

Hibernate

Thiago Costa Rizuti da Rocha



vá em cabeçalho e rodapé para editar esta seção e aplicar automaticamente em todos os slides

Sumário

- Motivação
- O que é o Hibernate ?
- Hibernate x JDBC
- Configurando o Ambiente de Trabalho
- Mapeamento ORM
- Mapeando Tabelas
- Mapeando Colunas
- Inserindo Dados

Sumário

- Criteria Queries
- HQL
- Mapeando Relacionamentos 1:1
- Mapeando Relacionamentos 1:N / N:1
- Mapeando Relacionamentos N:N
- Consultas - Relacionamentos

Motivação

Trabalhar com um software orientado a objetos e com um banco de dados relacional pode ser difícil e demorado.

A diferença de como os dados são representados pode aumentar significativamente o custo de desenvolvimento.

O que é o Hibernate ?



O Hibernate é uma framework opensource de mapeamento objeto-relacional para Java.

Foi a inspiração para a especificação JPA (Java Persistence API) e hoje é uma dentre suas várias implementações.

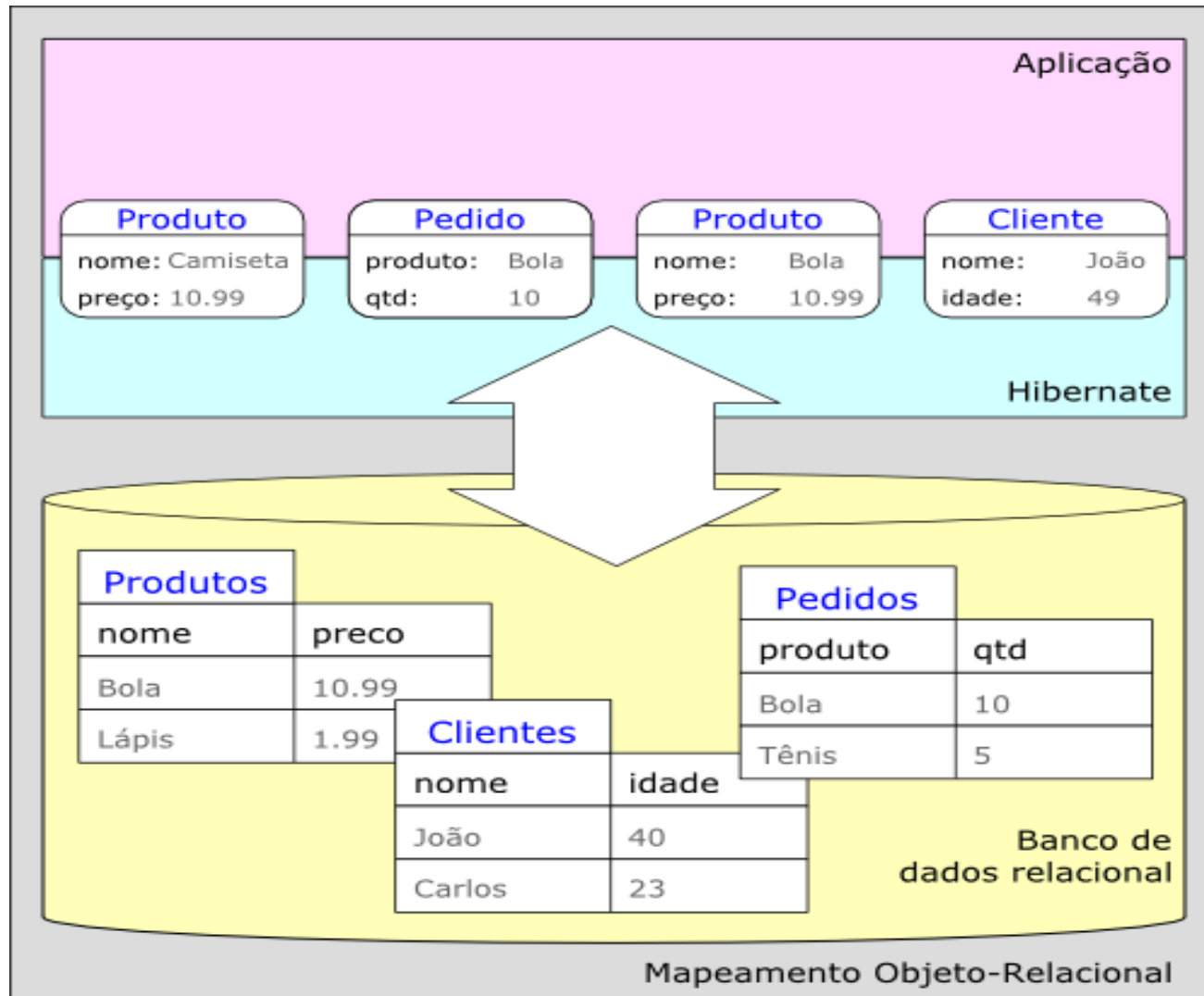
O que é o Hibernate ?

Com o Hibernate, fazemos o mapeamento entre classes Java e tabelas do banco de dados, e entre tipos Java e tipos SQL.

Ele abstrai o seu código SQL, toda a camada JDBC e o SQL será gerado em tempo de execução, facilitando a inserção e consulta de dados.

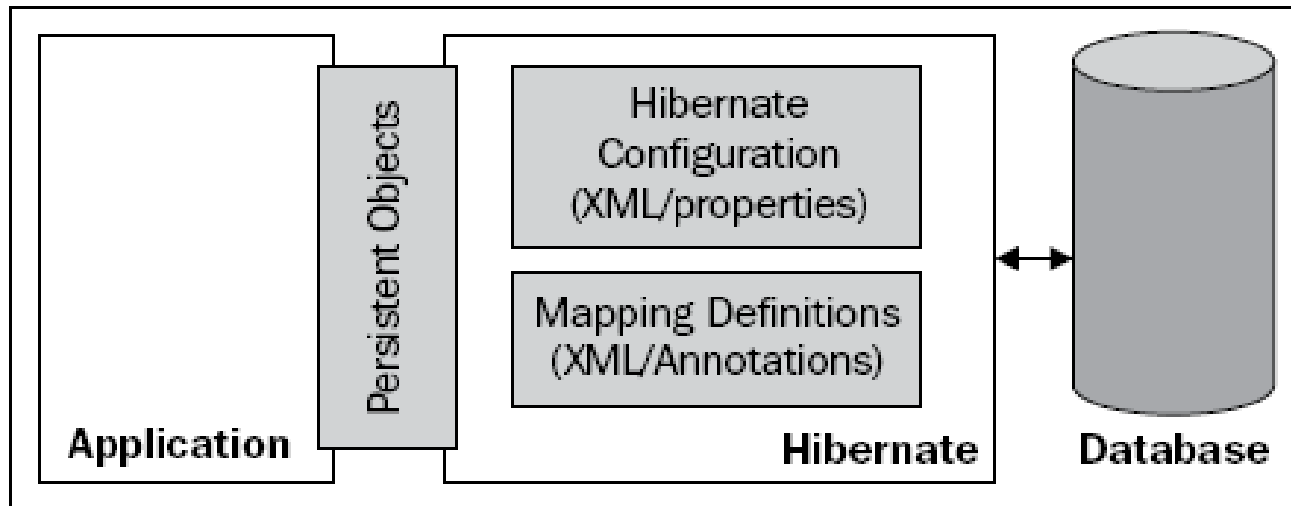
Em síntese, diminuímos a quantidade de código e economizamos tempo de trabalho.

O que é o Hibernate ?



O que é o Hibernate ?

A arquitetura do Hibernate:



Hibernate x JDBC

Usando JDBC, é necessário escrever código para mapear seu modelo orientado a objetos para o seu banco de dados.

O Hibernate facilita esse trabalho, com o mapeamento objeto-relacional através de arquivo XML.

Hibernate x JDBC

O Hibernate é independente do banco de dados, seu código irá funcionar para o Oracle, MySQL, SQLServer, etc.

No caso do JDBC, deve ser específico para o banco de dados utilizado.

Hibernate x JDBC

JDBC suporta apenas SQL.

Hibernate conta com diferentes maneiras de se fazer consultas: SQL, Hibernate Query Language (HQL) e Criteria Query.

Hibernate x JDBC

Se houver mudanças no banco de dados, com o JDBC, é necessário mudar também todo o seu código.

Com o Hibernate, você só precisa alterar as propriedades do arquivo XML de mapeamento.

Configurando o Ambiente de Trabalho

Precisaremos das bibliotecas do
Hibernate e do conector do MySQL:

hibernate.org/orm

dev.mysql.com/downloads/connector/j

Configurando o Ambiente de Trabalho

No pacote do Hibernate encontramos diretórios como `documentation`, `lib`, etc. Precisaremos dos `.jars` que se encontram em `lib/required` e `lib/jpa`.

Basta adicioná-los às bibliotecas do projeto.

O mesmo para o MySQL Connector.

Configurando o Ambiente de Trabalho

O Hibernate pode ser configurado através do arquivo *hibernate.cfg.xml*.

Para conectarmos com o banco de dados precisamos definir as propriedades:

connection.driver_class

connection.url

connection.username

connection.password

dialect

Configurando o Ambiente de Trabalho

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
    <property name="hibernate.connection.driver_class"> </property>
    <property name="hibernate.connection.url"> </property>
    <property name="hibernate.connection.username"> </property>
    <property name="hibernate.connection.password"> </property>
    <property name="dialect"> </property>
  </session-factory>
</hibernate-configuration>
```


Configurando o Ambiente de Trabalho

Outras propriedades interessantes:

show_sql : *true|false*

(exibe os comandos SQL no console, a medida que o hibernate os gera)

hibernate.connection.driver_class :

Validate | create | update | create-drop

(cria ou valida um esquema do banco de dados automaticamente)

Mapeamento ORM

O mapeamento pode ser feito através de arquivos XML ou com as anotações do JPA (Java Persistence API) .

Vamos utilizar o segundo método pela facilidade de uso, padronização e manutenção.

As anotações do JPA estão no pacote *javax.persistence.**

Mapeamento ORM

É recomendado que as classes nas quais serão mapeadas as entidades sigam o padrão ***JavaBean****.

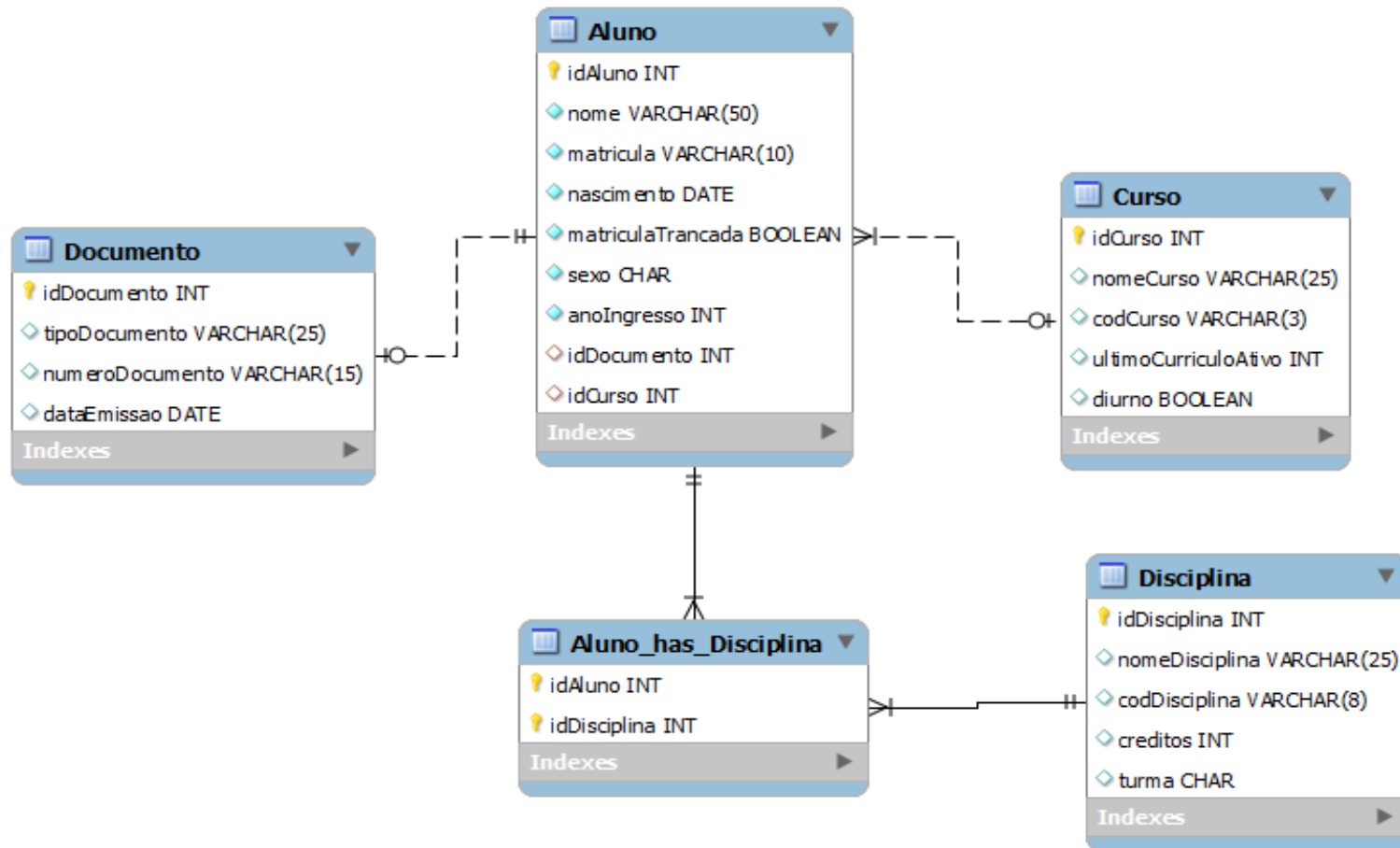
JavaBean*: classes serializadas, com construtor sem parâmetros e atributos acessíveis através de getters and setters.

**Serialização: salvar, gravar e capturar o estado de um objeto. Necessário para a persistência

Problema - Exercícios

Um sistema genérico para cadastro de alunos em uma universidade.

Problema - Exercícios



Mapeando Tabelas

@Entity

```
public class Classe implements  
    Serializable {  
    ...  
}
```

Por padrão a classe “Classe” será gravada na tabela “Classe”

Mapeando Tabelas

@Entity

@Table(name="Tabela")

```
public class Classe implements  
Serializable {  
  
    ...  
}
```

Neste caso, a classe “Classe” será persistida na tabela “Tabela”

Mapeando Tabelas

```
1 package minicurso;
2
3 import java.io.Serializable;
15
16 @Entity
17 @Table(name="Aluno")
18 public class Aluno implements Serializable {
19
20
21
22
23 }
24
```

Aluno	
idAluno	INT
nome	VARCHAR(50)
matricula	VARCHAR(10)
nascimento	DATE
matriculaTrancada	BOOLEAN
sexo	CHAR
anoIngresso	INT
Indexes	

Mapeando Tabelas

Agora, precisamos indicar no *hibernate.cfg.xml* as classes usadas no mapeamento.

Para cada classe adicionamos ao XML:

```
<mapping class= "classe" />
```

Mapeando Tabelas

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE hibernate-configuration PUBLIC
3     "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
4     "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
5 <hibernate-configuration>
6   <session-factory>
7     <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
8     <property name="hibernate.connection.url">jdbc:mysql://localhost/mydb</property>
9     <property name="hibernate.connection.username">root</property>
10    <property name="hibernate.connection.password">root</property>
11    <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
12    <property name="show_sql">true</property>
13
14    <mapping class="minicurso.Aluno" />
15    <mapping class="minicurso.Documento" />
16    <mapping class="minicurso.Disciplina" />
17    <mapping class="minicursoCurso" />
18  </session-factory>
19 </hibernate-configuration>
```

Exercício

1 - Mapear Documento, Curso e Disciplina.

Mapeando Colunas

As seguintes anotações são usadas sobre os atributos da classe a ser persistida:

@Column

@Transient

@Temporal

@Id

@GeneratedValue

Mapeando Colunas

@Column:

Por padrão, o atributo é armazenado na coluna de mesmo nome. Se quisermos configurações diferentes podemos usar essa anotação com seus suas propriedades:

name, **unique**, **nullable**, insertable, updatable, columnDefinition, table, **length**, precision, scale

Mapeando Colunas

Name: nome da coluna
(default: nome do atributo)

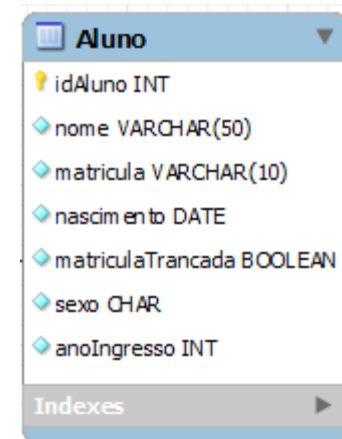
Unique: define se o valor deve ser único
(default: false)

Nullable: define se a coluna pode ser null
(default: true)

Length: define o comprimento máximo
(default: 255)

Mapeando Colunas

```
23 @Entity
24 @Table(name="Aluno")
25 public class Aluno implements Serializable {
26
27     private static final long serialVersionUID = 1L;
28
29     @Column(name="idAluno", nullable=false, unique=true)
30     private int idAluno;
31
32     @Column(name="nome", nullable=false, length=50)
33     private String nome;
34
35     @Column(name="matricula", nullable=false, length=10)
36     private String matricula;
37
38     @Column(name="nascimento", nullable=false)
39     private Calendar nascimento;
40
41     @Column(name="matriculaTrancada", nullable=false)
42     private boolean matriculaTrancada;
43
44     @Column(name="sexo", nullable=false)
45     private char sexo;
46
47     @Column(name="anoIngresso", nullable=false)
48     private int anoIngresso;
```



The screenshot shows a database schema viewer for a table named 'Aluno'. The table has the following columns:

Column Name	Data Type
idAluno	INT
nome	VARCHAR(50)
matricula	VARCHAR(10)
nascimento	DATE
matriculaTrancada	BOOLEAN
sexo	CHAR
anoIngresso	INT

Below the columns, there is a section for 'Indexes' with a right-pointing arrow.

Exercício

2 - Mapear as colunas das tabelas:
Documento, Curso e Disciplina.

Mapeando Colunas

Vamos ver mais algumas anotações:

@Transient:

Para indicar que o atributo não será persistido.

```
@Transient
```

```
private int quantidade;
```

Mapeando Colunas

@Temporal:

Para atributos que indicam tempo, podemos indicar com qual precisão vão ser gravados no banco: DATE, TIME ou TIMESTAMP.

```
@Temporal(TemporalType.DATE)
```

```
private Calendar dataFinalizacao;
```

Mapeando Colunas

@Id:

Para o identificador (chave primária) da tabela.

@Id

```
private int idAluno;
```

Mapeando Colunas

@GeneratedValue:

Para gerar automaticamente valores para a chave primária. O JPA oferece quatro estratégias: AUTO, IDENTITY, TABLE e SEQUENCE

@Id

```
@GeneratedValue(strategy=Generation.Type.TABLE);  
private int idAluno;
```

Mapeando Colunas

@GenericGenerator:

Podemos usar outras estratégias de geração do Hibernate: **increment**, identity, sequence, hilo, seqhilo, uuid, guid, native, assigned, select, foreigners, sequence-identity

Mapeando Colunas

Não entraremos em detalhes de cada uma dessas estratégias. Então, vamos usar as seguintes anotações para auto-gerar e incrementar um identificador:

```
@Id
```

```
@GeneratedValue(generator="increment")
```

```
@GenericGenerator(name = "increment",  
strategy="increment")
```

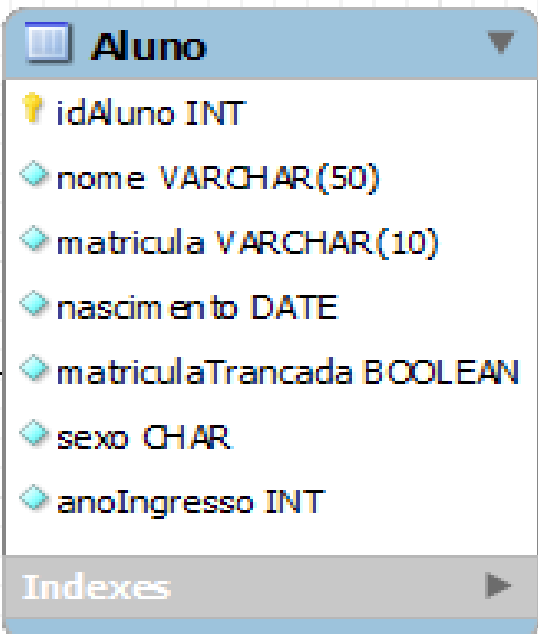
Mapeando Colunas

```
23 @Entity
24 @Table(name="Aluno")
25 public class Aluno implements Serializable {
26
27     private static final long serialVersionUID = 1L;
28
29     @Id
30     @GeneratedValue(generator="increment")
31     @GenericGenerator(name="increment", strategy="increment")
32     @Column(name="idAluno", nullable=false, unique=true)
33     private int idAluno;
34
35     @Column(name="nome", nullable=false, length=50)
36     private String nome;
37
38     @Column(name="matricula", nullable=false, length=10)
39     private String matricula;
40
41     @Temporal(TemporalType.DATE)
42     @Column(name="nascimento", nullable=false)
43     private Calendar nascimento;
44 }
```

Aluno	
idAluno	INT
nome	VARCHAR(50)
matricula	VARCHAR(10)
nascimento	DATE
matriculaTrancada	BOOLEAN
sexo	CHAR
anoIngresso	INT
Indexes	

Mapeando Colunas

```
45 @Column(name="matriculaTrancada", nullable=false)
46 private boolean matriculaTrancada;
47
48 @Column(name="sexo", nullable=false)
49 private char sexo;
50
51 @Column(name="anoIngresso", nullable=false)
52 private int anoIngresso;
```



The screenshot shows a database schema viewer for a table named 'Aluno'. The table has the following columns:

Column Name	Data Type
idAluno	INT
nome	VARCHAR(50)
matricula	VARCHAR(10)
nascimento	DATE
matriculaTrancada	BOOLEAN
sexo	CHAR
anoIngresso	INT

Below the columns, there is a section for 'Indexes'.

Exercício

3 - Reanotar os atributos de id e os do tipo Calendar com as anotações próprias.

Inserindo Dados

Para nos comunicarmos com o Banco de Dados, precisamos de uma sessão do Hibernate, representada pelo objeto Session.

Primeiramente, instanciamos uma classe que representa a configuração do Hibernate:

```
Configuration conf = new Configuration();
```

Inserindo Dados

E chamamos o método para ler o hibernate.cfg.xml:

```
conf.configure();
```

Inserindo Dados

Criamos uma Factory de sessões, um objeto que cria as sessões:

```
StandardServiceRegistryBuilder builder =  
new StandardServiceRegistryBuilder();
```

```
builder.applySettings(conf.getProperties());
```

```
SessionFactory factory =  
    conf.buildSessionFactory(builder.build());
```

Inserindo Dados

Agora conseguimos nossa Session:

```
Session session = factory.openSession();
```

Inserindo Dados

Podemos, então persistir nossos objetos:

```
Aluno = new Aluno();  
produto.setNome("Fulano");  
produto.setMatricula("201465043A");  
...
```

```
Transaction trans = session.beginTransaction();  
session.save(produto);  
trans.commit();  
Session.close();
```

Inserindo Dados

```
Configuration conf = new Configuration();
conf.configure();
StandardServiceRegistryBuilder builder = new StandardServiceRegistryBuilder();
builder.applySettings(conf.getProperties());
SessionFactory factory = conf.buildSessionFactory(builder.build());
Session session = factory.openSession();

Aluno aluno1 = new Aluno();
aluno1.setNome("João");
aluno1.setAnoIngresso(2011);
aluno1.setMatricula("201165156A");
aluno1.setMatriculaTrancada(true);
Calendar cal = Calendar.getInstance();
cal.set(1990, 01, 01);
aluno1.setNascimento(cal);
aluno1.setSexo('M');

Transaction trans = session.beginTransaction();
session.save(aluno1);
trans.commit();
session.close();
```

Exercício

4 - Insira um registro em cada uma das tabelas do banco de dados.

Criteria Queries

A Criteria API nos permite criar consultas em Java, independente de linguagens de consulta.

Fazemos uma query de uma classe específica com um objeto Criteria, que conseguimos através do Session (visto anteriormente):

```
Criteria criteria = session.createCriteria(Classe.class)
```

Criteria Queries

Podemos retornar uma lista de resultado

```
criteria.list();
```

Retorno: List

Criteria Queries

```
public static void main(String args[]){

    Configuration conf = new Configuration();
    conf.configure();
    StandardServiceRegistryBuilder builder = new StandardServiceRegistryBuilder();
    builder.applySettings(conf.getProperties());
    SessionFactory factory = conf.buildSessionFactory(builder.build());
    Session session = factory.openSession();

    Criteria criteria = session.createCriteria(Aluno.class);
    ArrayList<Aluno> alunos = new ArrayList<Aluno>();
    alunos = (ArrayList<Aluno>) criteria.list();
    for(Aluno aluno : alunos){
        System.out.println(aluno.getNome());
    }
}
```

Exercício

5 - Consultar os registros da tabela Curso e imprimir os resultados na tela.

Criteria Queries

Até então, estávamos consultando todos objetos persistidos de uma determinada classe.

Porém, podemos adicionar restrições com o método `add` e a classe `Restrictions`:

```
criteria.add(Restrictions.eq("nomedoatributo", valor));
```

Cada método da classe `Restriction` faz um tipo de restrição a nossa consulta. Vamos ver alguns:

Criteria Queries

eq(String nomedoatributo, Object valor) :

Restringe o resultado a objetos que tem o atributo igual ao valor.

Outros: ne()(não igual), ge()(maior ou igual),
gt() (maior que), le()(menor ou igual), lt()
(menor que)

Criteria Queries

isNull(String nomedoatributo)

Restringe o resultado a objetos que tem o atributo com valor null.

Outros: isNotNull()

idEq(Object valor)

Restringe o resultado a objetos que tem o identificador igual ao valor.

Criteria Queries

Alguns outros métodos para melhorar nossas consultas:

```
criteria.setMaxResults(5);
```

Define um valor inteiro como máximo de resultados de nossa consulta.

Criteria Queries

```
criteria.addOrder(Order.asc("atributo"));
```

```
criteria.addOrder(Order.desc("atributo"));
```

Ordena os resultados da consulta de acordo com o atributo especificado em ordem crescente ou decrescente.

Criteria Queries

Classe resultado = criteria.uniqueResult();

Utilizado para retornar apenas um resultado na consulta.

*Se houver mais de um resultado, o método irá lançar uma exceção.

Exercício

6 - Realizar as seguintes consultas e imprimir o resultado:

a) Todos os cursos noturnos.

b) O aluno de nome Maria, que ingressou em 2010 e que trancou o curso.

c) Todas as disciplinas de 2 créditos, ordenadas pelo nome.

HQL

Hibernate Query Language é uma linguagem de consultas, com a sintaxe bem parecida com SQL, porém voltado à orientação a objetos.

HQL

Para fazermos uma consulta com HQL:

```
Query query = session.createQuery("Expressão HQL");
```

HQL

Vamos ver como montar expressões HQL.

FROM: determina quais Classes, cujos objetos serão retornadas na consulta

Ex: *"FROM Aluno"*

Retorna todos Alunos armazenados no banco.

HQL

SELECT: especifica atributos da Classe a serem retornados na consulta.

Ex: *“SELECT a.nome FROM Aluno AS a”*

Retorna todos os valores de nome em todas instancias de Aluno.

HQL

AS: define um alias para a Classe, como ela deverá ser chamada no resto da consulta. (Pode ser omitida)

Ex: *“FROM Aluno AS a”*

HQL

SELECT: especifica atributos da Classe a serem retornados na consulta.

Ex: *“SELECT a.nome FROM Aluno AS a”*

Retorna todos os valores de nome em todas instancias de Aluno.

HQL

WHERE: define uma condição para a consulta.

Ex: *“FROM Aluno as a WHERE a.nome = João”*
Retorna todos os alunos com o nome João.

*As condições podem conter: operadores matemáticos, operadores de comparação, operadores lógicos, parênteses, dentre vários outros.

HQL

Podemos usar, também, outros parâmetros:

```
Query query =getSession().createQuery  
("FROM Aluno AS a WHERE a.nome = :nome);  
query.setParameter("nome",aluno1.getNome());
```

Retorna todos os alunos com nome igual ao do aluno1.

HQL

ORDER BY: ordena o resultado da consulta segundo um atributo de maneira crescente (ASC) ou decrescente (DESC).

Ex: *"FROM Aluno as a ORDER BY a.nome ASC"*

Retorna todos alunos ordenados pelos nomes em ordem crescente.

HQL

Para obtermos os resultados:

```
query.list;
```

Retorno:

Object caso seja selecionado um tipo.

Object[] caso seja selecionado tipos diferentes.

Para obtermos os resultados:

```
query.uniqueResult();
```

Retorno:

`List<Object>` caso seja selecionado um tipo.

`List<Object[]>` caso seja selecionado tipos diferentes.

HQL

```
public static void main(String args[]){

    Configuration conf = new Configuration();
    conf.configure();
    StandardServiceRegistryBuilder builder = new StandardServiceRegistryBuilder();
    builder.applySettings(conf.getProperties());
    SessionFactory factory = conf.buildSessionFactory(builder.build());
    Session session = factory.openSession();

    Aluno aluno1 = new Aluno();
    aluno1.setNome("João");

    Query query = session.createQuery("FROM Aluno AS a WHERE a.nome = :nome and a.anoIngresso > 2009 ORDER BY anoIngresso");
    query.setParameter("nome",aluno1.getNome());
}
```

HQL

```
public static void main(String args[]){  
  
    Configuration conf = new Configuration();  
    conf.configure();  
    StandardServiceRegistryBuilder builder = new StandardServiceRegistryBuilder();  
    builder.applySettings(conf.getProperties());  
    SessionFactory factory = conf.buildSessionFactory(builder.build());  
    Session session = factory.openSession();  
  
    Query query = session.createQuery("FROM Aluno AS a WHERE a.nome = 'Maria' and a.anoIngresso = 2010 and a.matriculaTrancada = true");  
    Aluno resultado = (Aluno) query.uniqueResult();  
    System.out.println(resultado.getNome());  
  
}
```


HQL

```
public static void main(String args[]){

    Configuration conf = new Configuration();
    conf.configure();
    StandardServiceRegistryBuilder builder = new StandardServiceRegistryBuilder();
    builder.applySettings(conf.getProperties());
    SessionFactory factory = conf.buildSessionFactory(builder.build());
    Session session = factory.openSession();

    Query query = session.createQuery("SELECT d.codDisciplina, d.nomeDisciplina FROM Disciplina AS d");
    List<Object[]> resultados = query.list();
    for(Object[] tupla : resultados){
        System.out.println(tupla[0] + "-" + tupla[1]);
    }
}
```

Exercício

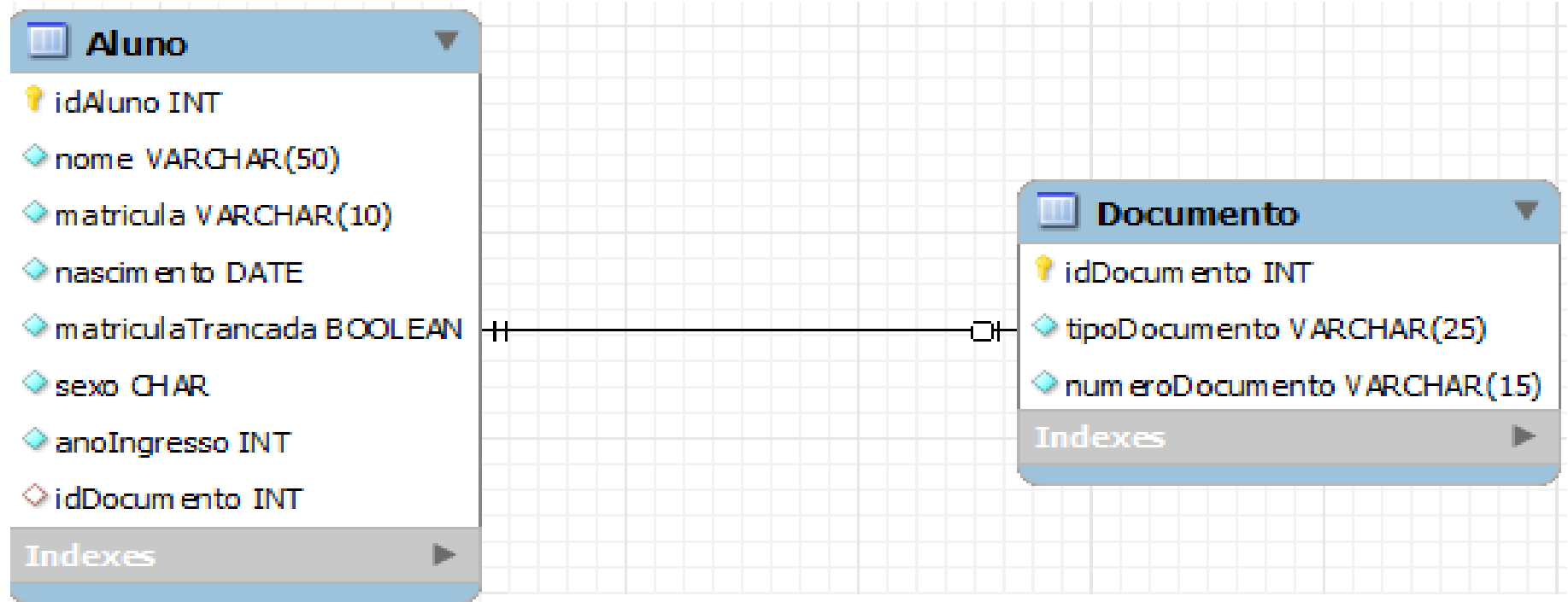
7 - Realizar as seguintes consultas e imprimir o resultado:

b) O nome e o turno do curso de código 65A.

b) O nome e o código das disciplinas de 4 créditos.

c) Todos alunos que ingressaram antes de 2011, ordenados pelo nome.

Mapeando Relacionamentos 1:1

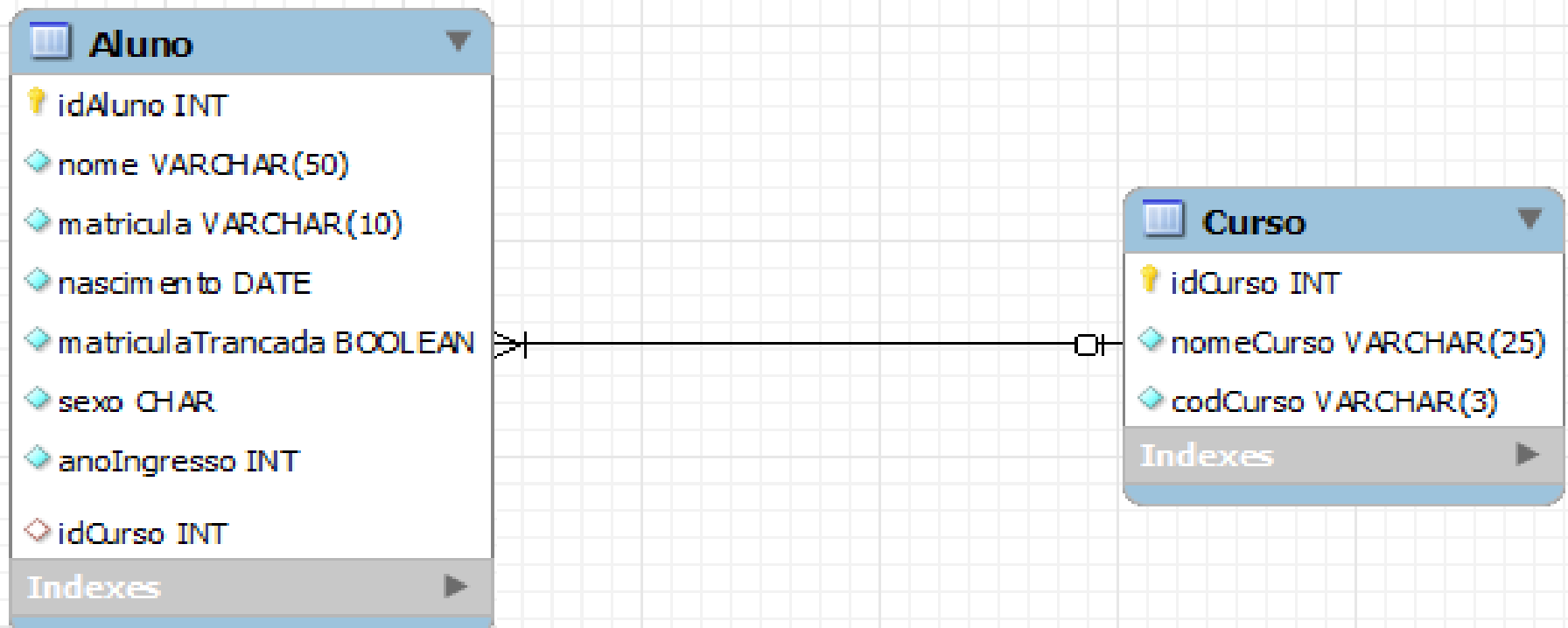


Mapeando Relacionamentos 1:1

```
@Entity
@Table (name="Aluno")
public class Aluno {
    ...
    @OneToOne
    @JoinColumn(name="idDocumento")
    private Documento documento;
}
```

```
@Entity
@Table(name = "Documento")
public class Documento {
    ...
    @OneToOne(mappedBy="documento")
    private Aluno aluno;
}
```

Mapeando Relacionamentos 1:N / N:1



Mapeando Relacionamentos 1:N / N:1

@Entity

@Table (name="Curso")

public class Curso {

...

@OneToMany(fetch = FetchType.LAZY,
mappedBy="curso")

private List<Aluno> alunos;

}

@Entity

@Table(name = "Aluno")

public class Aluno {

...

@ManyToOne

@JoinColumn(name="idCurso")

private Curso curso;

}

Mapeando Relacionamentos 1:N / N:1

Fetch Types:

- Eager: os dados são carregados automaticamente.
- Lazy: os dados só são carregados quando é feito o primeiro acesso.

Mapeando Relacionamentos 1:N / N:1

```
Query query = session.createQuery("FROM Curso AS c WHERE c.nomeCurso = 'Ciencia da computacao'");
System.out.println("query");
Curso curso = (Curso) query.uniqueResult();
System.out.println("get");
List<Aluno> alunos = curso.getAlunos();
```

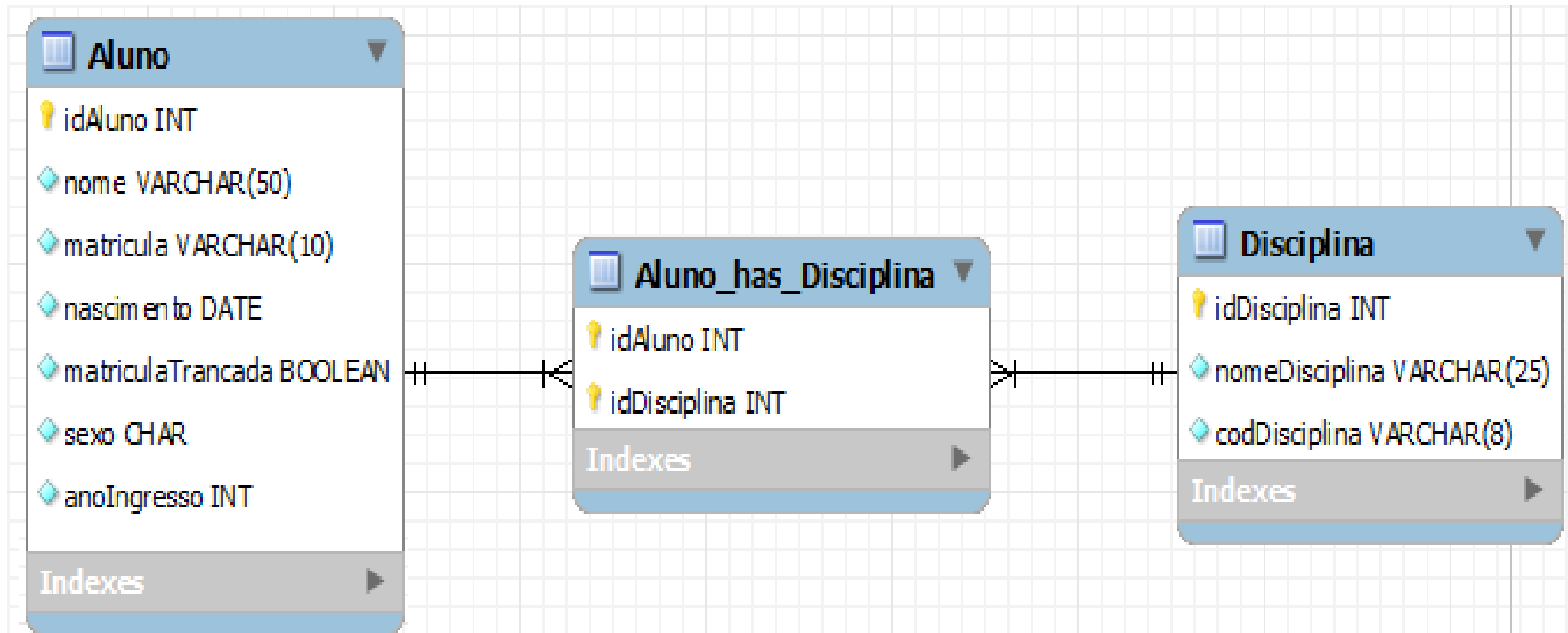
Lazy:

```
query
Hibernate: select curso0_.idCurso as idCurso1_2_, curso0_.codCurso as codCurso2_
get
Hibernate: select alunos0_.idCurso as idCurso8_2_0_, alunos0_.idAluno as idAlunc
```

Eager:

```
query
Hibernate: select curso0_.idCurso as idCurso1_2_, curso0_.codCurso as codCurso2_
Hibernate: select alunos0_.idCurso as idCurso8_2_0_, alunos0_.idAluno as idAlunc
get
```


Mapeando Relacionamentos N:N



Mapeando Relacionamentos N:N

@Entity

@Table (name="Aluno")

public class Aluno {

...

@ManyToMany(fetch = FetchType.LAZY)

@JoinTable(name="Aluno_Disciplina",

joinColumns=

@JoinColumn(name="idAluno"),

inverseJoinColumns=

@JoinColumn(name="idDisciplina"))

private List<Disciplina> disciplinas;

}

@Entity

@Table(name = "Disciplina")

public class Disciplina {

...

@ManyToMany(fetch = FetchType.LAZY,
mappedBy="disciplinas")

private List<Aluno> alunos;

}

Exercícios

8 - Mapeie os relacionamentos entre :

a) Aluno e Documento

b) Aluno e Curso

c) Aluno e Disciplina.

Consultas - Relacionamentos

Podemos restringir nossa consulta com o valor de uma coluna de uma tabela associada.

Ex: Queremos os alunos que estão matriculados na disciplina Algoritmos

Consultas - Relacionamentos

Criteria

```
Criteria criteria =  
    session.createCriteria(Aluno.class);  
criteria.createCriteria("disciplinas").add(Restrictions.  
    eq("nomeDisciplina", "Algoritmos"));
```

Consultas - Relacionamentos

HQL:

Join Explícito:

```
Query query = session.createQuery("Select  
    a.nome FROM Aluno AS a INNER JOIN  
    a.disciplinas AS d WHERE d.nomeDisciplina =  
    'Algoritmos'");
```

Consultas - Relacionamentos

HQL:

Join Implícito:

```
Query query = session.createQuery("FROM  
    Aluno AS a WHERE a.curso.nome = 'Ciencias  
    Exatas'");
```

Exercício

9 - Realize as consultas usando Criteria ou HQL:

- a) O aluno de documento de número 16253645.
- b) Todos os alunos do curso de Ciência da Computação (diurno e noturno).
- c) Os alunos da turma A de Algoritmos que entraram no ano de 2011 ou depois.

Update / Delete

Agora que sabemos fazer consultas, podemos atualizar ou excluir um objeto:

```
session.update(objeto);
```

```
session.delete(objeto);
```

Update / Delete

```
public static void main(String args[]){

    Configuration conf = new Configuration();
    conf.configure();
    StandardServiceRegistryBuilder builder = new StandardServiceRegistryBuilder();
    builder.applySettings(conf.getProperties());
    SessionFactory factory = conf.buildSessionFactory(builder.build());
    Session session = factory.openSession();

    Criteria criteria = session.createCriteria(Aluno.class);
    criteria.add(Restrictions.eq("matricula", "201465043A"));
    Aluno aluno = (Aluno) criteria.uniqueResult();

    aluno.setNome("João");

    Transaction trans = session.beginTransaction();
    session.update(aluno);
    trans.commit();

}
```

Update / Delete

```
public static void main(String args[]){  
  
    Configuration conf = new Configuration();  
    conf.configure();  
    StandardServiceRegistryBuilder builder = new StandardServiceRegistryBuilder();  
    builder.applySettings(conf.getProperties());  
    SessionFactory factory = conf.buildSessionFactory(builder.build());  
    Session session = factory.openSession();  
  
    Criteria criteria = session.createCriteria(Aluno.class);  
    criteria.add(Restrictions.eq("matricula", "201465043A"));  
    Aluno aluno = (Aluno) criteria.uniqueResult();  
  
    Transaction trans = session.beginTransaction();  
    session.delete(aluno);  
    trans.commit();  
  
}
```

Mais algumas coisas interessantes

Funções de agregação SQL

Mapeamento de Herança

Consultas Polimorficas

SQL Nativo

Mais dúvidas:

thiago.rizuti@ice.ufjf.br

<http://hibernate.org/orm/>