EMOCON
2017 S/S

n+1 problem

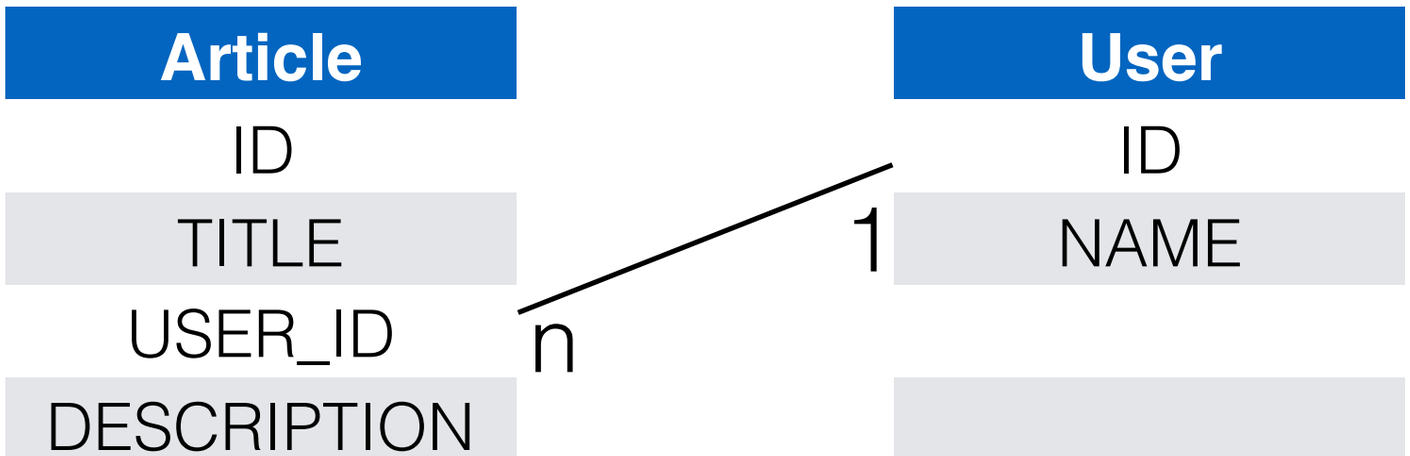
in GraphQL

Index

- What is this?
- How can we solve this?
  - Cache
  - ActiveRecord::QueryMethods#includes
  - Batch

n + 1 problem?

```
query {
  articles {
    id
    title
    description
    user {
      id
      name
    }
  }
}
```



- get all articles first
  select * from articles

- get user of each article
  select * from users where id = 1
  select * from users where id = 2
  select * from users where id = …
  select * from users where id = n

GraphQL?

- Resolver Only receives:
  - Parent object
  - Arguments
  - Context

```
Query: {
  articles(_obj, _args, context) {
    return context.db.loadArticles().then(
      articlesData => articlesData.map(articleData => new Article(articleData))
    )
  }
}

Article: {
  user(obj, args, context) {
    return context.db.loadUserByID(obj.userId).then(
      userData => new User(userData)
    )
  }
}
```

- Resolver could be async

n+1 problem in GraphQL

n+1 problem

in limited interface

1. Cache (cont.)

```
Article: {
  user(obj, args, context) {
    return context.db.loadCachedUserByID(obj.userId).then(
      userData ⇒ new User(userData)
    )
  }
}

loadCachedUserByID(id) {
  let user = cache.read(['user', id])
  if (!user) {
    user = loadUserByID(id)
    cache.write(['user', id], user)
  }
  return user
}
```

1. Cache (cont.)

- Pros
    - Very Easy at first time

1. Cache (cont.)

TwoHardThings

Martin Fowler

14 July 2009

> *There are only two hard things in Computer Science: cache invalidation and naming things.*
>
> *-- Phil Karlton*

1. Cache (end)

- Cons
  - Hard to maintain it
  - Still heavy at first time

2. ActiveRecord::QueryMethod#includes (cont.)

```
Query: {
  articles(_obj, _args, context) {
    return context.db.loadArticlesWithUser().then(
      articlesData => articlesData.map(articleData => new Article(articleData))
    )
  }
}

loadArticlesWithUser() {
  return loadArticles().includes(['user'])
}
```

2. ActiveRecord::QueryMethod#includes (cont.)

- Pros
  - Easy to understand

2. ActiveRecord::QueryMethod#includes (end)

- Cons
  - More concerns
  - Easy to make mistake

3. Batch (cont.)

```
Article: {
  user(obj, args, context) {
    return batch(loadUsersByIDs, obj.userId).then(
      usersObject ⇒ new User(usersObject[obj.userIds])
    )
  }
}

loadUsersByIDs(ids) {
  return db.loadUsersByIDs(ids).then(
    usersData ⇒ usersData.reduce((acc, userData) ⇒ {
      acc[userData.id] = userData
      return acc
    }, {})
  )
}
```

3. Batch (end)

- Cons
    - Hard to understand

3. Batch (cont.)

- Pros
  - Elegant solution
  - No cache
  - No complex concern

Refs

- http://absinthe-graphql.org/guides/ecto-best-practices/

- https://github.com/Shopify/graphql-batch

- http://edgeguides.rubyonrails.org/caching_with_rails.html#sql-caching

- http://api.rubyonrails.org/classes/ActiveRecord/QueryMethods.html#method-i-includes