Índice

1 – MySQL

MySQL é um servidor de banco de dados SQL multi-usuário e *multi-threaded*. SQL é a linguagem de banco de dados mais popular no mundo. MySQL é uma implementação cliente-servidor que consiste de um servidor e diferentes programas clientes e bibliotecas.

SQL é uma linguagem padronizada que torna fácil o armazenamento e acesso de informações. Por exemplo, pode-se usar SQL para recuperar informações de produtos e armazenar informações de clientes para um site Web.

O servidor MySQL é também rápido e flexível o suficiente para permitir armazenar *logs* e figuras nele. As principais vantagens do MySQL são velocidade, robustez e facilidade de uso. MySQL foi originalmente desenvolvido pois a equipe da T.c.X. DataKonsultAB (empresa que desenvolveu MySQL) precisava de um servidor SQL que pudesse manipular banco de dados grandes numa ordem de magnitude mais rápida que qualquer banco de dados comercial pudesse lhes oferecer. A equipe da TcX tem usado MySQL desde 1996 em um ambiente com mais de 40 banco de dados contendo 10.000 tabelas, das quais mais de 500 têm mais de 7 milhões de registros. Isto soma aproximadamente 100 Gbytes de dados.

1.1 - Principais Características

As principais características do MySQL são:

- Manipula um número ilimitado de usuários simultâneos;
- Alta velocidade de execução;
- Possui APIs C, C++, Eiffel, Java, Perl, PHP, Python e TCL;
- Trabalha com diferentes plataformas: Unix, Windows etc.;
- Disponibiliza diversos tipos de dados: INT (inteiros sinalizados e não-sinalizados de 1, 2, 3, 4 e 8 *bytes*), FLOAT, DOUBLE, CHAR, VARCHAR, TEXT, BLOB, DATE, TIME, DATETIME, TIMESTAMP, YEAR, SET e ENUM;
- Alta velocidade na execução de *joins* usando *multi-join* otimizado;
- Suporte completo a operadores e funções nas cláusulas SELECT e WHERE;
- Suporte às cláusulas GROUP BY e ORDER BY e a funções de grupo (COUNT(), AVG(), STD(), SUM(), MAX() e MIN());
- Suporte a LEFT OUTER JOIN com a sintaxe ANSI SQL e ODBC;
- Possibilidade de misturar tabelas de diferentes bancos de dados na mesma *query*;

- Sistema de privilégios flexível, simples, eficiente e seguro, que permite verificação baseada em *host*.
- Suporte a ODBC (Open DataBase Connectivity) para Windows95 e suas funções. É possível, por exemplo, usar o Access para conectar ao servidor MySQL;
- Tabelas de disco sob a forma *B-tree* rápidas com compressão de índices;
- Permite 16 índices por tabela;
- Disponibiliza registros de tamanho fixos e variados;
- Manipula grandes bancos de dados com vastos volumes de informações, na ordem de 50.000.000 registros;
- Escrita em C e C++. Testada com diferentes compiladores;
- Possui um sistema de alocação de memória extremamente rápido;
- Suporte total ao conjunto de caracteres ISO-8859-1 Latin1, todos os dados são salvos e ordenados neste formato;
- Permite a definição de *aliases* em colunas e tabelas como no padrão SQL92;
- DELETE, INSERT, REPLACE e UPDATE devolvem o número de linhas afetadas pelo comando;
- Nomes de funções não entram em conflito com nomes de tabelas ou colunas;
- O servidor pode emitir mensagens de erros em diversas linguagens;
- Clientes podem conectar ao servidor MySQL utilizando conexões TCP/IP, Unix *sockets* ou sob o Windows NT.

1.2 - A Estrutura

Um banco de dados nada mais é do que uma hierarquia de estruturas de dados complexas. Em MySQL, como em muitos outros bancos de dados, o conceito da estrutura que mantém os blocos (ou registros) de informações é chamado de tabela. Estes registros, por sua vez, são constituídos de objetos menores que podem ser manipulados pelos usuários, conhecidos por tipos de dados (*datatypes*). Juntos, um ou mais *datatypes*, formam um registro (*record*). Uma hierarquia de banco de dados pode ser considerada como: Banco de dados > Tabela > Registro > Tipo de dados. Os tipos de dados possuem diversas formas e tamanhos, permitindo ao programador criar tabelas específicas de acordo com suas necessidades. MySQL provê um conjunto bem grande de tipos de dados, entre eles:

- CHAR(M): *strings* de tamanho fixo entre 1 e 255 caracteres;
- VARCHAR(M): *strings* de tamanho flexível entre 1 e 255 caracteres. VARCHAR ocupa sempre o menor espaço possível, no entanto é 50% mais lento que o tipo CHAR;

- INT(M) [Unsigned]: números inteiros entre -2147483648 e 2147483647. A opção "unsigned" pode ser usada na declaração mudando o intervalo para 0 e 4294967295 para inteiros não-sinalizados;
- FLOAT [(M,D)]: números decimais com D casas decimais;
- DATE: armazena informação relativa a datas. O formato *default* é 'YYYY-MM-DD' e as datas variam entre '0000-00-00' e '9999-12-31'. MySQL provê um poderoso conjunto de comandos para formatação e manipulação de datas;
- TEXT/BLOB: *strings* entre 255 e 65535 caracteres. A diferença entre TEXT e BLOB é que no primeiro o texto não é sensível ao caso e no segundo sim;
- SET: conjunto de valores *strings*;
- ENUM: conjunto de valores *strings*, difere do SET pois só são armazenados valores previamente especificados.

Além dos tipos de dados existem outras opções a serem usadas em conjunto com os tipos de dados para a criação de tabelas e especificação de colunas:

- *Primary Key* (Chave Primária): usada para diferenciar um registro do outro. Cada registro, desta forma, não pode ter a mesma chave primária.
- Auto_increment: uma coluna com esta opção é automaticamente incrementada quando da inserção de um registro;
- NOT NULL: não permite a inserção de valores nulos.

1.3 - O Sistema de privilégios

Administrar o servidor MySQL, envolve a manutenção do banco de dados com as configurações do servidor (*hosts*, usuários e bancos de dados), ou seja, o sistema de privilégios. O conceito do sistema de privilégios é simples, pela atribuição de um conjunto de privilégios, um usuário em determinado *host* tem permissão para executar comandos sobre uma base de dados. Estes privilégios estabelecem um conjunto de regras no qual o servidor MySQL se baseia, e estas regras podem ser, por exemplo, permissão para inserir, selecionar, excluir informações de uma tabela, ou criar, modificar tabelas etc. Portanto, o sistema de privilégio, se resume em três tabelas principais: *host, user* e *db* tendo como hierarquia, da mais alta para a mais baixa, a ordem apresentada.

- A tabela host determina quais os hosts que estão habilitados a acessar o servidor MySQL. Sua estrutura de colunas é a seguinte: *Host,* Db, Select_priv, Insert_priv, Update_priv, Delete_priv,

Create_priv, Drop_priv, sendo que as duas primeiras colunas estabelecem, de qual *host* o banco de dados pode ser acessado e as colunas termindas em priv são privilégios de acesso especificados com Y ou N (o *default* é N).

- A tabela *user* determina os usuários que podem acessar o servidor e suas senhas de identificação a partir de um *host*. Sua estrutura de colunas é: *Host*, *User*, *Password*, Select_priv, Insert_priv, Update_priv, Delete_priv, Create_priv, Drop_priv, Reload_priv, Shutdown_priv, Process_priv, File_priv;
- A tabela *db* contém as informações relativas a qual banco de dados um usuário de um certo *host* pode acessar: *Host*, Db, *User*, Select_priv, Insert_priv, Update_priv, Delete_priv, Create_priv, Drop_priv.

1.4 - O Ambiente MySQL

As tabelas acima funcionam exatamente como tabelas normais MySQL. Elas podem ser facilmente modificadas usando comandos como INSERT, UPDATE e DELETE.

Para entrar no ambiente MySQL monitor, ou o programa cliente mysql, e acessar o servidor MySQL, o comando a ser executado é o seguinte:

```
shell> mysql –h host –u username –p databasename
```

A opção -h host significa especificar o nome do host, –u username significa especificar o nome do usuário que está acessando, -p solicita um pedido de senha e o databasename é o nome do banco de dados que se deseja acessar. A resposta a este comando é o pedido de identificação do usuário através da senha e em seguida a apresentação e o *prompt* do ambiente a espera de comandos:

```
Enter password: ******
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 459 to server version: 3.22.20a-log
Type 'help' for help.
mysql>
```

A partir do momento que se está conectado ao servidor, é possível realizar vários comandos sobre os bancos de dados que se tem permissão, como selecionar um banco de dados (“use db_name;”), buscar de dados em tabelas através de *queries* (consultas do tipo “select * from uma_tabela;”), inserir valores em uma tabela (“insert into

uma_tabela values(1,2);"), criar bancos de dados ("create uma_bd;"), criar tabelas ("create table teste(id int, descricao varchar(50));"), mostrar tabelas do banco de dados selecionado ("show tables;"), descrever a estrutura de uma tabela ("describe uma_tabela;" ou "show columns from uma_tabela;"), remover tabelas ("drop table teste;") e outros comandos SQL comuns à manipulação e controle de bancos de dados. O comando para desconectar-se do ambiente é "quit".

1.5 - Exemplos

```
[shell]$ mysql -u root

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2 to server version: 3.22.25

Type 'help' for help.

mysql> show databases;
+----------+
| Database |
+----------+
| mysql    |
| test     |
+----------+
2 rows in set (0.00 sec)

mysql> create database Curso;
Query OK, 1 row affected (0.00 sec)

mysql> show databases;
+----------+
| Database |
+----------+
| Curso    |
| mysql    |
| test     |
+----------+
3 rows in set (0.00 sec)

mysql> use Curso;
Database changed

mysql> create table Aluno (
    -> ID int not null auto_increment primary key,
    -> Nome varchar(40),
    -> Email varchar(30),
    -> DataNasc date,
    -> Matricula char(9) );
Query OK, 0 rows affected (0.00 sec)

mysql> show tables;
```

```
+-----------------+
| Tables in Curso |
+-----------------+
| Aluno           |
+-----------------+
1 row in set (0.00 sec)

mysql> desc Aluno;
+-----------+-------------+------+-----+---------+----------------+
| Field     | Type        | Null | Key | Default | Extra          |
+-----------+-------------+------+-----+---------+----------------+
| ID        | int(11)     |      | PRI | 0       | auto_increment |
| Nome      | varchar(40) | YES  |     | NULL    |                |
| Email     | varchar(30) | YES  |     | NULL    |                |
| DataNasc  | date        | YES  |     | NULL    |                |
| Matricula | varchar(9)  | YES  |     | NULL    |                |
+-----------+-------------+------+-----+---------+----------------+
5 rows in set (0.00 sec)

mysql> alter table Aluno drop Email;
Query OK, 0 rows affected (0.00 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> alter table Aluno add Endereco varchar(100) after Nome;
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc Aluno;
+-----------+--------------+------+-----+---------+----------------+
| Field     | Type         | Null | Key | Default | Extra          |
+-----------+--------------+------+-----+---------+----------------+
| ID        | int(11)      |      | PRI | 0       | auto_increment |
| Nome      | varchar(40)  | YES  |     | NULL    |                |
| Endereco  | varchar(100) | YES  |     | NULL    |                |
| DataNasc  | date         | YES  |     | NULL    |                |
| Matricula | varchar(9)   | YES  |     | NULL    |                |
+-----------+--------------+------+-----+---------+----------------+
5 rows in set (0.00 sec)

mysql> insert into Aluno values( NULL, 'Guilherme', 'Rua
Pirineus, 43', '1979-11-18', '9723220-3' );
Query OK, 1 row affected (0.00 sec)

mysql> insert into Aluno values( NULL, 'Fulano', 'Rua Pigmeus,
69', '1980-10-24', '9723299-9' );
Query OK, 1 row affected (0.01 sec)

mysql> select * from Aluno;
+----+-----------+------------------+------------+-----------+
| ID | Nome      | Endereco         | DataNasc   | Matricula |
+----+-----------+------------------+------------+-----------+
|  1 | Guilherme | Rua Pirineus, 43 | 1979-11-18 | 9723220-3 |
|  2 | Fulano    | Rua Pigmeus, 69  | 1980-10-24 | 9723299-9 |
+----+-----------+------------------+------------+-----------+
2 rows in set (0.00 sec)

mysql> select ID, Nome from Aluno where ID < 10;
```

```
+----+-----------+
| ID | Nome      |
+----+-----------+
|  1 | Guilherme |
|  2 | Fulano    |
+----+-----------+
2 rows in set (0.00 sec)

mysql> delete from Aluno where ID=2;
Query OK, 1 row affected (0.00 sec)

mysql> update Aluno set Nome='Guilherme Birckan' where ID=1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from Aluno;
+----+------------------+-----------------+------------+-----------+
| ID | Nome             | Endereco        | DataNasc   | Matricula |
+----+------------------+-----------------+------------+-----------+
|  1 | Guilherme Birckan | Rua Pirineus, 43 | 1979-11-18 | 9723220-3 |
+----+------------------+-----------------+------------+-----------+
1 row in set (0.00 sec)

mysql> grant all privileges on Curso.* to visitante@localhost
identified by ‘senha2000’;
Query OK, 0 rows affected (0.02 sec)

mysql> quit
Bye

[shell]$ mysql -u visitante;

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 3 to server version: 3.22.25

Type 'help' for help.

mysql> use Curso;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+-----------------+
| Tables in Curso |
+-----------------+
| Aluno           |
+-----------------+
1 row in set (0.00 sec)

mysql> select * from Aluno;
+----+------------------+-----------------+------------+-----------+
| ID | Nome             | Endereco        | DataNasc   | Matricula |
+----+------------------+-----------------+------------+-----------+
|  1 | Guilherme Birckan | Rua Pirineus, 43 | 1979-11-18 | 9723220-3 |
+----+------------------+-----------------+------------+-----------+
1 row in set (0.00 sec)
```

```
mysql> drop table Aluno;
Query OK, 0 rows affected (0.00 sec)

mysql> drop database Curso;
Query OK, 0 rows affected (0.00 sec)

mysql> show databases;

+----------+
| Database |
+----------+
| mysql    |
| test     |
+----------+
2 rows in set (0.00 sec)

mysql> quit
Bye
```

2 - PHP

PHP é uma linguagem de *script* no lado do servidor (*server-side*) embutida no HTML, portanto é necessário instalar o interpretador da linguagem no servidor de *Web*. PHP, assim como MySQL, estão disponíveis para *download* para sistemas UNIX, mas para o sistema operacional Windows precisam de uma licença.

PHP é diferente de um *script* CGI escrito em linguagens como Perl ou C pois, ao invés de escrever um programa com muitos comandos para saída em HTML, você escreve um *script* HTML com um código embutido para fazer a mesma coisa. O código PHP é encapsulado em *tags* especiais de início e fim que permitem você alternar para dentro e fora do modo PHP.

O que distingue PHP de algo como um Javascript no lado do cliente é que o código é executado no servidor. Se você tivesse um *script* PHP em seu servidor, o cliente iria receber os resultados da execução deste *script*, e de maneira alguma poderia determinar qual o código que está por baixo desta execução. É possível configurar um servidor *Web* para processar todos os arquivos HTML com código PHP, e então realmente não há maneira de os usuários perceberem que existe código embutido na página HTML.

No nível mais básico, PHP pode fazer qualquer outra coisa que um programa CGI pode fazer, tal como coletar dados de um formulário, gerar conteúdo de páginas dinâmicas, ou enviar e receber *cookies*.

Talvez a maior e mais significante característica em PHP é seu suporte a uma faixa muito ampla de bancos de dados. Escrever uma página Web baseada em um banco de dados é muito simples. Os seguintes bancos de dados são atualmente suportados: Adabas D, Interbase, Solid, Dbase, mSQL, Sybase, Empress, MySQL, Velocis, FilePro, Oracle, Unix dbm, Informix, PostgreSQL.

PHP também tem suporte a comunicação para outros serviços usando protocolos tais como IMAP, SNMP, NNTP, POP3, ou mesmo HTTP. Você pode também abrir *sockets* de rede e interagir usando outros protocolos.

Um exemplo de um *script* dinâmico que imprime a data atual está a seguir:

```
<HTML>
     <HEAD>
          <TITLE>Script de exemplo</TITLE></HEAD>
     <BODY>
          <CENTER>Bem-vindo ao script de exemplo:</CENTER>
          <?php
          /* "<?" acima indica o início do script PHP */
          $hoje = date("Y-m-d");
          print  "<BR><BR>Hoje é: $hoje.";
          # o sinal "?>" seguinte indica o fim do script
```

```
        ?>
    </BODY>
</HTML>
```

Assumindo que hoje é dia 06 de maio de 2000, a saída do *script* acima seria:

```
Bem-vindo ao script de exemplo:

Hoje é: 2000-05-06.
```

Alguns pontos a considerar:

1. Todos os comandos PHP3.0 devem ser envolvidos pelas tags <? e ?>. Uma segunda maneira de denotar comandos PHP é envolvendo-os nas tags <?php e ?>;
2. Todas as sentenças de saída para a tela devem ser envolvidas por aspas (“) e conduzidas pelos comandos print ou echo;
3. Quase todos os comandos PHP3.0 terminam com um ponto-e-vírgula;
4. Todo comando HTML dentro do comando print será executado normalmente pelo *browser* e desempenhará sua função usual;
5. Documentos incluindo código PHP devem ser salvos com a extensão .php ou .php3, isto informará ao interpretador PHP3.0 para executar os comandos encontrados dentro das *tags* <? e ?>. É possível também utilizar extensões diferentes do padrão, mas isso acarretará o uso das *tags* <?php e ?> no sentido de informar ao servidor *Web* que é o interpretador PHP3.0 que se encarregará de executar o *script*, já que isto não pode ser identificado pela extensão do arquivo;
6. A função date apresentada no *script* acima é uma das milhares de funções que o PHP disponibiliza, ela tem o formato: string date (string formato, int timestamp);, ou seja, retorna um *string* e aceita dois parâmetros: o tipo de formato a ser aprensentado e um valor *timestamp*[1] de data opcional (quando omitido, como no nosso caso, considera a data atual).

Uma grande característica de PHP3.0 é a capacidade de construção de *templates* HTML, que são muito úteis quando se está desenvolvendo um *site* com muitas páginas. Isso é possível através do comando include que permite a inserção de código, provindo de um arquivo separado, dentro de um documento HTML. Desta maneira é possível estabelecer, por exemplo, um arquivo de rodapé num arquivo chamado rodape.txt que aparecerá em várias páginas sem precisar reescrever o código, apenas utilizando o comando include, como segue:

[1] *Timestamp* é um formato especial de data, geralmente usado em sistemas UNIX, ele armazena sob a forma de um número inteiro, os segundos, minutos, horas, dia, mês e ano de uma data. Sendo que a cada segundo ele incrementa o seu valor, tornando simples a manipulação de data a partir de operadores como soma, subtração etc.

```
<? include("rodape.txt"); ?>
```

Um outro aspecto importante de PHP é a capacidade de modificar variáveis passadas de formulários HTML, tornando possível a realização de várias tarefas como: envio de um *e-mail* (através da função mail()) baseado em informações de uma página, impressão de páginas personalizadas, passagem e armazenamento de informações em um banco de dados etc.

Existem várias outras características interessantes a destacar sobre PHP, entre elas pode-se citar: Suporte ao modelo de orientação a objetos, Interação com bancos de dados, Criação de imagens GIF, Autenticação HTTP, Manipulação de erros, Manipulação de *cookies*, Suporte para *upload* de arquivos, Conexões persistentes de bancos de dados, Manipulação de arquivos remotos entre muitas outras.

Para se ter uma idéia algumas classes de funções disponíveis no PHP3.0 são listadas a seguir: funções de suporte a bancos de dados, específicas ao Apache (servidor de *Web*), de *array*, matemáticas, calendário, data, diretórios, execução de programas, HTTP, imagem, *filesystem*, *hashes*, Rede, NIS, PDF, Perl, expressões regulares, *strings*, URL, compressão, XML etc.

2.1 - Um breve histórico de PHP

PHP foi concebido num dia do outono de 1994 por Rasmus Lerdof. A primeira versão utilizada ficou disponível no início de 1995 e foi conhecida como Personal Home Page Tools. Ele consistia de um analisador muito simples que entendia somente algumas macros e um número de utilidades que estavam em uso comum nas home pages até então, um livro de visitantes (*Guestbook*), um contador e algumas outras coisas. O analisador foi escrito em meados de 1995 e foi chamado de PHP/FI versão 2. Rasmus combinou os scripts do Personal Home Page Tools com o Form Interpreter e adicionou suporte a mSQL. PHP/FI cresceu e as pessoas começaram a contribuir com o seu código.

É difícil dar estatísticas, mas estima-se que, no fim de 1996, PHP/FI estava em uso em pelo menos 15.000 sites pelo mundo. Na metade de 1997 este número cresceu para mais de 50.000 e nesta época ocorreram mudanças no desenvolvimento do PHP. O analisador foi reescrito por Zeev Suraski e Andi Gutmans e o novo analisador deles formou a base do PHP versão 3.

2.2 - Comentários

Todo programa deve possuir comentários, visando o entendimento do código em consultas posteriores. No PHP, existem três tipos de marcadores de comentário, que são:

// e # para comentário de uma linha. Por exemplo:

```
// atribui o nome à variável
$nome = "Guilherme Birckan";
$email = "birckan@inf.ufsc.br"; # atribui o E-mail à variável
```

e para comentários que ocupem mais de uma linha, usamos os marcadores /* */.

```
/*
Nas linhas abaixo, atribuiremos os valores
Do nome e do e-mail às respectivas variáveis
*/
$nome = "Guilherme Birckan";
$email = "birckan@inf.ufsc.br";
```

2.3 – Variáveis

Para começar, vamos ver como o PHP trata suas variáveis (ou constantes), que podem ser variáveis escalares ou não-escalares. As variáveis escalares são aquelas que podem ser retrabalhadas, ou "divididas em pedaços menores", enquanto as não escalares são as arrays (matrizes) e os objetos.

A identificação de uma variável, independente do seu tipo é pelo sinal $ colocado como primeiro caractere, como abaixo:

```
$nome = "Guilherme Birckan";
$matricula = 97232203;
```

A primeira variável é do tipo string, e a segunda, inteiro (ambas escalares). Vale lembrar que, como a linguagem C, as variáveis $nome e $Nome são consideradas diferentes, pois o PHP as trata como sensíveis ao caso.

2.3.1 – Inteiros e ponto flutuante

As variáveis inteiras são bastante simples de ser usadas, sem nenhuma diferença das demais linguagens que você está habituado a usar. Segue as sintaxes abaixo:

```
$a = 123;
$b = -123;
```

As variáveis em ponto flutuante também são bem simples, lembrando que no lugar da vírgula devemos usar um ponto (“.”):

```
$a = 1.23; // a recebe 1,23
$a = 1.2e3;
```

2.3.2 – Arrays

PHP suporta arrays simples e múltiplas dimensões (também chamadas de matrizes). Usa-se uma variável simples indexada para denotar um array. Esta indexação pode ser feita por números ou mesmo por strings usando colchetes:

```
$a[1] = “abc”;
$a[1] = “def”;
$b[“a”] = 15;
```

Para se adicionar valores no final do array você pode simplesmente usar esta sintaxe:

```
$c[] = “abc”; // $c[0] == “abc”
$c[] = “def”; // $c[1] == “def”
```

Existem funções já implementadas de ordenamento de vetores, tais como: **sort()**

```
$fruits = array ("lemon", "orange", "banana", "apple");
sort ($fruits);
```

Os arrays multidimensionais são usados quase que da mesma forma que os arrays simples:

```
$a[1][2] = $f;
$b[1][“bola”] = $f                  // Você pode misturar índices
$b[“bar”][5][“mesa”][2] = $f;       //array de 4 dimensões
```

Em PHP3 temos um problema de referenciar arrays multidimencionais dentro de strings. O exemplo a seguir não funciona:

```
$a[1][5] = $f;
echo “Isto não vai funcionar: $a[1][5]”;
```

Mas você pode fazer isso usando a concatenação:

```
echo “Agora funciona: “ . $a[1][5];
```

2.3.3 – Strings

	Significado
\n	Nova linha
\t	Tab horizontal
\\	Contra barra
\$	Dollar

Atribuições e concatenações:

```
$str = “Abacate”;
$str = $str . “ grande”;        //concatena “ grande” na string
$str .= “ e madura”;            // concatena “ e madura” na string
```

Pegando um caracter dentro de uma string:

```
$primeiro = $str[0];
$ultimo = $str[ strlen($str) – 1 ];
```

Alguns exemplos de conversão de strings:

```
$a = 1 + "10.5";              // $a é um double (11.5)
$b = 1 + "10 Small Pigs";     // $b é um inteiro (11)
$c = 1 + "10 Little Piggies"; // $c é um inteiro (11)
$d = "10.0 ratos " + 1;       // $d é um inteiro (11)
$e = "10.0 ratos " + 1.0;     // $e é um double (11)
```

2.3.4 – Variáveis de variáveis

Algumas vezes é conveniente você utilizar valores de variáveis como nomes de outras variáveis, utilizando assim variáveis de forma dinâmica. Isto é possível em PHP!

```
$a = “hello”;      // Isto é uma variável simples
$$a = “ world”;    /* Acabamos de criar uma variável $hello com o
                   conteúdo “ world” */
```

Você também pode imprimir estas variáveis de forma dinâmica:

```
echo “$a ${$a}”;
```

Terá como resultado: “hello world”

2.3.5 – Type casting

Type casting em PHP funciona praticamente como em C:

```
$a = 10;              # $a é um inteiro
$b = (double) $a;     # $b é um double
```

(int), (integer)	Converte para inteiro
(real), (double), (float)	Converte para double
(string)	Converte para string
(array)	Converte para array
(object)	Converte para objeto

2.3.6 – Variáveis por referência no PHP4

Na versão 4 do PHP, as variáveis podem receber valor por referência. Isto significa que ao para atribuir o valor a uma variável não usamos um valor, mas um "ponteiro" para o valor em questão. Na verdade, este "ponteiro" é uma outra variável:

```
$nome = "Guilherme Birckan";
$identificacao = &$nome;
```

Deste modo, a variável `$identificacao` recebe o valor de `$nome` e, se uma das duas for atualizada, a outra também será, mantendo o mesmo valor em ambas.

2.4 - Operações Matemáticas

As operações no PHP também seguem o padrão das outras linguagens (+, -, *, /, %[modulo da divisão], sin(), cos()). Além destas, o PHP tem um completo conjunto de operações matemáticas, que podem ser consultadas nesta página:

http://br.php.net/manual/ref.math.php3

Um exemplo para calcular o valor líquido de um preço, depois de aplicar 10% de desconto sobre o preço bruto:

```
$valorbruto = 10;
$desconto = 10 * $valorbruto / 100;
$valorliquido = $valorbruto - $desconto;
```

2.5 - Operadores

Operadores aritméticos:

Exemplo	**Nome**
$a + $b	Adição
$a - $b	Subtração
$a * $b	Multiplicação
$a / $b	Divisão
$a % $b	Modulo da divisão

Operador de atribuição:

Exemplo	**Nome**
$a = $b	Atribuição

```
$a = ($b = 4) + 5; // $b recebe 4 e $a recebe 9
```

Operadores lógicos:

Exemplo	Operador
$a and $b	E
$a && $a	E
$a or $b	OU
$a || $b	OU
$a xor $b	XOR
!$a	NOT

Operadores de comparação:

Exemplo	Nome
$a == $b	Igual
$a === $b	Idêntico
$a != $b	Não igual
$a < $b	Menor que
$a > $b	Maior que
$a <= $b	Menor ou igual
$a >= $b	Maior ou igual

Operadores de execução:

```
$output = `ls –l`;
echo “<pre>$output</pre>”;
```

Operadores de incremento/decremento:

Exemplo	Nome	Efeito
++$a	Pré-incremento	Incrementa $a, depois retorna seu valor
$a++	Pós-incremento	Retorna o valor de $a, depois incrementa
--$a	Pré-decremento	Decrementa $a, depois retorna seu valor
$a--	Pós-decremento	Retorna o valor de $a, depois decrementa

2.6 - Operações com strings

Operações com strings são uma das características mais desenvolvidas do PHP. Para concatenar-se dois strings, usamos o operador "." - Dentre as funções mais importantes estão:

- **strlen()**, que permite saber quantos caracteres possui a string:

```
echo "A palavra 'internet' possui " . strlen("internet") .
" caracteres ";
```

- **substr(),** que devolve uma substring da string informada:

```
echo substr("abcde", 2 , 2); // Esta linha irá exibir os
caracteres "cd";
```

- **ucwords (string),** converte os primeiros caracteres de strings em maiúsculo.

Exemplo:

```
$nome = ucwords("valdir henrique dias leite");
echo($nome); //Esta linha exibirá Valdir Henrique Dias Leite
```

- **strpos ()**, para saber se determinado caractere (ou substring) está contida em uma string:

```
if strpos ($email, "@") {
  echo("Seu e-mail parece estar correto!\n");
} else {
  echo("O e-mail está inválido\n");
}
```

No exemplo acima, verificamos se o caractere "@" está contida em uma variável $email. Se estiver, exibe a primeira mensagem. Do contrário, exibe a segunda.

Outras funções relacionadas à operações com strings podem ser encontradas em

http://br.php.net/manual/ref.strings.html

2.7 - Controlando o fluxo e LOOPS

As funções usadas para controlar o fluxo do programa e execução de "loops" são:

- **if ... else ... else if**, que segue o padrão da linguagem C:

```
if ($sexo == "m") {
     echo "Você é do sexo Masculino\n";
} elseif ($sexo == "f") {
     echo "Você é do sexo Feminino\n";
} else {
     echo "Por favor, informe corretamente seu sexo\n";
}
```

- **switch**, uma maneira de controlar o fluxo onde a variável de controle do fluxo pode ter várias opções de valores. Este tipo de controle poderia ser feito com uma seqüência de "ifs" e "elseifs", mas o uso do switch torna o código mais legível e faz com que seja executado mais rapidamente, pois a verificação da variável "$sexo" só é feita uma vez e depois comparada com as opções de cada "case". Se não estiver em nenhuma delas, é executado o bloco sob o "default". Já com o "elseif", a comparação é feita novamente a cada sentença. Neste exemplo, a diferença não é tão grande, mas quando o tipo de verificação vai ficando mais complexo a velocidade começa a ser sentida. Na maioria dos casos, vale a pena optar pelo switch.

```
switch ($sexo) {
     case "m":
        echo "Você é do sexo Masculino\n";
        break;
     case "f"
        echo "Você é do sexo Feminino\n";
        break;
     case default:
        echo "Por favor, informe corretamente seu sexo\n";
     break;
}
```

Sempre inclua o comando break no final do case. Caso contrário, a execução continuará até encontrar o final do switch (ou a instrução break), fazendo com que as instruções de mais de um case sejam executadas.

- ***while***, que permite repetir o código enquanto uma condição for verdadeira:

```
while ($contador > 0) {
     $contador = $contador - 2;
}
```

- ***for***, para execução de um loop determinada quantidade de vezes:

```
for ($i==0; $i<100; $i++) {
     echo "$i\n";
}
```

2.8 - Tratando formulários

Vamos fazer, passo-a-passo, um script para receber os dados de um formulário, consistir as informações e enviar o resultado por e-mail. Este formulário possui campos para digitação do nome, e-mail e telefone. Todos os campos são obrigatórios e a consistência do campo e-mail deve ser feita apenas verificando a existência do caractere @, para facilitar as coisas. Já o campo telefone deve ter sete ou oito caracteres. Tendo este cenário, mãos a obra!

```
<?php

$erro = "";

if ($nome == "") {
     $erro .= "Digite seu Nome\n"; }

if ((strlen($telefone) > 8) or (strlen($telefone) < 7)) {
     $erro .= "O número do telefone deve ter sete ou oito
caracteres\n";
}

if strpos ($email, "@") = 0 {
     $erro .= "O e-mail digitado não é válido\n"
}
```

Esta primeira parte faz a consistência dos dados e altera o valor da variável $erro, caso alguma das condições não seja satisfeita. Para prosseguir, devemos verificar a ocorrência de erros e então enviar o e-mail se erros não tiverem ocorrido ou enviar uma tela de resposta informando qual o erro aconteceu. Como o valor de $erro antes da verificação dos campos é "", basta testar se a variável ainda tem este valor para saber se aconteceu ou não um erro. Vamos continuar:

```
echo("<html><title>Envie o formulário
abaixo</title><body><center>\n"); # Cabeçalho de resposta.

if ($erro == "") { // Não houve nenhum erro no preenchimento
mail("birckan@inf.ufsc.br", "Dados do Formulário"," Nome:
$nome\n E-mail: $email\n Telefone: $telefone\n","From:
$email\nDate: $date\n" );

echo("Obrigado por enviar este formulário!\n");
} else
echo("Não foi possível enviar o formulário!<br>Verifique as
mensagens abaixo<br><br><b> $erro \n");
}
echo("</center></body></html>\n");
```

Pronto!

A novidades neste script é:

• e-mail. Sua sintaxe é a seguinte: mail(Destinatário, Assunto, Mensagem, Informações_Adicionais);

Depois do script que envia e-mail, vamos fazer um outro que guarde as informações de um formulário HTML em um banco de dados.

2.9 – Funções

As funções no PHP não diferem muito das outras linguagens. Algumas características das funções:

• Devem ser declaradas antes de serem usadas.
• Podem receber parâmetros por valor ou por referência.
• Podem ter quantidade variável de parâmetros (Apenas a partir da versão 4).
• Os parâmetros podem ser declarados com um valor default.
• Uma vez definida, uma função não poderá ser "redefinida".

Alguns exemplos de funções:

```
/*
Esta função retorna TRUE ou FALSE, dependendo da validade ou
não do e-mail informado.
*/
function verifica_email($email){
     if strpos ($email, "@") = 0 {
        return false;
     } else {
        return true;
     }
}

/*
Neste exemplo calculamos o valor líquido, tendo o valor bruto
e o desconto a ser aplicado. Se o desconto não for informado,
utilizaremos 10% como padrão.
*/
function valor_liquido($valor_bruto, $desconto = 10) {
  return ($valor_bruto - ($valor_bruto * $desconto/100));
}
```

Os dois exemplos acima receberam seus parâmetros por valor. Isso significa que as alterações de variáveis realizadas dentro da função só terão efeito no contexto da função, e estas mudanças não refletirão no resto do script. Em alguns casos pode ser interessante que os valores dos parâmetros sejam alterados pela função, e que seus novos valores reflitam no script como um todo. Para conseguir isto, usamos a técnica de passagem de parâmetro por referência. Vamos ver um exemplo:

```
function completaURL(&$mv_URL) {
     $mv_URL .= "http://".$mv_URL;
}

$URL = "www.inf.ufsc.br";
completaURL($URL);

echo "A URL completa fica assim: $URL\n";
```

2.10 – Classes

Como não poderia deixar de ter, PHP também possui suporte a criação de classes e objetos de forma simples:

```
class carro {

	var $estado; // Estado do carro: ligado ou desligado

	function liga() {
		if ($this->estado != “ligado”) {
			$this->estado = “ligado”;
			return true;
		} else {
			return false;
		}
	}
}

$carro1 = new carro;
$carro1->liga();
```

O método construtor da classe (método que é executado quando a classe é criada) é uma função com o mesmo nome da classe:

```
class Pessoa {
	var $idade;
	function Pessoa() {
		$idade = 0; //Todo objeto desta classe é criado com 0 anos
	}
}
```

As classes podem herdar características de outras classes. Criando uma classe derivada de uma outra classe, como no exemplo a seguir da criação da classe Aluno, derivada da classe Pessoa:

```
class Pessoa {
	var $nome;
	var $endereço;
}

class Aluno extends Pessoa {
	var $matricula;
}
```

Todas as características da classe mãe são herdadas pela classe filha. A herança múltipla não é suportada pelo PHP.

2.11 - Acesso à banco de dados

Como foi dito na apresentação do PHP, o acesso à banco de dados é um dos pontos fortes desta linguagem. Ele possui acesso nativo a ADABAS, ORACLE, SYBASE, SQL SERVER, DBASE, INFORMIX, mSQL, MySQL, POSTGRESQL, além de suportar ODBC, fazendo com que o PHP possa trabalhar praticamente com todos os bancos de dados existentes.

Neste módulo vamos ver apenas as apenas as funções relativas ao banco MySQL, pois esta dupla PHP/MySQL está sendo preferida por uma boa parte dos desenvolvedores, particularmente no ambiente Linux/Apache.

O MySQL é um servidor SQL e portanto devemos seguir alguns procedimentos e regras para acesso aos seus dados. Se você está acostumado com o Oracle ou SQL Server não terá dificuldades, mas se você usa somente bancos de dados do tipo Access ou DBF, poderá ter dificuldades em entender o mecanismo usado pelo MySQL.

A primeira regra é ter um banco de dados cadastrado e um usuário com acesso à este banco de dados. Vale lembrar que o MySQL não é um banco de dados, e sim um servidor de dados. Tenha isto em mente para entender o exemplo.

Digamos que temos um banco de dados Curso com o usuário visitante e senha temp99. O primeiro passo é "logar" ao servidor. Para isso usamos a função `mysql_connect` e informamos ao servidor *login* (usuário) e *senha*. Veja abaixo:

```
$conn = mysql_connect ("localhost", "visitante", "temp99");
```

Este comando abrirá uma conexão com o MySQL da máquina local (localhost), usando o usuário visitante cuja senha é temp99. Uma referência a esta conexão será gravada na variável $conn.

Depois de conectados ao servidor, devemos conectar ao banco de dados propriamente dito, usando o comando mysql_select_db, que precisa de dois parâmetros: O nome do banco de dados e a conexão. Caso a conexão não seja informada, ele tentará usar a última criada. Em nossos exemplos, iremos sempre informar os dois parâmetros.

```
$db = mysql_select_db("", $conn);
```

Neste ponto já temos uma conexão com o servidor e já criamos um link com o banco de dados. Agora podemos enviar os comandos SQL que desejarmos. Se você não souber SQL, aprenda :-)

Agora segue nosso exemplo prático: Vamos usar o script do módulo passado e alterá-lo de modo que os dados digitados no formulário sejam gravados no banco de dados Curso antes de enviar o e-mail.

```
<?php

$erro = "";

# Verificar se o campo NOME está vazio.
if ($nome == "") {
      erro .= "Digite seu Nome\n";
}

# Verificar a quantidade de caracteres no campo TELEFONE.
if ((strlen($telefone) > 8) or (strlen($telefone) < 7)) {
$erro .= "O número do telefone deve ter 7 ou 8 caracteres\n";
}

# Testar vamor do campo E-mail, verificando o caracter "@"
if strpos ($email, "@") = 0 {
  $erro .= "O e-mail digitado não é válido\n";
}

# Cabeçalho de resposta.
echo("\n");
echo("<center>\n");

if ($erro == "") {
  $conn = mysql_connect("localhost", "visitante", "temp99");
  $db = mysql_select_db("Curso", $conn);
  $sql = mysql_query("insert into Aluno (Nome, Email, Telefone) values
('".addslashes($nome). "',".addslashes($email)."',
'".addslashes($email)."')" or die ("Não foi possível atualizar a tabela");
      mysql_close($conn);
      mail("birckan@inf.ufsc.br", "Dados do Formulário"," Nome:
$nome\n
E-mail: $email\n Telefone: $telefone\n","From: $email\nDate:
$date\n");
      echo("Obrigado por enviar este formulário!\n");
} else {
      echo("Não foi possível enviar o formulário!
Verifique as mensagens abaixo:

\n");
```

```
     echo("<b>$erro </b>\n");
     echo("<br><br><a href=form.htm>Voltar\n");
}
     echo("</center>");
```

Este é o procedimento padrão para usar servidores de banco de dados com o PHP:

• Conectar ao servidor
• Abrir o banco de dados (um servidor SQL pode ter mais de um banco de dados)
• Enviar os comandos SQL
• Desconectar do servidor

A novidade deste exemplo fica por conta do comando die que finaliza o script caso a função que o precede não possa ser executada.

2.12 - FTP e HTTP

Algumas vezes pode ser útil que nosso script execute um outro script ou então transfira um arquivo para outro servidor. Para isso, podemos executar comandos HTTP e FTP de dentro do PHP. Veja os exemplos:

HTTP: Podemos fazer, dentro do script PHP, uma chamada a outro script ou programa CGI hospedado em outro servidor. Isto é muito útil quando queremos consultar algum dado em um servidor remoto, ou até mesmo para abrir uma página, usando o protocolo HTTP. Para isso, basta chamar a função Header("location: pagina.htm") para redirecionar para uma página específica ou então o usar o comando abaixo para executar um CGI passando parâmetros via URL:

```
header("location: http://server/cgi/script.pl?p=" . $param);
```

Onde $param é uma variável que pode vir de uma consulta a banco de dados ou mesmo de um formulário.

Outra função **HTTP** importante é o uso de "cookies" para gravar alguma informação no browser de quem estiver visitando sua página. Para gravar um "cookie", usamos a função setcookie(), como mostrado abaixo:

```
setcookie("Visitou", "Sim", time()+3600);
```

O comando acima gravará um cookie chamado "Visitou" com o valor "sim", com apenas uma hora de duração. Note que o 3600 é o número de segundos além do horário atual que o cookie deve ficar ativo. Se no lugar de 3600, usássemos 36000, o cookie seria ativo por 10 horas.

FTP: A seqüência de tarefas para uso do protocolo FTP é Conectar ao Servidor, Identificar-se (Login e Senha), Enviar/Buscar arquivo(s), Desconectar. Os comandos para cada uma destas tarefas são:

```
$conn = ftp_connect(“ftp.inf.ufsc.br”);
$log = ftp_login($conn, 'login', 'pass');
ftp_put($con, 'arquivo_remoto', 'arquivo_local',
FTP_ASCII/FTP_BINARY);
ftp_quit($conn);
```

2.13 - Sessões

Sessões HTTP servem para preservar dados em acessos subseqüentes, através de registros de variáveis de sessão. Usamos basicamente três funções:

- session_start();
- session_register(”VARIÁVEL");
- session_destroy();

Exemplo:

```
<?php
session_start();
session_register("VARIAVEL");
if (!isset($VARIAVEL)) {
  header("Location: error.php");
  exit();
}
print "Passou!!";
?>
```

2.14 - Tratamento de Arquivos

O PHP possui várias funções para o tratamento de arquivos a fim de facilitar sua manipulação. A primeira coisa que se tem que saber é

que para se manipular arquivos, tem-se que abrir e fechar o arquivo. O PHP possui as seguintes funções para se abrir e fechar arquivos:

- fopen(“*nome_do_arquivo*”, “*mode*”, *[diretório]*) – Abre o arquivo
- fclose(*fp*) - Fecha o arquivo

Estes modos de abrir o arquivo podem ser os seguintes:

r somente leitura; posiciona-se no início do arquivo.

r+ leitura e escrita; posiciona-se no início do arquivo.

w somente escrita; posiciona-se no início do arquivo e o trunca para tamanho zero. Caso o arquivo não exista o PHP tenta criá-lo.

w+ leitura e escrita; posiciona-se no início do arquivo e o trunca para tamanho zero. Caso o arquivo não exista o PHP tenta criá-lo.

a somente escrita; posiciona-se no final do arquivo. Caso o arquivo não exista o PHP tenta criá-lo.

a+ leitura e escrita; posiciona-se no final do arquivo. Caso o arquivo não exista o PHP tenta criá-lo.

Após abrir um arquivo, pode-se ler o conteúdo ou escrever em seu conteúdo, de acordo com a forma que se abriu o arquivo.

- fread(*fp*, *tamanho*)
- fgets(*fp*, *tamanho*)
- fwrite(*fp*, *string*, *[tamanho]*)

Exemplos:

```
$nomedoarquivo = "/tmp/teste.txt";
$fp = fopen ($nomedoarquivo, "r");
$contents = fread ($fd, filesize ($nomedoarquivo));
fclose ($fp);
```

```
$fp = fopen(“/tmp/teste.txt”, “a”);
fwrite($fp, “bla bla bla \n”);
fclose($fp);
```

```
$fp = fopen ("/tmp/teste.txt", "r");
while ( !feof($fp) ) {
  $buffer = fgets($fp, 4096);
  echo “$buffer <br>”;
}
fclose ($fp);
```

Neste último exemplo, pega-se linha a linha de um arquivo e imprime em HTML separando as linhas pela tag “
”.

2.15 - Tratamento de Erros

O PHP tem um esquema especial de "debugging" (tratamento e verificação de erros), que é acessado por uma porta TCP, que permite acompanhar a execução dos scripts e ver quaisquer erros que estiverem acontecendo.

Além disso o tratamento de erros pode ser feito no próprio script, conforme explicado abaixo:

O PHP possui 4 níveis de erros e avisos, que são:

1 - Erros de normais de Funções
2 - Avisos Normais
4 - Erro de interpretação
8 - Avisos que você pode ignorar, mas que podem causar danos à execução normal do script.

O padrão do PHP é o nível 7 (1 + 2 + 4), mas este nível pode ser alterado tanto no arquivo de configuração quanto em tempo de execução, chamando a função `error_reporting($nivel)` com o nível desejado.

Se usarmos o valor 0 (zero) nenhum aviso ou mensagem de erro será gerada em tempo de execução. Neste caso, podemos usar uma variável especial (`$php_errormsg`) que conterá o último erro gerado pelo script, para que possamos criar rotinas específicas para tratamento de erros. Podemos fazer uma analogia ao comando on error resume next do ASP, técnica muito útil para personalizarmos mensagens de erro para o usuário, entre outras coisas.

Bibliografia

http://www.ibestmasters.com.br

http://www.weberdev.com

http://www.php.net

http://www.webmonkey.com

http://www.devshed.com

http://www.mysql.com

http://www.phpbuilder.com

TRIDAPALI, Graziela W. & SANT'ANNA, Juliana S. Trabalho de Conclusão de Curso – "Construindo uma aplicação de Comércio Eletrônico"