Artigo

Hibernate Query Language - Do início ao fim

Veja neste artigo como usar o HQL (Hibernate Query Language) e os detalhes do seu funcionamento. Mostramos também como testar suas HQLs no próprio Eclipse.

Neste artigo veremos como usar o HQL como recurso do Hibernate para melhorar a produtividade na construção de um projeto com uma grande quantidade de relacionamento complexos, nosso foco será demonstrar na prática como usar tal recurso.

construção de queries usando os mapeamentos que o próprio Hibernate gerencia. Acontece que algumas queries, dada a quantidade de relacionamentos envolvidos, poderiam gerar SQL's enormes se comparados ao HQL que pode ser muito menor. Imagine, por exemplo, um relacionamento entre 10 classes para gerar um relatório, a construção de um SQL dessa dimensão poderia demorar horas ou quem sabe dias, e com HQL nós temos a facilidade de “comprimir” essa tempo ao máximo, mas isso teu sem custos e veremos mais a diante quais são.

Diferença do SQL, o HQL é totalmente orientado a objetos, isso significa que ele “entende” as técnicas envolvidas nesse paradigma, como por exemplo: herança, polimorfirmos, associações e etc.

Vamos começar nosso artigo apresentando alguns exemplos básicos retirados da própria documentação oficial e depois iremos aos exemplos mais reais e complexos. Observe a **Listagem 1**.

Listagem 1. HQL Básico

```
1 | SELECT c FROM Cat c
```

O que temos acima é o retorno de todas as instâncias do objeto Cat. Em SQL ficaria como o código presente na **Listagem 2.**

Listagem 2. SQL da **Listagem 1**

```
1 | SELECT c.* FROM tabela_cat c
```

Só foi possível gerar o SQL da **Listagem 2**, pois nosso mapeamento da classe Cat

```
@Entity
@Table(name = "tabela_cat")
public class Cat {

}
```

Perceba que o Hibernate foi até a classe Cat, procurou o nome da tabela relacionada a classe Cat e construiu o SQL a partir do HQL que nós montamos na **Listagem 1**. Em tese, sendo bem generalista, é isso que o Hibernate faz. Vai até o mapeamento da sua classe e busca nome das colunas e tabelas afim de construir o SQL apropriado, em outra palavras, se seu mapeamento está errado então inevitavelmente a construção do SQL também será feita da forma errada, não porque o Hibernate fez errado mas sim porque você mapeou errado.

Obviamente que o exemplo acima é bem básico e a diferença do SQL para o HQL é mínima mas tem suas vantagens, que veremos mais a frente.

HQL e seus recursos

Listagem 4. Associações e Uniões

```
Select cat from Cat as cat
    inner join cat.mate as mate
    left outer join cat.kittens as kitten
WHERE mate.age = '30' AND kitten.age > '10'
```

Na **Listagem 4** temos um exemplo que envolve o relacionamento entre três classes: Cat, Mate e Kitten. A Classe Cat tem um Mate associado a ela e pode ter vários “Kitten”. Estamos buscando todos os objetos Cat quem possuem um Mate

que a primeira é do tipo Mate e a segunda é uma Lista de Kitten. Veja como fica na **Listagem 5.**

Listagem 5. Classe Cat

```
@Entity
@Table(name = "tabela_cat")
public class Cat {
	private Mate mate;
	private List<Kitten> kittens;

	public Mate getMate() {
		return mate;
	}

	public void setMate(Mate mate) {
		this.mate = mate;
	}

	public List<Kitten> getKittens() {
		return kittens;
	}

	public void setKittens(List<Kitten> kittens) {
		this.kittens = kittens;
	}

}
```

Desconsideramos os mapeamentos das propriedades, apenas para que você possa ver como nossa classe Cat está montada. Então na **Listagem 4** retornamos uma lista de objetos Cat mas esses objetos não tem suas propriedades inicializadas (mate e kittens), ou seja, se você tentar executar o código da **Listagem 6** após a query da **Listagem 4**, você receberá um LazyInitializationException

```
3       left outer join cat.kittens as kitten
4   WHERE mate.age = '30' AND kitten.age > '10'";
5   Cat cat = findCat(meuHQL);
6   cat.getMate(); //EXCEPTION
7   cat.getKittens();//EXCEPTION
```

Nesse caso teríamos que buscar o Mate e atribuir ao Cat, depois buscar os Kitten's e atribuir também ao Cat, mas o HQL nos fornece um recurso muito interessante para evitar tanto stress: o **fetch**.

A palavra reservada **fetch** do HQL, força com o que o Hibernate inicialize as propriedades que possuem o fetch associado, ou seja, além de você realizar o JOIN, você ainda quer que esse objeto seja inicializado. Vejamos o código da **Listagem 7.**

Listagem 7. Usando fetch

```
1   SELECT cat from Cat as cat
2       inner join fetch cat.mate
3       left join fetch cat.kittens child
4       left join fetch child.kittens
```

Com a palavra reservada fetch após nosso JOIN (inner join, left join e etc) e antes do nome da propriedade, nós conseguimos incializá-la ignorando, por exemplo, o "fetch = LAZY " que deve estar setado no mapeamento da Classe Cat.

Ficou complicado? Vamos explicar de forma mais simples e completa. Nossa classe Cat da **Listagem 8** tem todos os mapeamentos necessários.

Listagem 8. Classe Cat completa

```
6       @JoinColumn(name = "id_mate")
7           private Mate mate;
8
9           @OneToMany(fetch = javax.persistence.FetchType.LAZY, cascade
10                      mappedBy = "cat", targetEntity = Kitten.class)
11          private List<Kitten> kittens;
12
13          public Mate getMate() {
14                return mate;
15          }
16
17          public void setMate(Mate mate) {
18                this.mate = mate;
19          }
20
21          public List<Kitten> getKittens() {
22                return kittens;
23          }
24
25          public void setKittens(List<Kitten> kittens) {
26                this.kittens = kittens;
27          }
28
29  }
```

Perceba que ambas as propriedades mate e kittens estão com o atributo “fetch = javax.persistence.FetchType.LAZY”. Isso significa que quando for executado qualquer HQL (que não possua o fetch) essas propriedades não estarão inicializadas por padrão, estratégia muito utilizada para aumento de performance na aplicação, pois nem sempre precisaremos chamar o “getKittens()” ou “getMate()” através do Cat e seria um desperdício carregar essas propriedades toda vez.

Então a única opção e geralmente a melhor, é trazer essas propriedades carregadas através do HQL, mas se usarmos apenas o JOIN sem o fetch o Hibernate não carregará internamente. Vejamos o código da **Listagem 9.**

Listagem 9. Com Fetch e Sem Fetch

```
/* SEM FETCH */
String meuHQL = “Select cat from Cat as cat
    inner join cat.mate as mate
    left outer join cat.kittens as kitten
WHERE mate.age = '30' AND kitten.age > '10'”;
Cat cat = findCat(meuHQL);

//Apresenta erro pois a propriedade mate não está inicializada
cat.getMate();
//Apresenta erro pois a propriedade kittens não está inicializada
cat.getKittens();

/* COM FETCH */
String meuHQL = “Select cat from Cat as cat
    inner join fetch cat.mate as mate
    left outer join fetch cat.kittens as kitten
WHERE mate.age = '30' AND kitten.age > '10'”;
Cat cat = findCat(meuHQL);

//O Fetch ignora o atributo 'fetch = LAZY' do mapeamento
cat.getMate();
cat.getKittens();
```

Se sua classe tiver muitas propriedades você pode optar por usar o **fetch all properties** que força o Hibernate a inicializar todas as propriedades de determinada classe. Veja como ficaria na classe Cat (**Listagem 10**).

Assim como no SQL, existem diversos recursos que você pode usar no HQL, na verdade quase todos: like, limit, where, <, >, % e muitos outros que você verá ao longo deste artigo. Veja o uso do like na **Listagem 11.**

Listagem 11. Usando like no HQL

```
1 | Select cat from Cat as cat where cat.mate.name like '%s%'
```

Vale lembrar que o HQL retorna outros objetos que estão dentro da classe Cat, se necessário, e não apenas o Cat. Vejamos a **Listagem 12.**

Listagem 12. Retornando mate

```
1 | select mate
2 | from Cat as cat
3 |     inner join cat.mate as mate
```

Perceba na **Listagem 12** que estamos retornando o mate que está dentro do Cat, podemos também optar por outra forma, como a presente na **Listagem 13.**

Listagem 13. Retornando mate de forma compacta

```
1 | select cat.mate from Cat cat
```

Também não somos obrigados a retornar o objeto inteiro, podemos retornar apenas propriedades específicas, como vemos na **Listagem 14.**

Listagem 14. Retornando propriedades específicas

Vejamos a **Listagem 15.**

Listagem 15. Instanciando classes dentro do HQL

```
select new Family(mother, mate, offspr)
from DomesticCat as mother
    join mother.mate as mate
    left join mother.kittens as offspr
```

Dado que a classe Family tem um construtor que recebe os três parâmetros que passamos, então nós podemos retornar um objeto Family como mostrado na **Listagem 15**. Essa forma de construção é muito interessante quando precisamos criar relatórios complexos que contém campos mixados.

Podemos fazer uso dos recursos da linguagem Java juntamente com o HQL, então nesse caso podemos criar até um List dentro do HQL e retorná-lo, como o código da **Listagem 16.**

Listagem 16. Instanciando um List dentro do HQL

```
select new list(mother, offspr, mate.name)
from DomesticCat as mother
    inner join mother.mate as mate
    left outer join mother.kittens as offspr
```

Você já deve ter imaginado que podemos usar outras classes como Map, que é muito útil para nomear nossos atributos e recuperá-los em qualquer parte do código. Vejamos a **Listagem 17.**

Listagem 17. Usando Map no HQL

Temos então três propriedades nomeados com max, min e n respectivamente. Assim podemos chamá-las pelo seu alias dentro do código. Olhe na **Listagem 18** como ficaria.

Listagem 18. Usando Map

```
String hql = “select new map( max(bodyWeight) as max, min(bodyWeight
from Cat cat”;
Map retorno = find(hql);
System.out.println(retorno.get(“max”));
System.out.println(retorno.get(“min”));
System.out.println(retorno.get(“n”));
```

Agregando valores com HQL

O HQL também nos permite usar as funções avg, sum, max, count e min. Vejamos exemplos práticos do seu uso. Vamos começar pela **Listagem 19.**

Listagem 19. Usando funções de agregação

```
select avg(cat.weight), sum(cat.weight), max(cat.weight), count(cat)
from Cat cat
```

Como dissemos anteriormente, o HQL é muito parecido com o SQL e prova disso é o uso de operadores aritméticos (**Listagem 20**) e de concatenação (**Listagem 21**) idênticos ao SQL. **Listagem 20.** Operador de soma com sum

Listagem 21. Concatenação

```
1 | select firstName||' '||initial||' '||upper(lastName) from Person
```

Temos também o count e o distinct idênticos ao SQL.

Listagem 22. Count e Distinct

```
1 | select distinct cat.name from Cat cat
2 |
3 | select count(distinct cat.name), count(cat) from Cat cat
```

Na **Listagem 22** realizamos um distinct na propriedade “cat.name” que nos trará todos os objetos da classe Cat sem repetir a propriedade name, e também realizamos um count com distinct em cat.name e depois em cat.

Aprofundando-se na cláusula WHERE

Igualmente no SQL, a cláusula WHERE no HQL filtra a quantidade de registros retornados e pode ser utilizada de inúmeras formas que veremos na **Listagem 23**.

Listagem 23. Where com alias e sem alias

```
1 | --Com alias
2 | from Cat as cat where cat.name='Fritz'
3 |
4 | --Sem alias
5 | from Cat where name='Fritz'
```

Podemos usar o WHERE com a alias (cat) ou sem ela, dependendo de como

```
select foo
from Foo foo, Bar bar
where foo.startDate = bar.date
```

Igualmente no SQL a comparação de datas é feita com um simples sinal de igualdade, e podemos também incrementar nosso HQL com a palavra reservada "is not null", como vemos na **Listagem 25.**

Listagem 25. Is not null

```
from Cat cat where cat.mate.name is not null
```

A grande diferença com o operador de igual do SQL e do HQL é que no HQL ele pode não apenas comparar propriedades, mas instâncias, ou seja comparar se dois objetos são iguais. Vejamos um exemplo na **Listagem 26.**

Listagem 26. Usando = para comparar objetos

```
select cat, mate
from Cat cat, Cat mate
where cat.mate = mate
```

Atente para a **Listagem 26** comparando o objeto cat.mate com outro mate. E como essa comparação é realizada ? Através do método "equals()" que deve estar implementado de forma correta na sua classe. Podemos além de comparar objetos (instâncias de classes), comparar as próprias classes, ou seja, checar se determinado resultado é da classe que esperamos, conforme mostra a **Listagem 27.**

Listagem 27. Comparando classes

Comparamos na **Listagem 27** se a classe do item que está dentro da classe AuditLog é da classe 'Payment'.

Usando Expressões no HQL

Esta poderosa linguagem é composta por diversos recursos extras, que tornam ainda mais poderosa a construção de uma query. Vejamos nessa seção esses recursos chamados de expressões.

De acordo com o Manual oficial temos as expressões permitidas em HQL listadas abaixo:

- *operadores matemáticos:+, -, *, /*
- *operadores de comparação binários:=, >=, <=, <>, !=, like*
- *operadores lógicos: and, or, not*
- *Parênteses: ( )que indica o agrupamento*
- *in,not in,between,is null,is not null,is empty,is not empty,member of and not member of*
- *case "simples" ,case ... when ... then ... else ... end, and "searched" case,case when ... then ... else ... end*
- *concatenação de string...||...ouconcat(...,...)*
- *current_date(),current_time()ecurrent_timestamp()*
- *second(...),minute(...),hour(...),day(...),month(...) e year(...)*
- *qualquer função ou operador definidos pela EJB-QL 3.0:substring(), trim(), lower(), upper(), length(), locate(), abs(), sqrt(), bit_length(), mod()*
- *coalesce() and nullif()*

dados adjacente

- *A função HQL index(), que se aplicam às referências de coleções associadas e indexadas*
- *As funções HQL que retornam expressões de coleções de valores:size(), minelement(), maxelement(), minindex(), maxindex(), junto com o elemento especial,elements() e funções de índices que podem ser quantificadas usando some, all, exists, any, in.*
- *Qualquer função escalar suportada pelo banco de dados como sign(),trunc(),rtrim() e sin()*
- *Parâmetros posicionais ao estilo JDBC?*
- *Parâmetros nomeados:name,:start_date e :x1*
- *Literais SQL 'foo', 69, 6.66E+2, '1970-01-01 10:00:01.0'*
- *Constantes Java final estático públicoex: Color.TABBY*

Dada a listagem acima, vamos ver como utilizar algumas dessas expressões na prática.

Listagem 28. Between

```
1 from DomesticCat cat where cat.name between 'A' and 'B'
```

Listagem 29. in

```
1 from DomesticCat cat where cat.name in ( 'Foo', 'Bar', 'Baz' )
```

A expressão between (**Listagem 28**) filtra registros que estão entre o argumento A e o argumento B, enquanto a expressão in (**Listagem 29**) filtra aqueles que estão dentro da listagem, ou seja, que tem o nome igual a 'Foo', 'Bar' ou 'Baz'.

Na **Listagem 30** filtramos todos os objetos Cat que tem pelo menos 1 Kitten, ou seja, que possui Kitten maior que zero. Muito útil para retornar Vendedores que possuem pelo menos um venda, ou aqueles que não venderam nada, como podemos ver na **Listagem 31**.

Listagem 31. size() com vendedores sem vendas

```
1 | from Vendedor vendedor where size(vendedor.vendas) = 0
```

Testando HQL no Eclipse

Após todas as seções acima você pode se perguntar onde testar tais queries? Normalmente apenas usando-as diretamente na aplicação para checar se está tudo como você precisa, mas mostraremos nessa seção que existe outra forma. Dentro do próprio IDE Eclipse você tem como testar suas HQL's antes de colocar as mesmas na sua aplicação, assim a produtividade aumenta e muito, sendo que determinadas queries só serão executadas em ocasiões muito específicas e seria difícil testar.

Primeiro você deve abrir a Perspectiva do Hibernate na sua IDE, igual a **Figura 1.**

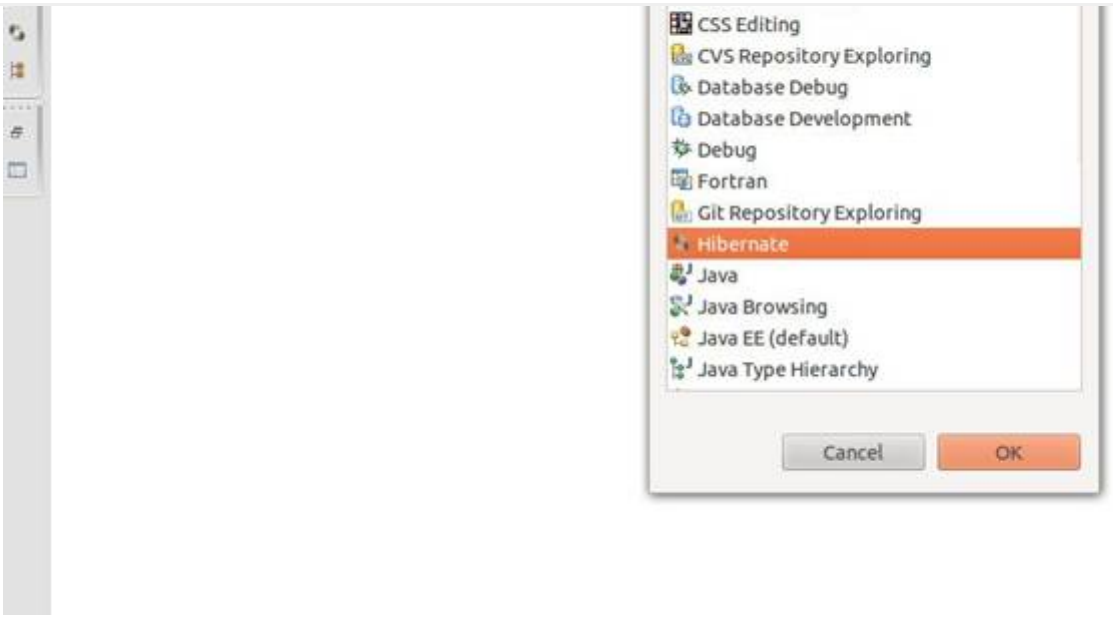

Figura 1. Perspectiva Hibernate

Depois você terá criar um arquivo de configuração a sua base de dados, lembrando que os mapeamentos já devem estar criados na sua aplicação, conforme a **Figura 2.**

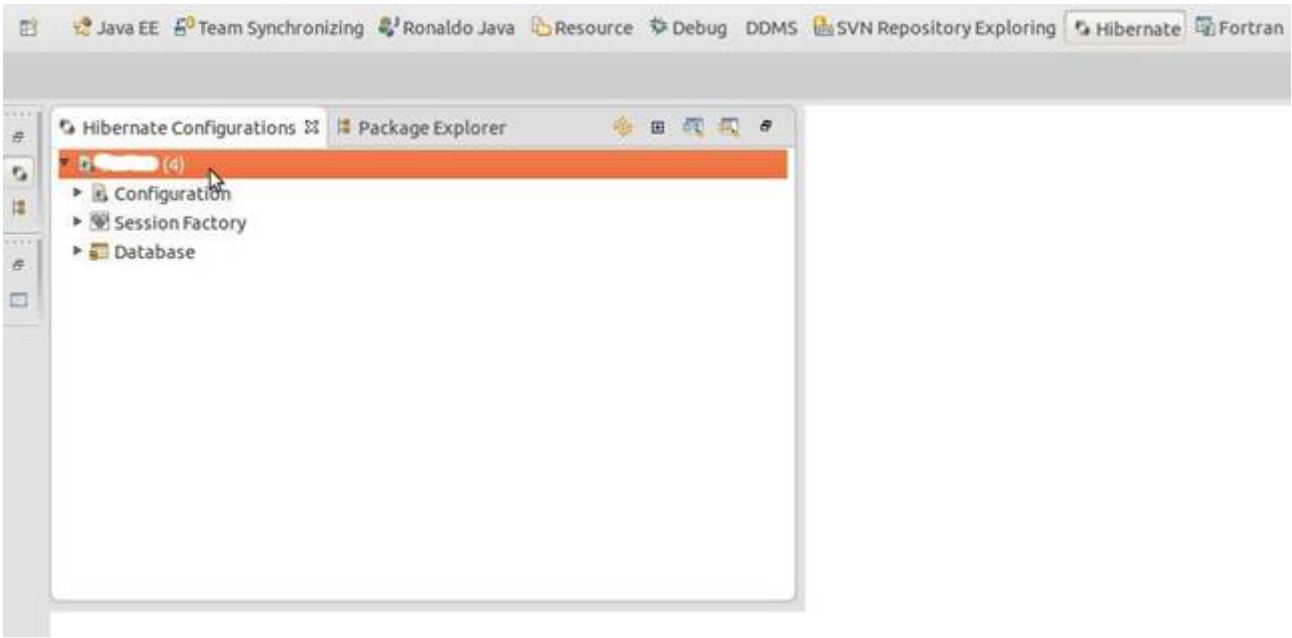

Figura 2. Arquivo de Configuração

linha “Database” devem aparecer seus esquemas e tabelas.

Por fim, com tudo configurado, basta você clicar no ícone azul escrito “HQL” com uma lupa, e você verá uma console para digitação de comandos HQL, conforme a **Figura 3**.

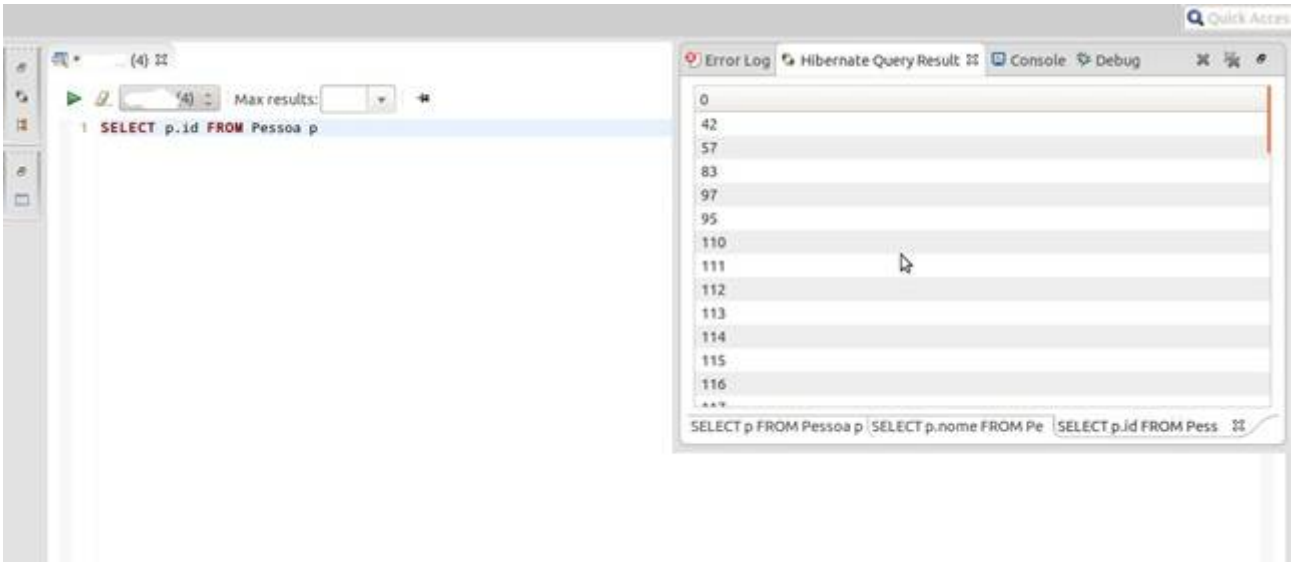

Figura 3. Console do HQL

Perceba que fizemos um simples “SELECT p.id FROM Pessoa p” que nos retornou a propriedade id de todos os objetos pessoas do banco de dados.

Prós e Contras

O HQL não tem apenas lados positivos mas também lados negativos, vejamos:

Prós

1. Facilidade na criação de queries complexas com muitos relacionamentos;
2. Aumento na produtividade do projeto devido o item 1;

4. Adaptado para trabalhar com conceitos de Orientação a Objetos;

Contras

1. Perca de performance comparado ao SQL comum, pois é feita a construção do SQL a partir de todo mapeamento de classes;
2. Pode tornar-se complexo se não o desenvolvedor não conhece muito bem como funcionam os mapeamentos de classes do Hibernate (OneToOne, OneToMany e ManyToOne);
3. Dependência do framework Hibernate, o que deixa o projeto totalmente dependente de uma tecnologia;

Dado alguns pontos positivos e negativos, cabe a você analisar as situações em que se faz útil o uso do HQL. Visto que, os pontos negativos 2 e 3 podem não ser tão impactantes dependendo do caso, mas o ponto negativo 1 pode fazer diferença na geração de um relatório muito complexo.

Tecnologias:

Hibernate SQL

Por **Ronaldo**
Em 2014