



Persistência de Dados com a API JDBC do Java

Prof. Evandro César Freiburger

Instituto Federal de Mato Grosso
Departamento da Área de Informática
evandro.freiberger@cba.ifmt.edu.br

2018

Sumário

- 1 Introdução
- 2 Conexão com o Banco de Dados
- 3 Execução de Comandos
- 4 Manipulação de Resultados
- 5 Metadados

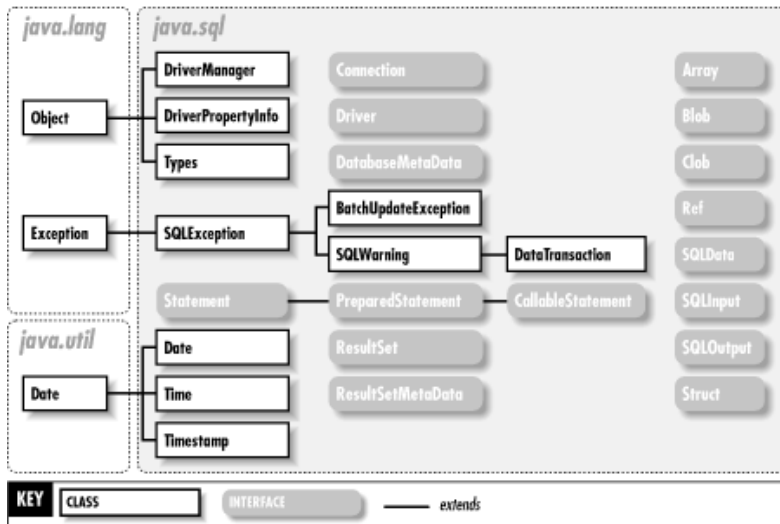
Manipulação de Banco de Dados

- Pode ser realizada com um conexão via Socket
- Conexão via socket exige o conhecimento do protocolo particular de cada SGBD
- O ideal é uma conexão em alto nível, para execução dos comandos no SGBD

API JDBC

- JDBC é uma interface baseada em Java para acesso a banco de dados por meio de comandos SQL
 - ▶ Está implementada no pacote *java.sql*
 - ▶ É baseada no conceito da ODBC (*Open Database Connectivity*)
- Com a JDBC, pode-se obter acesso direto a bancos de dados por meio de aplicações Java
- Com JDBC é possível construir uma aplicação Java para acesso a qualquer banco de dados SQL
 - ▶ É necessário ter um driver JDBC para o SGBD desejado
 - ▶ Caso não tenha um driver JDBC, deve ter pelo menos um driver ODBC

Pacote java.sql



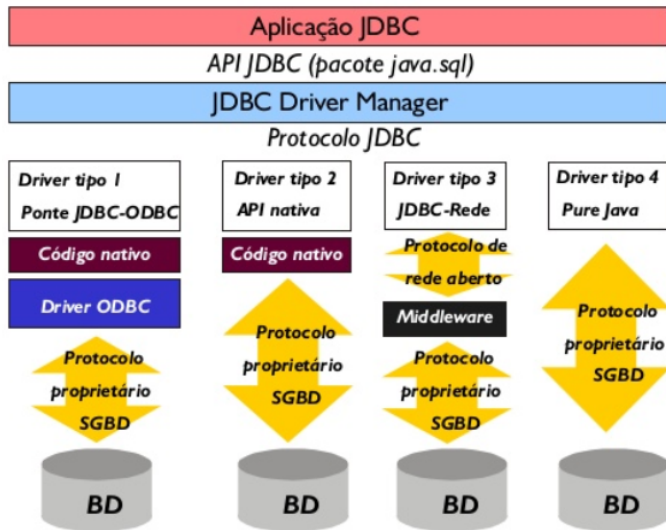
Tipos de Drivers

- Tipo 1: ponte ODBC-JDBC
 - Usam ponte para ter acesso ao banco de dados
 - Requer a instalação de software do lado cliente
 - Dependente de plataforma (software cliente)
- Tipo 2: solução com código nativo
 - Usam API nativa. Esses drivers contêm métodos Java implementados em C ou C++
 - Requer software no cliente
 - APIs dependentes de plataforma

Tipos de Drivers (cont.)

- Tipo 3: solução 100% java no cliente
 - ▶ Oferecem uma API de rede via middleware que traduz requisições para API do driver desejado
 - ▶ Não requer software no cliente
 - ▶ Chamadas são enviadas para um servidor que traduz para um protocolo específico de banco de dados
- Tipo 4: solução 100% java
 - ▶ Drivers que se comunicam diretamente com o banco de dados usando soquetes de rede
 - ▶ É uma solução puro java.
 - ▶ Não requer código adicional do lado do cliente

Tipos de Drivers



URL de Conexão da JDBC

- Contém informações para a conexão com o banco de dados
- Uma aplicação JDBC pode carregar ao mesmo tempo diversos drivers
- A URL é usada para determinar qual driver será utilizado
- A URL tem o formato geral: `jdbc:<subprotocolo>:<dsn>`
 - ▶ O dsn é utilizado para localizar um determinado servidor ou base de dados
 - ▶ A sintaxe da URL depende do fabricante do driver do banco
- Exemplos:
 - ▶ MySQL: `"jdbc:mysql://localhost:3306/produtos"`
 - ▶ Oracle: `"jdbc:oracle:thin:@//localhost:1521/produtos"`
 - ▶ PostgreSQL: `"jdbc:postgres://localhost:5432/produtos"`

DriverManager e Driver

- A interface Driver é utilizada apenas pelas implementações de drivers JDBC
- É preciso carregar a classe do driver na aplicação que irá utilizá-lo
- Isto pode ser feito com Class.forName()
 - ▶ `Class.forName("sun.jdbc.odbc.JdbcOdbcDriver")`
- A classe DriverManager manipula objetos do tipo Driver
- É usado para retornar Connection, que representa uma conexão a um banco de dados
 - ▶ `Connection con = DriverManager.getConnection("url","usuario","senha");`

Exemplo Conexão

Usando Class.forName()

```
1 package apps;
2 import java.sql.Connection;
3 import java.sql.DriverManager;
4 import java.sql.SQLException;
5
6 public class Conexao1 {
7
8     public static void main(String args[]){
9         Connection conexao;
10        String url = "jdbc:postgresql://localhost:5432/produto";
11        String usr = "postgres";
12        String pass = "postgres";
13        try{
14            Class.forName("org.postgresql.Driver");
15            conexao = DriverManager.getConnection(url, usr, pass);
16            System.out.println("Conexao estabelecida");
17            conexao.close();
18            System.out.println("Conexao encerrada");
19        } catch(ClassNotFoundException cnf){
20            System.out.println("Classe do driver nao encontrado - "+cnf.getMessage());
21        } catch(SQLException sqle){
22            System.out.println("Conexao nao estabelecida - "+sqle.getMessage());
23        }
24    }
25 }
```

Exemplo Conexão

Usando Registro Explícito de Driver

```
1 package apps;
2 import java.sql.Connection;
3 import java.sql.DriverManager;
4 import java.sql.SQLException;
5 import org.postgresql.Driver;
6
7 public class Conexao2 {
8     public static void main(String args[]) {
9         Connection conexao;
10        String url = "jdbc:postgresql://localhost:5432/produto";
11        String usr = "postgres";
12        String pass = "postgres";
13        try {
14            DriverManager.registerDriver (new Driver());
15            conexao = DriverManager.getConnection(url,usr,pass);
16            System.out.println("Conexao estabelecida");
17            conexao.close();
18            System.out.println("Conexao encerrada");
19        } catch (SQLException sqle) {
20            System.out.println("Conexao nao estabelecida - "+sqle.getMessage());
21        }
22    }
23 }
```

Exemplo Conexão (1)

Usando arquivo de propriedades (conexaoBD.properties)

```
1 driver=org.postgresql.Driver
2 url=jdbc:postgresql://localhost:5432/produto
3 usuario=postgres
4 senha=postgres
```

```
1 package apps;
2
3 import java.io.FileInputStream;
4 import java.io.FileNotFoundException;
5 import java.io.IOException;
6 import java.sql.Connection;
7 import java.sql.DriverManager;
8 import java.sql.SQLException;
9 import java.util.Properties;
10
11 public class Conexao3 {
12
13     public static void main(String args[]) {
14         Properties proBD = new Properties();
15         FileInputStream leitorArquivo;
16         try {
17             leitorArquivo = new FileInputStream("conexaoBD.properties");
18             proBD.load(leitorArquivo);
19             leitorArquivo.close();
20         } catch (FileNotFoundException ex) {
21             System.out.println("Arquivo de configuracoes nao encontrado - " + ex.getMessage());
22         } catch (IOException ex) {
```

Exemplo Conexão (2)

```
23     System.out.println("Erro ao ler o arquivo de configuracoes - " + ex.getMessage());
24 }
25
26 Connection conexao;
27 String url = proBD.getProperty("url");
28 String driver = proBD.getProperty("driver");
29 String usr = proBD.getProperty("usuario");
30 String pass = proBD.getProperty("senha");
31 try {
32     Class.forName(driver);
33     conexao = DriverManager.getConnection(url, usr, pass);
34     System.out.println("Conexao estabelecida");
35     conexao.close();
36     System.out.println("Conexao encerrada");
37 } catch (ClassNotFoundException cnf) {
38     System.out.println("Driver nao encontrado - " + cnf.getMessage());
39 } catch (SQLException sqle) {
40     System.out.println("Banco nao conectado - " + sqle.getMessage());
41 }
42 }
43 }
```

Fábrica de Conexão

Classe utilitária para gerar objetos Connection

```
1 package apps;
2 import java.sql.Connection;
3 import java.sql.DriverManager;
4 import java.sql.SQLException;
5
6 public class FabricaConexao {
7
8     // teste
9     private static final String url = "jdbc:postgresql://localhost:5432/produto";
10    private static final String driver = "org.postgresql.Driver";
11    private static final String user = "postgres";
12    private static final String pass = "postgres";
13
14    public static Connection obterConexao() {
15        Connection conexao = null;
16
17        try {
18            Class.forName(driver);
19            conexao = DriverManager.getConnection(url, user, pass);
20        } catch (ClassNotFoundException cnf) {
21            System.out.println("Driver nao encontrado - " + cnf.getMessage());
22        } catch (SQLException sqle) {
23            System.out.println("Banco nao conectado - " + sqle.getMessage());
24        }
25        return conexao;
26    }
27 }
```

Connection, ResultSet e Statement

- Todas as implementações de Drivers JDBC, implementam os métodos para essas interfaces
- Connection
 - Objeto que representa uma conexão ao banco de dados, é retornado pelo DriverManager
- Statement
 - Usado para passar instruções SQL para o sistema de banco de dados
- ResultSet
 - É um cursor para os dados recebidos

Statement

- O objeto Statement é obtido por meio do método *createStatement()* de objeto Connection
 - ▶ *Statement stmt = con.createStatement()*
 - ▶ A partir de um objeto de Statement poderão ser executados métodos para enviar instruções SQL ao BD
 - ▶ Exemplos: *execute()*, *executeQuery()*, *executeBatch()* e *executeUpdate()*
- Subinterface PreparedStatement - execução de comandos SQL com pré-compilação
- Subinterface CallableStatement - chamadas de procedimentos armazenados no BD
 - ▶ *PreparedStatement pstmt = con.createPreparedStatement("...sql...")*
 - ▶ *CallableStatement cstmt = con.prepareCall(...)*

Exemplo Inclusão

Usando Statement

```
1 package apps;
2 import java.sql.Connection;
3 import java.sql.SQLException;
4 import java.sql.Statement;
5 public class Incluir1 {
6     public static void main(String args[]) {
7         Connection conexao = FabricaConexao.obterConexao();
8         Statement comando = null;
9         String sql = "INSERT INTO GRUPOPRODUTO ( NOME, PROMOCAO, MARGEMLUCRO ) VALUES ";
10        String nome = "Bebidas destiladas";
11        float promocao = 10;
12        float margem = 40;
13        sql += "(" + nome + ", " + promocao + ", " + margem + ")";
14        try {
15            comando = conexao.createStatement();
16            comando.executeUpdate(sql);
17            System.out.println("Inclusao realizada com sucesso");
18        } catch (SQLException ex) {
19            System.out.println("Erro ao incluir grupo de produto" + ex.toString());
20        } finally {
21            try {
22                comando.close();
23                conexao.close();
24            } catch (SQLException ex) {
25                System.out.println("Erro ao desconectar" + ex.toString());
26            }
27        }
28    }
29 }
```

PreparedStatement

- PreparedStatement é uma subinterface de Statement e uma subclasse nas implementações JDBC
- Pré-compila o comando SQL para maior eficiência
- É útil quando várias ocorrências do mesmo comando são enviadas com parâmetros diferentes
- A String com a instrução SQL é preparada previamente, deixando-se “?” no lugar dos parâmetros
- Parâmetros são inseridos na ordem dos “?”, com métodos setXXX() onde XXX é um tipo do parâmetro

Fornecimento de Valores para o PreparedStatement

Tipos de Dados SQL	Métodos do PreparedStatement
INTEGER	setInt()
BIGINT	setLong()
TINYINT	setByte()
REAL	setFloat()
FLOAT, DOUBLE	setDouble()
DECIMAL, NUMERIC	setBigDecimal()
BIT, BOOLEAN	setBoolean()
CHAR, VARCHAR, LONGVARCHAR	setString()
BINARY, VARBINARY, LONGVARBINARY	setBytes()
DATE	setDate()
TIME	setTime()
TIMESTAMP	setTimestamp()
Qualquer Tipo	setBlob(), setClob()

Exemplo Inclusão com PreparedStatement (1)

```
1 package apps;
2
3 import java.sql.Connection;
4 import java.sql.PreparedStatement;
5 import java.sql.SQLException;
6
7 public class Incluir2 {
8
9     public static void main(String args[]) {
10         Connection conexao = FabricaConexao.obterConexao();
11         PreparedStatement comando = null;
12         String nome = "Refrigerantes";
13         float promocao = 0;
14         float margem = 50;
15         try {
16             comando = conexao.prepareStatement("INSERT INTO GRUPOPRODUTO ( NOME, PROMOCAO, MARGEMLUCRO ) VALUES (?, ?, ?)");
17             comando.setString(1, nome);
18             comando.setFloat(2, promocao);
19             comando.setFloat(3, margem);
20             comando.executeUpdate();
21             System.out.println("Inclusao realizada com sucesso");
22         } catch (SQLException ex) {
23             System.out.println("Erro ao incluir grupo de produto" + ex.toString());
24         } finally {
25             try {
26                 comando.close();
27                 conexao.close();
28             } catch (SQLException ex) {
29                 System.out.println("Erro ao desconectar" + ex.toString());
30             }
31         }
32     }
33 }
```

Exemplo Inclusão com PreparedStatement (2)

```
32     }  
33 }
```

Objeto ResultSet

- Representa uma tabela virtual com o resultado de um comando de consulta
- O método `executeQuery()`, da interface `Statement`, retorna um objeto `ResultSet`
- Representa um cursor para as linhas (tuplas) de uma tabela resultado
- A navegação nas linhas da tabela resultado é realizada por meio dos métodos:
 - `next()` - próxima linha
 - `previous()` - linha anterior
 - `absolute()` - para uma determinada linha (1 - N)
 - `first()` - primeira linha
 - `last()` - última linha
- Métodos para obtenção de dados:
 - `getInt()`
 - `getDate()`
 - `getXXXX()`, onde XXXX equivale aos tipos de dados

Recuperação de Dados do ResultSet

Tipos de Dados SQL	Métodos do ResultSet
INTEGER	getInt()
BIGINT	getLong()
TINYINT	getByte()
REAL	getFloat()
FLOAT, DOUBLE	getDouble()
DECIMAL, NUMERIC	getBigDecimal()
BIT, BOOLEAN	getBoolean()
CHAR, VARCHAR, LONGVARCHAR	getString()
BINARY, VARBINARY, LONGVARBINARY	getBytes()
DATE	getDate()
TIME	getTime()
TIMESTAMP	getTimestamp()
Qualquer Tipo	getBlob(), getClob()

Exemplo Seleção e Uso do ResultSet (1)

```
1 package apps;
2
3 import java.sql.Connection;
4 import java.sql.PreparedStatement;
5 import java.sql.ResultSet;
6 import java.sql.SQLException;
7
8 public class Selecao1 {
9
10     public static void main(String args[]) {
11         Connection conexao = FabricaConexao.obterConexao();
12         PreparedStatement comando = null;
13         try {
14             comando = conexao.prepareStatement("SELECT * FROM grupoproduto ORDER BY nome");
15             ResultSet resultado = comando.executeQuery();
16             while (resultado.next()) {
17                 System.out.println("Codigo: " + resultado.getInt("codigo"));
18                 System.out.println("Nome: " + resultado.getString("nome"));
19                 System.out.println("% Promocao: " + resultado.getFloat("promocao"));
20                 System.out.println("% Margem lucro: " + resultado.getFloat("margemlucro"));
21                 System.out.println("-----");
22             }
23             resultado.close();
24         } catch (SQLException ex) {
25             System.out.println("Erro ao recuperar os grupos" + ex.toString());
26         } finally {
27             try {
28                 comando.close();
29                 conexao.close();
30             } catch (SQLException ex) {
31                 System.out.println("Erro ao desconectar" + ex.toString());
32             }
33         }
34     }
35 }
```

Exemplo Seleção e Uso do ResultSet (2)

```
32  
33  
34  
35 }
```

Exemplo Seleção e Uso do ResultSet (1)

```
1 package apps;
2
3 import java.sql.Connection;
4 import java.sql.PreparedStatement;
5 import java.sql.ResultSet;
6 import java.sql.SQLException;
7 import javax.swing.JOptionPane;
8
9 public class Selecao2 {
10
11     public static void main(String args[]) {
12         Connection conexao = FabricaConexao.obterConexao();
13         PreparedStatement comando = null;
14         int codigo = Integer.parseInt(JOptionPane.showInputDialog("Forneca o codigo a ser pesquisado"));
15         try {
16             comando = conexao.prepareStatement("SELECT * FROM grupoproduto WHERE codigo=?");
17             comando.setInt(1, codigo);
18             ResultSet resultado = comando.executeQuery();
19             if (resultado.next()) {
20                 System.out.println("Encontrado: " + resultado.getString("nome"));
21             } else {
22                 System.out.println("Nao encontrado");
23             }
24         } catch (SQLException ex) {
25             System.out.println("Erro ao recuperar um grupo" + ex.toString());
26         } finally {
27             try {
28                 comando.close();
29                 conexao.close();
30             } catch (SQLException ex) {
31                 System.out.println("Erro ao desconectar" + ex.toString());
32             }
33         }
34     }
35 }
```

Exemplo Seleção e Uso do ResultSet (2)

```
32  
33  
34  
35 }
```

Gerenciamento de Transações

- Permite o gerenciamento da execução de comandos enviados ao banco (em lote ou atômica)
- Implementada através dos métodos do objeto Connection (con)
 - ▶ *con.setAutoCommit(boolean autoCommit)*: default é true
 - ▶ *con.commit()* - efetiva os comandos enviados e pendentes no banco de dados
 - ▶ *con.rollback()* - cancela todos os comandos enviados e pendentes no banco de dados
 - ▶ *Savepoint savept = con.setSavePoint()* - marca um ponto na sequência de comandos pendentes
 - ▶ *con.rollback(savept)* - cancela os comandos pendentes até um ponto marcado
 - ▶ *con.releaseSavepoint(savept)* - remove o ponto de marcação
- Por default, as informações são processadas a medida em que são recebidas
- Para mudar: *con.setAutoCommit(false)*
- Agora várias instruções podem ser acumuladas. Para processar: *con.commit()*
- Se houver algum erro e todo o processo necessitar ser desfeito, pode-se emitir um ROLLBACK usando: *con.rollback()*

Exemplo Inclusão com Retorno de Chave (1)

```
1 package apps;
2 import java.sql.Connection;
3 import java.sql.PreparedStatement;
4 import java.sql.ResultSet;
5 import java.sql.SQLException;
6 import java.sql.Statement;
7 public class Incluir3 {
8
9     public static void main(String args[]){
10         Connection conexao = FabricaConexao.obterConexao();
11         PreparedStatement comando = null;
12
13         String nome = "Bebidas Fermentadas";
14         float promocao = 10;
15         float margem = 40;
16
17         try {
18             String sql = "INSERT INTO GRUPOPRODUTO ( NOME, PROMOCAO, MARGEMLUCRO ) VALUES (?, ?, ?)";
19             comando = conexao.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS);
20             comando.setString(1, nome);
21             comando.setFloat(2, promocao);
22             comando.setFloat(3, margem);
23             comando.executeUpdate();
24
25             ResultSet rs = comando.getGeneratedKeys();
26             long chave=0;
27             if(rs.next()){
28                 chave = rs.getLong("codigo");
29             }
30             System.out.println("Inclusao realizada com sucesso [chave: "+chave+"]");
31         } catch (SQLException ex) {
```

Exemplo Inclusão com Retorno de Chave (2)

```
32         System.out.println("Erro ao incluir grupo de produto" + ex.toString());
33     } finally {
34         try {
35             comando.close();
36             conexao.close();
37         } catch (SQLException ex) {
38             System.out.println("Erro ao desconectar" + ex.toString());
39         }
40     }
41 }
42
43 }
```

Exemplo de Alteração (1)

```
1 package apps;
2
3 import java.sql.Connection;
4 import java.sql.PreparedStatement;
5 import java.sql.SQLException;
6 import javax.swing.JOptionPane;
7
8 public class Alterar1 {
9
10     public static void main(String args[]) {
11         Connection conexao = FabricaConexao.obterConexao();
12         PreparedStatement comando = null;
13
14         int codigo = Integer.parseInt(
15             JOptionPane.showInputDialog("Forneca o código do grupo de produto a ser alterado"));
16         float promocao = Integer.parseInt(
17             JOptionPane.showInputDialog("Forneca o valor da promoção"));
18         float margem = Integer.parseInt(
19             JOptionPane.showInputDialog("Forneca o valor da margem de lucro"));
20
21         try {
22             comando = conexao.prepareStatement("UPDATE grupoproduto SET promocao=?, margemlucro=? WHERE codigo=?");
23             comando.setFloat(1, promocao);
24             comando.setFloat(2, margem);
25             comando.setInt(3, codigo);
26             int contRec = comando.executeUpdate();
27             System.out.println("Alteração realizada com sucesso");
28             System.out.println("Número de registros alterados: "+contRec);
29         } catch (SQLException ex) {
30             System.out.println("Erro ao alterar o grupo" + ex.toString());
31         } finally {
```


Exemplo de Alteração (2)

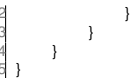
```
32         try {  
33             comando.close();  
34             conexao.close();  
35         } catch (SQLException ex) {  
36             System.out.println("Erro ao desconectar" + ex.toString());  
37         }  
38     }  
39 }  
40 }
```

Exemplo de Exclusão (1)

```
1 package apps;
2
3 import java.sql.Connection;
4 import java.sql.PreparedStatement;
5 import java.sql.SQLException;
6 import javax.swing.JOptionPane;
7
8 public class Excluir1 {
9
10     public static void main(String args[]) {
11         Connection conexao = FabricaConexao.obterConexao();
12         PreparedStatement comando = null;
13
14         int codigo = Integer.parseInt(
15             JOptionPane.showInputDialog("Forneca o código do grupo de produto a ser excluído"));
16
17         try {
18             comando = conexao.prepareStatement("DELETE FROM grupoproduto WHERE codigo=?");
19             comando.setInt(1, codigo);
20
21             int contRec = comando.executeUpdate();
22             System.out.println("Exclusão realizada com sucesso [" + contRec + " excluído]");
23
24         } catch (SQLException ex) {
25             System.out.println("Erro ao excluir o grupo de produto" + ex.toString());
26         } finally {
27             try {
28                 comando.close();
29                 conexao.close();
30             } catch (SQLException ex) {
31                 System.out.println("Erro ao desconectar" + ex.toString());
32             }
33         }
34     }
35 }
```

Exemplo de Exclusão (2)

```
32  
33  
34  
35
```



The diagram shows a vertical list of line numbers 32 through 35. To the right of each line number is a closing curly brace '}'. The braces are positioned at increasing horizontal offsets from the left margin, creating a staircase effect that represents a nested block of code being excluded. Specifically, the brace on line 32 is the furthest to the right, followed by line 33, then line 34, and finally line 35 which has the leftmost brace.

Finalização dos Objetos de Manipulação

- Após o uso, os objetos *Connection*, *Statement* e *ResultSet* devem ser fechados
- Isto pode ser feito com o método `close()`
 - ▶ `con.close()`
 - ▶ `stmt.close()`
 - ▶ `rs.close()`

Informações de Metadados

- **ResultSetMetaData** - permite obter informações sobre o ResultSet
 - ▶ número de colunas
 - ▶ número de linhas existentes na tabela de resultados
 - ▶ nome das colunas etc
- **DatabaseMetaData** - permite obter informações relacionadas ao banco de dados
- **ParameterMetaData** - permite obter informações sobre os parâmetros de um objeto PreparedStatement

Exemplo de Uso do ResultSetMetaData (1)

```
1 package apps;
2
3 import java.sql.Connection;
4 import java.sql.PreparedStatement;
5 import java.sql.ResultSet;
6 import java.sql.ResultSetMetaData;
7 import java.sql.SQLException;
8
9 public class MetaData1 {
10
11     public static void main(String args[]) {
12         Connection conexao = FabricaConexao.obterConexao();
13         PreparedStatement comando = null;
14         try {
15             comando = conexao.prepareStatement("SELECT * FROM produto ORDER BY nome");
16             ResultSet resultado = comando.executeQuery();
17             ResultSetMetaData rsmdt = resultado.getMetaData();
18             System.out.println("Numero de Colunas: " + rsmdt.getColumnCount());
19             for (int x = 1; x <= rsmdt.getColumnCount(); x++) {
20                 System.out.println("Nome da Coluna: " + rsmdt.getColumnName(x));
21                 System.out.println("Classe da Coluna: " + rsmdt.getColumnClassName(x));
22                 System.out.println("Tipo da Coluna: " + rsmdt.getColumnTypeName(x));
23                 System.out.println("Tamanho da Coluna: " + rsmdt.getColumnDisplaySize(x));
24                 System.out.println("-----");
25             }
26             resultado.close();
27         } catch (SQLException ex) {
28             System.out.println("Erro ao recuperar os grupos" + ex.toString());
29         } finally {
30             try {
31                 comando.close();
```

Exemplo de Uso do ResultSetMetaData (2)

```
32         conexao.close();  
33     } catch (SQLException ex) {  
34         System.out.println("Erro ao desconectar" + ex.toString());  
35     }  
36 }  
37 }  
38 }
```

```
Numero de Colunas: 7  
Nome da Coluna: codigo  
Classe da Coluna: java.lang.Integer  
Tipo da Coluna: serial  
Tamanho da Coluna: 11  
-----  
Nome da Coluna: nome  
Classe da Coluna: java.lang.String  
Tipo da Coluna: bpchar  
Tamanho da Coluna: 50  
-----  
Nome da Coluna: estoque  
Classe da Coluna: java.lang.Integer  
Tipo da Coluna: int4  
Tamanho da Coluna: 11  
-----
```

Exemplo de Uso do ResultSetMetaData (1)

```
1 package apps;
2
3 import java.sql.Connection;
4 import java.sql.DatabaseMetaData;
5 import java.sql.ResultSet;
6 import java.sql.SQLException;
7
8 public class Metadata2 {
9
10     public static void main(String args[]) {
11         Connection conexao = FabricaConexao.obterConexao();
12         DatabaseMetaData dbmdt;
13         try {
14             dbmdt = conexao.getMetaData();
15             System.out.println(dbmdt.getDatabaseProductName());
16             System.out.println(dbmdt.getDatabaseProductVersion());
17             System.out.println(dbmdt.getDriverName());
18             System.out.println(dbmdt.getMaxConnections());
19             String tipoTabela[] = {"TABLE"};
20             ResultSet rs = dbmdt.getTables(null, null, null, tipoTabela);
21             System.out.println("Lista de Tabelas: \n");
22             while (rs.next()) {
23                 System.out.println(rs.getString("TABLE_NAME"));
24                 System.out.println(rs.getString("TABLE_SCHEM"));
25                 System.out.println("-----");
26             }
27             rs.close();
28         } catch (SQLException ex) {
29             System.out.println("Erro ao recuperar os grupos" + ex.toString());
30         } finally {
31             try {
```


Exemplo de Uso do ResultSetMetaData (2)

```
32         conexao.close();
33     } catch (SQLException ex) {
34         System.out.println("Erro ao desconectar" + ex.toString());
35     }
36 }
37 }
38 }
```

```
PostgreSQL
9.4.1
PostgreSQL Native Driver
8192
Lista de Tabelas:

grupoproduto
public
-----
produto
public
-----
registrobaixa
public
-----
```

ResultSetMetaData

Assinatura do método:

ResultSet *getTables*(*String* *catalog*, *String* *schemaPattern*, *String* *tableNamePattern*, *String*[] *types*)

- *catalog* - nome do catálogo
- *schemaPattern* - nome do schema
- *tableNamePattern* - parte de nome de tabelas para filtrar
- *types* - tipo da tabela ("TABLE", "VIEW", "SYSTEM TABLE", "GLOBAL TEMPORARY", "LOCAL TEMPORARY", "ALIAS", "SYNONYM")