



PROGRAMAÇÃO JAVA AVANÇADA

AULA 08 JAVA E BANCO DE DADOS

PROF. LEONARDO CARLOS COMOTTI KASPERAVICIUS
SUN CERTIFIED JAVA PROGRAMMER 1.4



Tópicos Desta Aula

- Conversão de diagramas UML para Bancos de Dados Relacionais
 - Conversão de Classes, Atributos, Métodos
 - Conversão de Agregação e Composição, Herança, etc.
- Introdução a JDBC
 - Conceitos sobre JDBC (*Java DataBase Connectivity*)
 - O que é JDBC?
 - Especificação das API's
 - Escolha e utilização de drivers para conexão
 - Como se conectar a um banco de dados
 - Tratamento dos dados
 - Preparação do ambiente para utilização via JDBC
 - Integração de aplicações Java com Banco de Dados Relacionais
 - Boas práticas na utilização de JDBC



PROGRAMAÇÃO JAVA AVANÇADA

AULA 08-A

CONVERSÃO DE UML PARA UM BANCO RELACIONAL



Mapeamento de UML para Banco de Dados Relacional

- Existem vários mapeamentos do modelo OO para o modelo relacional:
 - Mapeamento de Classes
 - Mapeamento de Atributos
 - Mapeamento de Composições e Agregações
 - Mapeamento de Associações
 - Mapeamento de Generalizações



Conversão de Classes e Atributos

- Se um objeto persistente tem apenas tipos de dados primitivos como atributos, o mapeamento é claro. Mas na maioria das vezes esta não é uma questão simples, pois objetos podem conter referências a outros objetos complexos [LAR01].
- Cada classe persistente encontrada no diagrama é mapeada em uma ou mais tabelas. A regra geral dita que uma classe é representada no banco como uma tabela, e seus atributos são mapeados como colunas de tipos correspondentes no SGBD [MUL02].



Conversão de Classes e Atributos

- Na Programação OO, cada objeto instanciado possui um endereço único na memória física (identidade implícita), que permite sua manipulação pelo sistema [MUL02]. Este sistema permite, literalmente, a criação de dois objetos iguais em memória (com o mesmo conteúdo interno), o que não é permitido em um banco de dados relacional normalizado.
- É necessário na conversão, portanto, inserir detalhes como chaves primárias, chaves candidatas, se cada atributo pode ou não ser nulo, etc [RUM94].

Exemplo Simples

Pessoa
<ul style="list-style-type: none"> - nome:String - endereço:String
<ul style="list-style-type: none"> + getNome():String + setNome(_nome:String):void + getEndereço():String + setEndereço(_endereço:String):void

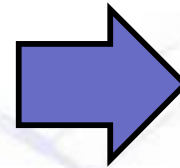



Tabela Pessoa

Nome	Nulos?
 ID	N
◆ nome	N
◆ endereco	S

```
CREATE TABLE `pessoa` (
  `id` int(10) unsigned NOT NULL,
  `nome` varchar(30) NOT NULL,
  `endereco` varchar(50),
  PRIMARY KEY (`id`)
);
```

- Para saber o código (ID) na hora de inserir novos dados na tabela, há pelo menos três maneiras possíveis [MUL02]:
 - Utilizar algum recurso de auto-numeração do banco de dados, como o atributo `IDENTITY` do Sybase e SQL Server, ou `AUTO-INCREMENT` do MySQL;
 - Consultar o valor máximo da coluna a partir da tabela, incrementá-lo em uma unidade, e utilizar esse número como o novo valor na inserção;
 - Utilizar um algoritmo de geração de números exclusivos, como um código hash ou uma geração GUID (como por exemplo o padrão utilizado no sistema de cadastro de disciplinas do BlackBoard).

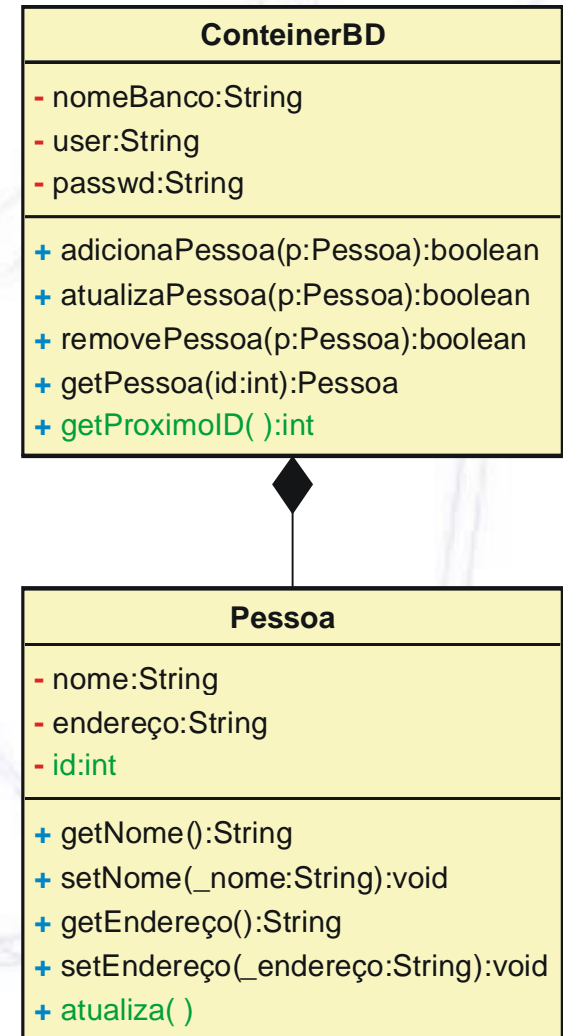


Atualização dos Dados no Banco

- As práticas da Modelagem Orientada a Objetos (encapsulamento, reuso, etc.) especificam que cada objeto deve ser responsável por si mesmo [SHA01].
- Para compensar deficiências do SGBDR e reforçar a integridade [RUM94], o aplicativo **tem** de saber o código (ID) do objeto, a não ser que um dos atributos da classe possa ser uma chave primária (no caso de *Pessoa*, **Cic** seria uma boa sugestão).
- Na prática, na fase de implementação a classe ganha mais um atributo, que representa o seu código no banco de dados. Esse código pode ser transparente para o usuário do sistema.

Implementação Possível

- A classe não-persistente *ContainerBD* é responsável pelo acesso ao banco de dados, inclusive pelo controle do próximo ID a ser inserido no banco.
- A adição do atributo ID na classe *Pessoa* é mais eficiente do que pesquisas frequentes ao banco.



Atributos Compostos (Arrays)

- Uma classe pode conter um atributo que represente um conjunto de valores, enquanto que o modelo relacional requer que os valores sejam atômicos, de modo a satisfazer a primeira Forma Normal [LAR01].
- A maneira mais correta de mapeamento para o banco de dados relacional seria a construção de uma tabela exclusiva para este atributo, com uma chave estrangeira referenciando o ID da tabela principal.

Conversão de Arrays (Exemplo)

Pessoa
<ul style="list-style-type: none"> - id:int - nome:String - endereço:String - telefone:String []

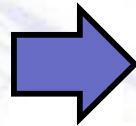


Tabela
Pessoa

Nome	Nulos?
🔑 ID	N
💎 nome	N
💎 endereço	S

Tabela
Pessoa_Telefone

Nome	Nulos?
🔑 ID	N
🔑 valor	N

1 — N

```
CREATE TABLE `pessoa` (
  `id` int(11) NOT NULL auto_increment, `nome` varchar(30) NOT NULL,
  `endereço` varchar(50) default NULL,
  PRIMARY KEY (`id`)
);

CREATE TABLE `pessoa_telefone` (
  `id` int(11) NOT NULL, `valor` varchar(15) NOT NULL,
  PRIMARY KEY (`id`, `valor`),
  CONSTRAINT `ibfk_1` FOREIGN KEY (`id`) REFERENCES `pessoa` (`id`)
);
```

Conversão de Métodos

- Os métodos de um diagrama UML podem ser transformados em **procedimentos** ou **funções** quando passados para o modelo relacional, porém essa transformação varia de banco para banco pois nem todos os bancos de dados disponíveis no mercado possuem suporte a esse tipo de mecanismo.
- Os procedimentos ou funções servem para realizar uma determinada ação no banco de dados em uma instrução SQL, ou seja, são blocos de códigos definidos que podem ser chamados por instruções SQL facilitando o trabalho e diminuindo a linha de comando, uma vez que boa parte do código já foi implementado.

Exemplos em Oracle

- A seguir mostraremos exemplos de transformação de métodos do diagrama UML para um banco relacional.
- Para este exemplo iremos utilizar os métodos da classe persistente **Aluno**.

Aluno
<ul style="list-style-type: none">- id:int- nome:String- rgm:int- curso:Curso
<ul style="list-style-type: none">+ addAluno(n:String,r:int,c:Curso):boolean+ getAluno(rgm:int):Aluno

Criando o Método getAluno(rgm)

- Criando a função getAluno no Oracle:

```
Create Function getAluno (p_RGM%TYPE) RETURN VARCHAR  
IS  
v_nome aluno.nome%TYPE;  
BEGIN  
SELECT nome FROM aluno where RGM = p_RGM;  
RETURN v_NOME;  
END getAluno;
```

- Executando a função getAluno no SQL:

```
SELECT * FROM aluno WHERE nome = getAluno('1234');
```



Criando o Método addAluno(...)

- Criando um procedimento addAluno:

```
CREATE OR REPLACE PROCEDURE addAluno (  
  p_NOME aluno.Nome%TYPE,  
  p_RGM aluno.RGM%TYPE,  
  p_CURSO alunoCurso%TYPE) AS  
BEGIN
```

- Inserindo uma nova linha na tabela aluno:

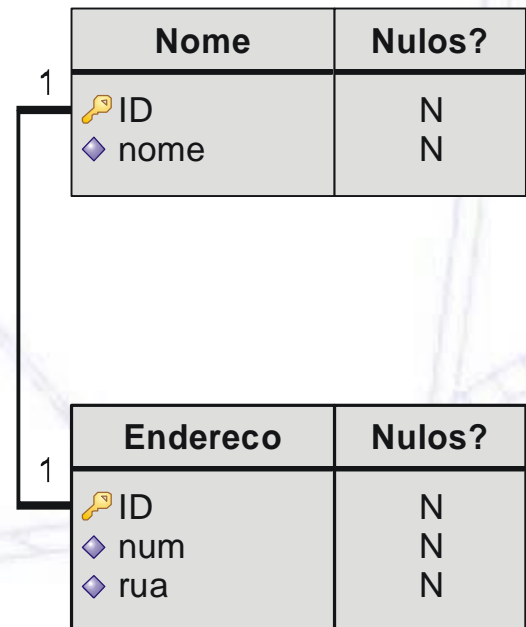
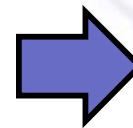
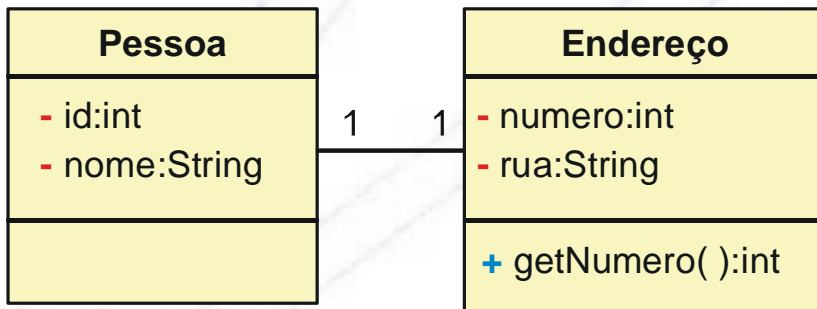
```
INSERT INTO aluno (Nome, RGM, Curso)  
VALUES (p_NOME, P_RGM,p_CURSO,);  
  
COMMIT;  
  
END addAluno;  
  
/
```

- Utilizando a procedure addAluno:

```
DECLARE  
  
p_NOME aluno.Nome%TYPE:='Rogerio';  
p_RGM aluno.RGM%TYPE:='123';  
p_CURSO alunoCurso%TYPE:='Banco de Dados';  
  
BEGIN  
  addAluno(p_NOME, p_RGM, p_CURSO);  
  
END;  
  
/
```

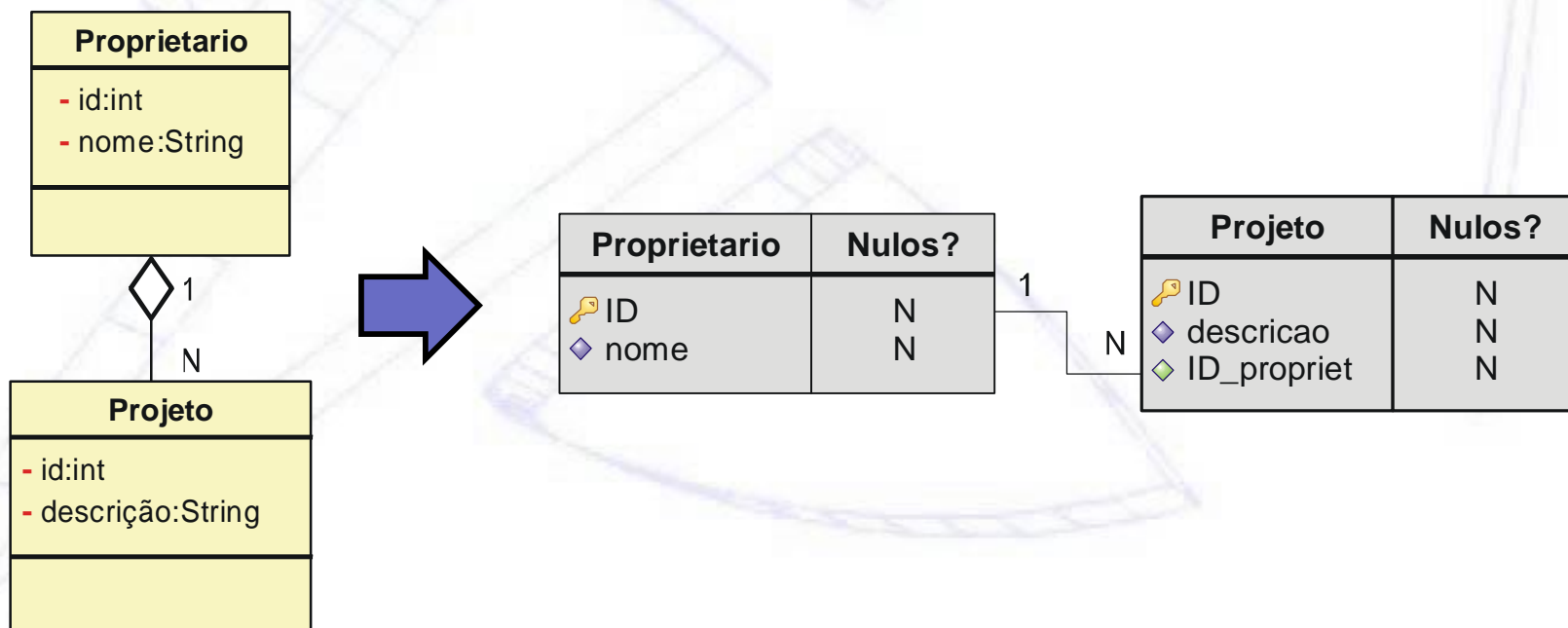
Mapeamento de Associações 1:1

- No mapeamento 1:1 pode-se escolher colocar a chave primária de X (Pessoa) como chave estrangeira de Y (Endereço) ou o inverso, procurando sempre escolher uma classe com participação total no relacionamento para evitar valores nulos.



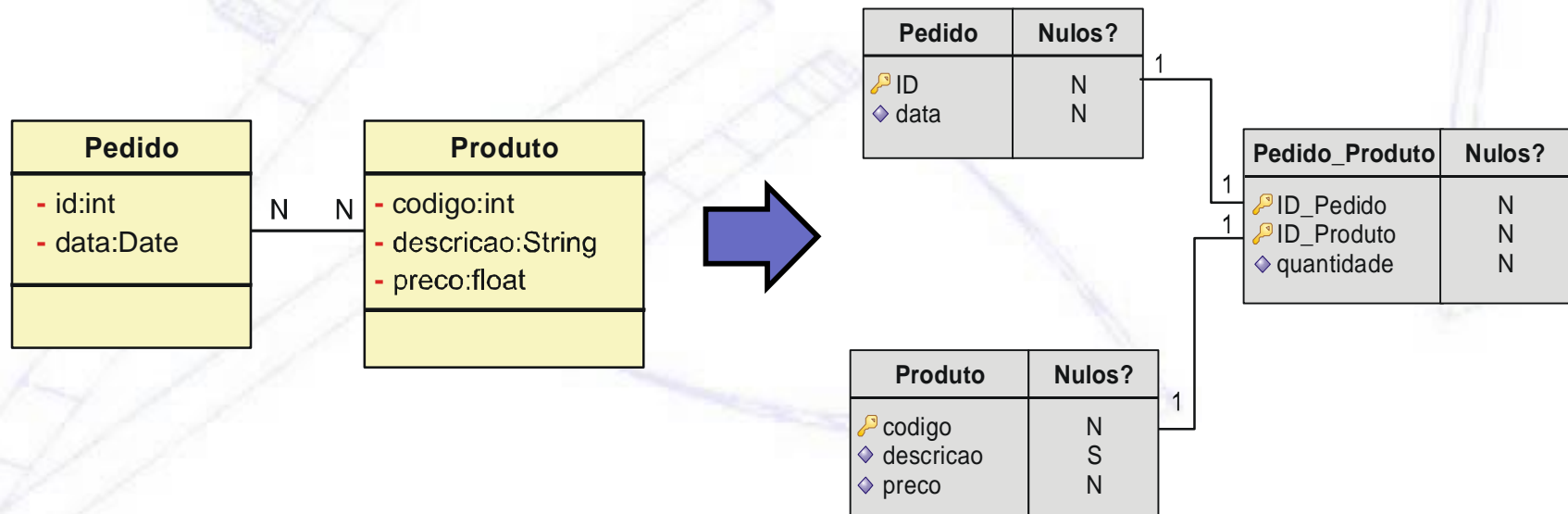
Mapeamento de Associações 1:N

- Para cada relacionamento 1-N coloca-se a chave primária de X (Projeto) como chave estrangeira de Y (Proprietario). Colocamos a chave estrangeira sempre na tabela do lado de muitos no relacionamento.



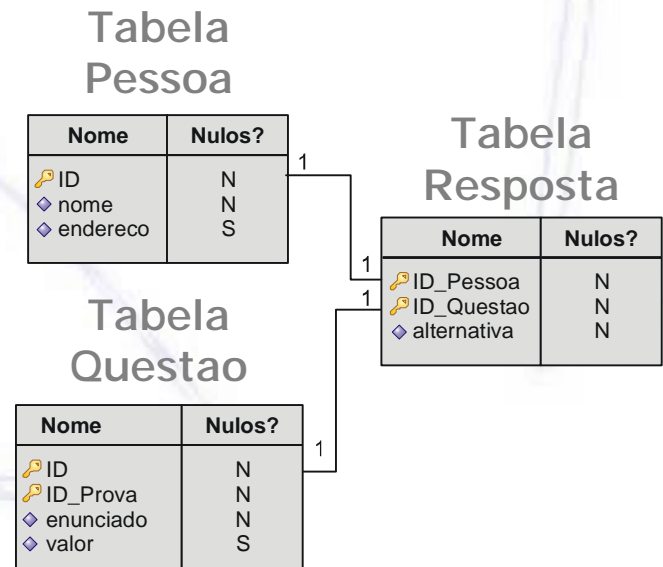
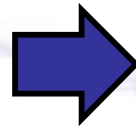
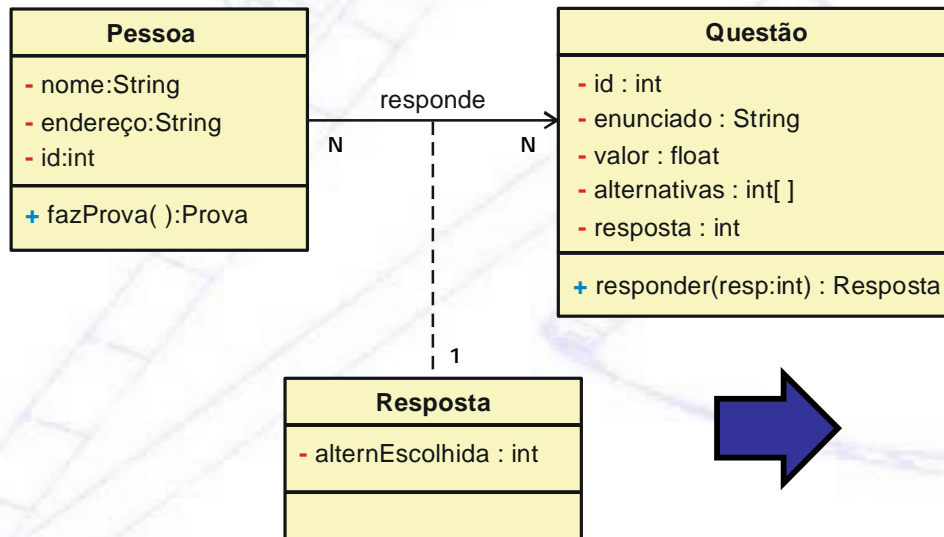
Mapeamento de Associações N:N

- Para cada relacionamento N-N cria-se uma nova tabela (tabela associativa) para a relação onde irá conter as chaves primárias de X e Y como chaves estrangeiras. O nome da tabela associativa é a junção do nome das duas tabelas.



Classes Associativas

- Em uma associação entre duas classes, a própria associação pode ter propriedades [BOO00], representadas em UML como as Classes Associativas.
- Novamente, a decisão final sobre quando embutir ou não uma associação em uma classe relacionada depende da aplicação [RUM94].



Mapeamento de Composição

- A associação de Composição diz que o objeto de composição possui o outro objeto e que nenhum outro objeto pode-se vincular a este. Esta forte forma de agregação corresponde diretamente a uma chave estrangeira com ações atualizar e excluir [MUL02].
- O mapeamento depende do **tipo** e **multiplicidade** da associação e das preferências do projetista de bancos de dados em termos de extensibilidade, do número de tabelas e do balanceamento do desempenho [RUM94].

Composição (Exemplos)

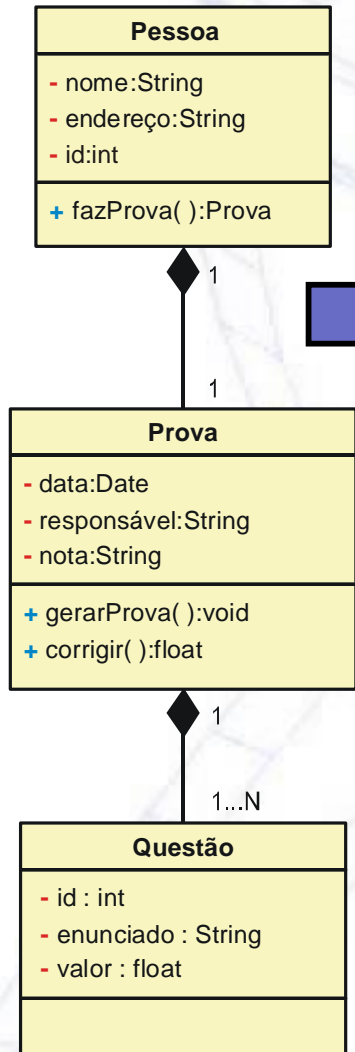


Tabela
Pessoa

Nome	Nulos?
ID	N
nome	N
endereço	S

Tabela
Prova

Nome	Nulos?
ID	N
data	N
responsável	N
nota	S

Tabela
Questao

Nome	Nulos?
ID	N
ID_Prova	N
enunciado	N
valor	S

```

CREATE TABLE `prova` (
  `id` int(11) NOT NULL,
  `data` date NOT NULL,
  `responsavel` varchar(30) NOT NULL,
  `nota` float,
  PRIMARY KEY (`id`),
  CONSTRAINT `ibfk_1` FOREIGN KEY (`id`) REFERENCES `pessoa` (`id`)
);

CREATE TABLE `questao` (
  `id` int(11) NOT NULL auto_increment,
  `id_prova` int(11) NOT NULL,
  `enunciado` varchar(100) NOT NULL,
  `valor` float NOT NULL,
  PRIMARY KEY (`id`, `id_prova`),
  KEY `id_prova` (`id_prova`),
  CONSTRAINT `ibfk_2` FOREIGN KEY (`id_prova`) REFERENCES `prova` (`id`)
);
    
```



Integração de Associações

- Quando a relação de uma composição é *um-para-um* pode-se incorporar os atributos na própria classe [RUM94].
- O desempenho aumenta por existirem menos tabelas no banco, mas em contrapartida a complexidade aumenta, além de extinguir o conceito de encapsulamento das linguagens baseadas em objetos, prejudicando a manutenção.

Incorporação (Exemplo)

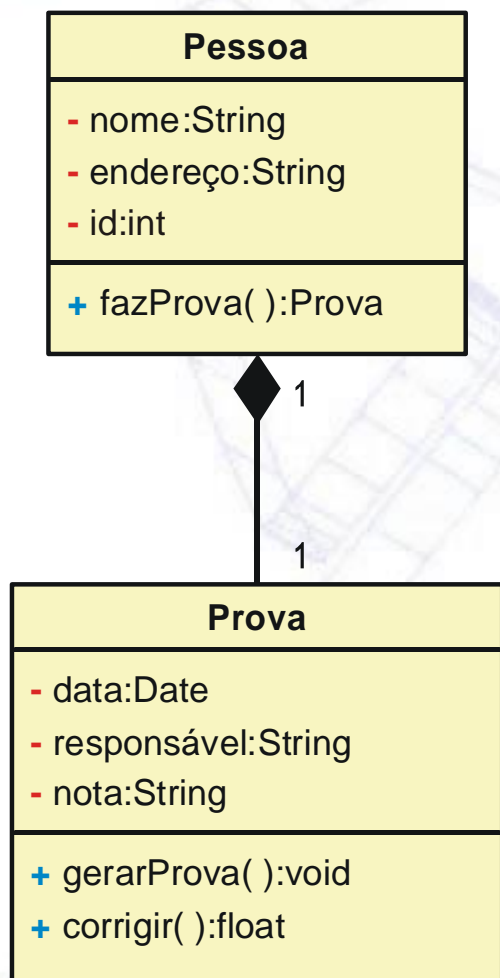


Tabela
Pessoa

Nome	Nulos?
ID	N
nome	N
endereço	S
data	N
responsável	N
nota	S

Tabela
Questao

Nome	Nulos?
ID	N
ID_Prova	N
enunciado	N
valor	S

1
1...N

```

CREATE TABLE `pessoa` (
  `id` int(11) NOT NULL auto_increment,
  `nome` varchar(30) NOT NULL,
  `endereço` varchar(50) default NULL,
  `data` date NOT NULL,
  `responsável` varchar(30) NOT NULL,
  `nota` float,
  PRIMARY KEY (`id`)
);
    
```

Generalizações (Herança)

- Uma Generalização é um relacionamento entre itens gerais (chamados superclasses) e tipos mais específicos desses itens (chamados subclasses), onde as subclasses herdam os atributos e métodos da classe-mãe [BOO00], que é geralmente uma classe Abstrata.
- Classes Abstratas são mais do que classes que não são podem ser instanciadas: elas oferecem uma maneira de definir uma interface genérica para classes derivadas, que possuem atributos e comportamentos próprios [SHA01].

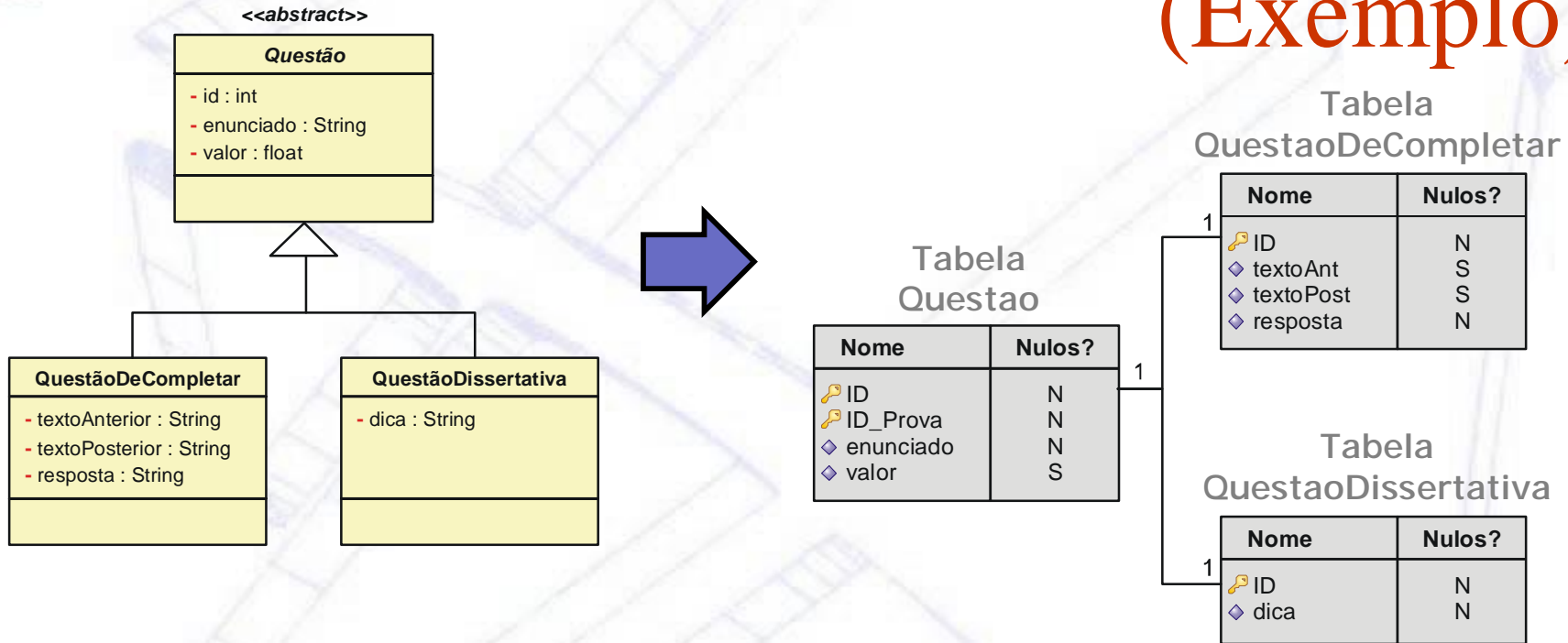


Mapeamento de Herança Simples

- A abordagem mais simples é mapear a superclasse e as subclasses cada uma em uma tabela (Mapeamento Direto), preservando a identidade do objeto em um ID compartilhado [RUM94].
- A implementação também depende se a superclasse é uma classe abstrata ou concreta [MUL02]:
 - Se a classe for concreta, você pode criar um objeto sem se preocupar com subclasses;
 - Se ela for uma classe abstrata, precisa de um gatilho que imponha a sua abstração.

- Deve-se garantir que, se você inserir um registro numa tabela correspondendo a uma classe abstrata, que você também insira um registro em uma das diversas subclasses [MUL02]. Muitas vezes é mais fácil manter a integridade em nível de aplicação, mas com bancos de dados grandes isto é impraticável, já que o próximo aplicativo pode arruinar tudo.
- Neste caso, pode-se criar os atributos diretamente nas subclasses (Espalhamento), eliminando junções, uniões e gatilhos, reduzindo a complexidade.
- A desvantagem desta técnica é a desnormalização do projeto de esquema do banco de dados e a redundância desnecessária em seus dados.

Herança Simples (Exemplo)



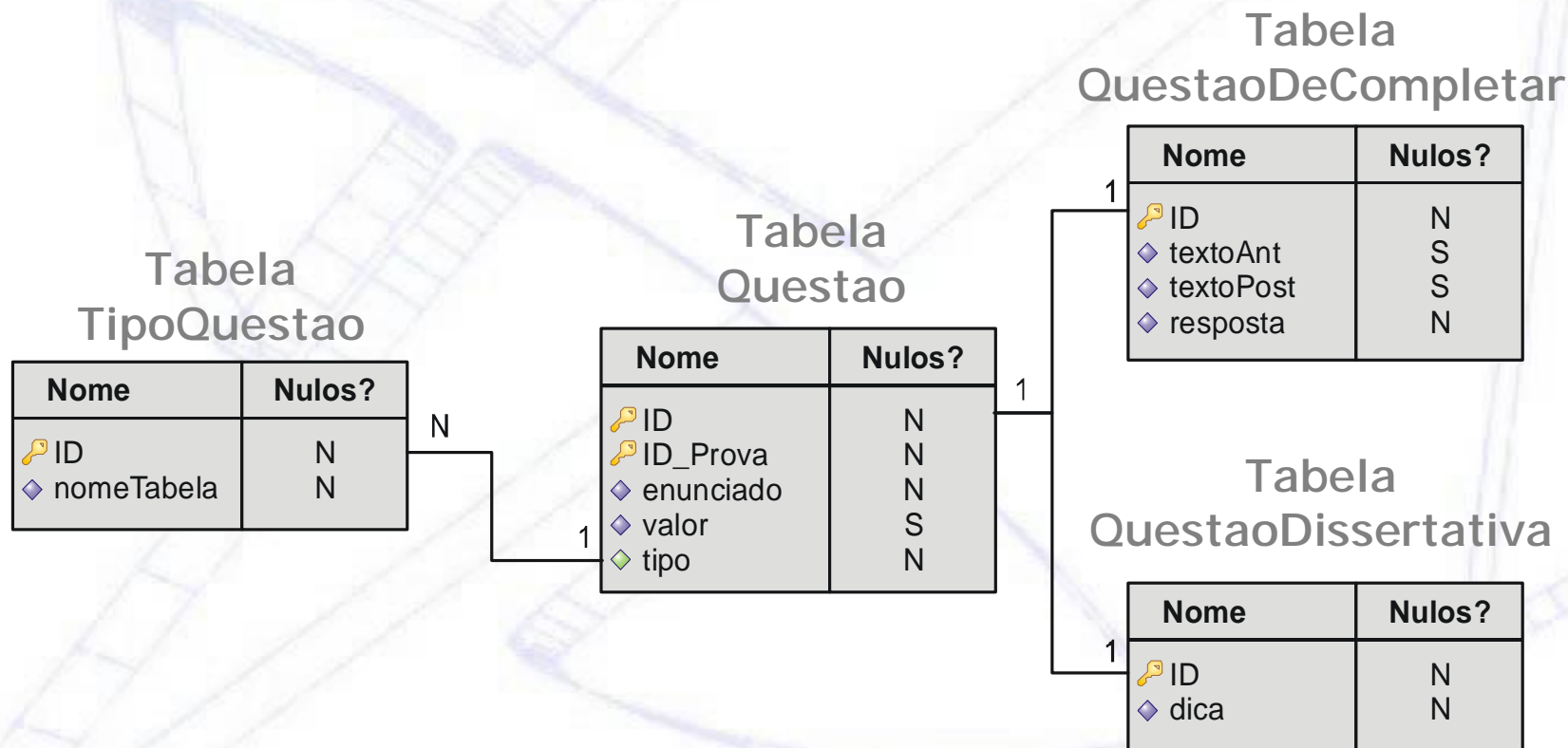
- Este tipo de mapeamento é mais vantajoso se pensarmos em manutenção de código: se a classe principal **Questão** receber mais um atributo posteriormente, basta adicionar mais uma coluna na tabela principal.



Recuperação e Atualização dos Dados

- Novamente, um problema a ser levado em consideração é a eficiência na recuperação dos dados armazenados no banco.
- Para o sistema construir uma generalização a partir de uma tabela que representa uma classe abstrata, necessitaria saber exatamente qual das sub-tabelas contém o registro correspondente.
- Para evitar a pesquisa em todas as tabelas, uma a uma, pode-se criar uma tabela correspondente aos tipos de generalização do objeto, e adicionar um campo na tabela da classe principal.
- O problema é que não há como impor a integridade deste relacionamento, a não ser pelo próprio aplicativo.

Exemplo de Implementação



- No mapeamento de classes para um banco de dados relacional existem várias técnicas e abordagens, que devem ser avaliadas na hora da implementação, de acordo com a especificidade de cada caso.
- O cenário atual apresenta a necessidade de uma padronização, tanto nos bancos de dados relacionais quanto na modelagem orientada a objetos.

- [BOO00] Booch, G; Rumbaugh, J.; Jacobson, I. – ***UML - Guia do Usuário*** – Editora Campus – Rio de Janeiro, 2000
- [MUL02] Muller, R.J. – ***Projeto de Banco de Dados - Utilizando UML para Modelagem de Dados*** – Editora Berkeley – São Paulo, 2002.
- [LAR01] Larman, C. – ***Applying UML and Patterns - Second Edition*** – Prentice Hall – 2001.
- [RUM94] Rumbaugh, J. et al – ***Modelagem e Projetos Baseados em Objetos*** – Editora Campus – Rio de Janeiro, 1994.
- [SHA01] Shalloway, A; Trott, J.R. – ***Design Patterns Explained*** – Addison-Wesley – Canada, 2001



Exercício

- A partir da definição de uma Pessoa (ProjetoJava, utilizado em aulas anteriores), crie a(s) tabela(s) correspondentes no banco MySQL.



PROGRAMAÇÃO JAVA AVANÇADA

AULA 08-B

**JDBC – JAVA DATABASE
CONNECTIVITY**

O que é JDBC ?

- Java Database Connectivity é a tecnologia que permite acesso e manipulação de banco de dados
- JDBC é composta de duas partes:
 - JDBC Core API (`java.sql`)
 - JDBC Optional Package API (`javax.sql`)
- Código SQL é utilizado explicitamente dentro do código Java, e a discussão a seguir supõe a familiaridade com a mesma.

O pacote *java.sql*

- Oferece a API para acessar e processar dados em uma fonte de dados.
- Membros mais importantes:
 - A classe *DriverManager*
 - A interface *Driver*
 - A interface *Connection*
 - As interfaces *Statement* e *PreparedStatement*
 - A interface *ResultSet*

A classe DriverManager

- Oferece métodos estáticos para gerenciar drivers JDBC.
- O driver JDBC para cada banco é fornecido ou pelo fabricante do mesmo, ou por terceiros.
- Cada banco de dados (Oracle, SQL Server, MySQL, etc.) possui um driver.
- Para carregar o driver JDBC em um programa, você copia a biblioteca (arquivo JAR) para o diretório de bibliotecas do projeto, e utiliza o seguinte código:

```
try{  
    Class.forName("NomeDoPacote.NomeDoDriver");  
} catch(ClassNotFoundException ex){  
    //driver não encontrado  
}
```

- O método mais importante desta classe é getConnection(), que retorna um objeto do tipo Connection. Este método tem três sobrecargas:
 - getConnection (String url)
 - getConnection (String url, Properties info)
 - getConnection (String url, String user, String password)

A interface Driver

- A interface Driver é implementada por cada classe de driver JDBC (padronização).
- A classe DriverManager carrega e registra os Drivers, podendo gerenciar múltiplos driver para qualquer solicitação de conexão dada.
- No caso onde há múltiplos drivers registrados, a DriverManager pedirá que um driver de cada vez tente se conectar com o URL alvo.



A interface Connection

- A interface Connection representa uma conexão ao banco de dados, e a implementação é obtida pelo método `getConnection()` da classe `DriverManager`.
- Métodos principais:
 - `close()`: fecha e libera imediatamente um objeto Connection.
 - `isClosed()`: Verifica se a conexão está fechada.
 - `commit()`: Compromete a transação.
 - `rollback()`: Utilizado para retornar uma transação.
 - `createStatement()`: Utilizado para criar um objeto Statement, que envia declarações SQL ao banco de dados.
 - `prepareStatement()`: Utilizado para criar um objeto PreparedStatement, que é mais eficiente quando uma mesma instrução SQL é executada muitas vezes.

A interface Statement

- Através desta interface, pode-se executar declarações SQL e obter os resultados produzidos por elas.
- São dois os métodos mais importantes:
 - executeUpdate(String sql)**: Executa uma instrução SQL de inserção, atualização ou remoção (INSERT, UPDATE e DELETE). Este método retorna um número inteiro, correspondente ao número de linhas afetadas pela instrução SQL.
 - executeQuery(String sql)**: Executa uma declaração SQL de pesquisa. Este método retorna um único objeto ResultSet.

A interface ResultSet

- A interface ResultSet representa um conjunto de resultados (como uma tabela).
- Um objeto ResultSet mantém um “cursor” indicando a fileira de dados atual.
- Inicialmente, o cursor é posicionado **antes** da primeira fila!
- Métodos importantes:
 - **next()**: Move o cursor para o registro a seguir, retornando verdadeiro se a fileira for válida e falso se não existirem mais registros no ResultSet.
 - **isFirst()**: Indica se o cursor aponta para o primeiro registro na ResultSet.
 - **isLast()**: Indica se o cursor aponta para o último registro na ResultSet.
 - **getXXX(int columnIndex)**: Métodos para obter o valor da coluna correspondente (na fileira indicada pelo cursor). XXX representa o tipo de dados retornado pelo método no índice especificado (sendo que o índice **1** é a primeira coluna).

A interface PreparedStatement

- A interface PreparedStatement amplia a interface Statement.
- Representa uma declaração SQL pré-compilada.
- Você usa uma cópia dessa interface para executar com eficácia uma declaração SQL diversas vezes.

Acessando o Banco

- Antes de poder manipular dados em um banco de dados, você precisa se conectar com aquele servidor do banco de dados.
- Acessar um banco de dados com JDBC pode ser resumido nas quatro etapas seguintes:
 1. Carregar o driver de banco de dados JDBC.
 2. Criar uma conexão.
 3. Criar uma declaração.
 4. Criar um conjunto de resultados, se você esperar que o servidor do banco envie de volta alguns resultados.

Etapa 1:

Carregar o Driver JDBC

- Os servidores de bancos de dados têm a sua própria “linguagem” para comunicação.
- Os drivers JDBC são “tradutores”, que fazem a interface de código Java com esses servidores.
- Atualmente, os driver JDBC estão disponíveis para os bancos de dados mais populares (alguns gratuitos e outros pagos).
- Alguns drivers gratuitos são:
 - [MySQL Connector/J](#) – Driver oficial para MySQL
 - [Oracle JDBC Driver](#) – Drivers oficiais para as várias versões do Oracle
 - [jTDS Project](#) – Drivers para SQL Server e Sybase
- Uma ferramenta para pesquisa de drivers pode ser encontrada em <http://developers.sun.com/product/jdbc/drivers>

Etapa 1:

Carregar o Driver JDBC

- Existem quatro tipos de drivers JDBC:
 - **Tipo 1:** Os drivers mais lentos, devem ser utilizados apenas se você não tem outra escolha. São dos drivers que oferecem acesso a drivers ODBC (chamados de JDBC-ODBC Bridge).
 - **Tipo 2:** São escritos parte em Java, parte em API original, para converter chamadas JDBC em chamadas para Oracle, Sybase, DB2 ou outros DBMS.
 - **Tipo 3:** São escritos para utilizar o protocolo de rede na tradução das chamadas JDBC. Capaz de conectar todos os clientes baseados em Java com muitos bancos diferentes, é a alternativa mais flexível.
 - **Tipo 4:** São drivers com tecnologia Java pura. Converter chamadas JDBC diretamente em protocolos de redes utilizados pelos DBMS's. Como a maioria dos protocolos são proprietários, geralmente a principal fonte para este tipo de driver é o próprio fabricante.



Etapa 1:

Carregar o Driver JDBC

- Instalação de drivers JDBC:
 - Normalmente, os drivers JDBC vêm em um arquivo .jar.
 - O modo mais simples de torná-lo acessível é colocá-lo nas pastas `JAVA_HOME\jre\lib\ext` e `JRE_HOME\lib\ext`.
 - Depois pode-se carregar o driver, utilizando o método estático `forName()` da classe `Class`, passando o nome completo (*fully-qualified name*) do driver:

```
//Exemplo com o MySQL Connector
```

```
Class.forName( "com.mysql.jdbc.Driver" );
```

```
//Exemplo com um ODBC (já vem com o JDK)
```

```
Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver" );
```

- Se você planeja conectar-se com vários bancos em seu código, basta carregar todos os drivers JDBC, como mostrado acima, e eles serão gerenciados pela classe `DriverManager`.



Etapa 2: Criação da Conexão

- Uma conexão de banco de dados é representada pela interface `Connection`.
- Você usa o método `getConnection()` da classe `DriverManager` para obter um objeto `Connection`, sendo que a maneira mais utilizada é passar a `url`, o `nome` de usuário e `senha` como parâmetros.
- A `url` é o elemento mais artiloso, e possui a seguinte sintaxe:
 - `jdbc:subprotocol:subname`
- Por exemplo, para se conectar a um banco MySQL chamado `Banco`, como usuário `root` e senha em branco seria:
 - ```
Connection c = DriverManager.getConnection(
 "jdbc:mysql://Banco", "root", "");
```
- Enquanto que, para se conectar com um banco ODBC chamado `Banco`, que é acessível sem o registro de entrada de um nome e senha, você usa o seguinte:
  - ```
Connection c = DriverManager.getConnection("jdbc:odbc:Banco");
```


Criação das Declarações

- Depois de ter um objeto **Connection**, a sua habilidade SQL assume.
- Você pode, basicamente, passar **qualquer** declaração SQL **que o servidor entenda** (procure tomar cuidado com a compatibilidade).
- Para fazer isto, é preciso criar outro objeto JDBC, chamado **Statement**, utilizando o método `createStatement()` do objeto **Connection**:
 - `Statement s = c.createStatement();`
- Em seguida, utilize os métodos para manipular seus dados: `executeUpdate()` ou `executeQuery()`.
- Os dois métodos recebem uma **String** contendo uma declaração SQL, que não precisa terminar com um encerramento de declaração DBMS (o driver que você estiver utilizando fará isto por você).
- O método `executeUpdate()` executa uma declaração **INSERT**, **UPDATE** ou **DELETE**, e também declarações **DDL** para criar, soltar ou alterar tabelas.
- O método `executeQuery()` executa uma declaração **SELECT** que retorna uma pesquisa. Esse método retorna um único objeto **ResultSet**, que é discutido a seguir. Esse objeto contém os dados da pesquisa, e nunca é um objeto nulo.

Etapa 4:

Criação de um ResultSet

- Um **ResultSet** é a representação de uma tabela de banco de dados que é resultado de uma pesquisa.
- Para acessar a primeira linha da pesquisa, é necessário chamar o método **next()**, que pode retornar verdadeiro ou falso.
- Para obter os dados de dentro do ResultSet, você utiliza um dos muitos métodos **getXXX()** (como **getInt**, **getString**, **getLong** e assim por diante), passando o número da coluna ou o nome da mesma.
- Por exemplo, este é o código utilizado para recuperar as colunas **codigo** e **nome** de uma tabela **Pessoa**, e mostrar os dados na tela:

```
String sql = "SELECT codigo, nome FROM Pessoa";
ResultSet rs = s.executeQuery(sql);
while (rs.next()){
    System.out.println(
        rs.getString(1) + " : " + rs.getString("nome"));
}
```

- Sempre feche o ResultSet com o método **close()**.

- Abra o ProjetoJava, utilizado em aulas anteriores.
- Crie um novo pacote (dados.bd) com uma classe `LeitorDePessoasMySQL`, que implementa `LeitorDePessoas`.
- Implemente todos os métodos para que o programa funcione acessando um banco de dados MySQL.