Conexão Java e BDs

Prof. José P. Alcázar

Agenda

- Introdução
- JDBC
- DAO

Programação por meio de funções: SQL/CLI e JDBC

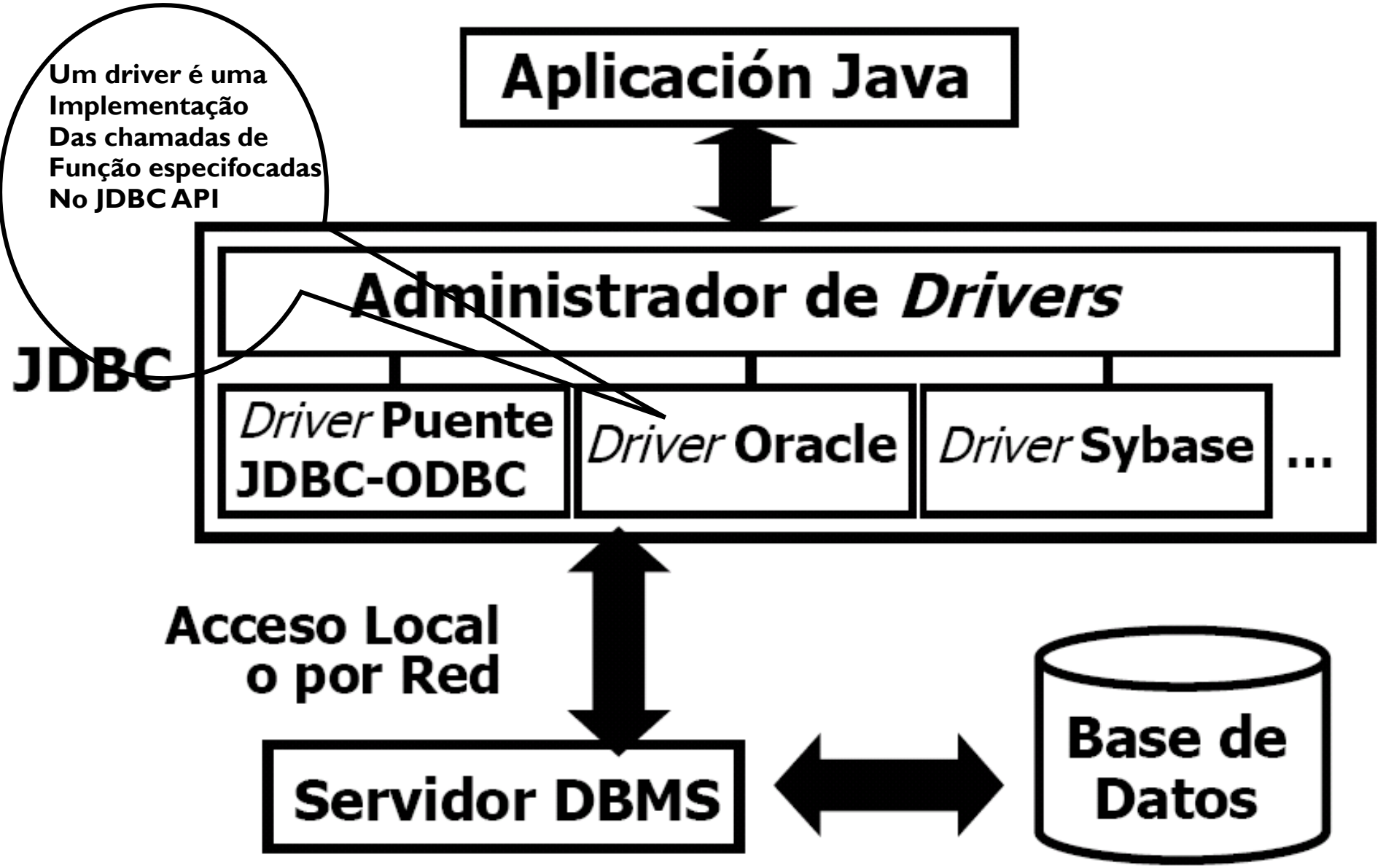
- SQL Embutida é referida como uma abordagem estática de programação de Bds.
- O uso de chamados de função é mais dinâmico – outra abordagem.
- Uma biblioteca de funções , também conhecida como uma interface para a programação de aplicações (API).
- Embora, proporcione maior flexibilidade, uma vez que não é necessário nenhum pré-processador, tem desvantagens:
  - Verificação de sintaxe dos comandos SQL precisa ser feita em tempo de execução.
  - É necessária uma programação mais complexa para acessar o resultado da consulta, porque pode ser que o resultado da consulta não seja conhecido antecipadamente.

Programação por meio de funções: SQL/CLI e JDBC

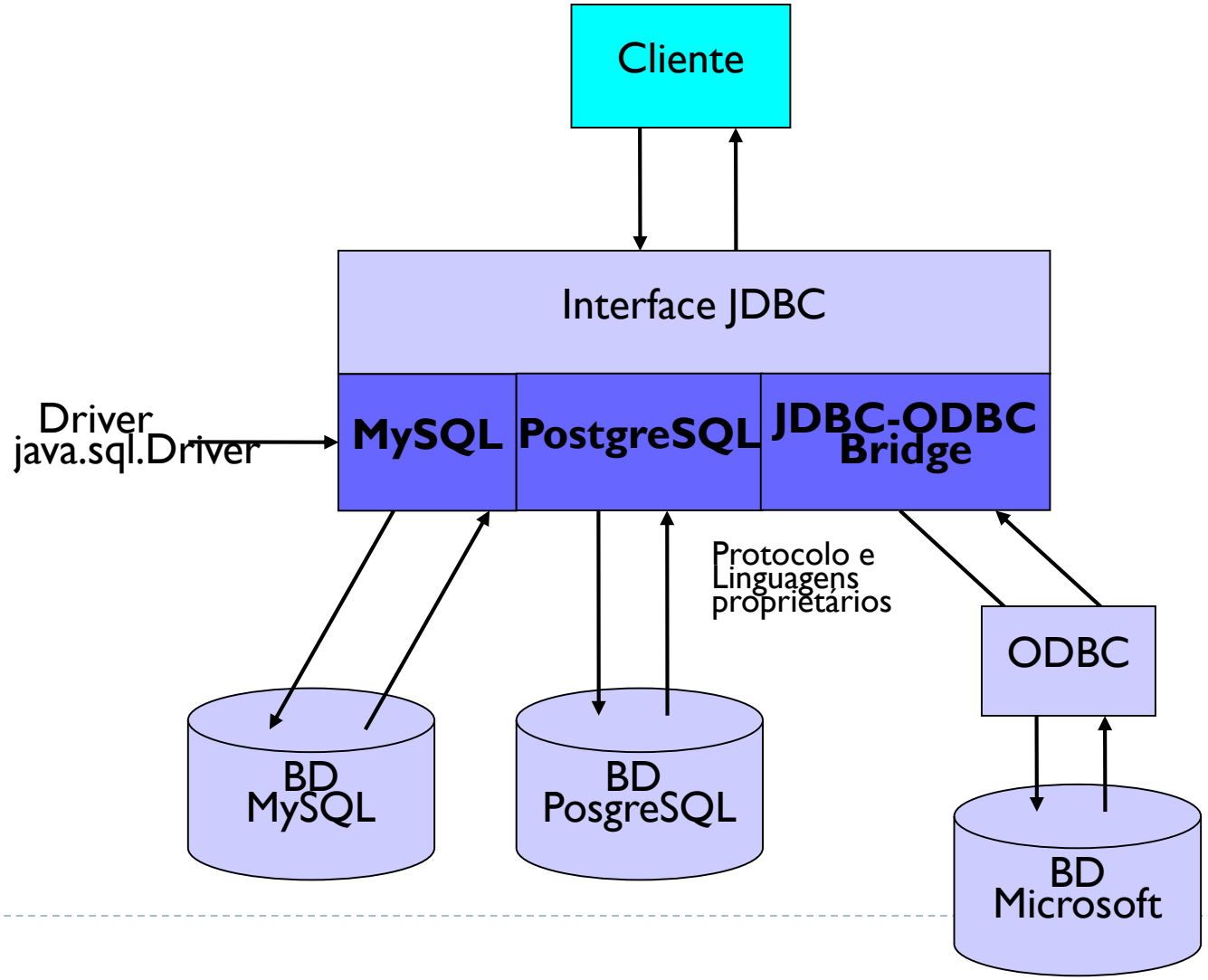
- Duas interfaces de chamada de função:
  - SQL/CLI (Call Level Interface), parte do padrão SQL. Continuação de ODBC (Open Data Base Conectivity). Exemplos com C como hospedeira.
  - JDBC, interface para JAVA. Parte integral de Java desde a versão 1.1.
- A principal vantagem do uso de funções é a maior facilidade de acesso a diversos Bds dentro de um mesmo programa, até mesmo se eles estiverem armazenados em SGBDs diferentes.

JBDC: Chamadas de Função SQL em Programaçao JAVA

- JDBC, é um API de Java para executar enunciados SQL.
- Consiste de um conjunto de classes e interfaces escritas para se ter acesso a dados contidos em BDs relacionais
- É parte integral de Java desde a versão 1.1


Um driver é uma
Implementação
Das chamadas de
Função especifocadas
No JDBC API
Aplicación Java
Administrador de Drivers
JDBC
Driver Puente
JDBC-ODBC
Driver Oracle
Driver Sybase
...
Acceso Local
o por Red
Servidor DBMS
Base de
Datos

Conexão em Java



JDBC

- Antes de poder processar as chamadas de função JDBC de JAVA, é necessário importar as bibliotecas de classes JDBC, que são chamadas java.sql.* Podem ser instaladas e carregadas pela web.
- Foi projetada para permitir que um programa JAVA pudesse conectar-se a vários BDs diferentes (“fontes de dados”)
- Portanto, são necessários drivers de diferentes fabricantes → papel do gerente de drivers. Este registra o driver antes de ser usado. Operações ou métodos: getDriver, registerDriver e deregisterDriver.

JDBC

- Para carregar um driver JDBC, uso de uma função JAVA genérica.
  - "Class.forname("oracle.jdbc.driver.OracleDriver")"
- Os passos típicos são (veja figura):

1. A biblioteca de classes JDBC deve ser importada. Na linha 1 "import java.sql.*" com as outras classes.
2. Carregue o driver JDBC: linhas 4 e 7. A exceção da linha 5 acontece se não for devidamente carregado.
3. Crie as variáveis apropriadas para o programa: linhas 8 e 9.
4. Um objeto de declaração é criado no programa. Classe básica de declaração, Statement, com duas subclasses especializadas: PreparedStatement e CallableStatement. O exemplo ilustra como os objetos de PreparedStatement são criados e utilizados. Linhas 14 e 15.

JDBC

1. O programador deve optar pelo objeto PreparedStatement se uma consulta for executada diversas vezes.
2. O ponto de interrogação (?) da linha 14 representa um parâmetro de declaração. Valor determinado em tempo de execução. Podem haver vários diferenciados pela ordem em que aparecem.
3. Antes de executar uma consulta com PreparedStatement, todos os parâmetros devem ser carregados em variáveis do programa. Dependendo do tipo: setInteger, setString, etc. Veja linha 18.
4. Seguindo estes procedimentos podemos executar a declaração SQL pelo objeto p usando a função executeQuery (linha 19). Existe executeUpdate.
5. Na linha 19, o resultado é devolvido em um objeto r do tipo ResultSet. Semelhante a um cursor de SQL embutido.

- Ao contrário de outras técnicas o JDBC não diferencia entre as consultas que devolvem uma única tupla e várias tuplas.

Segmento de programa JDBC1, um segmento de programa JAVA com o JDBC.

```
//Programa JDBC1:
0) import java.io.* ;
1) import java.sql.*
   …
2) class getEmpInfo {
3)     public static void main (String args []) throws SQLException, IOException {
4)     try { Class.forName("oracle.jdbc.driver.OracleDriver")
5)     } catch (ClassNotFoundException x) {
6)        System.out.println ("Driver nao pode ser carregado") ;
7)        }
8)     String dbacct, senha, ssn, unome ;
9)     Double salario ;
10)    dbacct = readentry("Entre com conta do banco de dados:") ;
11)    passwrd = readentry("Entre com a senha:") ;
12)    Connection conn = DriverManager.getConnection
13)          ("jdbc:oracle:oci8:" + dbacct + "/" + passwrd) ;
14)    String stmt1 = "select UNOME, SALARIO from EMPREGADO where SSN = ?" ;
15)    PreparedStatement p = conn.prepareStatement(stmt1) ;
16)    ssn = readentry("Entre com o Numero do Seguro Social: ") ;
17)    p.clearParameters() ;
18)    p.setString(1, ssn) ;
19)    ResultSet r = p.executeQuery() ;
20)    while (r.next()) {
21)       unome = r.getString(1) ;
22)       salario = r.getDouble(2) ;
23)       system.out.println(unome + salario) ;
24)    }  }
25) }
```

Arquitetura

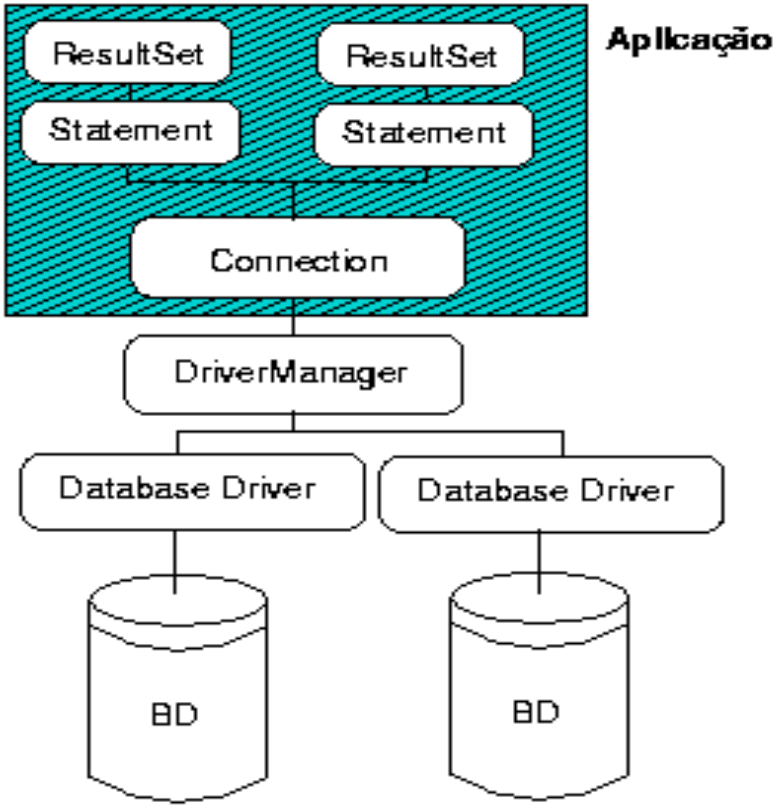
rs.getInt(“id_fornecedor")

statement.executeQuery(“select * from …”)

connection.createStatement()

DriverManager.getConnection(
“jdbc:postgresql://localhost:5432/teste?...“)

Class.*forName*("org.postgresql.Driver");



Segmento de programa JDBC2, um segmento de programa JAVA que usa o JDBC para uma consulta que tem, como resultado, uma coleção de tuplas

```
//Segmento de Programa JDBC2:
0)    import java.io.* ;
1)    import java.sql.*
      …
2)    class printDepartmentEmps {
3)       public static void main (String args []) throws SQLException, IOException {
4)       try { Class.forName("oracle.jdbc.driver.OracleDriver")
5)       } catch (ClassNotFoundException x) {
6)          System.out.println ("O Driver nao pode ser carregado") ;
7)          }
8)       String dbacct, senha, unome ;
9)       Double salario ;
10)      Integer dno ;
11)      dbacct = readentry("Entre com a conta do banco de dados:") ;
12)      passwrd = readentry("Entre com a senha:") ;
13)      Connection conn = DriverManager.getConnection
14)         ("jdbc:oracle:oci8:" + dbacct + "/" + passwrd) ;
15)      dno = readentry("Entre com o Numero do Departamento: ") ;
16)      String q = "select UNOME, SALARIO from EMPREGADO where DNO = " +
         dno.tostring() ;
17)      Statement s = conn.createStatement() ;
18)      ResultSet r = s.executeQuery(q) ;
19)      while (r.next()) {
20)         unome = r.getString(1) ;
21)         salary = r.getDouble(2) ;
22)         system.out.println(unome + salario) ;
23)      }  }
24)   }
```

Executa a consulta diretamente sem preparação.

Como acessar de forma organizada o BD?

- Os programas devem saber se comunicar com o banco de dados. Como fazer isso de maneira adequada? Uma alternativa muito viável é usar o pattern DAO (Data Access Object).
- DAO: o intermediário entre os mundos
- Abstração: o DAO abstrai a origem e o modo de obtenção / gravação dos dados, de modo que o restante do sistema manipula os dados de forma transparente, sem se preocupar com o que acontece por trás dos panos.
- É muito comum em códigos de programadores iniciantes vermos o banco de dados sendo acessada em diversos pontos da aplicação, de maneira extremamente explícita e repetitiva.
- O DAO também nos ajuda a resolver este problema, provendo pontos unificados de acesso a dados. Desse modo, a lógica de interação com o banco de dados fica em lugares específicos e especializados nisso, além de eliminar códigos redundantes, facilitando a manutenção e futuras migrações.

Vantagens do Padrão DAO

- Permite transparência
- Permite migração mais fácil
- Reduz complexidade dos códigos nos objetos de negócios
- Centraliza todo acesso de dados em uma nova camada
- Não útil para a persistência gerenciada pelo container

Fábrica de conexões

Às vezes queremos controlar um processo muito repetiti

```
public class ConnectionFactory {
    public Connection getConnection() {
        try {
            return DriverManager.getConnection("jdbc:mysql://localhe                    ");
        } catch(SQLException e) {
            throw new RuntimeException(e);
        }
    }
}
```

Até a versão 3 do JDBC, antes de chamar o DriverManager.getConnection() era necessário registrar o driver JDBC que iria ser utilizado através do método Class.forName("com.mysql.jdbc .Driver")

getConnection() é uma fábrica de conexões

```
Connection con = new ConnectionFactory().getConnection();
```

Exemplo.

```
create table contatos (
    id BIGINT NOT NULL AUTO_INCREMENT,
    nome VARCHAR(255),
    email VARCHAR(255),
    endereco VARCHAR(255),
    dataNascimento DATE,
    primary key (id)
);
```

Javabeans são classes que possuem o construtor sem argumentos e métodos de acesso do tipo get e set.

Javabeans

```
public class Contato {

    private Long id;
    private String nome;
    private String email;
    private String endereco;
    private Calendar dataNascimento;

    // métodos get e set para id, nome, email, endereço e dataNascimento

    public String getNome() {
        return this.nome;
    }
    public void setNome(String novo) {
        this.nome = novo;
    }

    public String getEmail() {
        return this.email;
    }
    public void setEmail(String novo) {
        this.email = novo;
    }
```

Javabeans

```
    public String getEndereco() {
        return this.endereco;
    }
    public void setEndereco(String novo) {
        this.endereco = novo;
    }

    public Long getId() {
        return this.id;
    }
    public void setId(Long novo) {
        this.id = novo;
    }

    public Calendar getDataNascimento() {
        return this.dataNascimento;
    }
    public void setDataNascimento(Calendar dataNascimento) {
        this.dataNascimento = dataNascimento;
    }
}
```

- Inserir código SQL dentro das classes de lógica é algo nem um pouco elegante e muito menos viável quando você precisa manter o seu código.
- Que tal se pudéssemos chamar um método que adiciona um Contato ao banco?

```
// adiciona um contato no banco
Misterio bd = new Misterio();

// método muito mais elegante
bd.adiciona(contato);
```

Com este código seremos capazes de acessar o BD. Com esta idéia vamos Isolar todo o acesso ao BD em classes bem simples.
Precisamos de uma classe ContatoDAO com um método adiciona.

```
public class ContatoDAO {


    // a conexão com o banco de dados
    private Connection connection;

    public ContatoDAO() {
        this.connection = new ConnectionFactory().getConnection();
    }

    public void adiciona(Contato contato) {
        String sql = "insert into contatos (nome,email,endereco,dataNascimento) values (?,?,?,?)";

        try {
              // prepared statement para inserção
              PreparedStatement stmt = connection.prepareStatement(sql);

             // seta os valores
             stmt.setString(1,contato.getNome());
             stmt.setString(2,contato.getEmail());
             stmt.setString(3,contato.getEndereco());
             stmt.setDate(4, new Date( contato.getDataNascimento().getTimeInMillis() ));

             // executa
             stmt.execute();
             stmt.close();
         } catch (SQLException e) {
             throw new RuntimeException(e);
         }
     }
}
```

Em outro pacote:

```
public class TestaInsere {
public static void main(String[] args) {
        // pronto para gravar
        Contato contato = new Contato( );
        contato.setNome("José Alcazar");
        contato.setEmail("jperezal@gemail.com");
        contato.setEndereco("R.  Jesuino Marcondes Machado, 1000");
         contato.setDataNascimento(Calendar.getInstance());

        // grave nessa conexão
        ContatoDAO dao = new ContatoDAO( );

        // método elegante
        dao.adiciona(contato);

        System.out.println("Gravado!");
    }
}
```

Fazendo buscas no BD.

Crie o método `getLista` na classe `ContatoDAO`. Importe `List` de `java.util`:

```
1 public List<Contato> getLista() {
2     try {
3         List<Contato> contatos = new ArrayList<Contato>();
4         PreparedStatement stmt = this.connection.prepareStatement("select * from contatos");
5         ResultSet rs = stmt.executeQuery();
6
7         while (rs.next()) {
8             // criando o objeto Contato
9             Contato contato = new Contato();
10            contato.setNome(rs.getString("nome"));
11            contato.setEmail(rs.getString("email"));
12            contato.setEndereco(rs.getString("endereco"));
13
14            // montando a data através do Calendar
15            Calendar data = Calendar.getInstance();
16            data.setTime(rs.getDate("dataNascimento"));
17            contato.setDataNascimento(data);
18
19            // adicionando o objeto à lista
20            contatos.add(contato);
21        }
22        rs.close();
23        stmt.close();
24        return contatos;
25    } catch (SQLException e) {
26        throw new RuntimeException(e);
27    }
28 }
```

Em outro pacote:

```
public class TestaLista {
public static void main(String[] args) {
        // Cria um ContatoDAO
        ContatoDAO dao = new ContatoDAO( );

        // Liste os contatos com o DAO
        List<Contato> contatos = dao.getLista( );

        // Itere nessa lista e imprime as informações dos contatos
        for (Contato contato : contatos) {
            System.out.println("Nome: " + contato.getNome( ));
            System.out.println("Email: " + contato.getEmail( ));
            System.out.println("Endereço: " + contato.getEndereco( ));
            System.out.println("Data de Nascimento " + contato.getDataNascimento( ).
            getTime() + "\n");
        }
}
```

Outros Métodos para o seu DAO

Veja primeiro o método `altera`, que recebe um `contato` cujos valores devem ser alterados:

```
1  public void altera(Contato contato) {
2      String sql = "update contatos set nome=?, email=?, endereco=?, dataNascimento=? where id=?";
3
4      try {
5          PreparedStatement stmt = connection.prepareStatement(sql);
6          stmt.setString(1, contato.getNome());
7          stmt.setString(2, contato.getEmail());
8          stmt.setString(3, contato.getEndereco());
9          stmt.setDate(4, new Date(contato.getDataNascimento().getTimeInMillis()));
10         stmt.setLong(5, contato.getId());
11         stmt.execute();
12         stmt.close();
13     } catch (SQLException e) {
14         throw new RuntimeException(e);
15     }
16 }
```

Outros Métodos para o seu DAO

Agora o código para remoção: começa com uma query baseada em um contato, mas usa somente o id dele para executar a query do tipo delete:

```
1 public void remove(Contato contato) {
2     try {
3         PreparedStatement stmt = connection.prepareStatement("delete from contatos where id=?");
4         stmt.setLong(1, contato.getId());
5         stmt.execute();
6         stmt.close();
7     } catch (SQLException e) {
8         throw new RuntimeException(e);
9     }
10 }
```