



Java Básico

JDBC

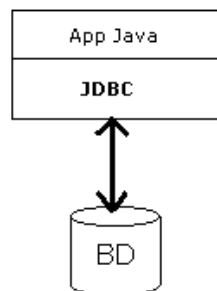
Razer Anthom Nizer Rojas Montaña
(razer@razer.org)
2008



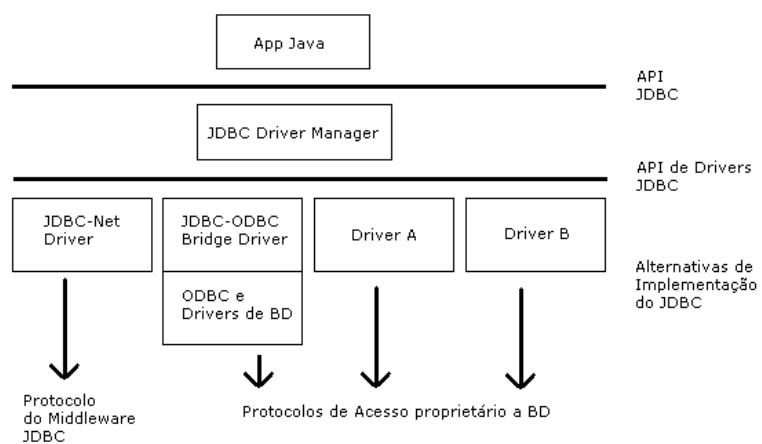
Banco de Dados: JDBC

- API de acesso a banco de dados
- Pacote `java.sql.*`
- Precisa-se de um driver de acesso. Ex para oracle: [`oracle.jdbc.driver.OracleDriver`](#)
- Existe uma ponte JDBC-ODBC: [`sun.jdbc.odbc.JdbcOdbcDriver`](#)
- Acessa uma DSN provida pelo ODBC da Microsoft

JDBC



JDBC



Bancos de Dados Relacionais

- Dados são armazenados baseados em tabelas (relações)

agencia_nome	conta_numero	cliente_nome	saldo
Jardim Social	101	Kafajeste	1049
Passarela	215	Butiá	860
Palácio Avenida	102	Razer	17890
Rua das Flores	305	Churrasqueiro	789
Cristo Rei	217	Ivaru	476

Consulta e Manutenção - SQL

- Comandos para:
 - ☐ Criar tabela
 - ☐ Alterar tabela
 - ☐ Inserir registros
 - ☐ Remover registros
 - ☐ Alterar registros
 - ☐ Buscar registros
 - ☐ etc

Inserção

■ Comando INSERT

```
insert into tb_agencia (  
    agencia_nome,  
    conta_numero,  
    cliente_nome,  
    saldo)  
values (  
    'Agencia da Sun',  
    333,  
    'Sun',  
    100000.0)
```

Armazenamento do Comando

■ Tudo em uma string só:

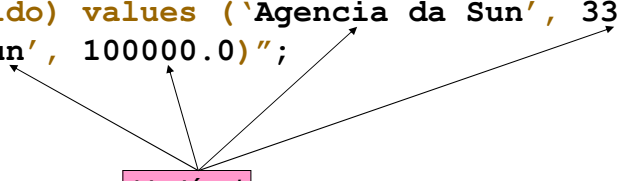
```
String str = "insert into tb_agencia  
(agencia_nome, conta_numero, cliente_nome,  
saldo) values ('Agencia da Sun', 333,  
'Sun', 100000.0)";
```

Usando Variáveis

- Tudo em uma string só:

```
String str = "insert into tb_agencia  
(agencia_nome, conta_numero, cliente_nome,  
saldo) values ('Agencia da Sun', 333,  
'Sun', 100000.0)";
```

Variável



Usando Variáveis

```
String nome = "Agencia Sun";  
int numero = 333;  
String nomeCliente = "Sun";  
double saldo = 100000.0;
```

```
String str = "insert into tb_agencia  
(agencia_nome, conta_numero, cliente_nome,  
saldo) values ('" + nome + "', " + numero + "  
", '" + nomeCliente + "', " + saldo + ")";
```

Remoção

■ Comando DELETE

```
delete from tb_agencia
```

```
delete from tb_agencia
```

```
Where nome_agencia = 'Agencia da Sun'
```

Atualização

■ Comando UPDATE

```
update tb_agencia
```

```
set agencia_nome='Agencia do Razer',
```

```
    conta_numero=777,
```

```
    cliente_nome='Razer',
```

```
    saldo=564
```

```
where agencia_nome='Agencia da Sun' AND
```

```
    cliente_nome='Sun'
```

Busca

■ Comando SELECT

```
select * from tb_agencia
```

```
select cliente_nome  
from tb_agencia  
where agencia_nome='Agencia da Sun'
```

```
select sum(saldo)  
from tb_agencia  
where cliente_nome='Sun'
```

JDBC: Abrindo Conexões

■ Primeiro registra-se o driver:

```
Class.forName(strDriver);
```

■ Onde

□ strDriver: "sun.jdbc.odbc.JdbcOdbcDriver"

■ Usa-se:

```
Connection con = DriverManager.getConnection(  
    url, user, pwd);
```

■ Onde:

□ url: "jdbc:odbc:banco_cliente"

■ Sendo que **banco_cliente** é o DSN no ODBC

□ user: usuário de conexão no banco

□ pwd: senha de conexão do usuário

JDBC: Executando Consultas

- Usa-se o objeto Statement

```
Statement st = con.createStatement();  
ResultSet rs = st.executeQuery(  
    "select * from tb_cliente");
```

- O *ResultSet* acima é usado para tratar o resultado da query

- Para executar alterações:

```
Statement st = con.createStatement();  
st.executeUpdate(  
    "create table teste (cod int, nome char(50))");
```

JDBC: Recuperando Resultados

- Para ir ao primeiro registro depois da consulta
`rs.next()`

- Para recuperar os dados

```
String name = rs.getString("nome");  
int cod = rs.getInt("cod");
```

- Pode-se usar também o índice do campo, ao invés do nome (começa pelo 1)

```
String name = rs.getString(2);  
int cod = rs.getInt(1);
```


JDBC: Recuperando Resultados

- Loop clássico de retorno de valores:

```
while (rs.next()) {  
    str = rs.getString("cliente_nome");  
    i = rs.getInt("cliente_idade");  
    ...  
}
```

JDBC: Instruções Preparadas

- Instruções já conhecidas pelo BD

- Exemplo:

```
PreparedStatement pstmt = con.prepareStatement(  
    "insert into teste (cod, nome) values (?, ?)");
```

- Os '?' são parâmetros

```
pstmt.setInt(1, 1);  
pstmt.setString(2, "Allexia");
```

- Para executar

```
pstmt.executeUpdate();
```

- Se fosse uma query

```
pstmt.executeQuery();
```

JDBC: Stored Procedures

- Usa-se:

```
CallableStatement cst = con.prepareCall(
    "{call sp_teste(?, ? ) } ");
cst.setInt(1, 1);
cst.registerOutParameter(2,
    java.sql.Types.VARCHAR);
cst.executeUpdate();
System.out.println(cst.getString(2));
```

- Para sp's que retornam resultados:

```
{?= call [, , ...]}
```

- Sendo que o primeiro “?” tem número 1 e deve ser registrado como **outParameter**

JDBC: Liberando Recursos

```
rs.close();
st.close();
pstmt.close();
cst.close();
con.close();
```

JDBC: Consulta Fluxo Completo

```
// registrar jdbc-odbc
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
// abrir conexão
Connection con = DriverManager.getConnection(
    "jdbc:odbc:banco_cliente" , "usr", "pwd");
// executar query
Statement st = con.createStatement();
ResultSet rs = st.executeQuery(
    "select * from tb_cliente");
// imprimir resultados
while (rs.next())
    System.out.println(rs.getString("cliente_nome"));
// liberar recursos
rs.close();
st.close();
con.close();
```

Razer Montaña

Java Básico

21

JDBC: Consulta Fluxo Completo

```
try {
    // registrar jdbc-odbc
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    // abrir conexão
    Connection con = DriverManager.getConnection(
        "jdbc:odbc:banco_cliente" , "usr", "pwd");
    // executar query
    Statement st = con.createStatement();
    ResultSet rs = st.executeQuery(
        "select * from tb_cliente");
    // imprimir resultados
    while (rs.next())
        System.out.println(rs.getString("cliente_nome"));
    // liberar recursos
    rs.close();
    st.close();
    con.close();
}
catch (Exception e) {
}
```

Razer Montaña

Java Básico

22

JDBC: Alteração Fluxo Completo

```
try {
    // registrar jdbc-odbc
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    // abrir conexão
    Connection con = DriverManager.getConnection(
        "jdbc:odbc:banco_cliente" , "usr", "pwd");
    // executar query
    Statement st = con.createStatement();
    st.executeUpdate("delete from cliente where id>10");
    // liberar recursos
    st.close();
    con.close();
}
catch(Exception e) {
}
```

JDBC: Transações

- São controladas via **Connection**
 - **void setAutoCommit(boolean)**: Se True, todas as operações serão commitadas assim que executadas
 - **void commit()**: Aplica commit a todas as operações desde o último commit
 - **void rollback()**: desfaz todas as operações desde o último commit/rollback

JDBC: Atualizações em Lote

- Usada nos Statements
- Deve estar dentro de uma transação
- Ao invés de executar `executeUpdate`, usa-se `addBatch`:

```
Statement st = con.createStatement();
String command = "CREATE TABLE x (cod int)";
st.addBatch(command);
while (true) {
    // lê numero
    command = "insert into x values (" + numero + ")";
    st.addBatch(command);
}
// retorna qtas linhas foram afetadas por cada comando
int[] counts = st.executeBatch();
```

JDBC: Aprimoramentos

- Conjunto de resultados **Roláveis**
 - Antigamente, `ResultSet` só ia para frente
 - Agora pode-se navegar para frente e para trás
 - Ou para qualquer posição no `ResultSet`
- Conjunto de resultados **Atualizáveis**
 - Pode-se atualizar um `ResultSet` e refletir estas atualizações no banco de dados

JDBC: Resultados Roláveis

- Cria-se Statement como:

```
Statement st = con.createStatement(  
    ResultSet.TYPE_SCROLL_INSENSITIVE,  
    ResultSet.CONCUR_READ_ONLY);
```

- Todos os ResultSet deste Statement serão roláveis.
Métodos de ResultSet:

- ☐ boolean previous(): Vai para o anterior
- ☐ void relative(n): Vai para o n-ésimo registro relativo ao elemento corrente (>0 p/ frente, <0 para trás)
- ☐ void absolute(n): Vai para n-ésimo registro
- ☐ int getRow(): Número do registro corrente
- ☐ first, last, beforeFirst, afterLast, isFirst, isLast, isBeforeFirst, isAfterLast

JDBC: Resultados Atualizáveis

- Cria-se Statement como:

```
Statement st = con.createStatement(  
    ResultSet.TYPE_SCROLL_INSENSITIVE,  
    ResultSet.CONCUR_UPDATABLE);
```

- Os ResultSet serão Atualizáveis. Métodos de ResultSet:

- ☐ void updateString(String, String): atualiza campos
- ☐ void updateRow(): Envia alterações para o banco de dados. Antes de ir p/ próxima linha, deve ser chamado
- ☐ void cancelRowUpdates(): Cancela atualizações
- ☐ void deleteRow(): Deleta registro corrente
- ☐ void moveToInsertRow(): Vai para linha nova. Deve ser chamada antes dos updateXXX para inserir nova linha
- ☐ void insertRow(): Insere nova linha. Depois dos updateXXX
- ☐ void moveToCurrentRow(): Move o cursor para a posição anterior à chamada de moveToInsertRow

Exercícios

- Fazer um cadastro de pessoas usando Access. Inserção, remoção, atualização, busca, busca de todos. Tudo implementado na classe Pessoa. Deve-se ter uma classe de Teste, para fazer as chamadas à Pessoa. Cuidado, deve-se tratar exceções.

