INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Câmpus São Carlos

PRW3 - Programação para a WEB III

JPA parte 2

Conteúdo 03

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Câmpus São Carlos

AVISO!! (será repetido todas as aulas)

- Mantenha a disposição de cabos nos computadores dos labs.
- Se retirar um cabo de rede (para usar notebook),
  não esqueça de repor no computador ao final da aula.
- Mantenha o filtro de linha e as tomadas, atrás da mesa,
  em seu estado original, especialmente para que não possa ser esbarrado por outros alunos.
- Não altere de jeito nenhum os cabos atrás dos computadores,
  especialmente mexendo nas “travas”.
- **Por favor, antes de sair, desligue(m) a(s) máquina(s).**
- Lembre-se! Os laboratórios são nossos!! Vamos cuidar!

O que fizemos na aula passada ?

- Lembramos jdbc e seus “problemas”
- Hibernate: Biblioteca para ORM (object relational mapping)
  - mapear entidades (objetos) diretamente para o BD
- JPA como ORM “oficial” e padronizado
- Criar projeto no IntelliJ, definindo as ferramentas (bibliotecas) que serão utilizadas
  - Eclipselink, Hibernate, Mysql, H2
    - No final, **Hibernate com H2**
- Arquivos de configuração: META-INF, persistence.xml, etc...
- Anotações para definir as entidades, campo id, etc...
- Testando a persistência de um objeto no BD
- Usando console jar do H2 para poder visualizar os dados no banco

Na aula passada, gravamos o BD do H2
em um arquivo no disco,
e vimos como usar um “console” texto,
fornecido pelo H2,
para visualizar os dados...

Legal! Temos o arquivo com os dados. Mas como “olhar dentro dele” ?

- O H2 fornece um “console” para visualização e manipulação de dados, que funciona via browser.
  - Porém é preciso ter um “servidor web” rodando localmente para poder usar este console. Usaremos bastante no futuro, quando trabalharmos com o Spring.
- O H2 também fornece um executável Java (“jar”) que é um console em tela de texto.
- Se você adicionou a dependência do H2 no seu projeto, ele gravou este JAR na sua máquina.
  - Localização:
    - Linux:
      `~/.m2/repository/com/h2database/h2/“num da versão”`
    - Windows:
      `C:\Users\SeuUsuario\.m2\repository\com\h2database\h2\versão\h2-versão.jar`
      - A conferir...

Legal! Temos o arquivo com os dados.
Mas como “olhar dentro dele” ?

- O H2 forne
  dados, que
  - Porém é
    para poc
    trabalha
- O H2 tamb
  em tela de
- Se você ac
  este JAR n
  - Localiza
    - Linux
      ~/.m
    - Wind
      C:\U
      vers
      - /

Para rodar: `java -cp h2.jar org.h2.tools.Shell`

```
carlao2005@celtasdesktop:~/HD_PRINCIPAL/Dropbox/intellijProjects/loja20251/data$
  java -cp h2-2.3.232.jar org.h2.tools.Shell

Welcome to H2 Shell 2.3.232 (2024-08-11)
Exit with Ctrl+C
[Enter]   jdbc:h2:~/test
URL       jdbc:h2:file:./loja
[Enter]   org.h2.Driver
Driver
[Enter]
User      sa
Password  Password  ><
Connected
Commands are case insensitive; SQL statements end with ';'
help or ?      Display this help
list           Toggle result list / stack trace mode
maxwidth       Set maximum column width (default is 100)
autocommit     Enable or disable autocommit
history        Show the last 20 statements
quit or exit   Close the connection and exit

sql>
```

enter

```
<property name="jakarta.persistence.jdbc.user" value="sa" />
<property name="jakarta.persistence.jdbc.password" value="" />
```

Legal! Temos o arquivo com os dados.
Mas como “olhar dentro dele” ?

- O H2 forne
  dados, que
  - Porém é
    para poc
    trabalha
- O H2 tamb
  em tela de
- Se você ac
  este JAR n
  - Localiza
    - Linux
      ~/.m
    - Wind
      c:\u
      vers
      -

Para rodar: `java -cp h2.jar org.h2.tools.Shell`

```
carlao2005@celtasdesktop:~/HD_PRINCIPAL/Dropbox/intellijProjects/loja20251/data$
  java -cp h2-2.3.232.jar org.h2.tools.Shell

Welcome to H2 She
Exit with Ctrl+C
[Enter]   jdbc:h2
URL       jdbc:h2
[Enter]   org.h2.
Driver
[Enter]
User      sa
Password  Passwor
Connected
Commands are case
help or ?      Di
list           To
maxwidth       Se
autocommit     En
history        Sh
quit or exit   Cl

sql>
```

```
sql> show tables;
TABLE_NAME | TABLE_SCHEMA
PRODUTOS   | PUBLIC
(1 row, 21 ms)
sql>
```

```
sql> select * from produtos;
ID | DESCRICAO                    | NOME         | PRECO
1  | Bom preço, boa performance | Samsung A51 | 1700.00
(1 row, 0 ms)
sql>
```

Pois bem, ontem descobri uma coisa bem interessante...

Pois bem, ontem descobri uma coisa bem interessante...

MESMO SEM SPRING DA PRA USAR O CONSOLE NA WEB !

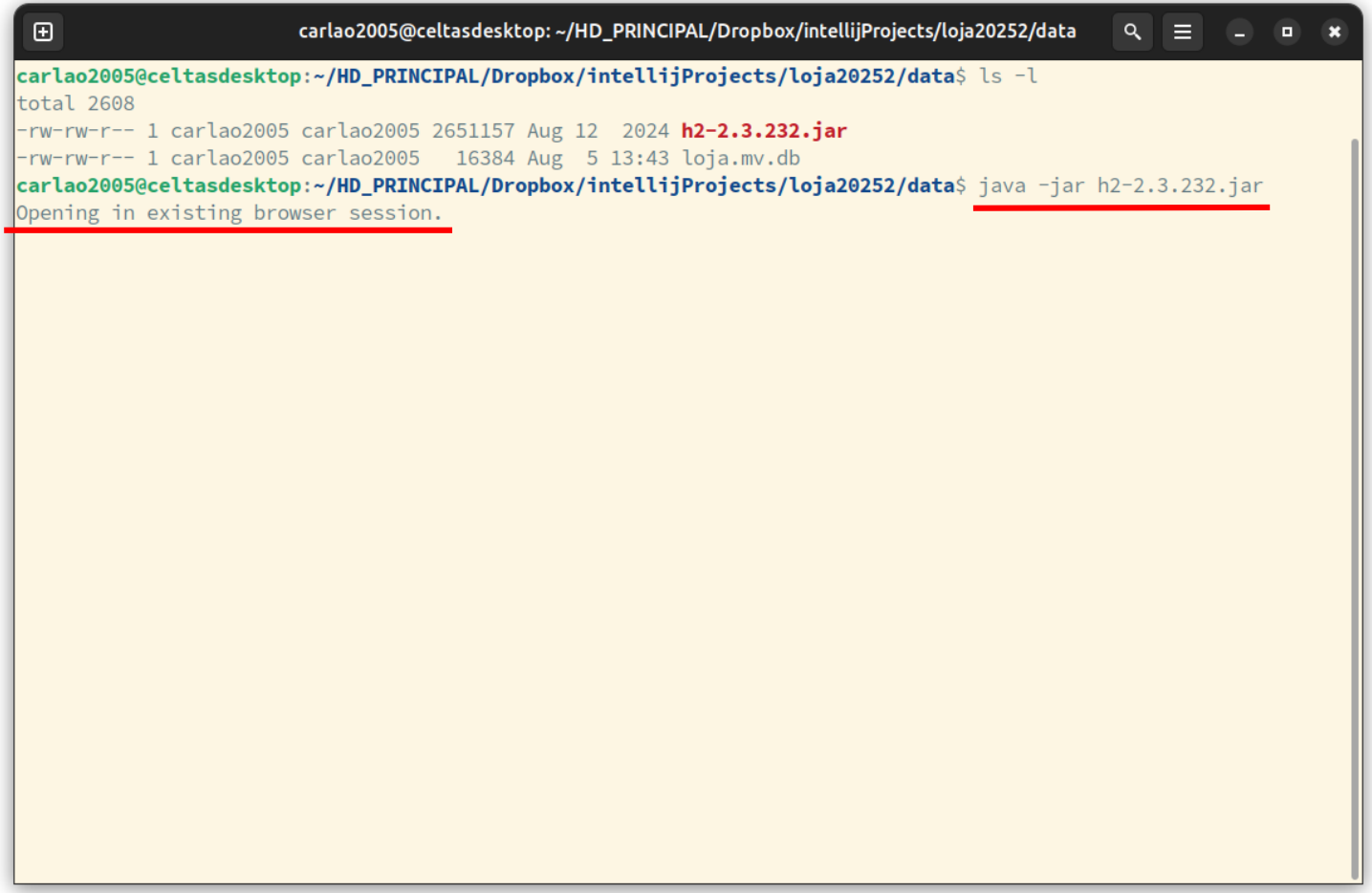
Para rodar: `java -cp h2.jar org.h2.tools.Shell`

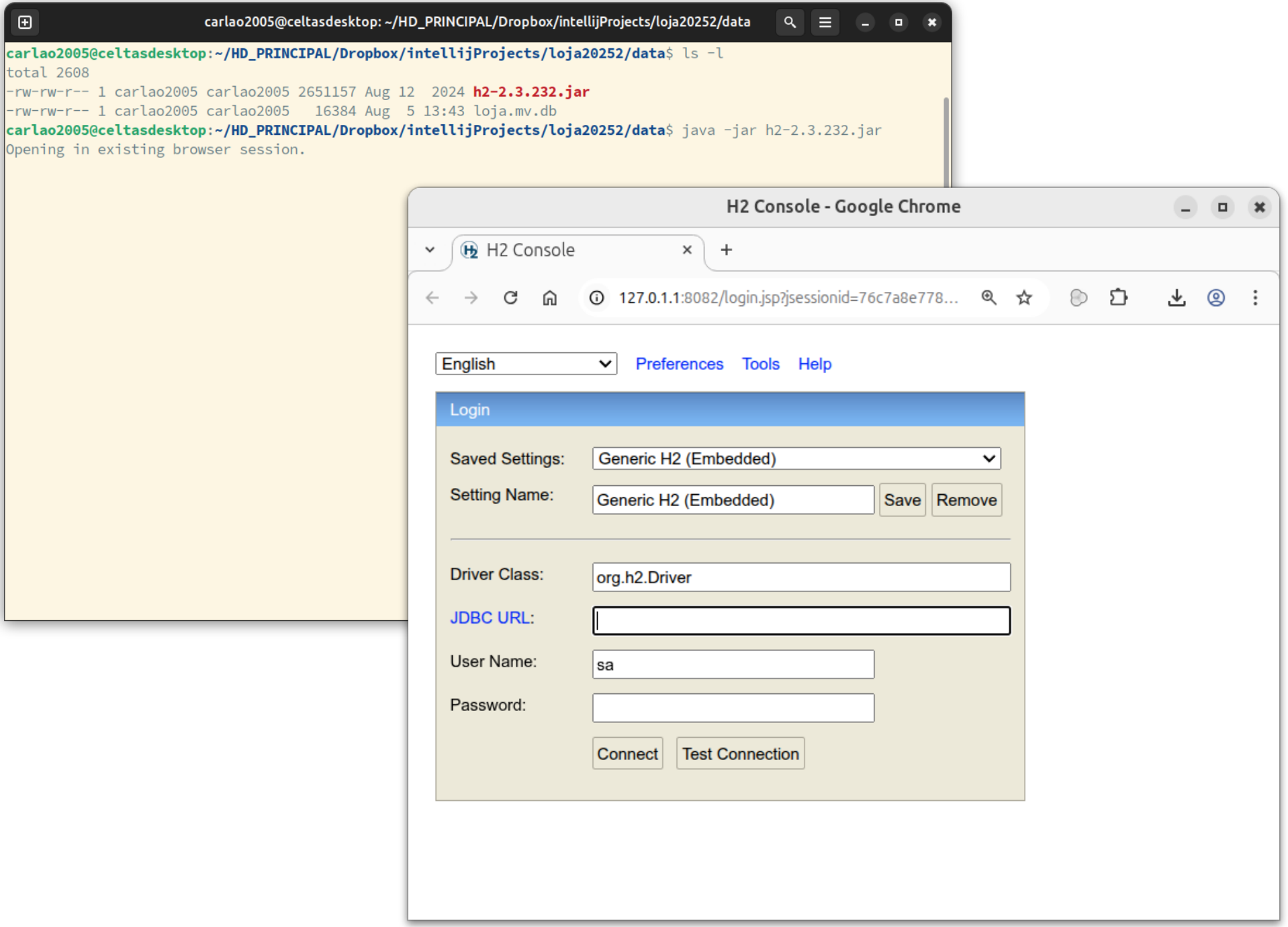
Se rodar apenas

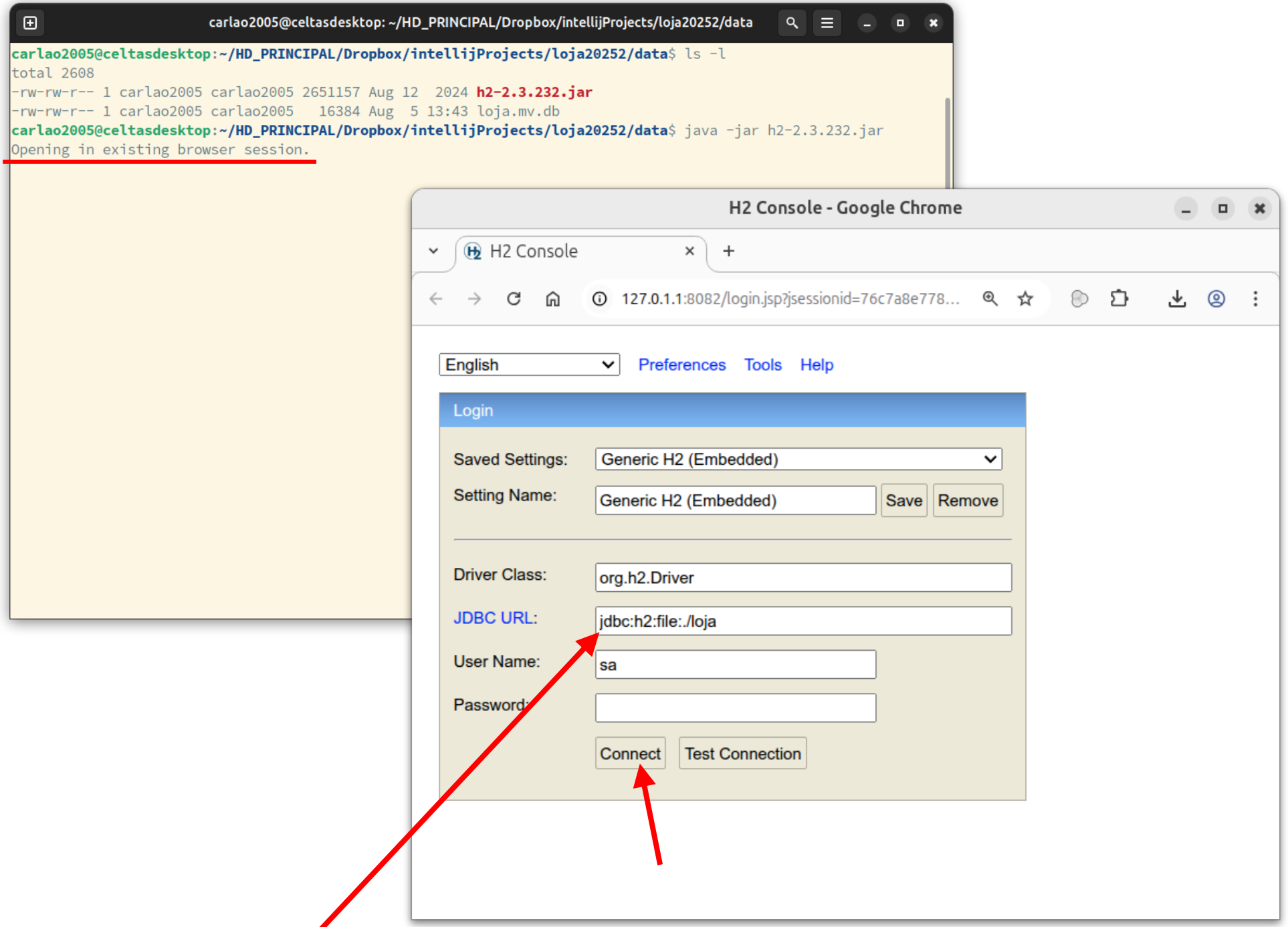
`java -jar h2.jar`

ele roda o console gráfico,
abrindo uma aba no browser!!!

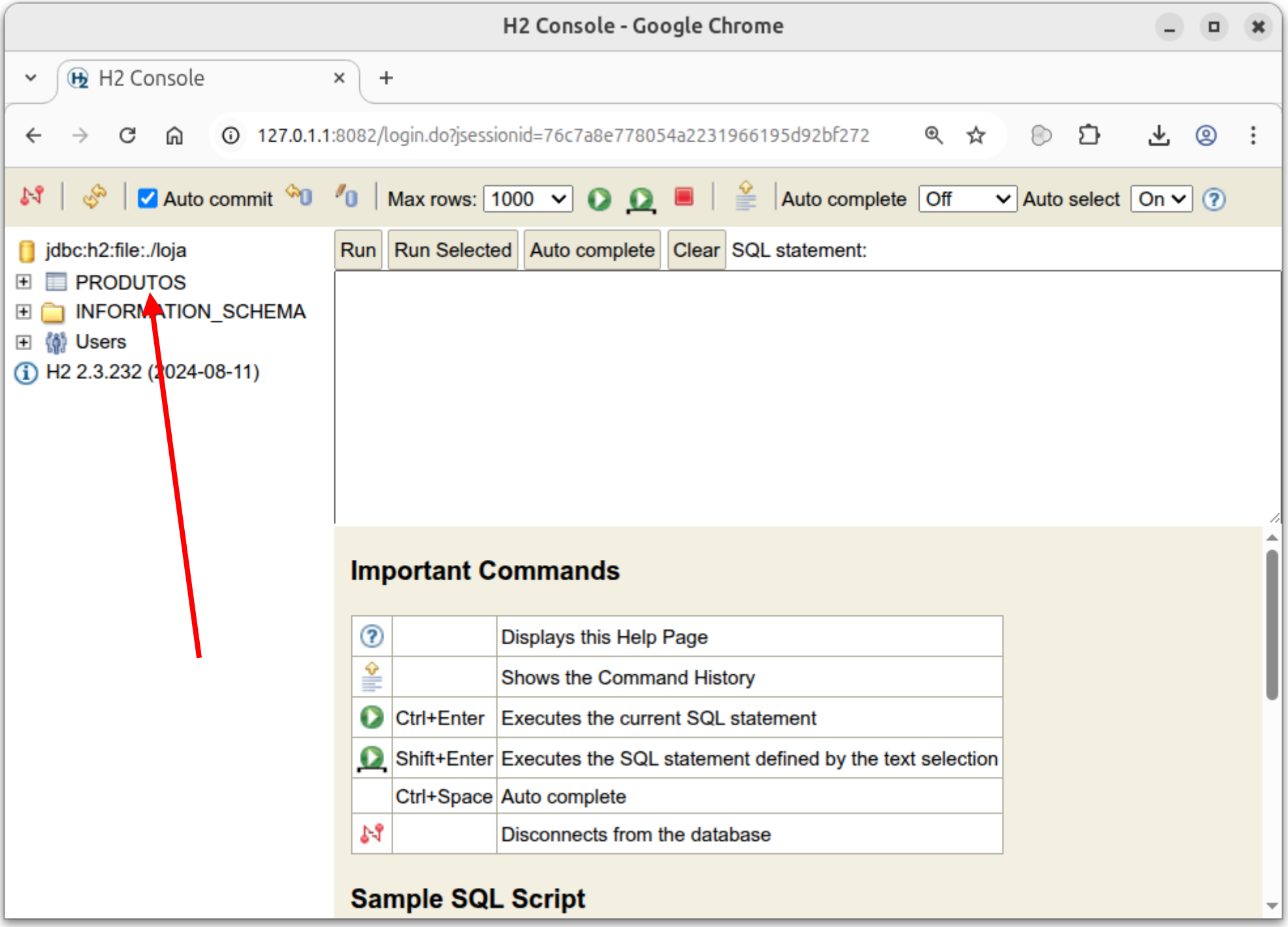

carlao2005@celtasdesktop: ~/HD_PRINCIPAL/Dropbox/intellijProjects/loja20252/data
carlao2005@celtasdesktop:~/HD_PRINCIPAL/Dropbox/intellijProjects/loja20252/data$ ls -l
total 2608
-rw-rw-r-- 1 carlao2005 carlao2005 2651157 Aug 12 2024 h2-2.3.232.jar
-rw-rw-r-- 1 carlao2005 carlao2005   16384 Aug 5 13:43 loja.mv.db
carlao2005@celtasdesktop:~/HD_PRINCIPAL/Dropbox/intellijProjects/loja20252/data$

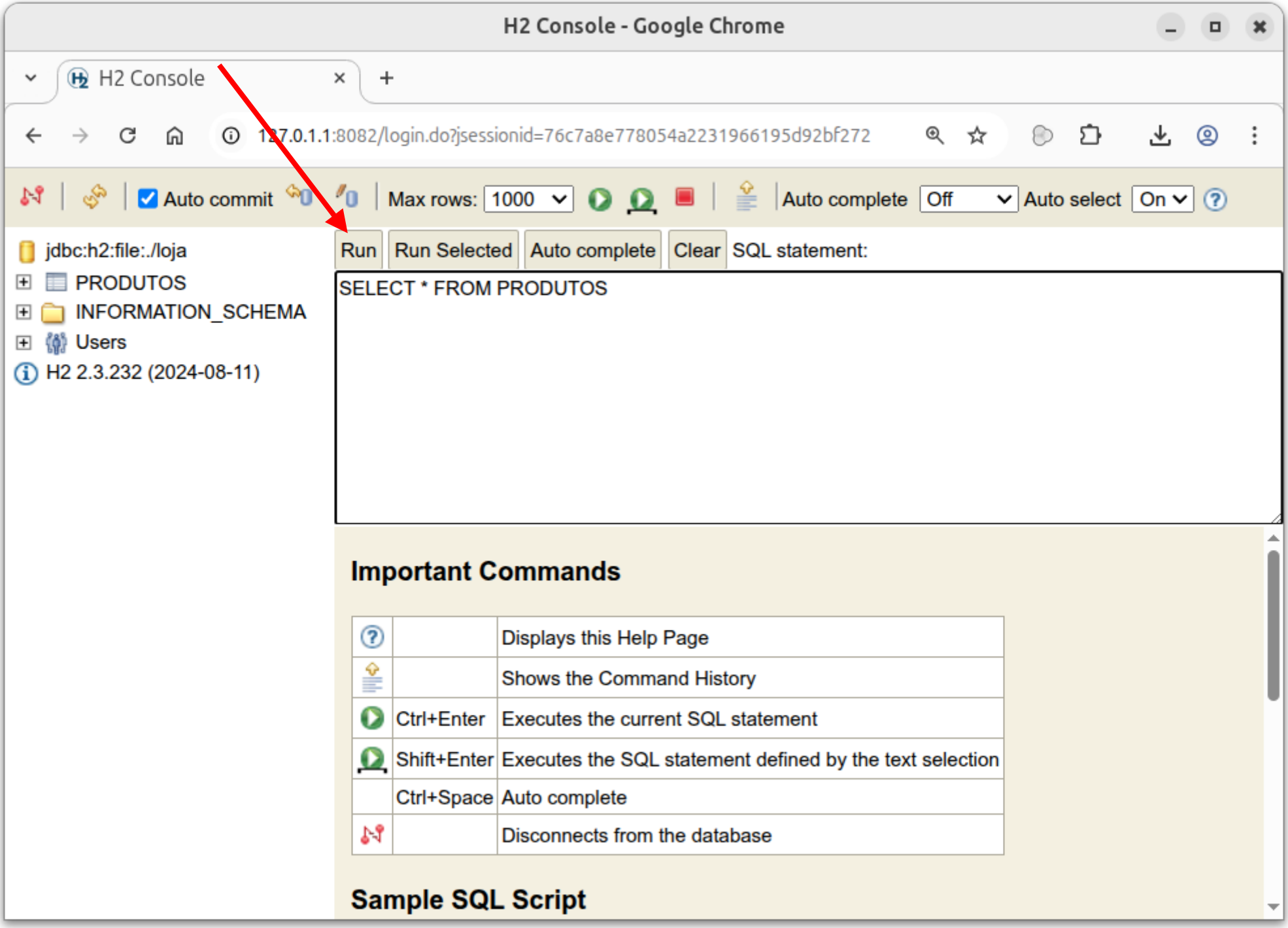

carlao2005@celtasdesktop: ~/HD_PRINCIPAL/Dropbox/intellijProjects/loja20252/data
carlao2005@celtasdesktop:~/HD_PRINCIPAL/Dropbox/intellijProjects/loja20252/data$ ls -l
total 2608
-rw-rw-r-- 1 carlao2005 carlao2005 2651157 Aug 12 2024 h2-2.3.232.jar
-rw-rw-r-- 1 carlao2005 carlao2005   16384 Aug 5 13:43 loja.mv.db
carlao2005@celtasdesktop:~/HD_PRINCIPAL/Dropbox/intellijProjects/loja20252/data$ java -jar h2-2.3.232.jar
Opening in existing browser session.

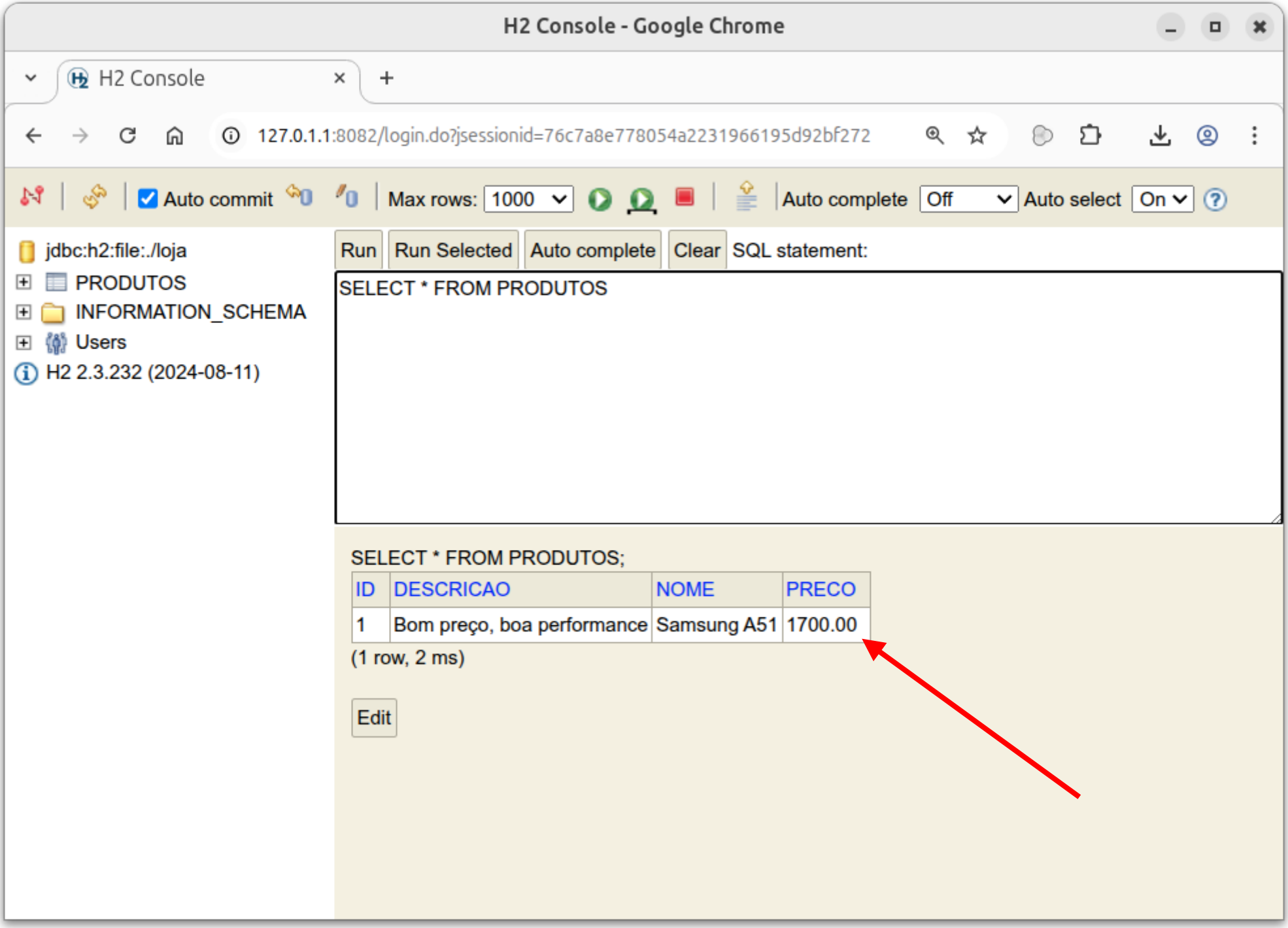

carlao2005@celtasdesktop: ~/HD_PRINCIPAL/Dropbox/intellijProjects/loja20252/data
carlao2005@celtasdesktop:~/HD_PRINCIPAL/Dropbox/intellijProjects/loja20252/data$ ls -l
total 2608
-rw-rw-r-- 1 carlao2005 carlao2005 2651157 Aug 12 2024 h2-2.3.232.jar
-rw-rw-r-- 1 carlao2005 carlao2005 16384 Aug 5 13:43 loja.mv.db
carlao2005@celtasdesktop:~/HD_PRINCIPAL/Dropbox/intellijProjects/loja20252/data$ java -jar h2-2.3.232.jar
Opening in existing browser session.
H2 Console - Google Chrome
H2 Console
127.0.1.1:8082/login.jsp?jsessionid=76c7a8e778...
English
Preferences
Tools
Help
Login
Saved Settings:
Generic H2 (Embedded)
Setting Name:
Generic H2 (Embedded)
Save
Remove
Driver Class:
org.h2.Driver
JDBC URL:
User Name:
sa
Password:
Connect
Test Connection



jdbc:h2:file:./loja


H2 Console - Google Chrome
H2 Console
127.0.1.1:8082/login.do?jsessionid=76c7a8e778054a2231966195d92bf272
Auto commit
Max rows: 1000
Auto complete Off
Auto select On
jdbc:h2:file:./loja
PRODUTOS
INFORMATION_SCHEMA
Users
H2 2.3.232 (2024-08-11)
Run
Run Selected
Auto complete
Clear
SQL statement:
Important Commands
Displays this Help Page
Shows the Command History
Ctrl+Enter
Executes the current SQL statement
Shift+Enter
Executes the SQL statement defined by the text selection
Ctrl+Space
Auto complete
Disconnects from the database
Sample SQL Script


H2 Console - Google Chrome
H2 Console
127.0.1.1:8082/login.do?jsessionid=76c7a8e778054a2231966195d92bf272
Auto commit
Max rows: 1000
Auto complete Off
Auto select On
jdbc:h2:file:./loja
PRODUTOS
INFORMATION_SCHEMA
Users
H2 2.3.232 (2024-08-11)
Run
Run Selected
Auto complete
Clear
SQL statement:
SELECT * FROM PRODUTOS
Important Commands
Displays this Help Page
Shows the Command History
Ctrl+Enter
Executes the current SQL statement
Shift+Enter
Executes the SQL statement defined by the text selection
Ctrl+Space
Auto complete
Disconnects from the database
Sample SQL Script


H2 Console - Google Chrome
H2 Console
127.0.1.1:8082/login.do?jsessionid=76c7a8e778054a2231966195d92bf272
Auto commit
Max rows: 1000
Auto complete Off
Auto select On
jdbc:h2:file:./loja
PRODUTOS
INFORMATION_SCHEMA
Users
H2 2.3.232 (2024-08-11)
Run
Run Selected
Auto complete
Clear
SQL statement:
SELECT * FROM PRODUTOS
SELECT * FROM PRODUTOS;
ID
DESCRICAO
NOME
PRECO
1
Bom preço, boa performance
Samsung A51
1700.00
(1 row, 2 ms)
Edit

H2 Console - Google Chrome

H2 Console

127.0.1.1:8082/login.do?jsessionid=76c7a8e778054a2231966195d92bf272

Auto commit | Max rows: 1000 | Auto complete Off | Auto select On

jdbc:h2:file:./loja
PRODUTOS
INFORMATION_SCHEMA
Users
H2 2.3.232 (2024-08-11)

Run | Run Selected | Auto complete | Clear | SQL statement:

```
SELECT * FROM PRODUTOS
```

PRODUTOS;

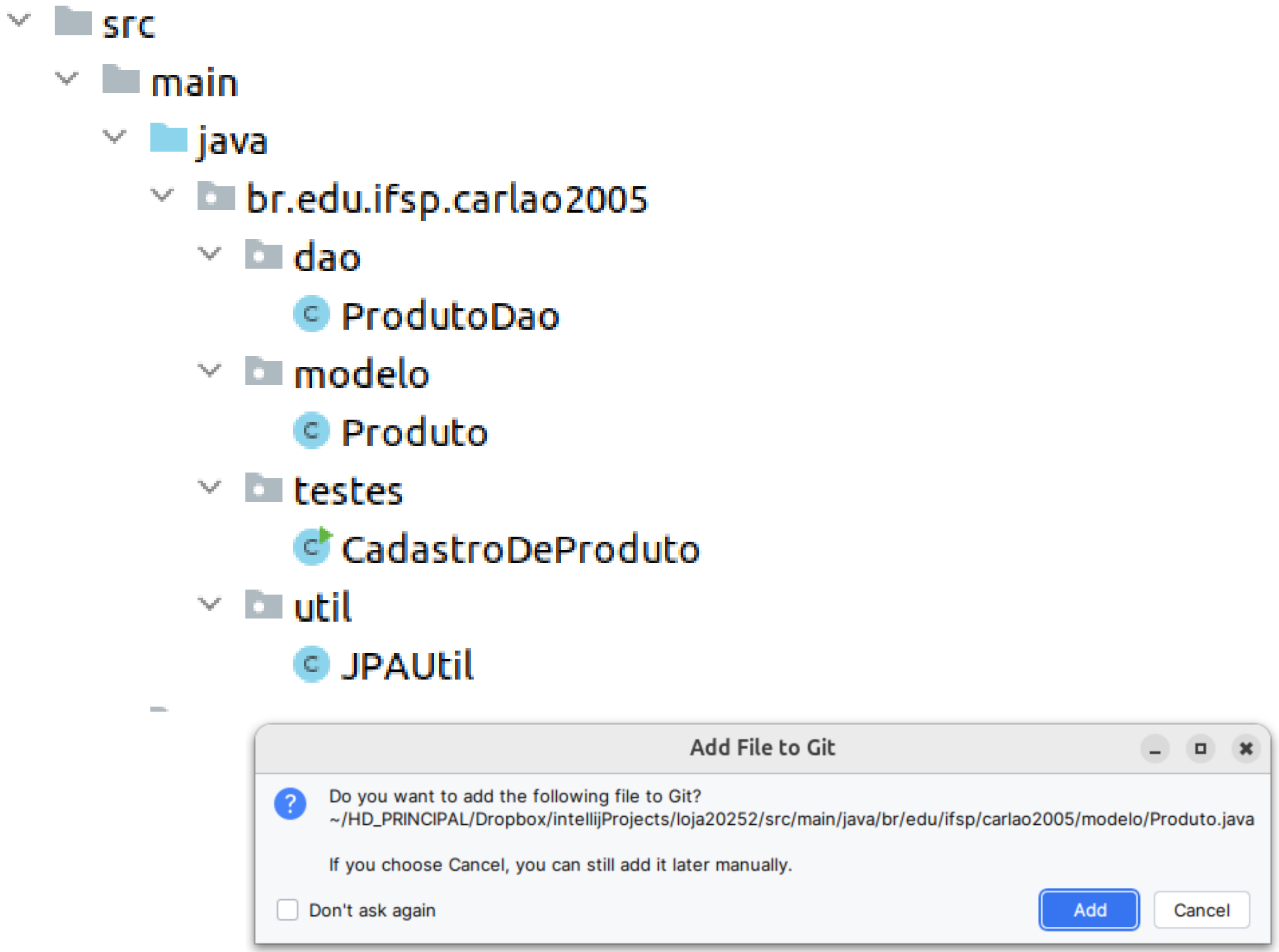
	NOME	PRECO
oa performance	Samsung A51	1700.00

Vivendo e aprendendo...

Retomando o projeto iniciado na aula anterior:

Organizando o código....

Classe JPAUtil
Classe DAO


src
main
java
br.edu.ifsp.carlao2005
dao
ProdutoDao
modelo
Produto
testes
CadastroDeProduto
util
JPAUtil
Add File to Git
Do you want to add the following file to Git?
~/HD_PRINCIPAL/Dropbox/intellijProjects/loja20252/src/main/java/br/edu/ifsp/carlao2005/modelo/Produto.java
If you choose Cancel, you can still add it later manually.
Don't ask again
Add
Cancel

Classe JPAUtil

```
import jakarta.persistence.EntityManager;
import jakarta.persistence.EntityManagerFactory;
import jakarta.persistence.Persistence;

public class JPAUtil {

    // Atributo FACTORY, constante, pertencente a esta classe.
    // Só vai criar a EntityManagerFactory apenas uma vez.
    private static final EntityManagerFactory FACTORY =
            Persistence.createEntityManagerFactory("loja");

    // Método que devolve um EntityManager.
    // Método static, ou seja, pertence a classe.
    public static EntityManager getEntityManager() {
        return FACTORY.createEntityManager();
    }

}
```

Classe ProdutoDao

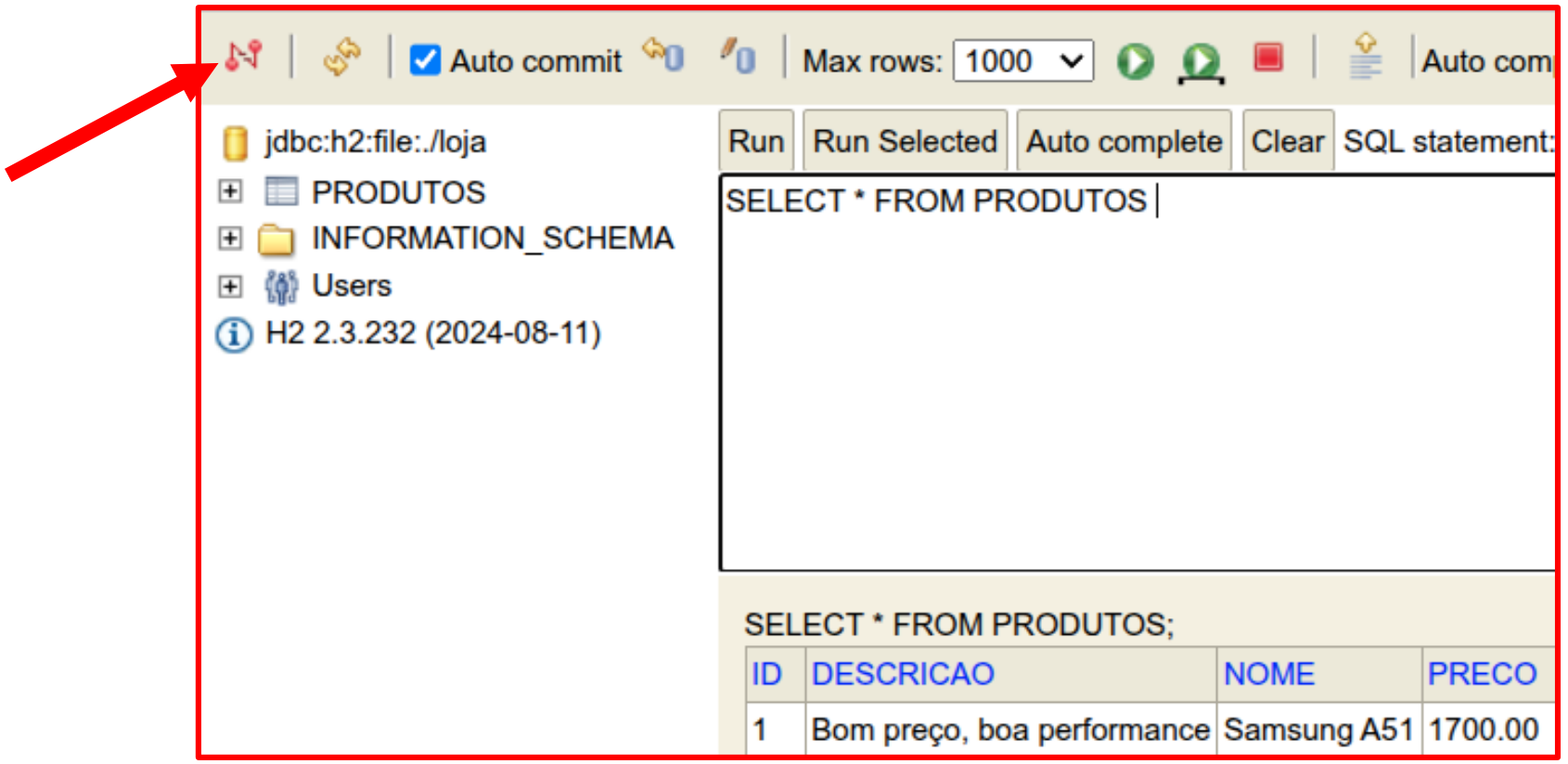
```java
import br.edu.ifsp.carlao2005.modelo.Produto;
import jakarta.persistence.EntityManager;

public class ProdutoDao {

    // EntityManager, que será usado por todos os métodos:
    private EntityManager em;

    // Construtor que já recebe o EntityManager criado:
    public ProdutoDao(EntityManager em) {
        this.em = em;
    }

    // Método para gravar um produto no BD:
    public void cadastrar(Produto produto) {
        this.em.persist(produto);
    }
}
```

Novo CadastroDeProduto:

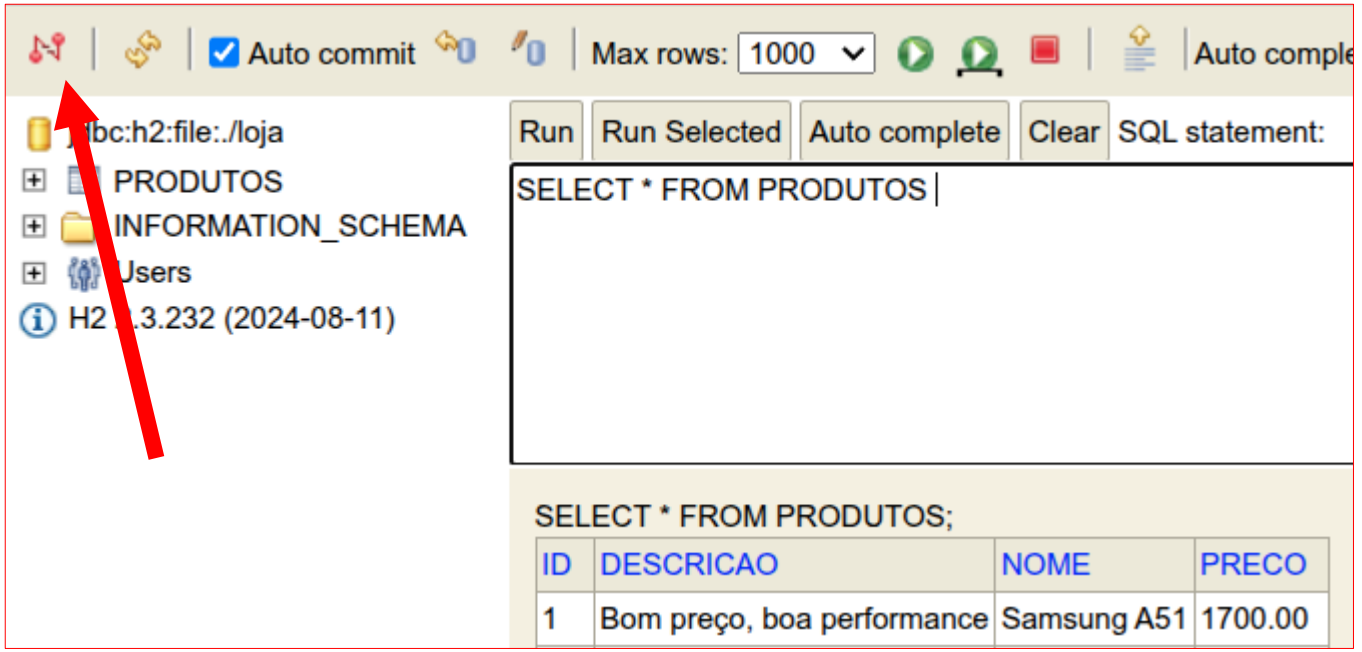
```
public class CadastroDeProduto {

    public static void main(String[] args) {


        // Criando um objeto da Classe Produto:
        Produto celular = new Produto();
        celular.setNome("Samsung S");
        celular.setDescricao("Muito caro");
        celular.setPreco( new BigDecimal("2700") );

        // Utilizando a classe JPAUtil para recuperar um EntityManager:
        EntityManager em = JPAUtil.getEntityManager();

        // Criando o ProdutoDao:
        ProdutoDao dao = new ProdutoDao(em);

        // Iniciando uma transação:
        em.getTransaction().begin();

        // Gravando o objeto no banco de dados:
        dao.cadastrar(celular);

        // "Comitando" a transação:
        em.getTransaction().commit();

        // Fechando este EntityManager, já que não precisaremos mais dele:
        em.close();
    }
}
```

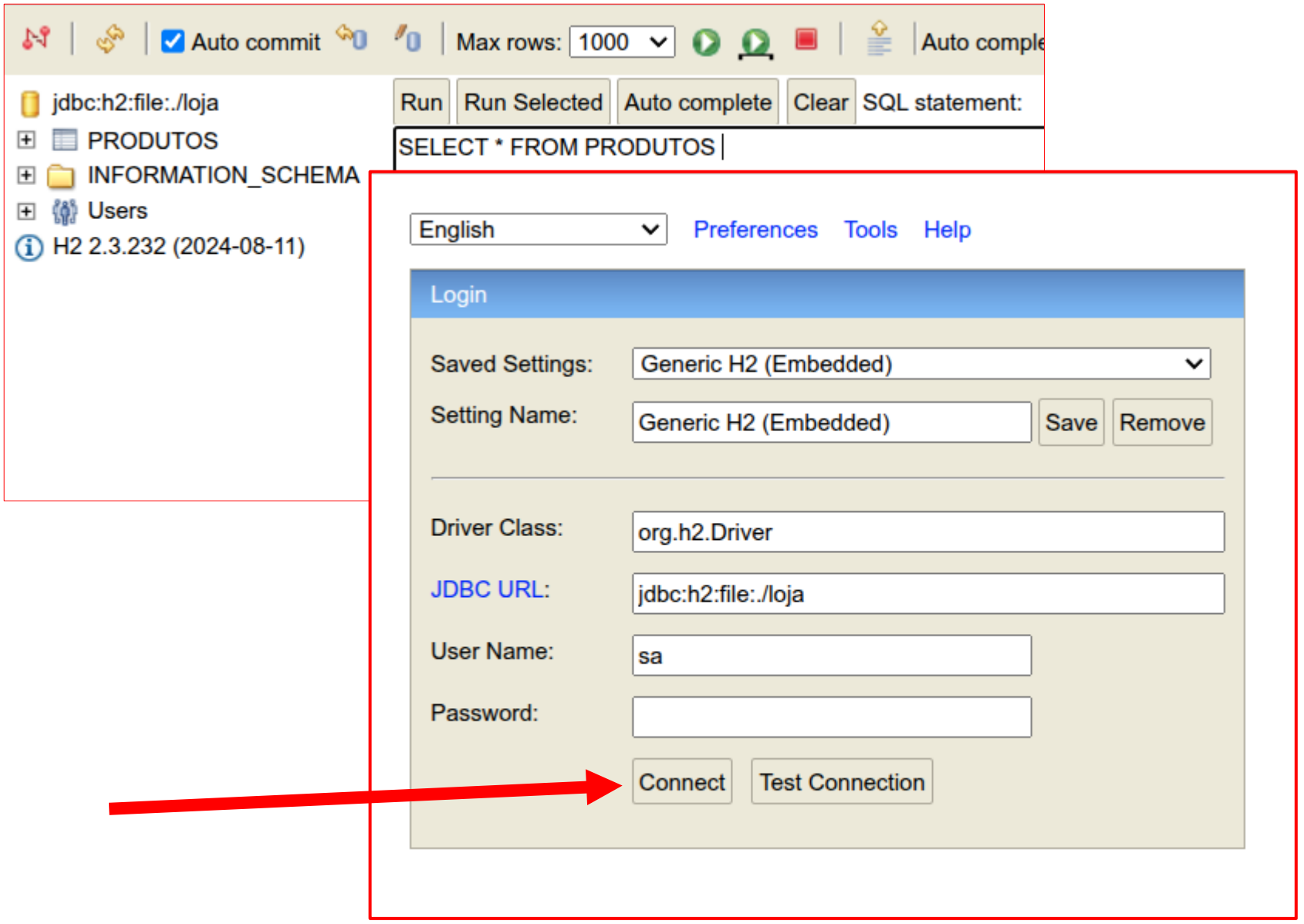
Executando...

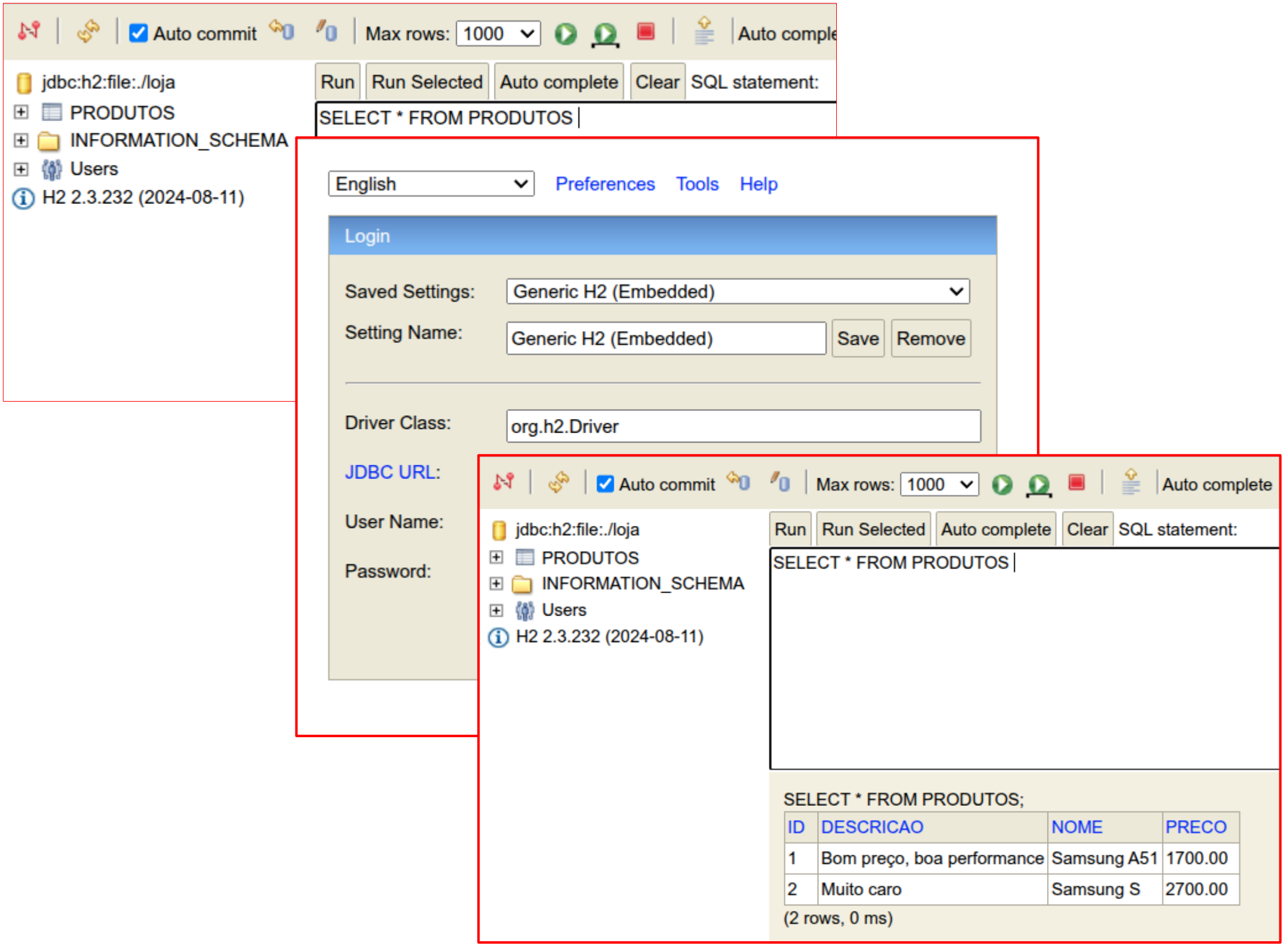
ATENÇÃO!!!!
DESCONECTAR O BANCO DO CONSOLE!!
SENÃO VAI DAR ERRO!!!

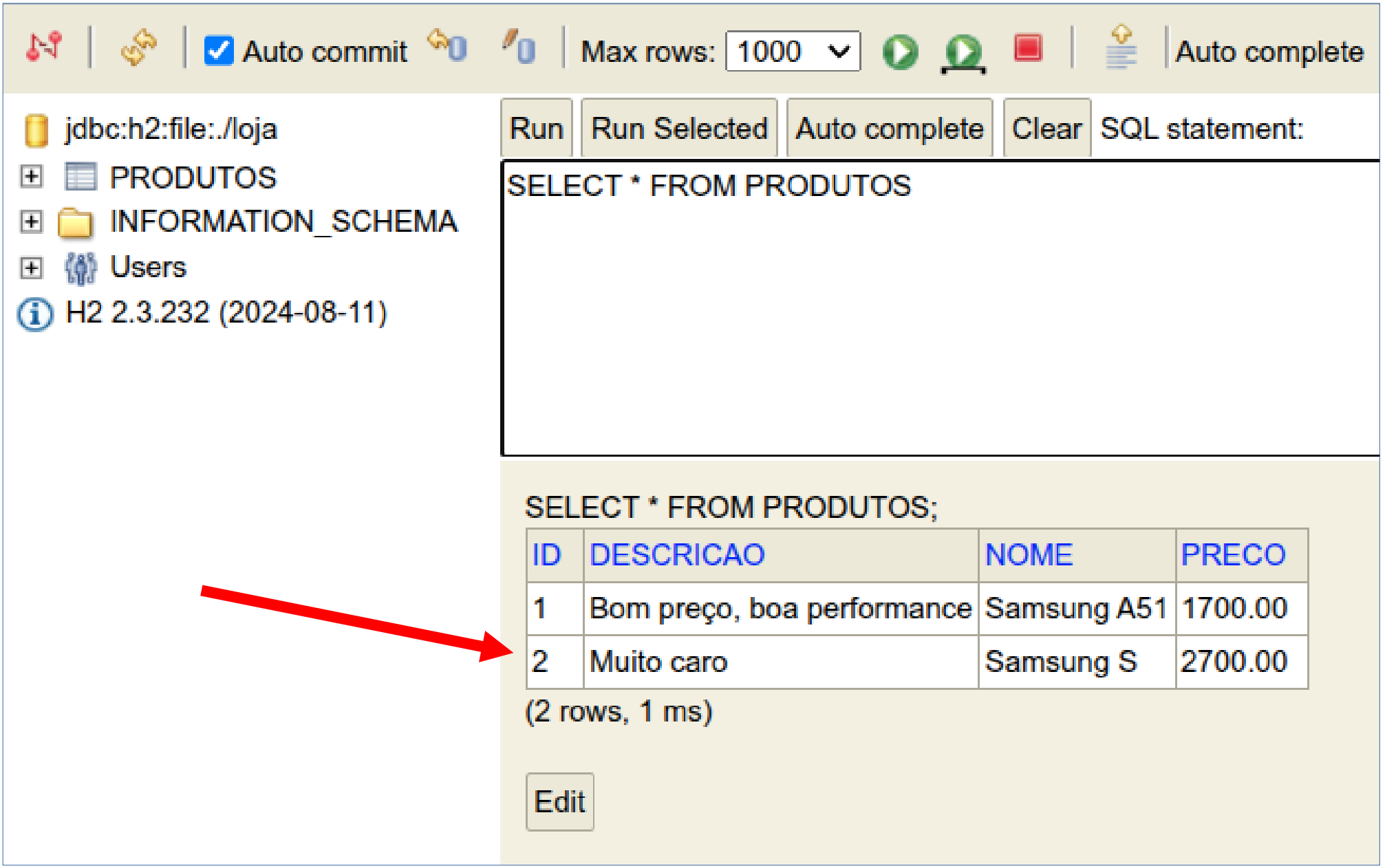


Desconecte, rode o projeto, e depois refaça a conexão do console para ver os dados.


Auto commit
Max rows: 1000
Auto comple
jdbc:h2:file:./loja
PRODUTOS
INFORMATION_SCHEMA
Users
H2 2.3.232 (2024-08-11)
Run
Run Selected
Auto complete
Clear
SQL statement:
SELECT * FROM PRODUTOS
SELECT * FROM PRODUTOS;
ID
DESCRICAO
NOME
PRECO
1
Bom preço, boa performance
Samsung A51
1700.00


Auto commit
Max rows: 1000
Auto comple
jdbc:h2:file:./loja
PRODUTOS
INFORMATION_SCHEMA
Users
H2 2.3.232 (2024-08-11)
Run
Run Selected
Auto complete
Clear
SQL statement:
SELECT * FROM PRODUTOS
English
Preferences
Tools
Help
Login
Saved Settings:
Generic H2 (Embedded)
Setting Name:
Generic H2 (Embedded)
Save
Remove
Driver Class:
org.h2.Driver
JDBC URL:
jdbc:h2:file:./loja
User Name:
sa
Password:
Connect
Test Connection


Auto commit
Max rows: 1000
Auto comple
jdbc:h2:file:./loja
PRODUTOS
INFORMATION_SCHEMA
Users
H2 2.3.232 (2024-08-11)
Run
Run Selected
Auto complete
Clear
SQL statement:
SELECT * FROM PRODUTOS
English
Preferences
Tools
Help
Login
Saved Settings:
Generic H2 (Embedded)
Setting Name:
Generic H2 (Embedded)
Save
Remove
Driver Class:
org.h2.Driver
JDBC URL:
User Name:
Password:
Auto commit
Max rows: 1000
Auto complete
jdbc:h2:file:./loja
PRODUTOS
INFORMATION_SCHEMA
Users
H2 2.3.232 (2024-08-11)
Run
Run Selected
Auto complete
Clear
SQL statement:
SELECT * FROM PRODUTOS
SELECT * FROM PRODUTOS;
ID
DESCRICAO
NOME
PRECO
1
Bom preço, boa performance
Samsung A51
1700.00
2
Muito caro
Samsung S
2700.00
(2 rows, 0 ms)


Auto commit
Max rows: 1000
Auto complete
jdbc:h2:file:./loja
PRODUTOS
INFORMATION_SCHEMA
Users
H2 2.3.232 (2024-08-11)
Run
Run Selected
Auto complete
Clear
SQL statement:
SELECT * FROM PRODUTOS
SELECT * FROM PRODUTOS;
ID DESCRICAO NOME PRECO
1 Bom preço, boa performance Samsung A51 1700.00
2 Muito caro Samsung S 2700.00
(2 rows, 1 ms)
Edit

Vamos mudar a classe Produto:

Inserir a data do cadastro.

Inserir uma categoria de produto:

CELULAR
INFORMATICA
LIVROS

Como definir a categoria ?

- String ?
  - Não é uma boa, pode inserir uma string errada
- Solução:
  - Usar o enum do Java

Em Java, o tipo enumerado (enum) é um recurso que permite definir um tipo de dado composto por um conjunto fixo de constantes nomeadas. Em outras palavras, os enums são uma maneira de representar um grupo de valores relacionados que são mutuamente exclusivos e constantes.

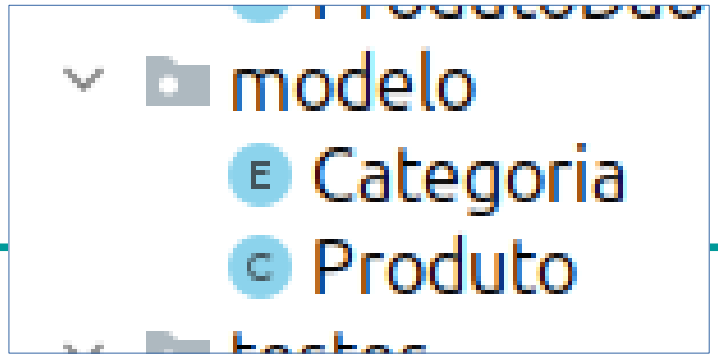
A declaração de um enum segue a seguinte sintaxe:

java Copy code

```java
enum NomeDoEnum {
    CONSTANTE1,
    CONSTANTE2,
    CONSTANTE3,
    // ...
}
```

Enum (no package modelo), Categoria.java:

```
public enum Categoria {
    CELULARES,
    INFORMATICA,
    LIVROS;
}
```



Categoria.java

```
1  package br.edu.ifsp.carlao2005.modelo;
2
3  public enum Categoria {
4
5      CELULARES,
6      INFORMATICA,
7      LIVROS;
8
9  }
```

Nova classe Produto:

```java
@Entity
@Table(name = "produtos")
public class Produto {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nome;
    private String descricao;
    private BigDecimal preco;
    // Atributo que representa a data de cadastro:
    private LocalDate dataCadastro = LocalDate.now();
    // Atributo que representa a categoria.
    // Vamos usar um enum.
    private Categoria categoria;

    // Construtor (não precisa id nem dataCadastro):
    public Produto(String nome, String descricao, BigDecimal preco, Categoria categoria) {
        this.nome = nome;
        this.descricao = descricao;
        this.preco = preco;
        this.categoria = categoria;
    }

    ...
```

Nova gravação (OBS: troquei o main inteiro...) :

```
public class CadastroDeProduto {

    public static void main(String[] args) {

        // Criando um objeto da Classe Produto:
        Produto celular = new Produto("Motorola XXX", "Tela pequena",
                                      new BigDecimal("700"),
                                      Categoria.CELULARES);

        EntityManager em = JPAUtil.getEntityManager();
        ProdutoDao dao = new ProdutoDao(em);
        em.getTransaction().begin();
        dao.cadastrar(celular);
        em.getTransaction().commit();
        em.close();
    }

}
```

Executando...

```
2024-02-16T10:52:59.760-03:00  INFO 11173 --- [          main] o.h.e.t.j.p.i.JtaPlatformInitiator       : HH
2024-02-16T10:52:59.770-03:00  INFO 11173 --- [          main] org.hibernate.orm.connections.access     : HH
Hibernate: alter table if exists produtos add column categoria tinyint check (categoria between 0 and 2)
Hibernate: alter table if exists produtos add column dataCadastro date
Hibernate: insert into produtos (categoria,dataCadastro,descricao,nome,preco,id) values (?,?,?,?,?,default)
```

```
2024-02-16T10:52:59.760-03:00  INFO 11173 --- [          main] o.h.e.t.j.p.i.JtaPlatformInitiator       : HH
2024-02-16T10:52:59.770-03:00  INFO 11173 --- [          main] org.hibernate.orm.connections.access     : HH
Hibernate: alter table if exists produtos add column categoria tinyint check (categoria between 0 and 2)
Hibernate: alter table if exists produtos add column dataCadastro date
Hibernate: insert into produtos (categoria,dataCadastro,descricao,nome,preco,id) values (?,?,?,?,?,default)
```

OBS: Não é toda alteração que consegue fazer automaticamente.
Veremos isso ainda hoje na aula.
Quando abordarmos Spring, utilizaremos uma biblioteca (flyway)
que nos ajudará a automatizar alterações no BD.

SELECT * FROM PRODUTOS;

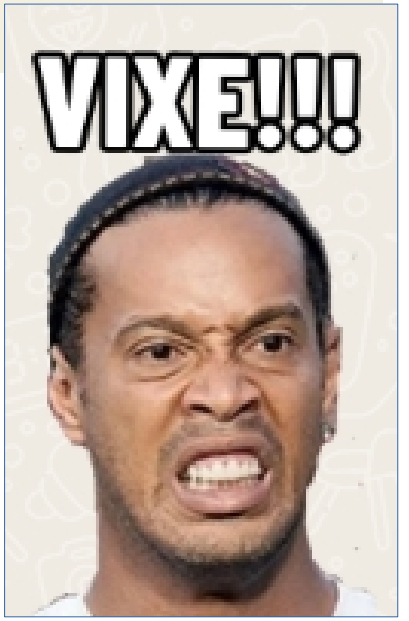
ID	DESCRICAO	NOME	PRECO	CATEGORIA	DATACADASTRO
1	Bom preço, boa performance	Samsung A51	1700.00	*null*	*null*
2	Muito caro	Samsung S	2700.00	*null*	*null*
3	Tela pequena	Motorola XXX	700.00	0	2025-08-05

(3 rows, 1 ms)

SELECT * FROM PRODUTOS;

ID	DESCRICAO	NOME	PRECO	CATEGORIA	DATACADASTRO
1	Bom preço, boa performance	Samsung A51	1700.00	*null*	*null*
2	Muito caro	Samsung S	2700.00	*null*	*null*
3	Tela pequena	Motorola XXX	700.00	0	2025-08-05

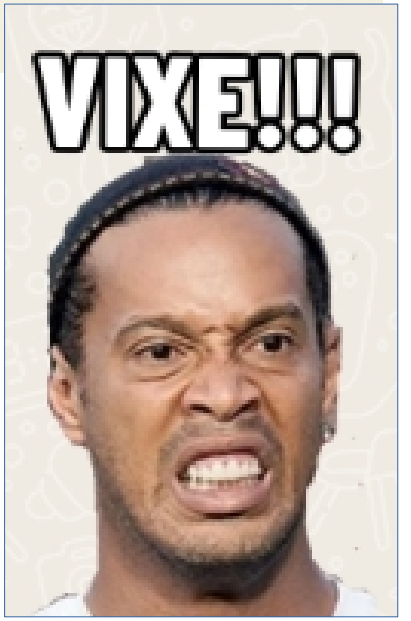
(3 rows, 1 ms)



SELECT * FROM PRODUTOS;

ID	DESCRICAO	NOME	PRECO	CATEGORIA	DATACADASTRO
1	Bom preço, boa performance	Samsung A51	1700.00	*null*	*null*
2	Muito caro	Samsung S	2700.00	*null*	*null*
3	Tela pequena	Motorola XXX	700.00	0	2025-08-05

(3 rows, 1 ms)



Se você não fala nada, *por default* ele coloca o campo como numérico.
E registra nele a "posição" do elemento no enum.

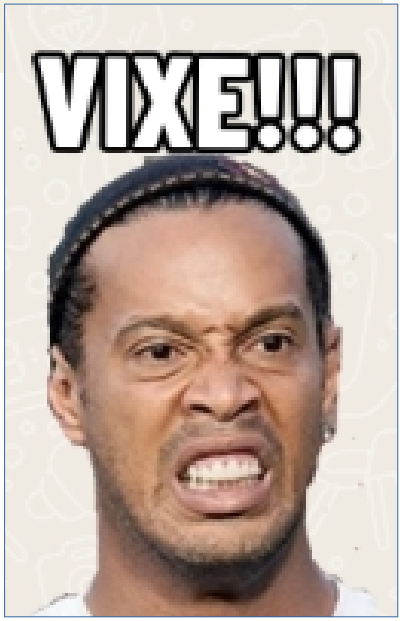
```
public enum Categoria {
    CELULARES,
    INFORMATICA,
    LIVROS;
}
```

0 → CELULARES
1 → INFORMATICA
2 → LIVROS

SELECT * FROM PRODUTOS;

ID	DESCRICAO	NOME	PRECO	CATEGORIA	DATACADASTRO
1	Bom preço, boa performance	Samsung A51	1700.00	*null*	*null*
2	Muito caro	Samsung S	2700.00	*null*	*null*
3	Tela pequena	Motorola XXX	700.00	0	2025-08-05

(3 rows, 1 ms)



Se você não fala nada, *por default* ele coloca o campo como numérico.
E registra nele a "posição" do elemento no enum.

```
public enum Categoria {
    CELULARES,
    INFORMATICA,
    LIVROS;
}
```

0
1
2

Não é uma boa, se depois você reordenar os elementos, inserir elementos novos causando reordenação...

Solução: anotação @Enumerated

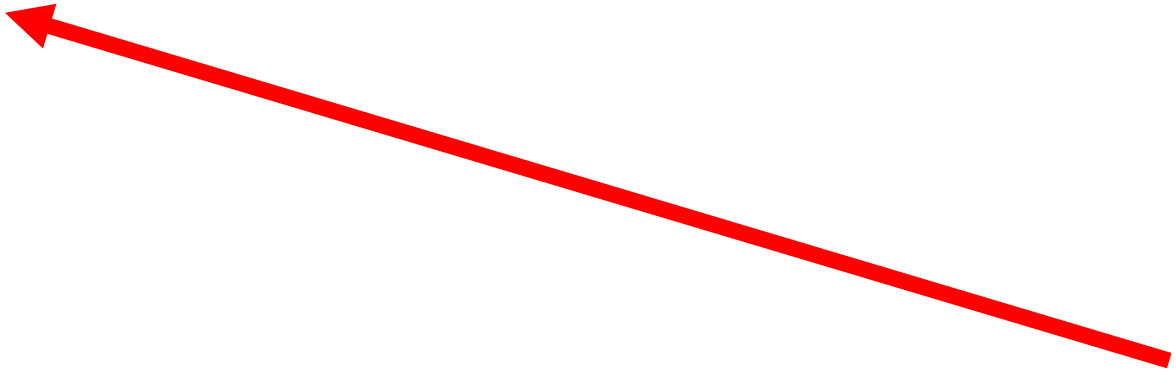
Quando você possui uma entidade JPA com um atributo do tipo enum, o JPA precisa saber como persistir esse atributo no banco de dados. É aí que a anotação `@Enumerated` entra em cena. Ela é usada para especificar o tipo de mapeamento que deve ser usado para persistir o enum.

Existem duas opções para a anotação `@Enumerated`:

1. `EnumType.STRING`: Neste caso, o enum é armazenado no banco de dados como uma string, usando o nome da constante do enum como valor. Por exemplo, se você tiver um enum chamado `Status` com as constantes `ATIVO` e `INATIVO`, usando `EnumType.STRING`, elas seriam armazenadas como "ATIVO" e "INATIVO" no banco de dados.
2. `EnumType.ORDINAL`: Nesta opção, o enum é armazenado no banco de dados como um número inteiro, representando a posição da constante do enum na lista de constantes. Por exemplo, usando `EnumType.ORDINAL`, se o enum `Status` tiver as constantes `ATIVO` e `INATIVO`, elas seriam armazenadas como 0 e 1, respectivamente, no banco de dados.

Alteração na classe Produto:

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;
private String nome;
private String descricao;
private BigDecimal preco;
// Atributo que representa a data de cadastro:
private LocalDate dataCadastro = LocalDate.now();

// Atributo que representa a categoria.
// Vamos usar um enum.
// ATENCAO!!! INDICAR O QUE VAI SER GRAVADO, COM A ANOTAÇÃO @Enumerated
@Enumerated(EnumType.STRING)
private Categoria categoria;
```



ATENÇÃO!

Quando criamos novos atributos,
o hibernate deu um alter table sozinho.

Porém agora estamos trocando o tipo de dados,
de numérico (0, 1, 2...) para string.

Ele não vai fazer sozinho! Vai dar erro!
Precisamos apagar o banco (que agora está em arquivo),
para que o mesmo seja regerado ao se executar o projeto,
agora com os atributos com seus tipos corretos.

ATENÇÃO!

Quando estivermos trabalhando com Spring,
adicionaremos uma biblioteca
que será a responsável por automatizar
alterações no banco para nós (Flyway),
desta forma não teremos
que realizar esta deleção do banco.

Nova gravação:

```
// Criando um objeto da Classe Produto:
Produto celular = new Produto("HP Deskjet 3776", "Cartucho não dá pra nada!",
                              new BigDecimal("750"), Categoria.INFORMATICA);
```

Antes de executar, não esqueça de desconectar, se você estiver usando o console do H2 para ver os dados!

Executando sem apagar o banco:

```
Hibernate: alter table if exists produtos alter column categoria set data type enum ('CELULARES','INFORMATICA','LIVROS')
Feb 10, 2025 3:03:02 PM org.hibernate.tool.schema.internal.ExceptionHandlerLoggedImpl handleException
WARN: GenerationTarget encountered exception accepting command : Error executing DDL "alter table if exists produtos alter column categoria
org.hibernate.tool.schema.spi.CommandAcceptanceException Create breakpoint : Error executing DDL "alter table if exists produtos alter column ca

Caused by: org.h2.jdbc.JdbcSQLDataException: Value not permitted for column "('CELULARES', 'INFORMATICA', 'LIVROS')": "0"; SQL statement:
INSERT INTO "PUBLIC"."PRODUTOS_COPY_3_0"("ID", "DESCRICAO", "NOME", "PRECO", "CATEGORIA", "DATACADASTRO") OVERRIDING SYSTEM VALUE SELECT "I
    at org.h2.message.DbException.getJdbcSQLException(DbException.java:518)


Hibernate: insert into produtos (categoria,dataCadastro,descricao,nome,preco,id) values (?,?,?,?,?,default)
Feb 10, 2025 3:03:02 PM org.hibernate.engine.jdbc.spi.SqlExceptionHelper logExceptions
WARN: SQL Error: 22018, SQLState: 22018
Feb 10, 2025 3:03:02 PM org.hibernate.engine.jdbc.spi.SqlExceptionHelper logExceptions
ERROR: Data conversion error converting "'INFORMATICA' (PRODUTOS: ""CATEGORIA"" TINYINT)"; SQL statement:
insert into produtos (categoria,dataCadastro,descricao,nome,preco,id) values (?,?,?,?,?,default) [22018-232]
Exception in thread "main" org.hibernate.exception.DataException: could not execute statement [Data conversion error converting "'INFORM
insert into produtos (categoria,dataCadastro,descricao,nome,preco,id) values (?,?,?,?,?,default) [22018-232]] [insert into produtos (cat
```

Apagar o arquivo com o banco,
e rodar novamente.

Após apagar o banco:

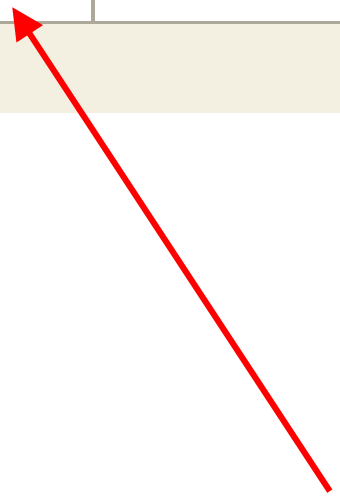
```
// Criando um objeto da Classe Produto:
Produto celular = new Produto("HP Deskjet 3776", "Cartucho não dá pra nada!",
                              new BigDecimal("750"), Categoria.INFORMATICA);
```

```
2024-02-16T11:11:38.545-03:00  INFO 11938 --- [          main] org.hibernate.orm.connections.pooling    : HHH10001115: Connection pool size: 20 (min=1)
2024-02-16T11:11:39.111-03:00  INFO 11938 --- [          main] o.h.m.i.EntityInstantiatorPojoStandard   : HHH000182: No default (no-argument) constructor for c
2024-02-16T11:11:39.192-03:00  INFO 11938 --- [          main] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000489: No JTA platform available (set 'hibernate.
2024-02-16T11:11:39.202-03:00  INFO 11938 --- [          main] org.hibernate.orm.connections.access     : HHH10001501: Connection obtained from JdbcConnectionA
Hibernate: create table produtos (id bigint generated by default as identity, categoria varchar(255) check (categoria in ('CELULARES','INFORMATICA','LIVROS')),
Hibernate: insert into produtos (categoria,dataCadastro,descricao,nome,preco,id) values (?,?,?,?,?,default)
```

SELECT * FROM PRODUTOS;

ID	CATEGORIA	DATACADASTRO	DESCRICAO	NOME	PRECO
1	INFORMATICA	2025-08-05	Cartucho não dá pra nada!	HP Deskjet 3776	750.00

(1 row, 1 ms)



SELECT * FROM PRODUTOS;

ID	CATEGORIA	DATACADASTRO	DESCRICAO	NOME	PRECO
1	INFORMATICA	2025-08-05	Cartucho não dá pra nada!	HP Deskjet 3776	750.00

(1 row, 1 ms)

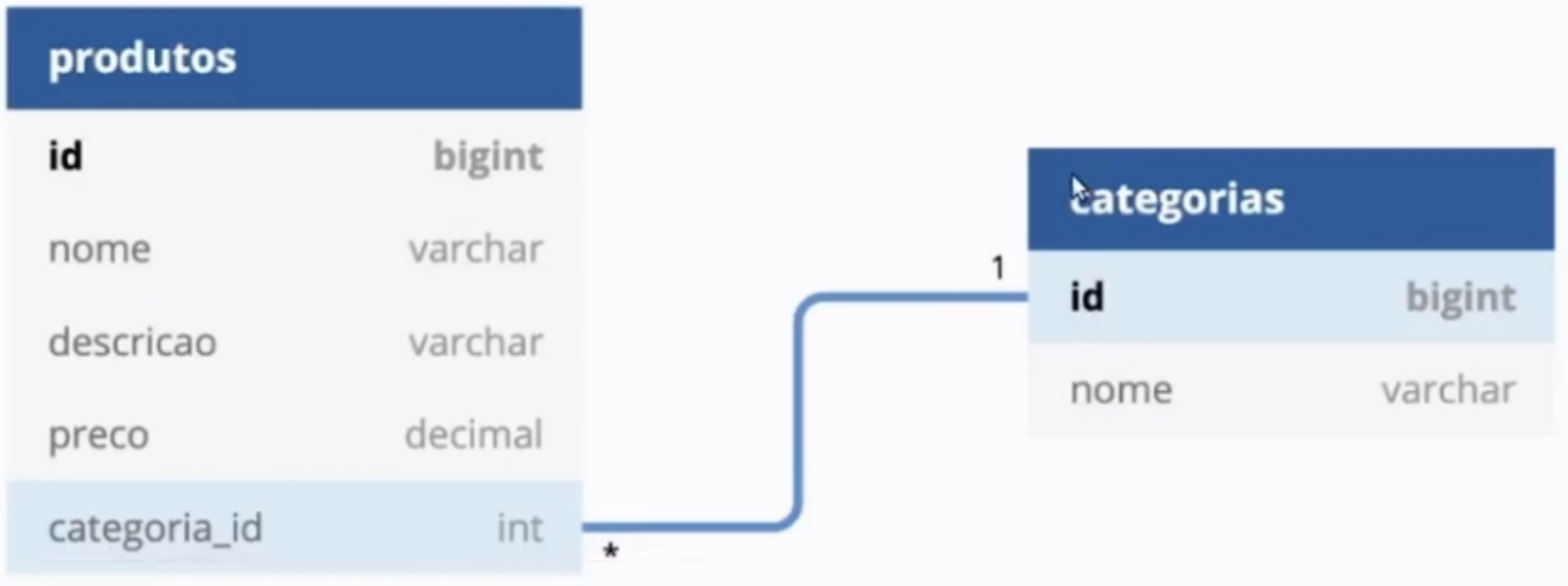
Legal...

Mas quando surgirem novas categorias,
teremos que mudar o código-fonte,
recompilar, disponibilizar...

Seria bom se as categorias também estivessem
no banco de dados.

Aí poderíamos ter recursos para
gerenciar categorias (criar, apagar, ...)

Isto é, ‘relacionamentos’ na JPA...


produtos
id bigint
nome varchar
descricao varchar
preco decimal
categoria_id int
categorias
id bigint
nome varchar
1
*

Classe Categoria (trocar de enum para classe/entidade)

```java
@Entity
@Table(name = "categorias")
public class Categoria {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nome;

    // Construtor:
    public Categoria(String nome) {
        this.nome = nome;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }
}
```

Colocar no persistence.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.2"
             xmlns="http://xmlns.jcp.org/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
                                 http://xmlns.jcp.org/xml/ns/persistence/persistence_2_2.xsd">

    <persistence-unit name="loja" transaction-type="RESOURCE_LOCAL">

        <class>br.edu.ifsp.carlao2005.modelo.Produto</class>
        <class>br.edu.ifsp.carlao2005.modelo.Categoria</class>

        <properties>

            <property name="jakarta.persistence.jdbc.driver" value="org.h2.Driver"/>
            <property name="jakarta.persistence.jdbc.url" value="jdbc:h2:file:./data/loja"/>
            <property name="jakarta.persistence.jdbc.user" value="sa" />
            <property name="jakarta.persistence.jdbc.password" value="" />

            <property name="hibernate.show_sql" value="true"/>
            <property name="hibernate.hbm2ddl.auto" value="update"/>

        </properties>

    </persistence-unit>

</persistence>
```

Alteração na classe Produto:

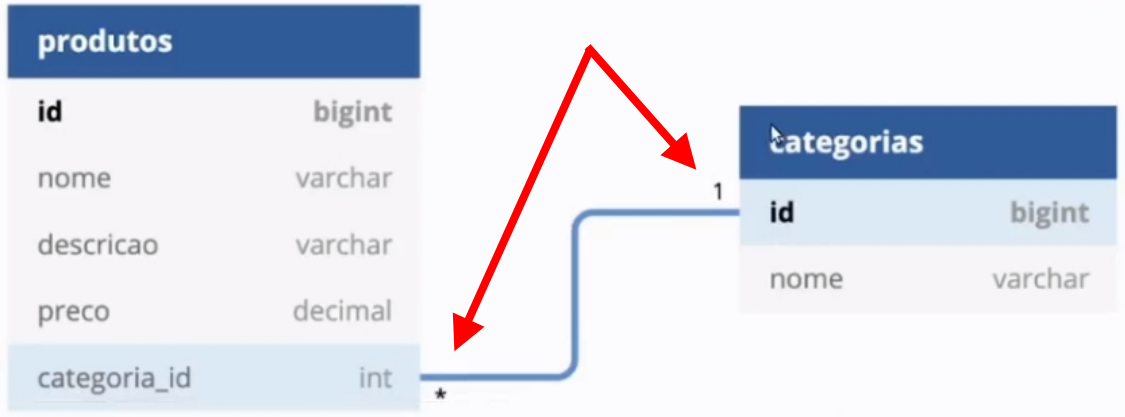
```
@Entity
@Table(name = "produtos")
public class Produto {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nome;
    private String descricao;
    private BigDecimal preco;

    // Atributo que representa a data de cadastro:
    private LocalDate dataCadastro = LocalDate.now();

    // Categoria não é mais um @Enumerated, retiramos a anotação.
    // Porém precisamos indicar a cardinalidade do relacionamento:
    @ManyToOne
    private Categoria categoria;
```

Alteração na classe Produto:

Como uma entidade tem como atributo uma classe que é outra entidade, a JPA já sabe que se trata de um relacionamento.

Somos obrigados a indicar qual é a cardinalidade deste relacionamento.

```
@Entity
@Table(name = "produtos")
public class Produto {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nome;
    private String descricao;
    private BigDecimal preco;

    // Atributo que representa a data de cadastro:
    private LocalDate dataCadastro = LocalDate.now();

    // Categoria não é mais um @Enumerated, retiramos a anotação.
    // Porém precisamos indicar a cardinalidade do relacionamento:
    @ManyToOne
    private Categoria categoria;
```



1. @ManyToOne: Essa anotação é usada para definir um relacionamento muitos-para-um entre duas entidades. Ela indica que a entidade atual possui uma associação com outra entidade e que vários registros da entidade atual podem se relacionar com um único registro da outra entidade.
2. @OneToMany: Essa anotação é usada para definir um relacionamento um-para-muitos entre duas entidades. Ela indica que a entidade atual possui uma coleção de outras entidades, onde um único registro da entidade atual está associado a vários registros da outra entidade.
3. @OneToOne: Essa anotação é usada para definir um relacionamento um-para-um entre duas entidades. Ela indica que a entidade atual possui uma associação com outra entidade, onde um único registro da entidade atual está associado a um único registro da outra entidade.
4. @ManyToMany: Essa anotação é usada para definir um relacionamento muitos-para-muitos entre duas entidades. Ela indica que vários registros da entidade atual podem se relacionar com vários registros da outra entidade.
5. @JoinColumn: Essa anotação é usada para especificar a coluna da tabela do banco de dados que é usada para manter o relacionamento entre as entidades. Ela é usada em conjunto com as anotações @ManyToOne, @OneToOne e @OneToMany.
6. @JoinTable: Essa anotação é usada em relacionamentos @ManyToMany para especificar uma tabela de associação separada que armazena as chaves estrangeiras das duas entidades que estão sendo associadas.
7. @MappedBy: Essa anotação é usada para indicar que o relacionamento já foi mapeado pela outra entidade e, portanto, não precisa ser mapeado novamente. Ela é usada em conjunto com as anotações @OneToMany e @OneToOne.

Veremos um pouco mais de detalhe desses relacionamentos mais a frente, já trabalhando com Spring...

Gravação:

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

    // Criando a Categoria "CELULARES":
    Categoria celulares = new Categoria("CELULARES");

    // Criando um objeto da Classe Produto, associado a categoria CELULARES:
    Produto celular = new Produto("XIAOMI sei la qual", "Xing Ling",
                                  new BigDecimal("1250"), celulares);

    // Gravando o produto:
    EntityManager em = JPAUtil.getEntityManager();
    ProdutoDao dao = new ProdutoDao(em);
    em.getTransaction().begin();
    dao.cadastrar(celular);
    em.getTransaction().commit();
    em.close();
}
```

Desconectei o console...

Apaguei o arquivo do banco...

Testando...

Gravação:

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

    // Criando a Categoria "CELULARES":
    Categoria celulares = new Categoria("CELULARES");

    // Criando um objeto da Classe Produto, associado a categoria CELULARES:
    Produto celular = new Produto("XIAOMI sei la qual", "Xing Ling",
                                  new BigDecimal("1250"), celulares);

    // Gravando o produto:
    EntityManager em = JPAUtil.getEntityManager();
    ProdutoDao dao = new ProdutoDao(em);
    em.getTransaction().begin();
    dao.cadastrar(celular);
    em.getTransaction().commit();
    em.close();
}
```

```
Hibernate: insert into produtos (categoria_id,dataCadastro,descricao,nome,preco) values (?,?,?,?,?)
Exception in thread "main" jakarta.persistence.RollbackException Create breakpoint : Error while committing the transaction
    at org.hibernate.internal.ExceptionConverterImpl.convertCommitException(ExceptionConverterImpl.java:65)
    at org.hibernate.engine.transaction.internal.TransactionImpl.commit(TransactionImpl.java:104)
    at br.edu.ifsp.carlao2005.testes.CadastroDeProduto.main(CadastroDeProduto.java:27)
Caused by: java.lang.IllegalStateException Create breakpoint : org.hibernate.TransientPropertyValueException: object referen
    at org.hibernate.internal.ExceptionConverterImpl.convert(ExceptionConverterImpl.java:152)
    at org.hibernate.internal.ExceptionConverterImpl.convert(ExceptionConverterImpl.java:162)
    at org.hibernate.internal.ExceptionConverterImpl.convert(ExceptionConverterImpl.java:168)
```

Gravação:

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

    // Criando a Categoria "CELULARES":
    Categoria celulares = new Categoria("CELULARES");

    // Criando um objeto da Classe Produto, associado a categoria CELULARES:
    Produto celular = new Produto("XIAOMI sei la qual", "Xing Ling",
                                  new BigDecimal("1250"), celulares);

    // Gravando o produto:
    EntityManager em = JPAUtil.getEntityManager();
    ProdutoDao dao = new ProdutoDao(em);
    em.getTransaction().begin();
    dao.cadastrar(celular);
    em.getTransaction().commit();
    em.close();
}
```

```
Hibernate: insert into produtos (categoria_id,dataCadastro,descricao,nome,preco) values (?,?,?,?,?)
Exception in thread "main" jakarta.persistence.RollbackException Create breakpoint : Error while committing the transaction
    at org.hibernate.internal.ExceptionConverterImpl.convertCommitException(ExceptionConverterImpl.java:65)
    at org.hibernate.engine.transaction.internal.TransactionImpl.commit(TransactionImpl.java:104)
    at br.edu.ifsp.carlao2005.testes.CadastroDeProduto.main(CadastroDeProduto.java:27)
Caused by: java.lang.IllegalStateException Create breakpoint : org.hibernate.TransientPropertyValueException: object referen
    at org.hibernate.internal.ExceptionConverterImpl.convert(ExceptionConverterImpl.java:152)
    at org.hibernate.internal.ExceptionConverterImpl.convert(ExceptionConverterImpl.java:162)
    at org.hibernate.internal.ExceptionConverterImpl.convert(ExceptionConverterImpl.java:168)
```

Nós tentamos gravar um Produto no BD,
relacionado com uma categoria
que ainda não foi gravada no BD!!!!

Criamos o objeto Categoria na memória,
porém precisamos gravar ele no BD
antes de poder gravar o Produto.

Classe CategoriaDao

```
public class CategoriaDao {

    // EntityManager, que será usado por todos os métodos:
    private EntityManager em;

    // Construtor que já recebe o EntityManager criado:
    public CategoriaDao(EntityManager em) {
        this.em = em;
    }

    // Método para gravar uma categoria no BD:
    public void cadastrar(Categoria categoria) {
        this.em.persist(categoria);
    }
}
```

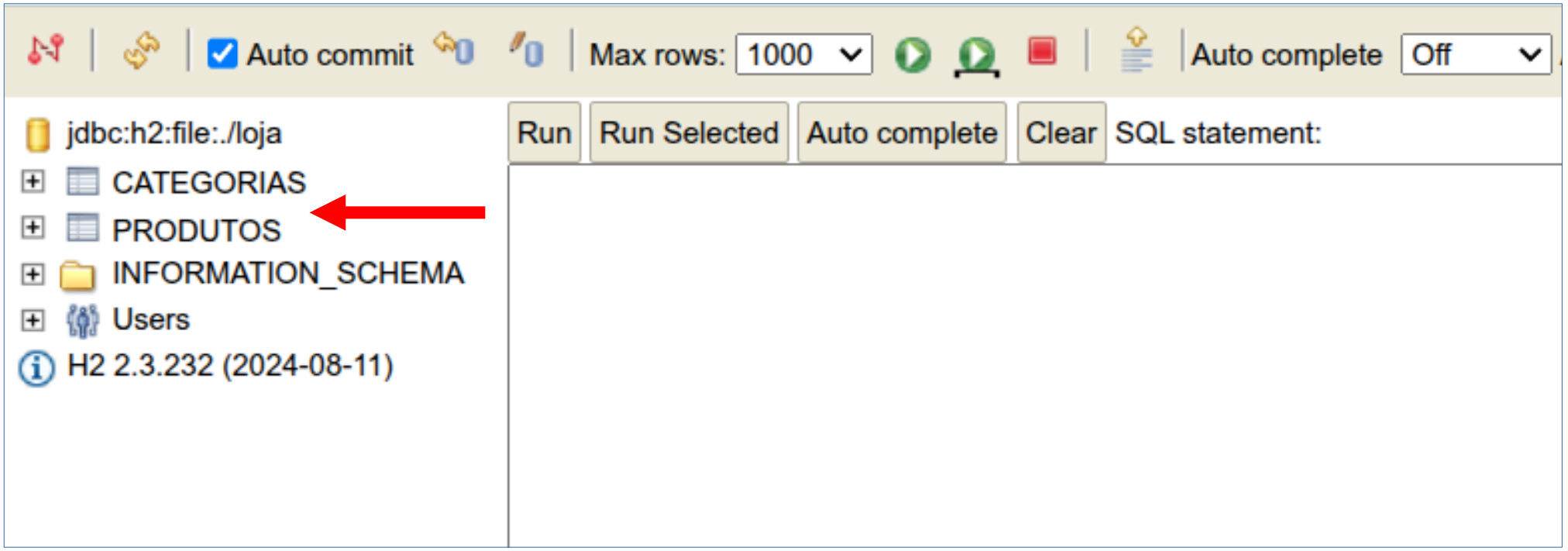
Alterando a gravação, agora categoria e produto:

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

    // Criando a Categoria "CELULARES":
    Categoria celulares = new Categoria("CELULARES");

    // Criando um objeto da Classe Produto, associado a categoria CELULARES:
    Produto celular = new Produto("XIAOMI sei la qual", "Xing Ling",
                                  new BigDecimal("1250"), celulares);

    // Gravando Categoria e Produto:

    EntityManager em = JPAUtil.getEntityManager();

    // Vamos compartilhar o mesmo EntityManager com as várias classes DAO:
    ProdutoDao produtoDao = new ProdutoDao(em);
    CategoriaDao categoriaDao = new CategoriaDao(em);

    em.getTransaction().begin();

    categoriaDao.cadastrar(celulares);
    produtoDao.cadastrar(celular);

    em.getTransaction().commit();

    em.close();
}
```

Apaguei os arquivos do banco...
Testando...

```
2024-02-16T11:27:07.668-03:00  INFO 12558 --- [          main] o.h.m.i.EntityInstantiatorPojoStandard   : HHH000182: No default (no-a
2024-02-16T11:27:07.742-03:00  INFO 12558 --- [          main] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000489: No JTA platform
2024-02-16T11:27:07.751-03:00  INFO 12558 --- [          main] org.hibernate.orm.connections.access     : HHH10001501: Connection obt
Hibernate: create table categorias (id bigint generated by default as identity, nome varchar(255), primary key (id))
Hibernate: create table produtos (id bigint generated by default as identity, dataCadastro date, descricao varchar(255), nome varchar(
Hibernate: alter table if exists produtos add constraint FK8rqw0ljwdaom34jr2t46bjtrn foreign key (categoria_id) references categorias
Hibernate: insert into categorias (nome,id) values (?,default)
Hibernate: insert into produtos (categoria_id,dataCadastro,descricao,nome,preco,id) values (?,?,?,?,?,default)
```

Tabelas criadas:



SELECT * FROM CATEGORIAS;

ID	NOME
1	CELULARES

(1 row, 1 ms)

SELECT * FROM PRODUTOS;

ID	DATACADASTRO	DESCRICAO	NOME	PRECO	CATEGORIA_ID
1	2025-08-05	Xing Ling	XIAOMI sei la qual	1250.00	1

(1 row, 0 ms)

Ciclo de vida de uma entidade na JPA

Antes: sobre transações...

(matéria de BD2, mas vamos entender umas coisas...)

Transação

- No contexto de banco de dados, uma transação é uma unidade lógica de trabalho que consiste em uma sequência de uma ou mais operações de banco de dados.
  - Essas operações podem incluir inserções, atualizações, exclusões e consultas de dados.
  - O **principal objetivo** de uma transação é garantir que todas as operações sejam tratadas como uma única unidade indivisível, ou seja, **todas as operações devem ser executadas com sucesso como um todo, ou todas devem ser revertidas (desfeitas) caso ocorra algum erro.**
- Uma transação possui quatro propriedades importantes, conhecidas como as propriedades ACID:
  - Atomicidade, Consistência, Isolamento, Durabilidade ...
- Em resumo, uma transação é uma sequência de operações de BD que garantem a integridade, consistência e confiabilidade dos dados. Ela **oferece a segurança de que todas as operações ocorrerão como um todo bem-sucedido ou serão desfeitas em caso de problemas,** mantendo o banco de dados em um estado consistente e válido.

Ciclo de vida de uma transação: ações possíveis

- **Iniciar (Begin)**: O primeiro passo é iniciar a transação explicitamente. Isso é feito chamando o método apropriado no contexto de persistência.
- **Realizar operações no banco de dados**: Dentro da transação, você pode executar várias operações de banco de dados, como inserções (persistência), atualizações (merge), exclusões (remove) e consultas. Todas essas operações serão tratadas como uma única unidade durante a transação.
- **Commit (Confirmar)**: Se todas as operações foram executadas com sucesso e você deseja tornar as mudanças permanentes, você pode chamar o método commit(). Isso confirmará a transação e todas as alterações serão aplicadas permanentemente no banco de dados.
- **Rollback (Reverter)**: Se ocorrer algum erro durante a transação ou se você decidir desfazer as alterações feitas, pode chamar o método rollback(). Isso reverterá todas as operações feitas dentro da transação, e o banco de dados retornará ao estado que estava antes do início da transação.

commit()

x

flush()

commit()

- Quando você chama `getTransaction().begin()`, inicia uma nova transação.
- Durante essa transação, você realiza operações de banco de dados (persistência, atualização, remoção, etc.) através do EntityManager.
- Quando você chama `getTransaction().commit()`, **a transação é finalizada** e as **alterações feitas durante a transação são confirmadas** e tornam-se permanentes no banco de dados.
- Após o `commit()`, **a transação é encerrada** e as alterações não podem ser revertidas. Elas se tornam permanentes no banco de dados.

flush()

- Sincroniza o contexto de persistência com o banco de dados, ou seja, **envia todas as alterações pendentes ao banco de dados.**
- O `flush()` **não finaliza a transação**; ele apenas garante que **as mudanças** feitas no contexto de persistência **são refletidas no banco de dados até esse ponto** da transação.
- Isso significa que, mesmo após o `flush()`, **você ainda está dentro da mesma transação** e pode **continuar realizando operações no banco de dados.**
- Se você chamar `flush()` e posteriormente ocorrer um `rollback()` na transação, **as alterações feitas até o ponto do `flush()` serão revertidas**, pois ainda estão dentro da mesma transação não confirmada.

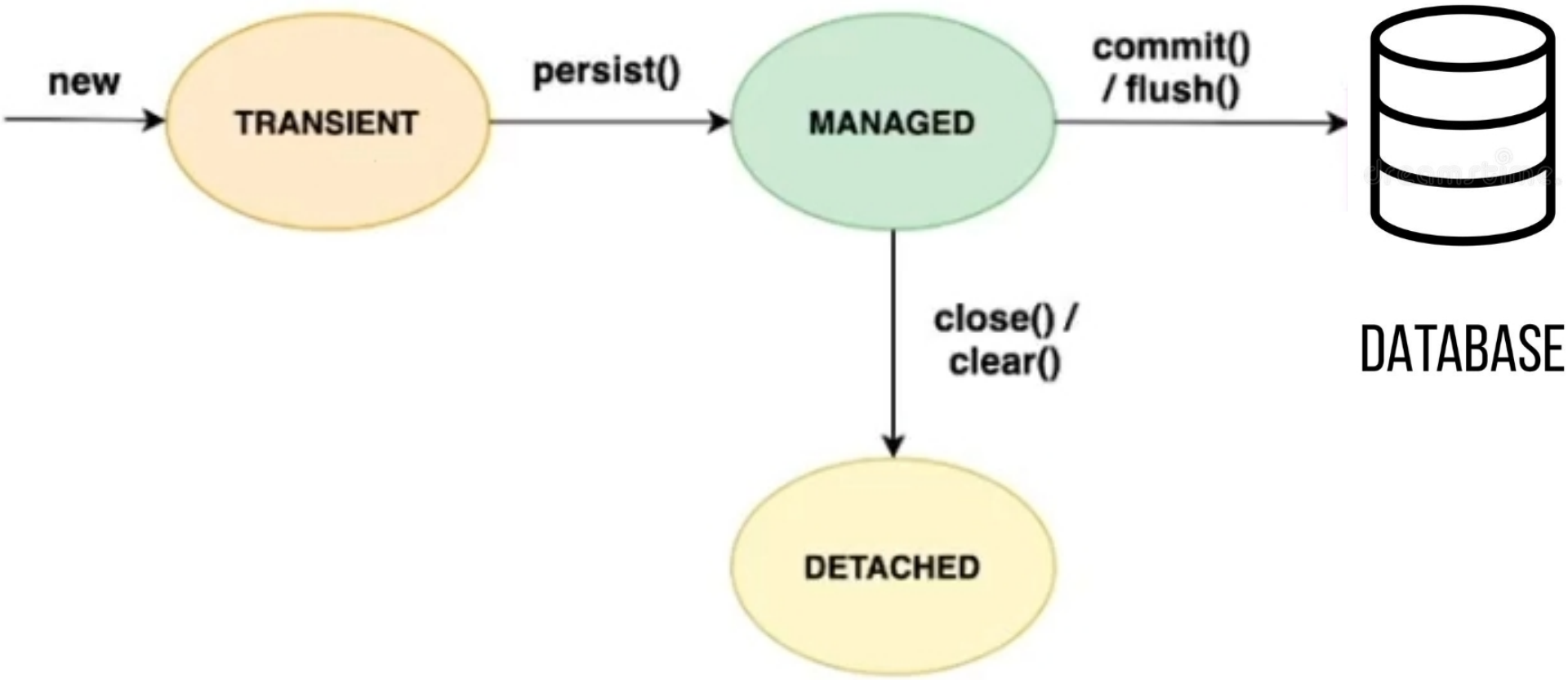
Em resumo,

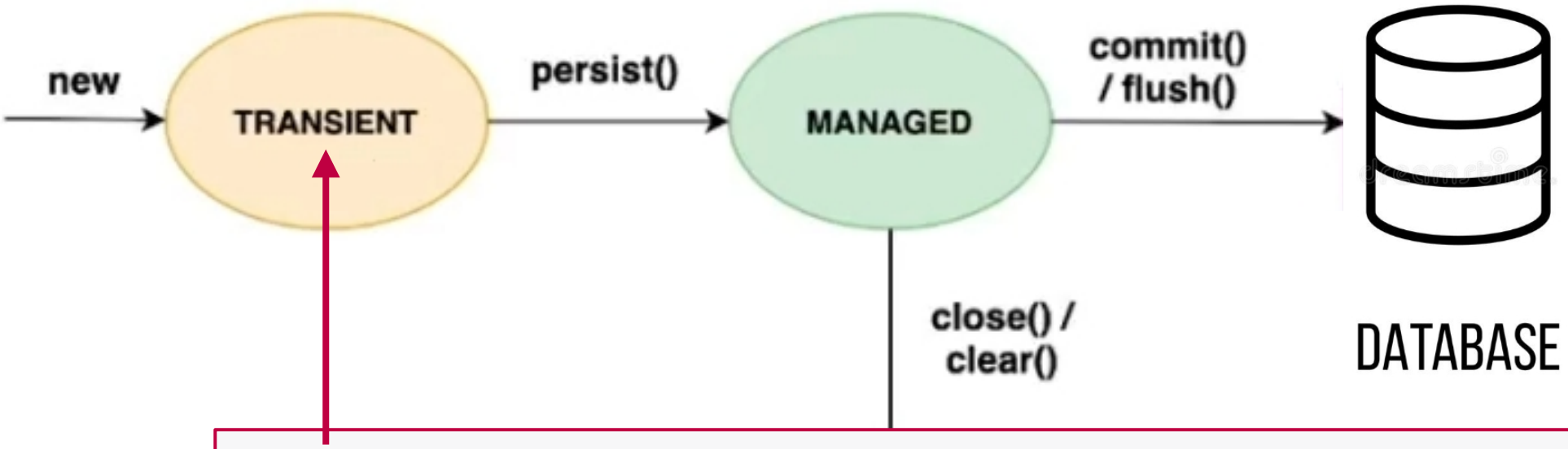
o **commit()** finaliza completamente a transação e torna as alterações permanentes no banco de dados,

enquanto o **flush()** sincroniza o contexto de persistência com o banco de dados dentro da mesma transação, mas não finaliza a transação.

Agora sim...

Ciclo de vida de uma entidade na JPA!


new
TRANSIENT
persist()
MANAGED
commit()
/ flush()
DATABASE
close() /
clear()
DETACHED



O estado transient ocorre quando uma entidade foi criada como um novo objeto Java, mas ainda não está associada a nenhum contexto de persistência. Isso significa que a JPA não está ciente da existência dessa entidade, e ela ainda não foi persistida no banco de dados. Geralmente, esse é o estado inicial de uma entidade que acabou de ser criada usando o operador "new" em Java.

Quando uma entidade está no estado transient, ela não é gerenciada pelo EntityManager (uma classe central da JPA para gerenciamento de entidades), e quaisquer alterações feitas em seus atributos não serão refletidas no banco de dados.

ATENÇÃO!

A partir de agora vou fazer
pequenos testes, com trechos de códigos.
Quase que alterando o main totalmente.

Não vou apagar os testes,
vou deixar comentados,
para disponibilizar o projeto completo
no final.

E quando você estiver refazendo esses testes,
não esqueça de desconectar o console do h2
a cada teste!

No exemplo a seguir,
vou criar um objeto java
(uma nova categoria)
com new().

Mas como ele é “transient”
(não teve um “persist”),
não será gravado no banco.

Exemplo (apaguei código anterior no main)

```
// Criando a Categoria "INFORMATICA".
Categoria informatica = new Categoria("INFORMATICA");

// Criando o EntityManager:
EntityManager em = JPAUtil.getEntityManager();

// Criando o CategoriaDAO:
CategoriaDao categoriaDao = new CategoriaDao(em);

// Iniciando a transação:
em.getTransaction().begin();

// Finalizando a transação:
em.getTransaction().commit();

// Fechando o EntityManager
em.close();
```

Exemplo (apaguei código anterior no main)

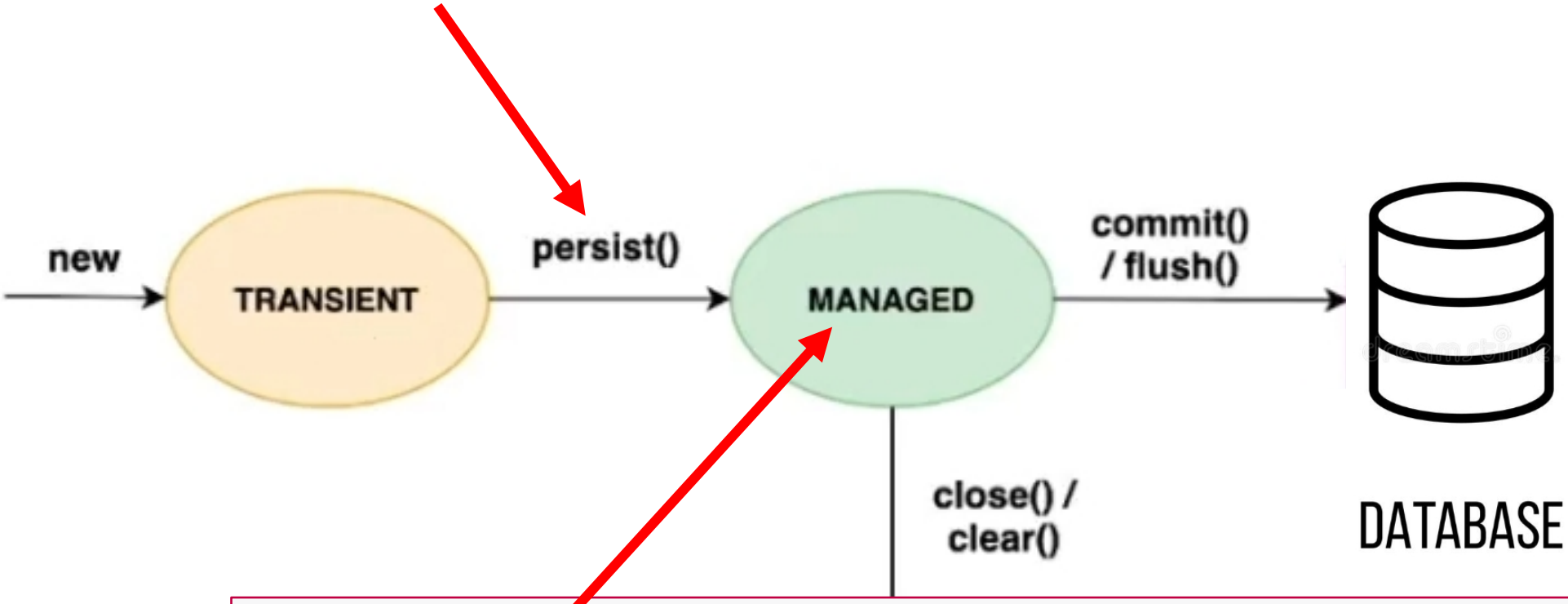
```
// Criando a Categoria "INFORMATICA".
Categoria informatica = new Categoria("INFORMATICA");

// Criando o EntityManager:
EntityManager em = JPAUtil.getEntityManager();

// Criando o CategoriaDAO:
CategoriaDao categoriaDao = new CategoriaDao(em);

// Iniciando a transação:
em.getTransaction().begin();

// Finalizando a transação:
em.getTransaction().commit();

// Fechando o EntityManager
em.close();
```

Não gravou “INFORMATICA”...

SELECT * FROM CATEGORIAS;

ID	NOME
1	CELULARES

(1 row, 1 ms)



Quando uma entidade está no estado "managed", significa que ela está sendo rastreada pela JPA, e qualquer alteração feita em seus atributos será sincronizada automaticamente com o banco de dados quando ocorrer uma transação de gravação (commit).

A transição para o estado "managed" ocorre quando uma entidade é associada a um contexto de persistência por meio do método "persist()" ou quando é recuperada do banco de dados usando operações como "find()" ou "createQuery()".

Enquanto uma entidade estiver no estado "managed", a JPA cuidará automaticamente de todas as mudanças feitas em seus atributos, garantindo que essas mudanças sejam refletidas no banco de dados sem que seja necessário escrever código adicional para realizar a sincronização manualmente.

Outro exemplo:
Criar a categoria “informática”,
persistir no banco,
e simplesmente mudar o nome
no objeto.

A mudança será refletida automaticamente
no registro no banco!

Exemplo

```
// Criando a Categoria "INFORMATICA".
Categoria informatica = new Categoria("INFORMATICA");

// Criando o EntityManager:
EntityManager em = JPAUtil.getEntityManager();

// Criando o CategoriaDAO:
CategoriaDao categoriaDao = new CategoriaDao(em);

// Iniciando a transação:
em.getTransaction().begin();

// Mudando para o estado managed:
em.persist(informatica);

// Trocando o texto no objeto:
informatica.setNome("INFORMATICA & COMPUTAÇÃO");

// Finalizando a transação:
em.getTransaction().commit();

// Fechando o EntityManager
em.close();
```

Exemplo

```
// Criando a Categoria "INFORMATICA".
Categoria informatica = new Categoria("INFORMATICA");

// Criando o EntityManager:
EntityManager em = JPAUtil.getEntityManager();

// Criando o CategoriaDAO:
CategoriaDao categoriaDao = new CategoriaDao(em);

// Iniciando a transação:
em.getTransaction().begin();

// Mudando para o estado managed:
em.persist(informatica);

// Trocando o texto no objeto:
informatica.setNome("INFORMATICA & COMPUTAÇÃO");

// Finalizando a transação:
em.getTransaction().commit();

// Fechando o EntityManager
em.close();
```

```
INFO: HHH10001501: Connection obtained from JdbcConn
Hibernate: insert into categorias (nome) values (?)
Hibernate: update categorias set nome=? where id=?

Process finished with exit code 0
```

Exemplo

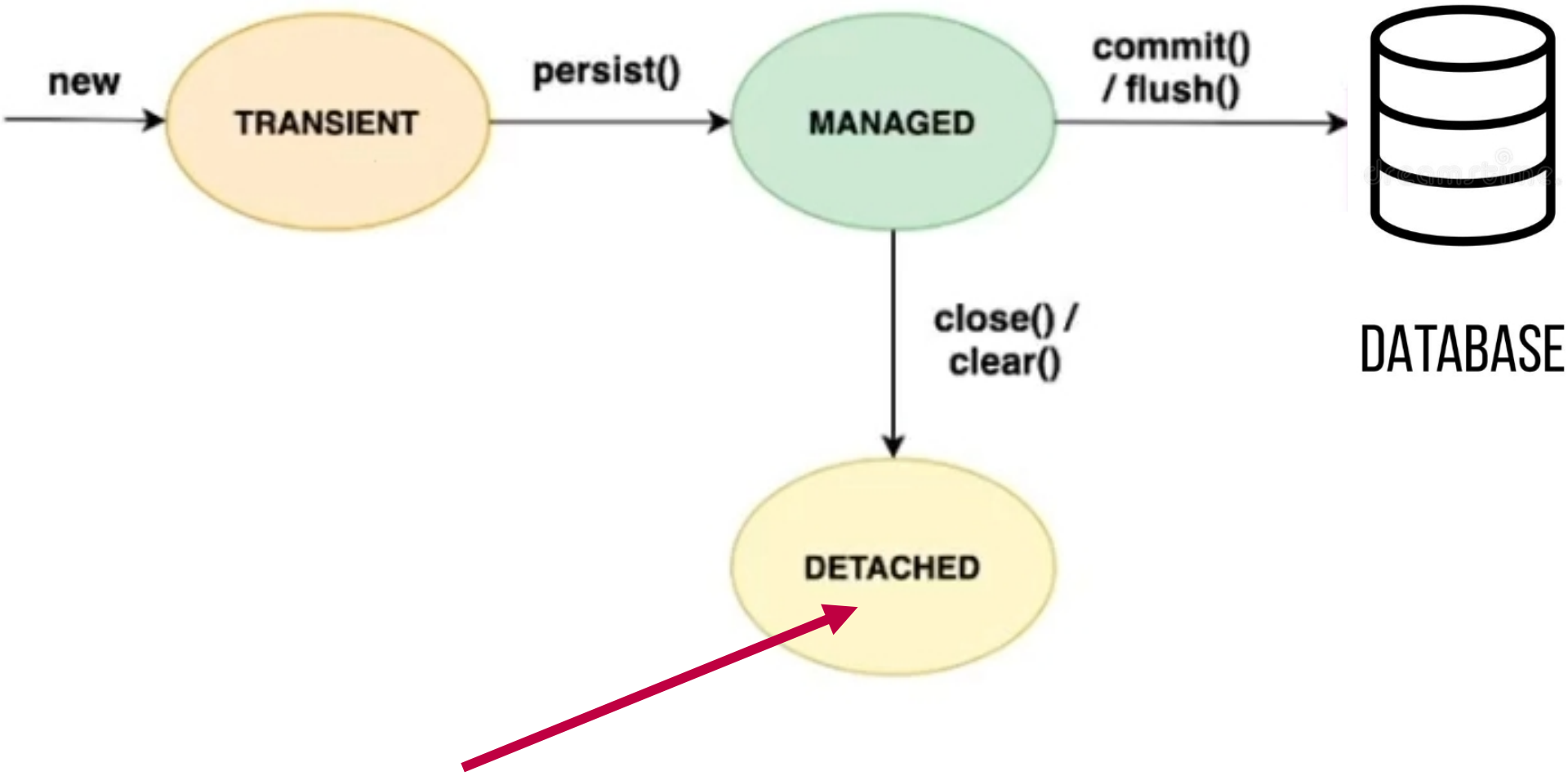
```
// Criando a Categoria "INFORMATICA".
Categoria informatica = new Categoria("INFORMATICA");

// Criando o EntityManager:
EntityManager em = JPAUtil.getEntityManager();

// Criando o CategoriaDAO:
CategoriaDao categoriaDao = new CategoriaDao(em);

// Iniciando a transação:
em.getTransaction().begin();

// Mudando para o estado managed:
em.persist(informatica);

// Trocando o texto no objeto:
informatica.setNome("INFORMATICA & COMPUTAÇÃO");

// Finalizando a transação:
em.getTransaction().commit();

// Fechando o EntityManager
em.close();
```

```
INFO: HHH10001501: Connection obtained from JdbcConn
Hibernate: insert into categorias (nome) values (?)
Hibernate: update categorias set nome=? where id=?

Process finished with exit code 0
```

SELECT * FROM CATEGORIAS;

ID	NOME
1	CELULARES
2	INFORMATICA & COMPUTAÇÃO

(2 rows, 1 ms)


new
TRANSIENT
persist()
MANAGED
commit()
/ flush()
DATABASE
close() /
clear()
DETACHED

Detach

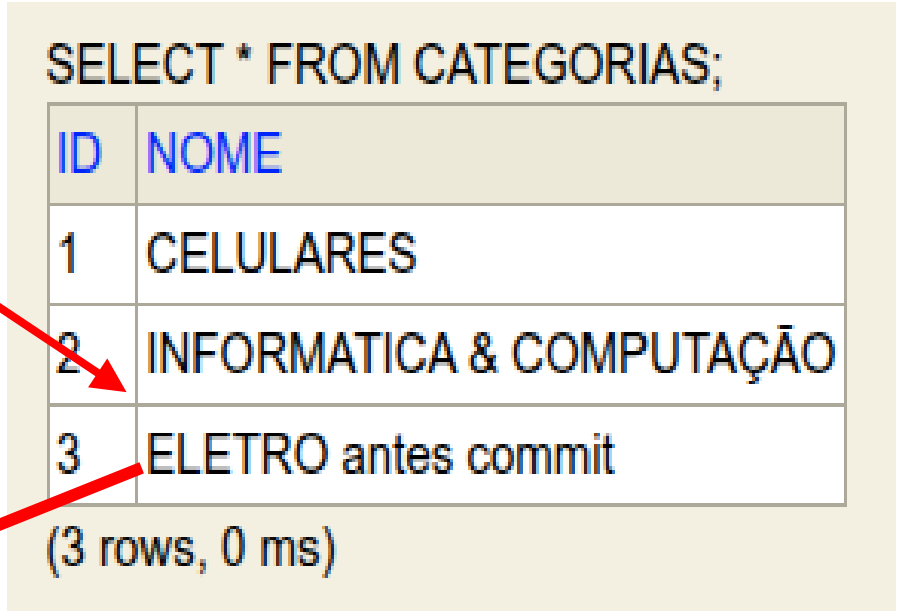
- Esse estado ocorre quando uma entidade já foi gerenciada pela JPA em algum momento, mas não está mais associada a um contexto de persistência.
  - Em outras palavras, quando uma entidade está no estado detached, ela não está mais sendo rastreada pela JPA e não está vinculada a nenhum EntityManager. Isso geralmente ocorre quando uma entidade que estava no estado "managed" é removida do contexto de persistência devido ao **término de uma transação**, fechamento do EntityManager ou explicitamente por meio do método "detach()" do EntityManager.
  - No estado detached, a JPA não rastreia mais as alterações feitas nos atributos da entidade, e quaisquer mudanças realizadas não serão sincronizadas automaticamente com o banco de dados.
  - É importante observar que uma entidade detached ainda representa um registro existente no banco de dados, e suas informações persistem no estado em que estavam quando saíram do estado "managed". Se você quiser salvar as mudanças feitas em uma entidade detached, você tem algumas opções:
    - Reanexar à JPA : ... usando o método "merge()" ...
    - Copiar os valores manualmente: ...
    - Reabrir a transação original: ...
  - Lembre-se de que o estado detached é um estado transitório, e as entidades podem ser movidas entre os estados "transient", "managed" e "detached" durante a execução de um aplicativo JPA ...

Exemplo

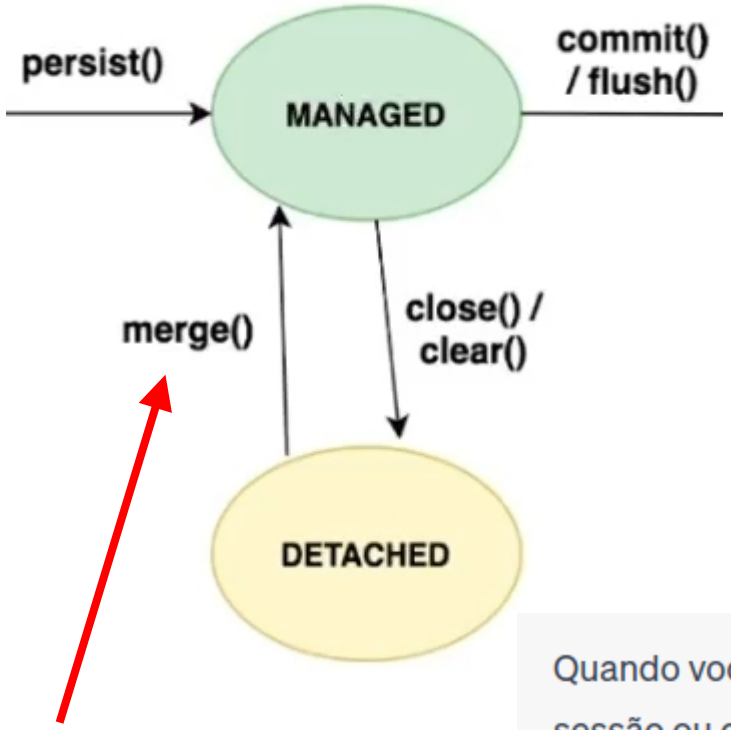
```
// Criando a Categoria "ELETRO".
Categoria informatica = new Categoria("ELETRO");

EntityManager em = JPAUtil.getEntityManager();
CategoriaDao categoriaDao = new CategoriaDao(em);

// Iniciando a transação:
em.getTransaction().begin();

// Mudando para o estado managed:
em.persist(informatica);

// Trocando o texto no objeto:
informatica.setNome("ELETRO antes commit");

// Finalizando a transação:
em.getTransaction().commit();

// Fechando o EntityManager
em.close();

// Trocando o texto no objeto:
informatica.setNome("ELETRO depois close");
```

Exemplo

```
// Criando a Categoria "ELETRO".
Categoria informatica = new Categoria("ELETRO");

EntityManager em = JPAUtil.getEntityManager();
CategoriaDao categoriaDao = new CategoriaDao(em);

// Iniciando a transação:
em.getTransaction().begin();

// Mudando para o estado managed:
em.persist(informatica);

// Trocando o texto no objeto:
informatica.setNome("ELETRO antes commit");

// Finalizando a transação:
em.getTransaction().commit();

// Fechando o EntityManager
em.close();

// Trocando o texto no objeto:
informatica.setNome("ELETRO depois close");
```



Voltando de detached para managed (para alterar)



Quando você obtém uma entidade do banco de dados através do EntityManager e fecha a sessão ou o contexto de persistência, essa entidade se torna desconectada. Isso significa que quaisquer alterações feitas nessa entidade após o fechamento do EntityManager não serão rastreadas automaticamente e refletidas no banco de dados.

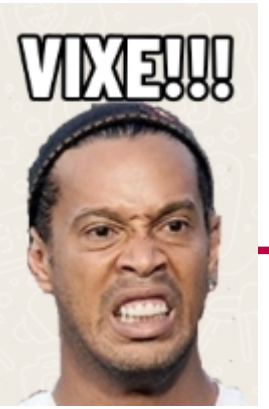
Ao utilizar o método `merge()`, você pode reconectar essa entidade desconectada ao contexto de persistência e fazer com que quaisquer alterações feitas na entidade sejam propagadas para o banco de dados quando a transação for confirmada.

Exemplo

```
// Criando a Categoria "ESPORTE":
Categoria esporte = new Categoria("ESPORTE");

EntityManager em = JPAUtil.getEntityManager();
CategoriaDao categoriaDao = new CategoriaDao(em);

// Iniciando a transação:
em.getTransaction().begin();

// Mudando para o estado managed:
em.persist(esporte);

// Trocando o texto no objeto:
esporte.setNome("ESPORTE atualizado");

// Não vamos fechar a transação....
// Vamos atualizar com flush():
em.flush();

// Vamos tornar 'esporte' detached:
em.clear();

// Precisamos agora alterar o nome... Como voltar de detached para managed?
// Usamos o método merge().
// Atenção! Não basta usar o método, precisa "renovar" a referência!
esporte = em.merge(esporte);

// Trocando o texto no objeto:
esporte.setNome("ESPORTE depois de merge");

// Finalizando a transação:
em.getTransaction().commit();

// Fechando o EntityManager
em.close();
```

Exemplo

```java
// Criando a Categoria "ESPORTE":
Categoria esporte = new Categoria("ESPORTE");

EntityManager em = JPAUtil.getEntityManager();
CategoriaDao categoriaDao = new CategoriaDao(em);

// Iniciando a transação:
em.getTransaction().begin();

// Mudando para o estado managed:
em.persist(esporte);

// Trocando o texto no objeto:
esporte.setNome("ESPORTE atualizado");

// Não vamos fechar a transação....
// Vamos atualizar com flush():
em.flush();

// Vamos tornar 'esporte' detached:
em.clear();
```




```
Hibernate: insert into categorias (nome) values (?)
Hibernate: update categorias set nome=? where id=?
Hibernate: select c1_0.id,c1_0.nome from categorias c1_0 where c1_0.id=?
jul. 30, 2023 11:20:19 PM org.hibernate.event.internal.DefaultLoadEventListener doOnLoad
INFO: HHH000327: Error performing load command
org.hibernate.InstantiationException Create breakpoint : No default constructor for entity : br.edu.ifsp.carlao2005.modelo.Categoria
    at org.hibernate.metamodel.internal.EntityInstantiatorPojoStandard.instantiate(EntityInstantiatorPojoStandard.java:93)
    at org.hibernate.persister.entity.AbstractEntityPersister.instantiate(AbstractEntityPersister.java:4297)
    at org.hibernate.internal.SessionImpl.instantiate(SessionImpl.java:1472)
    at org.hibernate.sql.results.graph.entity.AbstractEntityInitializer.instantiateEntity(AbstractEntityInitializer.java:676)
    at org.hibernate.sql.results.graph.entity.AbstractEntityInitializer.resolveEntityInstance(AbstractEntityInitializer.java:669)
    at org.hibernate.sql.results.graph.entity.AbstractEntityInitializer.resolveInstance(AbstractEntityInitializer.java:635)
    at org.hibernate.sql.results.graph.entity.AbstractEntityInitializer.resolveEntityInstance(AbstractEntityInitializer.java:529)
```

A JPA exige que se tenha
um construtor ‘default’ (vazio),
para poder **ler** dados do BD.

Não tinha dado problema ainda porque
até agora não tivemos que fazer leitura no BD.

O comando merge faz leitura do objeto no BD,
para torná-lo novamente “managed”
e podermos alterar e gravar novamente.

```
 c1_0 where c1_0.id=?
nal.DefaultLoadEventListener doOnLoad

 No default constructor for entity : br.edu.ifsp.carlao2005.modelo.Categoria
ntiatorPojoStandard.instantiate(EntityInstantiatorPojoStandard.java:93)
yPersister.instantiate(AbstractEntityPersister.java:4297)
te(SessionImpl.java:1472)
actEntityInitializer.instantiateEntity(AbstractEntityInitializer.java:676)
actEntityInitializer.resolveEntityInstance(AbstractEntityInitializer.java:669)
```

Vamos colocar um construtor default
nas duas classes de modelo
que temos até agora

(Categoria e Produto)

Categoria

```java
@Entity
@Table(name = "categorias")
public class Categoria {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nome;

    // Construtor default:
    public Categoria() {}

    // Construtor:
    public Categoria(String nome) {
        this.nome = nome;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }
}
```

Produto

```java
@Entity
@Table(name = "produtos")
public class Produto {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nome;
    private String descricao;
    private BigDecimal preco;
    // Atributo que representa a data de cadastro:
    private LocalDate dataCadastro = LocalDate.now();
    // Precisamos indicar a cardinalidade do relacionamento:
    @ManyToOne
    private Categoria categoria;

    // Construtor default:
    public Produto() {}

    // Construtor (não precisa id nem dataCadastro):
    public Produto(String nome,
                   String descricao,
                   BigDecimal preco,
                   Categoria categoria) {
        ...
```

Executando novamente...

Exemplo

```
// Criando a Categoria "ESPORTE":
Categoria esporte = new Categoria("ESPORTE");

EntityManager em = JPAUtil.getEntityManager();
CategoriaDao categoriaDao = new CategoriaDao(em);

// Iniciando a transação:
em.getTransaction().begin();

// Mudando para o estado managed:
em.persist(esporte);

// Trocando o texto no objeto:
esporte.setNome("ESPORTE atualizado");

// Não vamos fechar a transação....
// Vamos atualizar com flush():
em.flush();

// Vamos tornar 'esporte' detached:
em.clear();

// Precisamos agora alterar o nome... Como voltar de detached para managed?
// Usamos o método merge().
// Atenção! Não basta usar o método, precisa "renovar" a referência!
esporte = em.merge(esporte);

// Trocando o texto no objeto:
esporte.setNome("ESPORTE depois de merge");

// Finalizando a transação:
em.getTransaction().commit();

// Fechando o EntityManager
em.close();
```

```
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess [org.hibe
Hibernate: insert into categorias (nome) values (?)
Hibernate: update categorias set nome=? where id=?
Hibernate: select c1_0.id,c1_0.nome from categorias c1_0 where c1_0.id=?
Hibernate: update categorias set nome=? where id=?

Process finished with exit code 0
```

Exemplo

```
// Criando a Categoria "ESPORTE":
Categoria esporte = new Categoria("ESPORTE");

EntityManager em = JPAUtil.getEntityManager();
CategoriaDao categoriaDao = new CategoriaDao(em);

// Iniciando a transação:
em.getTransaction().begin();

// Mudando para o estado managed:
em.persist(esporte);

// Trocando o texto no objeto:
esporte.setNome("ESPORTE atualizado");

// Não vamos fechar a transação....
// Vamos atualizar com flush():
em.flush();

// Vamos tornar 'esporte' detached:
em.clear();

// Precisamos agora alterar o nome... Como voltar de detached
// Usamos o método merge().
// Atenção! Não basta usar o método, precisa "renovar" a refe
esporte = em.merge(esporte);

// Trocando o texto no objeto:
esporte.setNome("ESPORTE depois de merge");

// Finalizando a transação:
em.getTransaction().commit();

// Fechando o EntityManager
em.close();
```

```
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess [org.hibe
Hibernate: insert into categorias (nome) values (?)
Hibernate: update categorias set nome=? where id=?
Hibernate: select c1_0.id,c1_0.nome from categorias c1_0 where c1_0.id=?
Hibernate: update categorias set nome=? where id=?

Process finished with exit code 0
```

SELECT * FROM CATEGORIAS;

ID	NOME
1	CELULARES
2	INFORMATICA & COMPUTAÇÃO
3	ELETRO antes commit
5	ESPORTE depois de merge

(4 rows, 1 ms)

Para apagar um objeto managed:

método remove()

Exemplo:

```
// Criando a Categoria "ESPORTE2":
Categoria esporte = new Categoria("ESPORTE2");

EntityManager em = JPAUtil.getEntityManager();
CategoriaDao categoriaDao = new CategoriaDao(em);

// Iniciando a transação:
em.getTransaction().begin();

// Mudando para o estado managed:
em.persist(esporte);

// Trocando o texto no objeto:
esporte.setNome("ESPORTE2 atualizado");

// Apagando...
em.remove(esporte);

// Finalizando a transação:
em.getTransaction().commit();

// Fechando o EntityManager
em.close();
```

Exemplo:

```java
// Criando a Categoria "ESPORTE2":
Categoria esporte = new Categoria("ESPORTE2");

EntityManager em = JPAUtil.getEntityManager();
CategoriaDao categoriaDao = new CategoriaDao(em);

// Iniciando a transação:
em.getTransaction().begin();

// Mudando para o estado managed:
em.persist(esporte);

// Trocando o texto no objeto:
esporte.setNome("ESPORTE2 atualizado");

// Apagando...
em.remove(esporte);

// Finalizando a transação:
em.getTransaction().commit();

// Fechando o EntityManager
em.close();
```

```
INFO: HHH10001501: Connection obtained from JdbcConn
Hibernate: insert into categorias (nome) values (?)
Hibernate: delete from categorias where id=?

Process finished with exit code 0
```

Exemplo:

```
// Criando a Categoria "ESPORTE2":
Categoria esporte = new Categoria("ESPORTE2");

EntityManager em = JPAUtil.getEntityManager();
CategoriaDao categoriaDao = new CategoriaDao(em);

// Iniciando a transação:
em.getTransaction().begin();

// Mudando para o estado managed:
em.persist(esporte);

// Trocando o texto no objeto:
esporte.setNome("ESPORTE2 atualizado");

// Apagando...
em.remove(esporte);

// Finalizando a transação:
em.getTransaction().commit();

// Fechando o EntityManager
em.close();
```

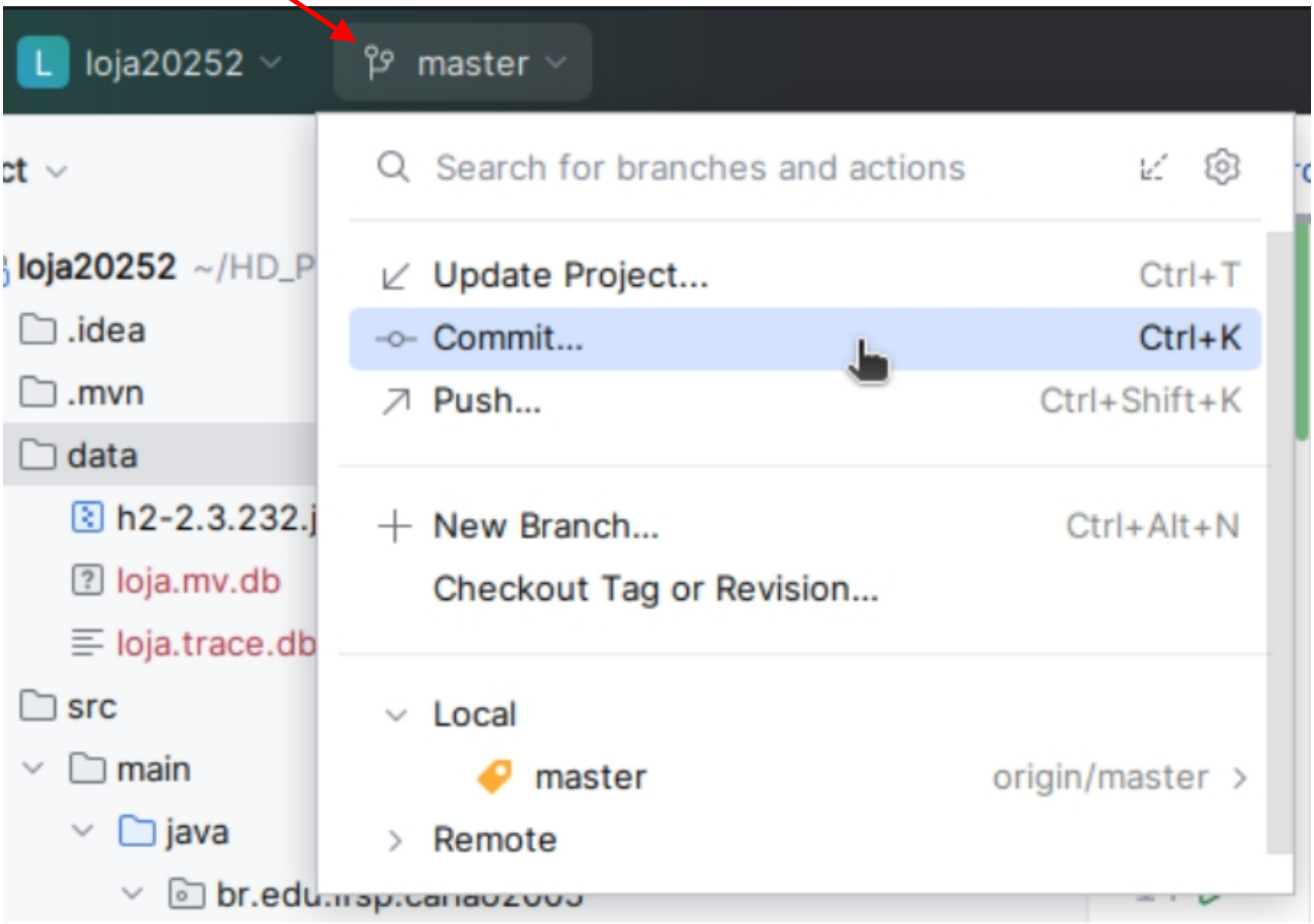
SELECT * FROM CATEGORIAS;

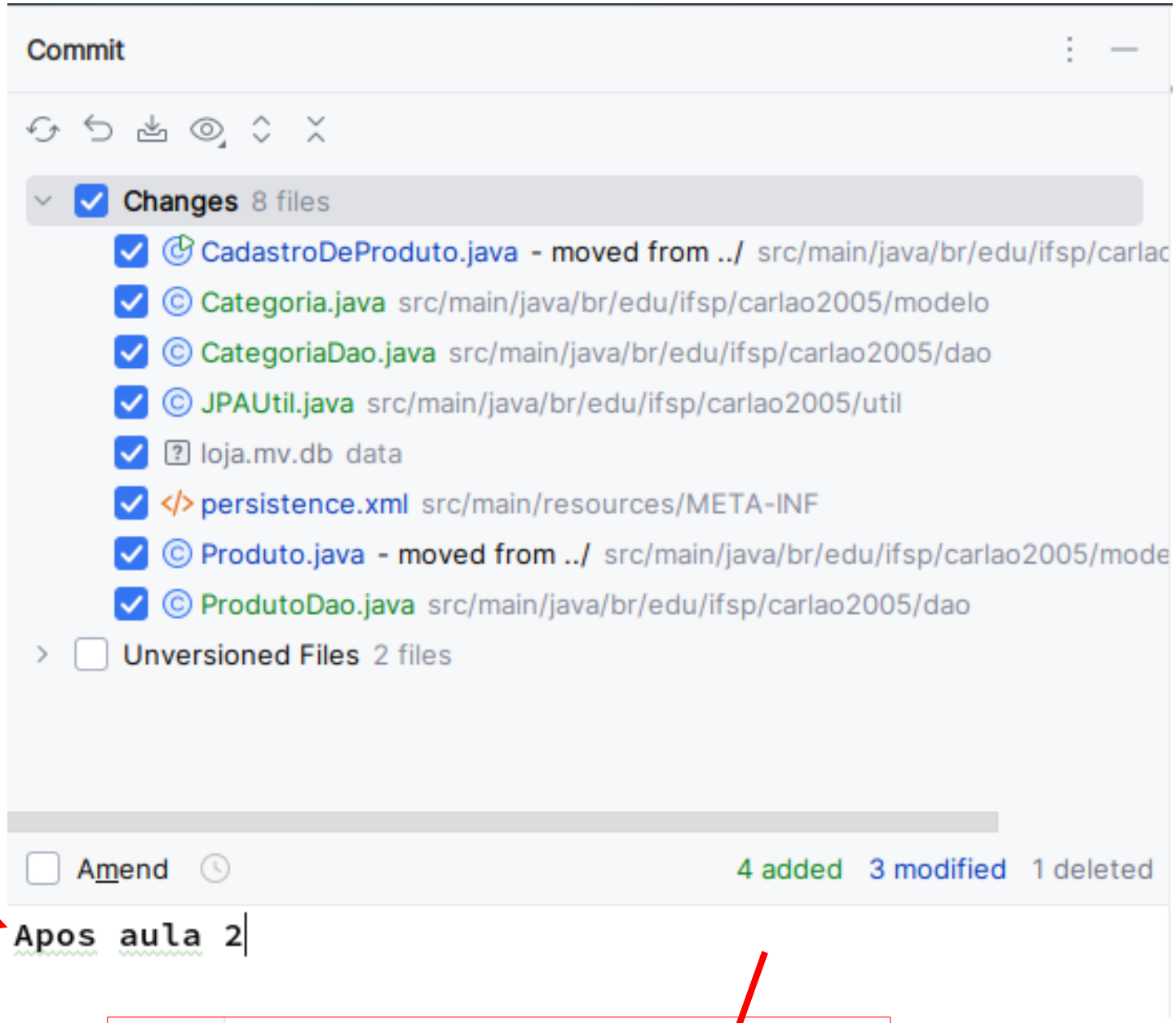
ID	NOME
1	CELULARES
2	INFORMATICA & COMPUTAÇÃO
3	ELETRO antes commit
5	ESPORTE depois de merge

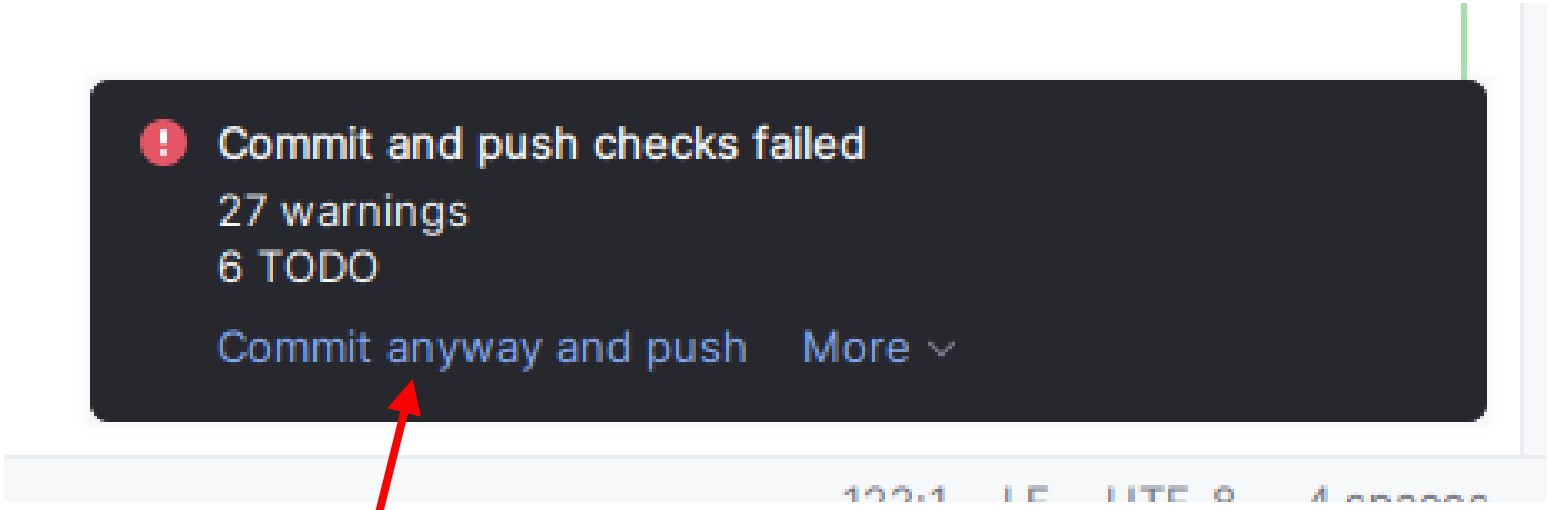
(4 rows, 1 ms)

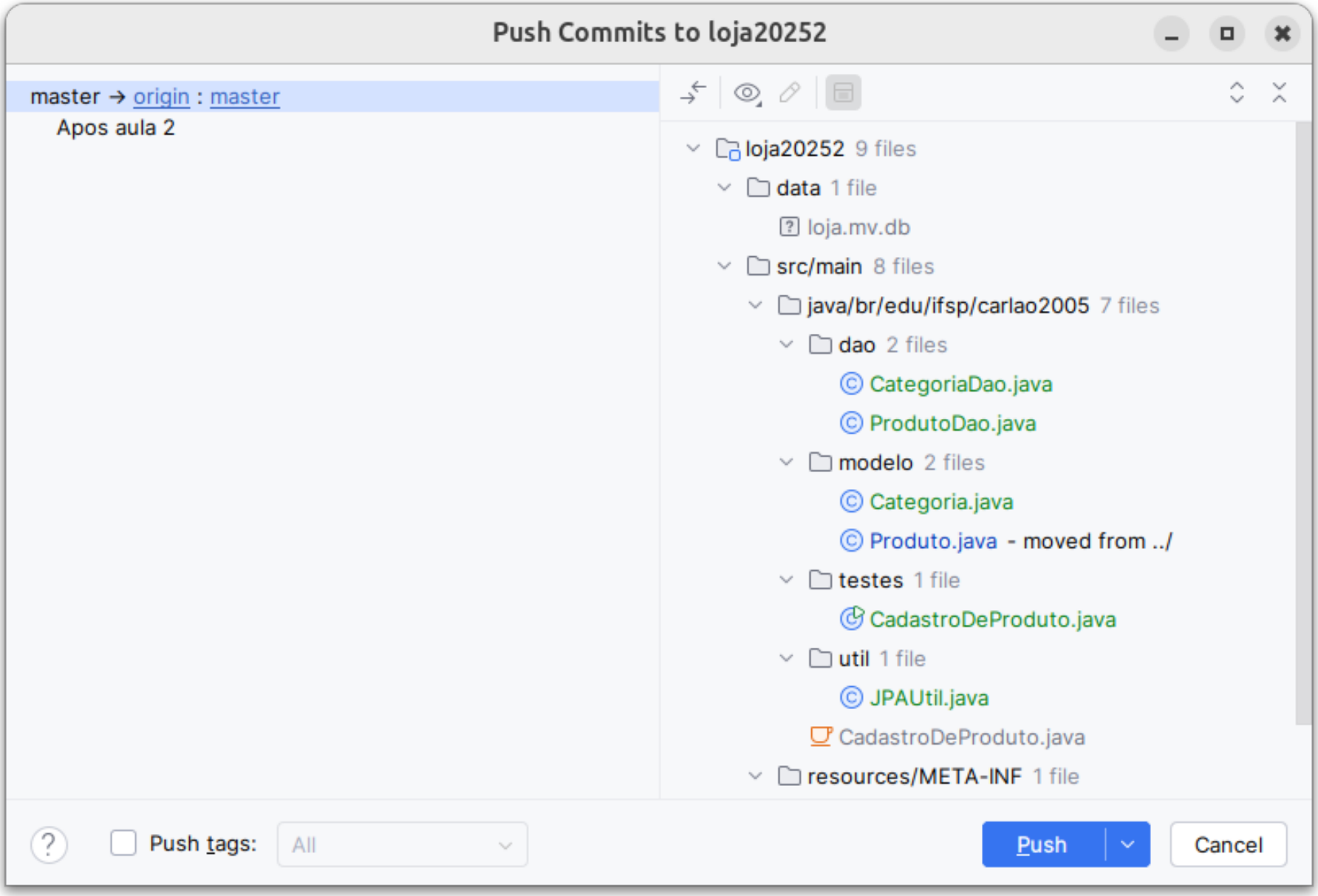
```
INFO: HHH10001501: Connection obtained from JdbcConn
Hibernate: insert into categorias (nome) values (?)
Hibernate: delete from categorias where id=?

Process finished with exit code 0
```

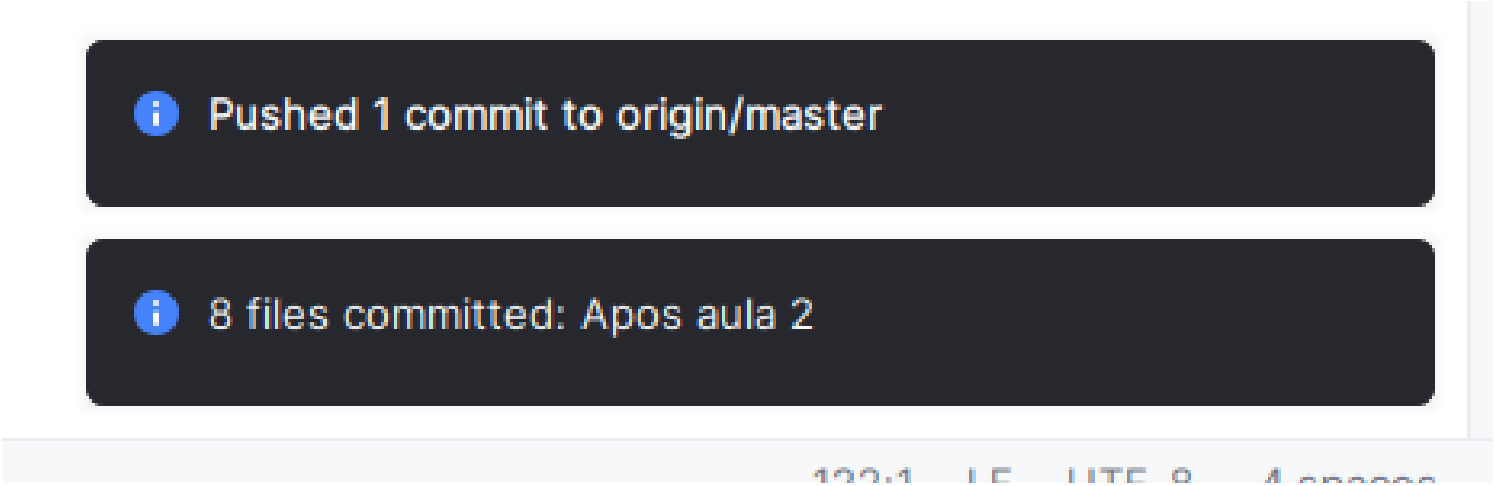
Atualizando no Github...


loja20252
master
Search for branches and actions
Update Project... Ctrl+T
Commit... Ctrl+K
Push... Ctrl+Shift+K
New Branch... Ctrl+Alt+N
Checkout Tag or Revision...
Local
master origin/master
Remote
loja20252 ~/HD_P
.idea
.mvn
data
h2-2.3.232.j
loja.mv.db
loja.trace.db
src
main
java


Commit
Changes 8 files
CadastroDeProduto.java - moved from ../ src/main/java/br/edu/ifsp/carlac
Categoria.java src/main/java/br/edu/ifsp/carlao2005/modelo
CategoriaDao.java src/main/java/br/edu/ifsp/carlao2005/dao
JPAUtil.java src/main/java/br/edu/ifsp/carlao2005/util
loja.mv.db data
persistence.xml src/main/resources/META-INF
Produto.java - moved from ../ src/main/java/br/edu/ifsp/carlao2005/mode
ProdutoDao.java src/main/java/br/edu/ifsp/carlao2005/dao
Unversioned Files 2 files
Amend
4 added 3 modified 1 deleted
Apos aula 2
Commit
Commit and Push...


Commit and push checks failed
27 warnings
6 TODO
Commit anyway and push
More


Push Commits to loja20252
master → origin : master
Apos aula 2
loja20252 9 files
data 1 file
loja.mv.db
src/main 8 files
java/br/edu/ifsp/carlao2005 7 files
dao 2 files
CategoriaDao.java
ProdutoDao.java
modelo 2 files
Categoria.java
Produto.java - moved from ../
testes 1 file
CadastroDeProduto.java
util 1 file
JPAUtil.java
CadastroDeProduto.java
resources/META-INF 1 file
Push tags:
All
Push
Cancel


Pushed 1 commit to origin/master
8 files committed: Apos aula 2

O projeto do IntelliJ,
no estado que ficou neste final de aula,
está disponível em

https://github.com/carlaopereirasanca/loja20252
(commit “apos aula 2”)

Os códigos intermediários (testes)
estão comentados (cada teste).

```
/*/ Criando a Categoria "ESPORTE":
Categoria esporte = new Categoria("ESPORTE");

EntityManager em = JPAUtil.getEntityManager();
CategoriaDao categoriaDao = new CategoriaDao(em);

// Iniciando a transação:
em.getTransaction().begin();

// Mudando para o estado managed:
em.persist(esporte);

// Trocando o texto no objeto:
esporte.setNome("ESPORTE atualizado");

// Não vamos fechar a transação....
// Vamos atualizar com flush():
em.flush();

// Vamos tornar 'esporte' detached:
em.clear();

// Precisamos agora alterar o nome... Como voltar de detached para managed?
// Usamos o método merge():
esporte = em.merge(esporte);

// Trocando o texto no objeto:
esporte.setNome("ESPORTE depois de merge");

// Finalizando a transação:
em.getTransaction().commit();

// Fechando o EntityManager
em.close(); */
```

Exercícios


Social Network
Public Cloud
Laptop
Mobile
Tablet
PC
APPs
entertainment
Telemedicine
Infrastructure
Docs
Training
Storage
Backup
Enterprise
Cloud
Private
internal
Cloud
depositphotos

Reproduza e experimente com os códigos vistos nesta aula.

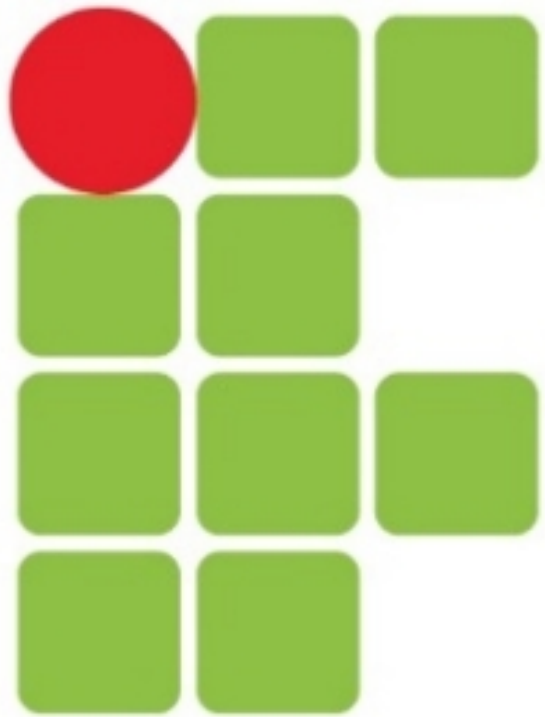
Próxima aula...

- Última aula de JPA: Consultas
- Enunciado da 1a avaliação.

AVISO!! (será repetido todas as aulas)

- Mantenha a disposição de cabos nos computadores dos labs.
- Se retirar um cabo de rede (para usar notebook),
  não esqueça de repor no computador ao final da aula.
- Mantenha o filtro de linha e as tomadas, atrás da mesa,
  em seu estado original, especialmente para que não possa ser esbarrado por outros alunos.
- Não altere de jeito nenhum os cabos atrás dos computadores,
  especialmente mexendo nas “travas”.
- **Por favor, antes de sair, desligue(m) a(s) máquina(s).**
- Lembre-se! Os laboratórios são nossos!! Vamos cuidar!


obrigado
pela atenção!
RENAN PAVINI



INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Câmpus São Carlos