CONEXÃO COM BANCO DE DADOS

Fatec Jundiaí

Prof. Humberto Zanetti - prof.zanetti@fatec.sp.gov.br

MySQL

- SGBD criado em 1996;
- Multi-plataforma;
- Suporte a múltiplos processadores;
- Um sofisticado sistema de senhas; criptografadas flexível e Seguro;
- Banco de dados de código aberto e gratuito;
- Suporte as API´s de várias linguagens;
- O Cliente conecta no MySQL através de conexões TCP/IP.

Tipos de dados numéricos

- Existem tipos de dados numéricos, que se podem dividir em dois grandes grupos, os de vírgula flutuante (com decimais) e os que não.
- **TinyInt**: é um número inteiro com ou sem sinal. Com sinal a margem de valores válidos é desde -128 até 127. Sem sinal, a margem de valores é de 0 até 255
- **SmallInt**: número inteiro com ou sem sinal. Com sinal a margem de valores válidos é desde -32768 até 32767. Sem sinal, a margem de valores é de 0 até 65535.
- **MediumInt**: número inteiro com ou sem sinal. Com sinal a margem de valores válidos é desde -8.388.608 até 8.388.607. Sem sinal, a margem de valores é de 0 até 16777215.
- **Int**: número inteiro com ou sem sinal. Com sinal a margem de valores válidos é desde -2147483648 até 2147483647. Sem sinal, a margem de valores é de 0 até 429.496.295
- **BigInt:** número inteiro com ou sem sinal. Com sinal a margem de valores válidos é desde -9.223.372.036.854.775.808 até 9.223.372.036.854.775.807. Sem sinal, a margem de valores é de 0 até 18.446.744.073.709.551.615.

Tipos de dados numéricos

- **Float:** número pequeno em vírgula flutuante de precisão simples. Os valores válidos vão desde - 3.402823466E+38 até -1.175494351E-38,0 e desde 175494351E-38 até 3.402823466E+38.
- **Double:** número em vírgula flutuante de dupla precisão. Os valores permitidos vão desde - 1.79769313486231 57E+308 até - 2.2250738585072014E-308, 0 e desde 2.2250738585072014E-308 até 1.79769313486231 57E+308
- **Decimal:** Número em vírgula flutuante desempacotado. O número armazena-se como uma cadeia. **Decimal (I,D)**

Tipos de dados numéricos

Tipo de Campo	Tamanho de Armazenamento
TINYINT	1 byte
SMALLINT	2 bytes
MEDIUMINT	3 bytes
INT	4 bytes
BIGINT	8 bytes
INTEGER	4 bytes
BIGINT	8 bytes
FLOAT	4 bytes
DOUBLE	8 bytes
DECIMAL(M,D)	M+2 bytes se D > 0, M+1 bytes se D = 0

Tipos de dados “data”

- **Date:** tipo data, armazena uma data. A margem de valores vai desde o 1 de Janeiro de 1001 ao 31 de dezembro de 9999. O formato de armazenamento é de ano-mes-dia.
- **DateTime:** Combinação de data e hora. A margem de valores vai desde o 1 ed Janeiro de 1001 às 0 horas, 0 minutos e 0 segundos ao 31 de Dezembro de 9999 às 23 horas, 59 minutos e 59 segundos. O formato de armazenamento é de ano-mes-dia horas:minutos:segundos
- **TimeStamp:** Combinação de data e hora. A margem vai desde o 1 de Janeiro de 1970 ao ano 2037. O formato de armazenamento depende do tamanho

Tamanhodo campo	Formato
14	AnoMesDiaHoraMinutoSegundo aaaammddhhmmss
12	AnoMesDiaHoraMinutoSegundo aammddhhmmss
8	AnoMesDia aaaammdd
6	AnoMesDia aammdd
4	AnoMes aamm
2	Ano aa

Tipos de dados “data”

- **Time:** armazena uma hora. A margem de horas vai desde -838 horas, 59 minutos e 59 segundos. O formato de armazenamento é 'HH:MM:SS'.
- **Year:** armazena um ano. A margem de valores permitidos vai desde o ano 1901 ao ano 2155. O campo pode ter tamanho dois ou tamanho 4 dependendo de se queremos armazenar o ano com dois ou quatro algarismos.

Tipo de Campo	Tamanho de Armazenamento
DATE	3 bytes
DATETIME	8 bytes
TIMESTAMP	4 bytes
TIME	3 bytes
YEAR	1 byte

Tipos de dados caracteres

- **Char(n):** armazena uma cadeia de longitude fixa. A cadeia poderá conter desde 0 até 255 caracteres.
- **VarChar(n):** armazena uma cadeia de longitude variável. A cadeia poderá conter desde 0 até 255 caracteres. Dentro dos tipos de cadeia pode-se distinguir dois subtipos, os tipo Test e os tipo Blob (Binary Large Object) A diferença entre um tipo e outro é o tratamento que recebem na hora de ordená-los e compará-los. No tipo test ordena-se sem ter importância as maiúsculas e as minúsculas e no tipo blob ordena-se tendo em conta as maiúsculas e minúsculas.

Tipos de dados caracteres

- **TinyText e TinyBlob:** Coluna com uma longitude máxima de 255 caracteres.
- **Blob e Text:** um texto com um máximo de 65535 caracteres.
- **MediumBlob e MediumText:** um texto com um máximo de 16.777.215 caracteres.
- **LongBlob e LongText:** um texto com um máximo de caracteres 4.294.967.295. Há que ter em conta que
- devido aos protocolos de comunicação os pacotes podem ter um máximo de 16 Mb.
- **Enum:** campo que pode ter um único valor de uma lista que se especifica. Aceita até 65535 valores diferentes.

Comandos básicos - DDL

- **Comando Create Database**
- Este comando permite a criação do banco de dados.
- **Sintaxe:**

```
CREATE DATABASE < nome_db >;
```

onde:

nome_db - indica o nome do Banco de Dados a ser criado.

- **Exemplo:**

```
Create database TESTE;
```

Comandos básicos - DDL

- **Comando Create Table**
  - Este comando permite a criação de tabelas no banco de dados.
- **Sintaxe:**

```
CREATE TABLE < nome_tabela > (
nome_atributo1 < tipo > [ NOT NULL ],
nome_atributo2 < tipo > [ NOT NULL ],
.....
nome_atributoN < tipo > [ NOT NULL ]
) ;
```

onde:

nome_table - indica o nome da tabela a ser criada.

nome_atributo - indica o nome do campo a ser criado na tabela.

tipo - indica a definição do tipo de atributo (integer(n), char(n), ...).

Comandos básicos - DDL

- **Exemplo:**

```
Create table alunos (
Id_aluno UNSIGNED INT(3) NOT NULL,
nome CHAR(40) NOT NULL,
endereco CHAR (50) NOT NULL
turma CHAR(20) NOT NULL,
PRIMARY KEY (matricula)
);
```

Comandos básicos - DDL

- **Comando Drop**
  - Este comando elimina a definição da tabela, seus dados e referências.
- **Sintaxe:**
  - `DROP TABLE < nome_tabela > ;`
- **Exemplo:**
  - `Drop table alunos;`

Comandos básicos - DDL

- **Comando Alter**
  - Este comando permite inserir/eliminar atributos nas tabelas já existentes.
- **Sintaxe:**

```
ALTER TABLE < nome_tabela > ADD / DROP (
nome_atributo1 < tipo > [ NOT NULL ],
nome_atributoN < tipo > [ NOT NULL ]
);
```

- **Exemplo:**

```
Alter table alunos ADD COLUMN turno char(10)
NOT NULL;
```

Comandos básicos - DML

- **Comando SELECT**
  - Permite recuperar informações existentes nas tabelas.
- **Sintaxe:**

```
SELECT [DISTINCT] expressao [AS nom-
atributo] [FROM from-list] [WHERE
condicao] [ORDER BY attr_name1 [ASC |
DESC ]
```

Comandos básicos - DML

- onde:
  - **DISTINCT** - Para eliminar linhas duplicadas na saída.
  - **Expressão** - Define os dados que queremos na saída, normalmente uma ou mais colunas de uma tabela da lista FROM.
  - **AS nome-atributo** - um alias para o nome da coluna
  - **FROM** - lista das tabelas na entrada
  - **WHERE** - critérios da seleção
  - **ORDER BY** - Critério de ordenação das tabelas de saída. Podem ser:
  - **ASC** - ordem ascendente (crescente);
  - **DESC** - ordem descendente (decrescente)
- **Exemplo:**

```
Select cidade, estado from brasil where
populacao > 100000 order by Desc;
```

Comandos básicos - DML

- **Comando INSERT**
  - Adiciona um ou vários registros a uma tabela. Isto é referido como consulta anexação.
- **Sintaxe:**

```
INSERT INTO destino [(campo1[, campo2[, ...]])]
VALUES (valor1[, valor2[, ...])
```

onde;

- **Destino -** O nome da tabela ou consulta em que os registros devem ser anexados.
- **campo1, campo2** - Os nomes dos campos aos quais os dados devem ser anexados
- **valor1, valor2** - Os valores para inserir em campos específicos do novo registro. Cada valor é inserido no campo que corresponde à posição do valor na lista: Valor1 é inserido no campo1 do novo registro, valor2 no campo2 e assim por diante.

Comandos básicos - DML

- Os valores devem ser separados com uma vírgula e os campos de textos entre aspas duplas ou simples.
- **Exemplo:**

```
Insert into alunos (Id_aluno, nome,
endereço, turma, turno)
Values (1, 'Maria', 'Av. das
Américas', '1101', 'manhã');
```

Comandos básicos - DML

- **Comando UPDATE**
  - Cria uma consulta atualização que altera os valores dos campos em uma tabela especificada com base em critérios específicos.
- **Sintaxe:**

```
UPDATE tabela SET campo1 = valornovo, ... WHERE critério;
```

onde:

- **Tabela** - O nome da tabela cujos dados você quer modificar.
- **Valornovo** - Uma expressão que determina o valor a ser inserido em um campo específico nos registros atualizados.
- **critério** - Uma expressão que determina quais registros devem ser atualizados. Só os registros que satisfazem a expressão são atualizado.
- **Exemplo:** `Update alunos Set turno = 'tarde' where turma = '1101';`

Comandos básicos - DML

- **Comando DELETE**
  - Remove registros de uma ou mais tabelas listadas na cláusula FROM que satisfaz a cláusula WHERE.
- **Sintaxe:**

```
DELETE [tabela.*]
FROM tabela
WHERE critério
```

onde:

- **tabela.*** - O nome opcional da tabela da qual os registros são excluídos.
- **tabela** - O nome da tabela da qual os registros são excluídos.
- **critério** - Uma expressão que determina qual registro deve ser excluído.
- **Exemplo:** `Delete from alunos WHERE turno='Manhã';`

Conexão – PHP com MySQL

- **mysql_connect( )**
  - Comando utilizado para criar a conexão com a base de dados num servidor MySQL. O comando **mysql_connect** também pode ser utilizado para criar a conexão. A diferença entre os dois comandos é que o **mysql_connect** estabelece uma conexão permanente, ou seja, que não é encerrada ao final da execução do script. Utilizaremos a primeira opção: ***mysql_connect***.
- **Sintaxe:**

```
mysql_connect(string [host[:porta]] , string [login] , string [senha] ) [ or die (“mensagem de erro”)];
```

onde:

- **string [host[:porta]] –** é o endereço do servidor, onde o banco está armazenado;
- **string [login] –** é o usuário do banco de dados;
- **string [senha] –** é a senha do usuário do banco de dados MySQL.
- **die –** parâmetro opcional que exibe uma mensagem indicando que a conexão não foi efetuada.

Conexão – PHP e MySQL

- **Exemplo:**

```
<?
$conexao = mysql_connect ("localhost", "root",
"teste") or die ("Conexão não efetuada");
?>
```

- O valor de retorno é um inteiro que identifica a conexão, ou falso se a conexão falhar. Antes de tentar estabelecer uma conexão, o interpretador PHP verifica se já existe uma conexão estabelecida com o mesmo host, o mesmo login e a mesma senha. Se existir, o identificador desta conexão é retornado. Senão, uma nova conexão é criada.
- Assim, se a conexão for bem sucedida (existir um servidor no endereço especificado que possua o usuário com a senha fornecida), o identificador da conexão fica armazenado na variável $conexao, caso contrário será mostrada a mensagem "Conexão não efetuada".

Conexão – PHP e MySQL

- **mysql_close()**
  - Comando utilizado para encerrar uma conexão estabelecida com o comando mysql_connect antes de chegar ao final do script. Caso esse comando não seja utilizado a conexão é encerrada no final do script.
- **Sintaxe:**

```
mysql_close(int [string da conexão] );
```

- Se o identificador não for fornecido, a última conexão estabelecida será encerrada.
- **Exemplo:**

```
<?
mysql_close ($conexao);
?>
```

Conexão – PHP e MySQL

- **mysql_select_db( )**
  - Comando utilizado para selecionar a base de dados depois que a conexão for estabelecida. Se nenhuma de conexão é especificado, a ultima conexão aberta é assumida. Se nenhuma conexão esta aberta, a função irá tentar abrir uma conexão como se mysql_connect() fosse chamada sem argumentos e usá-la.
- **Sintaxe:**

```
mysql_select_db(banco de dados, [string de
conexao]) or die [(“mensagem”)];
```

onde:

- **banco de dados –** é o banco de dados que será utililzado;
- **string de conexão –** é a conexão criada com o servidor MySQL.
- **die –** parâmetro opcional que exibe uma mensagem indicando que a conexão não foi efetuada.

Conexão – PHP e MySQL

- Exemplo:

```
<?
$conexao = mysql_connect(“localhost”, “root”, “teste”);
if ($conexao) {
     die (“Conexão não estabelecida”);
}
db_select = mysql_select_db(“teste”, $conexao);
?>
```

Conexão – PHP e MySQL

- **mysql_query( )**
  - Este comando é utilizado para realizar uma consulta SQL no MySQL.
- **Sintaxe:**

```
mysql_query(string da consulta);
```

onde:

- **string da consulta –** é um dos comandos utilizados do SQL para efetuar uma consulta, uma inclusão, uma alteração ou uma exclusão no banco de dados.O comando mysql_query() envia uma query (consulta) para o banco de dados ativo no servidor da conexão informada em **string da consulta**. Se o parâmetro **string da consulta** não é especificado, a ultima conexão aberta é usada. Se nenhuma conexão esta aberta, a função tenta estabelecer uma conexão como mysql_connect() seja chamada sem argumentos e usá-la. O resultado é guardado em buffer.
- **Exemplo:**

```
<? sql = "select * from alunos where id_aluno = 10";
        mysql_query (sql);
?>
```

Conexão – PHP e MySQL

- Exemplo de um script PHP

```
<?
$conexao = mysql_connect (“localhost”, “root”, “teste”);
 mysql_select_db(“teste”, $conexão);
$insere = “Insert into alunos (id_aluno, nome, endereço, turma, turno)
values (1, ‘Maria’, ‘Av. das Américas’, ‘1101’, ‘manhã’)”;
$insere1 = “Insert into alunos (id_aluno, nome, endereço, turma, turno)
values (2, ‘José, ‘Av. das Américas’, ‘1101’, ‘tarde’)”;
$insere2 = “Insert into alunos (id_aluno, nome, endereço, turma, turno)
values (3, ‘João’, ‘Av. das Américas’, ‘1101’, ‘tarde’)”;
mysql_query ($insere, $conexao) or die (“Não foi possível executar a inserção.”);
mysql_query ($insere1, $conexao) or die (“Não foi possível executar a inserção.”);
mysql_query ($insere2, $conexao) or die (“Não foi possível executar a inserção.”);
$delete = Delete from alunos where turno = “tarde”);
mysql_query ($delete, $conexao);
echo (“ todos os alunos do turno da tarde foram excluídos.”);
mysql_close ($conexao);
?>
```

Consultas

- **mysql_result( )**
  - Esta função retorna o resultado de uma query SQL.
- **Sintaxe:**

```
mysql_result(resultado, linha, mixed [campo]);
```

onde:

- **resultado** - é o identificador do resultado, obtido com o retorno da função mysql_query;
- **linha** - especifica o registro a ser exibido, já que uma query SELECT pode retornar diversos registros;
- **campo** - é o identificador do campo a ser exibido, sendo o tipo descrito como **mixed** pela possibilidade de ser de diversos tipos (neste caso, inteiro ou string).

Consultas

- **Exemplo:**

```
<?
$insere = "Insert into alunos (id_aluno, nome, endereço, turma, turno)
Values (1, 'Maria', 'Av. das Américas', '1101', 'manhã')";
$insere1 = "Insert into alunos (id_aluno, nome, endereço, turma, turno)
Values (2, 'José', 'Av. das Américas', '1101', 'tarde')";

mysql_query ($insere, $conexao) or die ("Não foi possível executar a
inserção.");
mysql_query ($insere1, $conexao) or die ("Não foi possível executar a
inserção.");

$consulta = "Select Id_Aluno, nome, turno from alunos";
$resultado = mysql_query ($consulta, $conexao);
$nome = mysql_result($resultado,0,"nome");
$turno = mysql_result($resultado,0,"turno");
echo ("Nome: ".$nome."<p>"."Turno: ".$turno);
?>
```

Consultas

- **mysql_fetch_array( )**
  - Esta função lê uma linha do resultado e devolve um array, cujos índices são os nomes dos campos. A execução seguinte do mesmo comando lerá a próxima linha, até chegar ao final do resultado.
- **Sintaxe:**
- `mysql_fecth_array(string da consulta SQL);`
- Exemplo:

```
<? ...
    $dados = mysql_fecth_array($resultado);
    $nome = $dados[“nome”];
    $turno =$dados[turno];
    ...
    >
```

Consultas

- **mysql_fetch_row( )**
  - Esta função é semelhante a função mysql_fetch_array, com a diferença que os índices do array são numéricos, iniciando pelo 0 (zero).
- **Sintaxe:**

```
mysql_fetch_row(string de consulta);
```

Consultas

- **mysql_free_result( )**
  - Esta função libera a memória do resultado de uma consulta.
- **Sintaxe:**

```
mysql_free_result ( resource result )
```

Exemplo:

```
<?
$dados = mysql_fecth_array($resultado);
$nome = $dados[“nome”];
$turno =$dados[turno];
echo (“Nome: “.$nome.”<p>”.“Turno: “.$turno);
mysql_free_result($resultado);
?>
```

- No exemplo acima, a função **mysql_free_result( )** irá liberar toda a memória usada com o identificador de resultado $resultado.

Obs.: mysql_free_result() somente precisa ser chamado se você esta preocupado em quantamemória esta sendo usada para query num grande conjunto de resultados. Toda a memória usada do resultado é liberada automaticamente ao final da execução do script.

Estudo de Caso – Sistema Login

- **setcookie():** define um cookie para ser enviado
  - http://www.php.net/manual/pt_BR/function.setcookie.php
- **header():** envia o cabeçalho HTTP
  - http://www.php.net/manual/pt_BR/function.header.php
- **md5():** método de encriptação de dados
  - http://www.php.net/manual/pt_BR/function.md5.php
- **empty():** informa se a variável é vazia
  - http://www.php.net/manual/pt_BR/function.empty.php
- **Cookies**
  - http://www.php.net/manual/pt_BR/features.cookies.php

Estudo de caso – Cadastro Aluno

Banco de Dados do Sistema (BD "turmas")

Aluno

Atributo	Tipo	Nulo?	Extra
ra (PK)	varchar(6)	Não	
nome	varchar(50)	Não	
telefone	varchar(12)	Não	
turma (FK)	int(10)	Não	

Turma

Atributo	Tipo	Nulo?	Extra
cod (PK)	int(10	Não	Auto_increment
curso	varchar(30)	Não	
telefone	int(2)	Não	
turma	int(4)	Não	