POO - PROGRAMAÇÃO ORIENTADA A OBJETOS

Conectividade com Banco de Dados

Rodrigo R Silva

Instituto Federal de Educação, Ciência e Tecnologia Sul-Rio-Grandense
Campus Bagé

TADS
Tecnologia em Análise e Desenvolvimento de Sistemas

INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA

Nesta Aula Veremos...

TADS Tecnologia em Análise e Desenvolvimento de Sistemas

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA

Introdução

Conceito

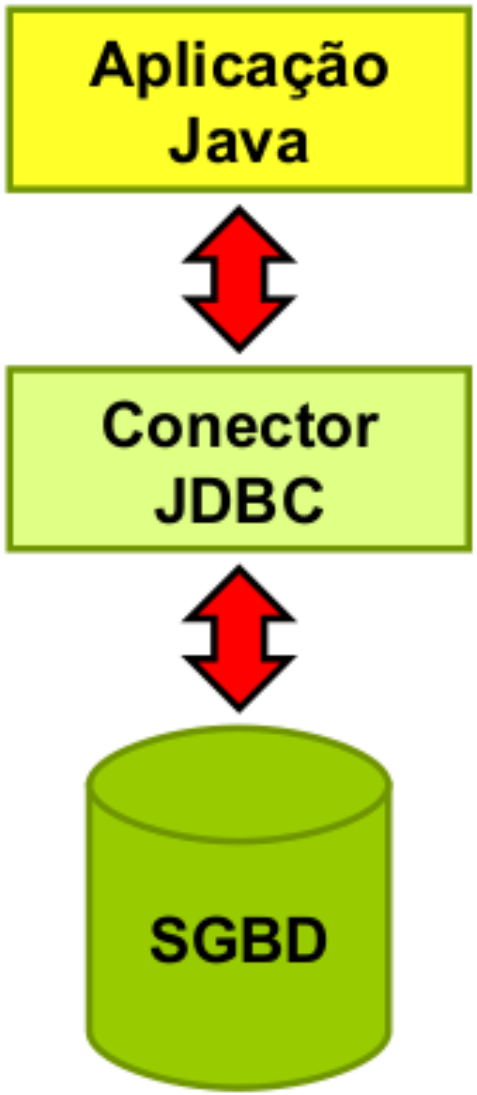
- Java apresenta uma biblioteca de classes e interfaces para conectividade de aplicações com Sistemas Gerenciadores de Banco de Dados (SGBD).
- **JDBC** – Java Database Connectivity.
- A JDBC provê um conjunto de recursos que permitem a uma aplicação cliente realizar conexão e manipulação de dados mantidos por um SGBD.
- A JDBC foi concebida para manipulação de bancos de dados relacionais, tecnologia dominante no mercado atual.

JDBC

JDBC

- As classes e interfaces da JDBC estão contidas no pacote **java.sql** o qual deve ser importado por aplicações que manipulam SGBD em Java.
- Uma aplicação Java realiza conexão com um SGBD através do conceito de **conector** (driver).
- Um conector é um pacote java (jar) que realiza a comunicação de uma aplicação com um SGBD.
- Cada SGBD deve fornecer seu conector para que o mesmo seja utilizado por uma aplicação Java.

Conectividade Java com SGBD



TADS Tecnologia em Análise e Desenvolvimento de Sistemas

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA

Registrando o conector

- Uma conexão Java com SGBD é gerenciada por uma classe denominada DriverManager.
- DriverManager é encarregada de gerenciar todos os conectores de banco de dados de uma aplicação.
- Para isso, a aplicação Java deverá registrar o conector de um banco de dados específico para que possa estabelecer a conexão.
- Somente após o registro do conector é que a aplicação Java poderá solicitar a abertura de uma conexão com um SGBD.

DriverManager

- DriverManager é a classe responsável por:
- Registrar o connector e abrir uma conexão.
- Registrando um conector:
- DriverManager.registerDriver(new org.postgresql.Driver());
- Método utilizado para registro do conector. O exemplo ilustra o registro do conector PostgreSQL.

Estabelecendo uma conexão

- Após o registro do conector no DriverManager já podemos estabelecer uma conexão com o SGBD.
- A conexão é estabelecida pelo método getConnection( ) , veja o exemplo abaixo:
- Connection con = DriverManager.getConnection("jdbc:postgresql://localhost/bd", "postgres", "postgres");
- No exemplo é apresentada uma conexão com PostgreSQL.
- O método getConnection( ) recebe como argumento a string de conexão, usuário e senha do banco. O método retorna um objeto do tipo Connection.

Connection

- Connection é uma interface definida na API JDBC que fornece métodos para:
- Controle de transações (commit e rollback).
- Criação e manipulação de instruções SQL.
- A instância de uma conexão é obtida através do método de classe getConnection() provido por DriverManager.

Métodos de Connection

Métodos

- void setAutoCommit(boolean autoCommit)
  1. Determina se os dados serão gravados no SGBD no momento da execução de uma instrução SQL de atualização de dados.
  2. Uso: con.setAutoCommit(true);
- void commit( )
  1. Esvazia o buffer enviando efetivamente os dados para o SGBD, finalizando a transação.
  2. Uso: con.commit( );

Métodos

- void rollback( )
  1. Não envia os dados para o SGBD, cancelando uma transação que estava em andamento.
  2. Uso: con.rollback( );
- void close( )
  1. Fecha um objeto de conexão com SGBD.
  2. Uso: con.close( );

Métodos

- java.sql.Statement createStatement( )
  1. Usado para criação de uma instrução SQL.
  2. Uso: Statement stmt = con.createStatement( );
- java.sql.PreparedStatement prepareStatement(String sql)
  1. Para criação de uma instrução SQL parametrizada.
  2. Uso: PreparedStatement pstm = con.prepareStatement(“select * from contas where id = ?”);

Statement

Statement

- Um Statement é uma interface da API JDBC utilizada para definição de instruções SQL.
- Possui métodos para execução de instruções de manipulação de dados.
- Principais métodos definidos em Statement:
- int executeUpdate(String sql)
- Utilizado para envio de instruções como insert, update, delete, ou instruções de definição de dados.
- Ex: stmt.executeUpdate(“delete from contas where id = 1”)
- java.sql.ResultSet executeQuery(String sql)
- Para o envio de select.
- Ex: ResultSet rs = stmt.executeQuery(“select * from contas”);

PreparedStatement

- Um PreparedStatement é uma extensão de Statement. A diferença é na utilização de atributos parametrizados.
- Sendo assim, possui métodos para passagem de parâmetros para instrução SQL.
- Vejamos alguns métodos para passarmos parâmetros:
- void setInt(int index, int val)
- void setLong(int index, long val)
- void setFloat(int index, float val)
- void setDouble(int index, double val)
- void setString(int index, String val)
- void setDate(int index, java.sql.Date val)
- void setNull(int index, java.sql.Types.NULL)

PreparedStatement

- Exemplo de utilização:
  String sql = "insert into alunojdbc(nome, email) values(?,?)";
  PreparedStatement statement =connection.prepareStatement(sql);
  statement.setString(1, aluno.getNome());
  statement.setString(2, aluno.getEmail());
  statement.execute();
  connection.commit();

ResultSet

ResultSet

- Um ResultSet é uma Interface utilizada para implementação de um objeto resultante de uma consulta.
- Métodos de ResultSet:
- boolean next(): move o cursor para o próximo dado de um conjunto de dados. O método retorna um boolean informando se a posição é válida ou não.
- Ex: rs.next();
- void close(): fecha um objeto ResultSet.

Métodos de ResultSet

- Os valores dos campos resultantes de uma consulta ao SGBD são recuperados através dos métodos:
- int getInt(String coluna): retorna um campo do tipo inteiro pelo nome do campo.
- int getInt(int posição): retorna um campo do tipo inteiro dada a sua posição no conjunto de dados resultantes.
- Ex:
- int matricula = rs.getInt(“matricula_aluno”);
- int matricula = rs.getInt(1);

Métodos de ResultSet

- Mais alguns métodos. Note que todos os métodos de recuperação de campos podem ser referenciados por nome do campo ou ordem.
- float getFloat(String coluna): retorna um campo do tipo float pelo nome do campo.
- java.sql.Date getDate(int posição): retorna um campo do tipo java.sql.Date dada a sua posição no conjunto de dados resultantes.
- Ex:
- float salario = rs.getFloat(“salario_func”);
- java.sql.Date nascimento = rs.getDate(“nasc”);

Exemplos

Inserindo um registro

```
Connection con;
PreparedStatement pstm;
try {
    DriverManager.registerDriver(new org.postgresql.Driver());
    con = DriverManager.getConnection("jdbc:postgresql://localhost/bd", "postgres", "postgres");
    pstm = con.prepareStatement("insert into contas(id, descricao, tipo, data) values (?, ?, ?, ?)")
    pstm.setInt(1, 209887);
    pstm.setString(2, "Mensalidade escolar");
    pstm.setString(3, "Despesa");
    pstm.setDate(4, new java.sql.Date(new java.util.Date().getTime()));
    pstm.executeUpdate(); //Aqui também poderia ser pstm.execute();
    pstm.close();
    con.close();
} catch (SQLException ex) {
    Logger.getLogger(Conexao.class.getName()).log(Level.SEVERE, null, ex);
}
```

Alterando um registro

```
Connection con;
PreparedStatement pstm;
try {
    DriverManager.registerDriver(new org.postgresql.Driver());
    con = DriverManager.getConnection("jdbc:postgresql://localhost/bd", "postgres", "postgres");
    pstm = con.prepareStatement("update contas set descricao = ?, tipo = ?, data = ? where id = ?");
    pstm.setString(1, "Pagamento de energia elétrica");
    pstm.setString(2, "Despesa");
    pstm.setDate(3, new java.sql.Date(new java.util.Date().getTime()));
    pstm.setInt(4, 209887);
    pstm.executeUpdate(); //Aqui também poderia ser pstm.execute();
    pstm.close();
    con.close();
} catch (SQLException ex) {
    Logger.getLogger(Conexao.class.getName()).log(Level.SEVERE, null, ex);
}
```

Excluindo um registro

```
public Conexao1(){

    Connection con;
    PreparedStatement pstm;
    try {
        DriverManager.registerDriver(new org.postgresql.Driver());
        con = DriverManager.getConnection("jdbc:postgresql://localhost/bd", "postgres", "postgres");
        pstm = con.prepareStatement("delete from contas where id = ?");
        pstm.setInt(1, 209887);
        pstm.executeUpdate(); //Aqui também poderia ser pstm.execute();
        pstm.close();
        con.close();
    } catch (SQLException ex) {
        Logger.getLogger(Conexao.class.getName()).log(Level.SEVERE, null, ex);
    }

}
```

Listando uma consulta

```
Connection con;
PreparedStatement pstm;
ResultSet rs;
try {
    DriverManager.registerDriver(new org.postgresql.Driver());
    con = DriverManager.getConnection("jdbc:postgresql://localhost/bd", "postgres", "postgres");
    pstm = con.prepareStatement("select * from contas order by descricao");
    rs = pstm.executeQuery();
    while(rs.next()){
        System.out.println("ID .........: " + rs.getInt("id"));
        System.out.println("Descrição ..: " + rs.getString("descricao"));
        System.out.println("Tipo .......: " + rs.getString("tipo"));
        System.out.println("Data .......: " +
                new SimpleDateFormat("dd/MM/yyyy").format(rs.getDate("data")));
    }
    pstm.close();
    con.close();
} catch (SQLException ex) {
    Logger.getLogger(Conexao.class.getName()).log(Level.SEVERE, null, ex);
}
```

Transações

```
Connection con = null;
PreparedStatement pstm = null;
ResultSet rs = null;
Savepoint inicioTransacao = null;

try {
    DriverManager.registerDriver(new org.postgresql.Driver());
    con = DriverManager.getConnection("jdbc:postgresql://localhost/bd", "postgres", "postgres");
    con.setAutoCommit(false);
    inicioTransacao = con.setSavepoint();
    pstm = con.prepareStatement("insert into contas(id, descricao, tipo, data) values(?, ?, ?, ?)");
    pstm.setInt(1, 234674);
    pstm.setString(2, "Conta do Telefone");
    pstm.setString(3, "Despesa");
    pstm.setDate(4, new java.sql.Date(new java.util.Date().getTime()));
    pstm.executeUpdate();
    con.commit();
} catch (SQLException ex) {
    System.out.println("Falha na execução da transação, realizando rollback...");
    try {
        con.rollback(inicioTransacao);
    } catch (SQLException ex1) {
        System.out.println("Não foi possível restaurar a transação...");
    }
}
finally{
    try {
        con.close();
    } catch (SQLException ex) {
        System.out.println("Falha no fechamento da conexão...");
    }
}
```

ADS
ologia em Análise e
nvolvimento de Sistemas

DERAL
, CIÊNCIA E TECNOLOGIA

Considerações sobre operações em BD

- Deve-se priorizar blocos de transações quando as operações envolvam instruções interdependentes.
- Para isso use setAutoCommit(false) e crie pontos de restauração (savepoints).
- O padrão de uma conexão é setAutoCommit(true).
- Opte por utilizar PreparedStatements ao invés de Statements. Além de serem mais otimizados, pois são pré-compilados, também são mais seguros.
- Sempre encerre conexões com BD, use close().
- Crie o hábito de tratar possíveis exceções.

OBRIGADO!