

# Minicurso: PHP & SQLite

# Minicurso: PHP & SQLite

Um breve momento até boa parte dos participantes chegar... (máx. 10 minutos)



# Apresentação

Ministrante:

Luiz Antonio Santos Gonsalves Junior,

Formação como Técnico em Informática, atualmente matriculado no curso Bacharelado em Ciência da Computação. Entusiasta/estudante da linguagem PHP, tendo sido apresentado a esta há cerca de 6 anos e trabalhando esporadicamente com a mesma há cerca de 3 anos e meio.

Lattes:

<http://buscatextual.cnpq.br/buscatextual/visualizacv.do?id=K8875076Y8>

# Disposições iniciais

## Pré-requisitos:

- Conhecimento de algoritmos e estrutura de dados;
- Conhecimento básico sobre o paradigma Orientado à Objetos;
- Noções de Banco de dados;
- Conhecimento básico sobre PHP (opcional);
- Conhecimento em tecnologias web (opcional);

Objetivo: Aquisição de conhecimentos básicos sobre as tecnologias PHP e SQLite, bem como, desenvolver tais conhecimentos através da prática em programação.

# Disposições iniciais

## Sugestões:

- Crie uma pasta com o nome “Minicurso PHP - SEUNOME”, para facilitar a locação das atividades;
- Verifique se a conexão com a Internet está funcionando e reporte qualquer outro possível problema com sua máquina;
- Recomenda-se o uso de um editor de texto / IDE para este minicurso. Pessoalmente, recomendo o uso do Visual Studio Code.

# Desenvolvimento Backend, cenário atual

Nos tempos mais primórdios, ou seja, em **meados da década de 1990**, surgia a World Wide Web (WWW), através dos esforços de Berners-Lee e de seus colaboradores.

No início, a mesma foi desenvolvida para intermediar a troca de informações dos experimentos do CERN (onde Berners-Lee trabalhava), com pesquisadores ao redor do planeta. Através da WWW, era possível a troca de informações em hipermídia, utilizando-se do HyperText Transfer Protocol (HTTP), protocolo desenvolvido com o intuito de reger a WWW e o HTML, uma linguagem de marcação, para criar documentos de hipertexto.

# Desenvolvimento Backend, cenário atual

Desde a metade da década de 1990, o desenvolvimento de tecnologias para a Internet vêm não só alavancando diversas profissões, como também, desenvolvendo suas próprias.

O desenvolvimento de aplicações para a web é talvez o mais claro exemplo disto. Hoje em dia, é praticamente impossível sustentar algum mercado sem que o mesmo necessite integração com a WWW, haja visto por exemplo, as diversas plataformas de comércio existentes na mesma.

# Desenvolvimento Backend, cenário atual

Também para outros aspectos como divulgação, educação, socialização e muitos outros, tal tipo de tecnologia se tornou indispensável.



Cenário atual: diversas tecnologias disponíveis, necessidade de mão de obra qualificada, inúmeras possibilidades dentro da área...



# Desenvolvimento Backend, cenário atual

Um exemplo mais “estatístico”:

- De uma amostra com **36** vagas de emprego para programadores, num raio de 100Km (Aprox.) da cidade de Rio Pomba, foram constatadas: **70%** (25) das vagas se destinavam a desenvolvedores web, **14%** (5) das vagas se destinavam desenvolvedores mobile, **8%** (3) das vagas se destinavam à desenvolvedores desktop e outros **8%** (3) não especificavam.

*Amostra avaliada em agosto/2019, considerando vagas para “programadores em geral” disponíveis na plataforma de busca de empregos do Google. Cálculos de porcentagem aproximados para melhor visualização. Margem de 1%.*

# Desenvolvimento Backend, cenário atual

Desenvolvedores web tendem a ter preferências específicas, geralmente de acordo com suas habilidades, sobre o foco dentro de sua área de atuação. Em suma, chega um momento na vida do desenvolvedor em que o mesmo tem que escolher entre:

- Backend: Trabalhar no lado da aplicação, desenvolvendo funcionalidades.
- Frontend: Trabalhar no lado do cliente, desenvolvendo a usabilidade.
- FullStack: Proficiência nas técnicas e tecnologias de Backend e Frontend.
- Outras denominações (if any???)...

# O que é PHP?

PHP é, de maneira curta e direta, uma linguagem de programação. Muito provavelmente você já ouviu falar dela, mas mesmo que não, se você já acessou algum site hoje há uma grande chance de você ter tido contato com alguma página que faz uso do PHP.

PHP é uma linguagem de programação interpretada, multiparadigma e de tipagem fraca e dinâmica/gradual. O foco da mesma é o desenvolvimento de aplicações para web no lado do servidor (backend).

“PHP é uma popular linguagem de scripting de propósito geral, que é especialmente apropriada para desenvolvimento Web” (php.net)

# O que é PHP?

Uma versão completa do histórico da linguagem está disponível pelo link:

<http://php.net/manual/history.php.php>

Atualmente, PHP é uma das linguagens mais utilizadas no mundo. Estima-se que cerca de 79% dos servidores no mundo executem alguma aplicação em PHP ([w3techs.com](http://w3techs.com), 2019). No Brasil, em especial, PHP é altamente adotada.

Fato é, que a mesma ganhou novamente foco em proporções muito elevadas após o lançamento da sua versão 7.

# O que é PHP?

Podemos ainda, citar algumas empresas que a utilizam:

- Google
- Facebook
- Spotify
- Baidu
- Toyota
- Entre outras...



WIKIPÉDIA  
A enciclopédia livre

# O que é um Banco de Dados?

Grande parte das aplicações rodando na web atualmente se baseia na premissa que pode-se armazenar dados em um sistema, e que tais dados podem posteriormente, vir a ser (re)utilizados.

Um banco de dados vem a ser uma **coleção estruturada de dados, com a finalidade de se criar algum sentido sobre tais dados e também, proporcionar eficiência no seu uso.**

Não são novidade, no entanto, pois desde o século passado são uma conhecida e usável tecnologia.

# O que é SQLite?

A tecnologia SQLite se trata de uma biblioteca que implementa por si uma **engine SQL bastante prática**. É em si a base de dados e seu sistema de gerenciamento.

Diferentemente de outras tecnologias, SQLite é voltada a portabilidade e praticidade. Não se baseia em cliente-servidor e não requer parâmetros de configuração. A base de dados inteira é contida dentro de um único arquivo.

# O que é SQLite?

Esta tecnologia foi desenvolvida em meados de 2000, pela empresa General Dynamics, mais especificamente pelo arquiteto de sistemas **D. R. Hipp**, através de um contrato com a marinha dos EUA. A mesma tinha a finalidade de ser aplicada em sistemas embarcados (piada não intencional) de **navios destroyers**.





# O que é SQLite?

A mesma foi distribuída livremente desde o princípio, chegando atualmente à versão 3.

Dentre os usos mais notáveis do SQLite se encontram aplicações para a plataforma Android, a nível de SO no Windows 10, em programas “Antivirus”, dispositivos embarcados, arquivos de configuração e, é claro, para desenvolvimento web.

Esta última tem sido mais recentemente explorada, porém com resultados bastante efetivos.

# Casos de uso

A primeira vista, pode-se visualizar o SQLite como uma versão “mínima” do MySQL (ou outro SGBD), talvez por conta do sufixo “Lite” (Leve, “com menor conteúdo”, “menor escala”). Apesar de não ser completamente errônea, **esta visão está longe de ser a realidade.**



# Casos de uso

Alguns casos de uso possíveis para PHP juntamente com SQLite seriam:

- Sistemas web simples (Blogs, plataformas comerciais, notícias...);
- Protótipos;
- APIs;
- Sistemas de rede local;
- Sistemas de gerência de IOT e Dashboards;
- O que quer que um programador que domina a linguagem queira...



# Ambiente do minicurso

PHP possui suporte à diversas plataformas, como Linux, Windows e MacOS. Por ser uma linguagem já bem consolidada, existem formas bastante simplificadas de instalação do mesmo na maior parte dos sistemas.

Não fugindo a isto, o SQLite também suporta grande parte das plataformas atuais.

Para integração de ambos, também é necessário o uso de pacotes específicos, geralmente de nomeação intuitiva (porém variando conforme a plataforma).

# Ambiente do minicurso

A partir da versão 5.4, o PHP dispõe de um servidor para testes implementado em sua distribuição padrão. Lembre-se, PHP é uma linguagem voltada ao desenvolvimento backend Web.

Isto nos permite muito mais facilidades ao testar nossos projetos, visto que não é necessária configuração prévia de servidor ou algo do tipo. Para executar o servidor embutido, utilize o comando (também é útil para verificarmos se o ambiente PHP está ok):

```
$ php -S localhost:8000
```

# Revisão: PHP

# Introdução

Como nosso primeiro programa em PHP para este minicurso, vamos começar com o famoso “Hello World!”.

Uma coisa importante sobre este primeiro programa é que o mesmo servirá para testarmos nosso ambiente de desenvolvimento e verificar se tudo está ok.

Crie um arquivo “index.php”, em seguida, preencha com o código a seguir:


```
1 <?php
2
3     echo "Hello World!";
4
5 ?>
```

# Introdução

Com o servidor de testes em execução, abra o link:

<http://localhost:8000>

O resultado deverá ser algo como:



Hello World!

O navegador fez uma requisição ao “servidor embutido” do PHP, que tratou de responder com a execução do script. Por padrão “index.php” é um nome especial para tais arquivos.



# Sintaxe

Vamos revisar o básico sobre a linguagem PHP em um primeiro momento.

Todo trecho de script php deve conter as tags de abertura e fechamento: “<?php ... ?>”. Existem outros formatos, porém este é considerado como o padrão, sendo uma boa prática adotá-lo.

Programas em PHP são interpretados “top-bottom”, como na maioria das linguagens. Uma engine interna (Geralmente construída sobre linguagens como C/C++) é responsável por essa interpretação. A mais comum delas é conhecida como **Zend Engine**.

# Sintaxe

Destacamos a seguir alguns tópicos de interesse para revisarmos a respeito da sintaxe:

- Variáveis e constantes:

```
<?php
/*Para qualquer script PHP, as tags de
abertura e fechamento devem existir*/

declare(strict_types = 1); //Ativação do modo strict

$boolean = TRUE;
$uma_variavel_numerica = 100; //Lembre-se sempre do ";"
$variavel_double = 1.666;
$variavel_texto = "Nao leia esta string";
$outro_texto = 'E também não leia esta...';
```

# Sintaxe

```
define("BD", "endereco/do/bd"); //Definindo uma constante "BD"
```

```
$arr = [2, 5, 0, 1];
```

```
$arr2 = arr(2, 5, 0, 1); //Equivalente ao anterior
```

```
$arr = [  
    produto => "Tomate",  
    quantidade => 3,  
    valor_un => 0.85  
]; //Array associativo
```

# Sintaxe

- Operadores:

```
//EXEMPLOS DE OPERADORES
```

```
$res = (((1 + 1) - 3) / 2) * 0; //Aritméticos  
$res = $res % 2;
```

```
TRUE == FALSE;  
TRUE != FALSE;  
2 >= 0;  
2 === 2; //Comparação
```

```
$hello = "Hello"."World"; //Concatenação
```

```
$cost = 1; //Atribuição  
$cost += 1;  
$cost++;
```

```
TRUE && TRUE;  
TRUE || FALSE; //Lógicos
```

# Sintaxe

- Condicionais e repetição:

```
$nome_usr = "Pastelarino";

if($nome_usr != ""){
    echo "Nome válido";
}else if(strlen($nome_usr) < 3){
    echo "Caracteres insuficientes";
}else{
    echo "Nome Inválido";
}
```

*//Condicionais e repetição*

```
for($i = 0; $i < 8; $i++){
```

```
    if($i % 2 == 0){
        echo $i;
    }
```

*}//Impressão de valores pares*

```
$num = 1;
```

```
while($num < 100){
```

```
    $num += $num;
```

```
}//Calculo de dobro
```

# Sintaxe

- Funções:

```
//FUNÇÕES
```

```
function calcula_desconto($valor_total, $desconto = 0){
```

```
    return $valor_total - ($valor_total * $desconto);
```

```
}
```

```
function calcula_desconto_strict($valor_total, $desconto = 0): float{
```

```
    return $valor_total - ($valor_total * $desconto);
```

```
}//Com especificação de retorno (modo strict);
```

```
//FIM DO SCRIPT
```

```
?>
```

# Exercício #0

Crie um script PHP para uma calculadora simples de duas entradas. Para tanto, implemente:

- Funções para cada uma das 4 operações básicas (duas entradas);
- Função “maior”: compara dois números e retorna o maior entre eles;
- Função de exponenciação (sem o operador, utilize um loop ao invés disso);

Teste cada uma das funções implementadas.

# Tipos de dados

O fato de que você não precisa declarar os tipos em PHP não significa que estes não existam. O que acontece é que os mesmos são inferidos automaticamente pela linguagem.

Dentre os tipos definidos pelo PHP, destacamos os Integers, Double, Boolean, Strings (tipos simples) e também os Arrays, Objects e Resources (tipos compostos).

Os mesmos podem também ser convertidos entre si. Para um panorama mais detalhado, visite: [https://www.php.net/manual/pt\\_BR/language.types.php](https://www.php.net/manual/pt_BR/language.types.php)



# Orientação à Objetos

Um paradigma muito importante e que a linguagem PHP possui suporte é o paradigma Orientado à Objetos (OO). Até a chegada da versão 5, o suporte à OO em PHP era bastante precário.

Na versão 5, no entanto, isto foi corrigido. Através dela, foi introduzido um suporte mais estável à OO, bem como, a remodelagem de parte daquilo que já existia na linguagem. A introdução de uma nova forma de se operar persistência de dados também foi algo notável nessa versão.

A seguir, alguns construtos básicos e através deles, como aplicar OO em PHP.

```
<?php
    class Automovel{

        private $marca; //ATRIBUTO de acesso privado
        private $modelo;
        private $autonomia;

        public function __construct($marca, $modelo, $autonomia){ //Construtor

            $this->marca = $marca;
            $this->modelo = $modelo;
            $this->autonomia = $autonomia;

        }

        //Métodos Getter e Setter para o atributo $autonomia
        public function get_autonomia(){
            return $this->autonomia;
        }

        public function set_autonomia($autonomia){
            $this->autonomia = $autonomia;
        }

    }
```

# Orientação à Objetos

```
class Carro extends Automovel{  
    public function __construct($marca, $modelo, $autonomia){  
        parent::__construct($marca, $modelo, $autonomia);  
    }  
  
    public function chegou_a_sexta(){  
        echo "Complete o meme depois...";  
    }  
}  
  
$carro1 = new Carro("Fiat", "Uno", "50");  
  
$carro1->chegou_a_sexta();  
echo $carro1->get_autonomia();
```

?>

# Exercício #1

Elabore e implemente uma classe “Empregado”. A mesma deverá conter, pelo menos 3 métodos e 3 atributos. A seguir, instancie um objeto desta classe e execute os seus métodos.



# PHP 7

A versão mais atual do PHP, na data de criação deste minicurso, é o PHP 7. Com ele, diversas novidades foram introduzidas, bem como, algumas correções de funcionalidades anteriormente adotadas.

Destacam-se algumas delas a seguir, levando em conta nossa contextualização.

# PHP 7

- Remoção das funções `mysql_*` ;
- Melhoria ao suporte de tratamento de erros e exceções;
- Suporte à especificação de tipos de retorno;
- Novo formato para construtores (`__construct()`);
- Melhorias em desempenho;
- E diversas outras, como novos operadores, recursos de tipagem e por aí vai...



```
new PDO($db);
```



```
mysql_connect(  
$server, $user, $pass);
```

# Como PHP trata múltiplas requisições?

A esta altura, havendo entendido os conceitos anteriores, você pode se questionar o seguinte (ou não, talvez questione agora...):

“Como o PHP faz o processamento do programa, para cada uma das requisições do cliente?”

A resposta (em termos MUITO simplificados):

“Isso é problema para o servidor...”

# Revisão: Banco de Dados & SQL



# Definições

Passando agora a revisar conceitos sobre banco de dados (BD).

Atualmente, para se operar um banco de dados, geralmente se faz uso de **um conjunto de softwares para gerenciá-lo**, denominado Sistema Gerenciador de Banco de Dados (SGBD).

Um SGBD é composto por:

- Linguagem de definição de dados (DDL): Definir a estrutura;
- Linguagem de manipulação de dados (DML): Gerenciar conteúdo;
- Data dictionary: Descrição de conteúdo;
- Subsistemas específicos: Partes específicas para cada tipo de SGBD;

# Definições

Existem vários tipos de SGBD, porém, para este minicurso será dado foco nos SGBDs relacionais. Estes são ditos assim por empregarem o modelo de dados relacional, o qual se baseia na **representação e modelagem dos dados na forma de relações**.

Você ainda irá aprender muito mais sobre este modelo, bem como outros, ao cursar a disciplina “Banco de Dados” (caso já tenha cursado, melhor ainda!). Ainda, caso se interesse pelo assunto, vale checar a seção de aprofundamento, que oferece um conteúdo interessante.

# Operações

Por simplificação, este minicurso não envolverá modelagem de dados (no entanto, recomenda-se quase que obrigatoriamente que busque desenvolver seus conhecimentos nisto depois!). Neste momento, vamos verificar o conceito das operações básicas sobre manipulação de dados.

Também conhecidas pelo acrônimo **CRUD** (Create, Read, Update e Delete), são ditas como as “operações fundamentais” da persistência de dados.

# Operações

- Create: Cria um novo registro no BD;
- Read: Busca e leitura de registros;
- Update: Alteração de um registro preexistente;
- Delete: Remoção de registros;

Todas estas operações podem ser definidas, em um BD funcional, através da Standard Query Language (SQL). SQL se refere à uma linguagem de domínio para SGBDs, bem como, a padronização que estas mesmas seguem. SQLite implementa **a maior parte da padronização** necessária para um sistema de banco de dados (SQL).

# SQLite

Grande parte dos dialetos SQL define comandos para a criação da base de dados. Tais comandos devem ser executados em vias de gerar a base de dados, que daí poderá ser estruturada usando a DDL.

Em SQLite, no entanto, **tais comandos não são necessários (autocontenção)**. O que ocorre é que, como já citado, **cada arquivo é responsável por uma base de dados**. Ou seja, para se criar uma base de dados utilizável, basta termos um arquivo na extensão “.db” (ou “.sqlite”).

# SQLite

A DDL em SQLite pode ser apresentada em termos dos comandos “CREATE TABLE”, “ALTER TABLE” e “DROP TABLE”. Sua sintaxe é como segue:

```
/*UM COMENTARIO:  
TABELA PLAYLIST: REPRESENTA UMA  
SELECAO HIPOTETICA DE MUSICAS PARA UM  
APP IGUALMENTE HIPOTETICO*/  
  
DROP TABLE IF EXISTS playlist; /*REMOVE CASO EXISTA*//*SUPER TRUQUE*/  
CREATE TABLE playlist ( /*DEFINE UMA TABELA*/  
  
    identificador_musica INT NOT NULL PRIMARY KEY, /*ISTO EH UM ATRIBUTO*/  
    nome_musica TEXT NOT NULL, /*OUTRO ATRIBUTO*/  
    album TEXT,  
    local_arquivo TEXT NOT NULL,  
  
); /*PONTO E VIRGULA NO FINAL DE CADA COMANDO*/  
  
/*PARECE QUE ESQUECEMOS O ARTISTA, NAO?*/  
/*SEM PROBLEMA! PRA ISSO, ALTERAMOS...*/  
ALTER TABLE playlist ADD COLUMN autor TEXT;  
/*EM SQLITE, NO ENTANTO, RECOMENDA SE EVITAR O USO DE ALTER*/
```

# SQLite

As tabelas (relações) nos SGBDs relacionais são compostas por linhas e colunas. Cada linha, um registro distinto e cada coluna, um atributo distinto. É necessário, no entanto a capacidade de **distinguirmos unicamente cada um dos elementos** (pleonasma é obrigatório). Para isto, há o conceito de chave primária.

Uma chave primária trata-se de um atributo específico, distinto para cada registro (linha) e que possa assim ser mantido.

SQLite nos permite identificar chaves primárias utilizando a designação **“PRIMARY KEY”**. Esta designação geralmente também é acompanhada por outra, **“NOT NULL”**, que por sua vez significa que tal elemento não pode assumir “valor vazio”.

# SQLite

Já se tratando da DML, teremos ela na forma:

```
/*CREATE: PODEMOS UTILIZAR O OPERADOR INSERT PARA INSERIR NO BD*/  
INSERT INTO playlist (identificador_musica, nome_musica, local_arquivo, autor) VALUES (1, "Hunting High and  
Low", "/home/user/songs", "Aha");  
INSERT INTO playlist (identificador_musica, nome_musica, local_arquivo, autor) VALUES (2, "Maneater",  
"/home/user/songs", "Hal & Oates");  
INSERT INTO playlist (identificador_musica, nome_musica, local_arquivo, autor) VALUES (3, "Hold The Line",  
"/home/user/desktop/files", "Toto");  
INSERT INTO playlist (identificador_musica, nome_musica, local_arquivo, autor) VALUES (4, "Africa",  
"/home/user/songs", "Toto");
```



# SQLite

```
/*READ: UTILIZAMOS O OPERADOR SELECT PARA OBTER ITENS DO BD*/  
SELECT * FROM playlist; /*SELECIONA E OBTEM TODOS OS ITENS DA TABELA playlist*/  
SELECT nome_musica, autor FROM playlist; /*SELECIONA TODOS OS ITENS DAS COLUNAS ESPECIFICADAS DE PLAYLIST*/  
SELECT local_arquivo FROM playlist WHERE autor = "Toto"; /*TODOS OS ITENS DE local_arquivo DAS LINHAS CUJO autor  
= "Toto"*/
```

```
/*UPDATE: UTILIZAMOS O OPERADOR HOMONIMO PARA ALTERACAO DE VALORES DOS ITENS DO BD*/  
UPDATE playlist SET autor = "A-ha" WHERE identificador_musica = 1; /*ALTERA O autor DO ITEM COM id = 1*/  
/*ATENCAO EXTRA COM A CLAUSULA WHERE!!!*/
```

```
/*DELETE UTILIZAMOS O OPERADOR HOMONIMO PARA DELECAO DE ITENS DO BD*/  
DELETE FROM playlist WHERE identificador_musica = 4; /*DELETA O ITEM COM identificador 4*/  
/*NOVAMENTE, ATENCAO EXTRA COM WHERE*/  
DELETE FROM playlist WHERE identificador_musica;
```

# SQLite

A respeito dos tipos de dados suportados pelo SQLite:

- NULL
- INTEGER
- REAL
- TEXT
- BLOB



Para aqueles que já tem algum contato, podem notar que tipos como Date ou Boolean, comuns em outros SGBDs, não se apresentam. No entanto, vários destes tipos ausentes podem ser facilmente representados em SQLite.

Outros tipos, quando definidos, são mapeados à estes suportados diretamente pelo SGBD.

# Exercício #2

Baseado na estrutura do BD citado como exemplo, elabore os comandos para os seguintes ações em SQLite:

- INSERIR na playlist a música de título “Then Bones” (local: /home/user/existencialismo , autor: Alice in Chains);
- SELECIONAR todas as músicas onde o autor não esteja especificado (autor = “”);
- SELECIONAR apenas os nomes dos autores;

Não é necessário executar estes comandos em um primeiro momento (pode fazer isto, se quiser...).

# Relacionamentos entre tabelas

Um relacionamento entre tabelas é definido como uma associação entre campos das tabelas. Isto vêm a ser um dos pontos mais fortes dos SGBDs que seguem o modelo relacional.

Para se **armazenar dados de entidades distintas porém relacionadas**, os mesmos são geralmente distribuídos entre múltiplas tabelas, podendo ser “anexados” através de operações específicas.

Uma das formas mais comuns de se anexar é através das operações de JOIN (junção).

# Relacionamentos entre tabelas

```
DROP TABLE IF EXISTS autor;
```

```
CREATE TABLE autor(  
  id_autor INTEGER NOT NULL PRIMARY KEY,  
  nome TEXT NOT NULL,  
  info TEXT  
);
```

```
SELECT * FROM playlist INNER JOIN autor ON playlist.autor = autor.nome ORDER BY autor.nome;
```

# Relacionamentos entre tabelas

Formalmente, um relacionamento entre tabelas pode ser representado entre atributos chave, onde em um lado da relação se representam através de chaves primárias e do outro, **representações destas denominadas chaves estrangeiras** (Foreign Keys).

Em SQLite, chaves estrangeiras precisam ser habilitadas antes do uso. Quando habilitada, esta designação reforça o relacionamento através da obrigatoriedade (Leia com mais calma, se não entender de primeira!).

Uma **recomendação pessoal**, no entanto, seria a de limitar o uso de Foreign Keys em SQLite somente a casos extremamente necessários. Para todos os outros, use comparações entre campos “comuns”.

# Exercício #3

Elabore a estrutura de uma tabela “Album”, a ser incluída na aplicação de músicas (do exemplo). Elabore sua DDL (considerando as colunas id\_album, ano e autor) e em seguida:

- Defina as modificações necessárias nas outras tabelas (o que deve mudar em Musica?);
- Elabore os comandos para a seguinte consulta: SELECIONAR todas as músicas que existam em um álbum do ano de 1995;

Novamente, não é necessário execução dos comandos, apenas sua definição.

# Linha de comando

Para trabalharmos com SQLite, podemos fazer o acesso ao SGBD diretamente através da linha de comando (Linux). Para isto, em uma instalação comum:

```
$ sqlite3
```

Um “ardil” interessante, e que certamente vai lhe ser útil: Para carregar arquivos com comandos SQL inteiros para a geração de um arquivo BD em SQLite (para a estruturação do BD, por exemplo), pode-se utilizar do operador de redirecionamento “<” do shell Linux:

```
$ sqlite3 banco.db < arquivo_de_comandos.sql
```



# Boas práticas

Dentre as boas práticas em SQLite, destacamos o uso de nomes simples, diretos e que expressem bem o que o elemento nomeado representa. Por exemplo, ao invés de “identificador\_musica”, do exemplo, um nome mais ideal seria “id\_musica”. Mantém o sentido e é mais direto.

Além disso, o SQLite proporciona uma facilidade no que tange elementos incrementais (ids, por exemplo, geralmente seguem uma contagem). Através da cláusula AUTOINCREMENT, o próprio SGBD irá tratar de tais incrementos. Em versões recentes, no entanto, esta atribuição já vem como padrão ao SQLite, não necessitando a especificação da cláusula.

PDO

# PHP Data Objects

Acesso e operações à banco de dados são um dos pontos fortes do PHP. Anteriormente, eram usadas funções como as `mysql_*` para isto. Apesar de operacionais, as mesmas eram carentes em suporte à diversos recursos. Com a chegada do PHP 5 no entanto, vieram também os chamados PHP Data Objects, ou PDOs.

Os **PDOs** tratam-se de uma extensão que promove ao PHP acesso aos mais diversos tipos de BDs, através de uma interface unificada. Podem também ser entendidas como uma camada funcional de abstração para acesso à BDs.

# PHP Data Objects

Dentre as vantagens conferidas pelo uso dos PDOs, destacam-se:

- Segurança;
- Abstração de conexão e interações diretas;
- Interface unificada para diversos drivers (Lista completa em: <https://www.php.net/manual/en/pdo.drivers.php>);
- Simplificação de tarefas;

Apresentaremos a seguir as formas de utilização do PDO, além de um sistema-exemplo hipotético (abertura de chamadas).

# Utilização do PDO

O PDO se apresenta na forma de uma classe PHP. Em primeiro lugar, para se utilizar do acesso ao BD, deve-se criar uma conexão com o mesmo.

As conexões se manterão ativas enquanto o objeto PDO existir no programa. Para se fechar uma conexão, deve-se destruir o objeto. Não se preocupe com isto, no entanto, pois se isto não for feito explicitamente, o próprio PHP termina a conexão quando o script chega ao fim.

Uma **conexão pode ser criada** como segue:

```
try{ //Tratamento de exceptions

    $this->conn = new PDO($this->db); //Utiliza o endereço do arquivo

}catch(PDOException $e){//Em caso de exceção...
    //Relata o erro e finaliza a execução
    echo $e->getMessage();
    die("ERRO");
}
```

# Utilização do PDO

Para se executar comandos SQL através do PDO, a maneira preferível de operar é através dos chamados “prepared statements”.

```
$db = new Database(); //Instanciando um novo objeto Database  
  
//$db é um objeto da classe Database, demonstrada a seguir  
  
$stmt = $db->conn->prepare("SELECT * FROM chamada");  
$stmt->execute();  
  
$chamadas = $stmt->fetchAll(PDO::FETCH_OBJ);
```

# Utilização do PDO

O sistema de abertura de chamadas (definido em 3 arquivos):

```
DROP TABLE IF EXISTS chamada;
CREATE TABLE chamada(

    id_chamada INTEGER NOT NULL PRIMARY KEY,
    id_usuario INT,
    titulo TEXT NOT NULL,
    descricao TEXT NOT NULL,
    resolvida INT DEFAULT 0

);

INSERT INTO chamada (titulo, descricao) VALUES ("Procurar carta na bolsa", "Porque quero evitar a fadiga");
INSERT INTO chamada (titulo, descricao) VALUES ("Encontrar o rato", "Nada de exaltações!");
```



```
<?php
```

```
/*A CLASSE DEFINIDA A SEGUIR É SOMENTE PARA ESTUDOS, A MESMA NÃO É RECOMENDÁVEL PARA USO EM DESENVOLVIMENTO, TAMPOUCO EM PRODUÇÃO, POR CONTER DIVERSAS INCONSISTÊNCIAS E FUGAS À NORMALIDADE. SERVE APENAS COMO UM PROTÓTIPO PARA REFERÊNCIAS*/
```

```
class Database{ //Uma classe específica para trato do BD  
    /*É ESSENCIAL MANTERMOS OS COMPONENTES DE UM SISTEMA INDEPENDENTES ENTRE SI. UMA DAS RAZÕES PARA ESPECIFICARMOS UMA CLASSE AQUI.*/  
    private $db = "sqlite:dbtaskcalls.db"; /*Parâmetro para criação do PDO, representando o arquivo do BD SQLite*/  
    public $conn = NULL; //Para armazenar o PDO  
  
    public function __construct(){  
        try{ //Tratamento de exceptions  
            $this->conn = new PDO($this->db); //Utiliza o endereço do arquivo  
        }catch(PDOException $e){  
            //Relata o erro e finaliza a execução  
            echo $e->getMessage();  
            die("ERRO");  
        }  
    }  
}  
  
?>
```

```
<!DOCTYPE html>
<html lang="en">
<?php

    /*A KEYWORD REQUIRE_ONCE SERÁ APRESENTADA AQUI*/
    require_once "Database.php"; //Carrega de outro arquivo PHP
    $db = new Database(); //Instanciando um novo objeto Database

    if($_SERVER["REQUEST_METHOD"] == "POST"){
        //var_dump($_POST); //DESCOMENTE PARA VER O REQUEST
        $_POST = filter_input_array(INPUT_POST, FILTER_SANITIZE_STRING); //LIMPEZA

        $params = [
            "titulo_call" => $_POST["titulo"],
            "descricao_call" => $_POST["descricao"]
        ];

        $insert_stmt = $db->conn->prepare("INSERT INTO chamada (titulo, descricao) VALUES (:titulo_call,
:descricao_call)");
        $insert_stmt->execute($params);

    }

    $stmt = $db->conn->prepare("SELECT * FROM chamada");
    $stmt->execute();

    $chamadas = $stmt->fetchAll(PDO::FETCH_OBJ);

?>
```

```
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
</head>
<body>
  <div class="">
    <ul>
      <?php foreach ($chamadas as $chamada):?>

        <li>
          <h3><?php echo $chamada->titulo; ?></h3>
          <h6><?php echo $chamada->descricao; ?></h6>
        </li>

      <?php endforeach;?>
    </ul>
  </div>
  <form action="taskcalls.php" method="post">
    <label for="titulo">Título</label>
    <input type="text" name="titulo" id="">
    <label for="descricao">Descrição</label>
    <input type="text" name="descricao" id="">
    <input type="submit" value="Enviar">
  </form>
</body>
</html>
```

# Utilização do PDO

Os prepared statements são uma facilidade providenciada pelo PHP para a construção de comandos SQL efetivos e seguros. Sempre procure utilizá-los.

Podemos também fazer a execução direta de comandos (que não é recomendada, por introduzir problemas de segurança). Para tanto, basta utilizar o método:

**`$pdo_object->exec($query);`**

Por viabilidade, não trabalharemos desta forma na duração do minicurso.

# Utilização do PDO

Questão: Como poderia ser implementado a funcionalidade de marcar uma chamada como resolvida?

Questão: E a respeito da deleção? Elabore (não se sinta compelido a implementar, no entanto...) a funcionalidade de deleção de chamadas.

# Exercício #4

A separação de códigos por funcionalidade é um princípio que rege as mais elementares noções de boas práticas em programação.

Altere o código do sistema-exemplo, a fim de separar a lógica de “criação de chamadas” da “exibição de chamadas”. Transfira a lógica de criação para um arquivo próprio.

Dicas: Atenção com o nome dos arquivos. Para adiantar seu trabalho, a linha de código a seguir serve para redirecionar de volta à página inicial de chamadas:

```
header("Location: "."taskcalls.php", TRUE, 301);
```

# PHP e Requisições

Dentre as facilidades conferidas pelo PHP, temos a forma como a linguagem permite o tratamento de requisições HTTP. Novamente, isto pelo fato de ser voltada à web.

Para obter as informações da requisição que levou à execução daquele script, basta acessar os arrays:

- **`$_GET`**
- **`$_POST`**

Há também diversas outras formas de acesso...

# Exercício #5

Elabore o código para “marcar uma chamada como resolvida”. Para isto, basta atualizar o valor de “resolvida” na tabela do exemplo para o valor 1.

Dicas: A maneira mais simples de implementar isto no momento é através de um script próprio para esta funcionalidade. Utilize a passagem de parâmetros através de requisição GET para tanto...



# Boas práticas

Se tratando do PDO, é uma boa prática a utilização de classes para a construção de sua aplicação.

Como já foi dito, SEMPRE opte por utilizar os prepared statements.

Outro fator a ser realizado é que o ideal é armazenarmos constantes necessárias à configuração (como o nome do BD por exemplo) em locais à parte do código ou, idealmente, em variáveis do sistema. Isto dá ao código uma menor dependência.

# INTERMISSION



# Prática



# Recuperando e exibindo uma lista

Talvez uma das tarefas mais costumeiras realizadas com conexões à bancos de dados seja a recuperação de conjuntos de dados relacionados, para serem exibidos em listas.

Uma lista pode ser facilmente exibida através da estruturação de uma página HTML, através das tags `<ul>`, `<ol>` e `<li>`.

Considerando a DDL a seguir, vamos implementar juntos uma forma de exibição em lista dos itens da tabela.

# Recuperando e exibindo uma lista

```
DROP TABLE IF EXISTS amigo;
```

```
CREATE TABLE amigo( id_amigo INTEGER NOT NULL PRIMARY KEY, nome TEXT NOT NULL,  
idade INT NOT NULL, grupo TEXT );
```

```
INSERT INTO amigo (nome, idade, grupo) VALUES ("Alan", 20, "Computeiros");
```

```
INSERT INTO amigo (nome, idade, grupo) VALUES ("Neumann", 19, "Computeiros");
```

```
INSERT INTO amigo (nome, idade, grupo) VALUES ("Ada", 22, "Computeiros");
```

```
INSERT INTO amigo (nome, idade, grupo) VALUES ("Tim", 18, "Computeiros");
```

# Botão “remove”

Outro aspecto comum de existir na maioria das interfaces são botões juntos à itens (especialmente em listas), para efetuar ações comuns àqueles itens. Edição, Incremento de valores, Marcação e Deleção são alguns exemplos.

Continuando o exemplo da lista anterior vamos elaborar a remoção de um item da tabela, através de um link (ou um botão, por exemplo) na própria tabela.

Considere o fato que o “id” do usuário já nos é acessível pela lista gerada (logo, uma maneira de alcançar seria construir uma requisição com tal id).

# Barra de Navegação

Construa a seguir, uma barra de navegação contendo links para acesso às paginas que foram desenvolvidas anteriormente.

Isto é bem simples, bastando utilizar HTML...

# Projeto: Bebolândia Adega e Cervejaria





# Projeto: Bebolândia Adega e Cervejaria

Nesta parte do minicurso, passaremos à um projeto prático, a fim de fazer uso dos conceitos apresentados, aplicando-os em uma situação “mais real”.



“Às 11:30 da noite de um dia de sábado, você está relaxando em sua casa e eis que um antigo colega lhe manda uma mensagem a respeito de seu novo empreendimento, a **Bebolândia**, um estabelecimento para venda de **bebidas dos mais variados tipos**. Ele informa no entanto, que irá abrir sua loja na próxima semana e não possui um website. Ele lhe oferece um desconto especial válido por 1 ano para a Bebolândia, em troca de um protótipo de um site, com as especificações: ”

# Projeto: Bebolândia Adega e Cervejaria

- Exibição de produtos;
- Cadastro de produtos;
- Cadastro de usuários;
- Navegação entre páginas;
- Exibição de informações acerca do estabelecimento;

Por se tratar de um protótipo, não se preocupe com funcionalidades que não as citadas! Por exemplo, obviamente um sistema de login seria necessário (tanto pra usuários comuns quanto para administradores), porém, por conta do tempo, não será necessária sua implementação.

Também não se preocupe com aspectos estéticos...



# Projeto: Bebolândia Adega e Cervejaria

Vamos desenvolver passo a passo este sistema.

A começar por uma modelagem elementar, porém mais elaborada:

- Pensando à nível de persistência, o que teríamos de armazenar no BD?
- Analisando os requerimentos, seriam:
  - Produtos;
  - Usuários;
  - ~~Info. do estabelecimento;~~ (Não no momento, é um protótipo, certo?)

Modele então, diretamente em SQL, as tabelas “Produto” e “Usuário”. **[Tempo]**

# Projeto: Bebolândia Adega e Cervejaria

Agora, vamos ao PHP em si. Podemos neste exemplo, partir diretamente para a conexão com o BD. Em primeiro lugar, gere o mesmo com os comandos DDL anteriores (arquivo do SQLite). **[Tempo]**

Por boa prática, vamos criar uma classe específica para o trato da conexão com o BD (lembre-se que em uma situação real, isto seria um tanto quanto diferente...). Utilize do PDO para isto! **[Tempo]**

Considere ainda preparar alguns dados de teste, à serem inseridos no BD.

# Projeto: Bebolândia Adega e Cervejaria

Por simplicidade, iremos manter juntos parte da lógica com a apresentação.

- Para o cadastro de produtos, utilizaremos um prepared statement “INSERT”, que deverá obter os parâmetros da inserção por uma requisição do tipo POST. O mesmo para cadastro de usuários.
- Já para a exibição dos produtos, utilizaremos um prepared statement “SELECT”.

Note que, apesar de implementarmos a lógica e a apresentação juntas, as páginas **DEVEM ser independentes umas das outras...**

# Projeto: Bebolândia Adega e Cervejaria

A navegação não precisa ser complexa. Links já bastam...

Na apresentação, será necessário iterarmos sobre uma lista contendo cada um dos itens que recuperarmos da base. Podemos utilizar listas em HTML para a estruturação aqui. **[Tempo]**

# Tópicos Finais

# Comparativo de tecnologias

Além do “combo” PHP e SQLite, diversas outras tecnologias podem vir à ser utilizada no desenvolvimento web, cada qual com seus respectivos pontos fortes...

Note que este **comparativo não será enviesado** para PHP e SQLite. Em vez disso, a ideia é uma observação quanto ao **panorama geral** das tecnologias backend em comparação à tal “combo”. Para tanto, a análise a seguir foi baseada em diversos artigos distintos.

Aqui também vale a máxima que “cada ferramenta possui seu propósito”.



# Comparativo de tecnologias

	PHP + SQLite	PHP + MySQL	Express + MongoDB
<b>Vantagens:</b>	Simplificação e Capacidade;	Formalização, Plataforma e mercado;	Ubiquidade;
<b>Desvantagem principal:</b>	Permissividade e manutenção;	Requerimentos de ambiente e manutenção;	Curva de aprendizado;
<b>Mercado (BR):</b>	Sem informações (atente ao MySQL)	Alto	Médio

# Comparativo de tecnologias

	Rails + MySQL	Java + MySQL	.NET + SQL Server
<b>Vantagens:</b>	Simplificação;	Formalização e mercado;	Simplificação e integração com tecnologias específicas;
<b>Desvantagem principal:</b>	Performance;	Curva de Aprendizado;	Plataforma fraca e financeiramente custosa;
<b>Mercado (BR):</b>	Baixo	Alto	Médio

# O que estudar a seguir?

Após finalizar este minicurso, tendo compreendido os conceitos apresentados e formalizado conhecimento, o aluno poderá optar por estudar alguns outros tópicos interessantes, bem como, revisar os conceitos deste minicurso.

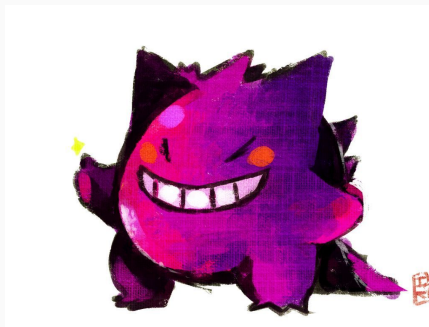
- Padrão DAO em PHP;
- SQLite com maior detalhamento;
- MySQL;
- Projetos pessoais com a linguagem. **Criar um Blog**, por exemplo...

Enfim, estes são alguns exemplos. Dependerá do aluno decidir o que melhor se encaixará para ele. Apenas tenha atenção com aquilo que decide!

# Conclusão

Chegamos ao fim deste minicurso! Para você que se esforçou e chegou até aqui, meus parabéns!

Pessoalmente, espero que possa aproveitar estes novos conhecimentos!



# Aprofundamentos

- Projeto: Crie seu próprio blog “from scratch”, com PHP e SQLite:  
<https://ilovephp.jondh.me.uk/en/tutorial/make-your-own-blog/introduction>
- Como PHP opera “por trás da cortina”:  
<https://www.sitepoint.com/how-php-executes-from-source-code-to-render/>
- Talk efetuada pelo criador da linguagem, com um apanhado geral sobre os 25 anos da mesma (Legendado): <https://www.youtube.com/watch?v=wCZ5TJCBWMg>
- Blog com uma referência completa sobre SQLite: <https://www.sqlitetutorial.net/>
- Livros disponibilizados gratuitamente (Open Source):  
<https://daylerees.com/books/>

# Aprofundamentos

Blog da comunidade PHP Brasil (Medium): <https://medium.com/php-brasil>

Comunidade de compartilhamento de dúvidas no reddit:

<https://www.reddit.com/r/PHPhelp/>

## Referências:

- Livro: Learning PHP, MySQL, JavaScript, CSS & HTML5 - Robin Nixon
- Livro: Using SQLite - J. A. Kreibich
- <https://www.php.net/>
- <https://phptherightway.com/>
- <https://www.sqlite.org/index.html>

# Fim

Por sua amável atenção, meus agradecimentos...