PHP e PostgreSQL

Diego Rodrigo Cabral Silva & George Christian Thó

6 de novembro de 2001

Sumário

1 Introdução

1.1 Client-side scripts

Estes são scripts executados no lado cliente, ou seja, no browser. Esse tipo de script não tem contato com o servidor. Servem para validar formulários e interagir com o usuário em algumas tarefas. Geralmente são implementados em JavaScript. Por eles não terem contato com o servidor, descentralizam o processamento e não provocam tráfego na rede.

1.2 Server-side scripts

Esses são os scripts responsáveis pela criação de páginas em tempo real. Como eles tem contato com o servidor, eles utilizam dados armazenados neles para formar a página. Na verdade, o que existe é um modelo da página que será mesclado com os dados no momento em que a página é requisitada.

1.3 O que é PHP?

PHP é uma linguagem que permite criar sites WEB dinâmicos, possibilitando uma interação com o usuário através de formulários, parâmetros da URL e links. A diferença de PHP com relação a linguagens semelhantes a JavaScript é que o código PHP é executado no servidor, sendo enviado para o cliente apenas HTML puro. Desta maneira é possível interagir com bancos de dados e aplicações existentes no servidor, com a vantagem de não expor o código fonte para o cliente. Isso pode ser útil quando o programa está lidando com senhas ou qualquer tipo de informação confidencial.

O que diferencia PHP de um script CGI escrito em C ou Perl é que o código PHP fica embutido no próprio HTML, enquanto no outro caso é necessário que o script CGI gere todo o código HTML, ou leia de outro aquivo.

1.4 O que pode ser feito com PHP

Basicamente, qualquer coisa que possa ser feita por algum programa CGI pode ser feita também com PHP, como coletar dados de um fomulário, gerar páginas dinamicamente ou enviar e receber *cookies*.

PHP também tem como uma característica muito importante o suporte a um grande número de banco de dados, como dBase, Interbase, mSQL, mySQL, Oracle, Sybase, PostgreSQL e vários outros. Construir uma página baseada em um banco de dados torna-se uma tarefa extremamente simples com PHP.

Além disso, PHP tem suporte a outros serviços através de protocolos como IMAP, SNMP, NNTP, POP3 e, logicamente, HTTP. Ainda é possível abrir *sockets* e interagir com outros protocolos.

2 Enviando os dados para o servidor web

Pode-se definir um programa em três partes principais: entrada de dados, processamento dos mesmos, e o retorno deles para o usuário. Na programação para a web não é diferente. Para que o interpretador PHP possa processar os dados, temos primeiro que passa-los pra o computador onde ele está instalado.

Dentre os métodos implementados pelo HTTP, os dois mais usados por programadores PHP para o envio de dados ao servidor são os métodos GET e POST.

O método GET envia informações através da URL (Uniform Resource Locator). Existem pelo menos duas desvantagens em se utilizar esse método. A primeira é que a quantidade de dados é limitada em 1024 caracteres, o que pode não ser suficiente para determinadas aplicações, a segunda é que os dados enviados aparecem no endereço do arquivo PHP e são visíveis para o usuário, o que não é recomendado para aplicações que utilizam dados sigilosos, como senhas por exemplo.

3 Formulários HTML

3.1 Definindo um formulário

Como todos os componentes em um documento html, um formulário é definido por "tags". A tag `<form>` indica o início do formulário e a tag `</form>` indica o final do mesmo. Dessa maneira é possível colocar vários formulários em um mesmo documento html. Abaixo estão algumas opções da tag `form`.

```
<form name="" action="" method="" enctype="">
```

Onde:

`name:` o nome que identifica o formulário no documento HTML;
`action:` caminho do script que receberá os dados do formulário;
`method:` método de envio dos dados (get ou post);
`enctype:` formato em que os dados serão enviados;

Um exemplo típico de definição de um formulário é:

```
<form name="sinec" action="sinec1.php" method="post">

Elementos do formulário

</form>
```

Os elementos do formulário são identificados pelo script receptor através de um nome. Esse nome, será passado via tag `<input>`, que além disso define outras características importantes como tipo e tamanho do elemento.

3.2 A tag <input>

A maioria dos elementos dos formulários são definidos pela tag `<input>`. Cada tipo de elemento possui parâmetros próprios, mas todos possuem pelo menos dois parâmetros em comum: `type`, que define o tipo de elemento, e `name`, que define o nome daquele elemento.

3.3 Campo de texto

```
<input type="" text="" name="" value="" size="" maxlength="">
```

Exibe na tela um campo para entrada de texto com apenas uma linha.

value - o valor pré-definido do elemento, que aparecerá quando a página for carregada;
size - o tamanho do elemento na tela, em caracteres; **maxlength** - o tamanho máximo do texto contido no elemento, em caracteres.

3.4 Campo de texto com máscara

```
<input type="password" name="" value="" size="" maxlength="">
```

A única diferença deste tipo para o anterior é que serão apresentados asteriscos no lugar do texto. São comumente utilizados para senhas.

3.5 checkbox

```
<input type="checkbox" name="" value="" checked>
```

Utilizado para campos de múltipla escolha, onde o usuário pode marcar mais de uma opção.

value - o valor que será enviado ao servidor quando o formulário for submetido, no caso do campo estar marcado; **checked** - o estado inicial do elemento. Quando presente, o elemento já aparece marcado.

3.6 radio button

```
<input type="radio" name="" value="" checked>
```

Parecido com o tipo anterior. A diferença é que são utilizados para o usuário poder marcar apenas uma opção do grupo. Para agrupar vários elementos deste tipo, basta atribuir o mesmo nome a eles.

3.7 submit button

```
<input type="submit" name="" value="">
```

Utilizado para enviar os dados do formulário para o script descrito na seção "action" da definição do formulário.

value - o texto que aparecerá no botão.

3.8 reset button

```
<input type="reset" name="" value="">
```

Utilizado para fazer todos os campos do formulário voltarem ao valor original, de quando a página foi carregada.

3.9 button

```
<input type="button" name="" value="">
```

Utilizado para ativar funções de scripts client-side (JavaScript, por exemplo).

3.10 textarea

```
<textarea cols="" rows="" name="" wrap="">texto</textarea>
```

Exibe na tela uma caixa de texto, com o tamanho definido pelos parâmetros "cols" e "rows".

cols - número de colunas do campo, em caracteres; **rows** - número de linhas do campo, em caracteres; **wrap** - maneira como são tratadas as quebras de linha automáticas. O valor "soft" faz com que o texto quebre somente na tela, sendo enviado para o servidor o texto da maneira como foi digitado; o valor "hard" envia o texto ao servidor com todas as quebras que aparecem na tela; o valor "off" faz com que o texto não quebre na tela nem quando é enviado ao servidor.

3.11 select

```
<select name="" size="" multiple>
  <option value="">text</option>
</select>
```

Se o valor "size" tiver o valor 1 e não houver o parâmetro "multiple", exibe na tela uma "combo box". Caso contrário, exibe uma "select list".

size - número de linhas exibidas. Default: 1; **multiple** - se presente, permite selecionar mais de uma linha, através das teclas Control ou Shift; **option** - cada item do tipo "option" acrescenta uma linha ao select; **value** - valor a ser enviado ao servidor se aquele elemento for selecionado. Default: o texto do item; **text** - valor a ser exibido para aquele item. Se posiciona entre as tags `<option>` e `</option>`.

3.12 file

```
<input type="file" name="" size="">
```

Exibe na tela um campo de texto e um botão, que ao clicado abre uma janela para localizar um arquivo no disco. Para utilizar este tipo de componente, o formulário deverá utilizar o método "POST" e ter o parâmetro "enctype" com o valor "`multipart/form-data`".

4 Sintaxe básica

4.1 Delimitador de códigos

O código PHP é inserido diretamente no documento HTML e é diferenciado a partir de tags especiais que indicam o seu início e fim. Abaixo estão as várias maneiras de se utilizar códigos PHP em páginas HTML.

```
<?php //tag que indica o início do script
corpo do script
?> //tag que indica o fim do script

<script language=''php''> //tag que indica o início do script
corpo do script
</script> //tag que indica o fim do script

<? //tag que indica o início do script
corpo do script
?> //tag que indica o fim do script

<% //tag que indica o início do script
corpo do script
%> //tag que indica o fim do script
```

Entre esses quatro, os delimitadores de código mais utilizados entre programadores PHP é o terceiro tipo, que é uma abreviação do primeiro. Para utilizá-lo, é necessário habilitar a opção *short-tags* na configuração do PHP, da mesma maneira, para usar o quarto tipo temos que habilitá-lo no arquivo de configuração *php.ini*.

4.2 Separador de instruções

Em PHP, cada instrução deve acabar com um ponto-e-vírgula, da mesma forma que as linguagens C e Perl. Na última instrução do bloco de script não é necessário o uso do ponto-e-vírgula, mas por questões estéticas recomenda-se o uso sempre.

4.3 Variáveis

As variáveis em PHP iniciam com o caracter $ seguido de uma string que identifica a variável. Essa string deve começar com uma letra ou o caracter "_". O nome da variável é caso sensitivo, ou seja, a variável $var é diferente da variável $Var.

```
$var = "Bob";
$Var = "Joe";
echo "$var, $Var";      // Saída "Bob, Joe"
```

4.4 Comentários

Basicamente existem dois tipos de comentários em PHP:

Comentários de uma linha:

Marca como comentário até o final da linha ou até o final do bloco de instruções PHP. Pode ser delimitado pelo caracter "#" ou por duas barras "//".

Comentários de mais de uma linha:

Tem como delimitadores os caracteres "/*" para o início do bloco e "*/" para o final do comentário.

4.5 Imprimindo código HTML

Na maioria dos casos, o objetivo de um programa PHP é gerar um documento HTML. Para que isso seja possível deve-se utilizar uma das funções de impressão: `{echo}` e `{print}`. Para utilizá-las deve-se escolher um dos seguintes formatos:

```
print(argumento);
echo (argumento1, argumento2, ...);
echo argumento;
```

5 Tipos

PHP suporta os seguintes tipo:

- Array
- Ponto flutuante
- Inteiro
- Objeto
- String

O tipo da variável usualmente não é escolhida pelo programador. Isso é decidido em tempo de execução pelo PHP dependendo do contexto no qual ela é usada.

è possível forçar uma variável a ser convertida para um certo tipo através do *typecasting*, ou usando a função settype nela.

5.1 Inteiros

Inteiros podem ser especificados usando qualquer umas das sintaxes abaixo:

```
$a = 1234; # número decimal
$a = -123; # número decimal negativo
$a = 0123; # número na base octal (equivalente ao 83 decimal)
$a = 0x12; # número na base hexadecimal (equivalente ao 18 decimal)
```

O tamanho de um inteiro depende da plataforma. Para uma plataforma de 32 bits o máximo valor inteiro é de 2 bilhões.

5.2 Números em ponto flutuante

Números em ponto flutuante (doubles) podem ser especificados usando qualquer umas das sintaxes abaixo:

```
$a = 1.234;
$a = 1.2e3; // equivalente a 1200
```

5.3 Strings

Strings podem ser especificadas usando um dos conjuntos de delimitadores:

a) utilizando aspas simples (') - desta maneira, o valor da variável será exatamente o texto contido entre as aspas (com exceção de \ e ')

b) utilizando aspas duplas (") - desta maneira, quaquer variável ou caracter de escape será expandido antes de ser atribuído.

```
<?
$teste = "III Sinec";
$teste2 = 'Apostila do $teste de PHP e PostgreSQL\n';
echo "$teste2";
?>
// A saída desse script será: ``Apostila do $teste de PHP e PostgreSQL\n''.

<?
$teste = "III Sinec";
$teste2 = "Apostila do $teste de PHP e PostgreSQL\n";
echo "$teste2";
?>
// A saída desse script será: ``Apostila do III Sinec de PHP e PostgreSQL''.
```

A tabela abaixo lista os caracteres de escape:

Sintaxe	**Significado**
\n	Nova linha
\r	Retorno (semelhante a \n)
\t	Tabulação horizontal
\\	A própria barra (\)
\$	O simbolo $
\'	Aspas simples
\"	Aspas duplas

Tabela 1: Caracteres de escape

5.4 Arrays

Em PHP existem arrays associativos e escalares. Ambos funcionam como tabelas de dispersão, onde os índices são as chaves de acesso aos dados. Por exemplo:

```
$a[0] = "abc";
$a[1] = "def";
$b["sinec"] = 13;
```

Equivalentemente pode-se escrever:

```
<?
$cor = array(1 => "vermelho", 2 => "verde", 3 => "azul", "teste" => 1);
?>
```

6 Operadores

6.1 Operadores aritméticos

Tais operadores devem ser usados com operandos numéricos. Se forem de outro tipo, serão convertidos antes da operação.

+	adição
-	subtração
*	multiplicação
/	divisão
%	módulo (resto da divisão)

Tabela 2: Operadores Aritméticos

6.2 Operadores de strings

O único operador exclusivo para strings vem a seguir:

.	concatenação

Tabela 3: Operador de string

6.3 Operadores de atribuição

Existe um operador básico de atribuição e diversos derivados. No caso dos operadores derivados de atribuição, a operação é feita entre os dois operandos, sendo atribuída o resultado para o primeiro.

=	atribuição simples
+=	atribuição com adição
-=	atribuição com subtração
*=	atribuição com multiplicação
/=	atribuição com divisão
%=	atribuição com módulo
.=	atribuição com concatenação

Tabela 4: Operadores de atribuição

exemplo:

```
<?
$a = 7; $a passa aconter o valor 7
$a += 2; // $a passa a conter o valor 9
?>
```

6.4 Operadores bit a bit

Comparam dois números bit a bit.

&	"e" lógico
|	"ou" lógico
^	ou exclusivo
~	não (inversão)
«	shift left
»	shift right

Tabela 5: Operadores bit a bit

6.5 Operadores lógicos

Utilizados para inteiros representando valores booleanos.

and	"e" lógico
or	"ou" lógico
xor	ou exclusivo
!	não (inversão)
&&	"e" lógico
||	"ou" lógico

Tabela 6: Operadores lógicos

Existem dois operadores para "e" e para "ou" porque eles tem diferentes posições na ordem de precedência.

6.6 Operadores de comparação

As comparações são feitas entre os valores contidos nas variáveis, e não as referências. Sempre retornam um valor booleano.

==	igual a
!=	diferente de
<	menor que
>	maior que
<=	menor ou igual a
>=	maior ou igual a

Tabela 7: Operadores lógicos

6.7 Operadores de expressão condicional

Existe um operador de seleção ternário que funciona da seguinte forma:

```
(expressão1)?(expressão2):(expressão3)
```

O interpretador PHP checa se a primeira expressão é verdadeira. Se for, a expressão retorna o valor da expressão2, senão, retorna o valor da expressão3.

6.8 Operadores de incremento e decremento

++	incremento
- -	decremento

Tabela 8: Operadores lógicos

Esses operadores podem ser usados antes ou depois da variável. Quando usados antes, retornam o valor da variável antes de incrementá-la ou decrementá-la. Quando utilizados depois, retornam o valor da variável já incrementado ou decrementado.

```
<?
$a = $b = 10; // ambas recebem 0 valor 10.
$c = $a++; // $c recebe 10 e $a passa a ter 11
$d = ++$b; // $d recebe 11, valor de $b já incrementado
?>
```

7 Estruturas de controle

As estruturas de controle que veremos a seguir são comuns para as linguagens de computação imperativas, bastando, portanto, descrever a sintaxe de cada uma delas, resumindo o funcionamento.

7.1 Blocos

Um bloco consiste de vários comandos agrupados com o objetivo de relacioná-los com determinada estrutura de controle ou função. Em estruturas de controle como `if`, `for`, `while`, `switch` e em declarações de funções blocos podem ser utilizdos para permitir que um comando faça parte do contexto desejado. Blocos em PHP sao delimitados pelos caracteres "{" e "}". Por exemplo:

```
<?
if ($x == $y)
  comando1;
  comando2;
?>
```

Para que o comando2 estaja relacionado ao `if` é preciso utilizar um bloco:

```
<?
if ($x == $y) {
  comando1;
  comando2;
}
?>
```

As várias estruturas de controle serão explicadas a seguir:

7.2 If

O mais simples e de grande importância para muitas linguagens é o `if`, inclusive PHP. Ele testa a condição e executa o comando indicado se o resultado for verdadeiro (diferente de zero). Ele possui duas sintaxes:

```
if (expressão)
  comando1;

if (expressão):
  comando1;
  comando2;
  . . .
  comando3;
endif;
```

Para incluir mais de um comando no `if` da primeira sintaxe, é preciso utilizar um bloco, demarcado por chaves.

O `else` complemnta o if (se necessário). Seus comandos são executados caso a expressão avaliada pelo `if` seja falsa (zero).

```
if ($x == $y)
  comando1;
else
  comando2;
```

```
if (expressão):
  comando1;
  comando2;
  . . .
  comando3;
else
  comando4;
  comando5;
  . . .
  comando6;
endif;
```

Exemplo de `if` usado com `else`:

```
if ($x > $y)
  $maior = $x;
else
  $maior = $y;
```

O `elseif` é uma combinação de `if` e `else`. Ele estende um `if` para executar um diferente comando ou bloco se a expressão avaliada pelo `if` for falsa. Nesse caso, ele avalia outra expressão e executa o comando ou bloco indicado se tal expressão for verdadeira. Por exemplo:

```
if ($a > $b) {
    print "a é maior que b";
} elseif ($a == $b) {
    print "a é igual a b";
} else {
    print "a é menor que b";
}
```

7.3 While

O loop `while` é a maneira mais fácil de se fazer um loop em PHP. A maneira básica de se fazer um loop while é:

```
while (expressão) comando;
```

A expressão é checada a cada vez que o comando ou bloco de instruções termina, além do teste inicial. Se o valor da expressão passar a ser falso no meio do bloco de instruções, a execução segue até que o bloco termine. Se no teste inicial a expressão for falsa, o bloco não será executado.

Uma outra sintaxe para o uso do `while` é:

```
while (expressão):
  comando1;
  . . .
  comando2;
endwhile;
```

Um exemplo simples de um loop `while` com as duas sintaxes pode ser:

```
/* exemplo 1 */

$i = 1;
while ($i <= 10) {
    print $i++;  /* o valor de $i deve ser
    impresso antes de ser
                    incrementado */
}

/* exemplo 2 */

$i = 1;
while ($i <= 10):
    print $i;
    $i++;
endwhile;
```

7.4 do..while

O laço `do..while` é muito parecido com o `while` exceto que a expressão é checada no final de cada execução do bloco e não no começo. A principal diferença de um loop `while` é que a primeira iteração de um `do..while` é executada de qualquer maneira.

Existe apenas uma sintaxe para o `do..while`. Que é a seguinte:

```
$i = 0;
do {
   print $i;
} while ($i>0);
```

O loop acima deverá rodar apenas uma vez. Quando a expressão for avaliada, será entendida como falsa e a execução do laço acabará.

7.5 for

O `for` é o laço mais complexo em PHP. As sintaxes para o laço `for` são:

```
for (inicialização; condição; incremento)
  comando;

for (inicialização; condição; incremento):
  comando1;
  comando2;
  . . .
  comando3;
endfor;
```

Onde:

inicialização: comando que será executado antes do início do laço. Usado para inicializar as variáveis.
condição: expressão avaliada a cada iteração, que define se os comandos que estão dentro do laço serão executados ou não. Enquanto a expressão for verdadeira os comandos serão executados.
incremento: comando executado ao final de cada execução do laço. Usado para incrementar as variáveis.

7.6 break

O comando `break` pode ser utilizados em laços `do`, `for` e `while`, além do uso no `switch`. Ao encontrar um `break` dentro de um desses laços, o interpretador PHP pára imediatamente a execução do laço, seguindo normalmente o fluxo do script. O `break` ainda aceita um argumento numérico opcional que indica o nível de laços que ele irá sair. Por exemplo:

```
$i = 0;
while (++$i) {
    switch ($i) {
    case 5:
        echo "At 5<br>\n";
        break 1;  /* ou apenas break, sai apenas do switch. */
```

```
    case 10:
        echo "At 10<br>\n";
        break 2;  /* sai do switch e do while. */
    default:
        break;
    }
}
```

7.7 continue

O `continue` também é usado dentro de laços. Ele serve para indicar ao interpretador PHP para ignorar o restante do bloco corrente e começar no início da próxima iteração. Assim como o `break`, o `continue` também aceita um argumento numérico opcional que serve para indicar quantos níveis de encapsulamento ele deveria ignorar. Por exemplo:

```
while (list ($key, $value) = each ($arr)) {
    if (!($key % 2)) { // pular valores pares
        continue;
    }
    fazer_algo_com_o_valor_ímpar($value);
}

 $i = 0;
while ($i++ < 5) {
    echo "Fora<br>\n";
    while (1) {
        echo "Fronteira<br>\n";
        while (1) {
            echo "Dentro<br>\n";
            continue 3;
        }
        echo "Isso nunca será impresso.<br>\n";
    }
    echo "Nem isso.<br>\n";
}
```

7.8 switch

O comando `switch` atua de maneira semelhante a uma série de comandos `if` na mesma expressão. Em muitas ocasiões o programador pode querer comparar a mesma variável (ou expressão) com muitos diferentes valores, e executar um diferente pedaço de código dependendo de qual valor a variável corresponde.

Os seguintes exemplos mostram duas maneiras de se escrever a mesma coisa. Uma usando uma série de `if'se` a outra usando o `switch`.

```
if ($i == 0) {
    print "i equals 0";
}
if ($i == 1) {
    print "i equals 1";
}
if ($i == 2) {
    print "i equals 2";
}

switch ($i) {
    case 0:
        print "i equals 0";
        break;
    case 1:
        print "i equals 1";
        break;
    case 2:
        print "i equals 2";
        break;
}
```

Nesse caso o `break` é utilizado quando se executar apenas uma das opções, visto que o `switch` testa linha a linha os `cases` encontrados, e a partir do momento que encontra um valor igual ao da variável testada, passa a executar todos os comandos seguintes, mesmo os que fazem parte de outros testes.

8 Funções

8.1 Definindo funções

A sintaxe básica para definir uma função é:

```
function nome_da_função([arg1, arg2]) {
  comanos;
  . . .
  [return <valor de retorno>];
}
```

Como a checagem de tipos em PHP é dinâmica, o tipo de retorno não deve ser declarado.

8.2 Valor de retorno

O retorno de uma função em PHP é opcional, ou seja a função pode tanto devolver um resultado de uma operação, como simplesmente executar um pedaço de código. Não é possível que uma função retorne mais de um valor, mas pode se fazer a função retornar um valor composto, como listas ou arrays.

8.3 Argumentos

Opcionalmente, pode-se passar dados de entrada para as funções. Tais dados são chamados "argumentos". Quando são usados, são declarados logo após o nome da função, e tronam-se variáveis pertencentes ao escopo local da função.

Assim como o tipo de retorno da função, os tipos dos argumentos não devem ser especificados.

```
function imprime($texto) {
  echo ‘‘$texto’’;
}
imprime(‘‘III Sinec’’);
```

Passagem de parâmetros por referência

Normalmente a passagem de parâmetros em PHP é feita por valor, ou seja, se o conteúdo da variável for alterado, essa alteração não afeta a variável original.

```
function incrementa($num) {
  $num++;
}
$a = 5;
incrementa($a); // $a ainda vale 5
```

No caso acima, como a passagem de parâmetros é por valor, o valor da variável $a não é alterado. Se a passagem fosse por referência, a variável $a teria assumido o valor 6, depois da execução da função.

Existem duas maneiras de se passar valores por referência: pode-se indicar na declaração da função, ou na chamada da mesma. Quando se indica a passagem por referência na declaração da função, tal passagem de argumentos sempre será por referência. Em ambos os casos se utiliza o caracter "&" para indicar que a passagem será por referência.

```
function incrementa(&$num1, $num2) {
  $num1++;
  $num2++;
}
$a = $b = 5;
incrementa($a, $b); /* neste caso $a vale 6 e $b continua a valer 5 no término da execução da

incrementa($a, &$b); /* aqui, as dus variáveis terão seus valores alterados */
```

Argumentos com valores pré-definidos (default)

Em PHP é possível ter valores *default* para argumentos de funções, ou seja, valores que serão assumidos em caso de nada ser passado no lugar do argumento. Quando um parâmetro é declarado desta maneira, a passagem do mesmo na chamada da função torna-se opcional. Por exemplo:

```
function imprime($texto = ‘‘III Sinec’’) {
  echo ‘‘$texto’’;
}
imprime(); // imprime ‘‘III Sinec’’
imprime(‘‘123 testando’’); // imprime ‘‘123 testando’’
```

Contexto

O contexto é o conjunto de variáveis e seus respectivos valores num determinado ponto do programa. Na chamada de uma função, ao iniciar a execução do bloco que contém a implementação da mesma é criado um novo contexto, contendo as variáveis declaradas dentro do bloco, ou seja, todas as variáveis utilizadas dentro daquele bloco serão eliminadas ao término da execução da função.

Escopo

O escopo de uma variável em PHP define a porção do programa onde ela pode ser utilizada. Na maioria dos casos todas as variáveis têm escopo global. Entretanto, em funções definidas pelo usuário um escopo local é criado. Uma variável de escopo global não pode ser utilizada no interior de uma função sem que haja uma declaração. Por exemplo:

```
$texto = ‘‘III Sinec’’;

function imprime() {
```

```
  echo ‘‘$texto’’;
}
imprime();
```

A chamada a função imprime() não produzirá nada na saída, pois a variável $texto é de escopo global, e não pode ser referida num escopo local, mesmo que não haja outra com nome igual que cubra a sua visibilidade. Para alcançar o objetivo do script acima, a variável global deve ser declarada. Por exemplo:

```
$texto = ‘‘III Sinec’’;

function imprime() {
  global $texto;
  echo ‘‘$texto’’;
}
imprime();
```

Uma outra maneira de acessar variáveis de escopo global dentro de uma função é utilizando um array pré-definido pelo PHP cujo nome é $GLOBALS. O índice para a variável referida é o próprio nome da variável, sem o caracter $. O exemplo abaixo produz um resultado semelhante ao anterior.

```
$texto = ‘‘III Sinec’’;

function imprime() {
  echo ‘‘$GLOBALS[texto]’’;
}
imprime();
```

9 Variáveis e Constantes

9.1 Declaração de uma variável

Como a verificação de tipos em PHP é dinâmica, as variáveis não precisam ser declaradas. Uma variável é iniciada no momento em que é feita a primeira atribuição. O tipo da variável será definido de acordo com o valor atribuído.

9.2 O modificador *static*

Uma variável estática é visível num escopo local, mas ela é inicializada apenas uma vez e seu valor não é perdido quando a execução do script deixa esse escopo. Por exemplo:

```
function imprime() {
  $a = 0;
  echo $a;
  $a++;
}
```

Como a variável $a será destruída no final da execução da função, é inútil tentar incrementá-la. Com o modificador `static` a variável $a terá o seu valor impresso e será incrementada. Veja o exemplo abaixo:

```
function imprime() {
  static $a = 0;
  echo $a;
  $a++;
}
```

O modificador `static` é muito utilizado em funções recursivas, já que o valor de algumas variáveis precisam ser mantidos. Esses valores são recuperados quando é encontrado novamente a declaração `static`.

9.3 Variáveis variáveis

As vezes é conveniente ter nomes de variáveis que são variáveis. Que é um nome de uma variável que pode ser mudado e usado dinamicamente. O PHP permite essa característica e sua utilização é feita através do duplo cifrão $$.

```
$a = "III";
$$a = "Sinec";
echo "$a ${$a}";
```

A saída do script acima será: “III Sinec”.

9.4 Variáveis enviadas pelo navegador

Uma das maneiras do navegador enviar dados para o interpretador PHP é através da URL. Por exemplo: se o script está localizado em "http://localhost/teste.php" pode-se chamá-lo com a url "'http://localhost/teste.p
Neste caso o PHP criará automaticamente uma variável com o nome $texto contendo a string "IIISinec". Este formato é conhecido como `urlencode`. Os formulários HTML já enviam informações automaticamente nesse formato, e o PHP trata esses dados como variáveis comuns, sem precisar de um tratamento especial por parte do programador.

O formato `urlencode` é obtido substituindo os espaços pelo caracter "+" e todos os outros caracteres não alfa-numéricos (com exeção de "_") pelo caracter "%" seguido do código ASCII em hexadecimal.

9.5 Utilizando arrays

Cada elemento de um formulário HTML submetido a um script cria no ambiente do mesmo uma variável cujo nome é o mesmo nome do elemento. Por exemplo

```
<input type=''text'' name=''email''>
```

O código acima fará com que seja criada uma variável com o nome $email no script ao qual esse código foi submetido.

Uma boa técnica de programação é utilizar a notação de arrays para itens de um formulário HTML. Para um conjunto de checkboxes, por exemplo, pode-se utilizar a seguinte notação:

```
<input type=''checkbox'' name=''teste[]'' value=''valor1''>Opção 1
<input type=''checkbox'' name=''teste[]'' value=''valor2''>Opção 2
<input type=''checkbox'' name=''teste[]'' value=''valor3''>Opção 3
```

Ao submeter o formulário, o script que recebe os valores criará uma variável chama $teste contendo os valores num array, com índices a partir do zero.

9.6 Variaváveis de ambiente

O PHP possui diversas variáveis de ambiente, como a $PHP_SELF, por exemplo, que contém o nome e o path do próprio arquivo. Algumas outras contém informações sobre o navegador do usuário, o servidor http, a versão do PHP e diversas outras informações. Para ver a listagem completa dessas variáveis e seus respectivos conteúdos, deve-se utilizar a função `phpinfo()`.

9.7 Verificando se uma variável possui um valor

Existem duas funções que podem ser usadas para saber se a variável está setada. São elas a função `isset` e a função `empty`.

A função `isset` possui o seguinte protótipo:

```
int isset(mixed var);
```

Ela retorna `true` se a variável estiver setada (mesmo com uma string vazia ou o valor zero), e `false` caso contrário.

A função `empty` possui o seguinte protótipo:

```
int empty(mixed var);
```

Ela retorna `true` se a variável não contiver um valor ou possuir valor zero ou uma string vazia. Caso contrário, retorna `false`.

9.8 Constantes pré-definidas

Assim como as variáveis de ambiente, o PHP possui algumas constantes pré-definidas, contendo a versão do PHP, o sistema operacional do servidor, etc. Para uma listagem completa dessas constantes pode-se usar a função `phpinfo()`.

9.9 Definindo constantes

A função `define` é usada para definir o valor de uma constante, que uma vez setada, não poderá mais ser alterado. Uma constante pode apenas conter um valor escalar, não podendo guardar arrays ou objetos. A assinatura da função `define` é a seguinte:

```
int define(string nome_da_constante, mixed valor);
```

A função retorna `true` se for bem sucedida. Por exemplo:

```
define (``pi'', 3.1415926536);
$circunf = 2*pi*$raio;
```

10 Utilizando cookies

10.1 O que são?

Cookies são variáveis gravadas no cliente (browser) por um determinado site. Somente o site que gravou o cookie pode ler a informação contida nele. Este recurso é muito útil para que determinadas informações sejam fornecidas pelo usuário apenas uma vez. Exemplos de utilização de cookies são sites que informam a quantidades de vezes que você já visitou, ou alguma informação fornecida numa visita anterior.

Existem cookies persistentes e cookies de sessão. Os persistentes são aqueles gravados em arquivo, e que permanecem após o browser ser fechado, e possuem data e hora de expiração. Os cookies de sessão não são armazenados em disco e permanecem ativos apenas enquanto a sessão do browser não for encerrada.

Por definição exostem algumas limitações para o uso de cookies, listadas a seguir:

- 300 cookies no total;
- 4 kilobytes por cookie;
- 20 cookies por servidor ou domínio.

10.2 Gravando cookies

Para gravar cookies no cliente, deve ser utilizada a função setcookie, que possui a seguinte assinatura:

```
int setcookie(string nome, string valor, int exp, string path, string domínio, int secure);
```

Onde o nome é o nome da variável, o valor é o conteúdo da variável, exp é opcional e indica a data de expiração no formato Unix. Se não for definida o cookie será de sessão; path é o path do script que gravou o cookie; o domínio é o domínio responsável pelo cookie; e secure se tiver valor 1 indica que o cookie só pode ser transferido por uma conexão segura (https).

Um cookie não pode ser recuperado na mesma página que a gravou, a menos que ela seja recarregada pelo browser.

Cookies só podem ser gravados antes do envio de qualquer informação para o cliente. Portanto todas as chamadas a função setcookie devem ser feitas antes do envio de qualquer header ou texto.

10.3 Lendo cookies gravados

Os cookies lidos por um script PHP ficam armazenados em duas variáveis. No array `$HTTP_COOKIE_VARS[]` , tendo como índice a string do nome do cookie, e numa variável cujo nome é o mesmo do cookie, precedido pelo símbolo $. Por exemplo:

```
Um cookie foi gravado numa página anterior
pelo seguinte comando:

setcookie(‘‘teste’’, ‘‘III Sinec’’);
Pode ser lida das seguintes formas:
```

```
$HTTP_COOKIE_VARS[teste];
```

ou

```
$teste;
```

11 A História do *PostgreSQL*

Tudo começou com o desenvolvimento do Ingres, ancestral do *PostgreSQL*, na Universidade da Califórnia de Berkeley, entre os anos de 1977 a 1985. Após isso, o código do Ingres foi aumentado pela *Ingres Corporation*, o qual produziu um dos principais servidores de banco de dados relacionais que tiveram sucesso comercialmente.

Também em Berkeley, Michael Stonebraker liderou uma equipe que desenvolveu um servidor de banco de dados relacional chamado *Postgres*, entre os anos de 1986 a 1994. A Illustra(comprada depois pela Informix) herdou o código *Postgres* e desenvolveu-o em um produto comercial.

Dois estudantes de graduação, Jolly Chen e Andrew Yu, adicionaram mais outras características, de forma que interpretasse SQL no Postgres. O projeto resultante foi chamado *Postgres95*, desenvolvido no período de 1994 a 1995.

Após a saída dos desenvolvedores de Berkeley, o projeto continuou com a inclusão de novos desenvolvedores até o lançamento de uma nova versão, no final de 1996 com o nome de *PostgreSQL*.

12 Software Open Source

PostgreSQL é um software open source. O termo software open source geralmente confunde as pessoas. Com um software comercial, uma companhia contrata programadores, desenvolve um produto, e vende aos usuários.
Com a internet, entretanto, novas possibilidades existem. Software open source não tem companhia. Em vez, programadores capazes com interesses e algum tempo livre se unem via internet e trocam idéias. Alguém escreve um programa e coloca o mesmo em um lugar que todos possam ver. Outros programadores olham e fazem as mudanças necessárias para melhorar o software.
Quando o programa se encontra em fase funcional, os desenvolvedores lançam o programa a outros usuários da internet.
Usuários acham erros e características que não existem e enviam background aos desenvolvedores, os quais, por sua vez, fazem as mudanças necessárias.
Para um ciclo que parece não ser funcional, ele possui algumas vantagens sobre o modo antigo de desenvolvimento.

- Uma estrutura de empresa não é requerida, dessa forma não existe overhead (sobrecarga) e não existem restrições econômicas.

- O desenvolvimento de programas não fica limitado aos programadores da empresa, usando a capacidade e a experiência de um largo número de programadores.

- O retorno do usuário é facilitado, permitindo os testes de programa por um número grande de usuários em pouco tempo.

- O encarecimento do programa pode ser distribuído rapidamente aos usuários.

13 Stored Procedures

Uma stored procedure é, em suma, um programa com comandos SQL, que é armazenado em um banco de dados *PostgreSQL*. A stored procedure é composta por comandos SQL, variáveis e comandos de fluxo lógico.

14 Trigger

São blocos de comandos em SQL que são automaticamente executados quando um comando INSERT, DELETE ou UPDATE é executado em uma tabela. A principal aplicação de um trigger é a criação de consistências e restrições de acesso ao banco de dados, como por exemplo, rotinas de segurança. Em vez de deixar o controle da aplicação para a própria tabela, por meio de triggers o banco passa a executar esses controles, tornando muito mais seguro o manuseio do banco de dados.

14.1 Aplicações

- Criar o conteúdo de uma coluna derivada de outras colunas;
- Criar mecanismos de validação que envolvam pesquisas em múltiplas tabelas;
- Criar logs para registrar a utilização de uma tabela;
- Atualizar outras tabelas em função da inclusão ou alteração da tabela atual;

15 Alguns comandos no banco de dados

15.1 Começando uma sessão de banco de dados

PostgreSQL usa um modelo de cliente/servidor de comunicação. Um servidor *PostgreSQL* sempre fica rodando, esperando conexões. O servidor processa a requisição e retorna a saída ao cliente.

15.2 Escolhendo qual interface

PostgreSQL possui várias interfaces de comunicação com o servidor. A que será usada no curso será o PSQL, por ser a mais utilizada e por ter a maior documentação.

15.3 Começando uma sessão

Para iniciar uma sessão PSQL, devemos digitar PSQL no prompt de comando. Ao iniciar um menu em modo texto aparecerá, indicando alguns comandos básicos.

- \ h –> help para comandos SQL;
- \ q –> para sair;
- \ g –> terminar uma query, equivalente;
- \ ? –> mostra todos os comandos do prompt PSQL;
- \ p –> mostra o buffer;
- \ r –> reseta o buffer;

16 Comandos básicos em SQL

Antes de criarmos aplicações exemplo no curso, daremos alguns comandos básicos que são os mais utilizados em criação, consulta, alteração e deleção em um banco de dados padrão SQL.

Os dados em um banco de dados relacional são guardados em estruturas chamadas tabelas, que podem ser relacionar entre si, surgindo em razão disso o termo banco de dados relacional, derivado de uma linguagem matemática baseada em conjuntos.

16.1 Criando tabelas

O comando para criar uma tabela em SQL é o comando CREATE TABLE. Sua sintaxe será demonstrada através do seguinte exemplo.

```
php=> CREATE TABLE noticia (
php(>               id          serial,
php(>               noticia     varchar(800) NOT NULL,
php(>               datains     date,
php(>               chamada     varchar(300) NOT NULL,
php(>               foto        varchar(300),
php(>               link        varchar(300),
php(>               datapub     date
php(> );
```

Como visto aqui, e na maioria dos comandos SQL, um pouco de conhecimento da língua inglesa é necessário para entender o que o comando faz. O resto da requisição tem um formato específico que é reconhecido pelo servidor de banco de dados. Espaçamento e capitalização são funcionais.

O comando CREATE TABLE segue um formato específico: primeiro, as duas palavras CREATE TABLE; depois o nome da tabela; após isso um parêntese de abertura; uma lista de nomes de colunas e seus respectivos tipos; seguido por um parêntese de fechamento.

O comando \d permite enxergar informação sobre uma tabela ou sobre todas as tabelas existentes no banco de dados corrente.

16.2 Adicionando dados com Insert

Para adicionar registros em uma tabela, é necessário usar o comando INSERT. Assim como o Create, o INSERT possui um formato específico, a ser descrito.

```
INSERT INTO noticia (noticia, datains,chamada,foto,link,datapub)
values (
            'A bezerra morreu na br 101',
```

```
data(06/11/2001),
'A morte da bezerra',
'',
'',
data(07/11/2001));
```

Para usar strings, é necessário usar aspas simples. Aspas duplas não funcionam. Espaçamento e capitalização são opcionais, exceto dentro de aspas simples, onde o texto é considerado literalmente. Para colunas numéricas, não é necessário o uso de aspas simples.

16.3 Visualizando dados com Select

Para reaver dados no banco de dados, o comando utilizado é o SELECT. O comando a seguir mostra todos os campos e todos os registro da tabela noticia.

```
SELECT * FROM noticia;
```

O comando SELECT seleciona registro do banco de dados. O asterisco é usado para indicar que na consulta é pedido todas as colunas da tabela. O FROM noticia, indica que a tabela que queremos consultar é a tabela noticia.

O SELECT tem um grande número de variações. Por exemplo, se quisésemos apenas o campo chamada da tabela noticia, o comando utilizado seria o seguinte `SELECT chamada FROM noticia;` , onde o asterisco foi substiuído apenas pelo campo requerido. Se mais de um for necessário, deve-se usar vírgulas para separar os mesmos.

16.4 Selecionando linhas específicas com a cláusula Where

Na seção 16.3 na página 32, ao utilizarmos o comando SELECT, temos todos os registros como resposta. Mas, de fato, dependendo da situação, existem casos em que queremos apenas um registro ou um conjunto de registros que sejam retornados e não todos os registros da tabela. Nesse caso, devemos usar a cláusula WHERE. Ela deve vir logo após ao nome da tabela, e após o WHERE deve-se colocar um filtro de pesquisa.

```
SELECT * FROM noticia WHERE id>65;
```

O comando produz uma saída onde todos os registros possuem um id maior que 65.

16.5 Removendo dados com Delete

Depois de aprender como incluir dados no banco, é necessário saber como se deleta a informação. A remoção é bem simples, como todos os outros comandos já mostrados. O comando DELETE pode rapidamente apagar um, mais de um ou todos os registros. Vale lembrar que o comando DELETE é usado para apagar registro e não tabelas. Abaixo teremos alguns exemplos.

```
DELETE FROM noticia WHERE id=32;
```

Esse comando deleta o registro da tabela notícia que contenha em seu id o número 32;

```
DELETE FROM noticia;
```

Esse comando apaga **todos** os registros da tabela noticia.

16.6 Modificando dados com Update

Para modificar dados, o comando utilizado é UPDATE. Ele segue um formato semelhante a outros comandos utilizados em SQL. Como acima daremos dois exemplos um pouco distintos.

```
UPDATE noticia set chamada='O diário de Fabita' WHERE id=24;
```

Esse comando atualiza o campo chamada para 'O diário de Fabita' onde o id do registro for igual a 24. Se for necessário atualizar mais de um campo, eles devem ser separados por vírgulas.

```
UPDATE noticia set link='/sinec/index.html'
```

Já essa instrução, atualiza o link de **todos os registros** para `'/sinec/index.html'`. Deve-se tomar cuidado com esse comando para, por um descuído de esquecer de colocar a cláusula WHERE, atualizar todos os registros quando se queria atualizar apenas parte deles.

16.7 Ordenando dados com Sort by

Em uma consulta SQL, registros são dipostos de uma forma indeterminada. Para garantir que os registros serão retornados de uma consulta SELECT em uma ordem específica, é necessário que se use a cláusula ORDER BY no final do comando SELECT.

Se quiser, pode-se usar após o ORDER BY condição, o argumento DESC que inverte a ordem de como foi disposta a saída.

```
SELECT id,noticia,chamada FROM noticia WHERE id>2 ORDER BY noticia;
```

Esse comando seleciona os campos id, noticia e chamada, para os registros que tiverem o id maior que 2 ordernando pelo campo noticia. Como o campo noticia é string, a ordenação é feita por ordem alfabética.

```
SELECT * FROM noticia where foto like 'j%' ORDER BY chamada DESC;
```

Esse comando seleciona todos os campos da tabela noticia, reavendo apenas os registros que tenham foto comecando com j, ordenando a saída pelo campo chamada por ordem alfabética inversa.

16.8 Destruindo tabelas

Para completar esta parte de noções de SQL, devemos mostrar como apagar tabelas em si, não só os registros localizados nela. Esta tarefa é realizada com o comando DROP TABLE. Por exemplo o comando `DROP TABLE noticia` apagaria toda a tabela noticia. Convém dizer que toda a estrutura e os dados existentes na tabela serão apagados.

17 Tipos de Dados

Postgres apresenta imensa variedade de tipos de dados disponíveis para o usuário, estes tipos estão listados na tabela 17. O usuário pode livremente criar novos tipos com o comando **CREATE TYPE** .

Nome do tipo	Apelidos	Descrição
bigint	int8	inteiro de oito byts
bit		bit string
bit varying(n)	varbit(n)	bit string de tamanho variável
boolean	bool	booleano lógico(true/false)
character(n)	char(n)	string de caracteres
character varying(n)	varchar(n)	bit string de tamanho variável
date		data(year, month, day)
double precision	float8	número de ponto flutuante,precisão maior
integer	int,int4	inteiro de quatro *bytes*
interval		uso geral de time stamp
money		monetário estilo americano
numeric(p, s)	decimal(p, s)	número exato com precisão variável
smallint	int2	inteiro de dois *bytes*
real	float4	número de ponto flutuante
serial		inteiro de quatro *bytes* autoincremental
time[sem fuso horário]		time of day
time with time zone		hora do dia, sem fuso horário
timestamp[com fuso horário]		data e hora

Tabela 9: Tipos de dados

17.1 Tipos Numéricos

Os tipos numéricos abrangem os inteiros de dois, quatro e oito *bytes*, números de ponto flutuante e decimais de precisão fixa.

17.1.1 Tipo `Serial`

O tipo `serial` é um tipo especial do *Postgres* construído através de outros componentes existentes. É usado para criar inteiros únicos em um determinado campo.

```
CREATE TABLE tabela (coluna SERIAL);
```

Isso é equivalente à:

```
CREATE SEQUENCE tabela_coluna_seq;
CREATE TABLE tabela
(coluna integer DEFAULT nextval('tabela_coluna_seq'));
CREATE UNIQUE INDEX tabela_coluna_key on tabela(coluna);
```

O tipo `serial` cria uma sequência independente da tabela, isto é, se a tabel for apagada. A sua sequência não será automáticamente removida. Para apagar a sequência deve-se proceder da seguinte maneira:

```
DROP SEQUENCE tabela_coluna_seq;
```

17.1.2 Tipo Monetário

17.2 Tipos caracter

Além dos tipos padrões suportados pelo SQL: `caracter` e `caracter varying`. *Postgres* também reconhece o tipo `text`.

Nome do Tipo	Armazenamento	Compatibilidade	Descrição
character(n),char(n)	(4+n) *bytes*	SQL	Tamanho fixo
character varying(n),varchar(n)	(4+n) bytes	SQL	Tamanho variável com limite
text	(4+n) *bytes*	Mais flexível	tamanho variável e ilimitado

Tabela 10: Tipos Caracter

17.3 Tipo data/hora

Postgres suporta todos os seguintes tipos de datas e horas do SQL.

- `timestamp`: Data e hora
- `timestamp with timezone` : Data e hora com fuso horário
- `interval`: Intervalos de tempo
- `date`: Apenas datas
- `time`: Apenas horas
- `time with time zone`: Apenas horas

Tipo	Armazenamento	Início	Fim	Resolução
timestamp	8 *bytes*	4713 AC	1465001 DC	1 microsegundo/ 14 dígitos
timestamp com fuso horário	8 *bytes*	1903 DC	2037 DC	1 microsegundo/ 14 dígitos
interval	12 *bytes*	-178000000 anos	178000000 anos	1 microsegundo
date	4 *bytes*	4713 DC	32767 DC	1 dia
time	4 *bytes*	00:00:00.00	23:59:59.99	1 microsegundo
time with time zone	4 *bytes*	00:00:00.00+12	23:59:59.99-12	1 microsegundo

Tabela 11: Tipos de Data/hora

Em versões anteriores eram usadas os tipos `datetime` e `timespan`, que são euivalentes à `timestamp` e `interval`, respectivamente. Atualmentesses tipos não são mais usados.

17.3.1 Entrada de Data/hora

As maiorias das entradas de data/hora conhecidos são aceitos no *Postgres*. O comando `SET DateStyle to 'US'` ou `SET DateStyle to 'NonEuropean'` especifica o estilo da data como "mês dia ano" e o comando `SET DateStyle to 'European'` especifica o estilo da data para "dia mês ano".

Nota:A entrada de data e/ou hora precisa ser feita com estas entre aspas simples. como o seguinte:

```
type 'value'
```

17.3.2 Hora sem Fuso Horário

Esse tipo pode ser chamado como `time` ou `time without timezone`. A tabela 17.3.2 mostra as entradas válidas para o `time`.

17.3.3 Hora com Fuso Horário

Este tipo foi definido pelo SQL92, mas sua definição contem algumas deficiências que dificulta o uso deste tipo como padrão. Na tabela 16 é mostrada todas as entradas válidas. Note que este tipo aceita todos as entradas do tipo `time` com o acréscimo de um fuso horário válido.

17.3.4 `interval`

O tipo `interval` pode ser especificado usando a seguinte *syntax*.

Exemplo	Descrição
1/8/2001	Estilo americano; 1 de agosto
8/1/2001	Estilo europeu; 1 de agosto

Tabela 12: Entrada de datas

Mês	Abreviação
Abril	Apr
Agosto	Aug
Dezembro	Dec
Fevereiro	Feb
Janeiro	Jan
Julho	Jul
Junho	Jun
Março	Mar
Novembro	Nov
Outubro	Oct
Setembro	Sep, Sept

Tabela 13: Abreviação dos meses

Dia	Abreviação
Domingo	Sun
Segunda	Mon
Terça	Tue, Tues
Quarta	Wed, Weds
Quinta	Thu, Thur, Thurs
Sexta	Fri
Sábado	Sat

Tabela 14: Abreviação dos dias da semana

Exemplo	Descrição
04:05:06.789	ISO-8601
04:05:06	ISO-8601
04:05	ISO-8601
040506	ISO-8601
04:05 AM	O mesmo que 04:05
04:05 PM	O mesmo que 16:05; entrada de horas devem ser menores que 12
z	o mesmo que 00:00:00
zulu	o mesmo que 00:00:00
allballs	o mesmo que 00:00:00

Tabela 15: Entrada de horas sem fuso horário

```
Quantity Unit [Quantity Unit...] [Direction]
@ Quantity Unit [Direction]
```

Onde: `Quantity` é qualquer número inteiroe `Unit` é `second, minute, hour, day, week, month, year, dacade, century, millenium` ou suas abreviações ou o plural dessas unidades. `Direction` pode ser `ago` ou vazio.

17.3.5 Valores Especiais

As funções `CURRENT_DATE`, `CURRENT_TIME`, `CURRENT_TIMESTAMP` podem ser usadas como entrada de data/hora, assim como constantes especiais listadas na tabela 18.

17.3.6 Saída de data/hora

O saída pode ser formatada para um dos quatro tipos citados na tabela **??**. Usando o comando **SET DateStyle**. O valor *default* é o formato ISO.

A estilo da saída dos tipos `time` e `date` são as partes de hora e data, respectivamente, de acordo com os estilos apresentados.

O estilo da saída do tipo `interval` assemelha-se com a entrada, exceto que unidades como `week` ou `century` são convertidos para anos e dias. No modo ISO a saída se assemelha à:

```
[ Quantity Units [ ... ] ] [ Days ] Hours:Minutes [ ago ]
```

Exemplo	Descrição
04:05:06.789-8	ISO-8601
04:05:06-08:00	ISO-8601
04:05-08:00	ISO-8601
040506-08	ISO-8601

Tabela 16: Entrada de horas com fuso horário

Constante	Descrição
current	Hora corrente da transação(obsoleto)
epoch	1970-01-01 00:00:00+00 (hora zero do sistema Unix)
infinity	Maior que qualquer hora válida
-infinity	Menor que qualquer hora válida
invalid	Entrada ilegal
now	Hora corrente da transação
today	Meia-noite de hoje
tomorrow	meia-noite de amanhã
yesterday	meia-noite de ontem

Tabela 17: Constantes especias de data/hora

Especificação de estilo	Descrição	Exemplo
'ISO'	ISO-8601 standard (padrão)	1997-12-17 07:37:16-08
'SQL'	estilo tradicional	12/17/1997 07:37:16 1997 PST
'Postgres'	estilo original	Wed Dec 17 07:37:16 1997 PST
'German'	estilo regional	17.12.1997 07:37:16.00 PST

Tabela 18: Estilos de saída de data/hora

17.4 Tipo Booleano

O *Postgres* oferece suporte para o tipo `boolean`. O tipo `boolean` pode assumir um dos dois valores válidos: `true` (verdadeiro) ou `false` (falso).

Valores válidos para o estado verdadeiro:

- `TRUE`
- 't'
- 'y'
- 'yes'
- '1'

E para o estado falso:

- `FALSE`
- 'f'
- 'n'
- 'no'
- '0'

18 Alguns comandos adicionais em SQL

Nesta seção, mostraremos alguns recursos adicionais existente na linguagem SQL, e além disso, algumas características existentes apenas no PostgreSQL.

18.1 Aspas dentro de um texto

Quando temos que utilizar aspas dentro de um texto, como por exemplo, na inserção de um campo do tipo char, onde o dado é `'Leca's'`, a inserção não funcionaria. A presença de uma aspa simples dentro de uma string cercada por aspas simples, geraria um parser error. O método correto para incluir esse tipo de dado seria o uso de duas aspas simples juntas como em `'Lecas''s'`. Outro método possível seria o uso da contrabarra para incluir esse dado `'Leca\'s'`. O uso da contrabarra ignora o sentido da aspa simples.

18.2 Usando valores nulos

Suponha que se queira incluir uma notícia mas que ela não contenha um link. A pergunta que fica é qual o valor assinalado ao campo link ? A reposta é que o campo link possui um valor nulo. `NULL` é um valor especial que é valido em qualquer coluna. Usa-se quando uma entrada válida para um campo não é conhecida ou não é aplicável.

Os testes `IS NULL` e `IS NOT NULL` foram desenvolvidos especificamente para testar a existência de valores `NULL`.
Abaixo, colocaremos um exemplo que ilustra bem a confusão que pode existir com o uso de valores `NULL`.

```
INSERT INTO noticia (noticia,datains,chamada,link,foto,datapub)
values (
            'Felipe viajou para o Marrocos e acabou
             comprando um Dromedário',
            '08/11/2001',
            'O dromedário de Felipe',
            NULL,
            NULL,
            '10/11/2001'
);
```

Neste exemplo, tanto o campo `LINK` como o campo `FOTO` possuem um valor nulo. Podia-se pensar que fazendo uma consulta do tipo:

```
SELECT * FROM noticia WHERE link=noticia;
```

teríamos como resposta a inclusão feita no exemplo anterior. O fato é que o retorno de query é igual a zero registros. O campo que contém `NULL`, significa desconhecido, desse modo não temos como saber se dois valores `NULL` são iguais. PostgreSQL não adivinha ou imprime o resultado. Convém dizer que " e `NULL` são diferentes.

18.3 O uso de valores DEFAULT

Como mostrado na seção 18.2, colunas não especificadas no comando INSERT possuem valor nulo. Você pode mudar isso usando a palavra chave DEFAULT. Quando na criação de uma tabela, a palavra DEFAULT e um valor podem ser usados logo após o tipo da coluna. Este valor será usado toda vez que não for colocado nenhum dado na coluna com o valor DEFAULT na inserção do registro.

```
CREATE TABLE ultima (
       id serial,
       noticia   varchar(800)
DEFAULT 'Seminário de Informática e Engenharia de Computação',
       hora      time,
       chamada   varchar(300)
DEFAULT 'SINEC',
       link      varchar(300),
       data      date
);
```

Quando for incluído um registro sem notícia e sem chamada, o valor default acima descrito será colocado nos campos noticia e chamada.

18.4 Comentários

Os comentários podem ser usados em qualquer lugar no prompt psql. Dois estilos de comentário são possíveis. A presença de dois traços (–) marca todo o texto da linha como um comentário. O estilo da programação C++ também é aceito, onde o comentário começa com /* e só termina com o */.

18.5 Uso de OR e AND

O uso de OR e AND possibilita a realização de consultas mais poderosas. Suponhamos que queremos todos os registros da tabela ultima, onde a data seja '08/11/2001' e a hora seja '16:00'. O comando utilizado assim seria:

```
SELECT * FROM ultima WHERE data='08/11/2001' AND hora='16:00';
```

De forma semelhante o OR pode ser usado em querys.

18.6 Alcance de Valores

Para reaver registros que tenham datas entre os dias '02/11/2001' e '07/11/2001'. Devemos usar os operadores `> => < <=` para determinar um alcance de valores.

```
SELECT * FROM ultima WHERE data=>'02/11/2001' AND data<='07/11/2001';
```

A tabela 19 na página 41, mostra os operadores existentes no *PostgreSQL*.

Comparador	Operador
menor que	<
menor ou igual	<=
igual	=
maior ou igual	>=
maior que	>
diferente	<> ou !=

Tabela 19: Operadores de Comparação

18.7 Comparação Like

Maior que e menor que são combinações possíveis usando-se os operadores descritos na tabela 19. Mas, comparações mais complexas podem ser feitas. Por exemplo, usuários geralmente precisam comparar strings para saber se as mesmas combinam com um certo modelo.
Algumas vezes é necessário reaver apenas strings que comecem com uma certa letra ou uma certa palavra. A palavra chave `LIKE` possibilita tais comparações.A consulta abaixo retorna todos as notícias que comecem com A.

```
SELECT * FROM noticia WHERE noticia like 'A%';
```

O símbolo (%) significa que qualquer quantidade de caracteres pode seguir o A.

18.8 Cláusula CASE

Muitas linguagens de programação possuem comandos de decisão, especificando se uma condição for verdadeira faça algo, senão faça outra coisa. Apesar de SQL não ser uma linguagem de programação procedural, ele permite controle condicional sobre dados retornados de uma query. A cláusula WHERE usa comparação para controlar a seleção de registros. O comando CASE permite comparações na saída de uma coluna. Aqui foi criado uma tabela de administradores para demonstrar um exemplo do comando CASE.

```
php=> CREATE TABLE administrador (
php(>               id serial,
php(>               nome varchar(300),
php(>               identidade varchar(15),
php(>               idade integer
php(> );
```

19 Unindo Tabelas

Nesta seção, mostraremos como armazenar dados em mais de uma tabela. Assim como o armazenamento, a seleção de dados em várias tabelas são fundamentais para bancos de dados relacionais. Primeiro examinaremos tabelas e referências à colunas, que são importantes em consultas que usam mais de uma tabela. Depois, mostraremos quais são as vantagens de dividir os dados em múltiplas tabelas.

Aqui, faremos a construção das tabelas que serão responsáveis pelo controle da enquete.

19.1 Tabelas e referências à colunas

Antes de lidar com agrupamentos, é necessário mencionar uma importante característica. Até agora, todos os comandos só envolveram uma tabela. Quando uma *query* envolve mais de uma tabela, os nomes de colunas podem ser tornar confusas ou mesmo ambíguas, quando, por exemplo existe mais de uma tabela usada na query com nomes de colunas iguais. Para evitar essa confusão, existe o que chamamos de nome completamente qualificado da coluna, onde o nome da mesma é antecido pelo nome da tabela a qual faz parte seguida por um ponto. Por exemplo, a coluna `noticia` da tabela `noticia` seria referenciado com `noticia.noticia`.
Outro modo de referenciar o nome da coluna de uma tabela é através da criação de um apelido para a tabela. Vide exemplo.

```
SELECT noticia.noticia,noticia.chamada FROM noticia;
```

Nesse caso, recuperamos todas as noticias e chamadas da tabela noticia. Aqui, não é necessário o uso do nome completamente especificado, já que não existem dúvidas sobre a origem das colunas, pois só existe uma tabela sendo consultada.

```
SELECT n.noticia,n.chamada FROM noticia n;
```

Aqui houve o uso do apelido para a tabela noticia, que passou a ser referenciada como n, pois após seu nome na query a letra que seguiu foi n, dizendo assim ao servidor que houve a criação de um apelido.

19.2 Tabelas Agrupadas

Até os exemplos passados, na criação das tabelas ultima e noticia, não houve a necessidade de criar tabelas relacionadas, pois toda a informação necessária podia existir em uma única tabela. Mas ao fazer a parte de enquete para o portal a situação muda. Em um primeiro impulso, poderíamos pensar que toda a informação poderia ficar armazenada em uma tabela só como se segue na tabela 20 na página 43.

Cometemos erros ao pensar assim. O primeiro deles é assumir que teremos sempre três respostas. Se tivermos menos respostas, a tabela ficará subutilizada, ocupando espaço desnecessário. Se quisermos em algum dia aumentar o número de respostas, será inviável, já que todo o banco de dados foi criado para receber sempre três respostas. Umas de regras de criação de um bom banco de dados, é que não existam campos com o mesmo significado dentro de uma tabela. Nesse exemplo, temos os campos voto* e resposta* que se repetem por três vezes na tabela. Considerando isso, a tabela enquete ficaria como na tabela 21 na página 43.

Tabela Enquete	Campos	Tipos
	id	Serial
	enquete	char(300)
	data	date
	resposta1	char(100)
	voto1	int
	resposta2	char(100)
	voto2	int
	resposta3	char(100)
	voto3	int

Tabela 20: Tabela errada de enquete

Tabela Enquete	Campos	Tipos
	id	serial
	enquete	char(300)
	data	date

Tabela 21: Tabela enquete

Após os campos voto* e resposta* terem sido retirados, a tabela obedece as leis de criação de um bom banco de dados. A pergunta que fica é: Onde ficarão as respostas dessa pergunta ? Logicamente, em uma outra tabela ! Essa tabela terá o nome resposta e será criado de acordo com a tabela 22 na página 43.

Tabela Resposta	Campos	Tipos
	id	serial
	resposta	char(300)
	contador	integer
	idpergunta	integer

Tabela 22: Tabela resposta

Resumindo, as vantages do uso de múltiplas tabelas incluem as seguintes:

- A modificação de dados torna-se mais fácil;

- Os dados são armazenados em um único espaço;

- Economia no armazenamento de dados;

- Procura de dados é mais fácil

A única condição para que dados duplicados não se movam para outra tabela ocorre quando **todas** as seguintes condições estão presentes:

- O tempo para a realização do agrupamento é proibitivo;

- A procura de dados é desnecessária;

- A duplicação de dados exige pouco espaço no armazenamento;

- É muito pouco provável que os dados mudem.

19.3 Criando tabelas agrupadas

Realizada a criação das tabelas enquete e resposta, veremos como podemos realizar o agrupamento das tabelas. Na criação das mesmas todas as colunas estão com letras minúsculas, podendo ter sido feito com letras maiúsculas, mas o PostgreSQL ignora e considera todos os nomes de coluna em letra minúscula. O único modo de fazer com que o nome da tabela ou o nome da coluna seja considerada em caixa alta é na definição da tabela, usar aspas duplas no nome do campo, como em "Nome do usuário", inclusive sendo permitido usar espaços. Não é recomendado essa prática pois cada vez que deve-se referenciar a coluna, devemos colocar as aspas duplas, dificultando a leitura dos comandos. Sem o uso das aspas duplas, nomes de coluna só podem ter letras, números e underscore. Qualquer outro caractere não é permitido. Em seguida, teremos os comandos de criação das tabelas antes desenhadas.

```
php=> CREATE TABLE enquete (
php(>               id         serial,
php(>               enquete    char(300),
php(>               data       date
php(> );
NOTICE:  CREATE TABLE will create implicit
sequence 'enquete_id_seq' for SERIAL column 'enquete.id'
NOTICE:  CREATE TABLE/UNIQUE will
create implicit index 'enquete_id_key' for table 'enquete'
CREATE

php=> CREATE TABLE resposta (
php(>               id          serial,
php(>               resposta    char(200),
php(>               contador    integer,
php(>               id_resposta integer
php(> );
NOTICE:  CREATE TABLE will create implicit
sequence 'resposta_id_seq' for SERIAL column 'resposta.id'
NOTICE:  CREATE TABLE/UNIQUE will
create implicit index 'resposta_id_key' for table 'resposta'
CREATE
```

Como já explanado anteriormente, o tipo serial será explicado mais a frente. Essas duas tabelas tem uma relação de um pra n, pois uma enquete pode ter várias respostas mas uma resposta é de apenas uma enquete. Agora, adicionaremos uma pergunta e colocaremos duas respostas.

```
php=> INSERT INTO enquete (enquete,data) VALUES
```

```
php-> ('Quando será que a greve acabará','07/11/2001');
INSERT 19013 1

php=> INSERT INTO resposta (resposta,contador,id_enquete) VALUES
php-> ('No natal',0,1);
INSERT 19073 1

php=> INSERT INTO resposta (resposta,contador,id_enquete) VALUES
php-> ('Quando Paulo Renato morrer',0,1);
INSERT 19074 1

php=> INSERT INTO resposta (resposta,contador,id_enquete) VALUES
php-> ('Quando a universidade acabar',0,1);
INSERT 19075 1
```

Neste exemplo, o id_enquete na tabela é 1 pois a primeira resposta recebe esse número com o uso do serial.

19.4 Realizando agrupamentos

Quando os dados estão separados através de múltiplas tabelas, recuperar os dados nessas tabelas se torna uma importante questão. Outro exemplo mostra como encontrar a enquete para dada resposta na tabela resposta.

```
php=> SELECT id,id_enquete FROM resposta WHERE resposta='No natal';
 id | id_enquete
----+------------
  1 |          1
(1 row)

php=> SELECT enquete FROM enquete WHERE id=1;
 enquete
---------------------------------
 Quando será que a greve acabará
(1 row)
```

Na primeira consulta, achamos qual o número da enquete a qual pertence a resposta 'No Natal' (o campo id_enquete). O id_enquete foi 1. O campo id_enquete tem o mesmo dado do campo id na tabela enquete. Achado esse valor, consultamos em uma segunda query, qual era a enquete que tinha o id igual a 1.
Nós chamamos essas consultas de agrupamento manual, devido ao fato do usuário ter tomado o resultado da primeira query e ter colocado o resultado na cláusula WHERE na segunda query. Já esta query mostra quais são os elementos necessários para realizar o agrupamento direto.

```
php=> SELECT enquete FROM resposta, enquete
php-> WHERE resposta.id_enquete=enquete.id AND resposta='No natal';
enquete
--------------------------------
 Quando será que a greve acabará
(1 row)
```

- As duas tabelas envolvidas no agrupamento são especificadas na cláusula FROM;
- As duas colunas necessárias para realizar o agrupamento estão especificadas na cláusula WHERE;
- A resposta é testada na cláusula WHERE;
- A pergunta é retornada pela query SELECT.

Internamente, o banco de dados realiza as seguintes tarefas:

- resposta='No natal': Encontra o registro que contém essa resposta;
- resposta.id_enquete=enquete.id: Do registro encontrado, pegue o campo id_enquete. Encontre na tabela enquete o registro que possua esse número;
- enquete: Retorna o nome da enquete procurada.

Ou seja, o banco de dados realiza as mesmas operações antes realizadas por nós manualmente, só que bem mais rápido. Constate que na maioria dos campos utilizados na última consulta, o nome foi completamente qualificado para evitar ambiguidades como seria se tivéssemos colocados só o campo id sem especificar a tabela, já que nas duas tabelas, existe o campo com o mesmo nome (id). A ordem em que aparece as tabelas ou as condições de consulta não são relevantes.

19.5 Escolhendo uma chave de agrupamento

Uma chave de agrupamento é o valor usado para ligar registros entre tabelas. Por exemplo, 1 é a chave da enquete, identificando unicamente o registro. Algumas pessoas questionam se um número de identificação é necessário. A resposta deveria ser usado como chave ? Usar strings como chave não é uma boa idéia por várias razões:

- Letras são mais fáceis de serem digitadas incorretamente;
- Duas respostas com o mesmo conteúdo seriam impossíveis de distiguir uma da outra;
- Se a resposta tiver que ser mudada, todas as referências à aquela resposta devem ser mudadas;
- Agrupamentos com números são mais eficientes do que agrupamentos de caracteres;
- Números requerem menos espaço de armazenamento que strings.

Pensando em uma tabela que tivesse todos os estados do brasil com as suas respectivas siglas. As siglas poderiam *sim* ser usadas como chave, pelos seguintes fatores:

- O código de duas letras é fácil para os usuários digitarem e de fácil memorização;

- Os códigos dos estados são únicos;

- Códigos de estados não mudam;

- Agrupamento usando chave com 2 caracteres não são muito mais lentos que agrupamento numérico;

- A diferença de armazenamento não é muito maior para guardar dois caracteres em vez de números.

Essencialmente, duas escolhas para chaves de agrupamento existem: números de indentificação ou pequenos códigos numéricos.Nenhuma regra universal dita quando você deverá escolher códigos alfanuméricos ou números de identificação. Os estados do Brasil são claramente um exemplo em que é melhor o uso de códigos de caracteres, por que só 27 existem. Os códigos resultantes são pequenos, únicos, e bem conhecidos pela maioria dos usuários. Geralmente é melhor usar códigos a números de identificação quando o campo exige poucos registros.

19.6 Agrupamentos um-para-muitos

Até agora, no agrupamento de duas tabelas, uma linha de uma tabela combinou exatamente com uma linha de outra tabela, realizando o chamado agrupamento um-para-um. Mas se unirmos agora todas as respostas que fazem parte de uma única enquete ? Múltiplas respostas seriam impressas. No agrupamento um-para-muitos, uma linha na tabela enquete seria agrupada a mais de uma linha na tabela respostas. Agora suponha que nenhuma resposta existe para dada enquete. Mesmo que existe uma enquete válida, se não existir nenhuma resposta para essa enquete, nenhuma linha será retornada. Nós podemos chamar esse caso de agrupamento de um-para-nenhum, o chamado *outer join*.

```
php=> SELECT resposta FROM enquete,resposta WHERE enquete.id=1;
resposta
----------------------------------
 No natal
 Quando Paulo Renato morrer
 Quando a universidade acabar
(3 rows)
```

Para a enquete com o código igual a 1, houve o retorno de 3 linhas, indicando claramente o agrupamento um para muitos.

19.7 Chaves primárias e Chaves estrangeiras

Um agrupamento é realizado pela comparação de duas colunas, como em enquete.id e resposta.id_enquete. A enquete.id é chamada de *chave primária* porque ela é o identificador único e primário para a tabela enquete. A resposta.id_enquete é chamada de *chave estrangeira* por que ela armazena dados de uma chave primária de outra tabela.

Esta seção lidou com técnica – a técnica de criar uma disposição organizada usando múltiplas tabelas. Para adquirir esta habilidade é necessário prática. Sempre espere mudar o seu banco de dados várias vezes para a criação de um banco de dados organizado e definitivo. Uma boa disposição pode fazer seu trabalho ficar mais fácil. A má disposição pode tornar seu trabalho em consultas um inferno. Ao começar a criar suas tabelas para uma necessidade real, você será capaz de identificar os bancos de dados bem planejados. Não tenha preguiça ou medo de recomeçar seu banco de dados do zero, pois é um duro trabalho, mas quando feito apropriadamente, as consultas tornam-se mais fáceis de serem realizadas.

20 Numerando linhas

Números únicos e códigos pequenos alfanuméricos permitem referências à linhas específicas em uma tabela. Elas foram bem discutidas na seção 19.5. Por exemplo, a tabela enquete tinha um campo chamado id que identificava unicamente um registro na tabela. A tabela resposta tinha um campo com o mesmo nome responsável pela mesma função. Enquanto caracteres de códigos devem ser inseridos pelo usuário, a numeração de linhas de uma tabela pode ser geradas automaticamente usando-se dois métodos. Esta seção descreve como usar esses dois métodos.

20.1 Números de Identificação de Objetos (OIDs)

A toda e qualquer linha no *PostgreSQL* é atribuído um único número normalmente invisível chamado de *número de identificação de objeto (OID)*. Quando o programa é inicializado com `initdb` [1], um contador é criado e atribuído a ele um valor próximo aos dezessete mil. O contador é usado para descrever unicamente toda linha. Apesar de banco de dados serem criados e destruídos, o contador continua a incrementar. Ele é usado por todos os bancos de dados, por essa razão ele é sempre único. Nenhuma linha em qualquer tabela em qualquer banco de dados terão o mesmo número de identificação de objeto.
Nessa apostila você já viu alguns OIDs. São aqueles números que aparecem depois de cada comando `INSERT`, como em `INSERT 19073 1`. `INSERT` é o comando que foi executado, 19073 é o OID e um é o número de linhas inseridas.
Normalmente, os OIDs só aparecem após cada comando `INSERT`. Entretanto, se o OID é especificado por um comando que não o `INSERT`, ele será mostrado. Apesar de nenhuma coluna OID ter sido mencionada no comando `CREATE TABLE`, toda tabela *PostgreSQL* inclui uma coluna invisível chamada OID. Essa coluna só é mostrada caso seja especificamente acessada. Qualquer query do tipo `SELECT * FROM "alguma tabela"` não mostrará a coluna OID. Entretanto, algo como `SELECT OID,* FROM "alguma tabela"` mostrará o campo. O OID pode ser usado como chave primária ou chave estrangeira em agrupamentos (*joins*).

[1] cria um banco de dados template1 usado por todos os outros banco de dados

```
php=> SELECT OID,* FROM enquete;
  oid  | id | enquete | data
-------+----+---------+------------
 19013 |  1 | Quando será que a greve acabará | 2001-07-11
(1 row)
```

Caso queiramos usar uma tabela que contenha chave estrangeira do tipo OID, temos que na sua criação, em vez de usar o tipo integer ou mesmo char, usar o tipo OID como no exemplo abaixo.

```
php=> CREATE TABLE teste (
php(>               id    integer,
php(>               nome  char(100),
php(>               chave oid
php(> );
CREATE
```

Uma coluna do tipo OID é semelhante a uma coluna do tipo integer, mas definindo-a como um tipo OID documenta que a coluna receberá valores OID. Não confunda uma coluna com o tipo OID com a coluna chamada OID. Toda linha tem uma coluna chamada OID, a qual normalmente é invisível. Uma linha pode ter zero, uma ou mais colunas com o tipo OID. Uma coluna do tipo OID não é automaticamente atribuída valor especial a ela. Somente a coluna OID é atribuída um valor especial.

20.2 Limitações dos Números de Identificação de Objetos

Esta seção cobre as três limitacões dos OIDs.

20.2.1 Numeração não sequencial

A natureza global de um OID significa que a maioria dos OIDs de uma tabela não são sequenciais. Por exemplo, se for inserida uma resposta hoje e um outra só amanhã, as duas respostas não terão números sequenciais. De fato, os seus OIDs podem ser diferentes em milhares por que qualquer comando `INSERT` em outros banco de dados em outras tabelas incrementariam o contador de objetos. Se o OID não está visível aos usuários isso não é um problema. A numeração não sequencial não interfere o processamento de query's. Entretanto, se os usuários podem ver esses números, parece estranho os grandes buracos entre esses números.

20.2.2 Não modificável

Um OID é atribuído para cada linha no comando `INSERT`. `UPDATE` não pode modificar esse campo gerado pelo sistema.

20.2.3 Não é feito o back up por padrão

Durante a cópia de segurança do banco de dados, o OID normalmente não é copiado. Um flag deve ser usado para habilitar essa cópia.

20.3 Sequências

PostgreSQL oferece outro método de numeração de linhas – sequências. Sequências são contadores nomeados criados por usuários. Após sua criação, uma sequência pode ser atribuída para uma tabela como um `DEFAULT` para a coluna. Usando sequências, números únicos podem ser automaticamente gerados durante o comando `INSERT`.
A vantagem das sequências é que elas evitam os buracos como no caso dos OIDs [2]. Sequências são ideais para o uso de um número de identificação visível pelo usuário. Se uma resposta é criada hoje e outra amanhã, as duas linhas terão números sequenciais porque nenhuma outra tabela reparte o contador [3].

20.4 Criando Sequências

Sequências não são criadas automaticamente, como os OIDs. Ao invés, é necessário usar o comando `CREATE SEQUENCE`. Três funções controlam o contador, como mostrado na tabela 23.

Função	**Ação**
nextval('nome')	Retorna o próximo valor da sequência e atualiza o contador
curval('nome')	Retorna o valor corrente da sequência
setval('nome','valor')	Muda o valor do contador para o especificado no segundo argumento

Tabela 23: Funções de Acesso aos contadores

Abaixo, alguns exemplos na criação de sequências e do uso das funções acima descritas.

```
php=> CREATE SEQUENCE ph_teste;
CREATE

php=> SELECT nextval('ph_teste');
 nextval
---------
       1
(1 row)

php=> SELECT currval('ph_teste');
 currval
---------
       1
(1 row)

php=> SELECT setval('ph_teste','20');
 setval
--------
     20
(1 row)
```

[2] Os buracos podem ocorrer quando uma transação é abortada
[3] Tabelas diferentes podem usar o mesmo contador

20.5 Usando Sequências para numerar linhas

Configurar uma sequência para identificar unicamente as linhas envolvem vários passos:

1. Criar a sequência;
2. Criar a tabela, definindo *nextval()* como sendo o valor padrão para a coluna;
3. Durante o comando `INSERT`, não incluir um valor na coluna em questão ou usar o valor *nextval()*.

```
php=> CREATE SEQUENCE aluno_id;
CREATE
php=> CREATE TABLE aluno (
php(>               id      INTEGER     DEFAULT nextval('aluno_id'),
php(>               nome    CHAR(300),
php(>               tel     CHAR(15)
php(> );
CREATE

php=> INSERT INTO aluno VALUES (nextval('aluno_id'),'Osama','666666');
INSERT 19182 1

php=> INSERT INTO aluno (nome,tel) VALUES ('Saddam','666');
INSERT 19183 1
```

Como podemos ver, o primeiro comando cria a sequência chamada *aluno_id*. O segundo comando cria a tabela aluno, e define *nextval('aluno_id')* como sendo o valor padrão para essa coluna. Os outros comandos mostram como deve ser realizada a inserção de dois modos diferentes mas, corretos.

20.6 Tipo Serial

Outro método de se usar sequências existe, sendo bem mais fácil o uso. Se você definir uma coluna do tipo `SERIAL`, uma sequência será automaticamente criada com o nome de *nomedatabela_nomedocampo_seq*, e um valor *nextval('nomedatabela_nomedocampos_seq')* será definido como padrão.
Não se preocupe com os avisos que na criação da tabela com o tipo serial irão ocorrer. O primeiro indica que o banco de dados está criando implicitamente a sequência para a coluna do tipo serial, e o segundo aviso indicando que um índice está sendo criado, assunto que será dado `EM ALGUM LUGAR NA FRENTE`.

```
php=> CREATE TABLE professor (
php(>               id        serial,
php(>               nome      char(300),
php(>               tel       char(15)
php(> );
NOTICE:  CREATE TABLE will create implicit sequence
'professor_id_seq' for SERIAL column 'professor.id'
```

```
NOTICE:  CREATE TABLE/UNIQUE will create implicit
index 'professor_id_key' for table 'professor'
CREATE
```

Algumas pessoas imaginam porque OIDs e sequências são necessárias. Por que um usuário do banco de dados não pode encontrar o maior número em uso, adicionar um, e usar o resultado para numerar a próxima linha a ser adicionada ? Realmente, OIDs e sequências são preferidas devidos a vários fatos:

- Performance;

- Concorrência;

- Padronização.

Primeiramente, usualmente é um processo demorado procurar todos os números que estão sendo usados correntemente para encontrar o próximo número possível. Referir-se a um contador localizado separadamente é mais rápido. Segundo, se um usuário achar o número mais alto, e um outro usuário está procurando pelo número mais alto ao mesmo tempo, os dois usuários podem achar o mesmo número. É claro, neste exemplo, o número deixaria de ser único. Tal problema de concorrência não ocorre com OIDs e sequências. Terceiro, é mais confiável usar um número dado pelo bando de dados do que um número baseado em uma consulta manual.

21 Performance

21.1 Índices

Quando acessando uma tabela, *PostgreSQL* normalmente lê do começo para o final da tabela, procurando por registros relevantes. Com um índice, é possível achar rapidamente valores específicos no índice, podendo assim ir diretamente para os registros procurados. Desse modo, índices permitem uma procura rápida de linhas específicas em uma tabela.
Por exemplo, considere a query `SELECT * FROM aluno where id=145;` . Sem um índice, *PostgreSQL* deve procurar em toda a tabela procurando por linhas que tenham 145 no campo 145. Com um índice em id, o banco de dados pode ir direto para as linhas onde o id é 145.
Para tabelas grandes, pode-se levar minutos para checar cada linha. Usando um índice, encontrar a linha desejada leva apenas segundos. *PostgreSQL* não cria índices automaticamente[4]. Em vez disso, usuários devem criar índices em colunas muito utilizadas em cláusulas WHERE. Para criar um índice, usa-se o comando `CREATE INDEX nome`, como mostrado abaixo.

```
php=> CREATE INDEX aluno_nome_idx ON aluno (nome);
CREATE
```

Onde nome_aluno é o nome do índice criado, aluno é a tabela a qual o índice vai se referenciar, e nome é o campo a ser indexado. Apesar de ser liberado o uso de qualquer nome para o índice, um bom uso de criação de nome de índices é como foi mostrado. O nome do índice contém a informação da tabela que ele se faz parte aluno, contém o nome da coluna indexada, nome, e após isso, o underscore idx para indicar que foi criado um índice.

Você pode criar quantos índices desejar. É claro, porém, um índice em uma coluna raramente usada é desperdício de espaço de disco. Também, a performance na verdade pode diminuir caso existam muitos índices, por que qualquer mudança de registros é necessária uma atualização em cada índice.

Também é possível a criação de um índice que indexe mais de uma coluna. A única mudança do comando anterior na criação de um índice simples é a inclusão de mais uma coluna entre os parentêses. Esse tipo de índice é indicado quando indexamos as linhas de uma tabela, mas mesmo assim, temos muitas linhas com o mesmo valor da primeira linha indexada. Nesse caso, começa a ser feito a indexação pela segunda coluna do índice.

Índices podem ser úteis também em agrupamentos. Eles podem aumentar a velocidade na pesquisa quando é usado cláusulas ORDER BY.

21.2 Indíces Únicos

A única diferença para o índice ordinário, como o próprio nome diz, eles previnem valores duplicados de ocorrerem na tabela, na coluna indexada. O comando é semelhante, apenas com o acréscimo da palavra

[4] exceção quando se utiliza o campo do tipo serial

UNIQUE logo após a palavra CREATE. Às vezes, índices únicos são criados apenas para evitar valores duplicados, não por questões de performance.

```
php=> CREATE UNIQUE INDEX aluno_tel_idxuni ON aluno (tel);
CREATE
```

Índices únicos PERMITEM múltiplos valores NULL, entretanto. Índices únicos aumenta a velocidade e proibem a duplicata de valores.

22 Controlando Resultados

Quando uma query `SELECT` é digitada no psql, o comando vai até o servidor de banco de dados, é executado, e o resultado é enviado de volta ao psql para ser mostrado. *PostgreSQL* permite que seja feito um controle sobre quais linhas são retornadas.

22.1 Limites

As cláusulas `LIMIT` e `OFFSET` do comando `SELECT` permitem ao usuário escolhes quais linhas a serem retornadas.

```
php=> SELECT COUNT(id) FROM aluno;
 count
-------
     2
(1 row)
```

A função count conta todos os registros existentes na tabela. Como visto, ela retornou o valor 2, indicando que existem apenas dois registros existem na tabela.

```
php=> SELECT * FROM aluno ORDER BY id LIMIT 2;
 id | nome |       tel
----+------+-----------------
  1 | Osama  | 666666
  2 | Saddam | 666
(2 rows)

php=> SELECT * FROM aluno ORDER BY id LIMIT 2 OFFSET 1;
 id | nome   | tel
----+--------+----
  2 | Saddam | 666
(1 row)
```

Constate que na segunda consulta apenas uma linha foi retornada. Isso ocorreu devido a existência de apenas dois registros. Portanto, quando a saída da consulta foi deslocadas usando `OFFSET`, mesmo com o `LIMIT` indicando duas linhas, apenas uma linha surgiu como resposta.
Verifique que as duas query usam `ORDER BY`. Apesar dessa cláusula não ser necessária, `LIMIT` sem `ORDER BY` retorna linhas randômicas da consulta, o que não é muito útil.

`LIMIT` aumenta a performance reduzindo o número de linhas retornadas ao cliente.

23 Administração de Tabelas

Essa seção mostra como criar tabelas temporárias e como realizar a alteração de tabelas ou de colunas.

23.1 Tabelas Temporárias

Tabelas temporárias são tabelas de curta duração de vida – elas só existem durante a sessão do banco de dados. Quando a sessão de banco de dados termina, suas tabelas temporárias são automaticamente destruídas. O exemplo na tabela 24ilustra esse conceito. Nele, `CREATE TEMPORARY TABLE` cria uma tabela temporária. Na saída do programa psql, a tabela temporária é destruída. Quando for reiniciado o programa psql, verificamos que não existe mais tal tabela.

As tabelas temporárias são visíveis apenas para a sessão onde foi criada, elas permanecem invisíveis para os outros usuários. De fato, vários usuários podem criar tabelas temporárias com o mesmo nome, e cada usuário verá apenas a tabela criada por ele.

Usuário1	**Usuário 2**
CREATE TEMPORARY TABLE temptest (col INTEGER) INSERT INTO temptest VALUES (1) SELECT col FROM temptest *retorna 1*	CREATE TEMPORARY TABLE temptest (col INTEGER) INSERT INTO temptest VALUES (2) SELECT col FROM temptest *retorna 2*

Tabela 24: Duas sessões diferentes

Seu uso é ideal para armazenar dados intermediários usado pela sessão. Por exemplo, suponha que é necessário realizar várias consultas para uma query complexa. Uma estratégia eficiente é realizar a consulta uma vez, e depois guardar o valor em uma tabela temporária.

```
SELECT *
INTO TEMPORARY telfonern
FROM aluno
WHERE tel like '84%'
```

Esse comando criou uma tabela temporária com dados 'herdados' da tabela aluno contendo os registros que tenham 84 no começo do número do telefone.

23.2 Alterações de Tabelas

`ALTER TABLE` permite as seguintes operações:

- Renomear tabelas;
- Renomear colunas;
- Adicionar colunas;
- Adicionar colunas com valores padrão;
- Remover colunas com valores padrão.

Abaixo temos alguns exemplos do uso do comando `ALTER TABLE`.

```
php=> CREATE TABLE altertest (col INTEGER);
CREATE
php=> ALTER TABLE altertest RENAME TO alterdemo;
ALTER
php=> ALTER TABLE alterdemo RENAME COLUMN col to democo;
ALTER
php=> ALTER TABLE alterdemo ADD COLUMN col2 INTEGER;
ALTER
```

24 Chaves e constantes

Essa seção cobrirá a última parte a ser vista no curso. Referências e chaves são os tópicos mais importantes para a criação de um banco de dados relacional, evitando assim a criação de um bando de dados sem utilidade.

24.1 Not Null

A palavra `NOT NULL` evita o aparecimento de valores nulos de aparecer em uma coluna. Essa cláusula deve ser colocada após o tipo da coluna. Após isso, qualquer inserção que tente colocar um valor nulo na coluna descrita, falhará, gerando um erro de execução. Uma boa saída para evitar esse tipo de erro é colocar um valor padrão para a coluna, podendo ser assim a definição.

```
php=> CREATE TABLE notenull (
php(>               id  INTEGER NOT NULL DEFAULT 32);
CREATE
```

24.2 Unique

A cláusula `UNIQUE` evita a existência de valores duplicados de aparecerem na coluna. É implementado pela criação de um índice único na coluna. O exemplo abaixo ilustra o uso da cláusula.

```
php=> CREATE TABLE unico (numero INTEGER UNIQUE);
NOTICE:  CREATE TABLE/UNIQUE will create implicit index
'unico_numero_key' for table 'unico'
CREATE
```

24.3 Chave Primária

A palavra `PRIMARY KEY`, define a coluna que identificará unicamente cada linha, é uma combinação do uso das cláusulas `NOT NULL` e `UNIQUE`. Com esses tipos de restrições, `UNIQUE` evita duplicação enquanto `NOT NULL` evita a inclusão de valores nulos. Abaixo temos a criação de uma chave primária .

```
php=> CREATE TABLE chaveprimaria (col INTEGER PRIMARY KEY);
NOTICE:  CREATE TABLE/PRIMARY KEY will create implicit index
'chaveprimaria_pkey' for table 'chaveprimaria'
CREATE
php=> \d chaveprimaria
     Table "chaveprimaria"
 Attribute |  Type   | Modifier
-----------+---------+----------
 col       | integer | not null
Index: chaveprimaria_pkey
```

Note que um índice é criado automaticamente, e a coluna é definida como sendo não nula. Se uma chave primária tiver mais de uma coluna, a clásula deve ser definida em uma linha separada para formar a chave.

```
php=> CREATE TABLE chaveprimaria (
php(>               chave1        INTEGER,
php(>               chave2        INTEGER,
php(>               PRIMARY KEY(chave1,chave2)
php(> );
NOTICE:  CREATE TABLE/PRIMARY KEY will create implicit index
'chaveprimaria_pkey' for table 'chaveprimaria'
CREATE
```

Uma tabela não pode ter mais que uma chave primária. Chaves primárias têm um valor especial quando usamos relações (chaves estrangeiras).

24.4 Chave Estrangeira, REFERENCES

Chaves estrangeiras são mais complexas que chaves primárias. Chaves primárias fazem uma coluna ser não nula e única. Chaves estrangeiras, por sua vez, armazenam dados baseados em outras tabelas. Elas são assim chamadas por que seus dados não as pertecem, e sim a outra tabela. Para exemplificar iremos escrever as tabelas enquete e resposta com suas chaves e relações.

```
php=> CREATE TABLE enquetef (
php(>               id          SERIAL          PRIMARY KEY,
php(>               enquete     CHAR(300),
php(>               data        DATE
php(> );
NOTICE:  CREATE TABLE will create implicit sequence
'enquetef_id_seq' for SERIAL column 'enquetef.id'
NOTICE:  CREATE TABLE/PRIMARY KEY will create implicit index
'enquetef_pkey' for table 'enquetef'
CREATE
```

O comando acima cria a tabela enquetef, sendo a chave primária o campo com o tipo serial.

```
php=> CREATE TABLE respostaf (
php(>               id          SERIAL          PRIMARY KEY,
php(>               resposta    CHAR(200),
php(>               contador    INTEGER,
php(>               id_enquete INTEGER REFERENCES enquetef
php(> );
NOTICE:  CREATE TABLE will create implicit sequence
'respostaf_id_seq' for SERIAL column 'respostaf.id'
```

```
NOTICE:  CREATE TABLE/PRIMARY KEY will create implicit index
'respostaf_pkey' for table 'respostaf'
NOTICE:  CREATE TABLE will create implicit trigger(s)
for FOREIGN KEY check(s)
CREATE
```

O comando acima cria a tabela respostaf, criando a chave estrangeira referenciando-se a tabela enquetef. Observe que a sintaxe de criação é simples mas o conceito envolvido é complexo. Esperamos com essa apostila clarear algumas dúvidas sobre o SQL em si e o próprio *PostgreSQL*.

25 Php e sua interação com PostgreSQL

Nesta seção veremos como acessar o banco de dados através do Php usando funções de fácil entendimento e clareza. Todos os comandos abaixo são assumidos e testados nas versões PHP 4 ou acima dela.

25.1 Abrir Conexão com o banco de dados

A função usada para abrir uma conexão com o banco de dados *PostgreSQL* é a função `pg_connect`. Vários parâmetros são necessários e alguns são opcionais. A seguir temos várias possibilidade do uso da função.

int	pg_connect	(string host, string port, string dbname)
int	pg_connect	(string host, string port, string options, string dbname)
int	pg_connect	(string host, string port, string options, string tty, string dbname)
int	pg_connect	(string conn_string)

Tabela 25: Alguns exemplos do cabeçalho da função

A seguir, temos o significado de cada um dos argumentos listados na Tabela 25.

host Indica qual máquina deseja-se conectar;

port Indica a porta utilizada na conexão;

dbname Indica o nome do banco de dados a ser aberta a conexão;

tty Indica qual terminal a conexão deve ser realizada;

options Define vários outros argumentos tal como usuário e senha.

```
<?php
$open = pg_Connect ("dbname=php");
//abre conexão com o banco de dados chamado "php"
$open2 = pg_Connect ("host=arrobinha port=5460 dbname=php");
//abre conexão com o banco de dados chamado php na máquina arrobinha pela porta "5460"
$open3 = pg_Connect ("host=newarrobinha port=5460 dbname=php user=sinec password=Ceni$");
//abre conexão com o banco de dados chamado php, na máquina newarrobinha, pela porta "5460"
, com usuário e senha determinados
?>
```

A função `pg_connect` retorna um índice em caso de sucesso, ou falso se a conexão não puder ser aberta. Os argumentos devem ser passados usando-se aspas duplas.
Se uma segunda chamada é feita pelo `pg_connect` com os mesmo argumentos, nenhuma nova conexão será aberta, mas em vez disso, será retornado apenas o índice da conexão já aberta anteriormente.

25.2 Executar uma consulta

A função usada para realizar uma consulta é a função `pg_exec`. Dois parâmetros são necessários para a conexão, a variável que guarda o índice da conexão e uma string, que no caso é a query a ser executada.

int pg_exec (int connection, string query)

Tabela 26: Cabeçalho da função `pg_exec`

A seguir, temos o significado de cada um dos argumentos listados na Tabela 26.

connection Indica qual índice de conexão previamente aberto deseja-se conectar;

query Indica a porta utilizada na conexão.

Abaixo, temos exemplos do uso da função `pg_exec`.

```
<?php
  $open=pg_connect("user=sinec,dbname=php,host=localhost") or die("Impossível abrir conexão");
  $consulta="select chamada,noticia from noticia";
  $exec=pg_exec($open,$consulta) or die("Impossível executar consulta");
?>
```

Usamos primeiramente a função `pg_connect` para abrir uma conexão com o banco de dados, e nomeamos o seu índice com a variável `open`. Após isso, usamos uma variável para guardar toda a string da consulta, prática indicada para facilitar o entendimento no uso das funções, não sendo preciso digitar todo o comando SQL dentro da função responsável pela execução da query, a função `pg_exec`, que recebe apenas duas variáveis, a primeira sendo o identificador da abertura da conexão e a segunda sendo a própria query a ser executada.
Retorna como resultado um índice se a query for executada, e falso se a conexão falhar ou o índice da conexão não for válido. Detalhes sobre erros podem ser recuperados usando a função `pg_ErrorMessage()` se a conexão for válida. O valor de retorno da função `pg_exec` é um índice que pode ser usado para acessar resultados da query por outras funções *PostgreSQL*.

25.3 Tratamento dos dados enviados na consulta

Tendo aprendido como abrir e executar comandos SQL por uma página PHP, temos que saber como tratar esses dados vindos do banco de dados, para poder utilizá-los de forma eficiente. Existem várias funções que podem tratar os dados, mas aqui por questão de simplicidade veremos apenas uma função, que é suficiente para adquirir os dados de uma forma simples e clara. A função `pg_result` tem como entrada a execução de uma query, a linha desejada e a coluna. Seu cabeçalho é definido na tabela 27.

mixed pg_result (int result_id, int row_number, mixed fieldname)

Tabela 27: Cabeçalho da função `pg_result`

O `pg_result()` retornará valores de um identificador de índice produzido pela função `pg_exec()`. O número da linha e o `fieldname` especificam que célula da tabela será retornada. Números de linha começam do zero. Ao invés de usar o nome do campo, pode-se usar um número como índice, começando do zero.

25.4 Exibir os dados

A exibição dos dados é a saída do programa propriamente dita. É aqui que vamos gerar o documento HTML com os resultados do programa PHP. Para isso, faz-se uso das funções `echo` ou `print` que tem como propósito imprimir dados no documento HTML. Para melhor compreensão, veja o exemplo abaixo:

```
$coneccao = pg_connect("user=sinec dbname=php host=arrobinha");

$SQL = "SELECT * FROM noticia";

$resultado_geral = pg_exec($coneccao, $SQL);

$id1 = pg_result($resultado_geral, 0, '"id"'); /* a variável $id1 receberá o
id da notícia da primeira linha da consulta */

echo "$id1"; /* esta linha de código faz com que seja impresso no documento
HTML o conteúdo da variável $id1 */
```

25.5 Fechar Conexão com o banco de dados

O último passo necessário na interação *PostgreSQL* PHP é o fechamento da conexão. A função utilizada é a função `pg_close`. Ela recebe como argumento um índice de uma conexão válida e retorna uma variável booleana. Retorna falso caso o índice não seja válido ou verdadeiro caso a conexão seja fechada.

bool pg_close (int connection)

Tabela 28: Cabeçalho da função `pg_close`

Nota: O uso dessa função usualmente não é necessária, por que conexões não persistentes [5] são automaticamente fechadas no término da execução do script.
`pg_close` não fecha conexões persistentes geradas pela função `pg_pconnect`.

[5] Aquelas conexões geradas com o uso da função `pg_connect`

Referências

[1] Barreto, Maurício Vivas de Souza, *Aplicações Web com PHP*, 2000.

[2] Página oficial do PHP, *PHP Manual*, www.php.net, 2001.

[3] Castagnetto Jesus, Rawat Harish, Schumann, Scollo Chris, Veliath Deepak, *Professional PHP Programming*, 2001.

[4] Momjian Bruce, *PostgreSQL: Introduction and Concepts* , 2000.