JPA EntityGraphs: una solución al problema de consultas N + 1

Resolución del problema de consultas N + 1: ejemplo en Spring Boot

Thameena S Seguir
Mar 11 · 4 min de lectura

Se dice que el ***problema de la consulta N + 1*** ocurre cuando un ORM, como hibernate, ejecuta 1 consulta para recuperar la entidad principal y N consultas para recuperar las entidades secundarias. A medida que aumenta el número de entidades en la base de datos, las consultas que se ejecutan por separado pueden afectar fácilmente el rendimiento de la aplicación. Este artículo demostrará cómo ocurren las consultas N + 1 y su solución a través de un ejemplo en Spring Boot.

El problema de las consultas N + 1

Considere un sistema de gestión de contenido que almacena una *lista de artículos* por *publicación* . Una *publicación* puede tener una *categoría* y una *lista de artículos* asociados. A continuación se muestra una entidad simplificada para publicación y artículo.

```
@Entity
@Table(name = "publication")
public class Publication {
```

```
@Entity
@Table(name = "article")
public class Article {
```

```java
    @Id
    private String publicationId;

    private String name;

    private String category;

    @OneToMany(cascade = CascadeType.ALL)
    @JoinColumn(name = "publicationId")
    private List<Article> articles;
}
```

```java
    @Id
    private String articleId;

    private String title;

    private String publicationId;
}
```

publicationId is the Foreign Key in Article

* entidades simplificadas con el fin de explicar consultas N + 1

Una publicación tiene una relación de *uno a varios* con una entidad de artículo. Suponga que la base de datos tiene actualmente los siguientes datos, con *publicación_1* con dos artículos y *publicación_2* con un artículo.

Table: publication

publication_id	name	category
publication_1	pb_name1	technology
publication_2	pb_name2	technology

Table: article

article_id	title	publication_id
article_1	title1	publication_1
article_2	title2	publication_1
article_3	title3	publication_2

Ahora, cuando intentamos buscar todas las publicaciones que ***pertenecen a*** una determinada categoría, digamos *tecnología* , usando ***findByCategory ('tecnología'),*** primero se ejecuta
una consulta *SELECT* para obtener registros de la tabla Publicación.

```
SELECT * FROM publication WHERE category = 'technology';
```

* Consultas simplificadas para mayor comprensión.

La consulta anterior devuelve dos registros con los identificadores ' *publicación_1* ' y ' *publicación_2* '. Ahora, para cada una de estas publicaciones, debe obtener los artículos

correspondientes de la tabla de *artículos* y JPA genera internamente dos consultas *SELECT* más :

```
SELECT * FROM article WHERE publication_id = 'publication_1';

SELECT * FROM article WHERE publication_id = 'publication_2';
```

* Consultas simplificadas para mayor comprensión.

Como puede ver aquí, después de la primera consulta para obtener datos de la tabla de publicación, se generaron dos consultas adicionales **(N = 2)** para obtener los artículos relacionados de la tabla secundaria. Por lo tanto, para obtener los datos, se generan consultas **N + 1** , donde **N** es el número de entidades en la tabla principal.

EntityGraphs de JPA

EntityGraphs proporciona una forma de formular consultas de mejor rendimiento al definir qué entidades deben recuperarse de la base de datos mediante SQL JOINS.

Hay dos tipos de entityGraphs, ***Fetch*** y ***Load*** , que definen si las entidades ***no*** especificadas por attributeNodes de entityGraphs deben buscarse de forma ***perezosa*** o ***ansiosa*** . Los atributos especificados por attributeNodes de entityGraph *siempre* se obtienen *con entusiasmo* .

> ***TIPO DE FETCH: Los*** *atributos que están especificados por attributeNodes de entityGraph se tratan como FetchType.EAGER y el resto de los atributos se tratan como FetchType.Lazy.*
>
> ***TIPO DE CARGA: Los*** *atributos que están especificados por attributeNodes de entityGraph se tratan como FetchType.EAGER y el resto de los atributos se tratan de acuerdo con sus fetchTypes especificados o predeterminados.*

EntityGraphs se puede definir de dos formas:

1. Uso de la anotación NamedEntityGraph

Para usar un NamedEntityGraph, primero anote la clase de entidad Publication con la anotación **@NamedEntityGraph** de JPA , y luego adjunte la anotación **@EntityGraph** al método del repositorio, con el nombre del gráfico.

```java
@Entity
@Table(name = "publication")
@NamedEntityGraph(name = "publication-articles-graph",
        attributeNodes = @NamedAttributeNode(value = "articles"))
public class Publication {

    @Id
    private String publicationId;

    private String name;

    private String category;

    @OneToMany(cascade = CascadeType.ALL)
    @JoinColumn(name = "publicationId")
    private List<Article> articles;
}
```

ATTRIBUTE TO FETCH VIA JOIN

NAME OF THE ENTTIY GRAPH

REPOSITORY

```java
public interface PublicationRepository extends JpaRepository<Publication, String> {
    @EntityGraph(type = EntityGraph.EntityGraphType.FETCH,
                 value = "publication-articles-graph")
    List<Publication> findByCategory(String category);
}
```

2. Sin anotación NamedEntityGraph

También puede definir un EntityGraph ad-hoc, usando ***attributePaths*** , sin usar la anotación NamedEntityGraph en la entidad. ***AttributePaths*** debe incluir los nombres de las entidades que se buscarán con entusiasmo.

REPOSITORY

```java
public interface PublicationRepository extends JpaRepository<Publication, String> {
    @EntityGraph(type = EntityGraph.EntityGraphType.FETCH,
                 attributePaths = "articles")
    List<Publication> findByCategory(String category);
}
```

NAME OF ATTRIBUTE TO FETCH VIA JOIN

Los EntityGraphs ad-hoc son más dinámicos y se utilizan para un caso de uso único o específico. Los NamedEntityGraphs son útiles sobre los ad-hoc, cuando hay múltiples usos de consultas en la misma entidad. Los atributos que se van a obtener mediante

JOIN en cada consulta se pueden especificar solo una vez en el NamedEntityGraph para todas las consultas.
Consulta generada por JPA después de usar EntityGraphs:

```
SELECT * FROM publication LEFT OUTER JOIN article
ON publication.publication_id = article.publication_id
WHERE publication.category = 'technology'
```

Por lo tanto, las consultas N + 1 se redujeron a una sola consulta para obtener datos de ambas tablas, utilizando JOIN.
EntityGraphs proporciona un mecanismo mediante el cual las entidades se pueden obtener con entusiasmo de la base de datos en una sola declaración de selección, lo que ayuda a mejorar el rendimiento de la aplicación. También puede usar Subgraphs para definir las entidades para la clase secundaria, que debe buscarse con entusiasmo junto con la clase principal.

Jpa Hibernar Jpa Bota de primavera Java Optimización de consultas