Hibernate - проблема декартова произведения при запросах с пагинацией

Артём
Гордиенко

Росбанк

JPoint

росбанк

О СЕБЕ

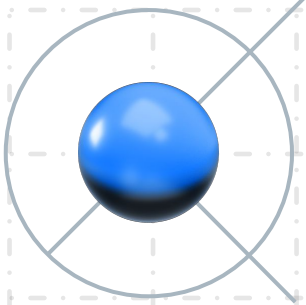
Java/Kotlin разработчик в Росбанк

Разрабатываю микросервисы
для осуществления внешнеэкономической
деятельности

@arvgord
arvgord@gmail.com
github.com/arvgord

О ЧЕМ

я хочу рассказать



01 Проблема реализации запросов с пагинацией в современных версиях Spring Data JPA и Hibernate

02 Подводные камни использования стандартных решений

03 Проблема декартова произведения в запросах с пагинацией

04 Баги в Hibernate 6

05 Какие есть эффективные реализации запросов с пагинацией, плюсы и минусы

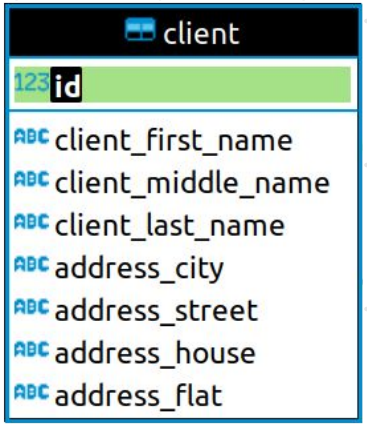
06 Выводы

Приложение, отображающее список с пагинацией

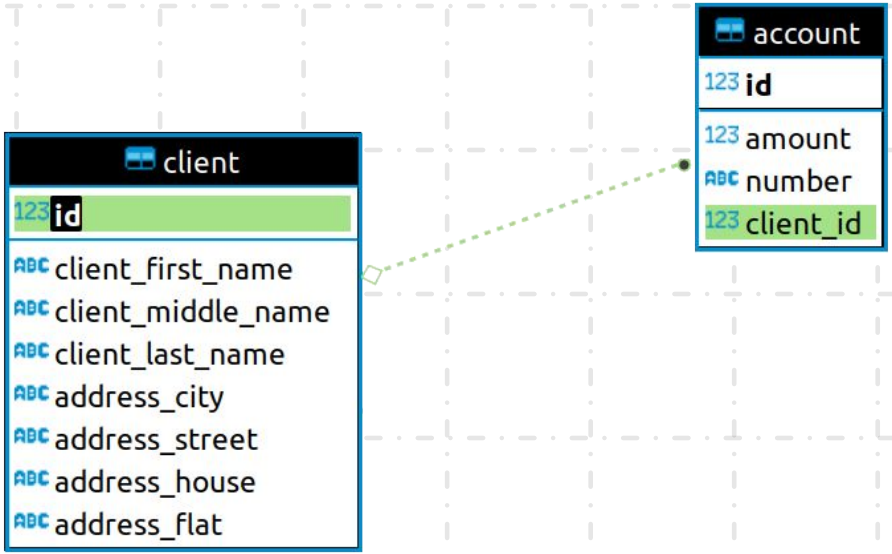
Rows per page: 50 1–50 of 10000

Client name	Total accounts	Total accounts balance	Total deposits	Average deposits rate	Total loans	Average loans rate	Total transfers	Total transfers amount
Филиппова В. А.	1	9573319	1	11.00	0	0	11	41170026
Никитина Е. Э.	1	2523438	1	5.00	1	13.00	9	57062982
Тарасова А. В.	4	12454590	2	10.50	0	0	4	25789204
Лапин Р. Ф.	1	9020201	1	9.00	1	3.00	12	51010125
Гришин А. А.	1	397470	3	7.33	3	10.00	4	28651446

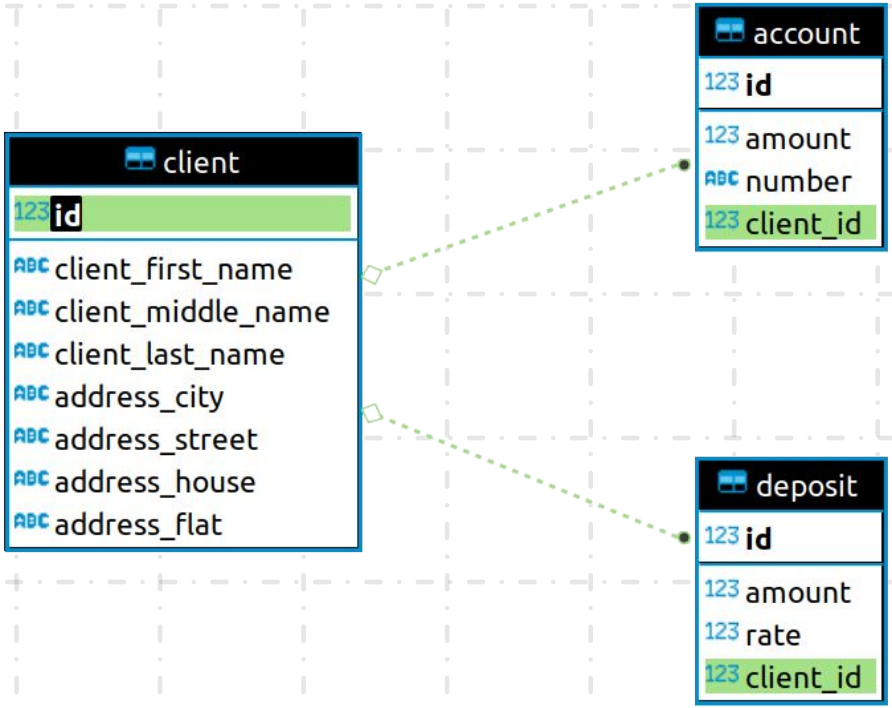
Модель данных



Модель данных

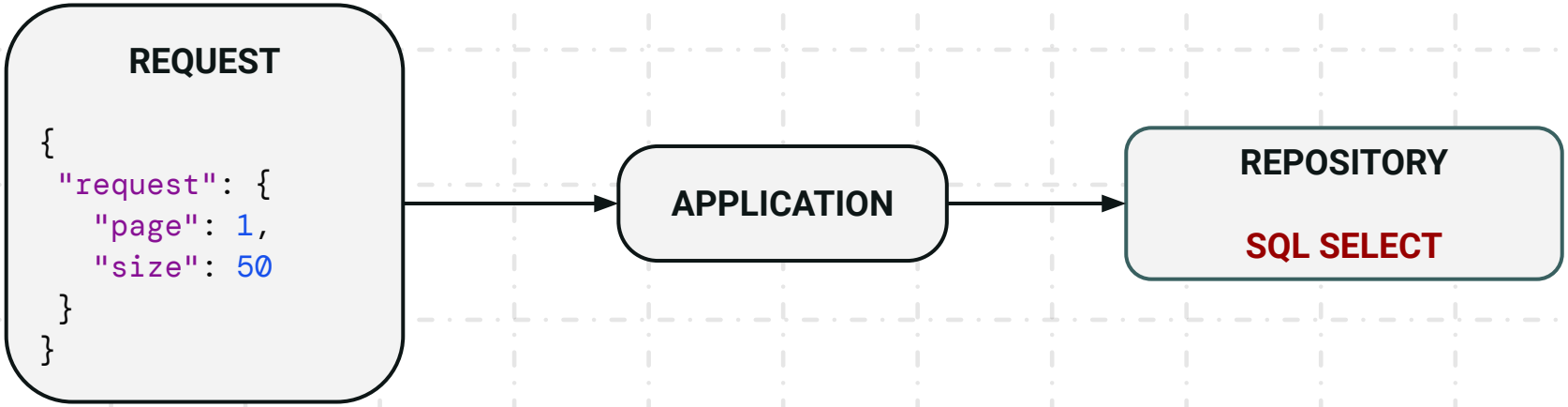


Модель данных



```kotlin
@Entity
@Table(name = "CLIENT")
class ClientEntity(

    @Id
    @Column(name = "ID")
    var id: Long? = null,

    @OneToMany(mappedBy = "client")
    var accounts: Set<AccountEntity> = mutableSetOf(),

    @OneToMany(mappedBy = "client")
    var deposits: Set<DepositEntity> = mutableSetOf()
)
```

Приложение, отображающее список с пагинацией



Реализация запроса с пагинацией с помощью Spring Data

```
@Repository
interface ClientRepository : JpaRepository<ClientEntity, Long>
```

Реализация запроса с пагинацией с помощью Spring Data

```
@Repository
interface ClientRepository : JpaRepository<ClientEntity, Long>

@Autowired
private lateinit var clientRepository: ClientRepository

clientRepository.findAll(PageRequest.of(pageNumber, pageSize))
```

Реализация запроса с пагинацией с помощью Hibernate

```
val count = entityManager.createQuery("select count(c) from ClientEntity c",
   Long::class.java)
  .singleResult

val result = entityManager.createQuery("select c from ClientEntity c",
   ClientEntity::class.java)
  .setFirstResult(firstResult)
  .setMaxResults(maxResult)
  .resultList
```

Выполним запрос с помощью Spring Data

```
clientRepository.findAll(PageRequest.of(pageNumber, pageSize))
```

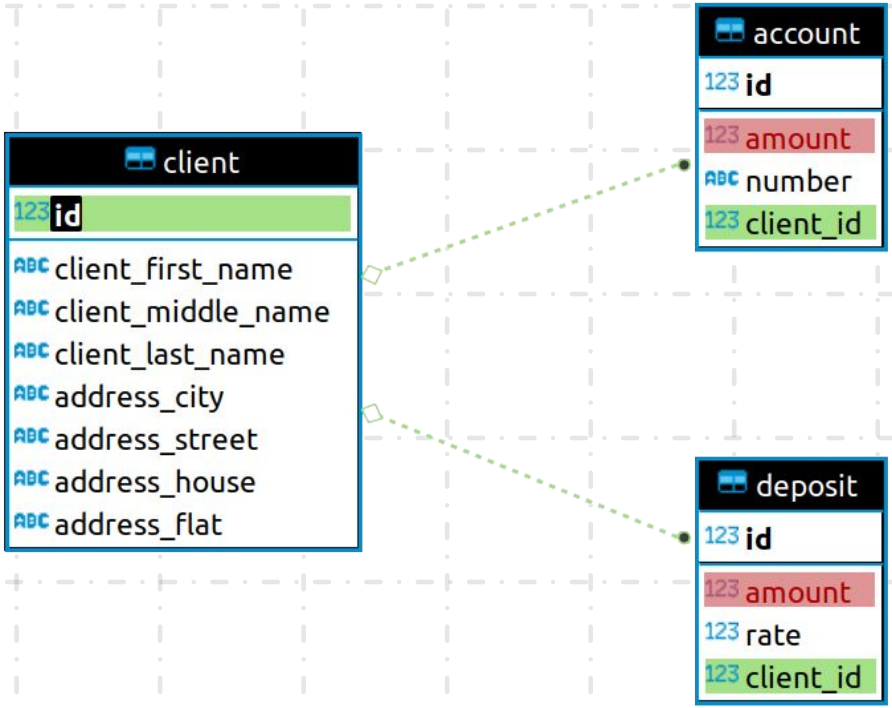
Выполним запрос с помощью Spring Data

```
clientRepository.findAll(PageRequest.of(pageNumber, pageSize))
```



```
1. Hibernate: select * from client c limit 50 offset 50
2. Hibernate: select count(c.id) from client c
```

Посчитаем суммы



Маппим Entity в Domain модель

```
clientRepository.findAll(PageRequest.of(pageNumber, pageSize))
```



```
fun map(entity: ClientEntity): Client
```

Проблема N+1

```
clientRepository.findAll(PageRequest.of(pageNumber, pageSize))
```

↓

```
fun map(entity: ClientEntity): Client
```

↓

```
1.  Hibernate: select * from client c limit 50 offset 50
2.  Hibernate: select count(c.id) from client c
3.  Hibernate: select * from account a where a.client_id=51
4.  Hibernate: select * deposit d from deposit where d.client_id=51
5.  Hibernate: select * from account a where a.client_id=52
6.  Hibernate: select * deposit d from deposit where d.client_id=52
7.  …
```

Причина N+1

```
@OneToMany(mappedBy = "client")
var accounts: Set<AccountEntity> = mutableSetOf()

fun map(entity: ClientEntity): Client

Set<Account> set = accountEntityLazySetToAccountSet(entity.getAccounts());
// При обращении к коллекции происходит запрос к БД
// Hibernate: select * from account a where a.client_id=?
```

Стратегия извлечения @OneToMany по умолчанию FetchType.LAZY

```
@OneToMany(mappedBy = "client")
var accounts: Set<AccountEntity> = mutableSetOf()

/** (Optional) Whether the association should be lazily loaded or
* must be eagerly fetched. The EAGER strategy is a requirement on
* the persistence provider runtime that the associated entities
* must be eagerly fetched.  The LAZY strategy is a hint to the
* persistence provider runtime.
*/
FetchType fetch() default LAZY;
```

А если использовать FetchType.EAGER?

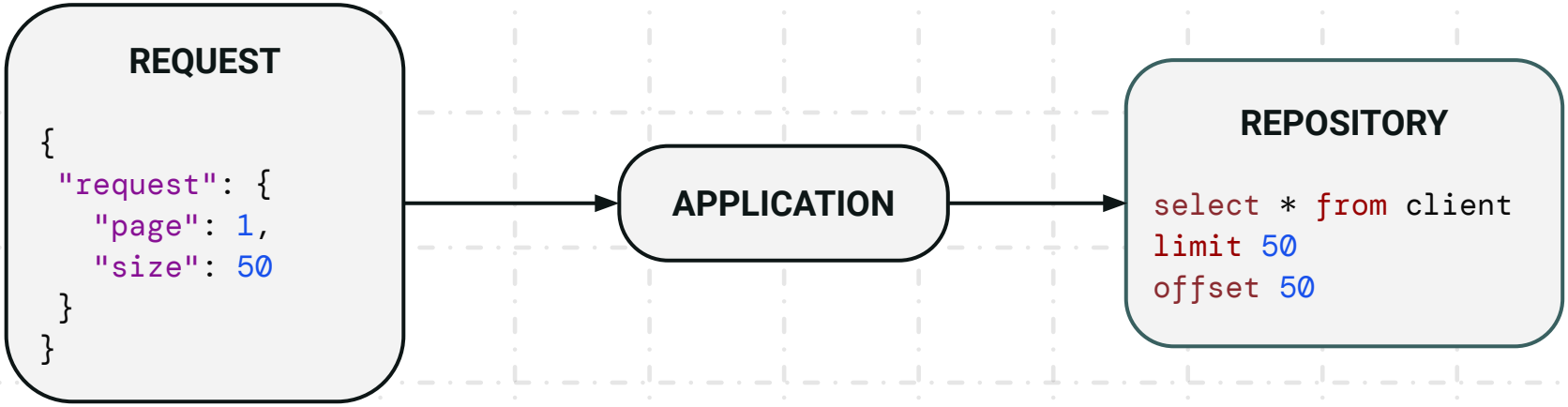
```
@OneToMany(mappedBy = "client", fetch = FetchType.EAGER)
var accounts: Set<AccountEntityEager> = mutableSetOf()
```

А если использовать FetchType.EAGER?

```
@OneToMany(mappedBy = "client", fetch = FetchType.EAGER)
var accounts: Set<AccountEntityEager> = mutableSetOf()
```

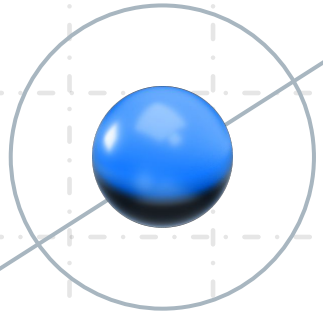
```
1. Hibernate: select * from client c limit 50 offset 50
2. Hibernate: select count(c.id) from client c
3. Hibernate: select * from account a where a.client_id=51
4. Hibernate: select * deposit d from deposit where d.client_id=51
5. Hibernate: select * from account a where a.client_id=52
6. Hibernate: select * deposit d from deposit where d.client_id=52
7. …
```

Приложение, отображающее список с пагинацией



01 Проблема N+1

FetchType.EAGER не работает, за решением N+1 идем на stackoverflow



1444 votes
✓ Accepted

What is the "N+1 selects problem" in ORM (Object-Relational Mapping)?

Alternatively, one could get all wheels and perform the lookups in memory: SELECT * FROM Wheel; This reduces the number of round-trips to the database from **N+1** to 2. … Most ORM to…

database orm

Matt Solnit **31.4k** answered Sep 18, 2008 at 21:36

---

75 votes
✓ 7 answers
86k views

What is the solution for the N+1 issue in JPA and Hibernate?

I understand that the **N+1** problem is where one query is executed to fetch **N** records and **N** queries to fetch some relational records. But how can it be avoided in **Hibernate**? …

java hibernate jpa design-patterns orm

Vipul Agarwal **1,463** asked Sep 8, 2015 at 9:24

---

1 vote
✓ 1 answer
72 views

How to resolve hibernate n+1 problem OneToMany and ManyToOne

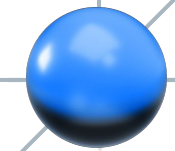
I have problem with **N+1 Hibernate**. … **1** Problem in **hibernate**. …

java hibernate jpql

Kenez92 **116** asked Jan 21, 2021 at 16:01

Типичные решения, которые предлагаются

- Использовать **@EntityGraph**
- Использовать **fetch**
- Нативный запрос с несколькими **JOIN**



Решение N+1 с помощью @EntityGraph в Spring

```
@Repository
interface ClientRepository : JpaRepository<ClientEntity, Long> {


    fun getAllBy(pageable: Pageable): Page<ClientEntity>
}
```

Решение N+1 с помощью @EntityGraph в Spring

```
@Repository
interface ClientRepository : JpaRepository<ClientEntity, Long> {

    @EntityGraph(attributePaths = ["accounts", "deposits"])
    fun getAllBy(pageable: Pageable): Page<ClientEntity>
}
```

Решение N+1 с помощью @EntityGraph в Hibernate

```
@NamedEntityGraph(
    name = "ClientEntityGraph",
    attributeNodes = [NamedAttributeNode("accounts"), NamedAttributeNode("deposits")]
)
@Entity
@Table(name = "CLIENT")
class ClientEntity(

    @Id
    @Column(name = "ID")
    var id: Long? = null,

    @OneToMany(mappedBy = "client")
    var accounts: Set<AccountEntity> = mutableSetOf(),

    @OneToMany(mappedBy = "client")
    var deposits: Set<DepositEntity> = mutableSetOf()
)
```

Решение N+1 с помощью @EntityGraph в Hibernate

```
val count = entityManager.createQuery("select count(c) from ClientEntity c",
   Long::class.java)
  .singleResult

val result = entityManager.createQuery("select c from ClientEntity c",
   ClientEntity::class.java)
  .setHint("jakarta.persistence.fetchgraph",
      entityManager.createEntityGraph("ClientEntityGraph"))
  .setFirstResult(firstResult)
  .setMaxResults(maxResult)
  .resultList
```

Используем fetch, если запросы построены с помощью Criteria API в Spring Data

```
clientRepository.findAll(getSpecification(), PageRequest.of(pageNumber, pageSize))

fun getSpecification(): Specification<ClientEntity> {
   return Specification<ClientEntity> { root, cq, cb ->
     if (cq.resultType == ClientEntity::class.java) {
          root.fetch("accounts", JoinType.LEFT)
          root.fetch("deposits", JoinType.LEFT)
       }
       cb.and()
   }
}
```

Используем fetch, если запросы построены с помощью Criteria API в Hibernate

```
val clientRoot = criteriaQueryEntity.from(ClientEntity::class.java)

clientRoot.fetch<AccountEntity, ClientEntity>("accounts", JoinType.LEFT)
clientRoot.fetch<DepositEntity, ClientEntity>("deposits", JoinType.LEFT)

val result = entityManager.createQuery("select c from ClientEntity c",
    ClientEntity::class.java)
   .setFirstResult(firstResult)
   .setMaxResults(maxResult)
   .resultList
```

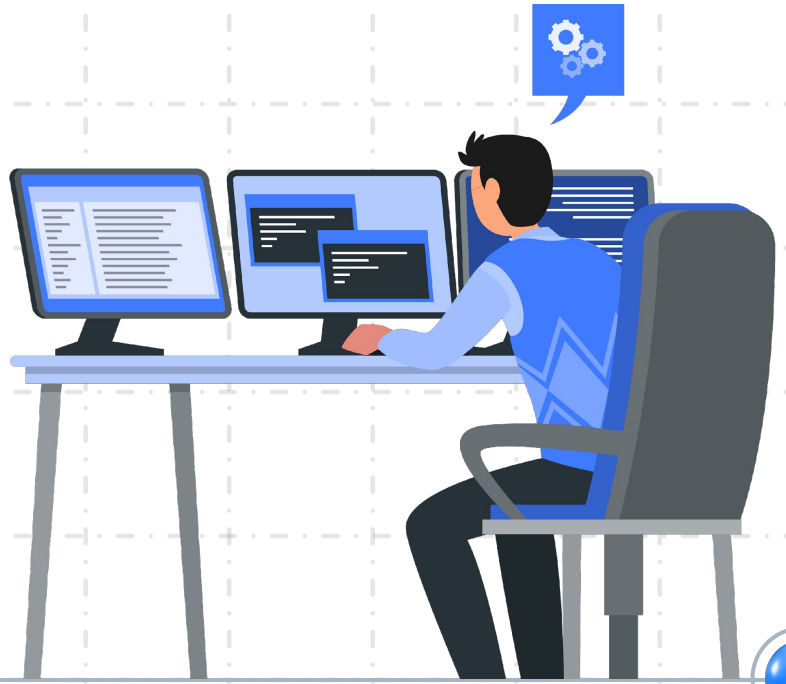
Проблема N+1 решена!
Можно идти пить кофе...

Проблема N+1 решена!
Можно идти пить кофе...

Через некоторое время
к вам приходит **product owner...**

Что же произошло?

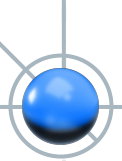
Разве мы должны были ускорить запросы?
Давайте разбираться…



Заглянем в логи...

Hibernate 5:
WARN: HHH000104: firstResult/maxResults specified with collection fetch; applying in memory!

Hibernate 6:
WARN: HHH90003004: firstResult/maxResults specified with collection fetch; applying in memory

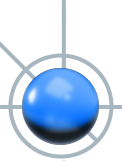


Заглянем в логи...

Hibernate 5:
WARN: HHH000104: firstResult/maxResults specified with collection fetch; applying in memory!

Hibernate 6:
WARN: HHH90003004: firstResult/maxResults specified with collection fetch; applying in memory

```
select * from client c
left outer join account a on a.client_id = c.id
left outer join deposit d on d.client_id = c.id
```

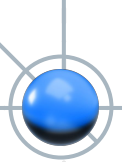


Заглянем в логи...

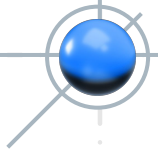
Hibernate 5:
WARN: HHH000104: firstResult/maxResults specified with collection fetch; applying in memory!

Hibernate 6:
WARN: HHH90003004: firstResult/maxResults specified with collection fetch; applying in memory

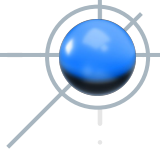
```
select * from client c
left outer join account a on a.client_id = c.id
left outer join deposit d on d.client_id = c.id
limit 50 offset 50
```



Почему исчезли limit и offset?



А вы уже перешли
на Spring-Boot 3.0 / Hibernate 6?



Если в SQL запросе оставить limit и offset, то получаем баг в Hibernate 6

Проекты / Hibernate ORM / HHH-15964

Incorrect results for pageable EntityGraph with Hibernate 6.1.6

Прикрепить | Добавить ссылку на задачу

Описание

Bug description

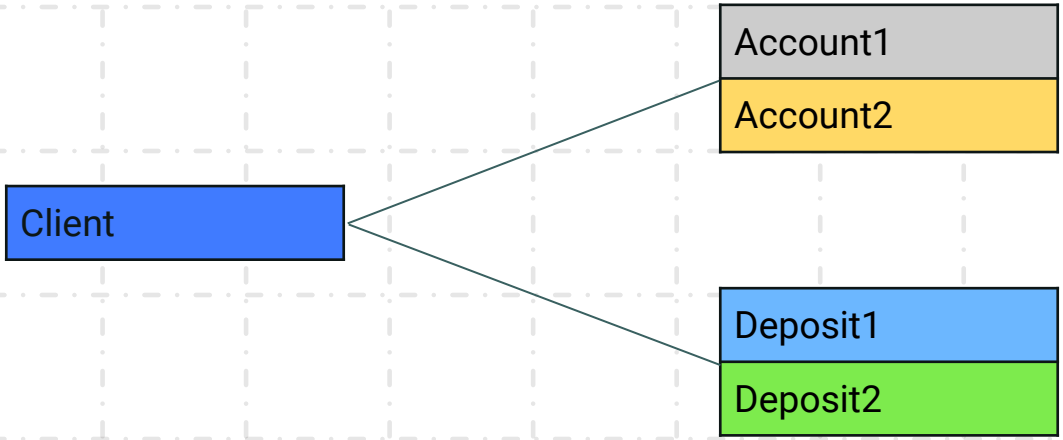
I created test application with custom repository with the help of **spring boot v3.0.1** and **org.springframework.boot:spring-boot-starter-data-jpa**. With findAll function in repository I want to load related entities with entity graph. When I call this method I get single SQL request to DB with offset and fetch at the end. It lead to getting wrong result from db because with entity graph I get a cartesian product fom DB as a result. For example I can get different totalElements result for same data in DB.

```
1 @Repository
2 interface ClientRepository : JpaRepository<ClientEntity, Long> {
3
4     @EntityGraph(attributePaths = ["accounts", "deposits"])
5     override fun findAll(pageable: Pageable): Page<ClientEntity>
6 }
```

Если в SQL запросе оставить limit и offset, то получаем баг в Hibernate 6

При пагинации происходит **SQL запрос с limit и offset**, что приводит к ошибочным результатам. Тестовое приложение: hibernate-cartesian-issue.

Воспроизведем баг

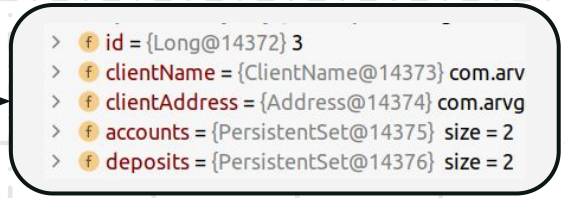



```
select * from client c
left outer join account a on a.client_id = c.id
left outer join deposit d on d.client_id = c.id
limit 4 offset 0
```

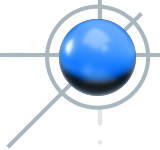
Воспроизведем баг

```
root.fetch<AccountEntity, ClientEntity>("accounts", JoinType.LEFT)
root.fetch<DepositEntity, ClientEntity>("deposits", JoinType.LEFT)
val result = entityManager.createQuery(select)
    .setFirstResult(0)
    .setMaxResults(4)
    .resultList
assertEquals(4, result.size) // Ошибка, ожидалось 4 клиента
```

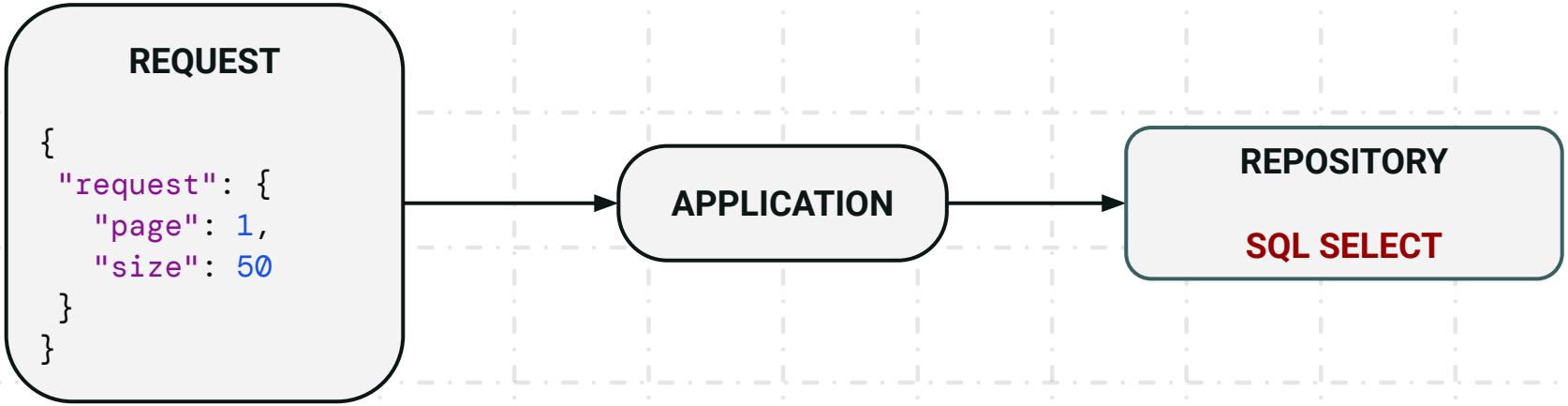
Client	Account1	Deposit1
Client	Account1	Deposit2
Client	Account2	Deposit1
Client	Account2	Deposit2



При использовании JOIN возникает проблема декартова произведения



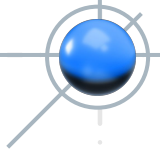
Приложение, отображающее список с пагинацией



01 Проблема N+1

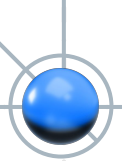
02 Проблема декартова произведения

Какие есть варианты решения проблемы N+1 и декартова произведения?



Старый добрый @BatchSize

```java
@Target({TYPE, METHOD, FIELD})
@Retention(RUNTIME)
public @interface BatchSize {
  /**
   * Strictly positive integer.
   */
  int size();
}
```



Старый добрый @BatchSize

```
@Entity
@Table(name = "CLIENT")
class ClientEntity(

    @Id
    @Column(name = "ID")
    var id: Long? = null,

    @OneToMany(mappedBy = "client")
    @BatchSize(size = 20)
    var accounts: Set<AccountEntity> = mutableSetOf(),

    @OneToMany(mappedBy = "client")
    @BatchSize(size = 50)
    var deposits: Set<DepositEntity> = mutableSetOf()
)
```

Старый добрый @BatchSize

```java
@Target({TYPE, METHOD, FIELD})
@Retention(RUNTIME)
public @interface BatchSize {
  /**
   * Strictly positive integer.
   */
  int size();
}
```

Посмотрим в логи:

```
1. Hibernate: select * from client c limit 50 offset 50
2. Hibernate: select count(c.id) from client c
3. Hibernate: select * from account a where a.client_id in (51, 52, …)
4. Hibernate: select * from deposit d where d.client_id in (51, 52, …)
```

Старый добрый @BatchSize

```java
@Target({TYPE, METHOD, FIELD})
@Retention(RUNTIME)
public @interface BatchSize {
  /**
   * Strictly positive integer.
   */
  int size();
}
```

- Решает проблему N+1
- Решает проблему декартова произведения

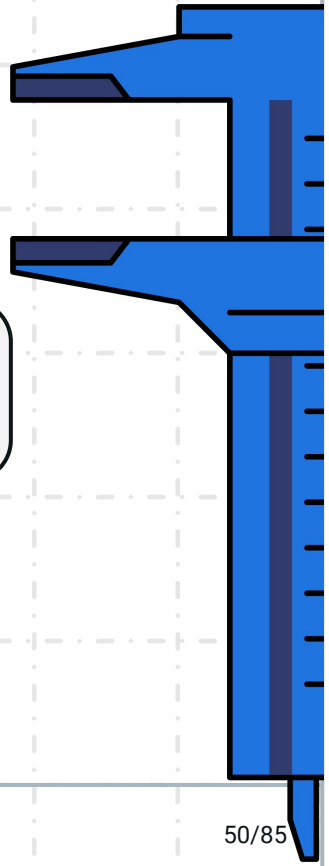
Посмотрим в логи:

```
1. Hibernate: select * from client c limit 50 offset 50
2. Hibernate: select count(c.id) from client c
3. Hibernate: select * from account a where a.client_id in (51, 52, …)
4. Hibernate: select * from deposit d where d.client_id in (51, 52, …)
```

Какой размер @BatchSize выбрать?

```
@OneToMany(mappedBy = "client")
@BatchSize(size = 50)
var accounts: Set<AccountEntity> = mutableSetOf()
```

- При `@BatchSize(size = 1)` снова получаем n+1
- Базы данных могут иметь ограничение в размере in(?, ?, ?, …)
- Возникает проблема производительности при увеличении in(?, ?, ?, …)



Как ещё можно задать @BatchSize?

В application.yaml

```yaml
spring:
 jpa:
   properties:
     hibernate:
       default_batch_fetch_size: 50
```

Будет применяться ко всем коллекциям.

Стратегия выборки @BatchSize в Hibernate 5

В application.yaml настройки **Hibernate 5**

```yaml
spring:
 jpa:
   properties:
     hibernate:
       batch_fetch_style: DYNAMIC
```

По умолчанию стратегия выборки **@BatchSize** **batch_fetch_style: LEGACY** для меньшего потребления памяти лучше его заменить на **PADDED** или **DYNAMIC.**

Стратегия выборки @BatchSize в Hibernate 5

До Hibernate 5.3.0.Final

Style	Batch Size	Heap
Legacy	10	788MB
Legacy	50	1000MB
Dynamic	10	149MB
Dynamic	50	150MB

Hibernate 5.3.0.Final

Style	Batch Size	Heap
Legacy	10	228MB
Legacy	50	269MB
Dynamic	10	119MB
Dynamic	50	120MB

Стратегия выборки @BatchSize в Hibernate 5

```
@OneToMany(mappedBy = "client")
@BatchSize(size = 50)
var accounts: Set<AccountEntity> = mutableSetOf()
```

```
// Что будет если @BatchSize(size = 50) но хотим получить аккаунты у 39 клиентов
// LEGACY
Hibernate: select * from account a where a.client_id in (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,
?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
Hibernate: select * from account a where a.client_id in (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
Hibernate: select * from account a where a.client_id in (?, ?)
// DYNAMIC
Hibernate: select * from account a where a.client_id in
(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)
```

Стратегия выборки @BatchSize в Hibernate 6

В application.yaml настройки **Hibernate 6**

```yaml
spring:
 jpa:
   properties:
     hibernate:
       batch_fetch_style: DYNAMIC
```

```java
/**
* Specifies the {@link org.hibernate.loader.BatchFetchStyle} to use,
* either the name of a {code BatchFetchStyle} instance, or an instance
* of {@code BatchFetchStyle}.
*
* @deprecated An appropriate batch-fetch style is selected automatically
*/
@Deprecated(since = "6.0")
@SuppressWarnings("DeprecatedIsStillUsed")
String BATCH_FETCH_STYLE = "hibernate.batch_fetch_style";
```

Стратегия выборки @BatchSize в Hibernate 6

```
@OneToMany(mappedBy = "client")
@BatchSize(size = 50)
var accounts: Set<AccountEntity> = mutableSetOf()
```

```
// Что будет если @BatchSize(size = 50) но хотим получить аккаунты у 39 клиентов
// По умолчанию будет использована стратегия аналогичная DYNAMIC в Hibernate 5
Hibernate: select * from account a where a.client_id in
(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)
```

Стратегия выборки @BatchSize была сломана в Hibernate 6 и исправлена в Hibernate 6.1.7.Final (Spring-Boot 3.0.5)

Плюсы и минусы **@BatchSize**

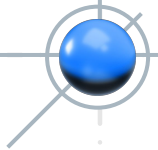
- Стандартная аннотация
- Достаточно добавить несколько аннотаций, чтобы всё заработало
- Решает проблему N+1
- Решает проблему декартова произведения

Плюсы и минусы **@BatchSize**

- Стандартная аннотация
- Достаточно добавить несколько аннотаций, чтобы всё заработало
- Решает проблему N+1
- Решает проблему декартова произведения

- Нужно указывать размер @BatchSize
- Размер @BatchSize задан на этапе компиляции

Какие есть альтернативы @BatchSize?



Разделение запроса на 2 части с использованием @EntityGraph

Запрос ids

```
@Repository
interface ClientRepository : JpaRepository<ClientEntity, Long> {

   @Query("select c.id from ClientEntity c")
   fun getAllIds(pageable: Pageable): Page<Long>



}

val clientIds = clientRepository.getAllIds(PageRequest.of(pageNumber, pageSize))
```

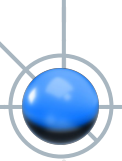
Разделение запроса на 2 части с использованием @EntityGraph


```
@Repository
interface ClientRepository : JpaRepository<ClientEntity, Long> {

   @Query("select c.id from ClientEntity c")
   fun getAllIds(pageable: Pageable): Page<Long>

   @EntityGraph(attributePaths = ["accounts", "deposits"])
   fun getAllByIdIn(clientIds: List<Long>): List<ClientEntity>
}

val clientIds = clientRepository.getAllIds(PageRequest.of(pageNumber, pageSize))
val result = clientRepository.getAllByIdIn(clientIds.content)
```

Разделение запроса на 2 части с использованием @EntityGraph

Запрос ids

```
1. Hibernate: select c.id from client c limit 50 offset 50
2. Hibernate: select count(c.id) from client c
```



Разделение запроса на 2 части с использованием @EntityGraph


```
1. Hibernate: select c.id from client c limit 50 offset 50
2. Hibernate: select count(c.id) from client c
3. Hibernate: select * from client c left outer join deposit d on
   c.id=d.client_id left outer join account a on c.id=a.client_id where c.id in
   (51, 52, …)
```

Плюсы и минусы запроса с **@EntityGraph**

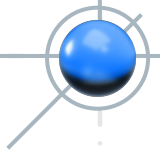
- Всегда 3 запроса
- Решает проблему N+1
- Решает проблему декартова произведения

Плюсы и минусы запроса с **@EntityGraph**

- Всегда 3 запроса
- Решает проблему N+1
- Решает проблему декартова произведения

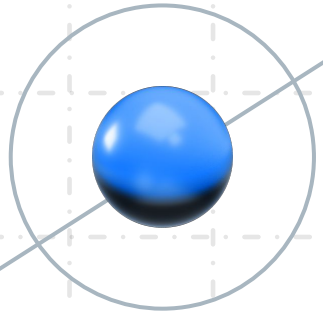
- Проблема декартова произведения решается **частично**
- Ручная реализация

Что быстрее –
@BatchSize или разделенный запрос с @EntityGraph?



Время выполнения

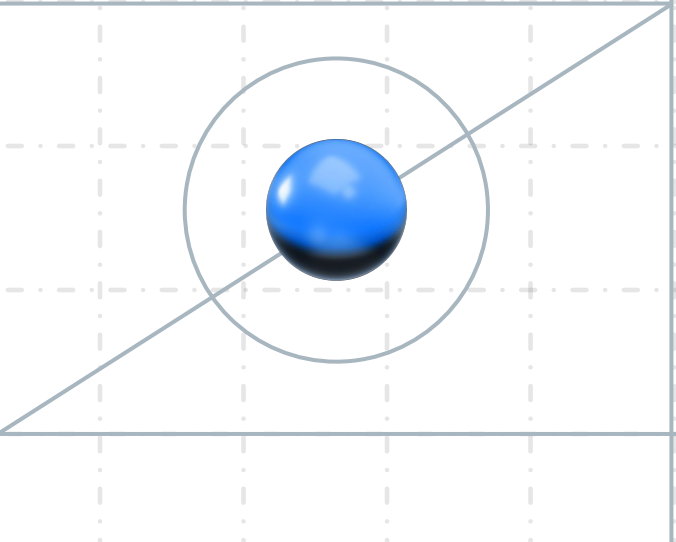
при различных реализациях
в тестовом приложении

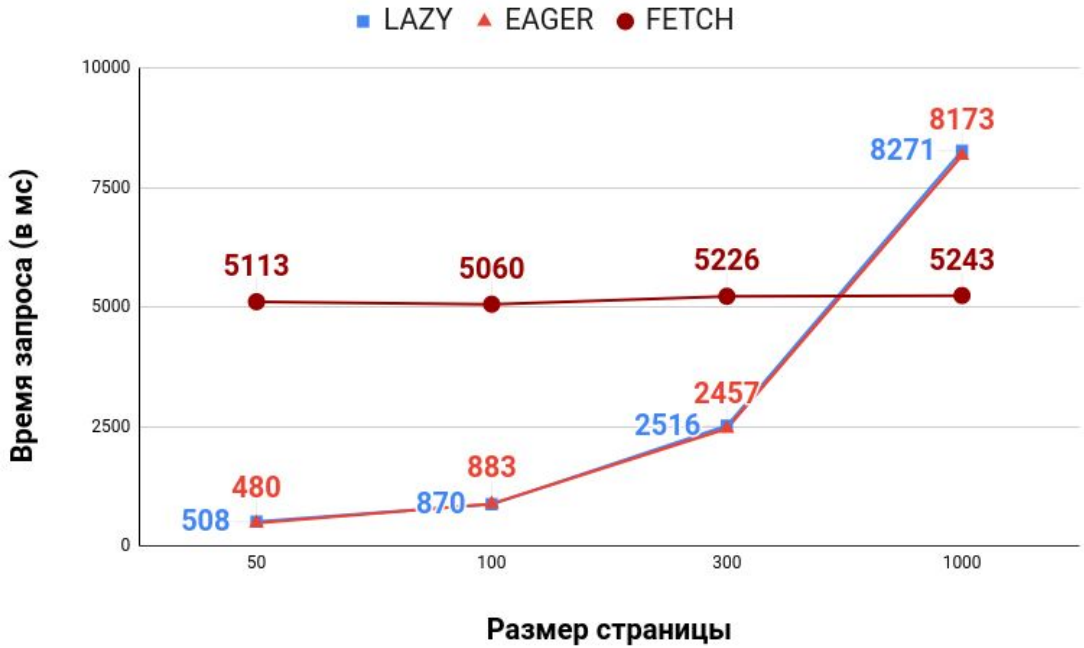


*** Результат теста зависит от производительности сервера**

Время выполнения

при различных реализациях
в тестовом приложении

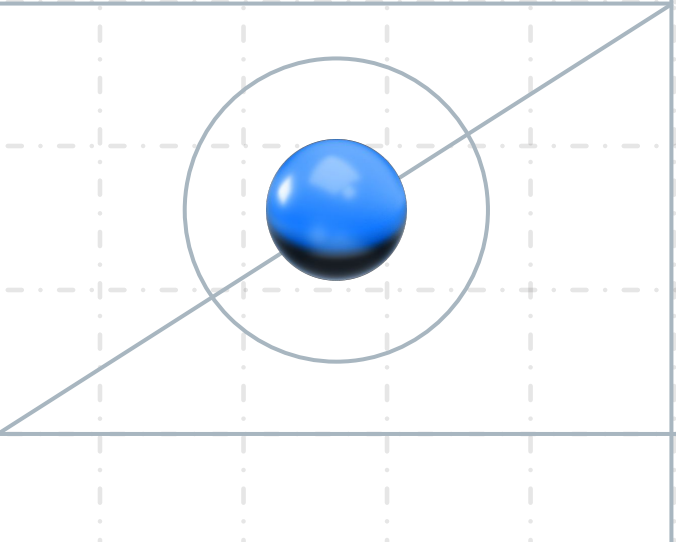


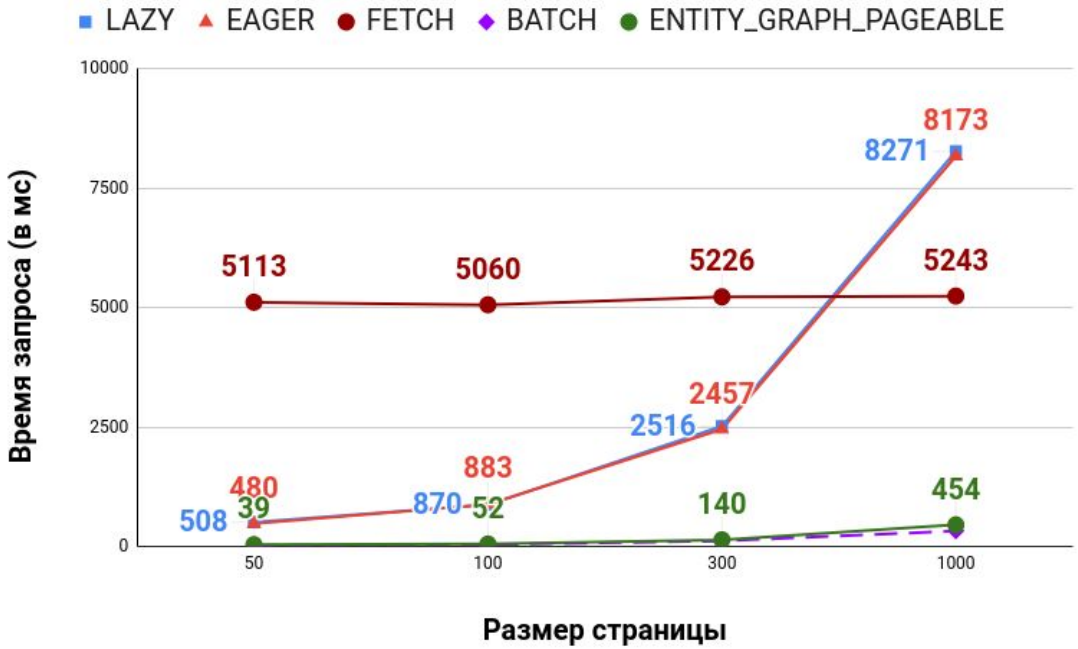


*** Результат теста зависит от производительности сервера**

Время выполнения

при различных реализациях
в тестовом приложении

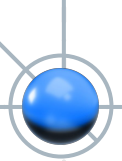




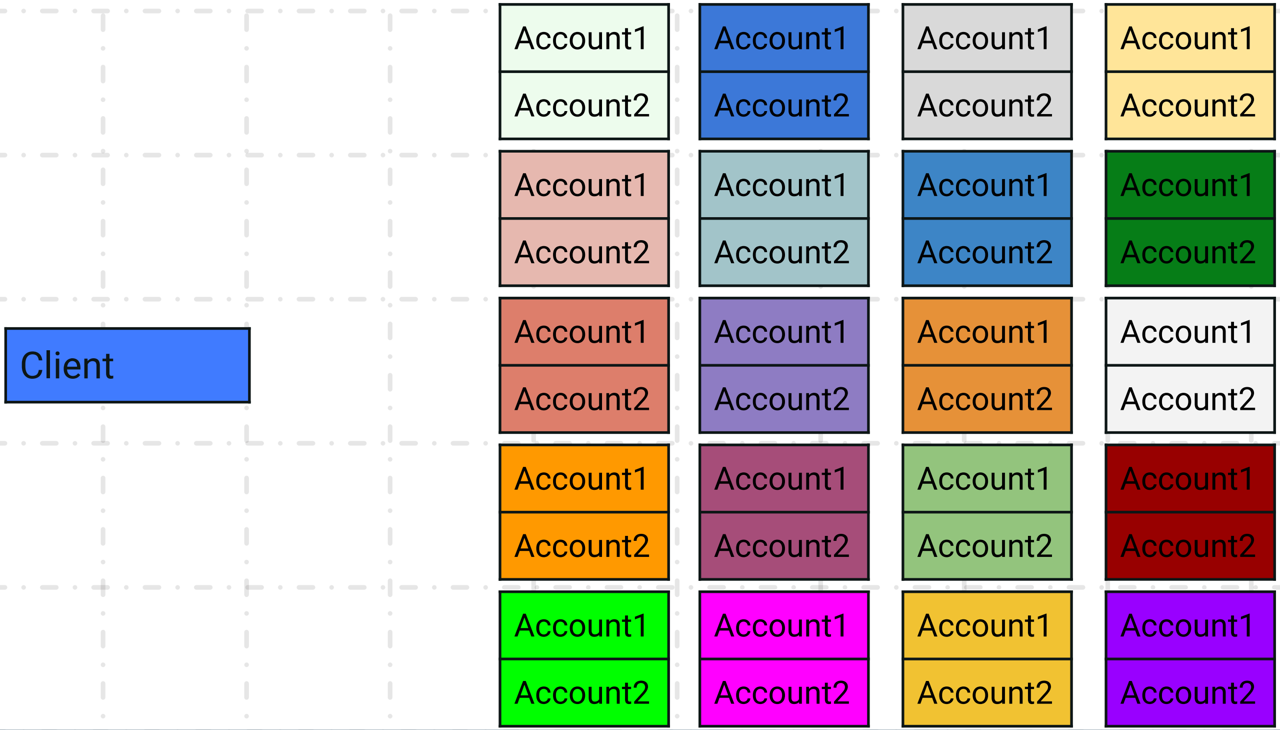
*** Результат теста зависит от производительности сервера**

Проведём эксперимент

- Создадим сущность **Client** с 20 ассоциациями **@OneToMany**
- В каждой коллекции будет по 2 аккаунта
- Выполним запросы с загрузкой 5, 10, 15 и 20 коллекций с помощью **@BatchSize** и **разделенного запроса с @EntityGraph**
- Оценим результаты работы на графике



Проведём эксперимент



Проведём эксперимент

```
@Entity
@Table(name = "CLIENT_TEST")
class ClientEntityTest(
    @Id
    @Column(name = "ID")
    var id: Long? = null,
    @OneToMany(mappedBy = "client")
    var accounts1: Set<Account1> = mutableSetOf(),
    @OneToMany(mappedBy = "client")
    var accounts2: Set<Account2> = mutableSetOf(),
    @OneToMany(mappedBy = "client")
    var accounts3: Set<Account3> = mutableSetOf(),
    @OneToMany(mappedBy = "client")
    var accounts4: Set<Account4> = mutableSetOf(),
    // ...
)
```

```
@Entity
@Table(name = "ACCOUNT_1")
class Account1(
    @Id
    @Column(name = "ID")
    var id: Long? = null,
    @Column(name = "NUMBER")
    var number: String? = null,
    @ManyToOne
    @JoinColumn(name = "CLIENT_ID")
    var client: ClientEntityTest? = null
)
```

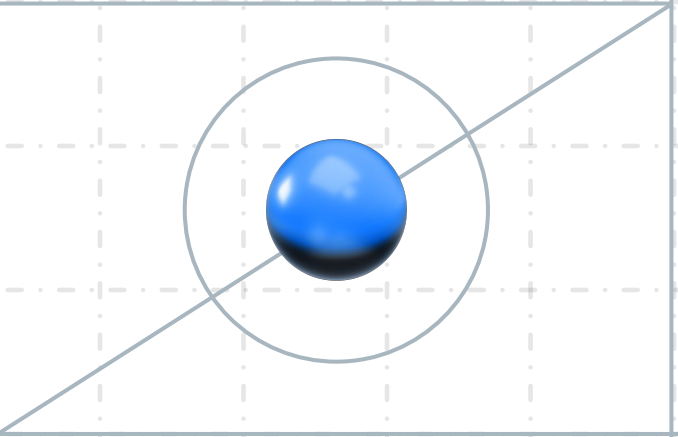
Проведём эксперимент

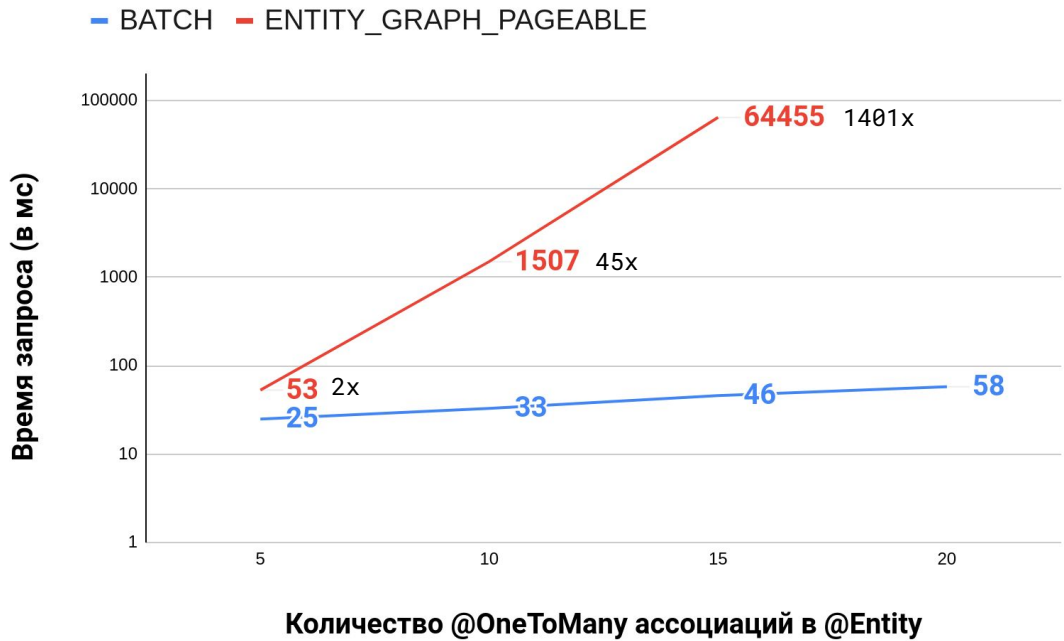
```
@Entity
@Table(name = "CLIENT_TEST")
class ClientEntityBatchTest(
    @Id
    @Column(name = "ID")
    var id: Long? = null,
    @OneToMany(mappedBy = "client")
    @BatchSize(size = 50)
    var accounts1: Set<Account1> = mutableSetOf(),
    @OneToMany(mappedBy = "client")
    @BatchSize(size = 50)
    var accounts2: Set<Account2> = mutableSetOf(),
    @OneToMany(mappedBy = "client")
    @BatchSize(size = 50)
    var accounts3: Set<Account3> = mutableSetOf(),
    // ...
)
```

```
@Entity
@Table(name = "ACCOUNT_1")
class Account1(
    @Id
    @Column(name = "ID")
    var id: Long? = null,
    @Column(name = "NUMBER")
    var number: String? = null,
    @ManyToOne
    @JoinColumn(name = "CLIENT_ID")
    var client: ClientEntityBatchTest? = null
)
```

Время выполнения

@BatchSize и **разделенного запроса с @EntityGraph** для списка из 100 клиентов

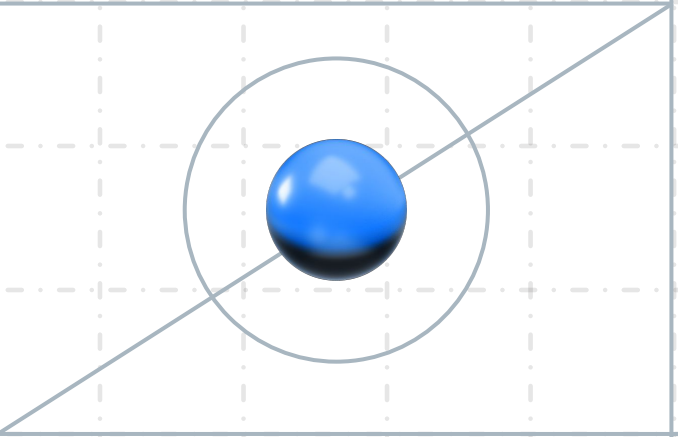


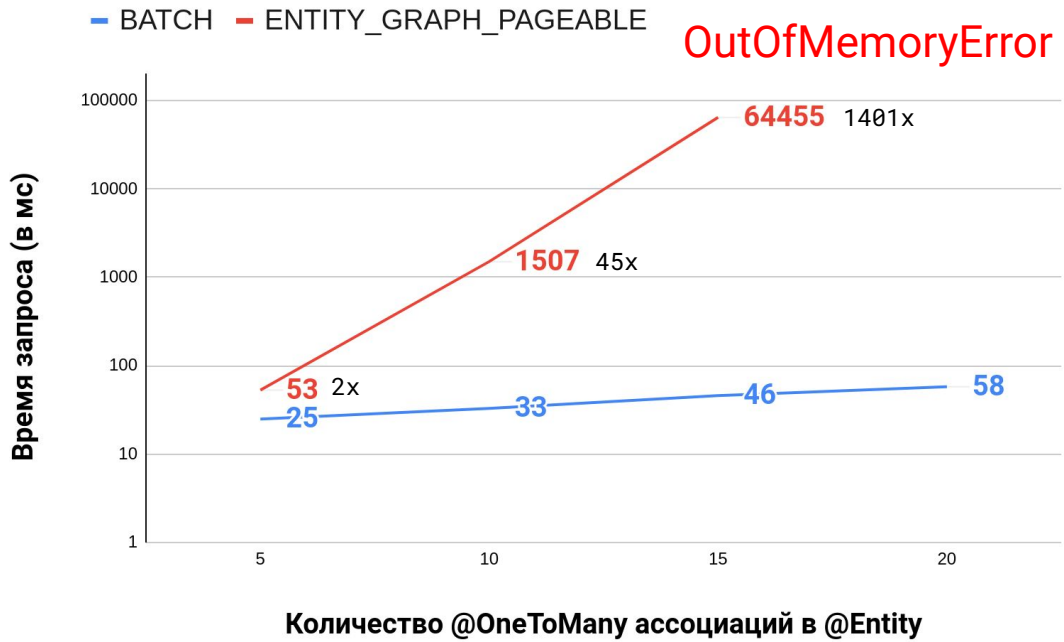


*** Результат теста зависит от производительности сервера**

Время выполнения

@BatchSize и **разделенного запроса с @EntityGraph** для списка из 100 клиентов





*** Результат теста зависит от производительности сервера**

Разделение запроса на 2 части с использованием @EntityGraph


```
@Repository
interface ClientRepository : JpaRepository<ClientEntity, Long> {

    @Query("select c.id from ClientEntity c")
    fun getAllIds(pageable: Pageable): Page<Long>

    @EntityGraph(attributePaths = ["account1", "account2", ...])
    fun getAllByIdIn(clientIds: List<Long>): List<ClientEntity>
}

val clientIds = clientRepository.getAllIds(PageRequest.of(pageNumber, pageSize))
val result = clientRepository.getAllByIdIn(clientIds.content)
```

Декартово произведение запроса с @EntityGraph

Запрос для 5 коллекций с @OneToMany:

```
Hibernate: select * from client c left outer join account_1 a_1 on
c.id=a_1.client_id left outer join account_2 a_2 on c.id=a_2.client_id left outer
join account_3 a_3 on c.id=a_3.client_id left outer join account_4 a_4 on
c.id=a_4.client_id left outer join account_5 a_5 on c.id=a_5.client_id where c.id
in (?, ?, …)
```

Декартово произведение запроса с @EntityGraph

Количество строк результата запроса для **одного** клиента в зависимости от количества **@OneToMany** с двумя аккаунтами в каждой коллекции

- 2^5 = 32
- 2^10 = 1 024
- 2^15 = 32 768
- 2^20 = 1 048 576

Чтобы получить **одного** клиента выгружаем **1 048 576 строк** -> КПД **0,000095%**

Запросы с @BatchSize

```
1.  Hibernate: select * from client c limit ? offset ?
2.  Hibernate: select count(c.id) from client c
3.  Hibernate: select * from account_1 a_1 where a_1.client_id in (?, ?, …)
4.  Hibernate: select * from account_2 a_2 where a_2.client_id in (?, ?, …)
5.  Hibernate: select * from account_3 a_3 where a_3.client_id in (?, ?, …)
6.  Hibernate: select * from account_4 a_4 where a_4.client_id in (?, ?, …)
7.  Hibernate: select * from account_5 a_5 where a_5.client_id in (?, ?, …)
8.  Hibernate: select * from account_6 a_6 where a_6.client_id in (?, ?, …)
9.  ...
```

В случае **@BatchSize** сколько отношений **@OneToMany** будет в основной сущности столько и дополнительных запросов будет выполнено

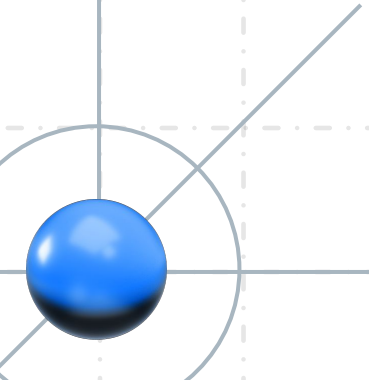
Плюсы и минусы запроса с **@EntityGraph**

- Всегда 3 запроса
- Решает проблему N+1
- Решает проблему декартова произведения

- Проблема декартова произведения решается частично
- Ручная реализация
- **Медленнее @BatchSize**
- **Может приводить к OutOfMemoryError**

В результате

- На основе данного доклада исправлен баг в Hibernate 6
- В JPA Buddy будет добавлена подсказка для in memory pagination



Тестовое приложение

LAZY | EAGER | BATCH | CRITERIA FETCH | ENTITY GRAPH | ENTITY GRAPH PAGEABLE

Request time: 0.41

Rows per page: 50 1–50 of 10000

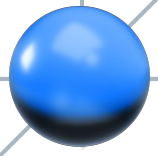
Client name	Total accounts	Total accounts balance	Total deposits	Average deposits rate	Total loans	Average loans rate	Total transfers	Total transfers amount
Филиппова В. А.	4	15483276	0	0	1	13.00	9	39269480
Никитина Е. Э.	2	12215981	1	10.00	1	6.00	11	61937937
Тарасова А. В.	2	9260506	2	6.50	0	0	9	36059436
Лапин Р. Ф.	4	22140272	2	4.00	1	14.00	10	57351697

APP

API

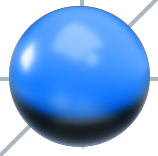
Вопросы реализации @BatchSize

- Почему всё ещё необходимо указывать размер size?
- Почему не используется передача массива в in?
- Есть идеи и предложения как улучшить?
- Предлагаю обсудить в задаче проекта Hibernate



Выводы

- Обращайте внимание на сообщения “firstResult/maxResults specified with collection fetch; applying in memory”
- Используйте **@BatchSize**



Q&A

Вопросы и ответы

Артём Гордиенко
Росбанк

@arvgord
arvgord@gmail.com
github.com/arvgord