JPA - Java Persistense API

ORM e JPA

ORM - O que é mapeamento objeto/relacional? Como funciona? Quais os  
principais ORM do mercado?

O que é?

Object-relacional mapping – Mapeamento objeto relacional é uma técnica de desenvolvimento que consiste em mapear as classes e transformá-las em tabelas no banco de dados. Ou seja, as tabelas do banco de dados são representadas através de classes e os registros de cada tabela são representados como instâncias das classes correspondentes. Dessa forma o programador não precisa se preocupar com os comandos em linguagem SQL, ou seja, ele irá usar uma interface de programação simples que faz todo o trabalho de persistência.

Como funciona?

Utiliza-se uma abstração bastante intuitiva no sentido de que uma classe do tipo persistente pode ser mapeada para uma tabela no banco de dados relacional e atributos da classe para campos da tabela. Porém, algumas diferenças entre os dois modelos, como OID (Object Identifiers – Identificador de Objetos), tipos de dados, herança e associações, demandam um estudo mais detalhado das estratégias de mapeamento.

Identificação de chave primária

As linhas das tabelas precisam ter identidade exclusiva. Elas são identificadas com exclusividade pelos valores de suas chaves primárias e conseqüentemente, nunca devem ser alteradas. Os nomes em texto sem formatação não são adequados, pois, geralmente, representam um *overhead* operacional para o recurso relacional persistente, além do que os nomes não são exclusivos. Como as comparações numéricas consomem menos recursos computacionais, as chaves primárias devem ser numéricas e, preferencialmente, não devem refletir domínio de negócio, para que não sejam alteradas.

OIDs (Objects Identifiers)

Os OIDs são identificadores únicos que representam um objeto, em linguagens de programação, existentes, este objeto é implícito e criado quando ocorre a criação de um novo objeto, já em um banco de dados relacional cabe ao desenvolvedor a responsabilidade desta criação. Quando ocorre o mapeamento, recomenda-se armazenar no banco de dados relacional o identificador do objeto como chave primária, ou qualquer outro atributo do objeto que possa identificá-lo como único, exemplo o CPF.

Existem várias estratégias para atribuir OIDs para objetos, inclusive pode-se criar uma ou mais classes cuja responsabilidade específica é a de atribuir OIDs para objetos, sendo estas estratégias separadas das classes que implementam as regras de negócio. É considerada uma boa prática de desenvolvimento separar a estratégia de atribuição de OIDs das classes de negócio, evitando utilizar um atributo qualquer da classe para ser o identificador. Identificadores que possuem um significado de negócio, certamente mudam em algum momento, pois as regras de negócio mudam freqüentemente, e o esforço necessário para realizar esta modificação pode ser imensurável.

Mapeamento de Classes em Tabelas

O mapeamento de classes pode ser feito mediante a paridade entre classe e tabela, ou seja, uma classe é mapeada para uma tabela. Este mapeamento direto de classes para tabelas representa a forma mais simples de mapeamento, tornando mais fácil o entendimento e a manutenção de uma aplicação. Com um modelo de classes bastante simples isto poderia ser feito. Porém, nem sempre é simples assim. No caso de uma estrutura hierárquica, várias classes podem ser mapeadas para uma tabela, como também uma classe pode ser mapeada para várias tabelas. Ainda, classes com atributos multivalorados ou compostos podem ser mapeadas para mais de uma tabela.



Mapeamento de Atributos em Colunas

Ao tratar do mapeamento de atributos de uma classe para colunas em tabelas de um banco de dados relacional, deve-se levar em conta que os atributos podem ser de tipos de dados simples ou primários como: inteiros, ponto flutuante, caracteres, booleanos e binários, mas também podem ser de tipos de dados complexos como tipos baseados em outras classes. Os atributos podem ser ainda multivalorados (listas de objetos), o que viola as regras de normalização do modelo relacional. Além disso, podem existir atributos de controle ou utilizados em cálculos, que geralmente não necessitam ser mapeados. Desta forma, os atributos simples podem ser mapeados diretamente para colunas em uma tabela, já os atributos complexos e multivalorados podem necessitar de tabelas adicionais para seu armazenamento. Estes atributos complexos geralmente possuem características recursivas, ou seja, são classes que possuem outros atributos e assim sucessivamente.

Mapeamento de Herança

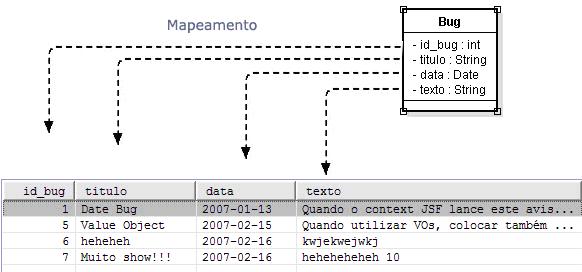
Existem fundamentalmente três estratégias para mapear herança em um banco de dados relacional:

**Uma tabela por hierarquia:** Mapear toda a hierarquia de classes para uma tabela, onde todos os atributos das classes da hierarquia são armazenados nesta única tabela. A desvantagem desta estratégia é que toda vez que um objeto da hierarquia for persistido no banco, é necessário persistir também os valores das demais classes vazias, causando uma grande quantidade de campos inutilizados. Entretanto o acesso ao banco para a manipulação dos dados é mais rápido, uma vez que todos os dados estão em somente uma tabela. É adicionada uma coluna *(Object Type)* na tabela que referência qual o tipo do objeto, ou seja, de qual classe aqueles dados pertencem;

**Uma tabela por classe concreta:** Cada classe concreta mapeada reflete uma tabela com todos os atributos herdados das superclasses abstratas. A vantagem desta estratégia é a facilidade de manipulação de dados, uma vez que todos os dados de cada classe estão em apenas uma única tabela. Como desvantagem, destacase que quando se modifica uma classe abstrata, é necessário modificar todas as tabelas geradas pelas classes filhas no modelo relacional;

**Uma tabela por classe:** Cada hierárquica mapeada reflete uma tabela, relacionadas através do mecanismo de especialização padrão do banco de dados relacional (utilização de chaves estrangeiras). Segunda esta modalidade de mapeamento, tentasse ao máximo manter a normalização de dados, de forma que a estrutura final das tabelas fica bastante parecida com a hierarquia das classes representada na UML. Esta é a técnica que mais naturalmente mapeia objetos para banco de dados relacionais.



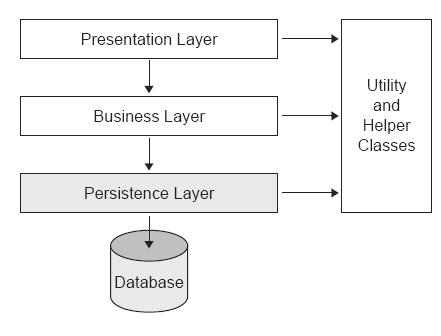


Principais ORM’s do Mercado

* [DBIx::Class](http://pt.wikipedia.org/w/index.php?title=DBIx::Class&action=edit&redlink=1) – Para à linguagem de programação Perl
* [SQLObject](http://pt.wikipedia.org/wiki/SQLObject) – Para à linguagem de programação [Python](http://pt.wikipedia.org/wiki/Python" \o "Python)
* [Hibernate](http://pt.wikipedia.org/wiki/Hibernate) – Para à linguagem de programação [Java](http://pt.wikipedia.org/wiki/Java)
* [OJB](http://pt.wikipedia.org/wiki/OJB) – Para à linguagem de programação [Java](http://pt.wikipedia.org/wiki/Java), da [Apache Software Foundation](http://pt.wikipedia.org/wiki/Apache_Software_Foundation)
* [Django (framework web)](http://pt.wikipedia.org/wiki/Django_(framework_web)) – Framework de desenvolvimento web escrito em [Python](http://pt.wikipedia.org/wiki/Python" \o "Python) que possui um [ORM](http://pt.wikipedia.org/wiki/ORM) próprio.
* [ECO - Enterprise Core Object](http://pt.wikipedia.org/w/index.php?title=ECO_-_Enterprise_Core_Object&action=edit&redlink=1) – Para à linguagem de programação [Delphi](http://pt.wikipedia.org/wiki/Delphi).
* [NHibernate](http://pt.wikipedia.org/wiki/NHibernate) - Para à linguagem de programação [.NET](http://pt.wikipedia.org/wiki/Microsoft_.NET)
* [EntityCloud](https://sourceforge.net/projects/entitycloud) – Um [ORM](http://pt.wikipedia.org/wiki/ORM) tipificado para [.NET](http://pt.wikipedia.org/wiki/Microsoft_.NET)
* [Doctrine (PHP)](http://pt.wikipedia.org/w/index.php?title=Doctrine_(PHP)&action=edit&redlink=1) - Para à linguagem de programação [PHP](http://pt.wikipedia.org/wiki/PHP)
* [Active Record](http://pt.wikipedia.org/w/index.php?title=Active_Record&action=edit&redlink=1) - Para à linguagem de programação [Ruby](http://pt.wikipedia.org/wiki/Ruby" \o "Ruby) on [Rails](http://pt.wikipedia.org/w/index.php?title=Rails&action=edit&redlink=1" \o "Rails (página não existe)).

O que é o padrão JPA? O que ele descreve? De onde surgiu?

O JPA é um framework utilizado na camada de persistência para o desenvolvedor ter uma maior produtividade, com impacto principal num modo para controlarmos a persistência dentro de Java. Ou seja, é um modo "padrão" para mapear nossos objetos para os do Banco de Dados. Persistência é uma abstração de alto-nível sobre JDBC. Sua primeira versão 1.0 foi lançada em 11 de maio de 2006, e a 2.0 em 10 de dezembro de 2009.



O Java Persistence API - JPA define um caminho para mapear Plain Old Java Objects POJOs para um banco de dados, estes POJOs são chamados de beans de entidade. Beans de Entidades são como qualquer outra classe Java, exceto que este tem que ser mapeado usando Java Persistence Metadata, para um banco de dados.

A nova Java Persistence Specitication define mapeamento entre o objeto Java e o banco de dados utilizando ORM, de forma que Beans de entidade podem ser portados facilmente entre um fabricante a outro.

 JPA define alguns elementos que vamos comentar abaixo:

1- Entity Classes

Um Entity Class representa uma tabela no banco de dados, ou seja, através desta classe, por meio de anotações, faremos o mapeamento para a tabela do banco de dados e de cada atributo para uma determinada coluna da tabela.

2- Persistent Unit

A fim de que o JPA localize o banco de dados e estabelecer as conexões JDBC, devemos criar um arquivo chamado persistence.xml no diretório META-INF. Dentro de tal arquivo é definida nossa "Persistent Unit", que define como conectar ao banco de dados (podemos trabalhar com mais de uma unidade de persistência no mesmo projeto). JPA é uma especificação para frameworks de persistência, então se faz necessário a utilização de um provedor para o mesmo, como por exemplo, TopLink Essentials, Hibernate Entity Manager, OpenJPA.

Tenha por exemplo a seguinte tabela Fornecedor (com codigo, nome e telefone) em SQL:

CREATE TABLE Fornecedor(

codigo int(11) NOT NULL auto\_increment,

nome varchar(60) NOT NULL,

telefone varchar(15) NULL,

PRIMARY KEY (codigo)

);

Agora analise a classe abaixo, veja que o mesmo tem os mesmos atributos: codigo, nome e telefone:

**@Entity**

public class Fornecedor implements Serializable {

private static final long serialVersionUID = 1L;

**@Id**

**@GeneratedValue**(strategy=GenerationType.IDENTITY)

**@Column**(unique=true, nullable=false)

private Long codigo;

**@Column**(length=60, nullable=false)

private String nome;

**@Column**(length=15)

private String telefone;

//getters e setters

}

Essa classe é a representação do código SQL acima com JPA. @Id, @Column, entre outros são chamados de anotações, eles são responsáveis por fazer o mapeamento dessa classe em código SQL. Em versões anteriores esses mapeamentos eram feitos por XML e não por anotações.

Para ficar mais claro vamos entender o que são essas anotações:

**@Entity:**essa anotação indica que esta é uma entidade gerenciada pelo JPA.

**@Id:**Define a chave primária da tabela.

**@GeneratedValue(strategy=GenerationType.IDENTITY):**corresponde ao auto\_increment do MySQL.

**@Column(...):**informa que este atributo é uma coluna da tabela Fornecedor. private Long codigo;

- Quais as principais implementações do JPA???  
- Como se configura o JPA? (persistence.xml)

Parte 2: Mapeamentos utilizando o JPA  
- O que são annotations? Como o JPA lê o mapeamento das classes?  
- Como se faz o mapeamento de classes -> tabelas? Dê exemplos.  
- Como se faz o mapeamento de propriedades da classe -> campos em  
tabelas? Dê exemplos.  
- Como se faz o mapeamento de uma chave primária de uma tabela em uma  
classe? Dê exemplos.  
- O que são coleções(collections)? Quais as principais coleções  
utilizadas em Java?  
- Como se faz o mapeamento de relacionamentos um-para-um entre duas  
classes mapeadas? Dê exemplos.  
- Como se faz o mapeamento de relacionamentos um-para-muitos entre  
duas classes mapeadas? Dê exemplos.  
- Como se faz o mapeamento de relacionamentos muitos-para-um entre  
duas classes mapeadas? Dê exemplos.  
- Como se faz o mapeamento de relacionamentos muitos-para-muitos  
entre duas classes mapeadas? Dê exemplos.  
  
Parte 3: O Gerenciador de Entidades  
- O que é o EntityManager? Para que ele serve? Dê exemplos.  
- Como se faz consultas no JPA? O que é o objeto Query?  
- O que é JPQL? Dê exemplos.  
- O que é Criteria? Dê exemplos.