2024 djangocon.us DURHAM

Level Up Your Django Performance:

Identifying and Taming N+1 Queries

Mohammad Ahtasham ul Hassan

(Works at Arbisoft)

mohammad-ahtasham

Shafqat Farhan Ahmed

(Works at Arbisoft)

shafqat-farhan-87677791

2024 djangocon.us DURHAM

Agenda

- Overview of ORMs with respect to relational databases
- Importance of performance in software applications
- What is N+1 problem
- Performance impact of N+1 problem
- Identifying N+1 queries (an example)
- Taming N+1 queries
- Considerations and key takeaways
- Questions?

Overview of ORMs with respect to relational databases

- Automatic query generation
- Django ORM operates on a lazy querying principle
- The database is only accessed when the query is actually evaluated
- N+1 query problem is particularly relevant to ORM and relational database queries
    - It can lead to performance issues

Importance of performance in software applications

- Why Performance matters?
  - User Experience
    - Timeoutes
    - Random crashes/Memory overloads
  - Search Engine Rankings
  - Impact on Customer Retention
- Side effects of ORM on performance
  - Abstraction overhead
  - Increased Memory usage
  - N+1 problems

What is N+1 problem?

- When do N+1 queries occur?
  - Foreign Key Relationships
  - Reverse Foreign Key Lookups
  - Many-to-Many Relationships
  - Templates with ORM queries
  - Serializers with ORM queries
- Why doesn't the ORM handle itself?
  - **Performance**: Lazy loading helps to minimize the initial database query size
    - Complex relationships
  - **Flexibility**: Allows developers to optimize queries based on the specific use case

What is N+1 problem? (contd.)


```python
class Conference(models.Model):
    name = models.CharField(max_length=100)
    date = models.DateField()


class Talk(models.Model):
    title = models.CharField(max_length=200)
    duration = models.DurationField()
    conference = models.ForeignKey(
        Conference, related_name="talks", on_delete=models.CASCADE
    )
```

What is N+1 problem? (contd.)

Code example:

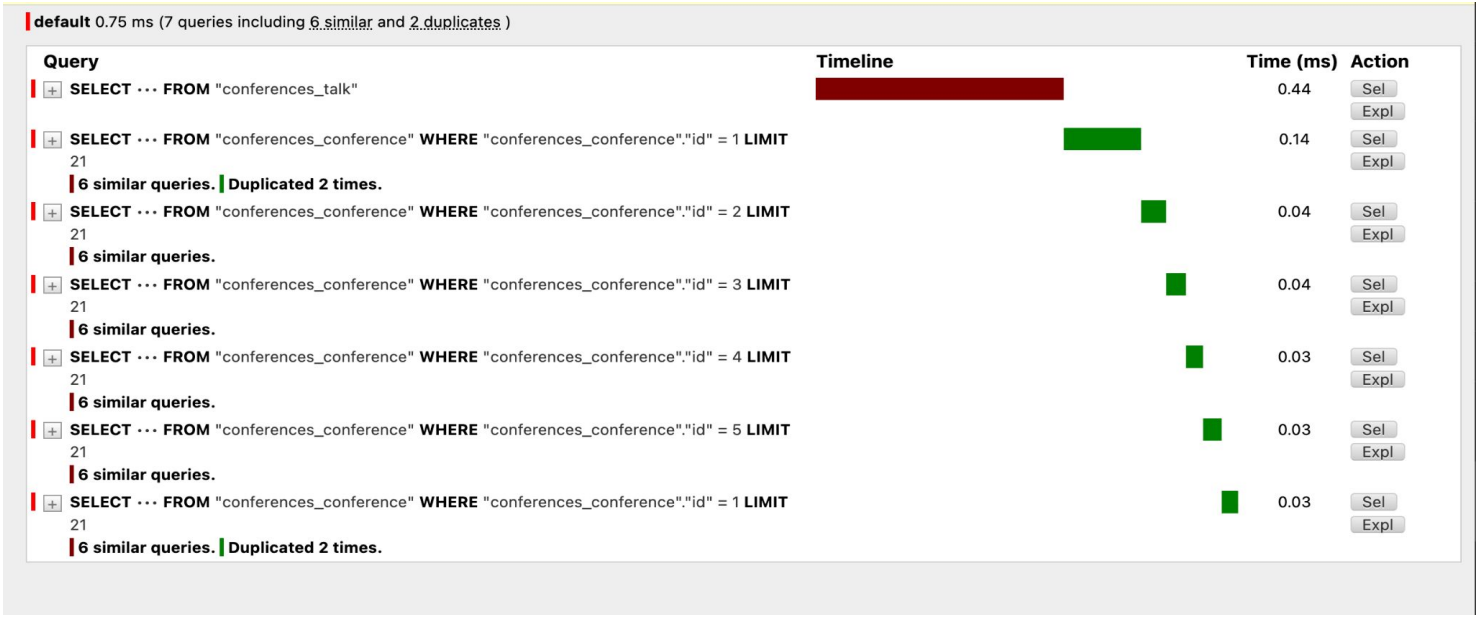
```python
# Fetch all talks and their related conference data

talks = Talk.objects.all()
conference_data = []
for talk in talks:
    conference_name = talk.conference.name
    conference_data.append(conference_name)
```

Performance impact of N+1 problem

- Increased database round trips for each query
- Degraded user experience
- Inefficient query execution
- Frequent API timeouts

Identifying N+1 queries (an example)

- Using Django Debug Toolbar
  - SQL query logging



default 0.75 ms (7 queries including 6 similar and 2 duplicates)

Query	Timeline	Time (ms)	Action
SELECT ··· FROM "conferences_talk"		0.44	Sel Expl
SELECT ··· FROM "conferences_conference" **WHERE** "conferences_conference"."id" = 1 **LIMIT** 21 **6 similar queries. Duplicated 2 times.**		0.14	Sel Expl
SELECT ··· FROM "conferences_conference" **WHERE** "conferences_conference"."id" = 2 **LIMIT** 21 **6 similar queries.**		0.04	Sel Expl
SELECT ··· FROM "conferences_conference" **WHERE** "conferences_conference"."id" = 3 **LIMIT** 21 **6 similar queries.**		0.04	Sel Expl
SELECT ··· FROM "conferences_conference" **WHERE** "conferences_conference"."id" = 4 **LIMIT** 21 **6 similar queries.**		0.03	Sel Expl
SELECT ··· FROM "conferences_conference" **WHERE** "conferences_conference"."id" = 5 **LIMIT** 21 **6 similar queries.**		0.03	Sel Expl
SELECT ··· FROM "conferences_conference" **WHERE** "conferences_conference"."id" = 1 **LIMIT** 21 **6 similar queries. Duplicated 2 times.**		0.03	Sel Expl

Identifying N+1 queries (contd.)

- Code Reviews
- Testing
  - Use unit tests to test for executed queries

```python
class MyTest(TestCase):
    def test_query_count(self):
        with self.assertNumQueries(1):
            response = self.client.get('/books/')
```

- Other options?

Taming N+1 queries

- select_related()
  - For foreign key relations
  - Loads related objects in one query

```
talks = Talk.objects.select_related("conference").all()
```

- prefetch_related()
  - For many-to-many relations or reverse lookups
  - Performs an additional query and then join in Python

```
conferences = Conference.objects.prefetch_related('talks').all()
```

Taming N+1 queries (contd.)

default 0.61 ms (1 query)

Query	Timeline	Time (ms)	Action
- **SELECT** "conferences_talk"."id", "conferences_talk"."title", "conferences_talk"."duration", "conferences_talk"."conference_id", "conferences_conference"."id", "conferences_conference"."name", "conferences_conference"."date" **FROM** "conferences_talk" **INNER JOIN** "conferences_conference" **ON** ("conferences_talk"."conference_id" = "conferences_conference"."id")		0.61	Sel Expl

Connection: default

/Users/mohammad.ahtasham/Desktop/djangocon/.venv/lib/python3.11/site-packages/django/contrib/staticfiles/**handlers.py** in **__call__**(**80**)

```
return self.application(environ, start_response)
```

/Users/mohammad.ahtasham/Desktop/djangocon/conferences/**views.py** in **conference_list_view**(**14**)

```
for talk in talks:
```

Considerations and key takeaways

- The performance impact is minimal with small datasets
- Solving the N+1 problem may cause ripple effects in large and complex datasets
    - Prefetching large datasets can lead to increased memory usage
- Avoid prefetching related data if there's no specific use case (mindful data fetching)
- Django REST Framework (DRF) serializers fetching relational data can lead to N+1 query issues

Resources:

https://docs.djangoproject.com/en/4.2/ref/models/querysets/#select-related

https://docs.djangoproject.com/en/4.2/ref/models/querysets/#prefetch-related

Questions?