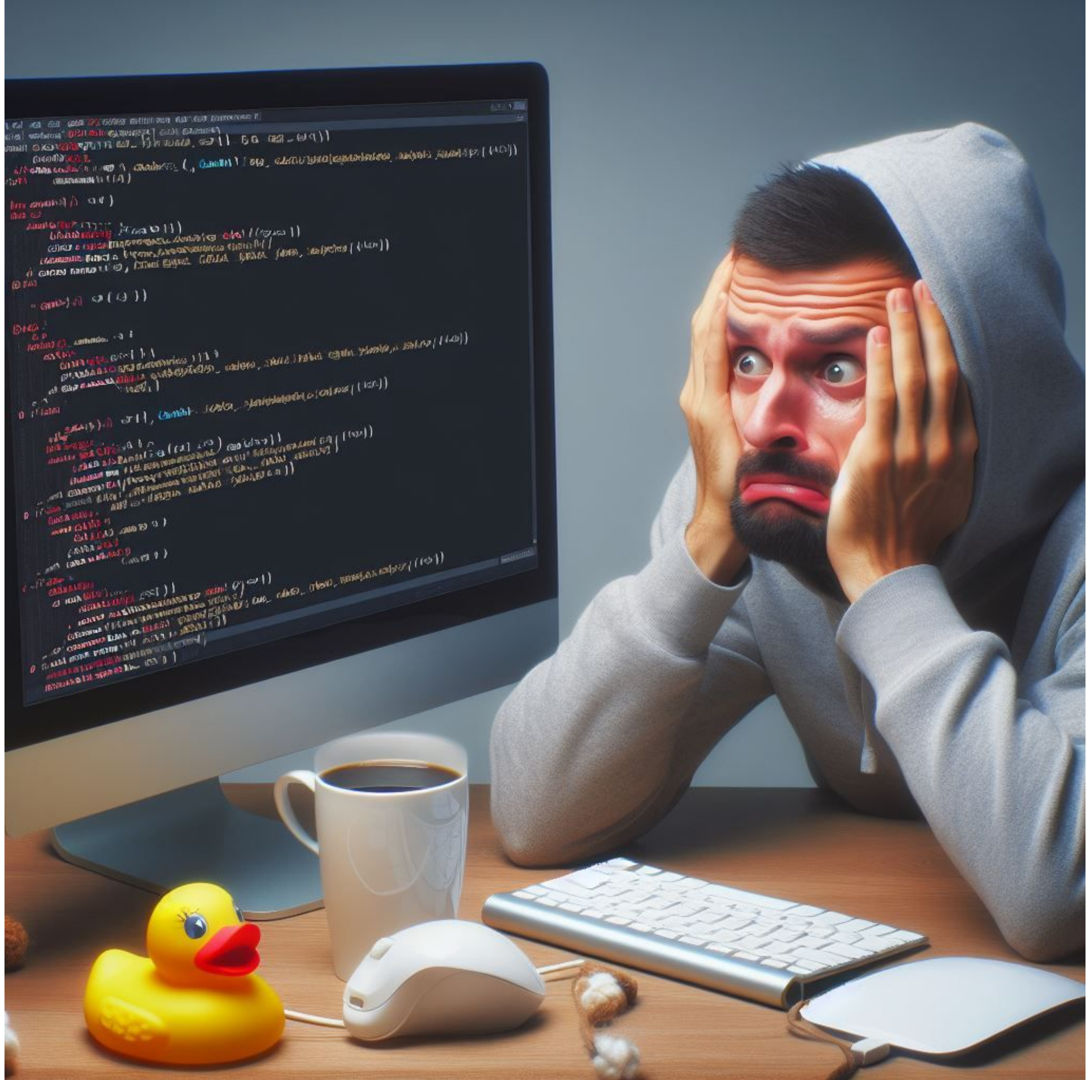
Latência alta

Bloqueios e deadlocks

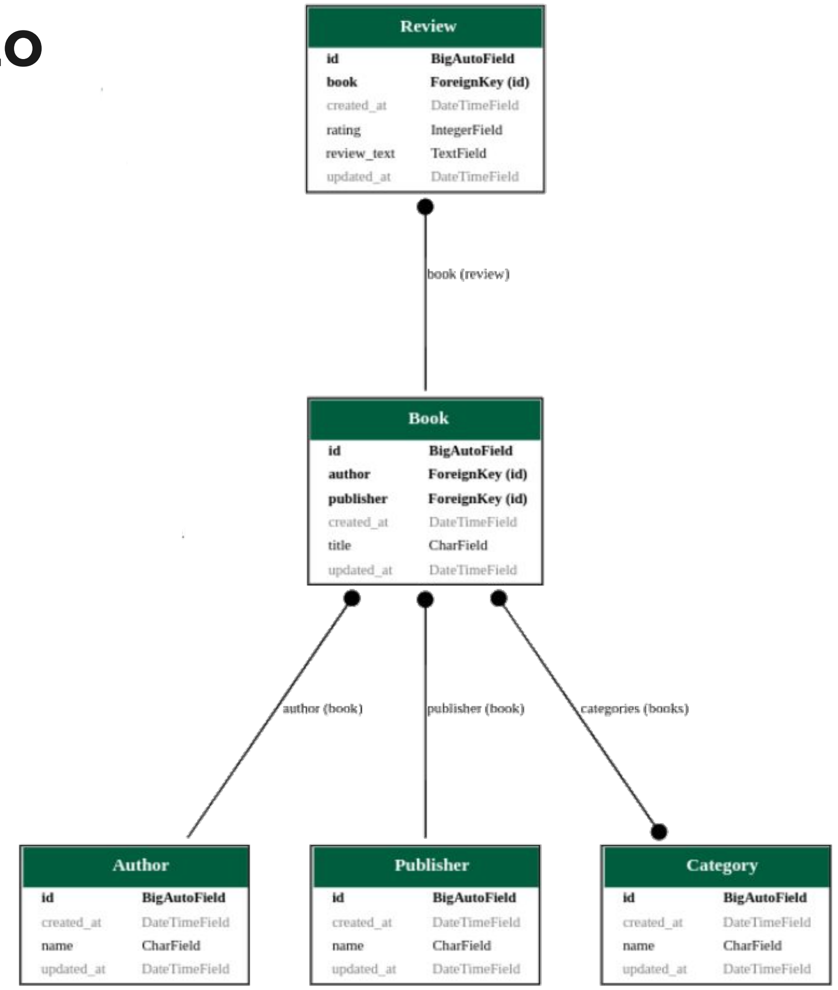
Escalabilidade comprometida

Custo elevado

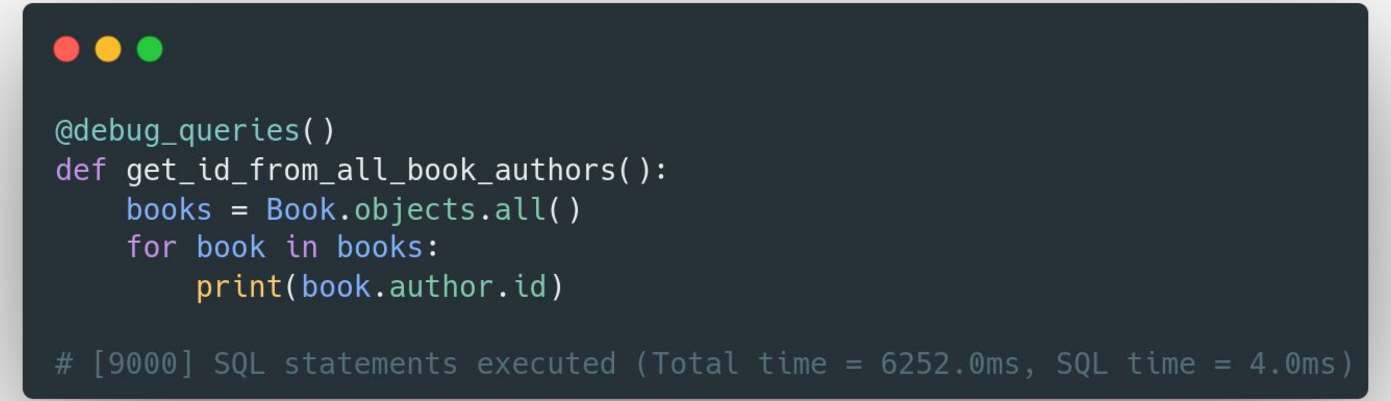
Impacto na UX



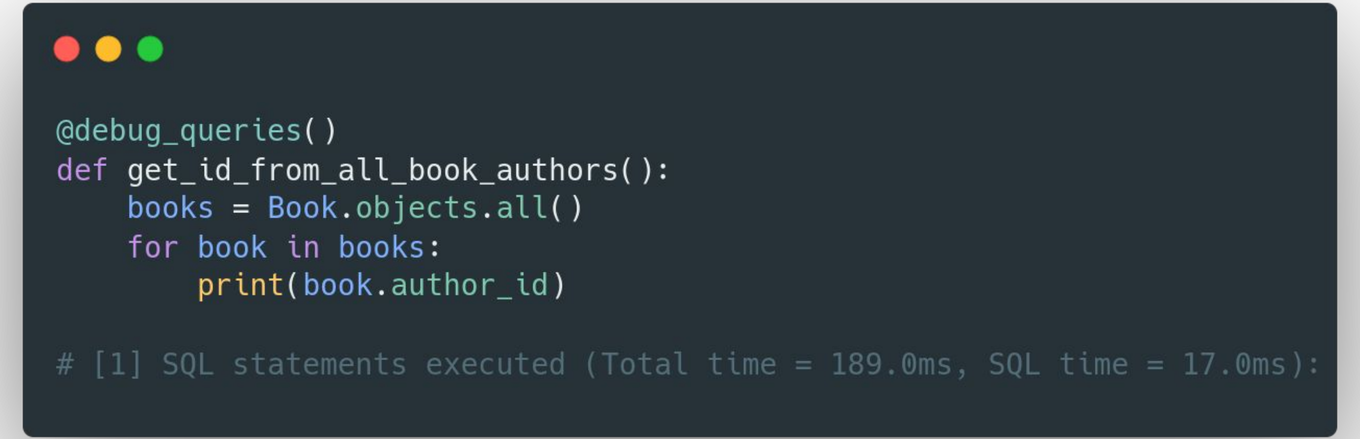
Nossa aplicação



Acessando o id de uma Foreign Key

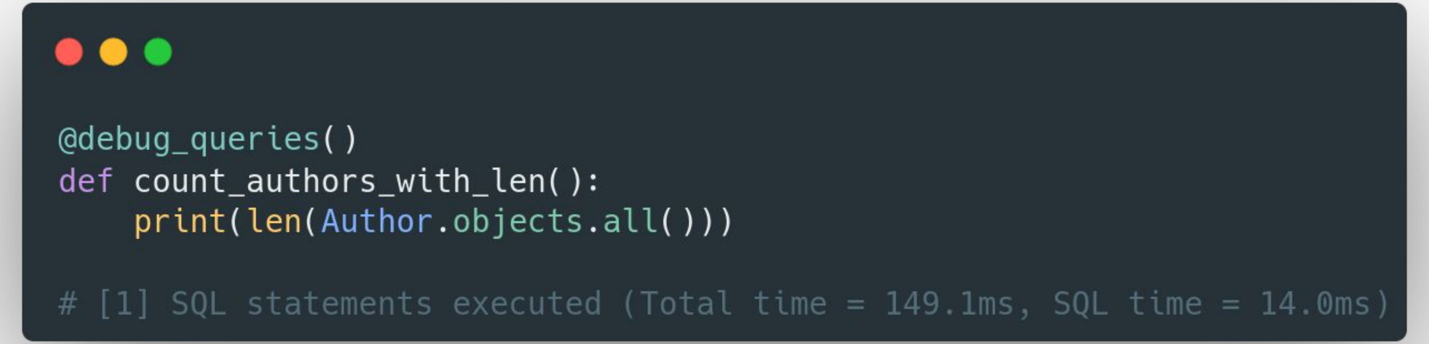


```
@debug_queries()
def get_id_from_all_book_authors():
    books = Book.objects.all()
    for book in books:
        print(book.author.id)

# [9000] SQL statements executed (Total time = 6252.0ms, SQL time = 4.0ms)
```

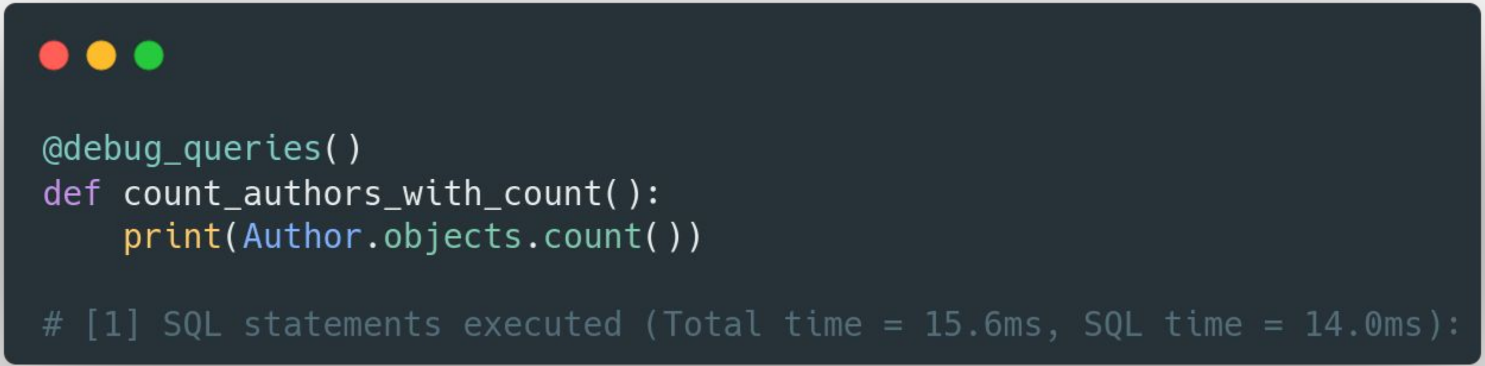


```python
@debug_queries()
def get_id_from_all_book_authors():
    books = Book.objects.all()
    for book in books:
        print(book.author_id)

# [1] SQL statements executed (Total time = 189.0ms, SQL time = 17.0ms):
```

len() vs count()

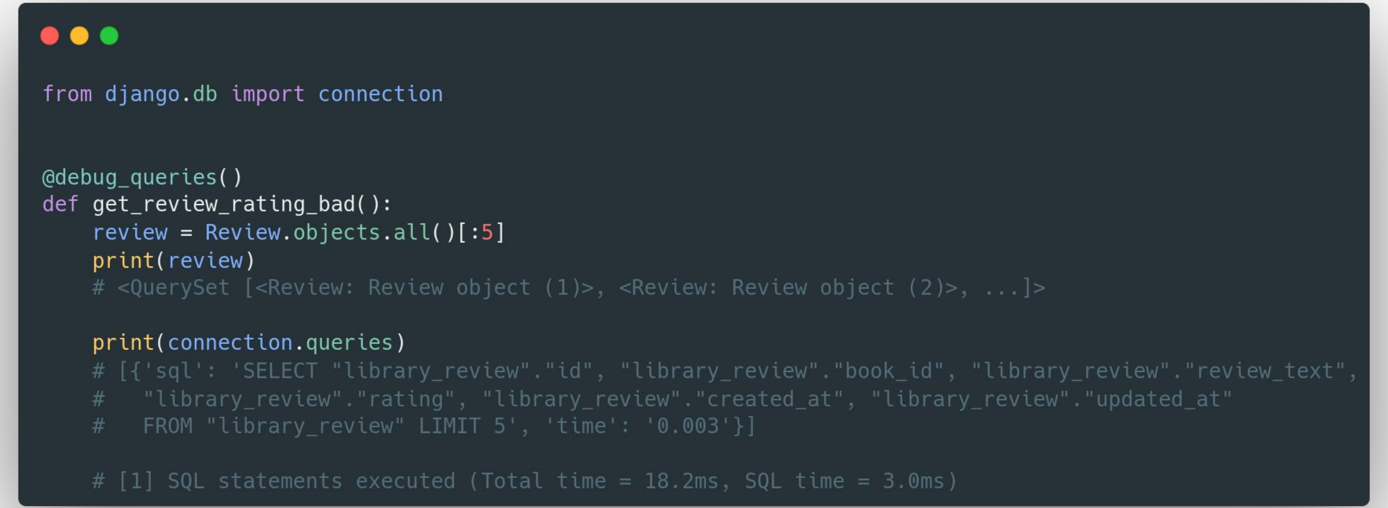


```
@debug_queries()
def count_authors_with_len():
    print(len(Author.objects.all()))

# [1] SQL statements executed (Total time = 149.1ms, SQL time = 14.0ms)
```

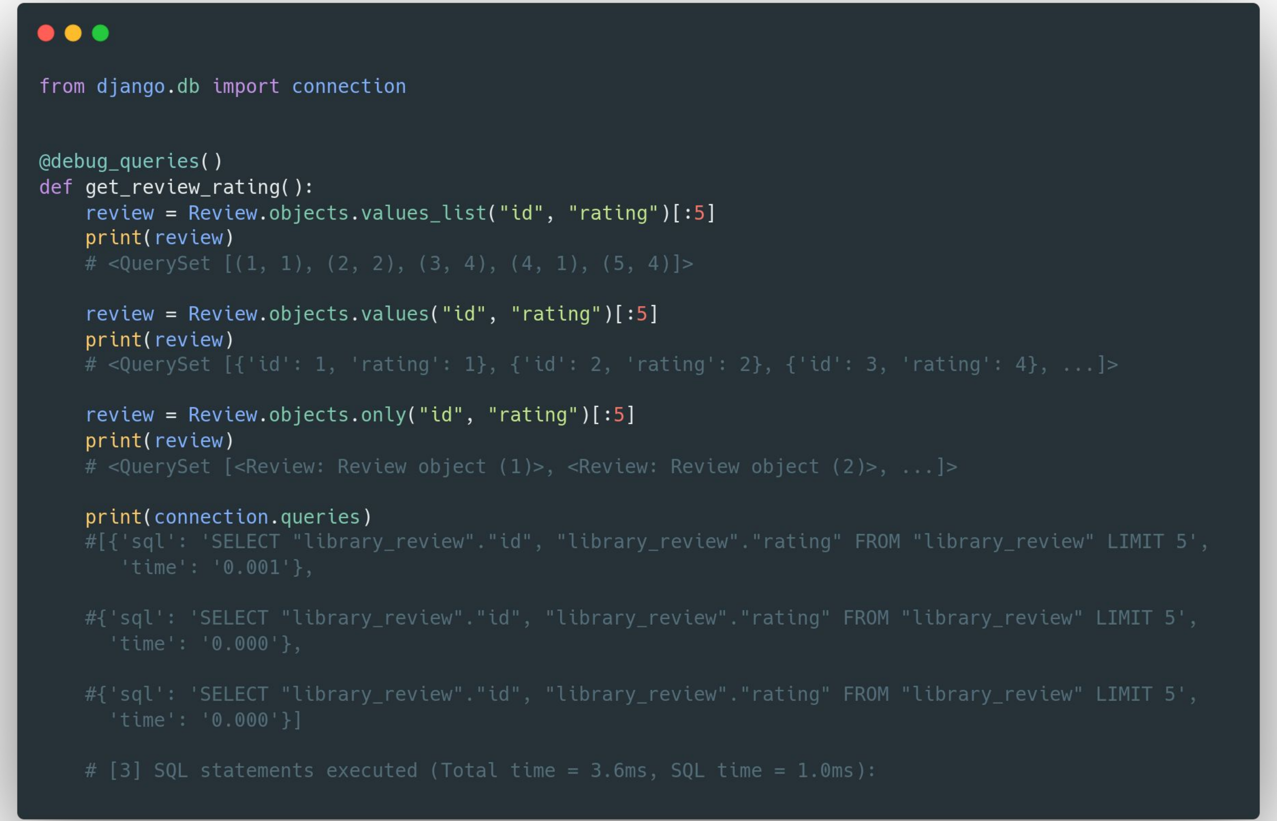


```
@debug_queries()
def count_authors_with_count():
    print(Author.objects.count())

# [1] SQL statements executed (Total time = 15.6ms, SQL time = 14.0ms):
```

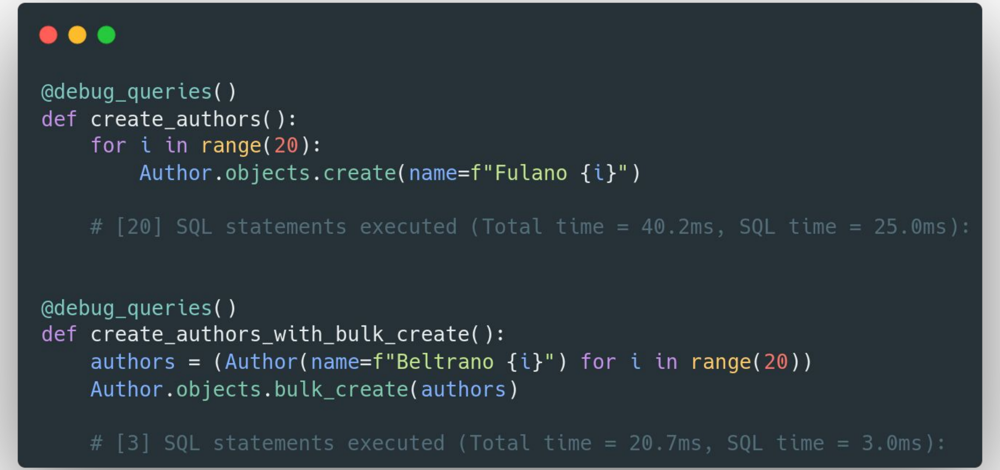
Retornar somente os campos necessários



```
from django.db import connection


@debug_queries()
def get_review_rating_bad():
    review = Review.objects.all()[:5]
    print(review)
    # <QuerySet [<Review: Review object (1)>, <Review: Review object (2)>, ...]>

    print(connection.queries)
    # [{'sql': 'SELECT "library_review"."id", "library_review"."book_id", "library_review"."review_text",
    #   "library_review"."rating", "library_review"."created_at", "library_review"."updated_at"
    #   FROM "library_review" LIMIT 5', 'time': '0.003'}]

    # [1] SQL statements executed (Total time = 18.2ms, SQL time = 3.0ms)
```


```python
from django.db import connection


@debug_queries()
def get_review_rating():
    review = Review.objects.values_list("id", "rating")[:5]
    print(review)
    # <QuerySet [(1, 1), (2, 2), (3, 4), (4, 1), (5, 4)]>

    review = Review.objects.values("id", "rating")[:5]
    print(review)
    # <QuerySet [{'id': 1, 'rating': 1}, {'id': 2, 'rating': 2}, {'id': 3, 'rating': 4}, ...]>

    review = Review.objects.only("id", "rating")[:5]
    print(review)
    # <QuerySet [<Review: Review object (1)>, <Review: Review object (2)>, ...]>

    print(connection.queries)
    #[{'sql': 'SELECT "library_review"."id", "library_review"."rating" FROM "library_review" LIMIT 5',
       'time': '0.001'},

    #{'sql': 'SELECT "library_review"."id", "library_review"."rating" FROM "library_review" LIMIT 5',
      'time': '0.000'},

    #{'sql': 'SELECT "library_review"."id", "library_review"."rating" FROM "library_review" LIMIT 5',
      'time': '0.000'}]

    # [3] SQL statements executed (Total time = 3.6ms, SQL time = 1.0ms):
```

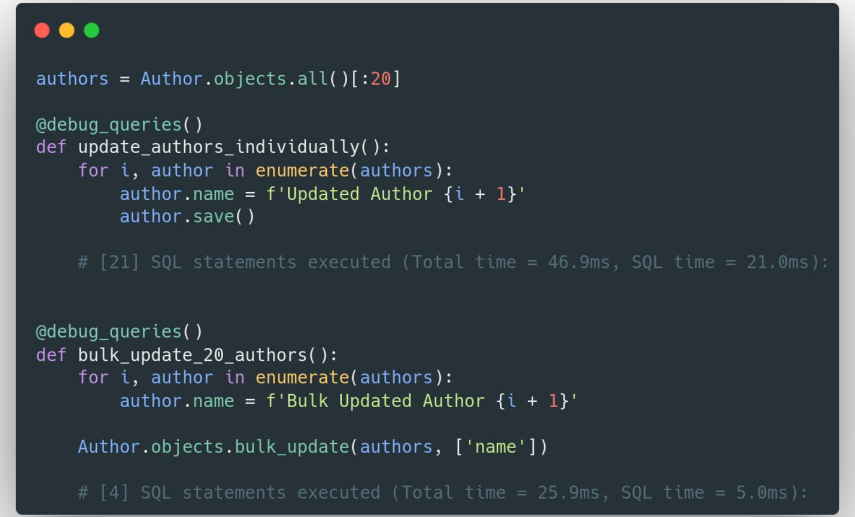
Bulk operations (create)



```
@debug_queries()
def create_authors():
    for i in range(20):
        Author.objects.create(name=f"Fulano {i}")

    # [20] SQL statements executed (Total time = 40.2ms, SQL time = 25.0ms):


@debug_queries()
def create_authors_with_bulk_create():
    authors = (Author(name=f"Beltrano {i}") for i in range(20))
    Author.objects.bulk_create(authors)

    # [3] SQL statements executed (Total time = 20.7ms, SQL time = 3.0ms):
```

- **Não** chama o método **save()**
- **Não** funciona com **Many-to-Many**
- **Não** suporta **signals**
- **auto_now** e **auto_now_add** não são definidos automaticamente

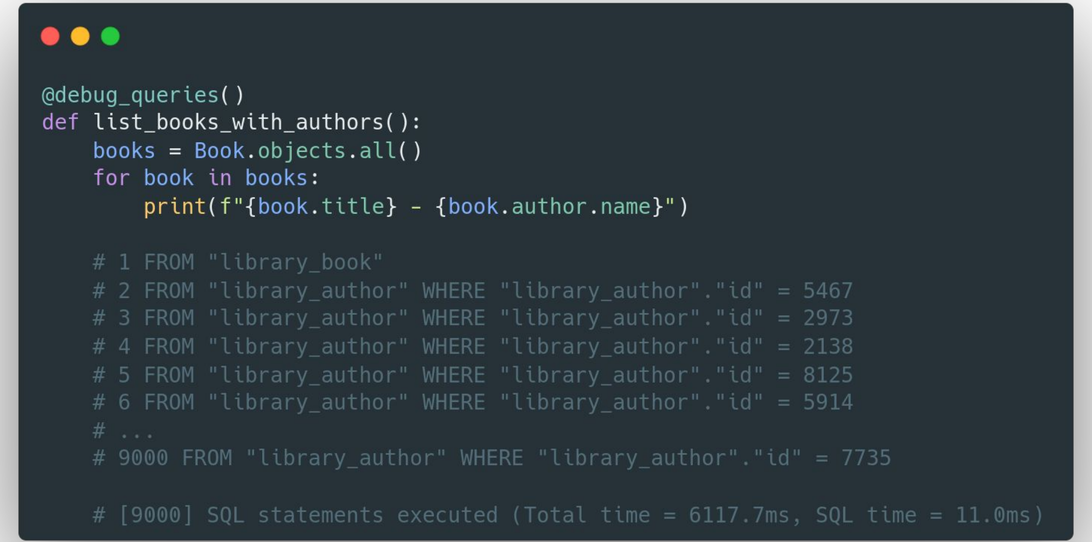
Bulk operations (update)



```
authors = Author.objects.all()[:20]

@debug_queries()
def update_authors_individually():
    for i, author in enumerate(authors):
        author.name = f'Updated Author {i + 1}'
        author.save()

    # [21] SQL statements executed (Total time = 46.9ms, SQL time = 21.0ms):


@debug_queries()
def bulk_update_20_authors():
    for i, author in enumerate(authors):
        author.name = f'Bulk Updated Author {i + 1}'

    Author.objects.bulk_update(authors, ['name'])

    # [4] SQL statements executed (Total time = 25.9ms, SQL time = 5.0ms):
```

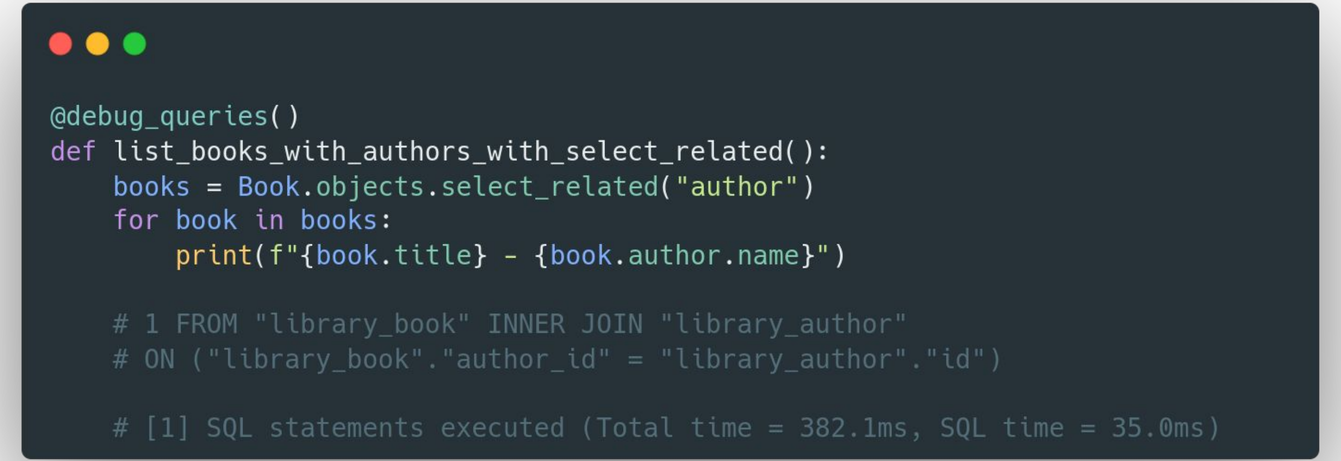
- Não chama o método **save()**
- Não funciona com **Many-to-Many**
- Não suporta **signals**
- Não atualiza **Primary Keys**

Problema do N+1

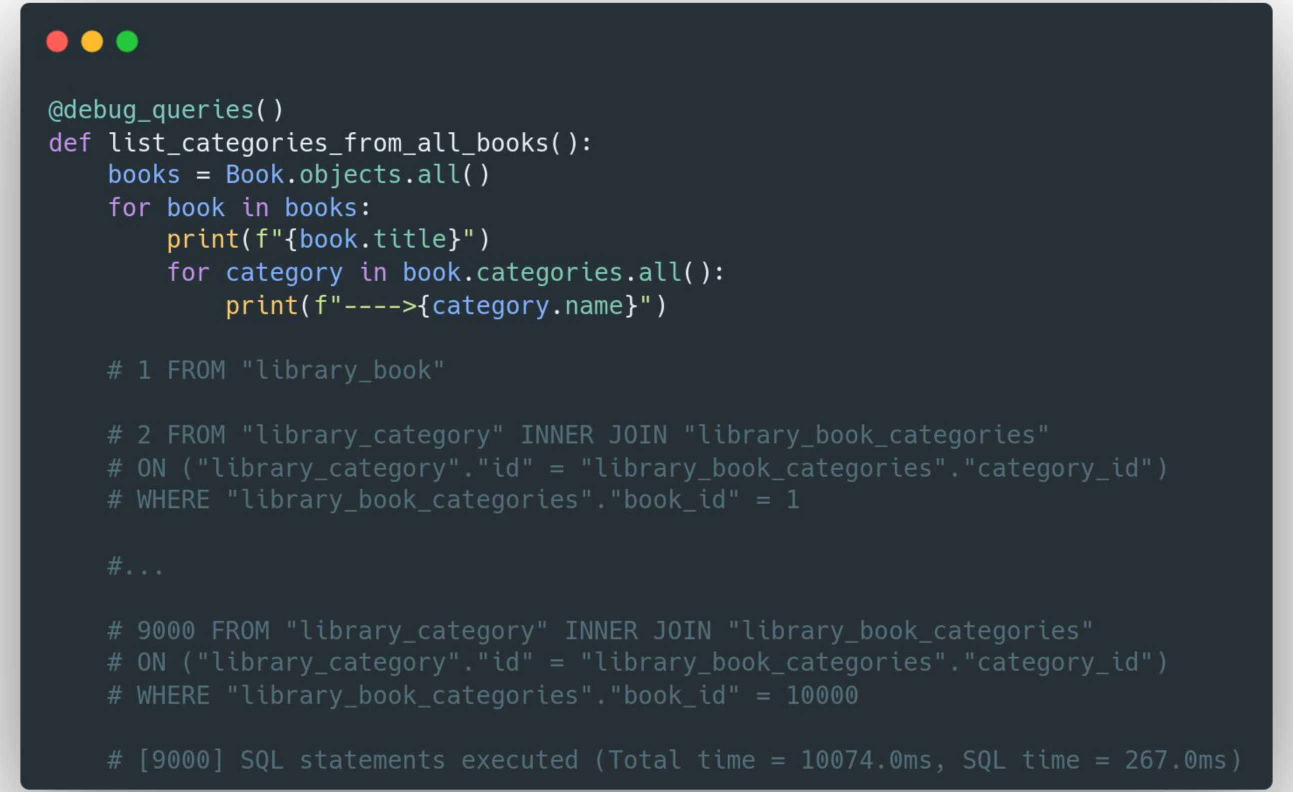


```
@debug_queries()
def list_books_with_authors():
    books = Book.objects.all()
    for book in books:
        print(f"{book.title} - {book.author.name}")

    # 1 FROM "library_book"
    # 2 FROM "library_author" WHERE "library_author"."id" = 5467
    # 3 FROM "library_author" WHERE "library_author"."id" = 2973
    # 4 FROM "library_author" WHERE "library_author"."id" = 2138
    # 5 FROM "library_author" WHERE "library_author"."id" = 8125
    # 6 FROM "library_author" WHERE "library_author"."id" = 5914
    # ...
    # 9000 FROM "library_author" WHERE "library_author"."id" = 7735

    # [9000] SQL statements executed (Total time = 6117.7ms, SQL time = 11.0ms)
```

select_related()

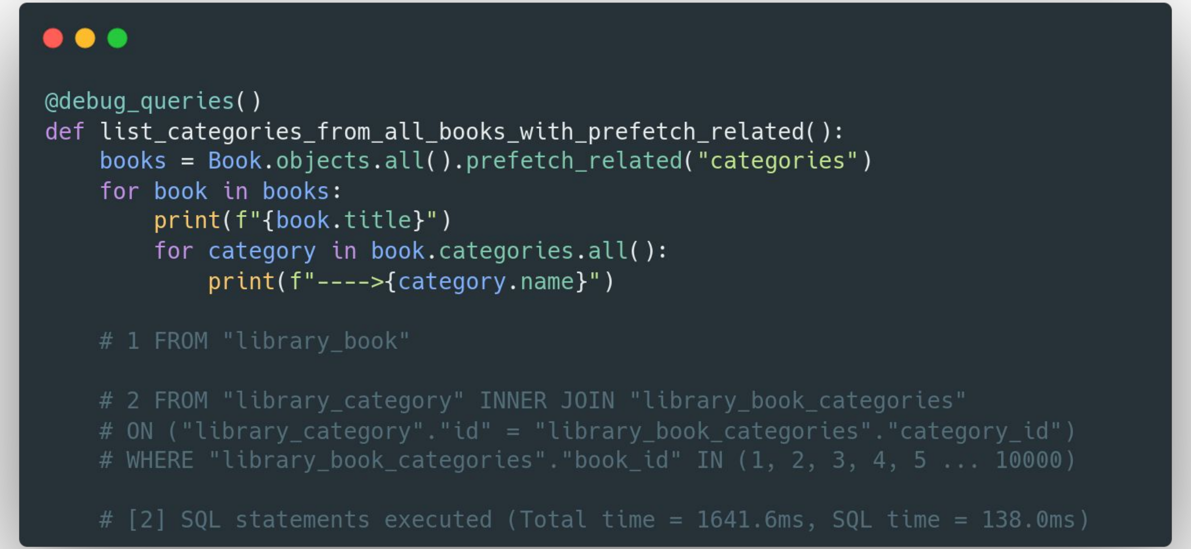


```
@debug_queries()
def list_books_with_authors_with_select_related():
    books = Book.objects.select_related("author")
    for book in books:
        print(f"{book.title} - {book.author.name}")

    # 1 FROM "library_book" INNER JOIN "library_author"
    # ON ("library_book"."author_id" = "library_author"."id")

    # [1] SQL statements executed (Total time = 382.1ms, SQL time = 35.0ms)
```



```
@debug_queries()
def list_categories_from_all_books():
    books = Book.objects.all()
    for book in books:
        print(f"{book.title}")
        for category in book.categories.all():
            print(f"---->{category.name}")

    # 1 FROM "library_book"

    # 2 FROM "library_category" INNER JOIN "library_book_categories"
    # ON ("library_category"."id" = "library_book_categories"."category_id")
    # WHERE "library_book_categories"."book_id" = 1

    #...

    # 9000 FROM "library_category" INNER JOIN "library_book_categories"
    # ON ("library_category"."id" = "library_book_categories"."category_id")
    # WHERE "library_book_categories"."book_id" = 10000

    # [9000] SQL statements executed (Total time = 10074.0ms, SQL time = 267.0ms)
```

prefetch_related()



```
@debug_queries()
def list_categories_from_all_books_with_prefetch_related():
    books = Book.objects.all().prefetch_related("categories")
    for book in books:
        print(f"{book.title}")
        for category in book.categories.all():
            print(f"---->{category.name}")

    # 1 FROM "library_book"

    # 2 FROM "library_category" INNER JOIN "library_book_categories"
    # ON ("library_category"."id" = "library_book_categories"."category_id")
    # WHERE "library_book_categories"."book_id" IN (1, 2, 3, 4, 5 ... 10000)

    # [2] SQL statements executed (Total time = 1641.6ms, SQL time = 138.0ms)
```

Menções honrosas

- Q expressions
- F expressions
- Indexação
- Cache

That's all Folks!

Yuri Sales

Desenvolvedor Back-End, pós-graduado
em Engenharia de Software

linkedin: /yuriscosta
github: @yurisalesc
twitter: @zelantissimus
instagram: @codecrusader.dev