



SQL embutida

Prof. Humberto Luiz Razente

humberto.razente@ufu.br

Bloco B - sala 1B144

Introdução

◆ Formas de conexão de aplicações com SGBD

- SQL CLI

- ◆ padrão SQL

- ODBC

- ◆ Open Data Base Connectivity

- JDBC

- ◆ Java Data Base Connectivity

- SQL J

Motivação

- ◆ Fabricantes de SGBD fornecem bibliotecas proprietárias para conexão da sua linguagem de programação com seus produtos
 - necessidade de padronização, para que os desenvolvedores não tenham que entender os detalhes das bibliotecas de cada SGBD

SQL embutida

- ◆ Idéia: incluir sentenças SQL em um programa escrito em uma linguagem de programação hospedeira

```
//Segmento de programa E1:  
0) loop = 1 ;  
1) while (loop) {  
2)     prompt("Digite um CPF: ", cpf) ;  
3)     EXEC SQL  
4)         select Pnome, Minicial, Unome, Endereco, Salario  
5)         into :pnome, :minicial, :unome, :endereco, :salario  
6)         from FUNCIONARIO where Cpf = :cpf ;  
7)     if (SQLCODE == 0) printf(pnome, minicial, unome, endereco, salario)  
8)     else printf("CPF não existe: ", cpf) ;  
9)     prompt("Mais CPF (digite 1 para Sim, 0 para Não): ", loop) ;  
10) }
```

Figura 13.2

Segmento de programa E1, um segmento de programa em C com SQL embutida.

SQL embutida

◆ Cursor

- ponteiro que aponta para uma tupla (linha) do resultado de uma consulta que recupera múltiplas tuplas
- declarado quando o comando de consulta SQL é declarado no programa
- OPEN CURSOR: executa a consulta e posiciona o cursor antes da primeira linha do resultado
- FETCH: move o cursor para a próxima linha do resultado, tornando-a a linha ativa e copiando seus valores para as variáveis do programa especificadas no FETCH por meio da cláusula INTO

```

//Segmento de programa E2:
0) prompt("Digite o Nome do Departamento: ", dnome);
1) EXEC SQL
2)     select Dnumero into :dnumero
3)     from DEPARTAMENTO where Dnome = :dnome;
4) EXEC SQL DECLARE FUNC CURSOR FOR
5)     select Cpf, Pnome, Minicial, Unome, Salario
6)     from FUNCIONARIO where Dnr = :dnumero
7)     FOR UPDATE OF Salario;
8) EXEC SQL OPEN FUNC;
9) EXEC SQL FETCH from FUNC into :cpf, :pnome, :minicial, :unome, :salario;
10) while (SQLCODE == 0) {
11)     printf("O nome do funcionario é:", Pnome, Minicial, Unome);
12)     prompt("digite o valor de aumento: ", aumento);
13)     EXEC SQL
14)         update FUNCIONARIO
15)         set Salario = Salario + :aumento
16)         where CURRENT OF FUNC;
17)     EXEC SQL FETCH from FUNC into :cpf, :pnome, :minicial, :unome, :salario;
18) }
19) EXEC SQL CLOSE FUNC;

```

Figura 13.3

SQLJ

- ◆ Padrão adotado por diversos fabricantes de SGBD
- ◆ Utiliza JDBC para comunicação

SQLJ

```
1) import java.sql.* ;
2) import java.io.* ;
3) import sqlj.runtime.* ;
4) import sqlj.runtime.ref.* ;
5) import oracle.sqlj.runtime.* ;
...
6) DefaultContext cntxt =
7) oracle.getConnection("<url name>", "<user name>",
   "<password>", true) ;
8) DefaultContext.setDefaultContext(cntxt) ;
...
```

Figura 13.5

Importando classes necessárias para incluir SQLJ em programas Java no Oracle e estabelecendo uma conexão e um contexto default.

SQLJ

//Segmento de programa J1:

```
1) cpf = readEntry("Digite o número do CPF: ") ;
2) try {
3)     #sql { select Pnome, Minicial, Unome, Endereco, Salario
4)         into :pnome, :minicial, :unome, :endereco, :salario
5)         from FUNCIONARIO where Cpf = :cpf} ;
6) } catch (SQLException se) {
7)     System.out.println("Número do CPF não existe: " + cpf) ;
8)     Return ;
9) }
10) System.out.println(pnome + " " + minicial + " " + unome + " " + endereco + " " + salario)
```

Figura 13.7

Segmento de programa J1, um segmento de programa Java com SQLJ.

SQL J

//Segmento de programa J2A:

```
0) dnome = readEntry("Digite o nome do departamento: ");
1) try {
2)     #sql { select Dnumero into :dnumero
3)         from DEPARTAMENTO where Dnome = :dnome} ;
4) } catch (SQLException se) {
5)     System.out.println("Departamento não existe: " + dnome) ;
6)     Return ;
7) }
8) System.out.println("Informação do funcionário para departamento: " + dnome) ;
9) #sql iterator Func(String cpf, String pnome, String minicial, String unome, double salario) ;
10) Func f = null ;
11) #sql f = { select cpf, pnome, minicial, unome, salario
12)     from FUNCIONARIO where Dnr = :dnumero} ;
13) while (f.next( )) {
14)     System.out.println(f.cpf + " " + f.pnome + " " + f.minicial + " " + f.unome + " " + f.salario) ;
15) } ;
16) f.close( ) ;
```

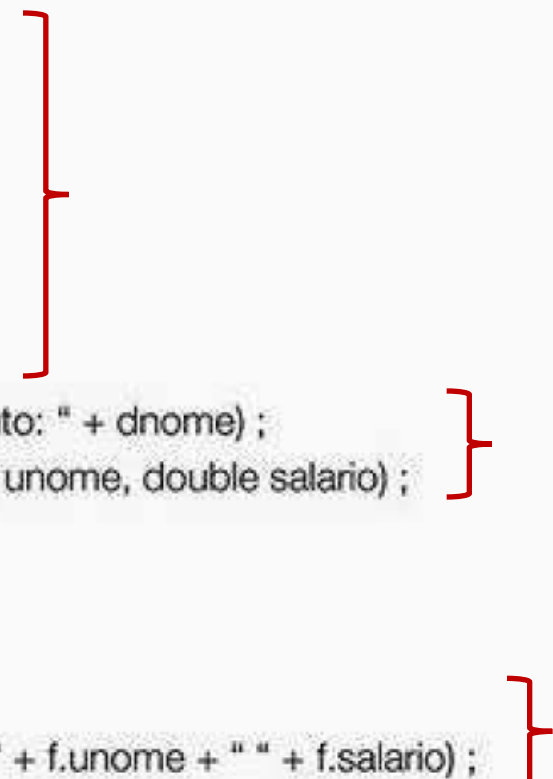


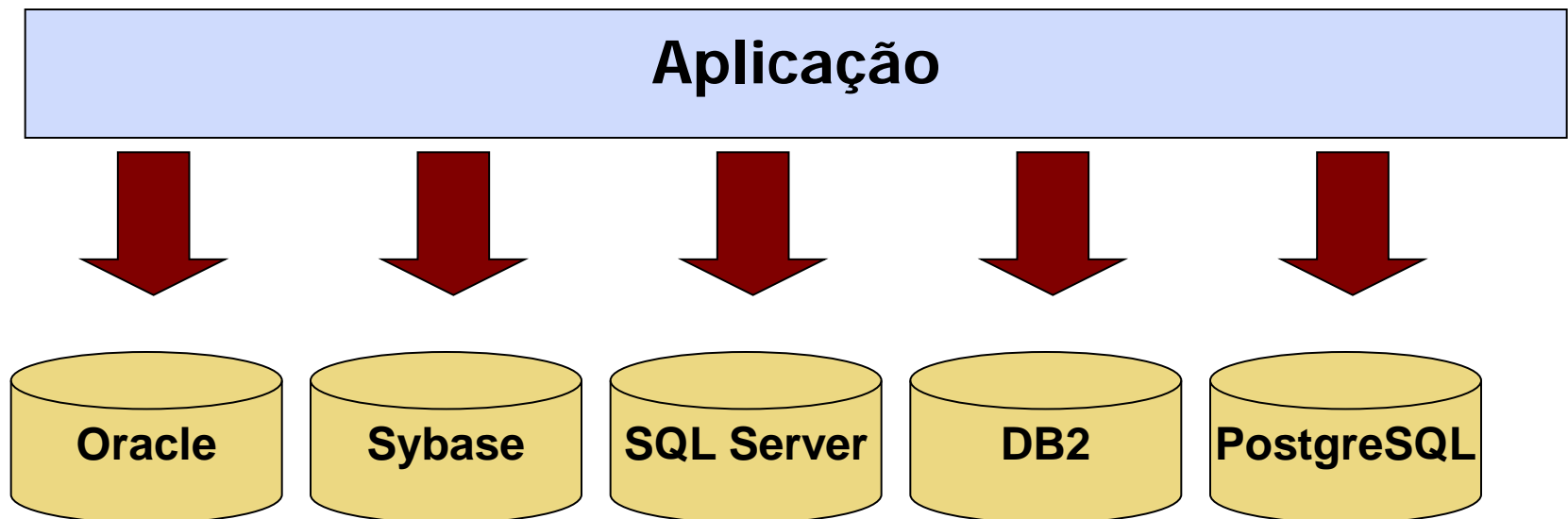
Figura 13.8

Segmento de programa J2A, um segmento de programa Java que usa um iterador nomeado para imprimir informações de funcionário em determinado departamento.

Interfaces de programação

- ◆ Se diferentes SGBD possuem diferentes formas de se comunicar com uma aplicação

Necessidade de aprendizado de várias APIs

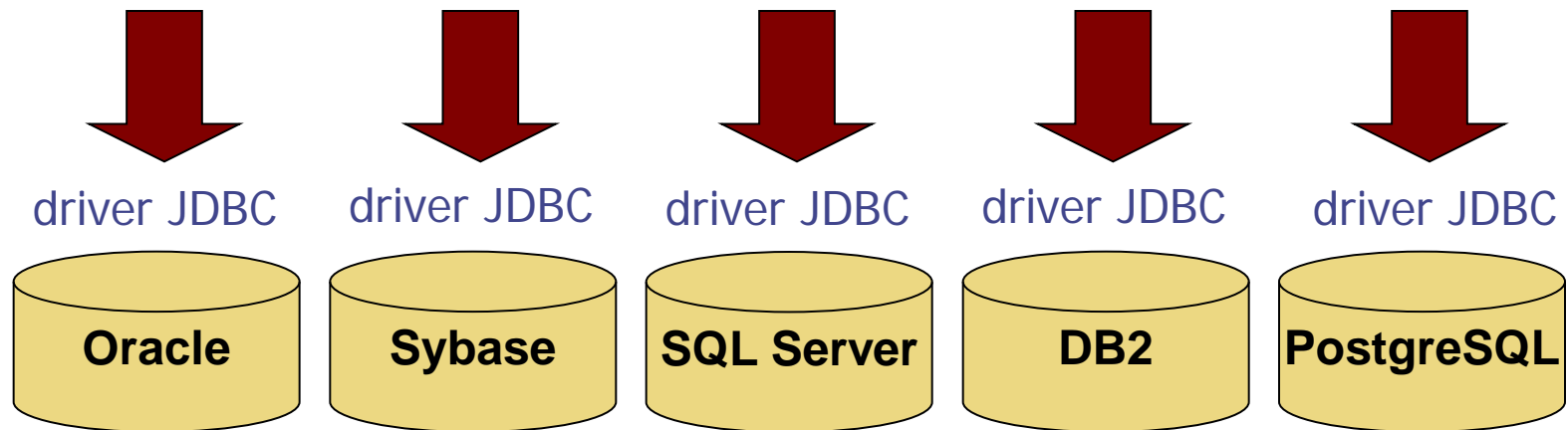


JDBC

◆ Solução:

Aplicação Java

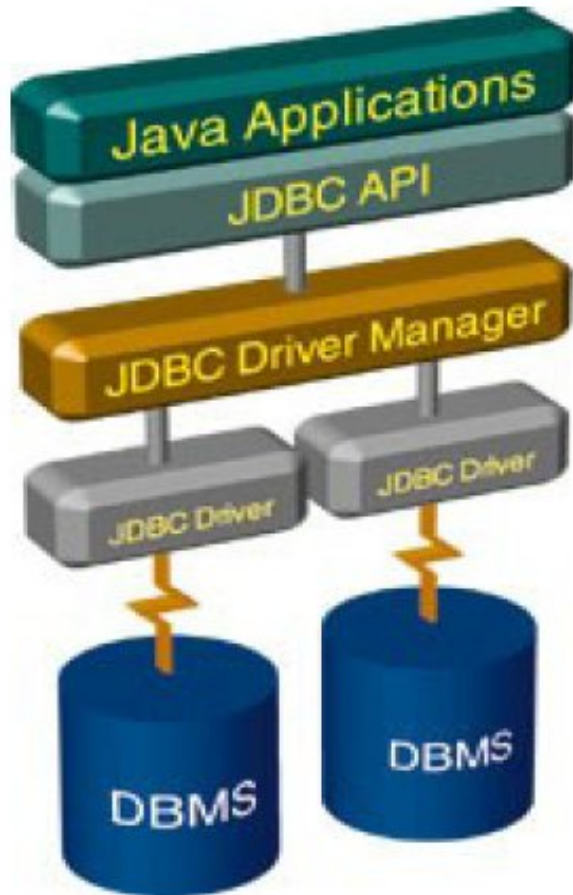
API do JDBC



JDBC - Arquitetura

- ◆ O JDBC provê um conjunto de interfaces para acesso ao BD
 - é um conjunto de APIs (bibliotecas de classes)
- ◆ Cada SGBD possui um Driver JDBC específico (que é usado de forma padrão - JDBC)
- ◆ Os drivers de outros fornecedores devem ser adicionados ao CLASSPATH da aplicação para serem utilizados
- ◆ Mudança do driver não afeta a aplicação
- ◆ Os fabricantes de SGBD são responsáveis por implementar, disponibilizar e atualizar os drivers de acesso para suas bases de dados

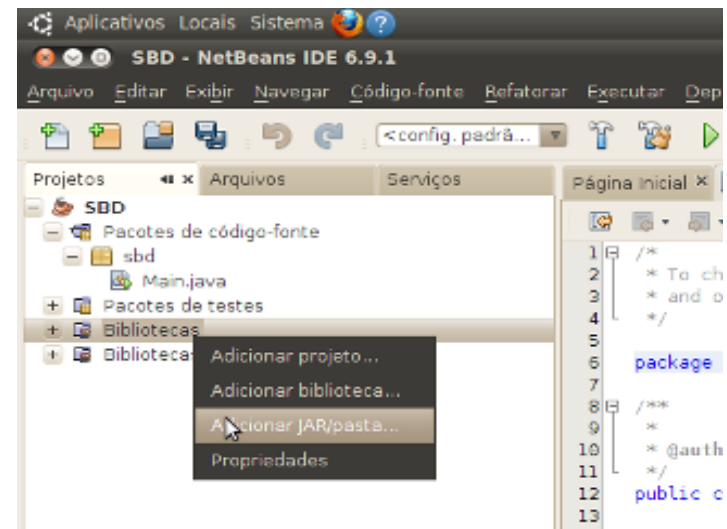
JDBC - Arquitetura



- ◆ `java.sql.*` fornece classes para serem usadas pelas aplicações
- ◆ Atualmente os fabricantes disponibilizam drivers JDBC do tipo 4 (pure Java driver)
 - totalmente implementados em Java
 - arquivo .jar único
 - conhece todo o protocolo de comunicação com o BD e pode acessar o BD sem software extra
- ◆ Obsoletos:
 - Tipo 1: ponte JDBC-ODBC
 - Tipo 2: driver com API nativa
 - Tipo 3: driver de rede
 - ◆ comunica com servidor de aplicação

Utilizando o Driver JDBC com o NetBeans

- ◆ Baixe e instale a ultima versão do driver JDBC para NetBeans (google: driver jdbc postgresql)
- ◆ Abra o NetBeans e crie um novo projeto java.
- ◆ Na lateral esquerda clique na aba projetos e depois clique com o botão direito em
 - Bibliotecas
 - Adicionar JAR/Pasta
 - Selecione o driver JDBC Baixado.



JDBC – Arquitetura

Pacote Java.sql

◆ Principais classes do JDBC

- **DriverManager** - gerencia o driver e cria uma conexão com o banco de dados
- **Connection** - é a classe que representa a conexão com o banco de dados
- **Statement** - controla e executa uma sentença SQL
- **PreparedStatement** – uma subclasse de Statement que controla e executa uma sentença SQL
- **ResultSet** - contém o conjunto de dados retornado por uma consulta SQL
- **ResultSetMetaData** - é a classe que trata dos metadados do banco

JDBC – Arquitetura

Pacote Java.sql

◆ DriverManager

- O método `DriverManager.getConnection (...)` é chamado para efetuar a conexão com o banco de dados

◆ Connection

- `executeUpdate (...)` - utilizados pelos comandos insert, update e delete
- `executeQuery (...)` - utilizado para o comando select

Utilizando o Driver do PostgreSQL

Etapas básicas

1. Criar e Popular o banco de dados
2. Carregar Driver
3. Definir URL para a conexão
4. Estabelecer conexão
5. Criar objeto do tipo statement
6. Executar uma consulta
7. Processar resultado
8. Fechar Conexão

Utilizando o Driver do PostgreSQL

Carregando o Driver

- ◆ Obter o Driver do PostgreSQL (*postgresql.jar*)
 - <http://jdbc.postgresql.org/download.html>
 - JDBC41 Postgresql Driver
- ◆ Adicionar o caminho para o arquivo *postgresql.jar* na configuração do CLASSPATH
 - adicione ao *Projeto* por meio do item *Libraries* (caso esteja usando Netbeans)

Utilizando o Driver do PostgreSQL

Carregando o Driver

◆ Carregando o Driver

```
System.out.print("Checando se o Driver está registrado com DriverManager: ");

try { // Carrega classe de driver do SGBD
    Class.forName("org.postgresql.Driver");
} catch (ClassNotFoundException cnfe) {
    System.out.println("não foi possível achar o driver!");
    cnfe.printStackTrace();
    System.exit(1);
}

System.out.println("o registro do driver está ok!");
```

Utilizando o Driver do PostgreSQL

Criando uma Conexão

```
Connection conexao = null; //gerencia a conexão
Statement sentenca = null; //instrução de consulta

try {
    // Estabelece conexão com o banco de dados
    System.out.print("Conectando com servidor: ");
    String url = "jdbc:postgresql://localhost/postgres?user=postgres&password=12345";
    conexao = DriverManager.getConnection(url);
    System.out.println("conectado!");
    // Cria uma sentença para consultar o banco de dados
    sentenca = conexao.createStatement();
} catch (SQLException se) {
    System.out.println("não foi possível conectar ao banco de dados.");
    se.printStackTrace();
}
```

Utilizando o Driver do PostgreSQL

Criando uma Sentença para alterar esquema

```
try {
    // drop tabela
    sentenca.execute("drop table pessoa");
} catch (SQLException se) { }

try {
    // Criar tabela
    sentenca.execute("create table pessoa(id decimal(10) primary key," +
        " nome varchar(20), endereco varchar(20))");
    // Inserir dados
    sentenca.execute("insert into pessoa values (123,'Andre Silva','Av Brasil 100')");
    sentenca.execute("insert into pessoa values (234,'Joao Bezerra','Av Joao Naves 300')");
    sentenca.execute("insert into pessoa values (345,'Maria Bonita','Av Tiradentes 400')");
    sentenca.execute("insert into pessoa values (456,'Joana Darc','Rua Principal 200')");
} catch (SQLException se) {
    System.out.println("Não foi possível executar a consulta.");
    se.printStackTrace();
    System.exit(1);
}
```

Utilizando o Driver do PostgreSQL

Criando uma Sentença para atualizar o BD

```
try {  
    // executa consulta  
    ResultSet resposta = sentenca.executeQuery("select * from pessoa");  
    // exibe o resultado da consulta  
    ResultSetMetaData metaDados = resposta.getMetaData();  
    int nroDeColunas = metaDados.getColumnCount();  
    for (int i = 1; i <= nroDeColunas; i++)  
        System.out.printf("%-8s\t", metaDados.getColumnName(i));  
    System.out.println();  
  
    // exibir o conteúdo da tabela  
    while (resposta.next()) {  
        for (int i = 1; i <= nroDeColunas; i++)  
            System.out.printf("%-8s\t", resposta.getObject(i));  
        System.out.println();  
    }  
} catch (SQLException se) {  
    System.out.println("Não foi possível executar a consulta.");  
    se.printStackTrace();  
    System.exit(1);  
}
```

Utilizando o Driver do PostgreSQL

Criando uma Sentença para atualizar o BD



PreparedStatement

- A principal característica de um objeto da classe PreparedStatement é que, ao contrário de objetos da classes Statement, ele recebe uma declaração SQL na hora de sua de criação
- Isto permite que a declaração seja enviada ao SGBD para ser compilada na hora de sua declaração. Como resultado, objetos PreparedStatement contêm declarações pré-compiladas
- Quando um PreparedStatement é executado, o SGBD pode executar seu comando SQL imediatamente sem ter que compilá-lo primeiro
 - ◆ é possível passar valores em parâmetros da consulta preparada

Utilizando o Driver do PostgreSQL

Criando uma Sentença para atualizar o BD

◆ PreparedStatement - Exemplo

...

```
String sql = "insert into aluno (matricula,  
    nome, sobrenome) values (?, ?, ?)";
```

```
PreparedStatement st =  
    Conexao.prepareStatement(sql);
```

```
st.setInt(1, mat);
```

```
st.setString(2, nome);
```

```
st.setString(3, sobrenome);
```

```
int nroInsercoes = st.executeUpdate();
```

...

Utilizando o Driver do PostgreSQL

Execução de uma stored procedure

◆ CallableStatement - Exemplo

```
...  
CallableStatement cs =  
    Conexao.prepareCall("procedure");  
cs.setInt(1, mat);  
cs.setString(2, nome);  
ResultSet rs = cs.executeQuery();  
...
```

Utilizando o Driver do PostgreSQL

Encerrando a conexão

```
// fechar conexoes
try {
    sentenca.close();
    conexao.close();
} catch (Exception exception) {
    exception.printStackTrace();
    System.exit(1);
}
```

Bibliografia

- ◆ Deitel, H. M. **Java: como programar**. 6 ed. São Paulo: Pearson Prentice Hall, 2005
 - capítulo 25

- ◆ Elmasri, Ramez; Navathe, Shamkant B. **Sistemas de banco de dados**. 6ª edição, Editora Pearson, 2011
 - capítulo 24 – segurança de banco de dados