Laboratório de Programação 3
Parte II

3 - Hibernate

Maj Mello

SE/8 – IME

cgmello@de9.ime.eb.br

http://www.des.ime.eb.br/~cgmello

Hibernate

- Framework de persistência Java
- Criado em 2001 como um projeto do SourceForge
- Em 2003, o Hibernate 2.0 se torna a solução de persistência java mais popular do mercado
- Atualmente na versão 3.0
- Seus recursos serviram de especificação para o EJB 3.0

Quando usar

- **Aplicação modelada orientada a objetos**
- **Máximo proveito de design patterns e outros frameworks OO**
- **Aplicação contém lógica sofisticada de negócios, e não apenas operações simples CRUD (create, retrieve, update and delete)**
- **Otimização do acesso ao banco de dados**
- **Independência do banco de dados**

Persistência de objetos x BD relacionais

- **Persistência Objeto-Relacional**
  - **unir as vantagens do modelo orientado a objetos para a construção de uma aplicação, com a performance e confiabilidade dos BD relacionais**
  - **no desenvolvimento de software, a análise, design e programação OO já é um modelo maduro**
  - **mas para o armazenamento e recuperação de informações, ainda não há nada que supere os BD relacionais**
  - **BD OO evoluem lentamente**
  - **outras propostas**
    - **persistência em XML**

Frameworks de persistência

- **Componentes principais**

1 - Descrição do mapeamento entre o modelo de objetos em memória e as tabelas no BD relacional

2 - API de acesso para armazenamento e recuperação de objetos persistentes

3 - Linguagem de consulta

Mapeamento Objeto-Relacional

- **Nem sempre é possível automatizar o mapeamento O-R**
- **O ideal é construir o modelo OO e depois fazer o mapeamento**
- **Ás vezes é necessário criar um modelo OO a partir de um BD relacional (ex: sistemas legados)**
- **Mapeamento é feito através de documentos XML com extensão default *.hbm.xml***
- **É possível mapear várias classes em um único documento XML, mas recomenda-se que cada classe tenha seu próprio arquivo XML de mapeamento para facilitar a manutenção e alterações no modelo**

Javabeans

- Hibernate trabalha com objetos simples
  - **POJO - Plain Old Java Object**
- Entretanto, recomenda-se o padrão Javabeans para definição dos objetos persistentes
  - **atributos private**
  - **métodos get/set para acesso aos atributos**
  - **construtor sem argumentos**
  - **classe serializável**
- Não é necessário estender nenhuma classe do Hibernate

Exemplo: cadastro simples

/lib

- antlr.jar
- cglib.jar
- asm.jar
- asm-attrs.jars
- commons-collections.jar
- commons-logging.jar
- hibernate3.jar
- jta.jar
- dom4j.jar
- log4j.jar

Classe Contato

```
package mapeamentoor;

import java.io.Serializable;

public class Contato implements Serializable {

    private Long id;
    private String nome;
    private String email;

    public Contato() {}

    // ... métodos de acesso getXXX / set XXX
```

Mapeamento Contato.hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-
3.0.dtd">

<hibernate-mapping>

    <class name="mapeamentoor.Contato" table="contato">
        <id name="id">
            <column name="id" not-null="true"/>
            <generator class="identity"/>
        </id>
        <property name="nome">
            <column name="nome" length="40"
                not-null="true"/>
        </property>

        <property name="email">
            <column name="email" length="30"
                not-null="true"/>
        </property>

    </class>

</hibernate-mapping>
```

Mapeamento

- Que colunas e tabelas serão utilizadas para cada atributo
- Atributos não mapeados não serão persistidos
- O Hibernate infere o tipo do atributo, mas também é possível informá-lo no mapeamento
- É necessário definir um identificador (id) para a classe persistente, que será a chave primária (PK) no banco de dados
- Especificar um gerador de chaves

Generators

- ***increment*** **- id para os tipos long, short e int; não use em clusters;**
- ***identity*** **- identity columns para DB2, MySQL, MS SQL Server, Sybase, HSQL**
- ***sequence*** **- sequence para DB2, PostgreSQL, Oracle, SAP, Interbase**
- **outros**
  - ***hilo***, ***seqhilo***, ***uuid*** **(tipo string, usa o IP),** ***guid*** **(tipo string, MS SQL Server e MySQL),** ***native***

Tabela no BD

- **O Hibernate pode gerar a tabela, mas em geral isso é uma atribuição do DBA**

```
create table CONTATO (
    ID identity primary key,
    NOME varchar(40) not null,
    EMAIL varchar(30) not null
)
```

Listando os dados (1/2)

```
package mapeamentoor;

import java.util.*;
import org.hibernate.*;
import org.hibernate.cfg.*;

public class Lista {

    public static void main(String[] args) {

        SessionFactory sessionFactory =
           new Configuration().configure()
               .buildSessionFactory();

        Session session = sessionFactory.openSession();

// continua...
```

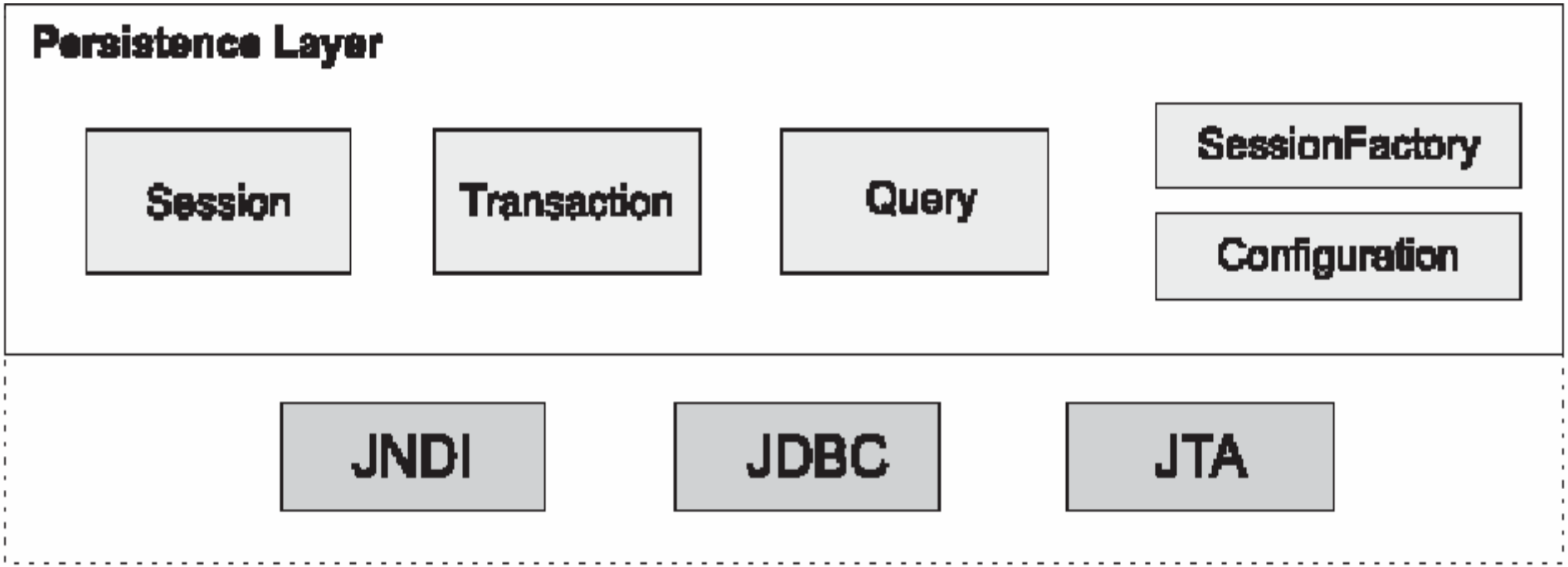
Listando os dados (2/2)

```
// ...continuação

        Transaction tx = session.beginTransaction();

        Query query = session.createQuery(
            "select c from Contato as c");
        Iterator it = query.iterate();
        while (it.hasNext()) {
            Contato contato = (Contato) it.next();
            System.out.println("[" + contato.getId() + "] "
                + contato.getNome()
                + " <" + contato.getEmail() + ">");
        }

        tx.commit();
        session.close();
    }
}
```

API do Hibernate

- *Configuration* - representa o arquivo de configuração
  - default é *hibernate.cfg.xml*
- *sessionFactory* - obtém a conexão com o BD
  - JDBC
  - JNDI - Java Naming and Directory Interface
- *Session* - gerencia a recuperação e atualização dos objetos persistentes (não é a session do servlet!)
- *Transaction* - representa a transação com o BD
  - JTA - Java Transaction API
- *Query* - representa a consulta
  - HQL - Hibernate Query Language

API do Hibernate



Singleton 1/2

```
public class TransacaoAplicacao {

    private static SessionFactory fabrica = null;

    private static ThreadLocal sessao = new ThreadLocal();
    private static ThreadLocal transacao = new ThreadLocal();

    public static Session getSessao() {
        try {
            if (fabrica == null)
                fabrica = new Configuration().configure().buildSessionFactory();
            if (sessao.get() == null)
                iniciaTransacao();
        }
        catch (Throwable e) {
            e.printStackTrace();
        }
        return (Session)sessao.get();
    }

    public static void iniciaTransacao() {
        sessao.set(fabrica.openSession());
        transacao.set(getSessao().beginTransaction());
    }
```

Singleton 2/2

```
public static void fechaSessao() {
    try {
        if (getSessao() != null)
            getSessao().close();
    }
    finally {
        sessao.set(null);
        transacao.set(null);
    }
}

public static void confirma() {
    try {
        if (getTransacao() != null)
            getTransacao().commit();
    }
    finally {
        fechaSessao();
    }
}

public static void aborta() {
    try {
        if (getTransacao() != null)
            getTransacao().rollback();
    }
    finally {
        fechaSessao();
    }
}
```

Usando o singleton

```
Session ses = TransacaoAplicacao.getSessao();
Query query = ses.createQuery(”...");
List resultado = query.list();
TransacaoAplicacao.confirma();
```

HQL - a linguagem de consulta do Hibernate

- Sintaxe semelhante ao SQL
- Faz referência a classes persistentes, não a tabelas
- Inclui ORDER BY, GROUP BY e HAVING
- Consultas parametrizadas
- O HQL é transformado em comandos SQL nativos do BD
- Pode-se habilitar o log para se visualizar os comandos SQL gerados e facilitar o debug

Consultas HQL parametrizadas

```
// ... igual ao exemplo Lista.java

        Query query = session.createQuery(
            "select c from Contato as c " +
            "where lcase(c.nome) like (:nome) " +
            "order by nome asc");
        query.setString("nome",
            "%" + args[0].toLowerCase() + "%");
        Iterator it = query.iterate();

// ... igual ao exemplo Lista.java
```

Retorno das consultas HQL

- As consultas HQL retornam
  - instâncias de objetos persistentes
  - coleções de instâncias de objetos persistentes
- Percorre-se a coleção via *Iterate* normal do Java
- A cláusula SELECT é opcional

Preparando o ambiente

- Já temos
  - classe persistente
  - seu mapeamento
  - uma aplicação que usa a classe persistente
- Preparando o ambiente
  - configuração do Hibernate
  - configuração do CLASSPATH
  - inicialização do BD
  - configuração do Log4j (opcional)

Configuração do Hibernate 1/2 (*hibernate.cfg.xml*)

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-
configuration-3.0.dtd">

<hibernate-configuration>

    <session-factory>
        <property name="connection.driver_class">
            org.hsqldb.jdbcDriver</property>
        <property name="connection.url">
             jdbc:hsqldb:contatos</property>

<!-- continua... -->
```

Configuração do Hibernate 2/2 (arquivo *hibernate.cfg.xml*)

```
<!-- ...continuação -->

        <property name="connection.username">
            sa</property>
        <property name="connection.password"></property>

        <property name="connection.pool_size">1</property>
        <property name="dialect">
            org.hibernate.dialect.HSQLDialect</property>
        <property name="show_sql">false</property>

        <mapping resource="mapeamentoor/Contato.hbm.xml"/>
    </session-factory>

</hibernate-configuration>
```

Configuração do Hibernate

- Parâmetros de conexão ao BD (JDBC ou JNDI)
  - driver, url, conta, senha
- Em servidores de aplicação JEE é melhor usar o JNDI do que o pool de conexões interno do Hibernate
- Dialeto HQL
- Opção de logar o comando SQL gerado
- Lista de arquivos de mapeamento

Usando JNDI

- **JNDI - Java Naming and Directory Interface**
- **É um datasource gerenciado pelo container web**
- **Configurar o datasource no arquivo de configuração *context.xml* do container**
- **Fazer referência ao datasource no *hibernate.cfg.xml***

Arquivo *context.xml* do container

```
<Context path="/minhaaplicacao"
    docBase="/home/jakarta-tomcat-5.0.28/webapps/minhaaplicacao"
    reloadable="true" debug="1">

  <Resource name="jdbc/mypool" type="javax.sql.DataSource" />

  <ResourceParams name="jdbc/mypool">
    <parameter>
     <name>username</name>
     <value>sa</value>
    </parameter>
    <parameter>
     <name>password</name>
     <value></value>
    </parameter>
    <parameter>
       <name>driverClassName</name>
       <value>org.hsqldb.jdbcDriver</value>
    </parameter>
    <parameter>
      <name>url</name>
      <value>jdbc:hsqldb:file:/home/minhaaplicacao;shutdown=true</value>
    </parameter>
  </ResourceParams>

</Context>
```

Usando o datasource via JNDI (arquivo *hibernate.cfg.xml*)

- **Alteração**

  <property name="connection.datasource">

  java:/comp/env/jdbc/mypool

  </property>

- **Substitui as declarações de**
  - **driver, url, conta, senha**

HSQLDB

- Banco de dados 100% java
- Vários modos de operação
  - em memória, stand-alone, server, outros
- Um único JAR
  - hsqldb.jar
    - servidor
    - driver JDBC
    - utilitários de administração
- Pode ser distribuído com a aplicação

Configuração do CLASSPATH

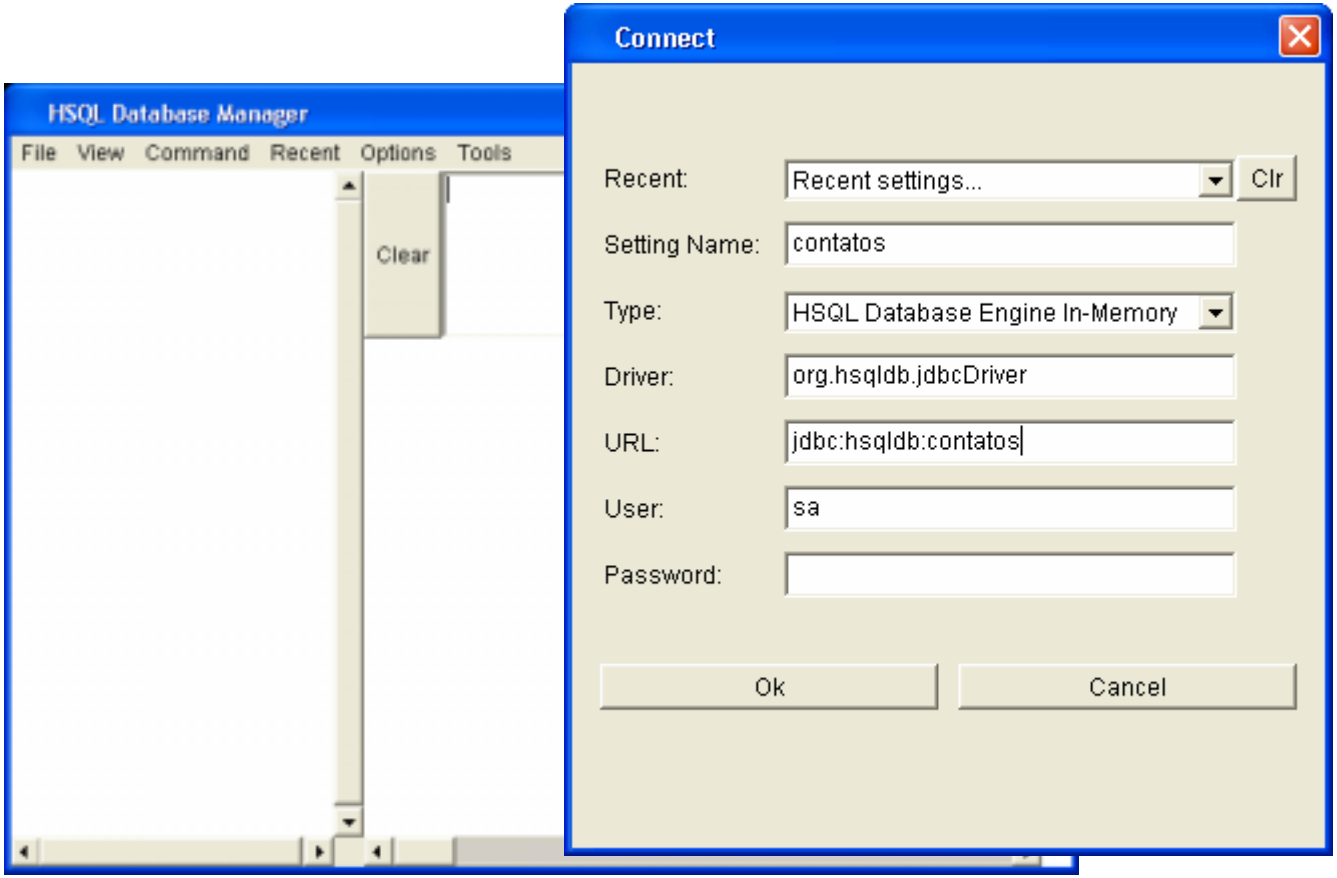
- **Inclua no CLASSPATH**
  - **rem mude para adequar ao seu ambiente**
  - **set HSQLDB_HOME=C:\hsqldb**
  - **set HIBERNATE_HOME=C:\hibernate-3.1**

  - **rem classes obrigatorias do Hibernate**
  - **set CLASSPATH=%CLASSPATH%;%HSQLDB_HOME%/lib/hsqldb.jar**
  - **set CLASSPATH=%CLASSPATH%;%HIBERNATE_HOME%/hibernate3.jar**
  - **...**
  - **set CLASSPATH=%CLASSPATH%;%HIBERNATE_HOME%/lib/commons-collections-2.1.1.jar**

  - **rem classes da aplicacao**
  - **set CLASSPATH=%CLASSPATH%;classes;.**

Inicialização do BD

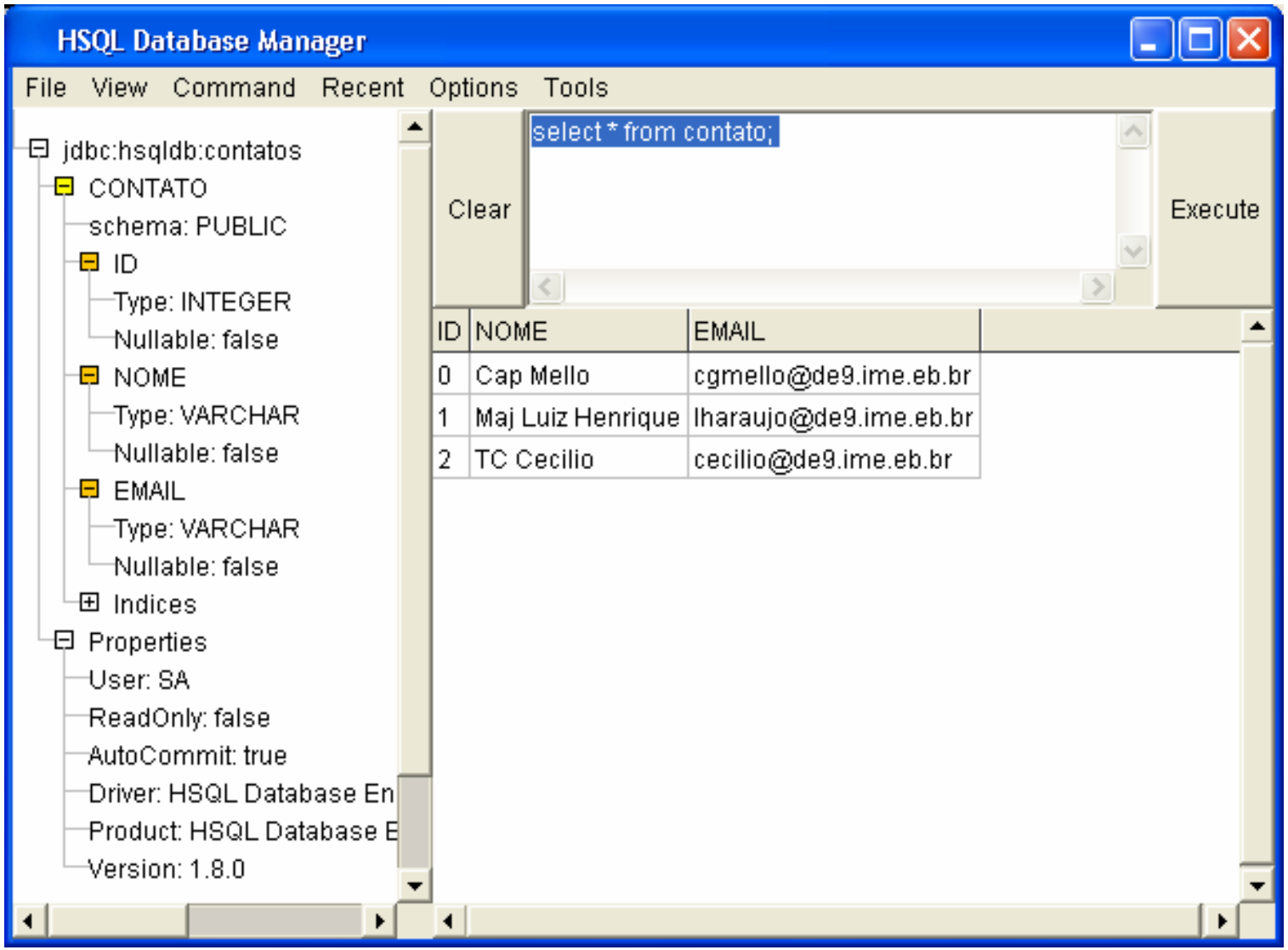
- **Database Manager**
  - ***java org.hsqldb.util.DatabaseManager***



Inicialização do BD

```
create table CONTATO (
  ID identity primary key,
  NOME varchar(40) not null,
  EMAIL varchar(30) not null
);

insert into CONTATO (nome, email) values ('Cap Mello',
    'cgmello@de9.ime.eb.br');
insert into CONTATO (nome, email) values ('Maj Luiz Henrique',
    'lharaujo@de9.ime.eb.br');
insert into CONTATO (nome, email) values ('TC Cecilio',
    'cecilio@de9.ime.eb.br');
```

Inicialização do BD



Inicialização do BD

- **Database Manager**
  - **Inicializando a partir de uma base já criada**
    - ***java org.hsqldb.util.DatabaseManager -url 'jdbc:hsqldb:contatos;shutdown=true'***

Configurando o Log4j (arquivo *log4j.properties*)

```
log4j.appender.log=org.apache.log4j.FileAppender
log4j.appender.log.File=hibernate.log
log4j.appender.log.layout=org.apache.log4j.PatternLayout
log4j.appender.log.layout.ConversionPattern=
%d{ABSOLUTE} %5p %c{1}:%L - %m%n
log4j.rootLogger=error, log
log4j.logger.org.hibernate=info
```

Log gerado pelo Log4j (arquivo *hibernate.log*)

```
1  04:49:04,191  INFO Environment:464 - Hibernate 3.0.5
2  04:49:04,200  INFO Environment:477 - hibernate.properties not found
3  04:49:04,206  INFO Environment:510 - using CGLIB reflection optimizer
4  04:49:04,213  INFO Environment:540 - using JDK 1.4 java.sql.Timestamp handling
5  04:49:04,418  INFO Configuration:1110 - configuring from resource: /hibernate.cfg.xml
6  04:49:04,419  INFO Configuration:1081 - Configuration resource: /hibernate.cfg.xml
7  04:49:05,060 DEBUG DTDEntityResolver:42 - trying to locate
   http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd in classpath under org/hibernate/
8  04:49:05,063 DEBUG DTDEntityResolver:53 - found
   http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd in classpath
9  04:49:05,183 DEBUG Configuration:1067 - connection.driver_class=org.hsqldb.jdbcDriver
10 04:49:05,184 DEBUG Configuration:1067 - connection.url=jdbc:hsqldb:contatos
11 04:49:05,185 DEBUG Configuration:1067 - connection.username=sa
12 04:49:05,186 DEBUG Configuration:1067 - connection.password=
13 04:49:05,186 DEBUG Configuration:1067 - connection.pool_size=1
14 04:49:05,197 DEBUG Configuration:1067 - dialect=org.hibernate.dialect.HSQLDialect
15 04:49:05,198 DEBUG Configuration:1067 - show_sql=true
16 04:49:05,199 DEBUG Configuration:1262 - null<-org.dom4j.tree.DefaultAttribute@21b6d [Attribute: name
   resource value "mapeamentoor/Contato.hbm.xml"]
17 04:49:05,200  INFO Configuration:444 - Mapping resource: mapeamentoor/Contato.hbm.xml
18 04:49:05,204 DEBUG DTDEntityResolver:42 - trying to locate
   http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd in classpath under org/hibernate/
19 04:49:05,206 DEBUG DTDEntityResolver:53 - found
   http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd in classpath
20 04:49:05,508  INFO HbmBinder:260 - Mapping class: mapeamentoor.Contato -> contato
21 04:49:05,551 DEBUG HbmBinder:1099 - Mapped property: id -> id
22 04:49:05,578 DEBUG HbmBinder:1099 - Mapped property: nome -> nome
23 04:49:05,579 DEBUG HbmBinder:1099 - Mapped property: email -> email
24 04:49:05,581 DEBUG Configuration:1262 - null<-org.dom4j.tree.DefaultAttribute@121cc40 [Attribute: name
   resource value "mapeamentoor/Contato2.hbm.xml"]
```

Compilando e executando

>classpath.bat

>javac mapeamentoor/.java*

>java mapeamentoor.Lista

```
>java mapeamentoor.Lista
Hibernate: select contato0_.id as id0_, contato0_.nome as nome0_,
contato0_.email as email0_ from contato contato0_
[0] Cap Mello <cgmello@de9.ime.eb.br>
[1] Maj Luiz Henrique <lharaujo@de9.ime.eb.br>
[2] TC Cecilio <cecilio@de9.ime.eb.br>
```

Criando as tabelas via Hibernate

- **O pacote do Hibernate já inclui uma ferramenta para a criação das tabelas mapeadas**
  - ***import org.hibernate.tool.hbm2ddl.****
  - ***SchemaExport schemaExport = new SchemaExport(new Configuration().configure());***
  - ***schemaExport.create(true, true);***
    - **script - print the DDL to the console**
    - **export - export the script to the database**

```
drop table contato if exists
create table contato (id bigint generated by default as identity
(start with 1), nome varchar(40) not null, email varchar(30) not
null, primary key (id))
```

Outras operações

- **Inserindo um objeto persistente**
- **Alterando**
- **Excluindo**

Inserindo um objeto persistente

- **Uso do método *save()* da *Session***
- **Hibernate gera a cláusula *INSERT***
- **Hibernate decide quando inserir realmente, geralmente só na confirmação da transação**

Inserindo um objeto persistente 1/2

```
package mapeamentoor;

import org.hibernate.*;
import org.hibernate.cfg.*;

public class Cria {

      public static void main(String[] args) {
         if (args.length != 2) {
             System.err.println(
                 "Uso: java Cria <nome> <email>");
             System.exit(1);
         }

         SessionFactory sessionFactory =
             new Configuration().configure()
                 .buildSessionFactory();
// continua...
```

Inserindo um objeto persistente 2/2

```
// ...continuação

        Session session = sessionFactory.openSession();

        Transaction tx = session.beginTransaction();

        Contato novo = new Contato();
        novo.setNome(args[0]);
        novo.setEmail(args[1]);
        session.save(novo);

        tx.commit();

        session.close();
    }
}
```

Inserindo um objeto persistente

[0] Cap Mello <cgmello@de9.ime.eb.br>

[1] Maj Luiz Henrique <lharaujo@de9.ime.eb.br>

[2] TC Cecilio <cecilio@de9.ime.eb.br>

>java mapeamentoor.Cria Teste1 teste1@gmail.com

Hibernate: insert into contato (nome, email, id) values (?, ?, null)

Hibernate: call identity()

/* chama Lista.main(null); para listar */

Hibernate: select contato0_.id as id2_, contato0_.nome as nome2_,
contato0_.email as email2_ from contato contato0_

[0] Cap Mello <cgmello@de9.ime.eb.br>

[1] Maj Luiz Henrique <lharaujo@de9.ime.eb.br>

[2] TC Cecilio <cecilio@de9.ime.eb.br>

[3] Teste1 <teste1@gmail.com>

Alterando um objeto persistente

- **Caso o objeto tenha sido recuperado via Hibernate, ao final da transação ele será verificado, e o Hibernate automaticamente irá atualizá-lo no BD caso tenha havido alguma alteração em seus atributos**
- **Não é necessário chamar nenhum comando explícito para a atualização no BD**
- **O Hibernate gera a cláusula *UPDATE***

Alterando um objeto persistente

```
//... imports omitidos

public class Altera
{
    public static void main(String[] args) {

        //... obter o objeto Session

        Transaction tx = session.beginTransaction();
        Contato contato = (Contato)session.get
            (Contato.class, Long.valueOf(args[0]));
        contato.setNome(args[1]);
        contato.setEmail(args[2]);
        tx.commit();

        //... libera a conexão
    }
}
```

Alterando um objeto persistente

[0] Cap Mello <cgmello@de9.ime.eb.br>

[1] Maj Luiz Henrique <lharaujo@de9.ime.eb.br>

[2] TC Cecilio <cecilio@de9.ime.eb.br>

>java mapeamentoor.Altera 0 "Claudio Mello" cgmello@de9.ime.eb.br

Hibernate: select contato0_.id as id0_0_, contato0_.nome as nome0_0_,
contato0_.email as email0_0_ from contato contato0_ where contato0_.id=?

Hibernate: update contato set nome=?, email=? where id=?

/* chama Lista.main(null); para listar */

Hibernate: select contato0_.id as id2_, contato0_.nome as nome2_,
contato0_.email as email2_ from contato contato0_

[0] Claudio Mello <cgmello@de9.ime.eb.br>

[1] Maj Luiz Henrique <lharaujo@de9.ime.eb.br>

[2] TC Cecilio <cecilio@de9.ime.eb.br>

Excluindo um objeto persistente

- **Recuperar o objeto pela sessão**
- **Comando o *delete()***

Excluindo um objeto persistente

```
//... imports omitidos

public class Altera
{
    public static void main(String[] args) {

        //... obter o objeto Session

        Transaction tx = session.beginTransaction();
        Contato contato = (Contato)session.get
            (Contato.class, Long.valueOf(args[0]));
        session.delete(contato);

        //... libera a conexão
    }
}
```

Excluindo um objeto persistente

[0] Cap Mello <cgmello@de9.ime.eb.br>

[1] Maj Luiz Henrique <lharaujo@de9.ime.eb.br>

[2] TC Cecilio <cecilio@de9.ime.eb.br>

>java mapeamentoor.Remove 0

Hibernate: select contato0_.id as id0_0_, contato0_.nome as nome0_0_,
contato0_.email as email0_0_ from contato contato0_ where contato0_.id=?

Hibernate: delete from contato where id=?

/* chama Lista.main(null); para listar */

Hibernate: select contato0_.id as id2_, contato0_.nome as nome2_,
contato0_.email as email2_ from contato contato0_

[1] Maj Luiz Henrique <lharaujo@de9.ime.eb.br>

[2] TC Cecilio <cecilio@de9.ime.eb.br>

Estados do objeto Java

- **Um objeto Java é persistente se ele está vinculado a uma sessão do Hibernate**
- **Um objeto não vinculado a uma sessão é um objeto transiente**
- **Esse vínculo tem a duração da sessão, depois da confirmação (ou cancelamento) da transação, dizemos que o objeto está desconectado (*detached*)**

Objetos desconectados

- **Um objeto pode ser reconectado a uma sessão Hibernate através do método update()**
- **Existe ainda o método saveOrUpdate(), caso não se saiba se o objeto é novo ou se foi lido de uma sessão Hibernate**
- **Assim podemos passar objetos persistentes para clientes remotos e depois sincronizá-los de volta**

Exceções

- **Sempre que houver uma exceção, a sessão deve ser descartada e o objeto persistente deve ser tratado por uma nova sessão (chamando novamente SessionFactory)**

Mapeando associações

- Vamos criar uma **categoria** para agrupar os contatos
- Vamos anexar a cada contato um **endereço**
- Categoria e contato serão mapeados como objetos persistentes diferentes
- Endereço será mapeado como um valor dentro de contato
  - como uma string, mas que será mapeado como colunas do BD

Contato e Endereco

```
public class Contato2 implements Serializable {

    private Long id;
    private String nome;
    private String email;
    private Endereco endereco;

// .. métodos get/set
```

```
public class Endereco
{
    private String logradouro;
    private int numero;
    private String complemento;
    private String bairro;
    private String cep;

// .. métodos get/set
```

Mapeamento

```
<hibernate-mapping>
    <class name="mapeamentoor.Contato2" table="contato2">
        <id name="id">
        //... propriedades nome e email

        <component name="endereco">
            <property name="logradouro">
                <column name="logradouro" length="40"
                    not-null="true"/>
            </property>
            <property name="numero">
                <column name="numero" not-null="true"/>
            </property>
            //... outras propriedades do endereço
        </component name="endereco">
    </class>
```

Elemento <component>

- **Permite mapear várias classes para apenas uma tabela**
- **Somente a classe que contém a componente possui ID para ser recuperada e atualizada no BD**

Contato e Categoria

```
public class Contato3 implements Serializable {

    private Long id;
    private String nome;
    private String email;
    private Categoria categoria;

// .. métodos get/set
```

```
public class Categoria
{
    private Long id;
    private String nome;
    private Set contatos;

// .. métodos get/set
```

Mapeamento Contato p/ Categoria

```
<hibernate-mapping>

    <class name="mapeamentoor.Contato3" table="contato3">

        //... id e propriedades do contato

        <many-to-one name="categoria" column="CATEGORIA"
            class="mapeamentoor.Categoria"/>
    </class>

</hibernate-mapping>
```

Relacionamento n:1

- Um Contato está vinculado a uma Categoria
- Uma Categoria está vinculada a vários Contatos
- Esse relacionamento pode ser uni- ou bi-direcional, dependendo da existência de atributos que relacionam as classes
- Normalmente implementamos essa relação com uma chave estrangeira (FK) num modelo relacional

Mapeamento Categoria p/ Contato

```
<hibernate-mapping>

    <class name="mapeamentoor.Categoria" table="categoria">

        //... id e propriedades da categoria

        <set name="contatos" inverse="true">
            <key column="CATEGORIA"/>
            <one-to-many class="mapeamentoor.Contato3"/>
        </set>

    </class>

</hibernate-mapping>
```

Relacionamentos

- Uma propriedade que referencia um outro objeto persistente é mapeada por
  - <many-to-one> ou <one-to-one>
- Quando referencia uma outra coleção de objetos persistentes, temos
  - <one-to-many> ou <many-to-many>
- Quando há um relacionamento bidirecional, um dos lados deve ter
  - inverse="true"
- O mapeamento de coleções pode ser feita para
  - set, list, map

Carga antecipada ou sob demanda

- Por default, objetos relacionados só são carregados quando acessados (sob demanda)
- Caso um relacionamento tenha lazy="false", ele será carregado antecipadamente
  - <set name="contatos" inverse="true" lazy="false">

Fetch join

- **Consultas HQL não seguem o mapeamento para a carga antecipada de objetos (somente para coleções)**
- **Para forçar a carga antecipada, use fetch join**
- **Os joins devem respeitar os mapeamentos**

```
Query query = session.createQuery(
    "from Categoria as cat " +
    "left join fetch cat.contatos ");
```

Lista contatos de uma categoria

```
Query query = session.createQuery("from Categoria as cat " +
    "left join fetch cat.contatos ");
List categorias = query.list();
Iterator it = categorias.iterator();
while (it.hasNext()) {
    Categoria categoria = (Categoria) it.next();
    System.out.println("*** [" + categoria.getId() + "] "
        + categoria.getNome());
    Iterator it2 = categoria.getContatos().iterator();
    while (it2.hasNext()) {
        Contato3 contato = (Contato3) it2.next();
        System.out.println("[" + contato.getId() + "] "
            + contato.getNome() + " <" + contato.getEmail() + ">");
    }
}
```

Total de contatos de uma categoria

```
Query query = session.createQuery("select cat.nome, count(con.nome) " +
    "from Categoria as cat " +
    "left join cat.contatos con " +
    "group by cat.nome");
List categorias = query.list();
Iterator it = categorias.iterator();
while (it.hasNext()) {
    Object[] dados = (Object[]) it.next();
    System.out.println(dados[0] + ":\t" + dados[1]);
}
```

Search

```
Query query = session.createQuery(
    "select c from Contato as c " +
    "where lcase(c.nome) like (:nome) " +
    "order by nome asc");
query.setString("nome", "%" + args[0].toLowerCase() + "%");
Iterator it = query.iterate();
//Iterator it = query.list().iterator();
while (it.hasNext()) {
    Contato contato = (Contato) it.next();
    System.out.println("[" + contato.getId() + "] "
        + contato.getNome()
        + " <" + contato.getEmail() + ">");
}
```