A *framework* orm-queries oferece a capacidade de qualquer data mapper poder executar uma operação `where`, conforme exemplificado no seguinte teste:

```
Iterable<Product> res = mapper.where("UnitPrice >= 50");
int count = 0;
for (Product p : res) {count++;}
Assert.assertEquals(count, 7);
```

Disponibiliza ainda a possibilidade de sobre o resultado da execução de uma query (operação *loadAll* ou *where*) possa ser adicionada uma cláusula de *order by*. O método `orderBy` recebe como parâmetro o nome da propriedade que serve de critério de comparação (as propriedades suportadas são de tipo primitivo ou tipos compatíveis com java.lang.`Comparable`). Exemplo:

```
Iterable<Product> res = mapper.where("UnitPrice >= 50").orderBy("UnitsInStock");
int [] expected = {29, 38, 51, 9, 20, 18, 59};
int i = 0;
for (Product p : res) {
  Assert.assertEquals(expected[i++], p.getId().intValue());
}
```

A operação `loadAll`, de modo a tornar o carregamento dos dados *lazy* e que sobre o seu resultado possa ainda ser aplicado uma operação de `where` ou `orderBy` (também lazy) – a *query* só deverá ser executada na bases de dados quando o seu resultado estiver a ser iterado. Exemplos:

```
//
// loadAll
//
Iterable<Product> res = mapper.loadAll();
int i = 1;
for (Product p : res) {
        Assert.assertEquals(i++, p.getId().intValue());
}
//
// loadAll + where
//
res = mapper.loadAll().where("UnitPrice >= 50");
int[]expected = {9,    18,     20,     29,     38,     51,     59};
i = 0;
for (Product p : res) {
        Assert.assertEquals(expected[i++], p.getId().intValue());
}
//
// loadAll + orderBy
//
res = mapper.loadAll().orderBy("ProductName");
expected = new int[]{17, 3, 40, 60, 18, 1, 2, 39, 4};
i = 0;
for (Product p : res) {
        Assert.assertEquals(expected[i++], p.getId().intValue());
        if(i >= expected.length) break;
}
//
// loadAll + where + orderBy
//
res = mapper.loadAll().where("UnitPrice >= 50").orderBy("ProductName");
expected = new int[]{18, 38, 51, 9, 59, 20, 29};
i = 0;
for (Product p : res) {
        Assert.assertEquals(expected[i++], p.getId().intValue());
}
```

Além destas operações a *framework* gera automaticamente *data mappers* para entidades de um **modelo de domínio**. O método principal desta *framework* é:

- `<K, T extends Entity<K>> DataMapper<K, T> make(Class<T> entityClass)`

e.g. de utilização: `DataMapper<Integer, Product> prodsMapper = mappersFactory.make(Product.class);`

Designa-se por **propriedade** um par de métodos *getter–setter*, com um campo associado. e.g. `int getUnitsInStock()`, `void setUnitsInStock(int units)` e o campo `int unitsInStock`.

Neste contexto, dada uma **tabela** '`As`', deve existir uma **classe de domínio** '`A`' com **propriedades** de nome igual às **colunas** da tabela '`As`' e com os seguintes tipos:

- primitivo, `String`, ou `java.util.Date`, se a coluna **não** for *foreign key*.
- `ValueHolder<`*`ClasseReferida`*`>`, se a coluna for uma *foreign key*, sendo a *`ClasseReferida`* a classe de domínio correspondente à tabela referida.
- `Iterable<`*`ClasseReferida`*`>`, por cada *foreign key* de outras tabelas (correspondentes à *`ClasseReferida`*) que apontem para a classe A.

Considerando parte do diagrama da base de dados *Northwind* esquematizado na Figura 1, então a definição da classe de domínio **`Product`** correspondente à tabela `Products` segue a definição do diagrama da Figura 2.

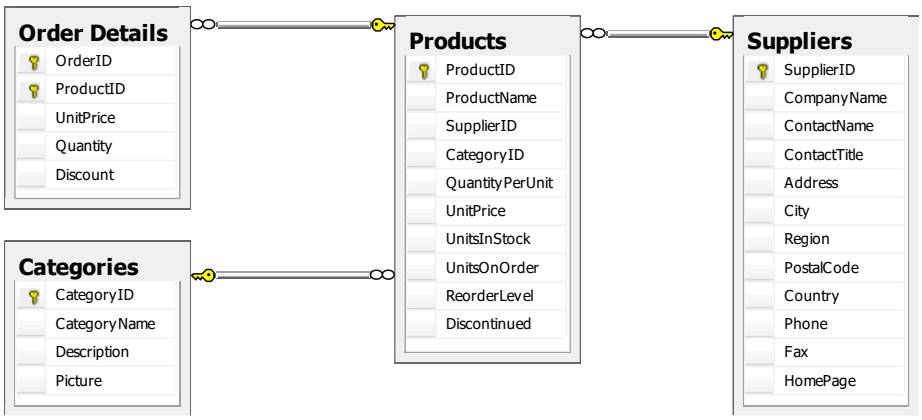

Figura 1

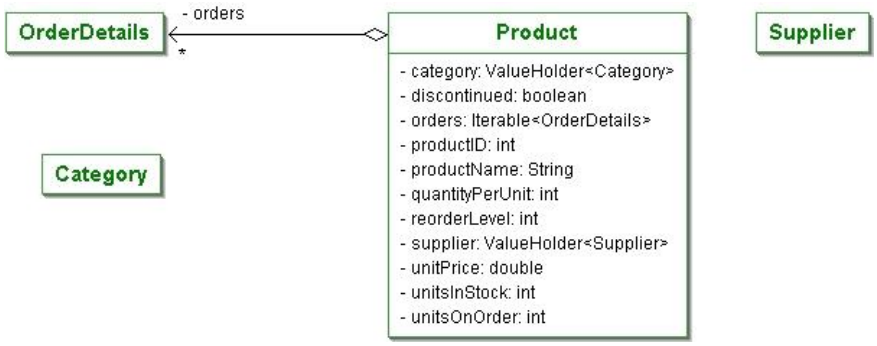

Figura 2

As classes do domínio podem definir um construtor anotado com `@JdbcMapper`, cujos parâmetros devem estar anotados de acordo com o exemplo da classe `Product` da Listagem 1 e obedecendo às seguintes convenções:

- Cada parâmetro do construtor está anotado com `@JdbcCol`, que tem `value` igual ao nome da respectiva **coluna da tabela/propriedade da classe de domínio**.
- Para as colunas que sejam chave primária, a anotação `@JdbcCol` deve indicar `isPk=true`.
- Se a chave primária for *identity* então a anotação `@JdbcCol` deve indicar o parâmetro `isIdentity = true`.
- Se o parâmetro for do tipo `ValueHolder` então a anotação `@JdbcCol` deve indicar no parâmetro `referencedKeyClass` o tipo de dados da coluna que é *foreign key*.
- Se o parâmetro for do tipo `Iterable` então a anotação `@JdbcCol` deve indicar no parâmetro `value` o nome da coluna da outra tabela que é *foreign key*.
  E.g. Na classe `Product` e no parâmetro do construtor `orders`, "`ProductId`" é o nome da coluna da tabela `Order Details` que aponta para a tabela `Product`.

```
public class Product implements Entity<Integer>{

  @JdbcMapper(table="Products")
  public Product(
    @JdbcCol(value = "ProductId", isPk=true, isIdentity = true) int productID,
    @JdbcCol("ProductName") String productName,
    @JdbcCol("UnitPrice") double unitPrice,
    @JdbcCol("UnitsInStock") int unitsInStock,
    @JdbcCol(value = "SupplierId", referencedKeyClass=Integer.class) ValueHolder<Supplier> supplier,
    @JdbcCol(value = "ProductId") Iterable<OrderDetails> orders,
    ...)
  {
    …
  }
}
```

Listagem 1

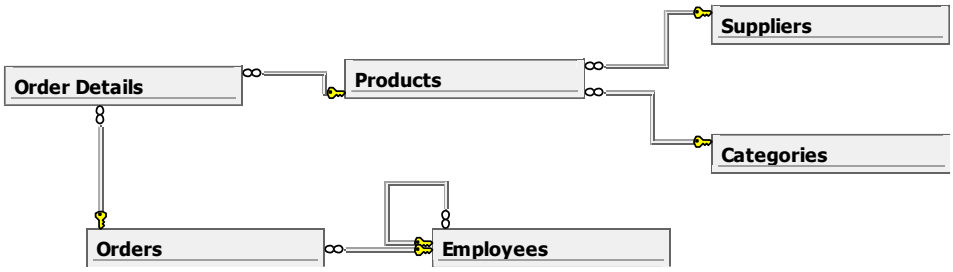

Figura 3