Desenvolvendo Aplicações usando Bancos de Dados

APIs: Uma alternativa à SQL Embutida

Ao invés da pré-compilação SQL pela aplicação, utilizam-se bibliotecas que fazem chamadas ao BD:

- API (*Application Programming Interface*)
- Interface padrão: procedimentos e objetos
    - ODBC: *Open Database Connectivity*
    - JDBC: API Java
- Manipulação do *Conjunto resultado SQL* de uma forma *language-friendly*

Arquitetura JDBC

Componentes:

- Aplicação:
  - Inicia e termina conexões
  - Submete instruções SQL
- Gerenciador de *driver*
  - Carrega o *driver* JDBC
- *Driver*
  - Conecta a aplicação ao SGBD
  - Transmite requisições
  - Retorna ou traduz resultados e erros
- SGBD
  - Processa instruções SQL

Classes e Interfaces JDBC

Passos para submeter uma consulta ao SGBD:

- Carregar o *driver* JDBC
- Conectar-se ao BD alvo
- Executar instruções SQL

Gerenciador de *driver* JDBC

- Todos os *drivers* são gerenciados pela classe DriveManager
- Carregando o *driver* JDBC:

```
String driverName = “oracle/jdbc.driver.Oracledriver”;
Class.forName (driverName);
```

Conexões no JDBC

- A interação com o BD ocorre através de *sessões* chamadas de conexões
- Exemplo:

```
//caminho do servidor do BD
String serverName = "localhost";
//nome do seu banco de dados
String myBD ="teste";
String url = "jdbc:oracle://" + serverName + "/" + myBD;
String username = "root";
String password = "123456";
Connection connection = DriverManager.getConnection(url,
   username, password);
```

Executando Instruções SQL

- 3 formas diferentes de executar instruções SQL:
  - *Statement*
    - Instruções estáticas e dinâmicas
  - *PrepareStatement*
    - Instruções semi-estáticas
  - *CallableStatment*
    - *Stored procedures*
- *Classe PrepareStatement*:

  Instruções pré-compiladas e parametrizadas:
  - Estrutura é fixa
  - Valores dos parâmetros são determinados em tempo de execução

Executando Instruções SQL

```
String sql="INSERT INTO Vendedor VALUES(?,?,?,?)";
 PreparedStatment pstmt=con.prepareStatement(sql);
 pstmt.clearParameters();
 pstmt.setInt(1,15284); //codigo
 pstmt.setString(2,"Roger Hack"); //nome
 pstmt.setInt(3, 12);//classificação
 pstmt.setFloat(4,42);//idade
```

int numRows = pstmt.executeUpdate();

ResultSets

- *PrepareStatement.executeUpdate* somente retorna o número de registros afetados
- *PrepareStatement.executeQuery* retorna dados encapsulados em um objeto da classe *ResultSet* (um cursor):

```
ResultSet rs=pstmt.executeQuery(sql);
// rs é um cursor
While (rs.next()) {
 // processa cada registro por laço
}
```

ResultSets (cont.)

- Um *ResultSet* é um cursor completo:
  - *previous()*
    - Retorna para o registro anterior
  - *next()*
    - Avança para o próximo registro
  - *absolute(int num)*:
    - Move para o registro de número específico
  - *relative(int num):*
    - Move para frente ou para trás a partir da sua posição corrente
  - *first()* e *last()*

Tipos de Dados Java x SQL

SQL Type	Java class	ResultSet get method
BIT	Boolean	getBoolean()
CHAR	String	getString()
VARCHAR	String	getString()
DOUBLE	Double	getDouble()
FLOAT	Double	getDouble()
INTEGER	Integer	getInt()
REAL	Double	getFloat()
DATE	java.sql.Date	getDate()
TIME	java.sql.Time	getTime()
TIMESTAMP	java.sql.TimeStamp	getTimestamp()

JDBC: Exceções e Alertas

- Classes do pacote *java.sql* podem lançar a exceção *SQLException* caso um erro ocorrer
- *SQLWarning* é uma subclasse:
    - Não são lançados e não interrompem a execução
    - Porém indicam anomalias que devem ser investigadas

JDBC: Exceções e Alertas (cont.)

```
try {
    stmt=con.createStatement();
    warning=con.getWarnings();
    while(warning != null) {
        // tratar SQLWarnings;
        warning = warning.getNextWarning():
    }
    con.clearWarnings();
    stmt.executeUpdate(queryString);
    warning = con.getWarnings();
    …
} //fim try
catch( SQLException SQLe) {
    // tratar a exceção
}
```

Examinando Metadados

- A classe *DatabaseMetaData* provê informações sobre o sistema de BD e seu catálogo

```
DatabaseMetaData md =
  con.getMetaData();
//imprime informações do driver:
System.out.println(
  “Nome:” + md.getDriverName() +
  “versão: ” + md.getDriverVersion());
```

Examinando Metadados (cont.)

```
DatabaseMetaData md=con.getMetaData();
ResultSet trs=md.getTables(null,null,null,null);
String tableName;
While(trs.next()) {
 tableName = trs.getString("TABLE_NAME");
 System.out.println("Tabela: " + tableName);
 //imprimir todos os atributos
 ResultSet crs = md.getColumns(null,null,tableName,null);
    while (crs.next()) {
        System.out.println(crs.getString("COLUMN_NAME" +
  ", ");
    }
}
```

Um Exemplo semi-completo

```
Connection con = // connect
   DriverManager.getConnection(url, ”login", ”pass");
Statement stmt = con.createStatement(); // set up stmt
String query = "SELECT nome, classificacao FROM Vendedor";
ResultSet rs = stmt.executeQuery(query);
try { // tratar exceções
    // itera sobre tuplas retornadas
    while (rs.next()) {
        String s = rs.getString(“nome");
        Int n = rs.getFloat(“classificacao");
        System.out.println(s + "   " + n);
    }
} catch(SQLException ex) {
    System.out.println(ex.getMessage ()
        + ex.getSQLState () + ex.getErrorCode ());
}
```

Stored Procedures

- O que é?
  - Programa executado através de uma instrução SQL
  - Executado como um processo no servidor de BD
  - São **definidas no SGBD** e podem ser acionadas pela aplicação
- Vantagens:
  - Pode encapsular a lógica da aplicação
  - Reutilizar a lógica da aplicação
  - Evite o retorno de um registro de cada vez como ocorre com o uso de cursores

Chamando *Stored Procedures* através de JDBC

```
CallableStatement cstmt=
  con.prepareCall("{call
  mostrarVendedores}");
ResultSet rs = cstmt.executeQuery();
while (rs.next()) {
    …
}
```