

**Spring Data JDBC. Где мы  
находимся сейчас**

# Введение

## whoami

- 3 года в коммерческой Java разработке
- Занимаюсь всякими аналитическими штуками на производстве
- В свободное время люблю потыкать Spring и Spring Data



# Введение

## Содержание

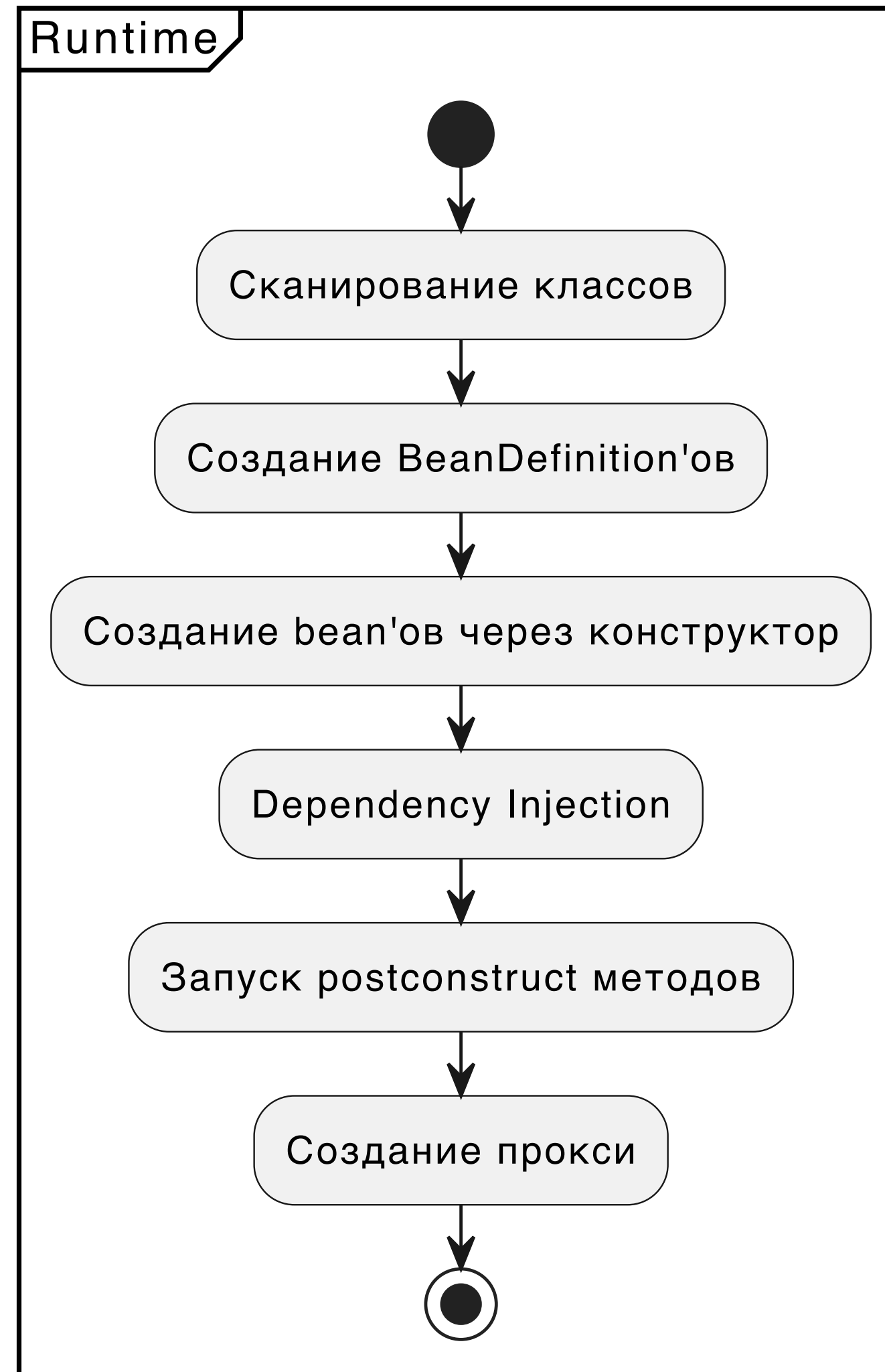
- AOT репозитории
- Composite Keys
- Single Query Loading
- Type-Safe Property Paths
- Scrolling API

# АОТ репозитории

**АОТ - общий вектор развития  
Spring на данный момент**

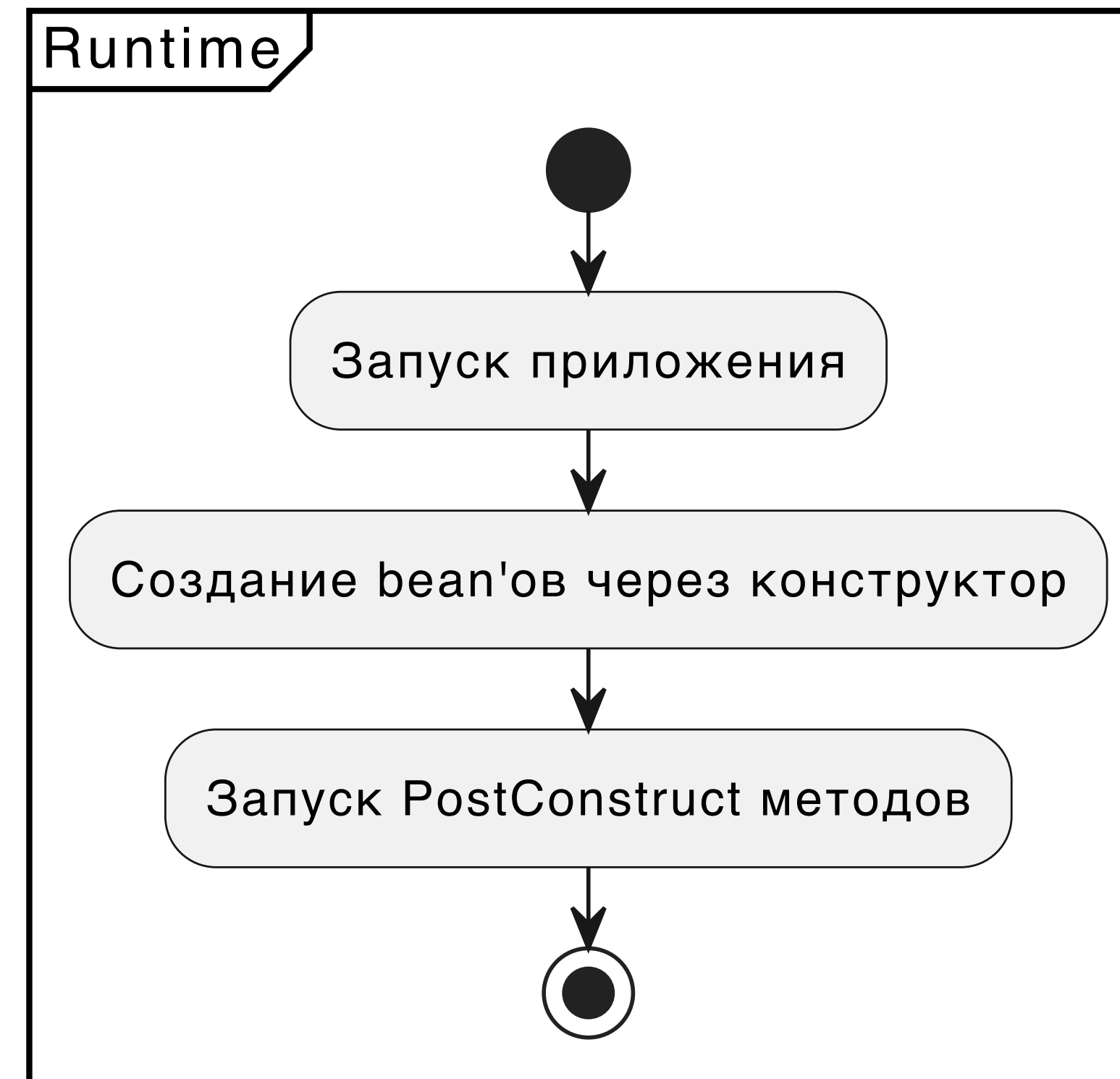
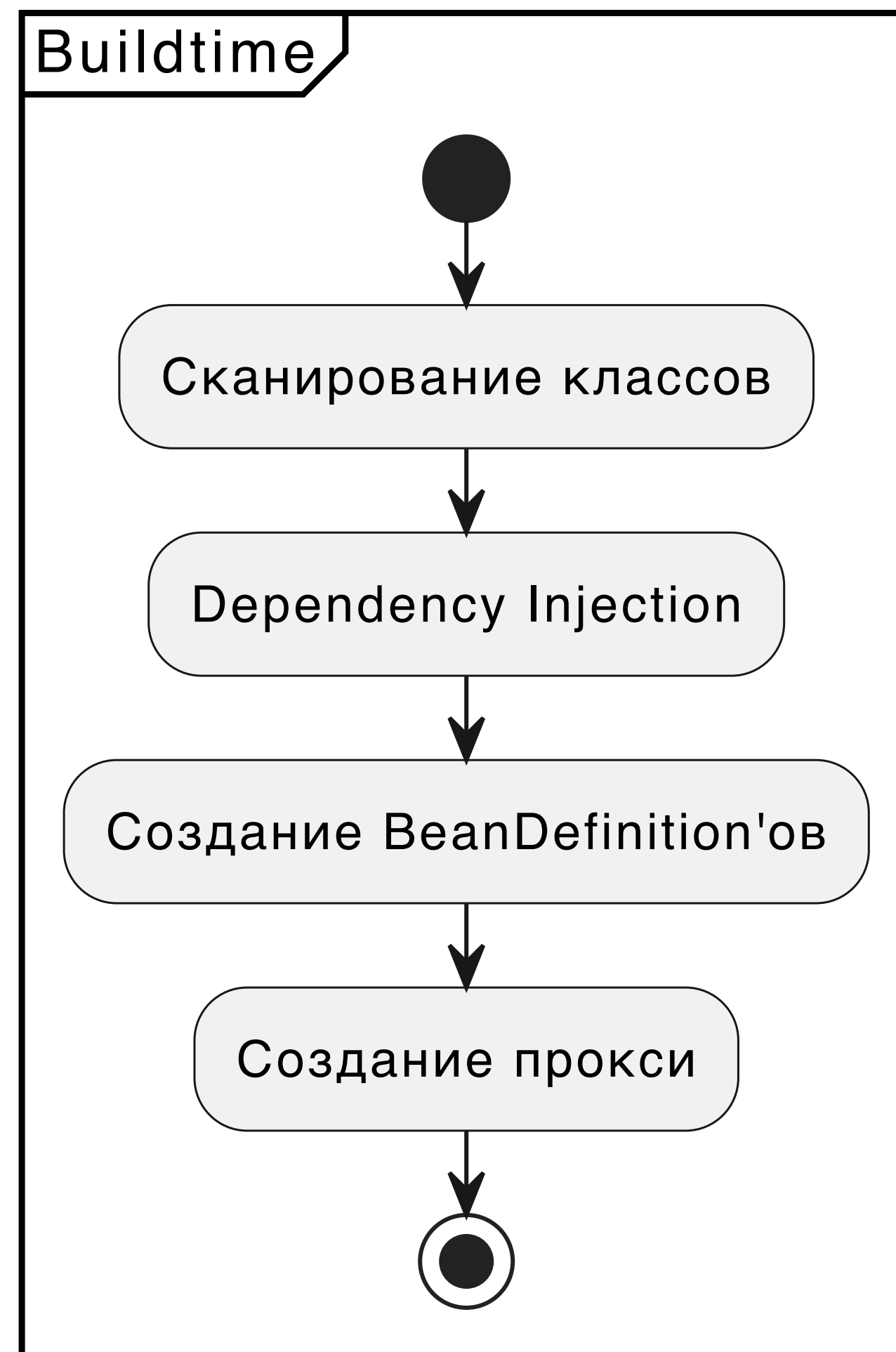
# АОТ репозитории. Почему это круто?

## Немного про АОТ



# AOT репозитории. Почему это круто?

## Немного про AOT



# **АОТ репозитории. Почему это круто?**

**В чем проблема обычных репозиториев?**

- Генерируются в рантайме

# АОТ репозитории. Почему это круто?

В чем проблема обычных репозиторий?

- Генерируются в рантайме
- Derived-query не валидируются при сборке

# АОТ репозитории. Почему это круто?

В чем проблема обычных репозиторий?

- Генерируются в рантайме
- Derived-query не валидируются при сборке
- Сложность дебага из-за обильного количества прокси и рефлексий

# АОТ репозитории. Почему это круто?

Что нам привнесли АОТ репозитории?

- Fail-fast благодаря build-time генерации derived-query методов

# АОТ репозитории. Почему это круто?

Что нам привнесли АОТ репозитории?

- Fail-fast благодаря build-time генерации derived-query методов
- Удобный дебаг из-за меньшего количества магии

# АОТ репозитории. Почему это круто?

## Что нам привнесли АОТ репозитории?

- Fail-fast благодаря build-time генерации derived-query методов
- Удобный дебаг из-за меньшего количества магии
- Более предсказуемый результат и удобный tooling благодаря сгенерированной в build-time мета информации

# АОТ репозитории. Почему это круто?

Что мы можем использовать на данный момент?

- Derived query и @Query методы
- @Modifying методы возвращающие void, int и long
- Методы возвращающие Page, Slice, Stream, и Optional
- Проекции
- Value Expressions (SpEL)
- Методы, принимающие в качестве аргумента ScrollPosition не поддерживаются.

# AOT репозитории. Как включить?

## Maven



```
<plugin>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-maven-plugin</artifactId>
  <executions>
    <execution>
      <id>process-aot</id>
      <goals>
        <goal>process-aot</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

# AOT репозитории. Как включить?

## Gradle



```
plugins {  
    // ...  
    id( "org.springframework.boot.aot" ) version "4.0.3"  
}
```

# AOT репозитории. Как включить? Gradle

spring-projects / spring-boot

Code Issues 475 Pull requests 13 Actions Wiki Security Insights

## Mention using org.springframework.boot.aot Gradle plugin directly for AOT processing with the JVM #49307

Edit New issue

Closed

chrshnv opened 2 days ago

The Gradle plugin "org.springframework.boot.aot" is mentioned in the general documentation about AOT in Spring Boot, but I would like to see it mentioned in the Gradle plugin documentation. Currently, the AOT section of the Gradle plugin documentation only mentions the Graal VM Native Image plugin.

spring-projects-issues added status: waiting-for-tri... 2 days ago

snicoll 2 days ago Member

@chrshnv thanks for the suggestion. Can you perhaps open a PR to document what you think is missing? We're happy to improve the doc but "see it mentioned" is a bit vague.

Assignees: wilkinsona

Labels: type: documentation

Type: No type

Projects: No projects

Milestone: 3.5.12 Due by March 19, 2026, 100% complete

Relationships: None yet

Display a menu

# АОТ репозитории. Как включить?

## Общие действия

- Включить АОТ контекст через `spring.aot.enabled`

# АОТ репозитории. Как включить?

## Общие действия

- Включить АОТ контекст через `spring.aot.enabled`
- Сконфигурировать `JdbcDialect`

# АОТ репозитории. Как включить?

## Общие действия

- Включить АОТ контекст через `spring.aot.enabled`
- Сконфигурировать `JdbcDialect`



```
@Bean
JdbcDialect jdbcDialect() {
    return JdbcPostgresDialect.INSTANCE;
}
```

# Demo

# AOT Репозитории

## И интересные баги

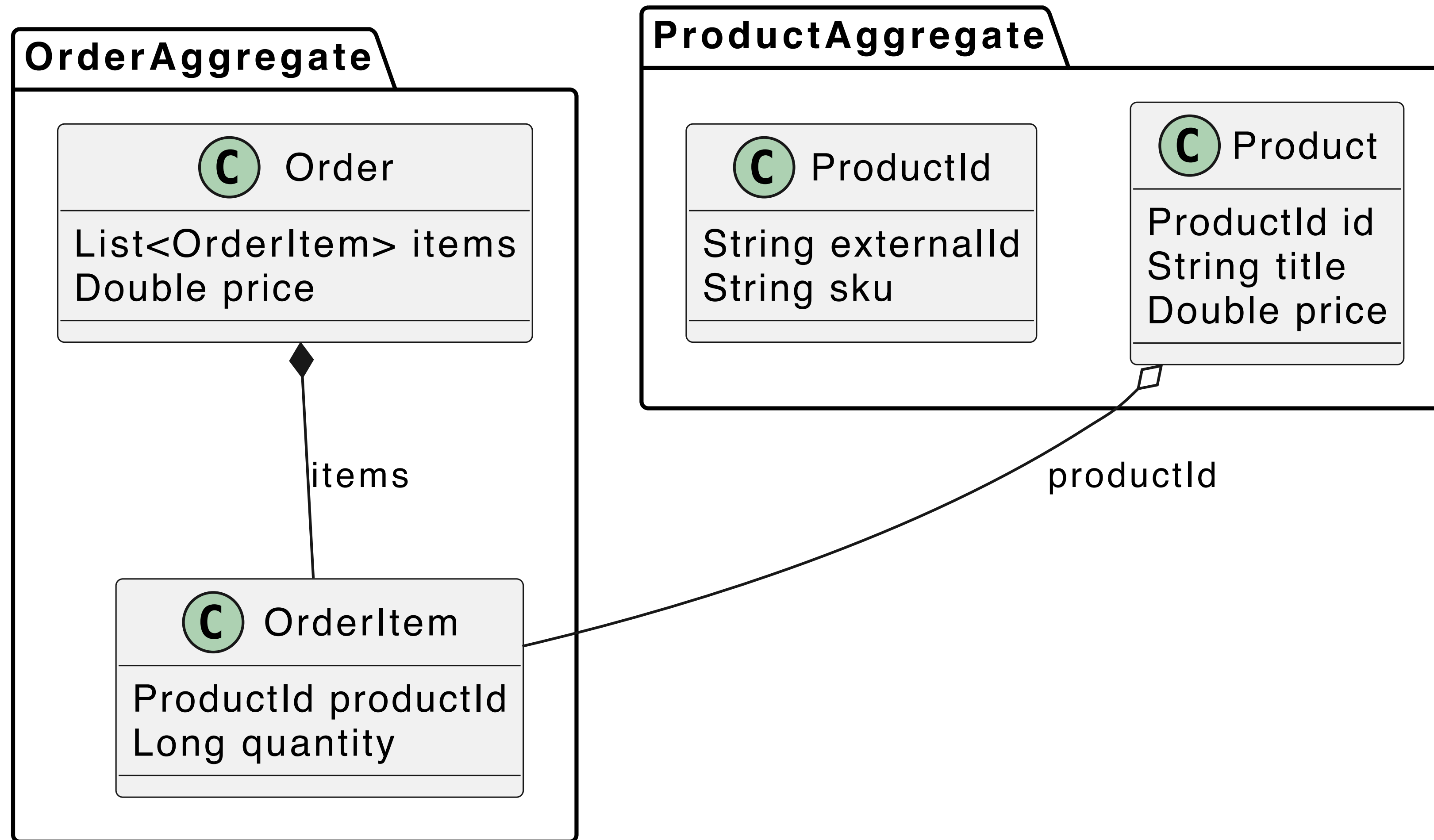
The screenshot shows a GitHub issue page for the repository 'spring-projects / spring-data-relational'. The issue title is 'AOT query metadata does not match actual query #2239', which is marked as 'Closed'. The issue was opened by user 'chrshnv' on February 12. The main content of the issue includes a Java interface definition for 'PersonRepository' and two SQL queries: one from repository metadata and one from a debugger. The repository metadata SQL query is: `SELECT "person"."id" AS "id", "person"."last_name" AS "last_name", "person"."first_name" AS "first_name" FROM "person" WHERE "person"."first_name" = UPPER(:first_name)`. The debugger SQL query is: `SELECT "person"."id" AS "id", "person"."last_name" AS "last_name", "person"."first_name" AS "first_name" FROM "person" WHERE UPPER("person"."first_name") = UPPER(:first_name)`. The issue history shows that 'spring-projects-issues' added the 'status: waiting-for-tri...' label on Feb 12, 'mp911de' added the 'type: bug' label and removed the 'status: waiting-for-tri...' label on Feb 12, and 'mp911de' changed the title to the current one on Feb 12. The right sidebar shows the issue is assigned to 'mp911de', has a 'type: bug' label, and no projects or milestones are associated with it. A 'Code with agent mode' button is visible at the bottom right.

# Composite keys

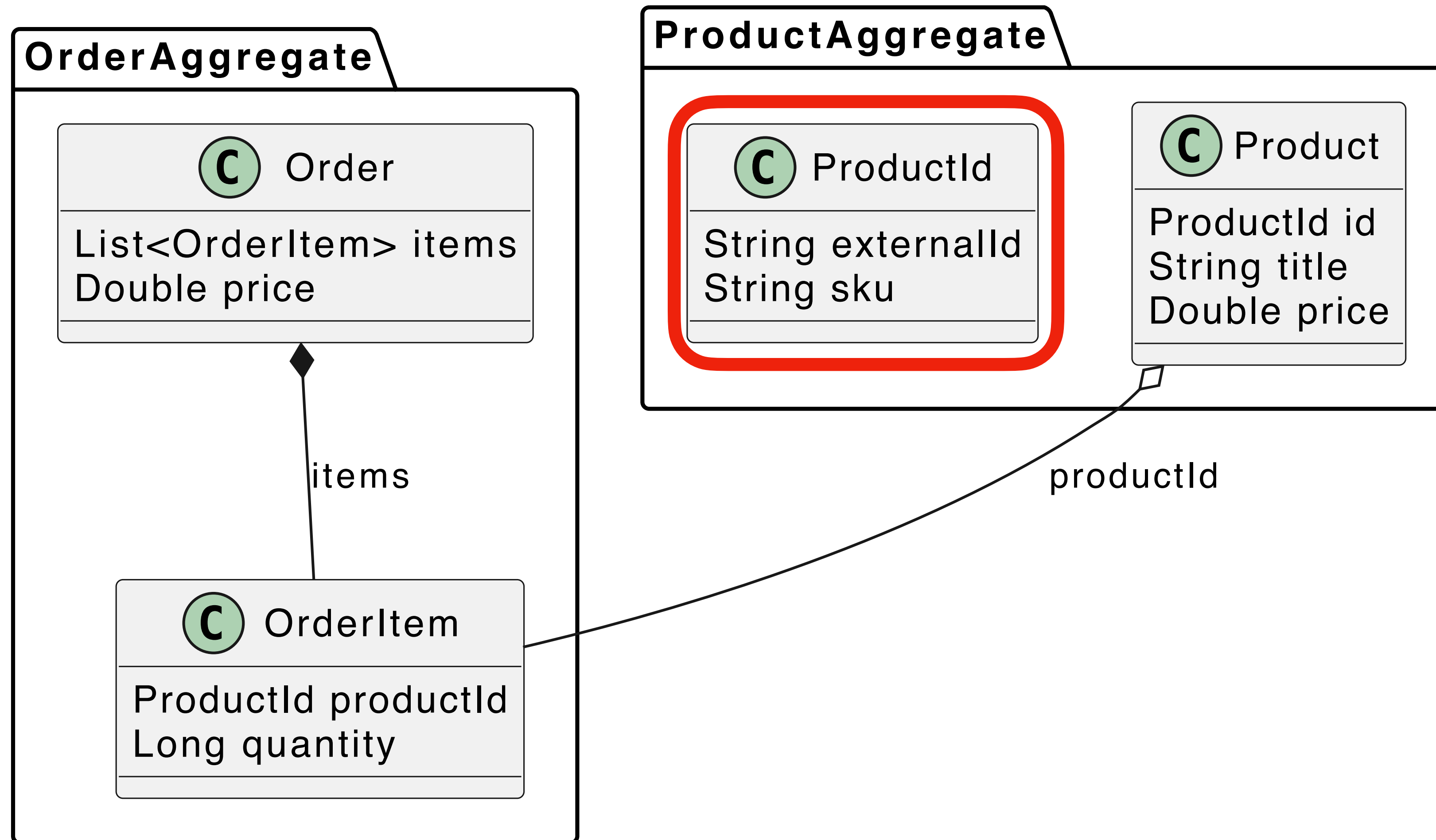
# Composite keys. Долгое ожидание

Issues opened on Apr 4, 2019

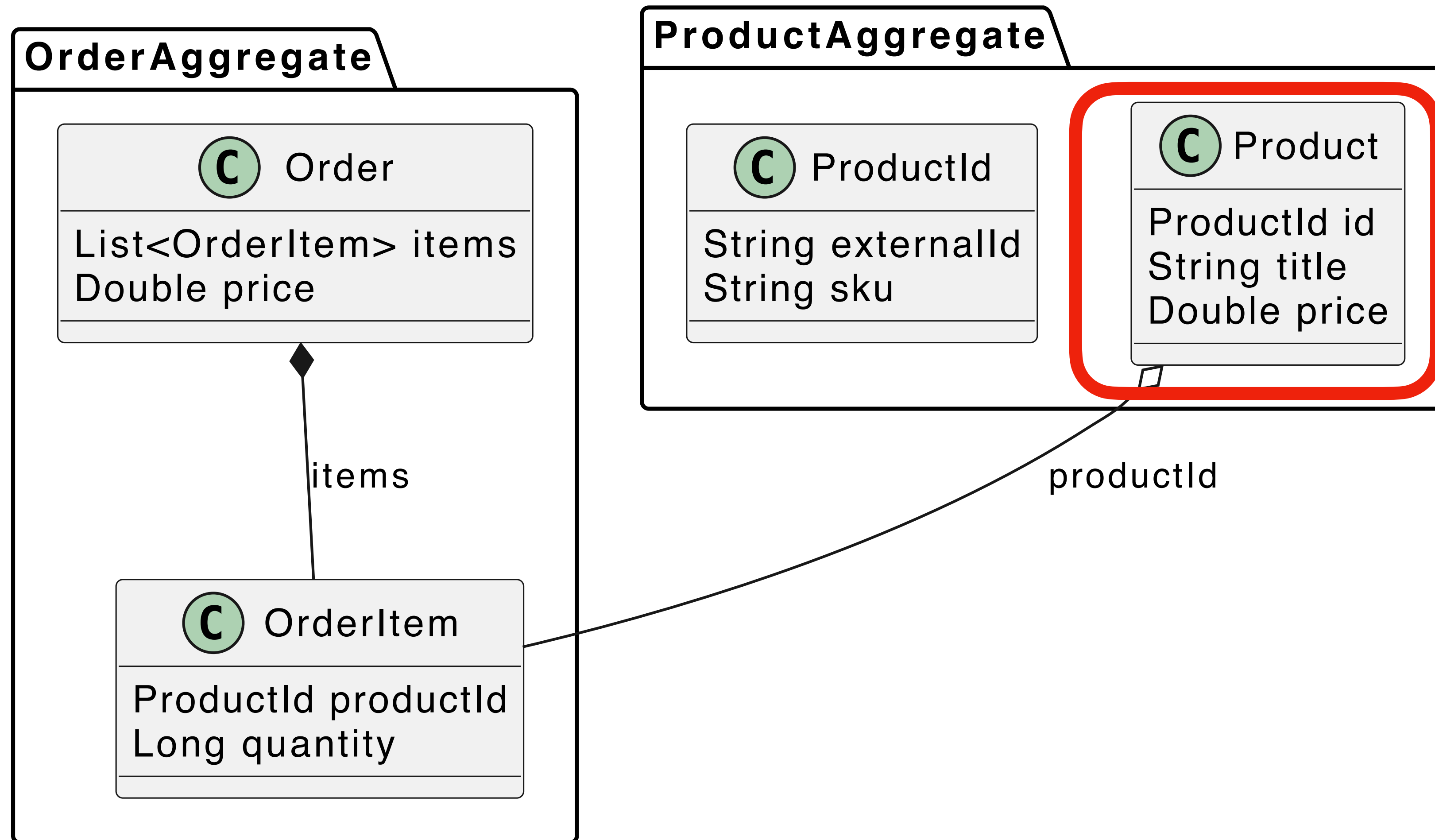
# Composite keys. Пример



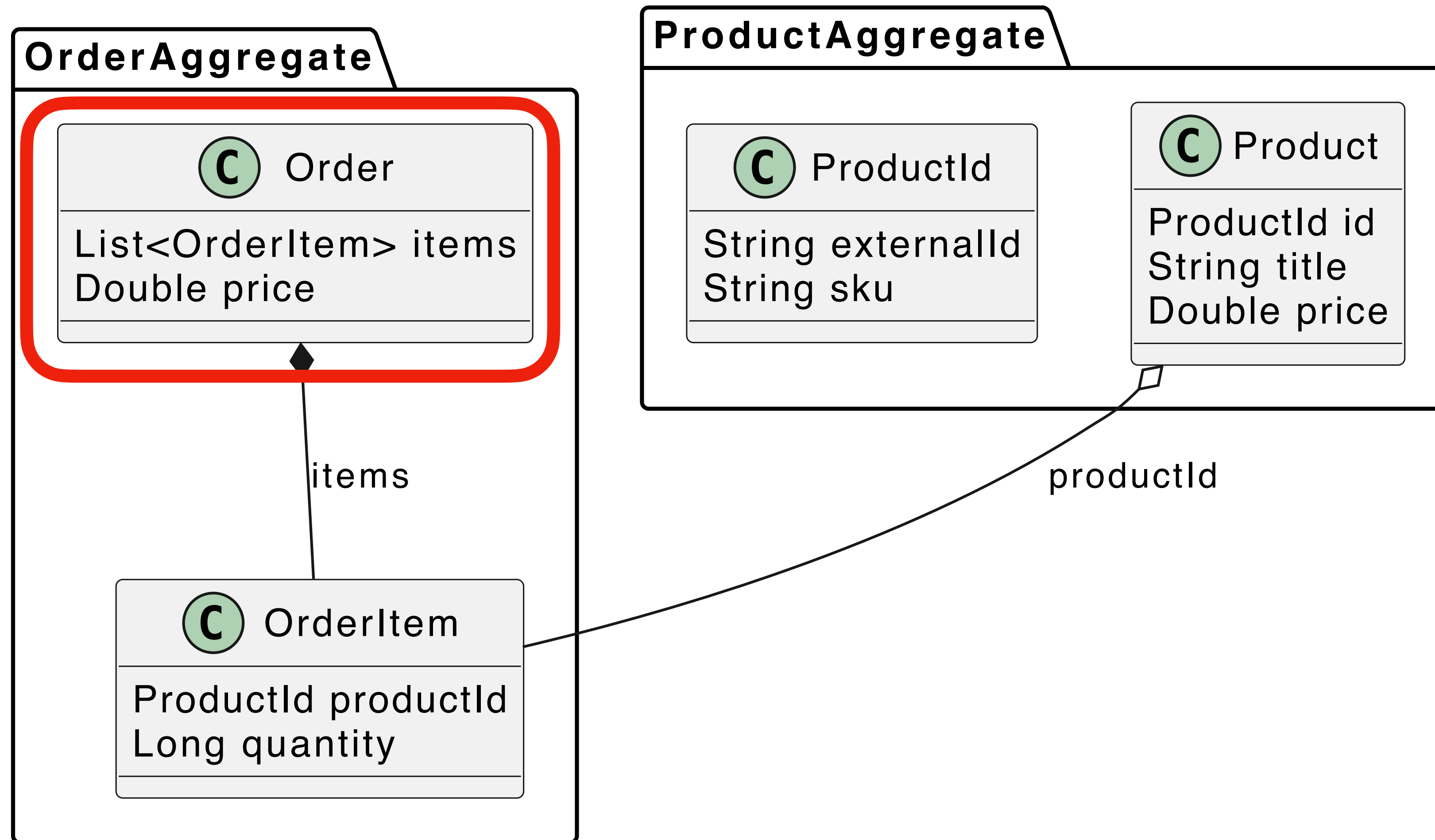
# Composite keys. Пример



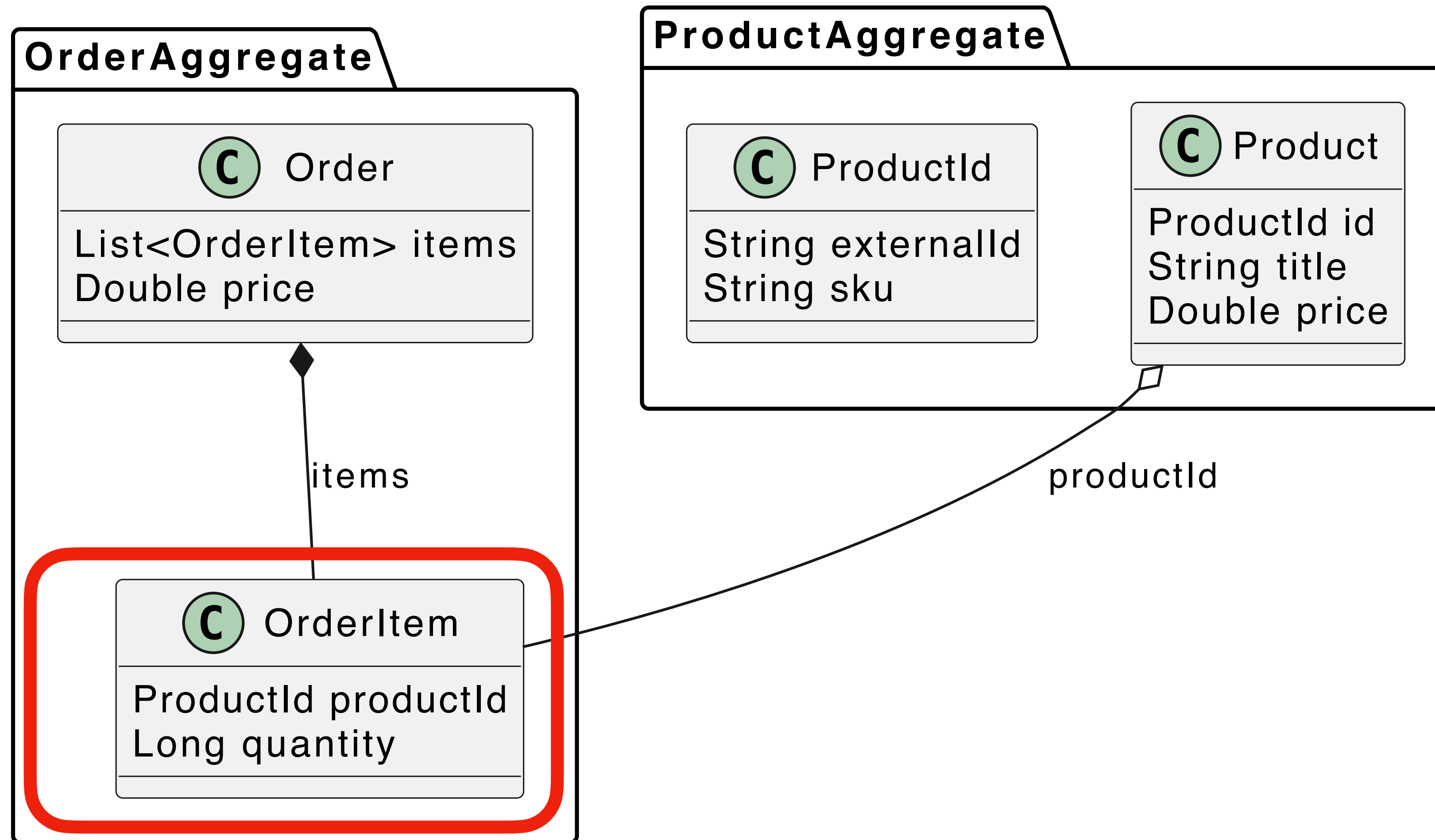
# Composite keys. Пример



# Composite keys. Пример



# Composite keys. Пример



# Demo

# Composite keys. История о нерабочем AggregateReference.

## Как правильно маппить в Spring Data JDBC?

- MappedCollection  
Подходит для внутренних частей агрегата, подгрузки 1-1, 1-n.

# Composite keys. История о нерабочем AggregateReference.

## Как правильно маппить в Spring Data JDBC?

- `MappedCollection`  
Подходит для внутренних частей агрегата, подгрузки 1-1, 1-n.
- `AggregateReference`  
Подходит для связи Aggregate-Aggregate, подгрузки 1-1, 1-n, n-1, n-m.

# Composite keys. История о нерабочем AggregateReference.

## Как правильно маппить в Spring Data JDBC?

- `MappedCollection`  
Подходит для внутренних частей агрегата, подгрузки 1-1, 1-n
- `AggregateReference`  
Подходит для связи Aggregate-Aggregate, подгрузки 1-1, 1-n, n-1, n-m
- Примитивы  
Подходит для связи Aggregate-Aggregate, подгрузки 1-1, 1-n, n-1, n-m

# Composite keys. История о нерабочем AggregateReference.

## Как правильно маппить в Spring Data JDBC?

- ~~MappedCollection~~  
~~Подходит для внутренних частей агрегата, подгрузки 1-1, 1-n~~
- AggregateReference  
Подходит для связи Aggregate-Aggregate, подгрузки 1-1, 1-n, n-1, n-m
- Примитивы  
Подходит для связи Aggregate-Aggregate, подгрузки 1-1, 1-n, n-1, n-m
- Согласно DDD мы можем ссылаться из одного агрегата на другой только при помощи Id

# Composite keys. История о нерабочем AggregateReference.

## Почему AggregateReference истинно верный?

«If you have n-1 or n-m references, you are, by definition, dealing with two separate aggregates. References between those may be encoded as simple id values, which map properly with Spring Data JDBC. A better way to encode these, is to make them instances of AggregateReference. An AggregateReference is a wrapper around an id value which marks that value as a reference to a different aggregate. Also, the type of that aggregate is encoded in a type parameter.»

# Composite keys. История о нерабочем AggregateReference.

Как правильно маппить в Spring Data JDBC?



# Composite keys. История о нерабочем AggregateReference.

## Как правильно маппить в Spring Data JDBC?

- ~~MappedCollection~~  
~~Подходит для внутренних частей агрегата, подгрузки 1-1, 1-n.~~
- AggregateReference  
Подходит для связи Aggregate-Aggregate, подгрузки 1-1, 1-n, n-1, n-m
- ~~Примитивы~~  
~~Подходит для связи Aggregate Aggregate, подгрузки 1-1, 1-n, n-1, n-m~~
- Согласно DDD мы можем ссылаться из одного агрегата на другой только при помощи Id

# Composite keys. История о нерабочем AggregateReference.

The screenshot shows a GitHub issue page for the repository 'spring-projects / spring-data-relational'. The issue title is 'Invalid SQL generated if an entity contains composite key reference #2113'. The issue is open and was created by 'belyaev-andrey' on August 13, 2025. The issue is assigned to 'schauder' and has a 'type: bug' label. The issue description includes the following code snippets:

```
Having entities:  
  
class Employee {  
    @Id  
    EmployeeId id; //Composite ID  
    //... Other fields  
}  
  
record EmployeeId(  
    Organization organization,  
    Long employeeNumber) {  
}  
  
@Table("locker")  
public class Locker {  
    @Id  
    private Long id;  
    private AggregateReference<Employee, EmployeeId> assignedTo; //Reference to employee  
}  
  
create table employee
```

The right sidebar shows the following information:

- Assignees: schauder
- Labels: type: bug
- Type: No type
- Projects: No projects
- Milestone: No milestone
- Relationships: None yet
- Development: Code with agent mode

# Demo

# Composite keys. История о нерабочем AggregateReference.

The screenshot shows a GitHub issue page for the repository 'spring-projects / spring-data-relational'. The issue title is 'Incorrect SQL for AggregateReference with composite id in findBy...In #2276'. The issue was opened 3 minutes ago by user 'chrshnv'. The issue description includes a 'Summary' section stating that derived query methods using 'AggregateReference' with a 'composite id' generate incorrect SQL. It explains that instead of expanding the composite key into multiple columns, Spring Data JDBC treats it as a single column. Below the summary is an 'Expected SQL' section with a code block containing a SQL query: 

```
SELECT "order_item"."external_id",
       "order_item"."sku",
       "order_item"."quantity"
FROM "order_item"
WHERE ("order_item"."external_id" = ? AND "order_item"."sku" = ?)
```

 The right sidebar shows the issue's metadata, including 'Assignees' (No one assigned), 'Labels' (status: waiting-for-triage), 'Type' (No type), 'Fields' (No fields configured for issues without a type), 'Projects' (No projects), 'Milestone' (No milestone), and 'Relationships' (None yet).

# Composite keys. История о нерабочем AggregateReference.

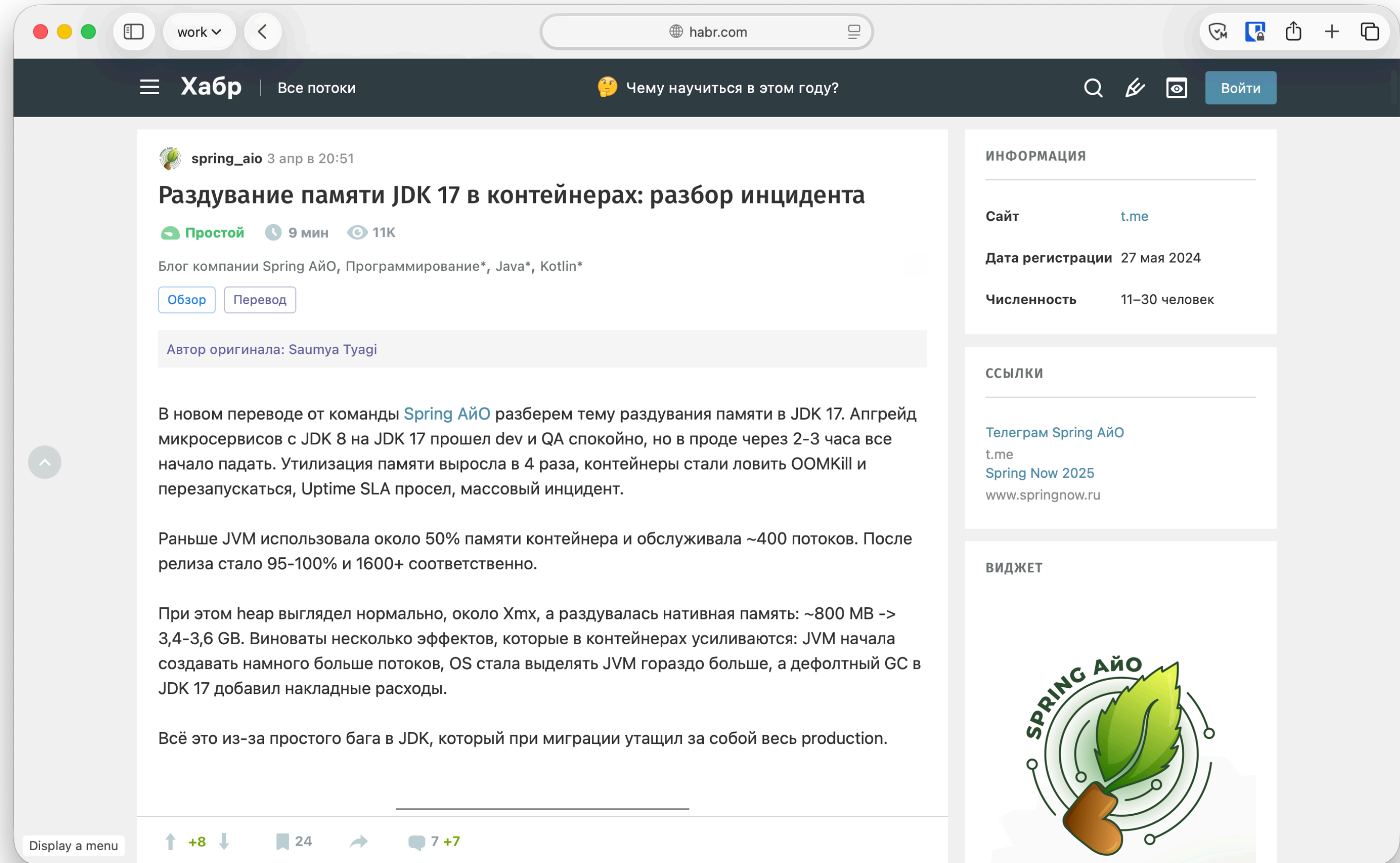
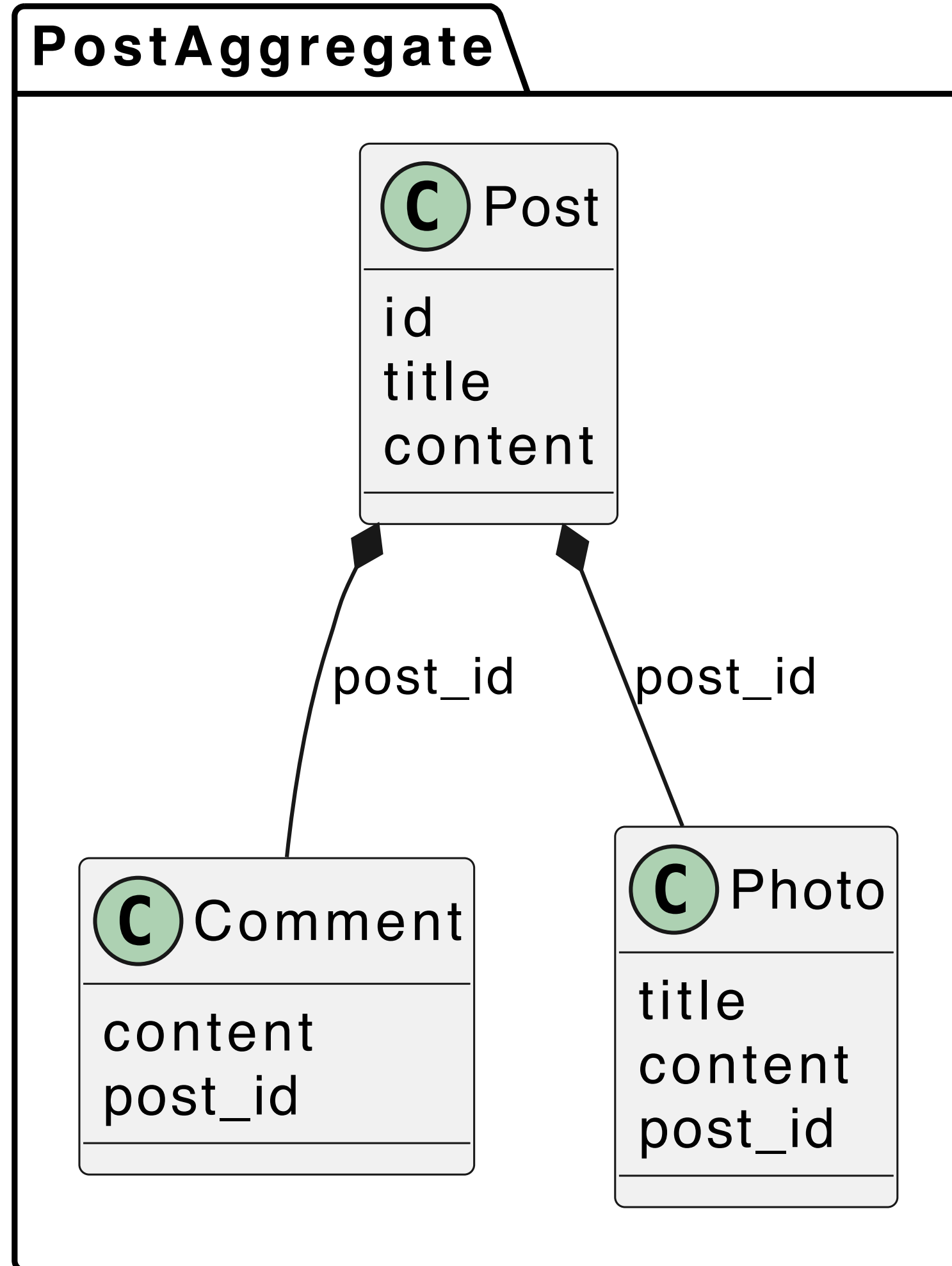


<https://github.com/spring-projects/spring-data-relational/issues/2276>

# Single Query Loading

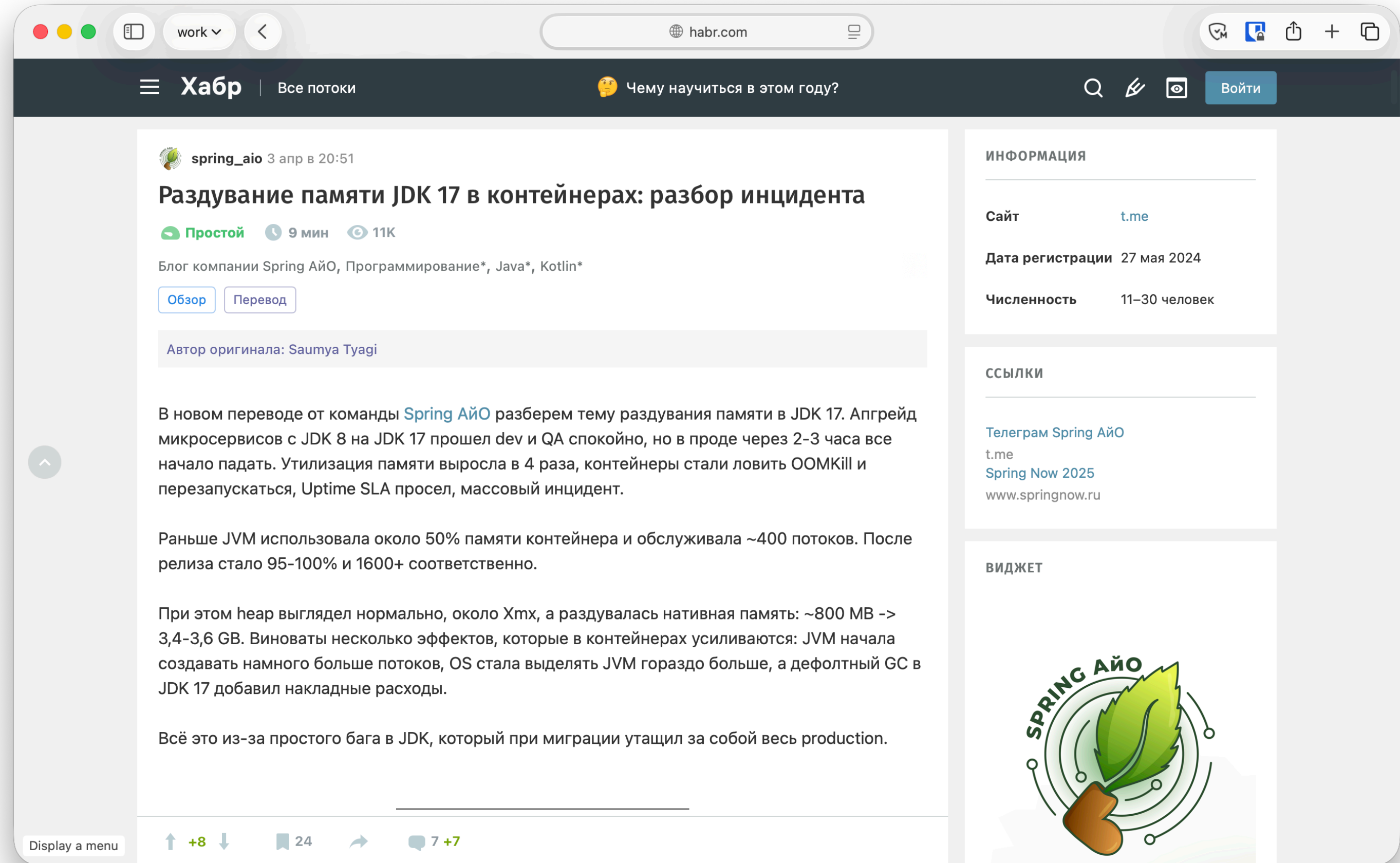
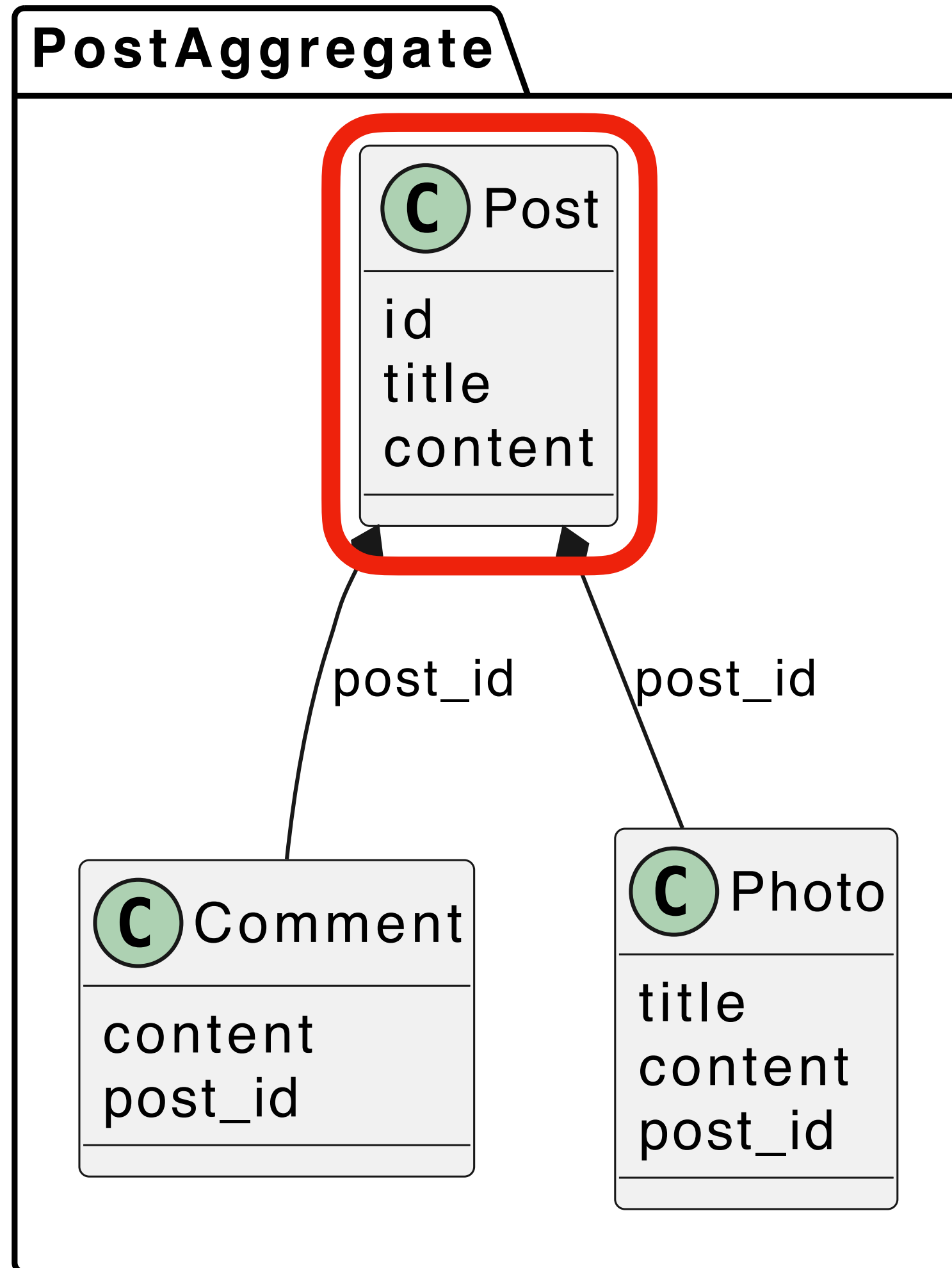
# Single Query Loading. Silver bullet?

## Пример



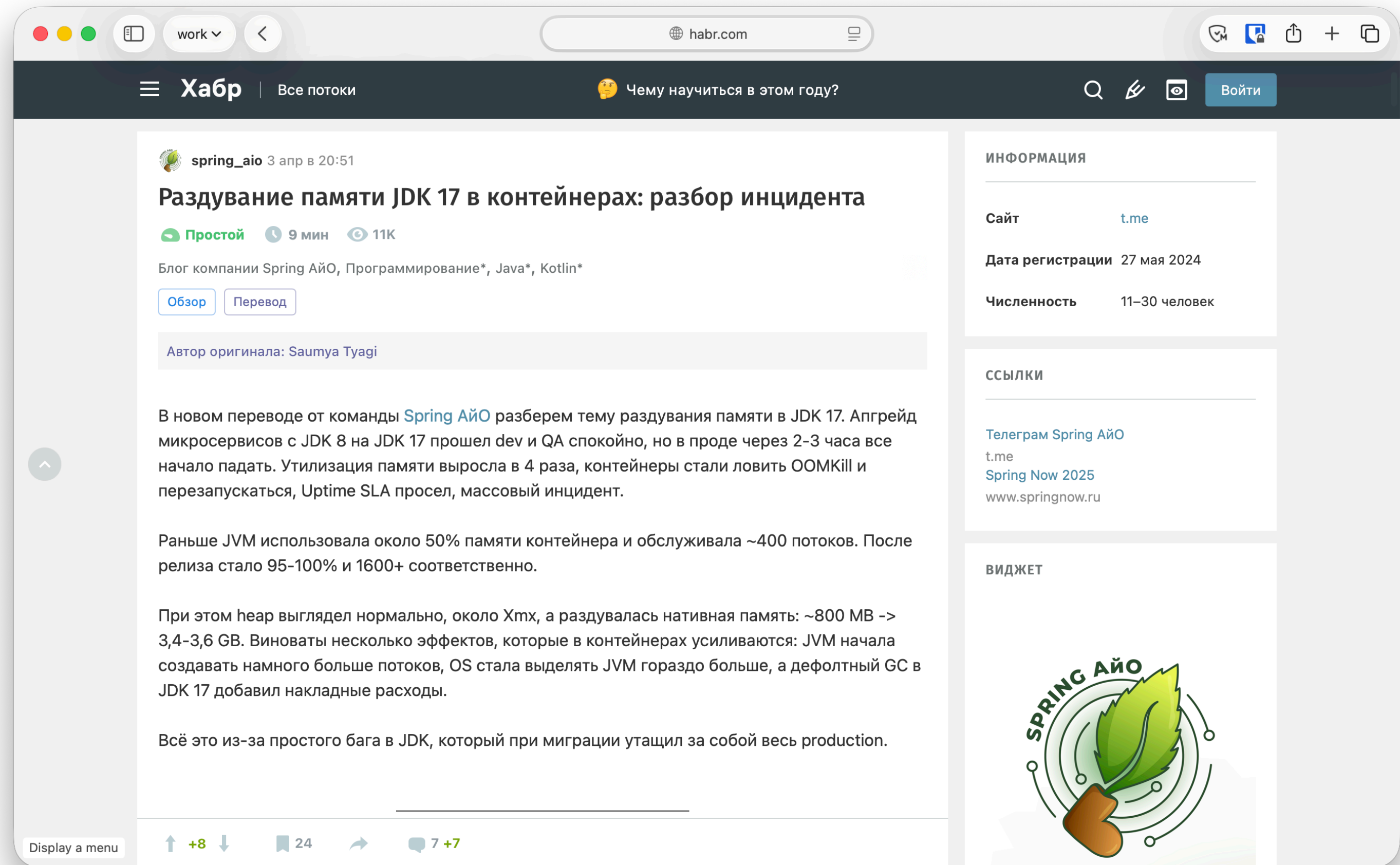
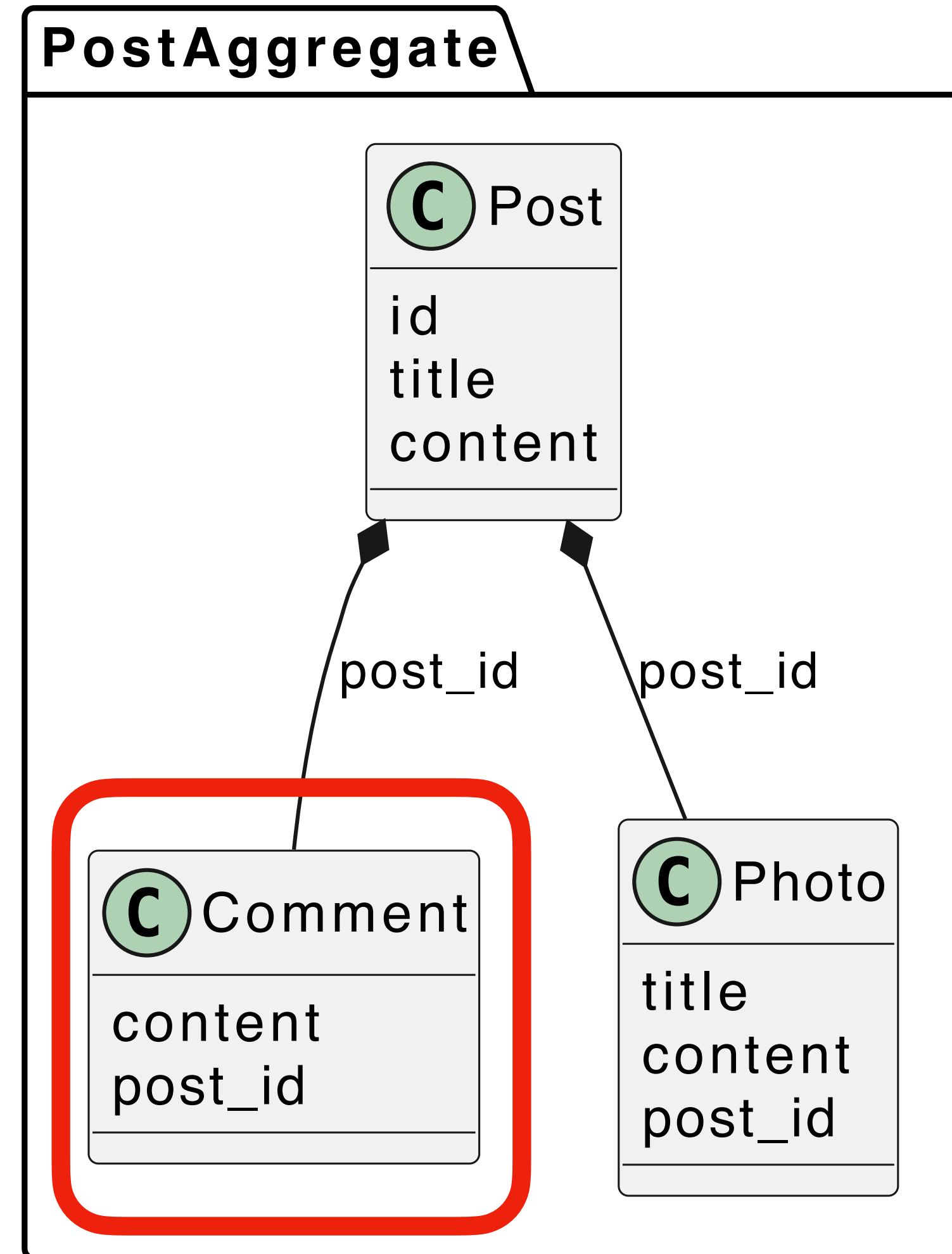
# Single Query Loading. Silver bullet?

## Пример



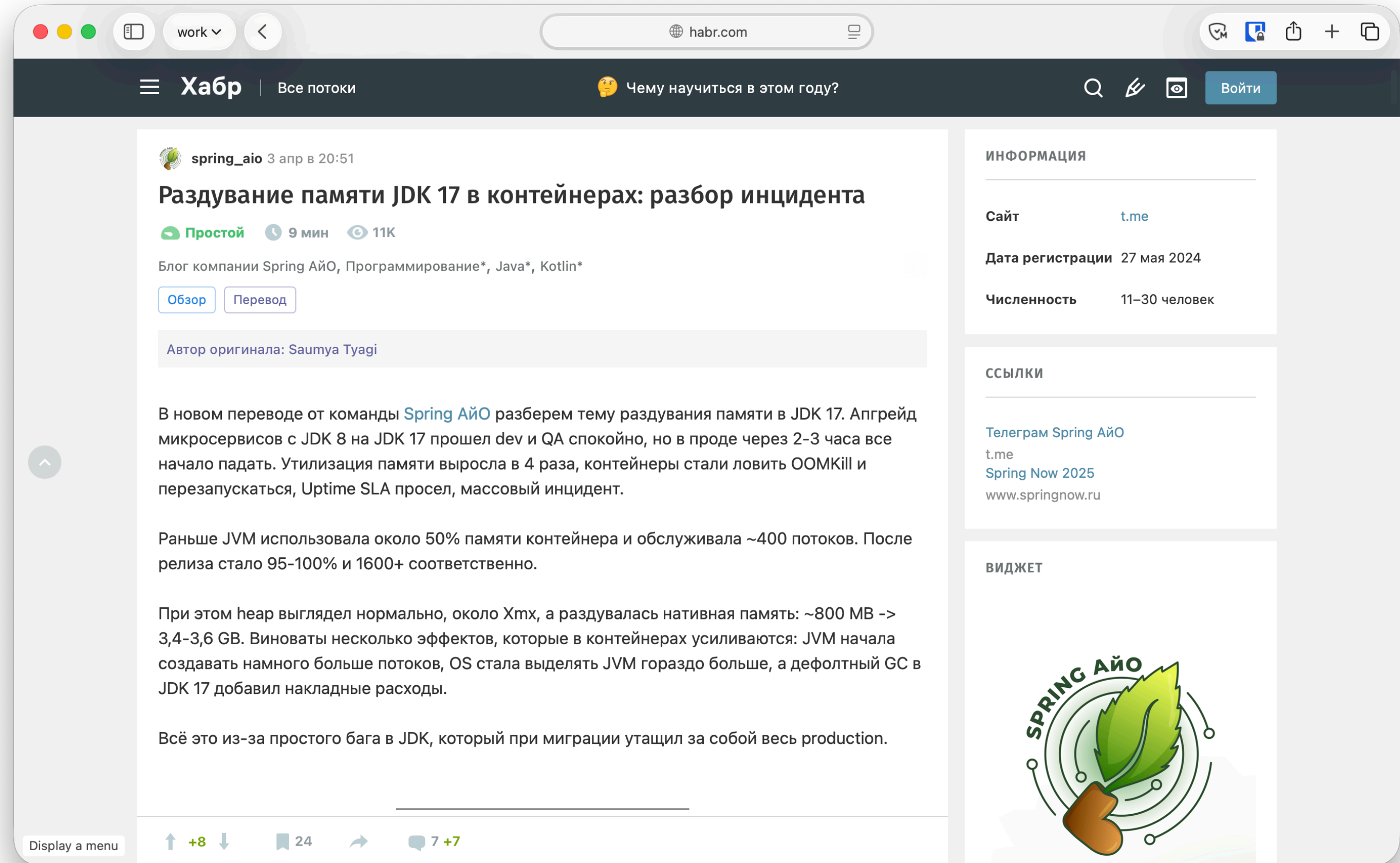
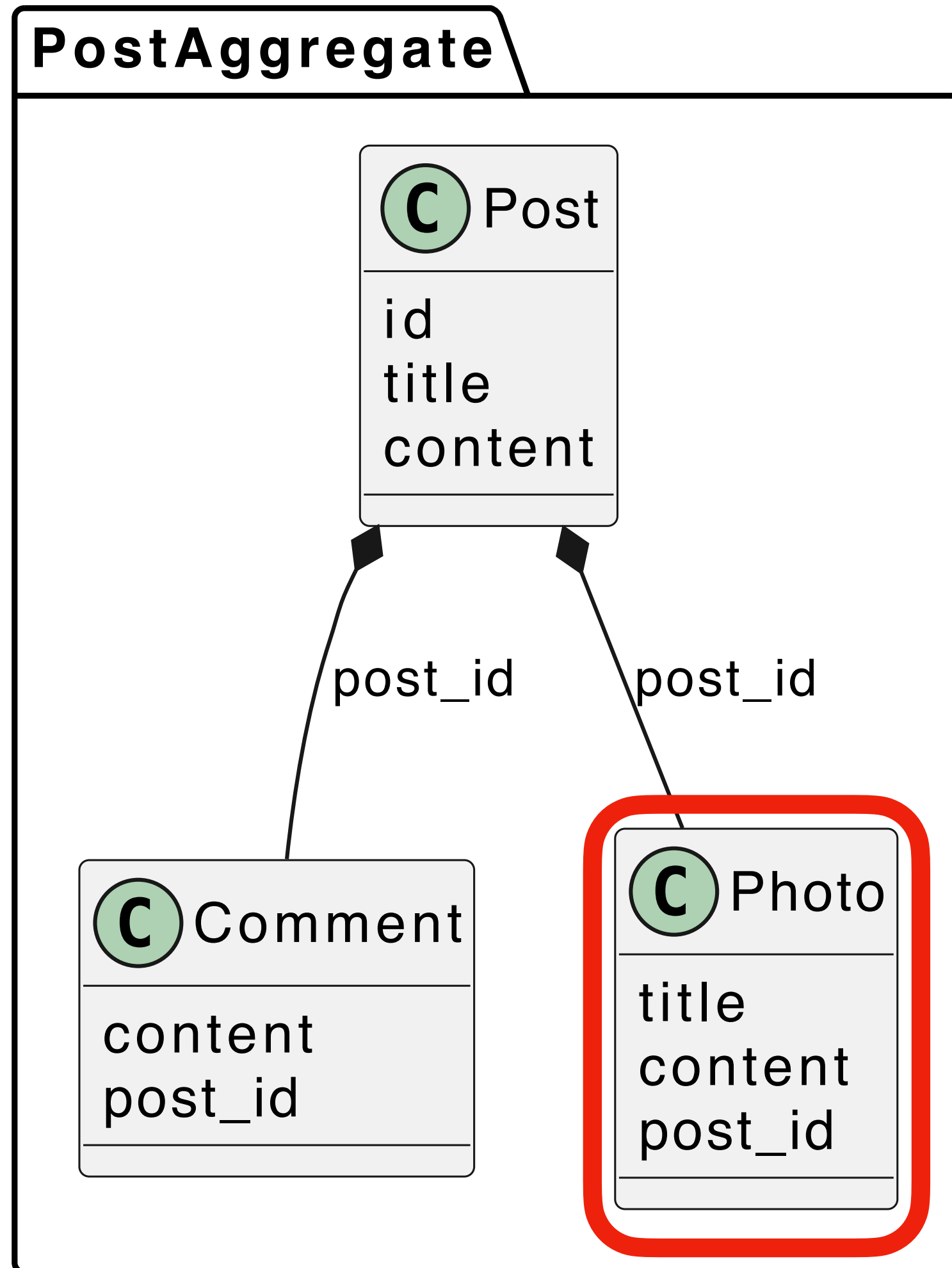
# Single Query Loading. Silver bullet?

## Пример



# Single Query Loading. Silver bullet?

## Пример



# Single Query Loading. Silver bullet?

## Пример



```
@Test
void should_find_all() {
    Post post = new Post("test-post-title", "test-post-content");
    postRepository.save(post);

    Post post2 = new Post("test-post-title-2", "test-post-content-2");
    postRepository.save(post2);

    postRepository.findAll();
}
```

# Single Query Loading. Silver bullet?

## Пример



```
SELECT "post"."id" AS "id", "post"."title" AS "title", "post"."content" AS "content" FROM "post"
```

```
SELECT "comment"."content" AS "content" FROM "comment" WHERE "comment"."post_id" = ?
```

```
SELECT "post_photo"."title" AS "title", "post_photo"."content" AS "content" FROM "post_photo" WHERE  
"post_photo"."post_id" = ?
```

```
SELECT "comment"."content" AS "content" FROM "comment" WHERE "comment"."post_id" = ?
```

```
SELECT "post_photo"."title" AS "title", "post_photo"."content" AS "content" FROM "post_photo" WHERE  
"post_photo"."post_id" = ?
```

# Single Query Loading. Silver bullet?

## Пример

```
SELECT "post"."id" AS "id", "post"."title" AS "title", "post"."content" AS "content" FROM "post"
```

```
SELECT "comment"."content" AS "content" FROM "comment" WHERE "comment"."post_id" = ?
```

```
SELECT "post_photo"."title" AS "title", "post_photo"."content" AS "content" FROM "post_photo" WHERE  
"post_photo"."post_id" = ?
```

```
SELECT "comment"."content" AS "content" FROM "comment" WHERE "comment"."post_id" = ?
```

```
SELECT "post_photo"."title" AS "title", "post_photo"."content" AS "content" FROM "post_photo" WHERE  
"post_photo"."post_id" = ?
```

# Single Query Loading. Silver bullet?

## Пример



```
SELECT "post"."id" AS "id", "post"."title" AS "title", "post"."content" AS "content" FROM "post"
```

```
SELECT "comment"."content" AS "content" FROM "comment" WHERE "comment"."post_id" = ?  
SELECT "post_photo"."title" AS "title", "post_photo"."content" AS "content" FROM "post_photo" WHERE  
"post_photo"."post_id" = ?
```

```
SELECT "comment"."content" AS "content" FROM "comment" WHERE "comment"."post_id" = ?  
SELECT "post_photo"."title" AS "title", "post_photo"."content" AS "content" FROM "post_photo" WHERE  
"post_photo"."post_id" = ?
```

# Single Query Loading. Silver bullet?

Как включить?



```
@Bean
```

```
JdbcMappingContext jdbcMappingContext() {  
    JdbcMappingContext jdbcMappingContext = new JdbcMappingContext();  
    jdbcMappingContext.setSingleQueryLoadingEnabled(true);  
  
    return jdbcMappingContext;  
}
```

# Single Query Loading. Silver bullet?

Ожидание



```
SELECT
  p.id
  p.title,
  p.content,
  c.content as comment_content,
  ph.title as photo_title,
  ph.content as photo_content
FROM post p
LEFT JOIN comment c ON p.id = c.post_id
LEFT JOIN post_photo ON p.id = ph.post_id
ORDER BY p.id
```

# Demo

# Single Query Loading. Silver bullet?

## Реальность

```
SELECT
case when rn_post_1 = rn THEN c_title_4 else null end as c_title_4,
case when rn_post_1 = rn THEN c_content_5 else null end as c_content_5,
case when rn_post_photo_7 = rn THEN c_title_11 else null end as c_title_11,
case when rn_post_photo_7 = rn THEN c_content_12 else null end as c_content_12,
key_post_photo_10,
case when rn_comment_14 = rn THEN c_content_18 else null end as c_content_18,
key_comment_17,
c_id_3
FROM
(
SELECT
c_title_4,
c_content_5,
rn_post_1,
c_id_3,
c_title_11,
c_content_12,
rn_post_photo_7,
br_post_photo_9,
key_post_photo_10,
c_content_18,
rn_comment_14,
br_comment_16,
key_comment_17,
GREATEST(
COALESCE(rn_post_1, 1),
COALESCE(rn_post_photo_7, 1),
COALESCE(rn_comment_14, 1)
) AS rn
) AS rn
FROM
(
SELECT
1 AS rn_post_1,
1 AS rc_post_2,
"post"."id" AS c_id_3,
"post"."title" AS c_title_4,
"post"."content" AS c_content_5
FROM
"post"
) t_post_6
LEFT OUTER JOIN (
SELECT
row_number() OVER(
PARTITION BY "post_photo"."post_id"
ORDER BY
"post_photo"."post_id"
) AS rn_post_photo_7,
count(*) OVER(
PARTITION BY "post_photo"."post_id"
) AS rc_post_photo_8,
"post_photo"."post_id" AS br_post_photo_9,
row_number() OVER(
PARTITION BY "post_photo"."post_id"
ORDER BY
"post_photo"."post_id"
) AS key_post_photo_10,
"post_photo"."title" AS c_title_11,
"post_photo"."content" AS c_content_12
FROM
"post_photo"
) t_post_photo_13 ON c_id_3 = br_post_photo_9
LEFT OUTER JOIN (
SELECT
row_number() OVER(
PARTITION BY "comment"."post_id"
ORDER BY
"comment"."post_id"
) AS rn_comment_14,
count(*) OVER(PARTITION BY "comment"."post_id") AS rc_comment_15,
"comment"."post_id" AS br_comment_16,
row_number() OVER(
PARTITION BY "comment"."post_id"
ORDER BY
"comment"."post_id"
) AS key_comment_17,
"comment"."content" AS c_content_18
FROM
"comment"
) t_comment_19 ON c_id_3 = br_comment_16
WHERE
(
rn_post_photo_7 = rn_comment_14
OR rn_post_photo_7 IS NULL
OR rn_comment_14 IS NULL
OR (
rn_post_photo_7 > rc_comment_15
AND rn_comment_14 = 1
)
)
OR (
rn_comment_14 > rc_post_photo_8
AND rn_post_photo_7 = 1
)
)
) main
ORDER BY
c_id_3,
rn
```

очень хорошо  
а теперь в палату



# Single Query Loading. Silver bullet?

Почему жизнь такая жестокая?



```
SELECT
  p.id
  p.title,
  p.content,
  c.content as comment_content,
  ph.title as photo_title,
  ph.content as photo_content
FROM post p
LEFT JOIN comment c ON p.id = c.post_id
LEFT JOIN post_photo ON p.id = ph.post_id
ORDER BY p.id
```

# Single Query Loading. Silver bullet?

Почему жизнь такая жестокая?

| id | title       | content               | comment_content             | photo_title           | photo_content      |
|----|-------------|-----------------------|-----------------------------|-----------------------|--------------------|
| 1  | Первый пост | Контент первого поста | Контент первого комментария | Название первого фото | Урл на первое фото |
| 1  | Первый пост | Контент первого поста | Контент второго комментария | Название первого фото | Урл на первое фото |
| 1  | Первый пост | Контент первого поста | Контент первого комментария | Название второго фото | Урл на второе фото |
| 1  | Первый пост | Контент первого поста | Контент второго комментария | Название второго фото | Урл на второе фото |

# Single Query Loading. Silver bullet?

Почему жизнь такая жестокая?

| id | title       | content               | comment_content             | photo_title           | photo_content      |
|----|-------------|-----------------------|-----------------------------|-----------------------|--------------------|
| 1  | Первый пост | Контент первого поста | Контент первого комментария | Название первого фото | Урл на первое фото |
| 1  | Первый пост | Контент первого поста | Контент второго комментария | Название первого фото | Урл на первое фото |
| 1  | Первый пост | Контент первого поста | Контент первого комментария | Название второго фото | Урл на второе фото |
| 1  | Первый пост | Контент первого поста | Контент второго комментария | Название второго фото | Урл на второе фото |

# Single Query Loading. Silver bullet?

Почему жизнь такая жестокая?



```
SELECT
  1 AS rn_post_1,
  1 AS rc_post_2,
  "post"."id" AS c_id_3,
  "post"."title" AS c_title_4,
  "post"."content" AS c_content_5
FROM
  "post"
```

# Single Query Loading. Silver bullet?

Почему жизнь такая жестокая?

| rn_post | rc_post | id | title       | content               |
|---------|---------|----|-------------|-----------------------|
| 1       | 1       | 1  | Первый пост | Контент первого поста |
| 1       | 1       | 2  | Второй пост | Контент второго поста |

# Single Query Loading. Silver bullet?

## Почему жизнь такая жестокая?

```
LEFT OUTER JOIN (  
  SELECT  
    row_number() OVER(  
      PARTITION BY "post_photo"."post_id"  
      ORDER BY  
        "post_photo"."post_id"  
    ) AS rn_post_photo_7,  
    count(*) OVER(  
      PARTITION BY "post_photo"."post_id"  
    ) AS rc_post_photo_8,  
    "post_photo"."post_id" AS br_post_photo_9,  
    row_number() OVER(  
      PARTITION BY "post_photo"."post_id"  
      ORDER BY  
        "post_photo"."post_id"  
    ) AS key_post_photo_10,  
    "post_photo"."title" AS c_title_11,  
    "post_photo"."content" AS c_content_12  
  FROM  
    "post_photo"  
) t_post_photo_13 ON c_id_3 = br_post_photo_9  
LEFT OUTER JOIN (  
  SELECT  
    row_number() OVER(  
      PARTITION BY "comment"."post_id"  
      ORDER BY  
        "comment"."post_id"  
    ) AS rn_comment_14,  
    count(*) OVER(PARTITION BY "comment"."post_id") AS rc_comment_15,  
    "comment"."post_id" AS br_comment_16,  
    row_number() OVER(  
      PARTITION BY "comment"."post_id"  
      ORDER BY  
        "comment"."post_id"  
    ) AS key_comment_17,  
    "comment"."content" AS c_content_18  
  FROM  
    "comment"  
) t_comment_19 ON c_id_3 = br_comment_16
```

# Single Query Loading. Silver bullet?

Почему жизнь такая жестокая?

| rn_post | rc_post | id | title       | content               | rn_comment | rc_comment | comment_content         | rn_photo | rc_photo | photo_title      | photo_content      |
|---------|---------|----|-------------|-----------------------|------------|------------|-------------------------|----------|----------|------------------|--------------------|
| 1       | 1       | 1  | Первый пост | Контент первого поста | 1          | 2          | Контент первого коммент | 1        | 2        | Название первого | Урл на первое фото |
| 1       | 1       | 1  | Первый пост | Контент первого поста | 2          | 2          | Контент второго коммент | 1        | 2        | Название первого | Урл на первое фото |
| 1       | 1       | 1  | Первый пост | Контент первого поста | 1          | 2          | Контент первого коммент | 2        | 2        | Название второго | Урл на второе фото |
| 1       | 1       | 1  | Первый пост | Контент первого поста | 2          | 2          | Контент второго коммент | 2        | 2        | Название второго | Урл на второе фото |

# Single Query Loading. Silver bullet?

Почему жизнь такая жестокая?



```
WHERE  
  (  
    rn_post_photo_7 = rn_comment_14  
  OR rn_post_photo_7 IS NULL  
  OR rn_comment_14 IS NULL  
  OR (  
    rn_post_photo_7 > rc_comment_15  
    AND rn_comment_14 = 1  
  )  
  OR (  
    rn_comment_14 > rc_post_photo_8  
    AND rn_post_photo_7 = 1  
  )  
  )  
)
```

# Single Query Loading. Silver bullet?

Почему жизнь такая жестокая?



```
WHERE
```

```
(
```

```
rn_post_photo_7 = rn_comment_14
```

```
OR rn_post_photo_7 IS NULL
```

```
OR rn_comment_14 IS NULL
```

```
OR (
```

```
rn_post_photo_7 > rc_comment_15
```

```
AND rn_comment_14 = 1
```

```
)
```

```
OR (
```

```
rn_comment_14 > rc_post_photo_8
```

```
AND rn_post_photo_7 = 1
```

```
)
```

```
)
```

# Single Query Loading. Silver bullet?

Почему жизнь такая жестокая?



WHERE

(

rn\_post\_photo\_7 = rn\_comment\_14

OR rn\_post\_photo\_7 IS NULL

OR rn\_comment\_14 IS NULL

OR (

rn\_post\_photo\_7 > rc\_comment\_15

AND rn\_comment\_14 = 1

)

OR (

rn\_comment\_14 > rc\_post\_photo\_8

AND rn\_post\_photo\_7 = 1

)

)

# Single Query Loading. Silver bullet?

Почему жизнь такая жестокая?



WHERE

(

rn\_post\_photo\_7 = rn\_comment\_14

OR rn\_post\_photo\_7 IS NULL

OR rn\_comment\_14 IS NULL

OR (

rn\_post\_photo\_7 > rc\_comment\_15

AND rn\_comment\_14 = 1

)

OR (

rn\_comment\_14 > rc\_post\_photo\_8

AND rn\_post\_photo\_7 = 1

)

)

# Single Query Loading. Silver bullet?

Почему жизнь такая жестокая?

| rn_post | rc_post | id | title       | content               | rn_comment | rc_comment | comment_content             | rn_photo | rc_photo | photo_title           | photo_content      |
|---------|---------|----|-------------|-----------------------|------------|------------|-----------------------------|----------|----------|-----------------------|--------------------|
| 1       | 1       | 1  | Первый пост | Контент первого поста | 1          | 2          | Контент первого комментария | 1        | 2        | Название первого фото | Урл на первое фото |
| 1       | 1       | 1  | Первый пост | Контент первого поста | 2          | 2          | Контент второго комментария | 2        | 2        | Название второго фото | Урл на второе фото |

# Single Query Loading. Silver bullet?

Почему жизнь такая жестокая?

| rn_post | rc_post | id | title       | content               | rn_comment | rc_comment | comment_content             | rn_photo | rc_photo | photo_title            | photo_content      |
|---------|---------|----|-------------|-----------------------|------------|------------|-----------------------------|----------|----------|------------------------|--------------------|
| 1       | 1       | 2  | Второй пост | Контент второго поста | 1          | 2          | Контент первого комментария | 1        | 3        | Название первого фото  | Урл на первое фото |
| 1       | 1       | 2  | Второй пост | Контент второго поста | 2          | 2          | Контент второго комментария | 2        | 3        | Название второго фото  | Урл на второе фото |
| 1       | 1       | 2  | Второй пост | Контент второго поста | 1          | 2          | Контент первого комментария | 3        | 3        | Название третьего фото | Урл на третье фото |

# Single Query Loading. Silver bullet?

Почему жизнь такая жестокая?



SELECT

```
c_title_4,  
c_content_5,  
rn_post_1,  
c_id_3,  
c_title_11,  
c_content_12,  
rn_post_photo_7,  
br_post_photo_9,  
key_post_photo_10,  
c_content_18,  
rn_comment_14,  
br_comment_16,  
key_comment_17
```

```
GREATEST(  
  COALESCE(rn_post_1, 1),  
  COALESCE(rn_post_photo_7, 1),  
  COALESCE(rn_comment_14, 1)  
) AS rn
```

# Single Query Loading. Silver bullet?

Почему жизнь такая жестокая?

| rn | rn_post | rc_post | id | title       | content               | rn_comment | rc_comment | comment_content             | rn_photo | rc_photo | photo_title           | photo_content      |
|----|---------|---------|----|-------------|-----------------------|------------|------------|-----------------------------|----------|----------|-----------------------|--------------------|
| 1  | 1       | 1       | 1  | Первый пост | Контент первого поста | 1          | 2          | Контент первого комментария | 1        | 2        | Название первого фото | Урл на первое фото |
| 2  | 1       | 1       | 1  | Первый пост | Контент первого поста | 2          | 2          | Контент второго комментария | 2        | 2        | Название второго фото | Урл на второе фото |

# Single Query Loading. Silver bullet?

Почему жизнь такая жестокая?

| rn | rn_post | rc_post | id | title       | content               | rn_comment | rc_comment | comment_content             | rn_photo | rc_photo | photo_title            | photo_content      |
|----|---------|---------|----|-------------|-----------------------|------------|------------|-----------------------------|----------|----------|------------------------|--------------------|
| 1  | 1       | 1       | 2  | Второй пост | Контент второго поста | 1          | 2          | Контент первого комментария | 1        | 3        | Название первого фото  | Урл на первое фото |
| 2  | 1       | 1       | 2  | Второй пост | Контент второго поста | 2          | 2          | Контент второго комментария | 2        | 3        | Название второго фото  | Урл на второе фото |
| 3  | 1       | 1       | 2  | Второй пост | Контент второго поста | 1          | 2          | Контент первого комментария | 3        | 3        | Название третьего фото | Урл на третье фото |

# Single Query Loading. Silver bullet?

Почему жизнь такая жестокая?



SELECT

```
case when rn_post_1 = rn THEN c_title_4 else null end as c_title_4,  
case when rn_post_1 = rn THEN c_content_5 else null end as c_content_5,  
case when rn_post_photo_7 = rn THEN c_title_11 else null end as c_title_11,  
case when rn_post_photo_7 = rn THEN c_content_12 else null end as c_content_12,  
key_post_photo_10,  
case when rn_comment_14 = rn THEN c_content_18 else null end as c_content_18,  
key_comment_17,  
c_id_3
```

# Single Query Loading. Silver bullet?

Почему жизнь такая жестокая?

| id | title       | content               | comment_content             | photo_title           | photo_content      |
|----|-------------|-----------------------|-----------------------------|-----------------------|--------------------|
| 1  | Первый пост | Контент первого поста | Контент первого комментария | Название первого фото | Урл на первое фото |
| 1  |             |                       | Контент второго комментария | Название второго фото | Урл на второе фото |

# Single Query Loading. Silver bullet?

Почему жизнь такая жестокая?

| id | title       | content               | comment_content             | photo_title            | photo_content      |
|----|-------------|-----------------------|-----------------------------|------------------------|--------------------|
| 2  | Второй пост | Контент второго поста | Контент первого комментария | Название первого фото  | Урл на первое фото |
| 2  |             |                       | Контент второго комментария | Название второго фото  | Урл на второе фото |
| 2  |             |                       |                             | Название третьего фото | Урл на третье фото |

# Single Query Loading. Silver bullet?

Почему жизнь такая жестокая?

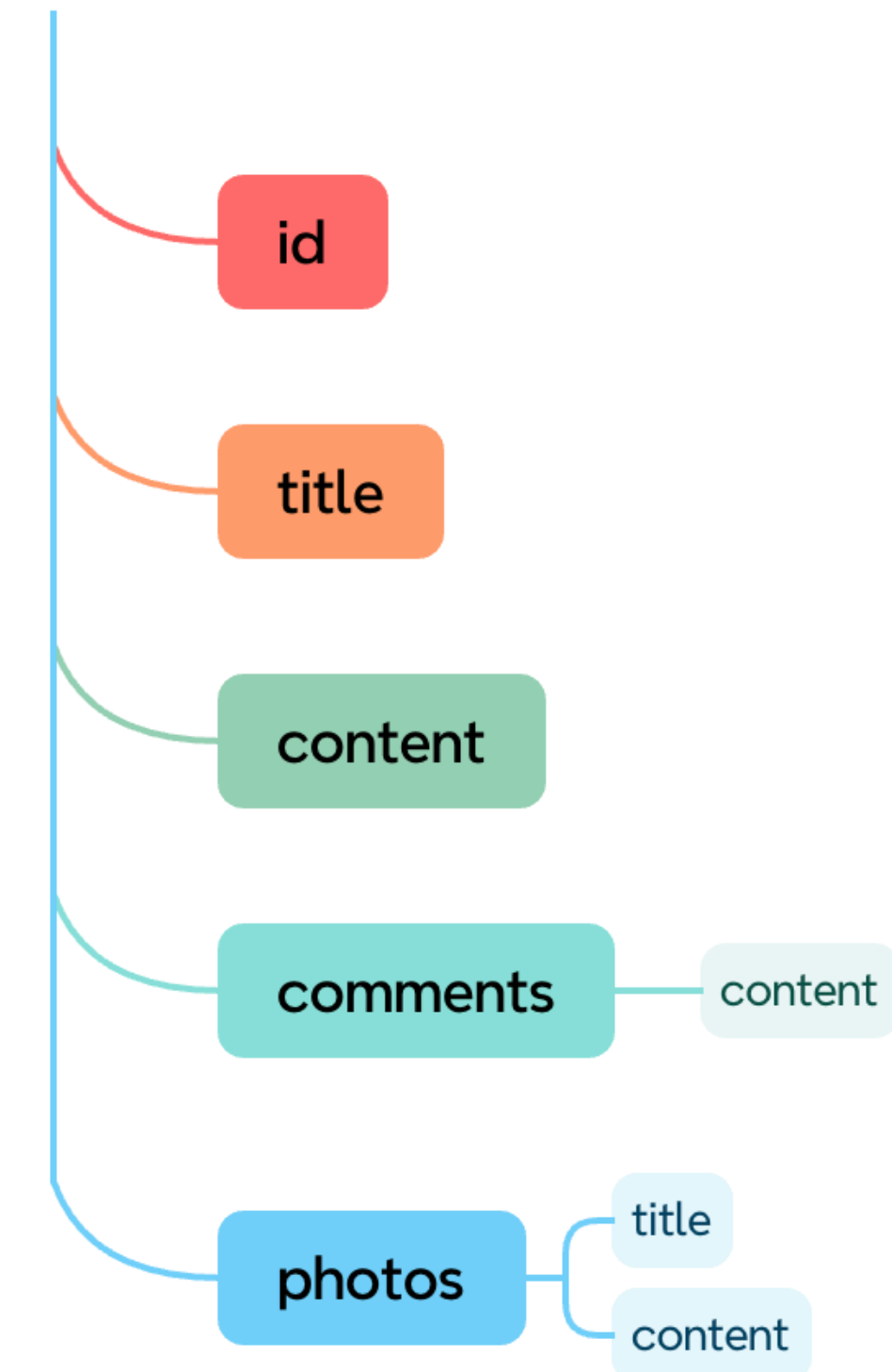
|                        |                             |                             |                        |
|------------------------|-----------------------------|-----------------------------|------------------------|
| <b>id</b>              | 2                           | 2                           | 2                      |
| <b>title</b>           | Второй пост                 |                             |                        |
| <b>content</b>         | Контент второго поста       |                             |                        |
| <b>comment_content</b> | Контент первого комментария | Контент второго комментария |                        |
| <b>photo_title</b>     | Название первого фото       | Название второго фото       | Название третьего фото |
| <b>photo_content</b>   | Урл на первое фото          | Урл на второе фото          | Урл на третье фото     |

# Single Query Loading. Silver bullet?

Почему жизнь такая жестокая?

|                        |                             |                             |                        |
|------------------------|-----------------------------|-----------------------------|------------------------|
| <b>id</b>              | 2                           | 2                           | 2                      |
| <b>title</b>           | Второй пост                 |                             |                        |
| <b>content</b>         | Контент второго поста       |                             |                        |
| <b>comment_content</b> | Контент первого комментария | Контент второго комментария |                        |
| <b>photo_title</b>     | Название первого фото       | Название второго фото       | Название третьего фото |
| <b>photo_content</b>   | Урл на первое фото          | Урл на второе фото          | Урл на третье фото     |

**Post**



# Single Query Loading. Silver bullet?

Почему жизнь такая жестокая?

- Можно сделать не in-memory оффсет пагинацию (в отличии от Hibernate)

# Single Query Loading. Silver bullet?

## Почему жизнь такая жестокая?

- Можно сделать не in-memory оффсет пагинацию (в отличии от Hibernate)



```
SELECT
  1 AS rn_post_1,
  1 AS rc_post_2,
  "post"."id" AS c_id_3,
  "post"."title" AS c_title_4,
  "post"."content" AS c_content_5
FROM
  "post"
```

# Single Query Loading. Silver bullet?

## Почему жизнь такая жестокая?

- Можно сделать не in-memory оффсет пагинацию (в отличии от Hibernate)

```
SELECT
  case when rn_post_1 = rn THEN c_title_4 else null end as c_title_4,
  case when rn_post_1 = rn THEN c_content_5 else null end as c_content_5,
  case when rn_post_photo_7 = rn THEN c_title_11 else null end as c_title_11,
  case when rn_post_photo_7 = rn THEN c_content_12 else null end as c_content_12,
  key_post_photo_18,
  case when rn_comment_14 = rn THEN c_content_18 else null end as c_content_18,
  key_comment_17,
  c_id_3
FROM
  (
    SELECT
      c_title_4,
      c_content_5,
      rn_post_1,
      c_id_3,
      c_title_11,
      c_content_12,
      rn_post_photo_7,
      br_post_photo_9,
      key_post_photo_18,
      c_content_18,
      rn_comment_14,
      br_comment_16,
      key_comment_17,
      GREATEST(
        COALESCE(rn_post_1, 1),
        COALESCE(rn_post_photo_7, 1),
        COALESCE(rn_comment_14, 1)
      ) AS rn
    FROM
      (
        SELECT
          1 AS rn_post_1,
          1 AS rc_post_2,
          "post"."id" AS c_id_3,
          "post"."title" AS c_title_4,
          "post"."content" AS c_content_5
        FROM
          "post"
      ) AS t
    LEFT OUTER JOIN t
    LEFT OUTER JOIN t
    SELECT
      row_number() OVER(
        PARTITION BY "post_photo"."post_id"
        ORDER BY
          "post_photo"."post_id"
        ) AS rn_post_photo_7,
      count(*) OVER(
        PARTITION BY "post_photo"."post_id"
        ) AS rc_post_photo_9,
      row_number() OVER(
        PARTITION BY "post_photo"."post_id"
        ORDER BY
          "post_photo"."post_id"
        ) AS key_post_photo_18,
      "post_photo"."title" AS c_title_11,
      "post_photo"."content" AS c_content_12
    FROM
      "post_photo"
    ) t_post_photo_13 ON c_id_3 = br_post_photo_9
    LEFT OUTER JOIN t
    SELECT
      row_number() OVER(
        PARTITION BY "comment"."post_id"
        ORDER BY
          "comment"."post_id"
        ) AS rn_comment_14,
      count(*) OVER(PARTITION BY "comment"."post_id" AS rc_comment_15,
        "comment"."post_id" AS br_comment_16,
        row_number() OVER(
        PARTITION BY "comment"."post_id"
        ORDER BY
          "comment"."post_id"
        ) AS key_comment_17,
      "comment"."content" AS c_content_18
    FROM
      "comment"
    ) t_comment_19 ON c_id_3 = br_comment_16
  ) WHERE
  (
    rn_post_photo_7 = rn_comment_14
    OR rn_post_photo_7 IS NULL
    OR rn_comment_14 IS NULL
  )
  OR (
    rn_post_photo_7 > rc_comment_15
    AND rn_comment_14 = 1
  )
  OR (
    rn_comment_14 > rc_post_photo_8
    AND rn_post_photo_7 = 1
  )
) main
ORDER BY
  c_id_3,
  rn
```



SELECT

```
1 AS rn_post_1,
1 AS rc_post_2,
"post"."id" AS c_id_3,
"post"."title" AS c_title_4,
"post"."content" AS c_content_5
```

FROM

```
"post"
```

# Single Query Loading. Silver bullet?

## Почему жизнь такая жестокая?

- Можно сделать не in-memory оффсет пагинацию (в отличии от Hibernate)

```
SELECT
  case when rn_post_1 = rn THEN c_title_4 else null end as c_title_4,
  case when rn_post_1 = rn THEN c_content_5 else null end as c_content_5,
  case when rn_post_photo_7 = rn THEN c_title_11 else null end as c_title_11,
  case when rn_post_photo_7 = rn THEN c_content_12 else null end as c_content_12,
  key_post_photo_18,
  case when rn_comment_14 = rn THEN c_content_18 else null end as c_content_18,
  key_comment_17,
  c_id_3
FROM
  (
    SELECT
      c_title_4,
      c_content_5,
      rn_post_1,
      c_id_3,
      c_title_11,
      c_content_12,
      rn_post_photo_7,
      br_post_photo_9,
      key_post_photo_18,
      c_content_18,
      rn_comment_14,
      br_comment_16,
      key_comment_17,
      GREATEST(
        COALESCE(rn_post_1, 1),
        COALESCE(rn_post_photo_7, 1),
        COALESCE(rn_comment_14, 1)
      ) AS rn
    FROM
      (
        SELECT
          1 AS rn_post_1,
          1 AS rc_post_2,
          "post"."id" AS c_id_3,
          "post"."title" AS c_title_4,
          "post"."content" AS c_content_5
        FROM
          "post"
      ) AS t
    LEFT OUTER JOIN t
    ON t.rc_post_2 = br_post_photo_9
    LEFT OUTER JOIN t
    ON t.c_id_3 = br_post_photo_9
    LEFT OUTER JOIN t
    ON t.rc_post_2 = rc_comment_15
    LEFT OUTER JOIN t
    ON t.c_id_3 = br_comment_16
  ) AS rn
ORDER BY
  c_id_3,
  rn
```

```
SELECT
  1 AS rn_post_1,
  1 AS rc_post_2,
  "post"."id" AS c_id_3,
  "post"."title" AS c_title_4,
  "post"."content" AS c_content_5
FROM
  "post"
LIMIT 10
```

# Single Query Loading. Silver bullet?

Почему жизнь такая жестокая?

- Можно сделать не in-memory оффсет пагинацию (в отличии от Hibernate)
- Решает проблему декартова произведения (в отличии от Hibernate)

# Single Query Loading. Silver bullet?

## Почему жизнь такая жестокая?

- Можно сделать не in-memory оффсет пагинацию (в отличии от Hibernate)
- Решает проблему декартова произведения (в отличии от Hibernate)
- ORM точно знает где кончается один пост и начинается другой

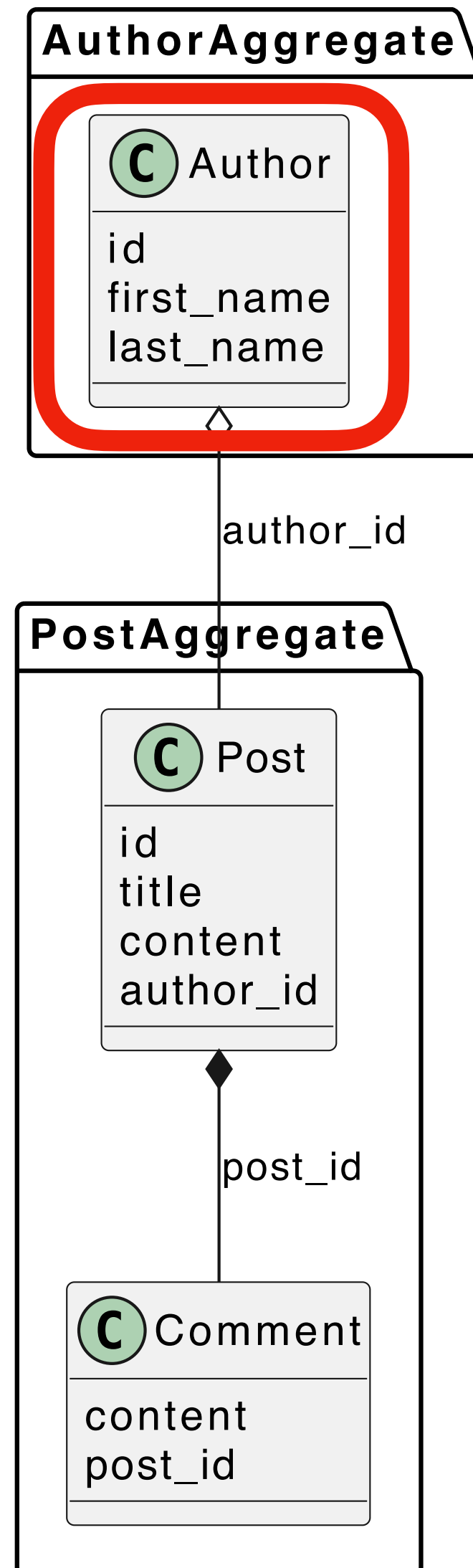
# Single Query Loading. Silver bullet?

## Почему жизнь такая жестокая?

- Можно сделать не in-memory оффсет пагинацию (в отличии от Hibernate)
- Решает проблему декартова произведения (в отличии от Hibernate)
- ORM точно знает где кончается один пост и начинается другой
- **И все это без потерь по перформансу**

# Single Query Loading. Silver bullet?

Что сделать нельзя?



# Demo

# Single Query Loading. Silver bullet?

Что сделать нельзя?

- Родительский агрегат должен не использовать `AggregateReference` или `Embedded entity`

# Single Query Loading. Silver bullet?

## Что сделать нельзя?

- Родительский агрегат должен не использовать AggregateReference или Embedded entity
- Диалект базы данных должен поддерживать оконные функции

# Single Query Loading. Silver bullet?

## Что сделать нельзя?

- Родительский агрегат должен не использовать `AggregateReference` или `Embedded entity`
- Диалект базы данных должен поддерживать оконные функции
- Работает только для `find` методов в `CrudRepository`. Не поддерживает `derived queries` и `annotated queries`

# Type-Safe Property Paths

# Type-Safe Property Paths

В чем проблема `string-based property references`?

- Нету валидации при компиляции

# Type-Safe Property Paths

В чем проблема `string-based property references`?

- Нету валидации при компиляции
- Убивается тулинг от IDE

# Type-Safe Property Paths

В чем проблема 'Metamodel-based' решений?



```
@Generated(value = "org.hibernate.jpamodelgen.JPAMetaModelEntityProcessor")
@StaticMetamodel(Student.class)
public abstract class Student_ {

    public static volatile SingularAttribute<Student, String> firstName;
    public static volatile SingularAttribute<Student, String> lastName;
    public static volatile SingularAttribute<Student, Integer> id;
    public static volatile SingularAttribute<Student, Integer> gradYear;

    public static final String FIRST_NAME = "firstName";
    public static final String LAST_NAME = "lastName";
    public static final String ID = "id";
    public static final String GRAD_YEAR = "gradYear";
}
```

# Type-Safe Property Paths

В чем проблема 'Metamodel-based' решений?

- Генерируются в билд-тайме

# Type-Safe Property Paths

В чем проблема 'Metamodel-based' решений?

- Генерируются в билд-тайме
- Метамоделли работают только в ограниченном контексте

# Type-Safe Property Paths

В чем проблема 'Metamodel-based' решений?

- Генерируются в билд-тайме
- Метамоделли работают только в ограниченном контексте
- Не позволяют сделать type-safe nested path

# Type-Safe Property Paths

В чем проблема 'Metamodel-based' решений?

- Генерируются в билд-тайме
- Метамоделли работают только в ограниченном контексте
- Не позволяют сделать type-safe nested path
- В зависимости от IDE работа с generated sources может отличаться по dev experience

# Type-Safe Property Paths

The screenshot shows the Spring.io website. At the top, there is a navigation bar with the Spring logo and menu items: Why Spring, Learn, Projects, Resources, Community, and Enterprise. Below the navigation bar is the "Spring Blog" section with an RSS feeds link. The main content area features a blog post titled "Spring Data 2026.0.0-M1 released" by Mark Paluch, dated February 13, 2026. The post text mentions the announcement of the 2026.0.0 release train and highlights "Support for Type-Safe Property Paths". It explains that this initiative aims to reduce the need for stringly-typed programming when referring to properties within an entity. A code block provides Java examples for using PropertyPath and PropertyReference. A "SUBSCRIBE" button is visible in a sidebar on the right. At the bottom left, there is a "Kotlin variants" section with a "Display a menu" button.

spring by VMware Tanzu

Why Spring ▾ Learn ▾ Projects ▾ Resources ▾ Community ▾ Enterprise ▾

Spring Blog [RSS feeds ▾](#)

## Spring Data 2026.0.0-M1 released

RELEASES | MARK PALUCH | FEBRUARY 13, 2026 | 1 MIN READ | 0 COMMENTS

On behalf of the team and everyone who has contributed, I'm delighted to announce the first milestone of the `2026.0.0` release train.

### Support for Type-Safe Property Paths

We now support type-safe property paths and property references as initiative to reduce the need for stringly-typed programming when referring to properties within an entity.

Java variants:

```
PropertyPath.from("name", Person.class) // existing String-based API
PropertyPath.of(Person::getName) // type-safe property reference expression

PropertyPath.from("address.country", Person.class) // existing nested path API
PropertyPath.of(Person::getAddress).then(Address::getCountry) // type-safe composed path expre

PropertyReference.of(Secret::getSecret)
```

[COPY](#)

Kotlin variants: [Display a menu](#)

**Get the Spring newsletter**

Stay connected with the Spring newsletter

[SUBSCRIBE](#)

# Demo

# Scrolling API

# Scrolling API. Что это вообще такое?

## Offset



```
Window<User> users = repository.findFirst10ByLastnameOrderByFirstname("Doe", ScrollPosition.offset());
do {

    for (User u : users) {
        // consume the user
    }

    if (users.isLast() || users.isEmpty()) {
        break;
    }

    // obtain the next Scroll
    users = repository.findFirst10ByLastnameOrderByFirstname("Doe", users.positionAt(users.size() - 1));
} while (!users.isEmpty());
```

# Scrolling API. Что это вообще такое?

## Keyset



```
interface UserRepository extends Repository<User, Long> {  
    Window<User> findFirst10ByLastnameOrderByFirstname(String lastname, KeysetScrollPosition position);  
}  
  
WindowIterator<User> users = WindowIterator.of(position ->  
    repository.findFirst10ByLastnameOrderByFirstname("Doe", position))  
    .startingAt(ScrollPosition.keySet());
```

# Scrolling API. Текущее состояние

## Реализация

The screenshot shows a GitHub pull request page for the repository 'spring-projects / spring-data-relational'. The pull request is titled 'FEATURE: Window Scrolling API (Offset/Keyset pagination) [Spring Data JDBC] #2149'. It is a pull request from 'chrshnv' to 'spring-projects:main', containing 8 commits. The pull request is currently in progress, with 4 conversations, 8 commits, 1 check, and 8 files changed. The pull request has a net change of +716 lines and -214 lines. The pull request is labeled 'type: enhancement'. The pull request is currently in progress, with 4 conversations, 8 commits, 1 check, and 8 files changed. The pull request has a net change of +716 lines and -214 lines. The pull request is labeled 'type: enhancement'. The pull request is currently in progress, with 4 conversations, 8 commits, 1 check, and 8 files changed. The pull request has a net change of +716 lines and -214 lines. The pull request is labeled 'type: enhancement'.

**FEATURE: Window Scrolling API (Offset/Keyset pagination) [Spring Data JDBC] #2149**

chrshnv wants to merge 8 commits into spring-projects:main from chrshnv:main

Conversation 4 | Commits 8 | Checks 1 | Files changed 8 | +716 -214

chrshnv commented on Oct 6, 2025 · edited

This PR introduces an implementation of Window Scrolling API for Spring Data JDBC, providing efficient, index-friendly pagination over ordered datasets. Unlike traditional offset pagination, the new API uses keyset (window) navigation, which enables constant-time performance even on large tables.

**Example query:**

```
WHERE ("DUMMY_ENTITY"."ID_PROP" >= :id_prop AND ("DUMMY_ENTITY"."ID_PROP" > :id_prop1 OR ("DUMMY_ENTITY"."ID_PROP" < :id_prop2 AND ("DUMMY_ENTITY"."ID_PROP" < :id_prop3)))
```

Reviewers: No reviews. Still in progress? [Convert to draft](#)

Assignees: No one assigned

Labels: type: enhancement

Projects: None yet

Milestone: No milestone

spring-projects-issues added the status: waiting-for-triage label on Oct 6, 2025

Display a menu for "https://github.com/spring-projects/spring-data-relational/pull/2149/changes"

# Scrolling API. Текущее состояние

## Ограничения и особенности

- Поля входящие в ScrollPosition должны быть явно отсортированы

# Scrolling API. Текущее состояние

## Ограничения и особенности

- Поля входящие в ScrollPosition должны быть явно отсортированы
- Если какое-либо из полей - null, кидается exception

# Scrolling API. Текущее состояние

## Ограничения и особенности

- Поля входящие в ScrollPosition должны быть явно отсортированы
- Если какое-либо из полей - null, кидается exception
- **Может быть разная сортировка на разных полях**

# Scrolling API. Текущее состояние

Какой квери на выходе?



```
WHERE (  
    "DUMMY_ENTITY"."ID_PROP" >= :id_prop  
    AND (  
        "DUMMY_ENTITY"."ID_PROP" > :id_prop1  
        OR (  
            "DUMMY_ENTITY"."POINT_IN_TIME" > :point_in_time  
        )  
    )  
)  
) ORDER BY "DUMMY_ENTITY"."ID_PROP" ASC, "DUMMY_ENTITY"."POINT_IN_TIME" ASC LIMIT 50
```

# Scrolling API. Текущее состояние

Какой квери на выходе?



```
WHERE ("DUMMY_ENTITY"."ID_PROP", "DUMMY_ENTITY"."POINT_IN_TIME") > (:id_prop, :point_in_time)
```

# Scrolling API. Текущее состояние

Какой квери на выходе в Spring Data Jpa?



```
WHERE
```

```
    de1_0.id>?
```

```
    OR de1_0.id=?
```

```
    AND de1_0.point_in_time>?
```

```
ORDER BY
```

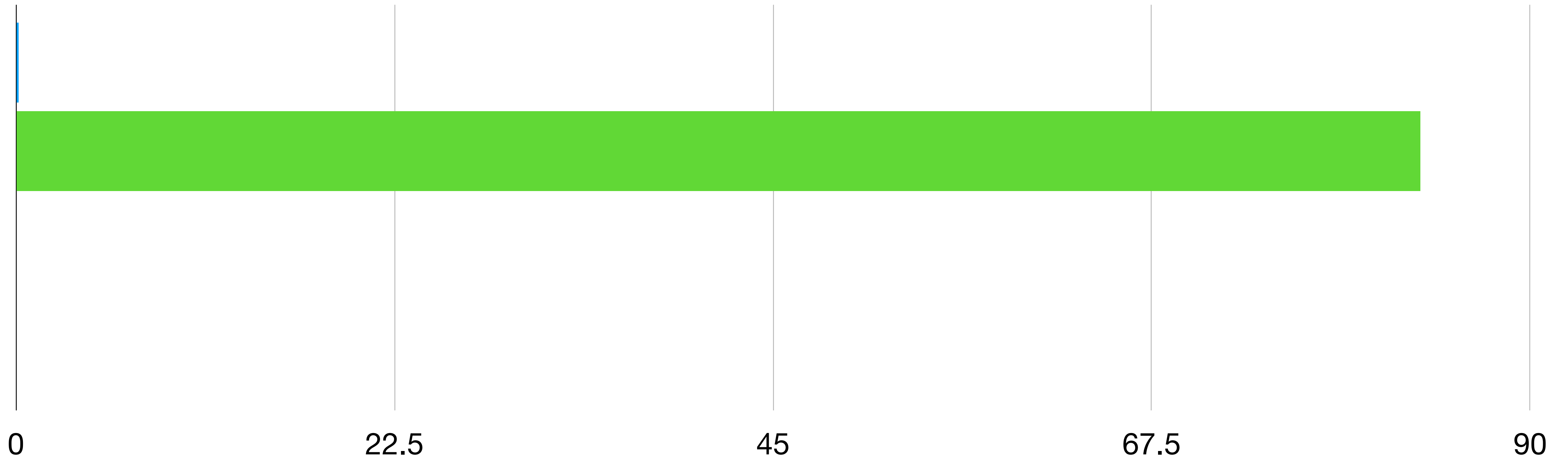
```
    de1_0.id
```

```
    de1_0.point_in_time
```

```
LIMIT 50
```

# Spring Data Jdbc > Spring Data Jpa?

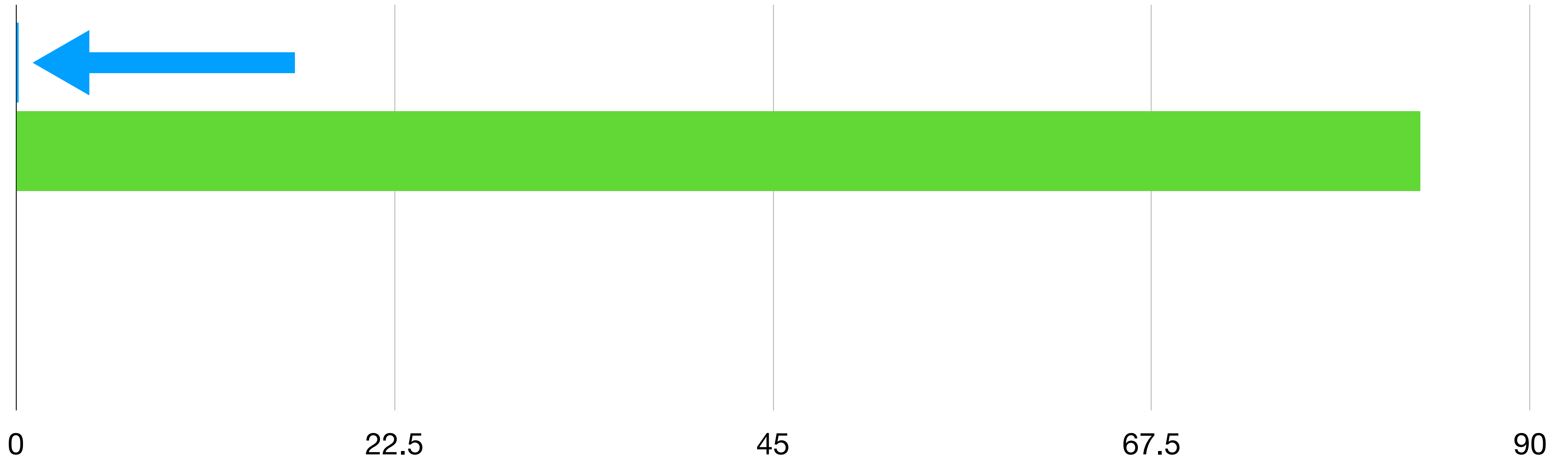
Actual time(мс) на запрос (меньше - лучше).



Зелёное - Spring Data Jpa, синее - Spring Data Jdbc

# Spring Data Jdbc > Spring Data Jpa?

Actual time(мс) на запрос (меньше - лучше).



Зелёное - Spring Data Jpa, синее - Spring Data Jdbc

# Spring Data Jdbc > Spring Data Jpa?

Actual time(мс) на запрос (меньше - лучше).

actual time=0.153..0.179 rows=50.00 loops=1

actual time=83.457..83.465 rows=50.00 loops=1

# Scrolling API. Spring Data Jdbc > Spring Data Jpa?

The screenshot shows a YouTube video player. The video title is "Илья Сазонов и Фёдор Сазонов — Offset и keyset: почём пагинация для продакшена?". The video content displays two SQL queries in a Notepad window, comparing their performance. The first query is labeled "медленно" (slow) and uses a keyset pagination approach. The second query is labeled "быстро" (fast) and uses an offset pagination approach. The video also shows a speaker at a podium with the JPoint logo.

```
select b
from Book b
where
    b.createdAt > :createdAt
or (b.createdAt = :createdAt and b.id > :id)
order by
    b.rating, b.id
```

медленно

```
select b
from Book b
where
    b.createdAt >= :createdAt
and (b.createdAt > :createdAt or b.id > :id)
order by
    b.createdAt, b.id
```

быстро

Э  
К  
В  
И  
В  
А  
Л  
Е  
Н  
Т  
Н  
Ы

Илья и Фёдор Сазоновы

19:44 / 45:35

СБЕР Яндекс МИР Plat.Form

Илья Сазонов и Фёдор Сазонов — Offset и keyset: почём пагинация для продакшена?

JPoint, Joker и JUG ru — Java-конференции 57.5K subscribers

334 Share Save

Владимир Ситников — В-Tree

# Scrolling API. Spring Data Jdbc > Spring Data Jpa?



<https://www.youtube.com/watch?v=wi6h9ox1wwM>

# Scrolling API. Pull request



<https://github.com/spring-projects/spring-data-relational/pull/2149>

# Выводы

- Используйте AOT репозитории

# Выводы

- Используйте AOT репозитории
- Композитные ключи все еще нуждаются в фиксах

# Выводы

- Используйте AOT репозитории
- Композитные ключи все еще нуждаются в фиксах
- Single Query Loading в Spring Data Jdbc имеет уникальную реализацию и достаточно интересно фиксирует проблемы которые не фиксирует Hibernate

# Выводы

- Используйте AOT репозитории
- Композитные ключи все еще нуждаются в фиксах
- Single Query Loading в Spring Data Jdbc имеет уникальную реализацию и достаточно интересно фиксирует проблемы которые не фиксирует Hibernate
- N+1 существует не только в Hibernate))0)))

# Выводы

- Используйте AOT репозитории
- Композитные ключи все еще нуждаются в фиксах
- Single Query Loading в Spring Data Jdbc имеет уникальную реализацию и достаточно интересно фиксирует проблемы которые не фиксирует Hibernate
- N+1 существует не только в Hibernate))0)))
- Type-Safe Property Paths решают проблемы dev experience которые есть у аналогичных решений

# Выводы

- Используйте AOT репозитории
- Композитные ключи все еще нуждаются в фиксах
- Single Query Loading в Spring Data Jdbc имеет уникальную реализацию и достаточно интересно фиксирует проблемы которые не фиксирует Hibernate
- N+1 существует не только в Hibernate))0)))
- Type-Safe Property Paths решают проблемы dev experience которые есть у аналогичных решений
- Реализация Scrolling API для Spring Data Jdbc на данный момент намного лучше по перформансу чем для Spring Data Jpa

# Scrolling API. Pull request



<https://github.com/spring-projects/spring-data-relational/pull/2149>