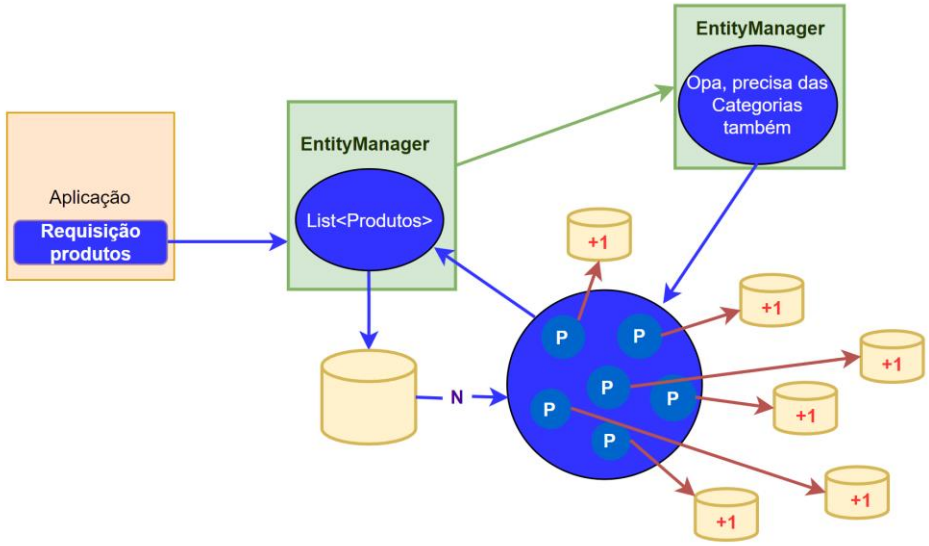
Querys N+1

Querys N+1: é um problema que acontece quando fazemos mais consultas para complementar a primeira.

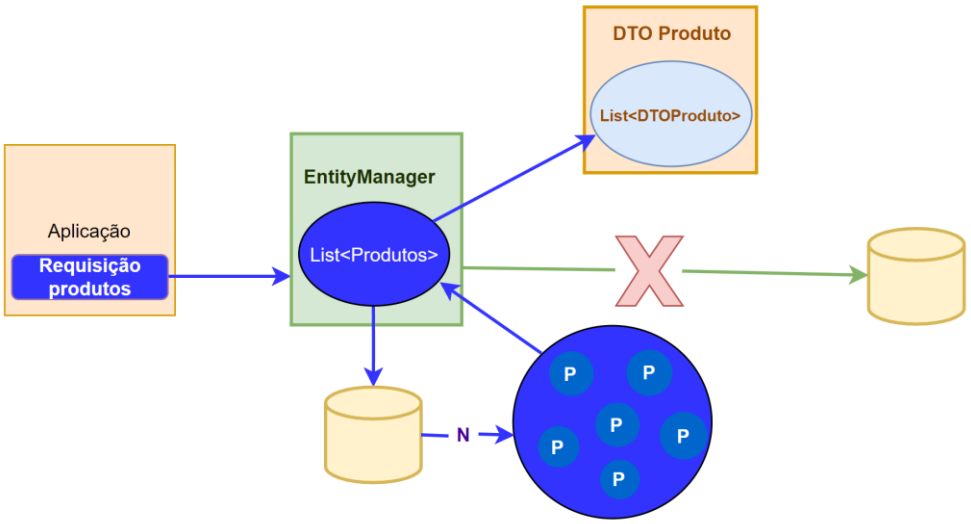
Por exemplo, eu faço **UMA CONSULTA** de uma lista de **Produtos**, o **EntityManager** me entrega essa lista, mas ele percebe que **produto** se relaciona com outra Entidade que é Categoria, então o **EntityManager** vai fazer **MAIS UM select** das categorias de **cada um** dos **Produtos** retornados na **primeira query:**



Usar DTO

Um dos primeiros passos para resolver esse tipo de problema é usar **DTO's**, **DTO's** são **objetos de transferência de dados** (na **prática ele é uma classe espelho construído a partir da "entidade original"**).

Essa prática garante que **eu sempre vou transferir os dados que eu quiser** além de **tirar o monitoramento do JPA,** ou seja, eu busco os **registros da "ENTIDADE ORIGINAL"** e passo pro **DTO** dela, quando eu passo pro DTO já era minha conexão, eu **NÃO DEPENDO MAIS DE UMA CONEXÃO**:



Usar JOIN FETCH não permite consulta paginada

Um **JOIN FETCH** faz a **junção entre as entidades referenciadas**, esse tipo de consulta **NÃO PERMITE** usar a clausula **LIMIT** ***(na vdd permite no banco de dados, mas o resultado não vai funcionar como esperado pro SDJ, que vai lançar uma exceção)*** que é o que a paginação do **Spring Data JPA** utiliza. Um **JOIN FETCH** é equivalente a isso:

```
SELECT * FROM tb_product
INNER JOIN tb_product_category ON tb_product.id = tb_product_category.product_id
INNER JOIN tb_category ON tb_category.id = tb_product_category.category_id
```

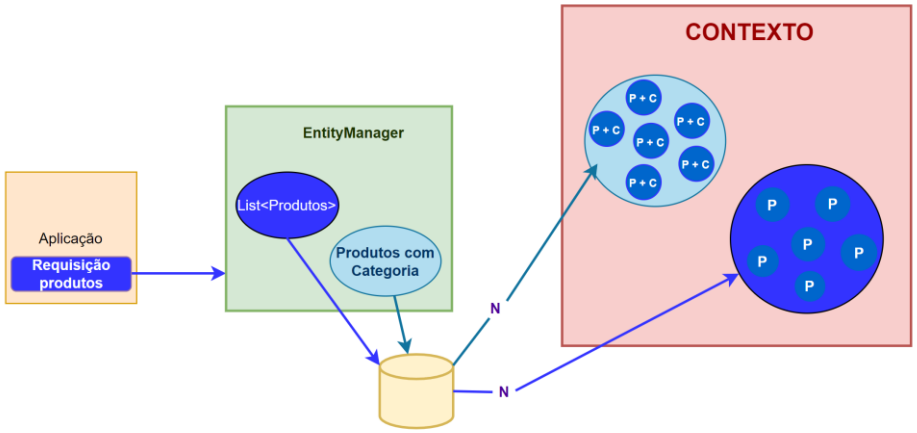
Alternativa

Uma alternativa para esse problema seria fazer **uma outra query** que **funcionária** em cima do **request paginado**. Por exemplo, primeiro eu busco a **página de Produtos**, depois disso eu faço a **busca das categorias** com um **SELECT** limitado pela clausula **"IN" produtos da página:**

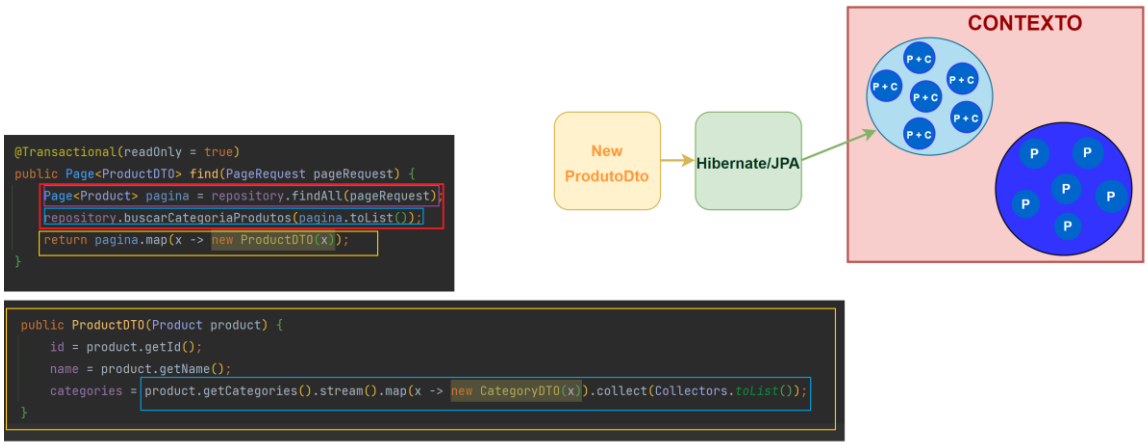
```
public interface ProductRepository extends JpaRepository<Product, Long> {

    @Query("from Product p JOIN FETCH p.categories WHERE p IN :produtos")
    List<Product> buscarCategoriaProdutos(List<Product> produtos);

}
```

```
@Service
public class ProductService {

    @Autowired
    private ProductRepository repository;

    @Transactional(readOnly = true)
    public Page<ProductDTO> find(PageRequest pageRequest) {
        Page<Product> pagina = repository.findAll(pageRequest);
        repository.buscarCategoriaProdutos(pagina.toList());
        return pagina.map(x -> new ProductDTO(x));
    }
}
```

E por que isso funciona? Quando eu faço essa **consulta dos Produtos paginados** eu também faço a **consulta das categorias desses Produtos**, eu estou **fazendo duas querys no mesmo CONTEXTO:**



O JPA/Hibernate criam ***"mapas de identidade"***, esses mapas são **caches em memória dos resultados buscados no banco de dados**, então o **Hibernate/JPA** sabem que eles já têm esses objetos no **CONTEXTO** da busca, dessa maneira quando eu instanciar o **DTO de Produto**, eu vou buscar esses Produtos com categoria da **memória/contexto**:



Dessa maneira **conseguimos fazer duas consultas por página de Produto**, invés das **+1 geradas para cada produto anteriormente:**