
Banco de Dados

Técnicas de Programação de Banco de Dados

DCC-UFLA

Prof. Denilson Alves Pereira

denilsonpereira@dcc.ufla.br

<http://lattes.cnpq.br/4120230814124499>

Introdução

■ Aplicações de banco de dados

- ◆ Desenvolvidas em linguagens hospedeiras e usando a sublinguagem SQL
 - Linguagens de programação de uso geral: C/C++/C#, Cobol, Java,...
 - Linguagens de script: PHP, JavaScript
- ◆ Desenvolvidas em linguagens específicas para escrita de aplicações de BD
 - PL/SQL da Oracle
 - Transact-SQL da Microsoft
- ◆ Usadas como transações programadas para usuários finais

■ Interface interativa

- ◆ Comandos SQL digitados diretamente em um monitor

Técnicas de Programação de BD

(1) Técnica da SQL Embutida

- ◆ Embute comandos de banco de dados em uma linguagem de programação de uso geral
- ◆ Comandos de banco de dados identificados por um prefixo especial na linguagem hospedeira
- ◆ Um pré-compilador ou pré-processador varre o código fonte para identificar os comandos de bd e extraí-los para processamento pelo SGBD
 - Eles são substituídos por chamadas de função ao código gerado pelo SGBD

Técnicas de Programação de BD

(2) Técnica da Biblioteca de Chamadas de Função

- ♦ Uma biblioteca de funções se torna disponível à linguagem de programação hospedeira
 - Ex: funções para conectar ao bd, executar atualização
 - Comandos reais incluídos como parâmetros
- ♦ API – interface de programação de aplicação

Técnicas de Programação de BD

(3) Técnica da Linguagem de Programação de Banco de Dados

- ♦ Cria-se uma linguagem de programação que roda no SGBD
- ♦ Linguagem de programação compatível com o modelo de banco de dados e a linguagem de consulta
- ♦ Contém estruturas como loops e comandos condicionais
- ♦ Ex: PL/SQL da Oracle

Divergência de Impedância

- Diferenças entre o modelo do banco de dados e o modelo da linguagem de programação
- Tipos de dados da linguagem de programação diferem dos tipos de dados de atributos do modelo de dados
 - ◆ Necessário um vínculo para cada linguagem hospedeira especificando os tipos compatíveis
- Resultado de consulta no formato de tabelas
 - ◆ Necessário mapeá-lo para uma estrutura de dados apropriada na linguagem hospedeira
 - Uso de cursor ou variável de iteração

Sequência de Interação Típica

- Abrir uma conexão com o servidor de banco de dados
 - ◆ Especificar URL, login e senha
- Interagir com o banco de dados submetendo consultas, atualizações e outros comandos do banco de dados
- Terminar ou fechar a conexão com o banco de dados

Técnica 1

Técnica da SQL Embutida

SQL Embutida, SQLJ e SQL Dinâmica

- Instruções SQL podem ser embutidas em uma linguagem de programação
 - ◆ SQL Embutida: linguagem C e outras
 - ◆ SQLJ: linguagem Java
 - ◆ As instruções são parte do código fonte
 - Qualquer alteração precisa passar por uma nova compilação
- Instruções SQL podem ser especificadas em tempo de execução
 - ◆ SQL dinâmica

Recuperando Tuplas isoladas com SQL Embutida

- Um pré-processador separa instruções SQL embutidas no código da linguagem hospedeira
 - ◆ EXEC SQL: inicia instrução SQL
 - ◆ EXEC END ou ponto e vírgula (;): termina instrução SQL
- Variáveis compartilhadas são usadas tanto no programa quanto nas instruções SQL embutidas
 - ◆ Iniciadas com dois pontos (:) na instrução SQL

Recuperando Tuplas isoladas com SQL Embutida

- 0) int loop ;
- 1) EXEC SQL BEGIN DECLARE SECTION ;
- 2) varchar dnrme [16], pnome [16],
unome [16], endereco [31] ;
- 3) char cpf [10], datanasc [11], sex [2], minicial [2] ;
- 4) float salario, aumento ;
- 5) int dnr, dnumero ;
- 6) int SQLCODE ; char SQLSTATE [6] ;
- 7) EXEC SQL END DECLARE SECTION ;

Recuperando Tuplas isoladas com SQL Embutida

//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) }
```

Recuperando Múltiplas Tuplas com SQL Embutida

■ Cursor

- ◆ Ponteiro que aponta para uma única tupla (linha) do resultado de uma consulta
- ◆ Declarado junto com o comando de consulta SQL

■ Comando OPEN CURSOR

- ◆ Busca o resultado na consulta no banco de dados
- ◆ Define o cursor para uma posição antes da primeira linha

■ Comando FETCH

- ◆ Move o cursor para a próxima linha do resultado
- ◆ Copia seus valores para as variáveis do programa

■ Comando CLOSE

- ◆ Finaliza o cursor

Recuperando Múltiplas Tuplas com SQL Embutida

//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 ;

Recuperando Múltiplas Tuplas com SQL Embutida

■ Opções gerais de declaração de cursor:

```
DECLARE <nome cursor> [ INSENSITIVE ]  
[ SCROLL ] CURSOR [ WITH HOLD ] FOR  
<especificação da consulta>  
  
[ ORDER BY  
<especificação de ordenação> ] [ FOR READ  
ONLY | FOR UPDATE [ OF <lista de atributos>  
]];
```

- ◆ **FOR READ ONLY:** somente leitura (opção padrão)
- ◆ **FOR UPDATE:** atualização ou exclusão
- ◆ **SCROLL:** cursor pode ser posicionado
 - Orientação de busca acrescentada ao comando **FETCH**
 - ➔ **NEXT, PRIOR, FIRST, LAST, ABSOLUTE i, RELATIVE i**
- ◆ **INSENSITIVE e WITH HOLD:** processamento de transações

Consultas com SQL Dinâmica

- Comandos SQL podem ser especificados dinamicamente em tempo de execução
 - ◆ Não é necessário modificar o código fonte e compilá-lo novamente para novas consultas ou atualizações

//Segmento de programa E3:

- 0) EXEC SQL BEGIN DECLARE SECTION ;
- 1) varchar sqlupdatestring [256] ;
- 2) EXEC SQL END DECLARE SECTION ;
- ...
- 3) prompt("Digite o comando Update: ",
sqlupdatestring) ;
- 4) EXEC SQL PREPARE sqlcommand FROM
:sqlupdatestring ;
- 5) EXEC SQL EXECUTE sqlcommand ;
- ...

SQLJ – SQL Embutida em Java

- SQLJ: padrão adotado por diversos vendedores para embutir SQL em Java
- Necessita de driver JDBC

//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)
```

SQLJ – SQL Embutida em Java

- Iteradores podem ser usados para recuperar múltiplas tuplas
- Iterador é um tipo de objeto associado a uma coleção de registros em um resultado de consulta
- Dois tipos de iteradores
 - ◆ Iterador nomeado: lista nomes e tipos de atributos
 - ◆ Iterador posicional: lista apenas os tipos de atributos

SQLJ – SQL Embutida em Java

//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( ) ;
```

Iterador nomeado

SQLJ – SQL Embutida em Java

//Segmento de programa J2B:

```
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: " + dnrme) ;
6)     Return ;
7) }
8) System.out.println("Informação do funcionário para departamento: " + dnome) ;
9) #sql iterator Funcpos(String, String, String, String, double) ;
10) Funcpos f = null ;
11) #sq f = { select cpf, pnome, inicial, unome, salario
12)     from FUNCIONARIO where Dnr = :dnumero} ;
13) #sql { fetch :f into :cpf, :pn, :mi, :un, :sal} ;
14) while (!f.endFetch( )) {
15)     System.out.println(cpf + " " + pn + " " + mi + " " + un + " " + sal) ;
16)     #sql { fetch :f into :cpf, :pn, :mi, :un, :sal} ;
17) } ;
18) f.close( ) ;
```

Iterador posicional

Técnica 2

Técnica da Biblioteca de Chamadas de Função

Técnica da Chamada de Funções

- Técnica mais dinâmica para programação
- Uma API é usada para acessar o banco de dados
- Não é necessário um pré-processador
- Verificações sobre comandos feita em tempo de execução
- Duas APIs discutidas:
 - ◆ SQL Call Level Interface (SQL/CLI)
 - Parte do padrão SQL
 - Complemento para a ODBC (Open Database Connectivity), que é uma técnica mais antiga
 - ◆ JDBC
 - Acesso a banco de dados por Java

SQL/CLI em C

- Informações sobre as interações do programa com o banco de dados devem ser registradas
- Quatro tipos de registros
 - ◆ Registro de ambiente: recipiente para registrar conexões
 - ◆ Registro de conexão: informações sobre determinada conexão
 - ◆ Registro de instrução: informações para uma instrução SQL
 - ◆ Registro de descrição: informações sobre tuplas ou parâmetros
- Cada registro é acessível ao programa por meio do identificador (ou *handle*) do registro

SQL/CLI em C

//Segmento de programa CLI2:

```
0) #include sqlcli.h ;
1) void imprimeFuncsDepartamento( ) {
2)   SQLHSTMT inst1 ;
3)   SQLHDBC con1 ;
4)   SQLHENV amb1 ;
5)   SQLRETURN ret1, ret2, ret3, ret4 ;
6)   ret1 = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &amb1) ;
7)   if (!ret1) ret2 = SQLAllocHandle(SQL_HANDLE_DBC, amb1, &con1) else exit ;
8)   if (!ret2) ret3 = SQLConnect(con1, "dbs", SQL_NTS, "js", SQL_NTS, "xyz",
   SQL_NTS) else exit ;
9)   if (!ret3) ret4 = SQLAllocHandle(SQL_HANDLE_STMT, con1, &inst1) else exit ;
10)  SQLPrepare(inst1, "select Unome, Salario from FUNCIONARIO where Dnr = ?", SQL_NTS) ;
11)  prompt("Digite o número do Departamento: ", dnr) ;
12)  SQLBindParameter(inst1, 1, SQL_INTEGER, &dnr, 4, &fetchlen1) ;
13)  ret1 = SQLExecute(inst1) ;
14)  if (!ret1) {
15)      SQLBindCol(inst1, 1, SQL_CHAR, &unome, 15, &fetchlen1) ;
16)      SQLBindCol(inst1, 2, SQL_FLOAT, &salario, 4, &fetchlen2) ;
17)      ret2 = SQLFetch(inst1) ;
18)      while (!ret2) {
19)          printf(unome, salario) ;
20)          ret2 = SQLFetch(inst1) ;
21)      }
22)  }
23) }
```


JDBC em Java

- JDBC
 - ◆ Biblioteca de funções para acesso a bancos de dados usando a linguagem Java
- Driver JDBC
 - ◆ API JDBC para um SGBD específico
- Permite que um único programa Java se conecte a vários bancos de dados diferentes

JDBC em Java

■ Objeto Connection

- ◆ Objeto para criar uma conexão com o banco de dados
- ◆ Método getConnection da classe DriverManager

■ Objeto Statement

- ◆ Objeto para execução de comandos SQL
- ◆ Classe Statement e suas subclasses PreparedStatement e CallableStatement

■ Parâmetros de instrução

- ◆ Representado pelo ponto de interrogação (?)
- ◆ Determinado em tempo de execução

■ Objeto ResultSet

- ◆ Objeto de acesso ao resultado de uma consulta
- ◆ Semelhante a um cursor (SQL embutida) ou iterador (SQLJ)

JDBC em Java

```
//Programa JDBC1:
0) import java.io.* ;
1) import java.sql.*
...
2) class obterInfFunc {
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 não pode ser carregado") ;
7)         }
8)         String dbacct, senha, cpf, unome ;
9)         Double salario ;
10)        dbacct = readentry("Digite a conta do banco de dados:") ;
11)        senha = readentry("Digite a senha:") ;
12)        Connection con = DriverManager.getConnection
13)            ("jdbc:oracle:oci8:" + dbacct + "/" + senha) ;
14)        String inst1 = "select Unome, Salario from FUNCIONARIO where Cpf = ?" ;
15)        PreparedStatement p = conn.prepareStatement(inst1) ;
16)        cpf = readentry("Digite um número de CPF: ") ;
17)        p.clearParameters() ;
18)        p.setString(1, cpf) ;
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)    }
```

Técnica 3

Técnica da Linguagem de Programação de Banco de Dados

Procedimentos Armazenados e SQL/PSM

- Procedimentos armazenados
 - ◆ Módulos de programa armazenados pelo SGBD no servidor de banco de dados
 - ◆ Funções ou procedimentos
- SQL/PSM (SQL/Persistent Stored Modules)
 - ◆ Linguagem de programação de banco de dados
 - ◆ Inclui construções de programação de uso geral em SQL

Procedimentos Armazenados

- Módulos armazenados e executados pelo SGBD no servidor de banco de dados
 - ◆ Diferente das técnicas 1 e 2, onde o programa roda na máquina cliente ou no servidor de aplicação em uma arquitetura cliente-servidor de três camadas
 - ◆ Conhecidos como Stored Procedures
- Úteis para:
 - ◆ Utilizar o mesmo código em várias aplicações
 - ◆ Reduzir a transferência de dados e o custo de comunicação entre o cliente e o servidor
 - ◆ Melhorar o poder de modelagem fornecido por visões
 - ◆ Implementar restrições complexas não expressas por asserções e triggers

Procedimentos Armazenados

- Criados pelos comandos:
 - ◆ CREATE PROCEDURE ou CREATE FUNCTION
 - ◆ Permitem parâmetros de entrada e/ou saída
- Após criados, ficam armazenados no servidor e podem ser chamados diversas vezes das interfaces SQL ou linguagens de programação
 - ◆ CALL <nome-do-procedimento-ou-função> (<lista-de-parametros>);

SQL/PSM

- Parte do padrão SQL
- Inclui comandos para:
 - ◆ Criar e chamar procedimentos e funções (slide anterior)
 - ◆ Comandos condicionais (IF-ELSE) (próximo slide)
 - ◆ Comandos de repetição

```
WHILE <condicao> DO  
    <lista de instrucoes>  
END WHILE ;  
REPEAT  
    <lista de instrucoes>  
UNTIL <condicao>  
END REPEAT ;
```

```
FOR <nome do loop> AS <nome do cursor>  
CURSOR FOR <consulta> DO  
    <lista de instrucoes>  
END FOR ;
```


SQL/PSM

//Função PSM1:

- 0) CREATE FUNCTION Tam_dep(IN nrdep INTEGER)
- 1) RETURNS VARCHAR [7]
- 2) DECLARE Nr_de_funcs INTEGER ;
- 3) SELECT COUNT(*) INTO Nr_de_funcs
- 4) FROM FUNCIONARIO WHERE Dnr = nrdep ;
- 5) IF Nr_de_funcs > 100 THEN RETURN "ENORME"
- 6) ELSEIF Nr_de_funcs > 25 THEN RETURN
 "GRANDE"
- 7) ELSEIF Nr_de_funcs > 10 THEN
 RETURN "MEDIO"
- 8) ELSE RETURN "PEQUENO"
- 9) END IF;

Comparando as Três Técnicas

Comparando as Três Técnicas

■ Técnica da SQL Embutida

◆ Vantagens:

- Erros de sintaxe e validação do esquema do banco de dados verificados em tempo de compilação
- Programas mais simples e mais legíveis

◆ Desvantagens:

- Necessidade de recompilação nas mudanças nas consultas
- Não adequado para geração de consultas em tempo de execução

Comparando as Três Técnicas

- Técnica da Biblioteca de Chamadas de Função
 - ◆ Vantagem:
 - Mais flexibilidade, consultas podem ser geradas em tempo de execução
 - ◆ Desvantagens:
 - Programação mais complexa
 - Nenhuma verificação de sintaxe e esquema pode ser feita em tempo de compilação

Comparando as Três Técnicas

■ Técnica da Linguagem de Programação

◆ Vantagens:

- Não sofre do problema de divergência de impedância
- Pode reduzir a transferência de dados e o custo de comunicação entre o cliente e o servidor

◆ Desvantagens:

- Programadores precisam aprender uma nova linguagem
- Algumas linguagens são específicas do vendedor do SGBD

Programação de Aplicações Web com Acesso a Banco de Dados usando a Linguagem PHP

Linguagem PHP

- Linguagem de *scripting* de fonte aberto
- Interpretadores fornecidos gratuitamente
- Disponível na maioria das plataformas de computador
- Muito usada para programação de recursos dinâmicos nas páginas Web
 - ◆ Parte dos dados extraída de bancos de dados
 - ◆ Valores exibidos dinamicamente em páginas HTML

Linguagem PHP

- Em uma arquitetura de três camadas:
 - ◆ O SGBD roda na camada inferior
 - ◆ Os programas em PHP são executados no servidor Web da camada intermediária
 - O programa manipula arquivos HTML para criar páginas dinâmicas
 - Diferente de algumas linguagens de *scripting*, como JavaScript, que são executadas na camada cliente
 - ◆ A página HTML é enviada para a camada cliente para exibição e interação com o usuário

Exemplo em PHP

(a) //Segmento de programa P1:

```
0) <?php
1) // Imprimindo mensagem de boas-vindas se o usuário submeteu
   // seu nome por este formulário HTML
2) if ($_POST['nome_usuario']) {
3)     print("Bem-vindo, ");
4)     print($_POST['nome_usuario']);
5) }
6) else {
7)     // Imprimindo o formulário para entrar com o nome do usuário
   // pois nenhum nome foi informado ainda
8)     print <<<_HTML_
9)     <FORM method="post" action="$_SERVER['PHP_SELF']">
10)    Digite seu nome: <input type="text" name="nome_usuario">
11)    <BR/>
12)    <INPUT type="submit" value="SUBMETER NOME">
13)    </FORM>
14)    _HTML_;
15) }
16) ?>
```

(b)

Digite seu nome:

(c)

Digite seu nome:

(d)

Bem-vindo, João Silva

Figura 14.1

(a) Segmento de programa PHP para a entrada de uma saudação. (b) Formulário inicial exibido pelo segmento de programa PHP. (c) Usuário informa o nome *João Silva*. (d) Formulário imprime saudação para *João Silva*.

Exemplo em PHP

- Detalhes do exemplo de código:
 - ◆ Código PHP é incluído em um arquivo HTML entre as tags `<?php` e `?>`
 - ◆ Comentários: `//` ou `/* */`
 - ◆ **`$_POST`**
 - Variável autoglobal predefinida
 - Vetor que mantém todos os valores inseridos por meio de parâmetros do formulário
 - Vetores são dinâmicos, indexados numericamente ou associativos
 - ◆ Todo o texto entre um `<<<_HTML_` e um `_HTML_;` é impresso no arquivo HTML tal como está

Exemplo em PHP

- Detalhes do exemplo de código:
 - ◆ Nomes de variáveis começam com \$
 - \$_SERVER inclui informações sobre o servidor local
 - action="\$_SERVER['PHP_SELF']" instrui o interpretador a reprocessar o mesmo arquivo quando os parâmetros do formulário forem inseridos pelo usuário
 - Nomes de variáveis diferenciam maiúsculas e minúsculas
 - ◆ Quando o usuário clica no botão SUBMETER, o programa é reprocessado e o valor digitado é capturado por \$_POST['nome_usuario']
- Um programa PHP pode criar diversas variações de texto HTML, dependendo dos caminhos condicionais criados

Recursos Básicos de PHP

- Tipo das variáveis determinados pelos valores atribuídas a elas, podendo mudar durante a execução
- Formas de expressar strings:
 - ◆ Aspas simples: strings literais
 - ◆ Aspas duplas: strings contendo variáveis, as quais são substituídas pelos seus valores
 - ◆ Here documents (heredoc):
 - parte entre **<<<DOCNAME** e **DOCNAME** (sozinho em uma linha)
 - Semelhante às strings com aspas duplas
 - Conveniente para texto de múltiplas linhas
- Operador de concatenação: ponto (.)
- Tipos de dados, operadores de comparação, construções condicionais e de repetição semelhantes à linguagem C

Recursos Básicos de PHP

- 0) `print 'Bem-vindo ao meu Web site.';`
- 1) `print 'Eu disse a ele, "Bem-vindo à casa";`
- 2) `print 'Vamos agora visitar o próximo site';`
- 3) `printf('O preço é $%.2f e o imposto é R$%.2f, $preço, $imposto) ;`
- 4) `print strtolower('AbCdE');`
- 5) `print ucwords(strtolower('JOAO silva'));`
- 6) `print 'abc' . 'efg'`
- 7) `print "envie seu e-mail para: $endereco_email"`
- 8) `print <<<FORM_HTML`
- 9) `<FORM method="post" action="$_SERVER['PHP_SELF']">`
- 10) `Digite seu nome: <input type="text" name="nome_usuario">`
- 11) `FORM_HTML`

Vetores em PHP

- Dois tipos principais de vetores:
 - ◆ Numérico
 - Associa um índice numérico a cada elemento no vetor
 - Começa no zero, incrementado de um
 - ◆ Associativo
 - Pares chave => valor
 - Chaves exclusivas
 - Valores podem ser strings, números ou vetores (permitindo criar vetores multidimensionais)
- Resultados de consultas em bancos de dados usam vetores bidimensionais

Vetores em PHP

```
0) $ensinar = array('banco dados' => 'Silva',
    'SO' => 'Carrick', 'Grafico' => 'Kam');

1) $ensinar['Grafico']          =      'Benson';
    $ensinar['Mineração dados'] = 'Kam';

2) sort($ensinar);

3) foreach ($ensinar as $chave => $valor) {

4)         print " $chave : $valor\n";}

5) $disciplinas = array('Banco dados', 'SO',
    'Grafico', 'Mineração dados');

6) $alterna_cor = array('azul', 'amarelo');

7) for ($i = 0, $num = count($disciplinas); i <
    $num; $i++) {

8)         print '<TR bgcolor="' . $alterna_cor[$i
    % 2] . "'>';

9)         print  "<TD>Disciplina  $i  is</
    TD><TD>$disciplinas[$i]</TD></
    TR>\n";

10) }
```

Funções em PHP

- Funções podem ser usadas para estruturar melhor um programa complexo e compartilhar seções comuns de código
- Passagem de parâmetros por valor
- Variáveis globais podem ser acessadas nas funções
 - ◆ `$GLOBAL['abc']` acessa o valor em uma variável global `$abc`

Funções em PHP

```
0) function professor_disciplina ($disciplina, $atividades_ensino) {
1)     if (array_key_exists($disciplina, $atividades_ensino)) {
2)         $professor = $atividades_ensino [$disciplina];
3)         RETURN "$professor está lecionando $disciplina";
4)     }
5)     else {
6)         RETURN "não existe a disciplina $disciplina";
7)     }
8) }
9) $sensinar = array('Banco dados' => 'Silva', 'SO' => 'Carrick', 'Grafico' => 'Kam');
10) $sensinar['Grafico'] = 'Benson'; $sensinar['Mineracao dados'] = 'Kam';
11) $x = professor_disciplina('Banco dados', $sensinar);
12) print($x);
13) $x = professor_disciplina('Arquitetura Computadores', $sensinar);
14) print($x);
```

Programação de Banco de Dados em PHP

- Biblioteca PEAR DB
 - ◆ Faz parte do PHP Extension and Application Repository (PEAR)
 - ◆ Oferece funções para acesso a banco de dados
- Conectando com um banco de dados:
 - ◆ Carregue o módulo DB.php
 - ◆ `DB::connect('<software SGBD>://<conta do usuário>:<senha>@<servidor de banco de dados>')`
 - Função para criar conexão

Programação de Banco de Dados em PHP

■ Outras funções:

- ◆ DB::isError(\$d)
 - Verifica erros
- ◆ query
 - Envia comando SQL ao servidor
- ◆ setErrorHandling (PEAR_ERROR_DIE)
 - Termina o programa e imprime mensagens de erro padrão se quaisquer erros subsequentes ocorrerem
- ◆ nextID('nome_tabela')
 - Obtém um novo identificador de registro exclusivo para uma tabela

■ Marcadores de lugar

- ◆ Modo mais seguro de executar comandos SQL
- ◆ Especificados pelo símbolo ?
- ◆ Passa-se um vetor contendo os valores para os marcadores

Programação de Banco de Dados em PHP

```
0) require 'DB.php';
1) $d = DB::connect('oci8://conta1:senha12@www.host.com/db1');
2) if (DB::isError($d)) { die("não pode conectar - " . $d->getMessage( ));}
...
3) $q = $d->query("CREATE TABLE FUNCIONARIO
4)           (Func_Id INT,
5)           Nome VARCHAR(15),
6)           Cargo VARCHAR(10),
7)           Dnr INT)");
8) if (DB::isError($q)) { die("criacao de tabela sem sucesso - " .
    $d->getMessage( )); }
...
9) $d->setErrorHandler(PEAR_ERROR_DIE);
...
10) $fid = $d->nextID('FUNCIONARIO');
11) $d = $d->query("INSERT INTO FUNCIONARIO VALUES
12)           ($fid, $_POST['func_nome'], $_POST['func_cargo'], $_POST['func_dnr'])");
...
13) $fid = $d->nextID('FUNCIONARIO');
14) $q = $d->query("INSERT INTO FUNCIONARIO VALUES (?, ?, ?, ?)",
15) vetor($fid, $_POST['func_nome'], $_POST['func_cargo'], $_POST['func_dnr']));
```

Programação de Banco de Dados em PHP

- Funções para recuperar resultados de consulta
 - ◆ FetchRow()
 - Recupera o próximo registro no resultado
 - ◆ getAll('consulta')
 - Obtém todos os resultados de uma consulta para uma única variável

Programação de Banco de Dados em PHP

```
0) require 'DB.php';
1) $d = DB::connect('oci8://conta1:senha12@www.host.com/dbname');
2) if (DB::isError($d)) { die("não pode conectar - " . $d->getMessage( )); }
3) $d->setErrorHandler(PEAR_ERROR_DIE);
   ...
4) $q = $d->query('SELECT Nome, Dnr FROM FUNCIONARIO');
5) while ($r = $q->fetchRow( )) {
6)     print "funcionario $r[0] trabalha para o departamento $r[1] \n" ;
7) }
   ...
8) $q = $d->query('SELECT Nome FROM FUNCIONARIO WHERE Cargo = ? AND Dnr = ?',
9)     vetor($_POST['func_cargo'], $_POST['func_dnr']) );
10) print "funcionarios no dep $_POST['func_dnr'] cujo cargo é $_POST['func_cargo']: \n"
11) while ($r = $q->fetchRow( )) {
12) print "funcionario $r[0] \n" ;
13) }
   ...
14) $allresult = $d->getAll('SELECT Nome, Cargo, Dnr FROM FUNCIONARIO');
15) foreach ($allresult as $r) {
16)     print "funcionario $r[0] tem cargo $r[1] e trabalha para o departamento $r[2] \n" ;
17) }
   ...
```

Bibliografia Básica

ELMASRI, Ramez; NAVATHE, Shamkant B. Sistemas de Bancos de Dados. Pearson Education, 6ª edição, 2011.
ISBN-978-85-7936-085-5

Capítulos 13 e 14