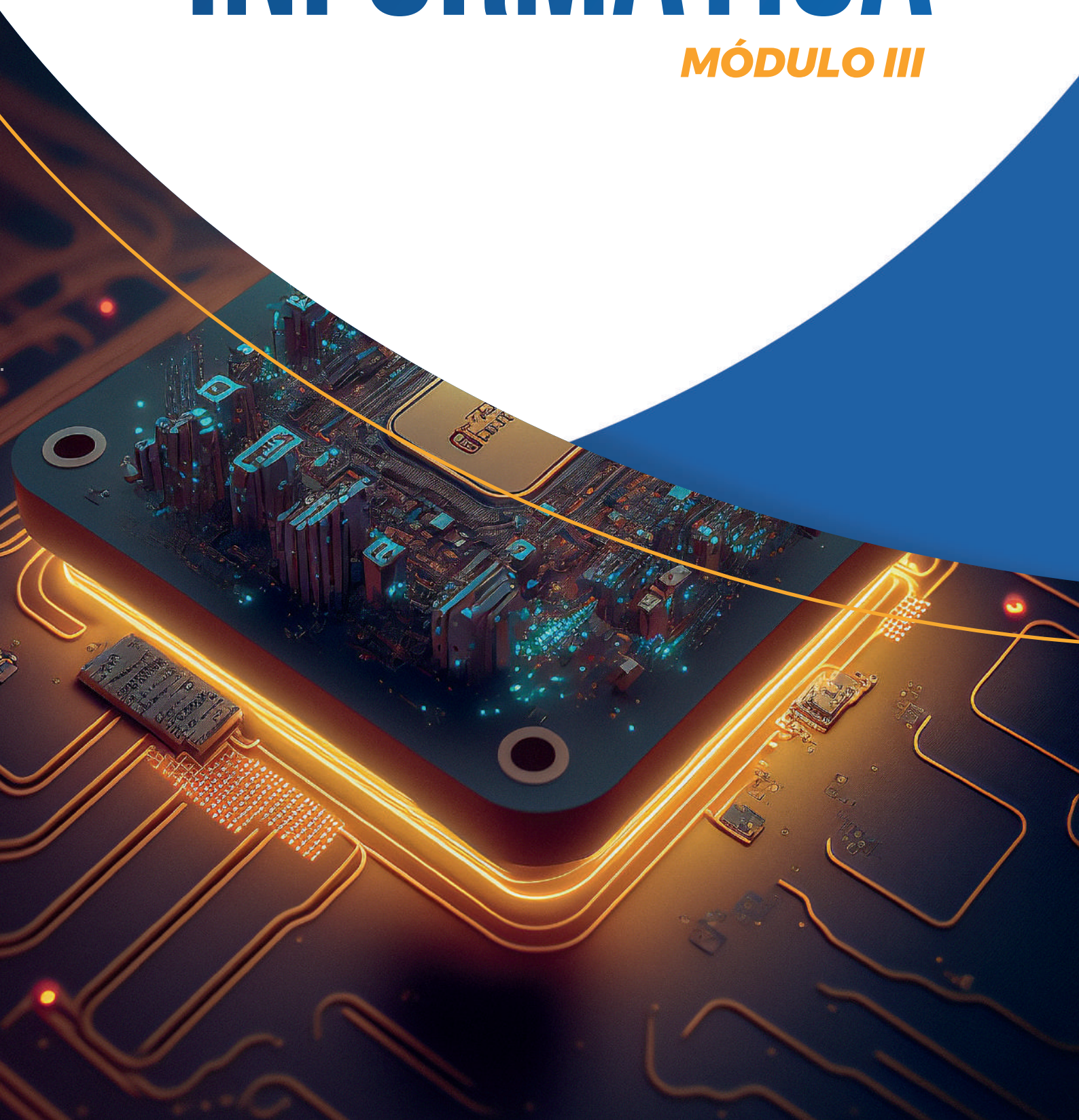
CURSO TÉCNICO EM

INFORMÁTICA

MÓDULO III



Rafael Tajra Fonteles
Governador do Estado

Francisco Washington Bandeira Santos Filho
Secretário de Estado da Educação

Paulo Henrique da Costa Pinheiro
Superintendente de Educação Técnica e Profissional e Educação de Jovens e Adultos

Viviane Holanda Barros Carvalhedo
Superintendência de Gestão da Educação Básica e Superior - SUGED

Adriana Moura Silva
Diretora da Unidade de Educação Profissional

Karoline Mendes de Sousa
Diretora de Mediação Tecnológica (UEMTEC)

Leila Coelho Pinto Leite
Coordenadora de Apoio Pedagógico - Concomitante

Escola Estadual com muito orgulho

SEDUC TEC

Canal Educação

SECRETARIA DA EDUCAÇÃO - SEDUC

GOVERNO DO PIAUÍ
AQUI TEM TRABALHO.
AQUI TEM FUTURO.

CURSO TÉCNICO EM

INFORMÁTICA

MÓDULO III

Banco de Dados

Amanda Souza

Apostila de Banco de Dados

1. Introdução aos bancos de dados

Fonte: pixabay

Bancos de dados são uma parte fundamental da tecnologia da informação e são utilizados em diferentes tipos de sistemas para armazenar, gerenciar e recuperar informações. Eles são amplamente utilizados em sistemas empresariais, aplicativos web e muitos outros tipos de software.

Um banco de dados é um conjunto estruturado de informações organizadas e armazenadas em um computador. Ele pode ser utilizado por sistemas de software para armazenar, gerenciar e recuperar informações. Os bancos de dados podem ser classificados em diferentes tipos, dependendo do modelo de dados, como os bancos de dados relacionais, orientados a objetos, hierárquicos ou em rede.

Os bancos de dados relacionais são os mais comuns e usam tabelas e chaves para organizar as informações. Eles são amplamente utilizados em sistemas de gestão empresarial (ERP), sistemas de gerenciamento de conteúdo (CMS) e aplicativos web. Os bancos de dados também possuem linguagens específicas para consultar e manipular seus dados, como SQL (Structured Query Language) para bancos de dados relacionais.

A segurança dos bancos de dados é extremamente importante porque eles geralmente contêm informações sensíveis e confidenciais. Portanto, é necessário ter proteções adequadas contra ataques cibernéticos e gerenciamento de acesso aos dados.

Os bancos de dados são importantes para as empresas porque permitem que elas gerenciem grandes quantidades de dados de maneira eficiente, economizando tempo e recursos. Eles também permitem que as empresas tomem decisões informadas com base nas informações armazenadas nos bancos de dados.

Os bancos de dados são essenciais para os sistemas de informação modernos e permitem que as empresas armazenem, gerenciem e recuperem grandes quantidades de dados de maneira eficiente. Se você está começando a aprender sobre bancos de dados, este é um bom lugar para começar. O conhecimento básico sobre o assunto pode ajudá-lo a entender melhor como as informações são armazenadas e gerenciadas em diferentes sistemas de software.

1. O que são Banco de Dados?

Fonte: pixabay

Bancos de dados são conjuntos organizados de informações armazenadas em um computador ou servidor. Eles permitem que as empresas e organizações gerenciem grandes quantidades de dados de maneira eficiente, economizando tempo e recursos. Os bancos de dados podem ser usados para armazenar informações como nomes de clientes, produtos, detalhes de transações, entre outros dados importantes.

Segundo a Oracle, um banco de dados é uma coleção organizada de dados que pode ser facilmente acessada, gerenciada e atualizada. Ele permite que os usuários organizem, armazenem e recuperem informações de forma eficiente. Existem diferentes tipos de bancos de dados, incluindo bancos de dados relacionais, orientados a objeto, hierárquicos e em rede.

Os bancos de dados relacionais são os mais comuns e utilizam tabelas e chaves para organizar as informações. Eles são amplamente utilizados em sistemas de gestão empresarial (ERP), sistemas de gerenciamento de conteúdo (CMS) e aplicativos web. Os bancos de dados também possuem linguagens específicas para consultar e manipular seus dados, como SQL (Structured Query Language) para bancos de dados relacionais.

A segurança dos bancos de dados é extremamente importante porque eles geralmente contêm informações sensíveis e confidenciais. Por isso, medidas

adequadas devem ser tomadas para proteger os dados contra ataques cibernéticos e gerenciar o acesso aos dados de forma segura.

2. Como os bancos de dados surgiram

Fonte: pixabay

A história dos bancos de dados remonta aos anos 1960, quando a necessidade de armazenar grandes quantidades de informações em sistemas computacionais começou a surgir. Naquela época, o armazenamento de dados era feito em fitas magnéticas e os arquivos eram acessados através de programas de software.

De acordo com a IBM, um dos primeiros exemplos de uso de banco de dados foi o Sistema Integrado de Processamento de Dados (SIPD), que foi criado para o Departamento de Defesa dos Estados Unidos em meados dos anos 1960. O SIPD utilizava um modelo hierárquico para armazenar informações.

Nos anos 1970, surgiram os primeiros sistemas de gerenciamento de bancos de dados, como o Sistema de Gerenciamento de Banco de Dados (DBMS) IBM IMS e o Sistema de Gerenciamento de Banco de Dados em Rede (Network DBMS).

Na década de 1980, os bancos de dados relacionais começaram a ganhar popularidade. O modelo relacional, baseado em tabelas e chaves primárias e estrangeiras, foi introduzido pela primeira vez pelo cientista da computação Edgar F. Codd em 1970, mas só foi amplamente adotado na década de 1980. O modelo relacional tornou-se rapidamente o padrão da indústria e é ainda hoje o modelo mais utilizado.

Com o surgimento da Internet, a necessidade de bancos de dados aumentou ainda mais. Os bancos de dados se tornaram fundamentais para o desenvolvimento de aplicativos web e sistemas de gerenciamento de conteúdo. Hoje, os bancos de dados são uma parte essencial do mundo tecnológico,

auxiliando sistemas empresariais, aplicações móveis e muitos outros tipos de software.

1.3. Qual a importância dos bancos de dados?

Os bancos de dados são extremamente importantes para empresas e organizações, pois permitem que elas gerenciem grandes quantidades de dados de maneira eficiente, economizando tempo e recursos. Eles são amplamente utilizados em sistemas de gestão empresarial (ERP), sistemas de gerenciamento de conteúdo (CMS) e aplicativos web.

Fonte: pixabay

Com o aumento constante da quantidade de dados gerados por empresas e indivíduos, a gestão adequada desses dados se tornou cada vez mais importante. Os bancos de dados oferecem muitos benefícios, como:

- **Organização e gerenciamento eficiente de grandes quantidades de dados:** os bancos de dados permitem a organização e o armazenamento de grandes volumes de dados de maneira estruturada, permitindo que as empresas possam acessar e usar esses dados de forma eficiente e confiável.
- **Tomada de decisões informadas:** os bancos de dados fornecem informações precisas e atualizadas sobre os negócios das empresas, permitindo que os líderes tomem decisões com base em dados concretos.
- **Eficiência operacional:** os bancos de dados podem ajudar a automatizar processos e tarefas, aumentando a eficiência das operações comerciais.

- **Segurança dos dados:** os bancos de dados podem ser projetados para proteger os dados contra ameaças cibernéticas e gerenciar o acesso aos dados de forma segura.
- **Escalabilidade:** os bancos de dados podem ser escalados para atender às necessidades crescentes de armazenamento e processamento de dados à medida que os negócios crescem.

2. Sistemas de gerenciamento de bancos de dados (SGBD)

Os sistemas de gerenciamento de bancos de dados (SGBDs) são programas que permitem que os usuários criem, mantenham e administrem bancos de dados. Eles fornecem uma interface para que os usuários possam trabalhar com os dados armazenados no banco de dados e permitem a criação de consultas e relatórios personalizados.

Fonte: pixabay

Existem diferentes tipos de SGBDs, cada um projetado para atender às necessidades específicas de diferentes tipos de bancos de dados. Alguns exemplos incluem:

- **SGBD relacionais:** usam tabelas e chaves para organizar as informações em bancos de dados relacionais, como MySQL, Oracle e Microsoft SQL Server.
- **SGBD orientadas a objetos:** usam objetos para armazenar informações, como o MongoDB.
- **SGBD hierárquicos:** organizam as informações em uma estrutura hierárquica, como o IBM Information Management System (IMS).
- **SGBD em rede:** permitem que os dados sejam armazenados em várias tabelas interconectadas, como o Integrated Data Store (IDS) da Informix.

Os SGBDs oferecem muitos benefícios para empresas e organizações, incluindo a capacidade de acessar e analisar grandes quantidades de dados de

maneira eficiente, garantir a segurança dos dados e permitir a colaboração entre equipes em diferentes áreas ou localizações.

2.1. O que é um SGBD

Fonte: pixabay

SGBD é a sigla para Sistema de Gerenciamento de Banco de Dados. É um software que permite aos usuários criar, manipular e gerenciar bancos de dados, além de oferecer uma interface para acessar seus dados.

Os SGBDs são projetados para ajudar as empresas e organizações a gerenciar grandes quantidades de informações de forma eficiente e segura. Eles fornecem recursos como backup e restauração de dados, controle de acesso, criptografia e auditoria de dados, entre outros.

Existem diferentes tipos de bancos de dados, cada um projetado para atender às necessidades específicas de diferentes aplicações. A seguir, estão alguns dos principais tipos de banco de dados e suas características:

Banco de dados relacional

Os bancos de dados relacionais são o tipo mais comum de banco de dados utilizado em aplicações comerciais. Eles usam tabelas para organizar as informações, com chaves primárias e estrangeiras para estabelecer relacionamentos entre elas.

Vantagens	Desvantagens
Estruturado; Escalável; Flexível; Fácil de usar e alta segurança	Pode ser difícil de manter em grandes quantidades de dados

Exemplos incluem:

- MySQL;
- Oracle e Microsoft SQL Server.

Banco de dados orientado a objetos

Os bancos de dados orientados a objetos permitem que as informações sejam armazenadas como objetos, com propriedades e métodos associados. Isso pode tornar o desenvolvimento do software mais fácil e mais rápido, especialmente para aplicações complexas que envolvem muitos objetos interconectados.

Vantagens

Estrutura semelhante a objetos do mundo real, permite armazenar e representar dados complexos

Desvantagens

Requer conhecimento avançado de programação orientada a objetos, pode ser menos eficiente que bancos de dados relacionais em certas situações

Exemplos incluem:

- MongoDB e Apache Cassandra.

Banco de dados hierárquico

Os bancos de dados hierárquicos organizam as informações em uma estrutura hierárquica, semelhante a uma árvore. Cada nó na árvore representa um registro que pode ter vários filhos e pais.

Os bancos de dados hierárquicos são úteis para modelar sistemas com muitas relações pai-filho.

Vantagens

É eficiente, garante a integridade dos dados e é relativamente mais simples de entender.

Desvantag

A rigidez da estrutura pode ser uma desvantagem para aplicações mais complexas, a navegação pelos dados pode ser difícil e o modelo é altamente dependente do hardware subjacente.

Exemplo incluem:

- O IBM Information Management System (IMS).

Banco de dados em rede

Os bancos de dados em rede permitem que os dados sejam armazenados em várias tabelas interconectadas. Os registros no banco de dados podem ter vários pais e filhos, permitindo que as relações entre eles sejam mais flexíveis do que nos bancos de dados hierárquicos.

Exemplos incluem:

- O Integrated Data Store (IDS) da Informix;
- O IDMS da CA Technologies.

Vantagem

Representação eficiente de relacionamentos complexos entre entidades;

A possibilidade de uma entidade possuir vários registros, assim como um registro pode possuir várias entidades relacionadas.

Desvantagem

- Dificuldade de entendimento em comparação com outros modelos de banco de dados, como o modelo relacional;
- Modificação da estrutura pode ser trabalhosa, pois adições ou remoções de entidades podem afetar outras partes do banco de dados, exigindo mudanças em várias áreas do sistema;
- Não é muito utilizado atualmente, o que pode dificultar a obtenção de profissionais especializados para o trabalho com esse tipo de banco de dados.

Os SGBDs (Sistemas Gerenciadores de Bancos de Dados) são organizados em categorias, pois cada uma delas possui a sua própria abordagem para armazenar, gerenciar e acessar dados. Cada categoria é adequada para um conjunto específico de requisitos de aplicativos e nem todas as categorias de SGBDs se encaixam bem em todos os cenários.

Por exemplo, o SGBD Relacional é adequado para aplicações que requerem dados estruturados e precisam garantir a consistência e integridade dos dados. Enquanto isso, o SGBD NoSQL é mais adequado para aplicações que exigem alta escalabilidade e flexibilidade na modelagem de dados, como é o caso de redes sociais, sistemas de recomendação e análise de big data.

2.2. Como escolher o melhor SGBD para cada caso

A escolha do melhor Sistema Gerenciador de Banco de Dados (SGBD) para cada caso pode ser uma tarefa complexa, pois envolve diversos fatores que precisam ser levados em consideração. Abaixo estão os principais aspectos a serem avaliados na escolha do SGBD:

Tipo de banco de dados

Fonte: pixabay

O primeiro passo na escolha do SGBD é identificar qual tipo de banco de dados será gerenciado. Existem dois tipos principais: bancos de dados relacionais (SQL) e bancos de dados NoSQL. Bancos de dados relacionais são ideais para aplicações com transações complexas e relacionamentos entre tabelas, enquanto bancos de dados NoSQL são mais adequados para aplicações com grandes volumes de dados não estruturados.

Escalabilidade

Fonte: pixabay

A escalabilidade do SGBD deve ser considerada caso haja a possibilidade de aumento significativo no volume de dados gerenciados em um futuro próximo. SGBDs escaláveis como o MongoDB e o Cassandra são ideais para aplicações que exigem alta escalabilidade horizontal, ou seja, quando é necessário adicionar novos servidores conforme o volume de dados aumenta.

Desempenho.

Fonte: pixabay

O desempenho do SGBD é outro fator importante a ser considerado. Bancos de dados SQL como o PostgreSQL e o MySQL são conhecidos pela sua alta performance, enquanto bancos de dados NoSQL, como o MongoDB, podem oferecer melhor desempenho em casos específicos, como em aplicações com grande número de leituras e escritas simultâneas.

Segurança

Fonte: pixabay

A segurança dos dados é essencial para qualquer aplicação empresarial. SGBDs como Oracle e Microsoft SQL Server são conhecidos por seus recursos avançados de segurança, enquanto bancos de dados NoSQL como o MongoDB exigem configuração cuidadosa para garantir a segurança dos dados.

Tipo de aplicação

O tipo de aplicação em que o SGBD será utilizado também deve ser levado em consideração. Por exemplo, o MySQL é frequentemente usado em aplicações web, enquanto o Oracle pode ser usado em grandes empresas que precisam gerenciar grandes volumes de dados.

Requisitos de suporte técnico

A escolha do SGBD também deve levar em consideração os requisitos de suporte técnico. Grandes empresas, como Oracle e Microsoft, oferecem suporte técnico robusto, enquanto bancos de dados de código aberto, como PostgreSQL e MySQL, geralmente contam com comunidades de usuários ativas para ajudar a solucionar problemas.

Licenciamento e custo

Finalmente, o licenciamento e o custo devem ser avaliados na escolha do SGBD. Bancos de dados de código aberto, como PostgreSQL e MySQL, são gratuitos, enquanto bancos de dados comerciais, como Oracle e Microsoft SQL Server, têm custos associados à licença e manutenção.

Modelagem de dados

Fonte: pixabay

Modelagem de dados é o processo de criar um modelo conceitual, lógico e físico de como os dados serão armazenados, organizados e acessados em um sistema de banco de dados. É uma parte fundamental do design e desenvolvimento do banco de dados, pois determina a estrutura do banco de dados e como os dados serão relacionados entre si.

O modelo conceitual descreve as principais entidades e relacionamentos do sistema, independentemente de como os dados serão armazenados fisicamente. O modelo lógico define os detalhes da estrutura do banco de dados e sua relação com as entidades do modelo conceitual. Já o modelo físico especifica como os dados serão armazenados em um determinado sistema de gerenciamento de banco de dados (SGBD), como tabelas, índices e chaves estrangeiras.

A modelagem de dados é uma parte crítica do processo de desenvolvimento de software e ajuda a garantir que os dados sejam armazenados e acessados de forma eficiente e segura. Além disso, a modelagem de dados também ajuda a garantir que os dados estejam normalizados para evitar problemas de redundância e inconsistência.

3.1. O que é a modelagem de dados?

A modelagem de dados é uma técnica utilizada para descrever e organizar os dados de um sistema de informação em um modelo que representa a estrutura e os relacionamentos entre as informações.

A modelagem de dados é uma parte importante do desenvolvimento de sistemas de informação, pois permite aos desenvolvedores entenderem os requisitos e características dos dados que precisam ser armazenados, processados e acessados pelo sistema.

Segundo Silva (2015), a modelagem de dados pode ser feita em vários níveis, incluindo o modelo conceitual, modelo lógico e modelo físico. O modelo conceitual, também conhecido como modelo de entidade-relacionamento (ER), é utilizado para representar as entidades e relacionamentos entre elas em um nível abstrato e independente de qualquer tecnologia ou ferramenta específica de banco de dados.

O modelo lógico, por sua vez, é uma representação mais detalhada da estrutura das informações em termos de tabelas, campos e relacionamentos, utilizando uma linguagem específica de modelagem, como o DER. Já o modelo físico define como as informações serão armazenadas no disco, incluindo detalhes sobre o tipo de armazenamento, índices, chaves e outras propriedades específicas do SGBD utilizado.

Conforme Elmasri e Navathe (2011), a modelagem de dados é um processo iterativo e interativo, onde os modelos vão sendo refinados à medida que novas informações e requisitos são levantados e analisados. Esse processo envolve a identificação dos requisitos de negócio, análise das informações existentes, definição do modelo conceitual, elaboração do modelo lógico e finalização com o modelo físico.

A modelagem de dados é uma parte crítica do desenvolvimento de sistemas de informação, pois permite aos desenvolvedores entenderem as informações que precisam ser armazenadas e como elas se relacionam entre si.

Essa compreensão dos dados é fundamental para garantir a consistência, integridade e eficiência do acesso às informações pelo sistema.

Essa modelagem pode ser feita em vários níveis, incluindo:

- Modelo conceitual;
- Modelo lógico;
- Modelo físico.

A modelagem de dados é uma etapa fundamental no processo de desenvolvimento de software, pois permite que os desenvolvedores capturem as informações importantes do negócio e as organizem de maneira estruturada.

A partir da modelagem de dados, é possível criar um esquema que define como os diferentes tipos de dados serão armazenados, processados e acessados pelo sistema.

A importância da modelagem de dados pode ser resumida em três pontos principais:

- **Precisão:** Ao modelar os dados, é possível capturar com precisão as informações importantes do negócio e definir como elas serão armazenadas e gerenciadas.
- **Consistência:** Uma boa modelagem de dados garante que os dados sejam consistentes e coerentes em todo o sistema, evitando problemas como redundância e inconsistência de dados.
- **Eficiência:** A modelagem de dados ajuda a otimizar o desempenho do sistema, permitindo que os desenvolvedores projetem bancos de dados eficientes e bem-estruturados que possam lidar com grandes volumes de dados e realizar consultas complexas de forma rápida e eficiente.

3.2. Elementos da modelagem de dados

A Modelagem de Dados é um processo utilizado para representar o mundo real em termos de informações estruturadas para serem armazenadas e processadas por um sistema de computador. O objetivo principal da modelagem de dados é criar um modelo conceitual do banco de dados que será implementado em um Sistema de Gerenciamento de Banco de Dados (SGBD).

Existem quatro elementos básicos na modelagem de dados:

Entidades

As entidades em uma modelagem de dados são as coisas ou objetos do mundo real que precisam ser representados no banco de dados, como clientes,

produtos, pedidos etc. Cada entidade possui atributos que descrevem suas características e propriedades.

Na modelagem de dados, entidades são objetos que representam informações sobre um determinado domínio. Essas entidades podem ser pessoas, objetos, eventos, lugares, conceitos ou qualquer outra coisa que possa ser identificada e descrita em um sistema de banco de dados.

As entidades são representadas por uma tabela no modelo relacional, onde cada linha da tabela representa uma instância da entidade e cada coluna representa um atributo da entidade. A modelagem de entidades é usada para descrever a estrutura e as características dos dados armazenados em um banco de dados.

Um exemplo simples de entidade seria uma tabela de clientes em um sistema de gerenciamento de vendas. Cada linha na tabela representa um cliente e cada coluna representa um atributo do cliente, como nome, endereço, telefone e e-mail.

Cliente ID	Nome	Endereço	Telefone	E-mail
1	João	Rua A, 123	(11) 99998888	joao@email.com
2	Maria	Avenida B, 456	(21) 77776666	maria@email.com
3	José	Travessa C, 789	(31) 55554444	jose@email.com

A seguir, apresento alguns exemplos de entidades em diferentes contextos:

- **Em um sistema de gerenciamento de escola, as entidades podem incluir:** alunos, professores, turmas, disciplinas, notas, horários etc.
- **Em um sistema de gerenciamento de vendas, as entidades podem incluir:** clientes, produtos, pedidos, faturas, estoques etc.
- **Em um sistema de gerenciamento hospitalar, as entidades podem incluir:** pacientes, médicos, enfermeiros, consultas, exames, tratamentos etc.

Esses exemplos demonstram como as entidades são fundamentais na modelagem de dados, pois permitem a organização e estruturação dos dados em um banco de dados de maneira coerente e consistente.

Outro exemplo seria uma tabela de produtos em um sistema de gerenciamento de estoque. Cada linha na tabela representa um produto e cada

coluna representa um atributo do produto, como nome, descrição, preço, quantidade em estoque etc.

Esses exemplos demonstram como as entidades são fundamentais na modelagem de dados, pois permitem a organização e estruturação dos dados em um banco de dados de maneira coerente e consistente.

Produto ID	Nome	Descrição	Preço	Quantidade em Estoque
1	Celular	Smartphone com tela de 6 polegadas	R$ 1.499,00	50
2	Notebook	Notebook com processador Intel Core i5	R$ 3.199,99	30
3	TV	TV LED de 55 polegadas	R$ 2.999,00	20

Questões para o aprofundamento do conteúdo

1. O que são entidades na modelagem de dados e qual é o seu papel em um sistema de gerenciamento de banco de dados?

   _____

   _____

   _____

2. Qual é a diferença entre uma entidade forte e uma entidade fraca na modelagem de dados? Dê exemplos de cada tipo.

   _____

   _____

   _____

3. Na modelagem de dados, como é representada uma entidade no modelo relacional? E quais são as características de uma tabela que representa uma entidade?

   _____

   _____

   _____

4. Como você identifica as entidades ao projetar um banco de dados para um sistema de gerenciamento de vendas? Quais entidades são geralmente incluídas nesse tipo de sistema?

   _____

   _____

   _____

5. Explique a importância da modelagem de entidades em um projeto de banco de dados e como isso pode ajudar a garantir a integridade dos dados armazenados. Dê exemplos práticos.

   _____

   _____

   _____

Atributos

Em modelagem de dados, atributos são características ou propriedades que descrevem uma entidade. Cada entidade tem um conjunto de atributos que a definem e distinguem de outras entidades. Por exemplo, em um banco de dados de clientes, uma entidade pode ser uma pessoa e seus atributos podem incluir nome, endereço, telefone, e-mail, entre outros.

Os atributos são representados como colunas em uma tabela de banco de dados relacional, onde cada linha representa uma instância da entidade e os valores nas colunas representam os valores dos atributos para essa instância.

Tipos de Atributos

- **Atributos Simples:** que contém apenas um valor (por exemplo, idade, preço);
- **Atributos Compostos:** que consistem em várias partes (por exemplo, endereço, com nome da rua, número, cidade, estado e código postal);
- **Atributos Multivalorados:** que permitem múltiplos valores para um único atributo (por exemplo, telefones de contato);

- **Atributos Derivados:** que são calculados a partir de outros atributos (por exemplo, idade, calculada a partir da data de nascimento).

É importante identificar corretamente os atributos de uma entidade durante o processo de modelagem de dados, pois isso ajuda a garantir que todos os dados relevantes sejam armazenados no banco de dados.

Os atributos são as características ou propriedades das entidades que precisam ser registradas no banco de dados. Por exemplo, um cliente pode ter atributos como nome, endereço, telefone etc.

Relacionamentos

Os relacionamentos definem a forma como as entidades estão relacionadas entre si. Por exemplo, um pedido está relacionado a um cliente e um produto está relacionado a um pedido. Esses relacionamentos permitem que os usuários do sistema acessem informações de diferentes entidades de forma integrada.

Na modelagem de dados, os relacionamentos representam a forma como as tabelas ou entidades em um banco de dados estão interligadas. Esses relacionamentos são definidos por meio de chaves estrangeiras que vinculam uma tabela a outra.

Tipos de relacionamentos na modelagem de dados

- **Um para Um (1:1):** Cada registro em uma tabela está relacionado a apenas um registro em outra tabela e vice-versa.
- **Um para Muitos (1:N):** Cada registro em uma tabela pode estar relacionado a vários registros em outra tabela, mas cada registro nessa outra tabela só pode estar relacionado a um registro na primeira tabela.
- **Muitos para Muitos (N:N):** Cada registro em uma tabela pode estar relacionado a vários registros em outra tabela e vice-versa.

Os relacionamentos na modelagem de dados são importantes porque permitem que os dados sejam organizados de forma eficiente e evitam a

duplicação desnecessária dos mesmos. Além disso, eles permitem que os dados sejam recuperados rapidamente e com precisão quando necessário.

Um para Um (1:1)

Nesse tipo de relacionamento, cada registro em uma tabela está relacionado a apenas um registro em outra tabela e vice-versa. Geralmente, esse tipo de relacionamento é usado quando as informações que precisam ser armazenadas em diferentes tabelas estão fortemente relacionadas e fazem mais sentido se estiverem em tabelas separadas.

EXEMPLO

Imagine que uma empresa precise armazenar informações sobre seus funcionários e sobre os carros que eles dirigem. Como cada funcionário só pode dirigir um carro, faz sentido criar duas tabelas separadas: uma para armazenar informações sobre os funcionários e outra para armazenar informações sobre os carros.

Em seguida, é criado um relacionamento 1:1 entre as duas tabelas, usando uma chave estrangeira (geralmente o ID do funcionário) em ambas as tabelas.

ATIVIDADE COMPLEMENTAR

- De acordo com o exemplo acima elabore uma modelagem de dados, com seus atributos e o relacionamento de (1:1)

Um para Muitos (1:N)

Nesse tipo de relacionamento, cada registro em uma tabela pode estar relacionado a vários registros em outra tabela, mas cada registro nesta outra tabela só pode estar relacionado a um registro na primeira tabela. Esse tipo de relacionamento é muito comum em bancos de dados, porque muitas vezes há várias informações relacionadas a um único item principal.

EXEMPLO

Imagine que uma loja precisa armazenar informações sobre seus clientes e suas compras. Cada cliente pode ter feito várias compras ao longo do tempo, mas cada compra só pode ser atribuída a um cliente. Assim, cria-se uma tabela para armazenar informações sobre os clientes e outra tabela para armazenar informações sobre as compras, e é criado um relacionamento 1:N entre as duas tabelas, usando uma chave estrangeira (geralmente o ID do cliente) na tabela de compras.

ATIVIDADE COMPLEMENTAR

- De acordo com o exemplo acima elabore uma modelagem de dados, com seus atributos e o relacionamento de (1:N)

Muitos para Muitos (N:N)

Nesse tipo de relacionamento, cada registro em uma tabela pode estar relacionado a vários registros em outra tabela e vice-versa. Isso significa que há uma relação muitos-para-muitos entre as duas tabelas. Para criar esse tipo de relacionamento, é necessário usar uma tabela intermediária que conecta as duas tabelas principais.

EXEMPLO

Imagine que uma escola precise armazenar informações sobre seus alunos e suas disciplinas. Cada aluno pode se inscrever em várias disciplinas e cada disciplina pode ter vários alunos matriculados. Nesse caso, é criada uma tabela intermediária para armazenar as informações da matrícula, que contém pelo menos duas chaves estrangeiras: uma para a tabela de alunos e outra para a tabela de disciplinas. Em seguida, é criado um relacionamento N:N entre as tabelas de alunos e disciplinas, usando a tabela intermediária como ponte para conectar os dois.

ATIVIDADE COMPLEMENTAR

- De acordo com o exemplo acima elabore uma modelagem de dados, com seus atributos e o relacionamento de (N:N)

Restrições 

São as regras que definem como as entidades podem ser manipuladas no banco de dados. As restrições garantem a integridade dos dados, evitando que informações inválidas ou inconsistentes sejam armazenadas no banco de dados.

O processo de modelagem de dados envolve a criação de um diagrama de Entidade-Relacionamento (DER), que é uma representação visual das entidades, atributos, relacionamentos e restrições do banco de dados. O DER permite que os usuários visualizem a estrutura do banco de dados de forma clara e compreensível, facilitando o trabalho dos desenvolvedores na implementação do banco de dados.

Restrições na modelagem de dados são regras que são definidas para garantir a integridade dos dados armazenados em um banco de dados. Elas podem ser definidas para garantir que determinadas regras de negócio sejam atendidas ou para evitar a inserção de dados inválidos ou inconsistentes.

Restrições na Modelagem de Dados

- **Restrições de chave primária:** Garantem que cada registro em uma tabela de banco de dados possua uma chave única que o identifique de forma exclusiva.

- **Restrições de chave estrangeira:** Garantem que os registros em uma tabela estejam relacionados corretamente com os registros em outra tabela, evitando assim inconsistências e erros de referência.
- **Restrições de integridade referencial:** Garantem que as regras de integridade referencial sejam respeitadas, impedindo a exclusão ou a atualização de registros em uma tabela que possam afetar outros registros em outras tabelas relacionadas.
- **Restrições de check:** Permitem que você defina critérios para validar dados inseridos ou atualizados em uma tabela, como por exemplo, limitar a entrada de valores numéricos entre um intervalo específico.

Em resumo, a modelagem de dados é uma etapa fundamental no desenvolvimento de software, pois permite que as informações importantes do negócio sejam organizadas em um modelo estruturado e eficiente para serem armazenadas e gerenciadas por um sistema de computador.

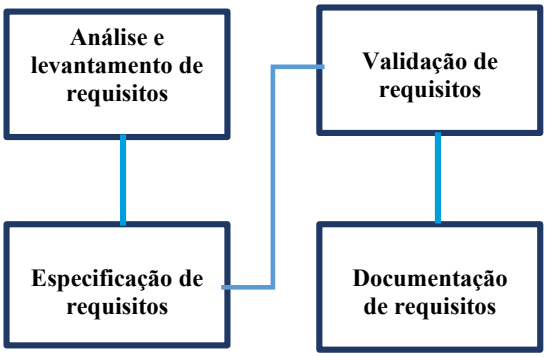
3.3. Etapas da Modelagem de Dados

As etapas da modelagem de dados são um conjunto de atividades que visam criar um modelo conceitual e lógico de como os dados serão armazenados, organizados e acessados em um sistema.

As quatro etapas da modelagem de dados

- **Requisitos;**
- **Modelagem Conceitual;**
- **Modelagem Lógica;**
- **Implementação Física**

Requisitos



Essa etapa envolve a compreensão dos requisitos do sistema e a identificação das entidades (objetos), atributos (características) e relacionamentos entre eles.

Geralmente, é feita através de entrevistas com os usuários do sistema e análise da documentação existente.

O objetivo aqui é entender o que o sistema deve fazer e como os dados são usados nele

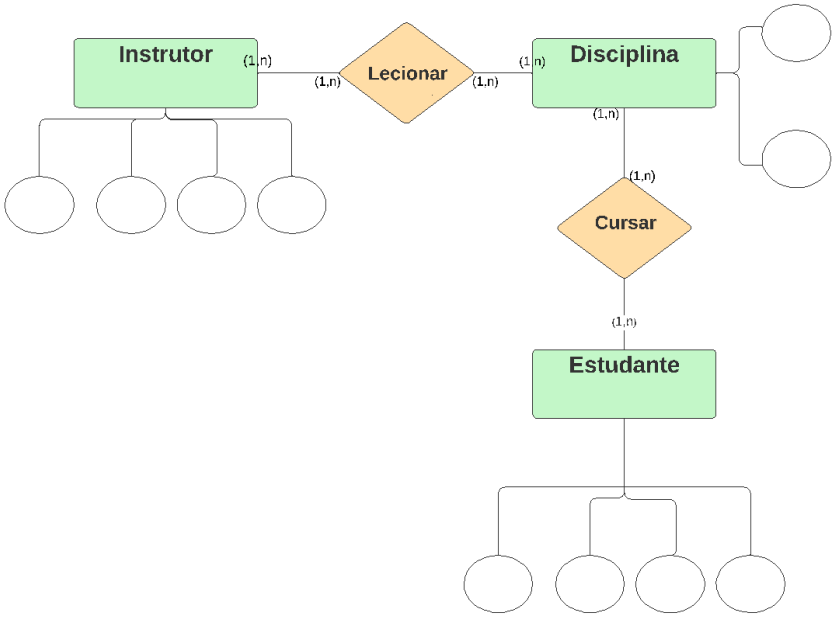
Nesta etapa são coletadas todas as regras de negócio que compõem o sistema a ser desenvolvido. Essas regras são revisadas diretamente com o Product Owner do produto. Essas regras serão catalogadas e documentadas nas etapas subsequentes da modelagem de dados.

Nesta fase, quanto mais informações relevantes tivermos, mais confiantes poderemos modelar.

Modelagem Conceitual

Nesta etapa, é criado um modelo conceitual que descreve as entidades, atributos e relacionamentos em alto nível de abstração. É utilizado um diagrama de Entidade-Relacionamento (ER) para representar o modelo. Este modelo é independente do software ou do banco de dados que será usado para implementá-lo.

O objetivo é ter uma visão geral do



sistema e as relações entre seus componentes para que seja possível criar um modelo lógico mais detalhado posteriormente.

Aqui está um exemplo de um Diagrama Entidade Relacionamento, ou ER:

Exemplo de tabelas que compõem o diagrama exemplificado acima.

Relacionamento	Atributos
Possui	Codigo_Disciplina, Codigo_Instrutora, Codigo_Estudante, Nome.

Figura – Representação de Entidade-Relacionamento (entidades)

Nesta etapa, são criados os primeiros desenhos envolvendo o sistema de banco de dados do projeto. A linguagem UML é utilizada para realizar esses diagramas, possibilitando uma visualização gráfica do que o sistema fará e dos atores envolvidos nesses processos. Daí a importância da fase de coleta de dados.

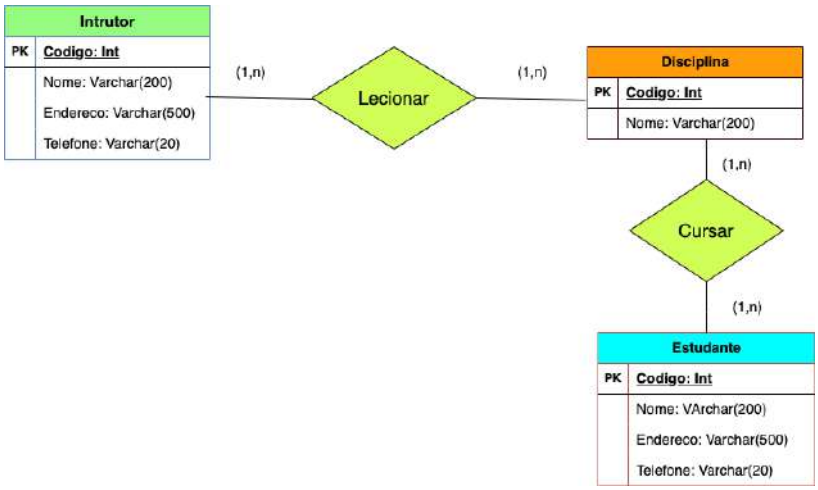
Assim, será possível estabelecer todas as entidades e suas dependências.

Entidade	Atributos	Relacionamento
Instrutor	Código, Nome, Endereço, Telefone.	Com Disciplina 1:N total
Disciplina	Código, Nome	Com Endereço 1:N total & com Pessoa Estudante 1:N total
Estudante	Código, Nome	Com Disciplina 1: N total

Figura – Representação de Entidade-Relacionamento (Relacionamento com atributos)

Modelagem Lógica

A partir do modelo conceitual, é criado um modelo lógico que descreve as tabelas, colunas e relacionamentos necessários para implementar o sistema no banco de dados.



É utilizado um diagrama Entidade-Relacionamento (ER) detalhado ou um modelo relacional para representar o modelo. Este modelo é sempre específico para um determinado software ou banco de dados e inclui detalhes sobre as chaves primárias e estrangeiras, tipos de campos, restrições e outros detalhes técnicos. O objetivo é fornecer uma especificação detalhada do design do banco de dados.

A etapa Modelo Lógico de Dados permite visualizar todas as interações lógicas entre cada chave primária que garante a unicidade de cada dado registrado no banco de dados e chaves estrangeiras que representam diferentes relacionamentos entre duas tabelas.

Uma chave primária determina que um registro no banco de dados apareça apenas uma vez, evita registros duplicados e ajuda a garantir a confiabilidade das informações. As chaves estrangeiras representam conexões entre dados no mesmo modelo de dados relacional.

Implementação Física

Nesta etapa, o modelo lógico é implementado em um banco de dados real. As tabelas são criadas, as restrições são definidas, as chaves primárias e estrangeiras são estabelecidas e os tipos de dados são definidos. Também é feita a carga inicial de dados no banco de dados.

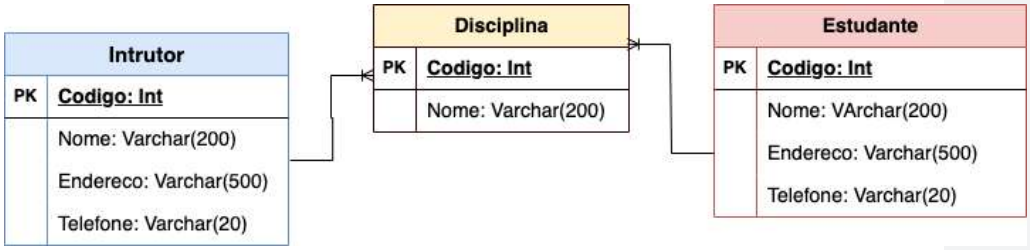
O objetivo é criar um banco de dados que atenda aos requisitos do sistema e permita a manipulação eficiente dos dados.

O próximo passo é estabelecer o banco de dados, seguindo a estruturação lógica dos dados que foi elaborada anteriormente. Para isso, será gerada uma instância de dados e, em seguida, um ou mais bancos de dados com suas respectivas tabelas e colunas.

A criação dessa estrutura será realizada por meio da linguagem SQL, que possibilitará não só a construção do modelo, mas também a manipulação dos registros e outras funções mais avançadas.

É importante destacar que a escolha do SGBD a ser utilizado na criação do modelo físico não interfere na definição do modelo lógico já estabelecido.

Para ilustrar melhor essa ideia, confira o exemplo de um modelo físico de dados abaixo:



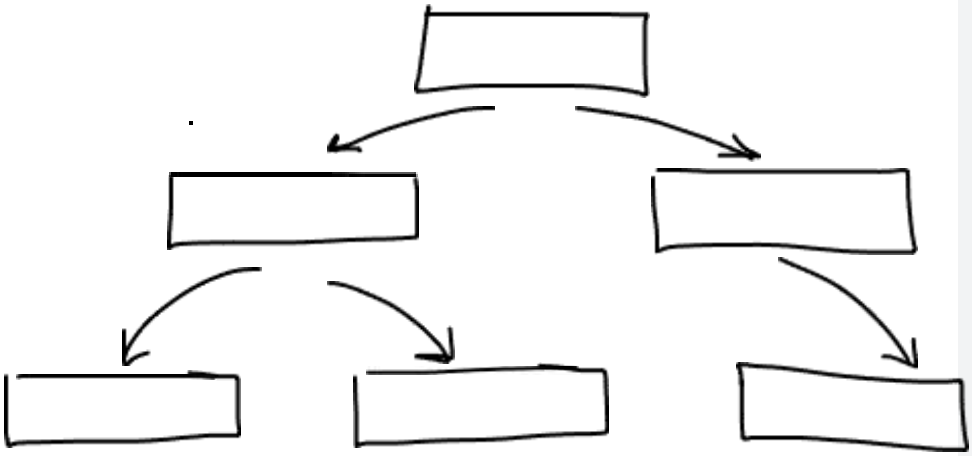
3.4. Principais tipos de modelos de dados

Existem vários tipos de modelos de dados, e cada um tem sua própria aplicação específica. Será exemplificado de forma breve os principais tipos de modelos de dados e exemplos de sua aplicação:

Modelo hierárquico

O modelo hierárquico é estruturado em forma de árvore e utiliza conceitos de níveis e pais/filhos para representar as relações entre os dados. Esse modelo foi muito utilizado em sistemas mainframe e pode ser encontrado em aplicações legadas.

Exemplo de Modelo Hierárquico

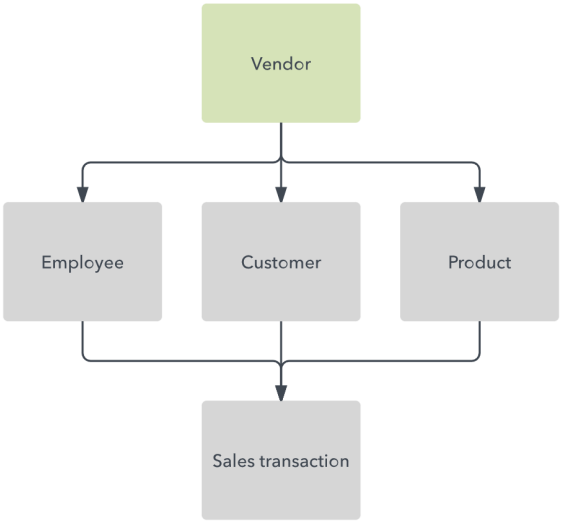


Exemplo: O sistema de arquivos do Windows pode ser considerado uma aplicação do modelo hierárquico, onde pastas e subpastas são organizadas como uma árvore.

Modelo em rede

O modelo de rede é uma variação do modelo hierárquico que permite múltiplas relações entre diferentes entidades. Utiliza conceito de registros e conjuntos de registros, onde cada registro pode estar relacionado com outros registros em diferentes conjuntos.

Exemplo: Uma rede social, como o Facebook, pode ser considerada uma aplicação do modelo em rede, onde usuários podem ter várias conexões com outros usuários, formando uma rede complexa.

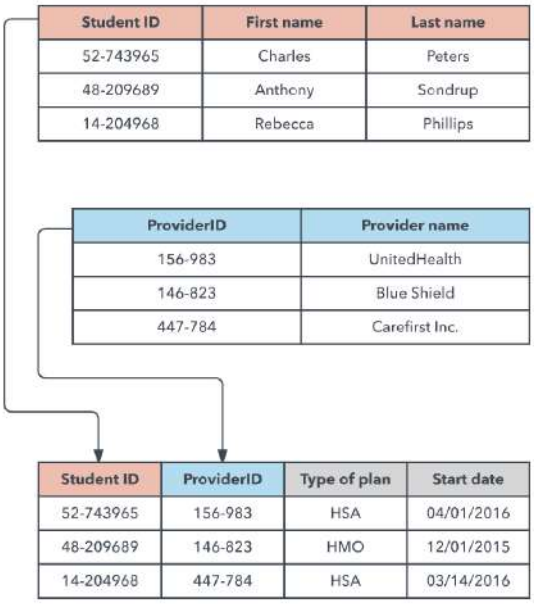


Exemplo de Modelo de Rede

Modelo relacional

O modelo relacional utiliza tabelas com chaves primárias e estrangeiras para representar as relações entre os dados. É o modelo mais utilizado atualmente e é amplamente suportado por sistemas de gerenciamento de bancos de dados.

Exemplo: Um sistema de gerenciamento de vendas pode utilizar o modelo relacional para armazenar informações sobre clientes, produtos, pedidos e faturas, em uma série de tabelas interconectadas.



Student ID	First name	Last name
52-743965	Charles	Peters
48-209689	Anthony	Sondrup
14-204968	Rebecca	Phillips

ProviderID	Provider name
156-983	UnitedHealth
146-823	Blue Shield
447-784	Carefirst Inc.

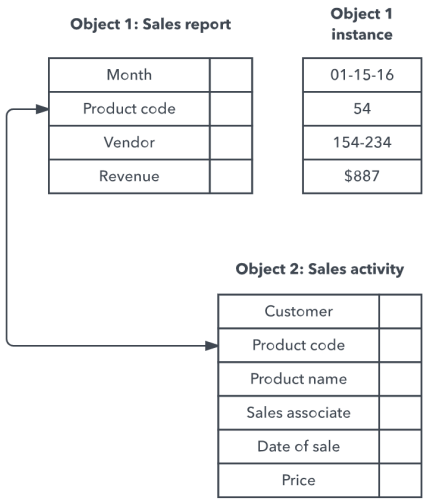
Student ID	ProviderID	Type of plan	Start date
52-743965	156-983	HSA	04/01/2016
48-209689	146-823	HMO	12/01/2015
14-204968	447-784	HSA	03/14/2016

Exemplo de Modelo Relacional

Modelo orientado a objetos

O modelo orientado a objetos utiliza conceitos de classes e objetos para representar os dados e suas relações. Esse modelo é muito utilizado em linguagens de programação orientadas a objetos e é bastante eficiente quando se trabalha com dados complexos.

Exemplo: Um jogo de videogame pode utilizar o modelo orientado a objetos para representar personagens, inimigos, itens, cenários e outras entidades do jogo, em uma estrutura hierárquica de classes e objetos.

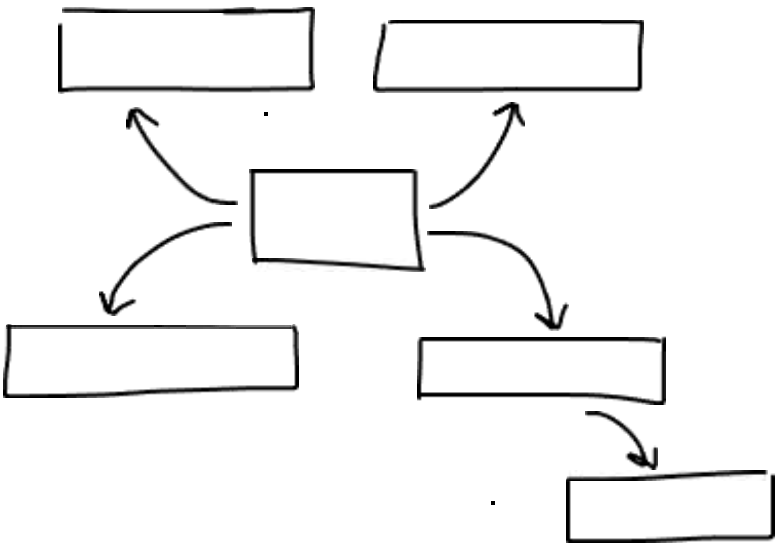


Exemplo de Modelo Orientado a Objeto

Modelo dimensional

O modelo dimensional é utilizado em sistemas de data warehousing e utiliza conceitos de fatos e dimensões para representar os dados. Esse modelo apresenta-se como uma alternativa ao modelo relacional em cenários em que se trabalha com grande quantidade de informações.

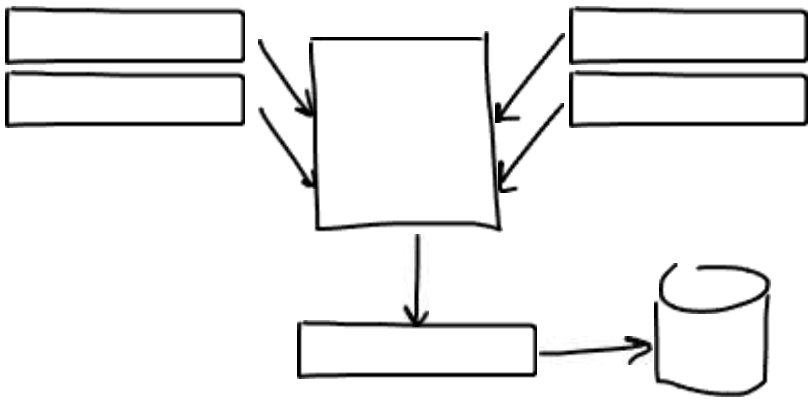
Exemplo de Modelo Dimensional



Exemplo: Uma empresa de varejo pode utilizar o modelo dimensional para armazenar informações sobre vendas, produtos, filiais e clientes, em um cubo de dados multidimensional que permite análises mais eficientes.

Modelo semântico

O modelo semântico representa o significado dos dados e suas relações, ao invés de apenas sua estrutura física. Ele utiliza ontologias e taxonomias para organizar os dados. Esse modelo é especialmente útil em sistemas que precisam compreender e interpretar informações de forma inteligente.



Exemplo de Modelo Semântico

Exemplo: Um assistente virtual, como a Siri da Apple, pode utilizar o modelo semântico para entender e responder perguntas dos usuários, utilizando ontologias e taxonomias para interpretar o significado das palavras e frases utilizadas.

3.5. Práticas para a criação de um modelo de dados eficiente

A criação de um modelo de dados eficiente envolve uma combinação de habilidades técnicas e compreensão dos requisitos do negócio. É importante seguir boas práticas de modelagem e manter o modelo atualizado à medida que o negócio evolui.

Os passos para a criação de um modelo de dados eficiente 

1. **Identificação dos requisitos do negócio:** O primeiro passo na criação de um modelo de dados eficiente é entender os requisitos do negócio e como os dados serão utilizados. Isso pode ser feito por meio de entrevistas com os usuários finais, análise de documentos e outros métodos.
2. **Identificação das entidades e relacionamentos:** Em seguida, é preciso identificar as entidades que fazem parte do negócio e como elas se relacionam. Isso pode ser feito utilizando técnicas como diagrama de entidade-relacionamento (ER) ou modelagem de objetos.
3. **Normalização do modelo:** O modelo deve ser normalizado para eliminar redundâncias e inconsistências, o que ajuda a garantir a integridade dos dados e aumentar a eficiência do banco de dados.
4. **Definição das chaves primárias e estrangeiras:** As chaves primárias devem ser definidas para cada tabela e as relações entre as tabelas devem ser estabelecidas usando chaves estrangeiras.
5. **Definição dos tipos de dados:** Os tipos de dados apropriados devem ser escolhidos para cada coluna, garantindo a precisão e coerência dos dados armazenados.
6. **Garantia da escalabilidade e desempenho:** O modelo deve ser projetado para garantir a escalabilidade e o desempenho do banco de dados ao longo do tempo. Isso inclui planejar a capacidade do banco de dados e otimizar as consultas para melhorar o desempenho.
7. **Documentação do modelo:** Todas as decisões tomadas durante o processo de modelagem devem ser documentadas e atualizadas conforme necessário.

4. Bancos de dados relacionais 

Um banco de dados relacional é um tipo de banco de dados que armazena e gerencia dados organizados em tabelas relacionadas. Essas tabelas são compostas por colunas (campos) e linhas (registros), onde cada linha representa uma entidade e cada coluna representa um atributo dessa entidade.

As tabelas em um banco de dados relacional são interconectadas por meio de chaves estrangeiras, que estabelecem relacionamentos entre as entidades.

Por exemplo, uma tabela de clientes pode estar relacionada a uma tabela de pedidos por meio da chave estrangeira "id_cliente". Isso permite que os dados sejam consultados e combinados de maneira eficiente, facilitando a realização de consultas complexas sobre os dados armazenados.

Tabela 01: Cliente

Coluna	Tipo de Dado	Restrições
id_cliente	int	PK, Autoincremento
nome	varchar(50)	Not Null
sobrenome	varchar(50)	Not Null
endereco	varchar(100)	Not Null
cidade	varchar(50)	Not Null
estado_provincia	varchar(50)	Not Null
pais	varchar(50)	Not Null
email	varchar(50)	Not Null, Único
telefone	varchar(20)	Not Null

Tabela 02: Pedidos

Coluna	Tipo de Dado	Restrições
id_pedido	int	PK, Autoincremento
data_pedido	date	Not Null
valor_total	decimal(10, 2)	Not Null
id_cliente	int	FK

(Imagens de Tabelas: Criação própria)

Alguns dos principais benefícios dos bancos de dados relacionais incluem a facilidade de uso, escalabilidade, segurança,

flexibilidade e integridade dos dados. Eles são amplamente utilizados em sistemas empresariais para armazenar informações como estoque, vendas, folha de pagamento e outros dados importantes relacionados aos negócios.

4.1. Estrutura dos bancos de dados relacionais

Os bancos de dados relacionais são compostos por várias tabelas que armazenam informações relacionadas entre si. Cada tabela é composta por colunas e linhas, onde cada linha representa um registro ou uma entidade e cada coluna representa um atributo dessa entidade.

As colunas em uma tabela têm um tipo de dado específico, como texto (varchar), número inteiro (int), número decimal (decimal), data (date) etc. Além disso, as colunas podem ter restrições para garantir que os dados sejam válidos e consistentes, como "Not Null" (não pode ser nulo), "Unique" (deve ser único), "Primary Key" (chave primária) e "Foreign Key" (chave estrangeira).

Coluna	Tipo de Dado	Restrições
id_produto	int	PK, Autoincremento
nome	varchar(50)	Not Null
descricao	varchar(100)	Not Null
preco_unitario	decimal(10,2)	Not Null
quantidade	int	Not Null
data_validade	date	Not Null

A chave primária (PK) é um campo ou conjunto de campos que identifica exclusivamente cada registro na tabela. A chave estrangeira (FK) é um campo ou conjunto de campos em uma tabela que faz referência à chave primária em outra tabela, estabelecendo um relacionamento entre as duas tabelas.

Além das tabelas, os bancos de dados relacionais também incluem outros objetos, como índices, visões e procedimentos armazenados. Os índices são utilizados para melhorar o desempenho das consultas ao banco de dados, visões são consultas predefinidas que podem ser usadas como uma tabela virtual e procedimentos armazenados são rotinas que são armazenadas no banco de dados e executadas quando chamadas.

As estruturas dos bancos de dados relacionais são baseadas em tabelas que armazenam informações relacionadas, com colunas que representam os atributos dessas entidades. As chaves primárias e estrangeiras são usadas para estabelecer relacionamentos entre as tabelas, enquanto índices, visões e procedimentos armazenados são usados para melhorar o desempenho e a funcionalidade do banco de dados.

4.2. Normalização de tabelas

A normalização de tabelas em banco de dados é um processo para reduzir a redundância e a inconsistência dos dados, organizando-os em estruturas mais eficientes e sem repetições desnecessárias. As principais formas normais são:

Primeira Forma Normal (1FN):

Na primeira forma nominal(1FN) cada tabela deve ter uma chave primária e todos os valores devem ser atômicos, ou seja, não podem ser subdivididos em partes menores.

Exemplo

Suponha que temos uma tabela chamada "Funcionários" com as seguintes colunas:

Tabela 3: Funcionários

ID do Funcionário
Nome do Funcionário
Endereço Completo
Telefone

Essa tabela não está em 1FN, pois a coluna "Endereço Completo" contém múltiplos valores. Para normalizá-la, podemos dividir em duas tabelas relacionais:

Tabela 4: Funcionários

ID do Funcionário (PK)
Nome do Funcionário
Telefone

Tabela 2: Endereços dos Funcionários

ID do Endereço (PK)
ID do Funcionário (FK)
Rua e Número
Bairro
Cidade
Estado
CEP

Com essa divisão, agora cada funcionário pode ter um ou mais endereços associados a ele na tabela Endereços dos Funcionários, eliminando a coluna "Endereço Completo" que continha múltiplos valores. Dessa forma, a tabela Funcionários está agora em 1FN.

Segue abaixo exemplos de como ficariam as tabelas após a normalização:

Tabela 3: Funcionários normatizada **(1FN)**

ID do Funcionário	Nome do Funcionário	Telefone
1	João	(11) 9999-9999
2	Maria	(21) 8888-8888
3	Pedro	(31) 7777-7777

Tabela 4: Endereços dos Funcionários normatizada **(1FN)**

ID do Endereço	ID do Funcionário	Rua e Número	Bairro Cidade	Estado	CEP
1	1	Rua A, 123	Centro	São Paulo SP	01000-000
2	2	Rua B, 456	Copacabana	Rio de Janeiro RJ	22000-000
3	2	Av. C, 789	Ipanema	Rio de Janeiro RJ	22000-000
4	3	Rua D, 321	Savassi	Belo Horizonte MG	30140-000

Segunda Forma Normal (2FN)

Na segunda forma nominal, cada coluna não-chave da tabela deve depender da chave primária inteira e não de apenas uma parte dela.

Exemplo

Suponha que temos uma loja virtual de roupas e queremos criar um banco de dados para gerenciar nossos produtos e vendas.

Inicialmente, criamos uma tabela chamada "Produtos" com as seguintes colunas: ID do Produto, Nome do Produto, Marca do Produto, Tamanho do Produto, Cor do Produto, Categoria do Produto e Preço do Produto.

Porém, depois de analisar a estrutura da tabela, percebemos que existe uma dependência funcional entre a coluna "Categoria do Produto" e as colunas "Nome do Produto", "Marca do Produto", "Tamanho do Produto" e "Cor do Produto". Ou seja, cada produto pertence a uma única categoria e essa categoria determina o nome, marca, tamanho e cor do produto.

Para resolver esse problema e atingir a Segunda Forma Normal (2FN), precisamos dividir a tabela em duas:

- **Tabela "Categorias":** com os atributos ID da Categoria | Nome da Categoria
- **Tabela "Produtos":** com os atributos ID do Produto | Nome do Produto | Marca do Produto | Tamanho do Produto | Cor do Produto | ID da Categoria | Preço do Produto

A tabela "Produtos" agora atende à 2ª Forma Normal, pois suas colunas dependem completamente da chave primária **(ID do Produto).** Já a tabela "Categorias" armazena informações exclusivamente relacionadas às categorias dos produtos, sem interferir nas outras informações da tabela "Produtos".

A seguir, você poderá observar esta tabela com seus devidos atributos e sugestões de valores

Tabela: Categorias normatizada **(2 FN)**

ID da Categoria	Nome da Categoria
1	Camisetas

2	Calças
3	Blusas

Tabela: Produtos normatizada **(2 FN)**

ID do Produto	Nome do Produto	Marca do Produto	Tamanho do Produto	Cor do Produto	ID da Categoria	Preço do Produto
1	Camiseta básica	Nike	G	Preta	1	39.90
2	Calça jeans	Levi's	42	Azul	2	129.90
3	Blusa de lã	Adidas	P	Vermelha	3	89.90

Note que cada produto pertence a uma única categoria e que essa relação é definida pelo ID da Categoria presente na tabela "Produtos". Isso garante uma organização mais precisa e eficiente do banco de dados, permitindo que as informações sejam acessadas e atualizadas de forma mais fácil e confiável.

Terceira Forma Normal (3FN)

Na terceira forma nominal todas as colunas não-chave da tabela devem depender apenas da chave primária e não de outras colunas que não fazem parte da chave.

A normalização de tabelas ajuda a garantir a integridade dos dados, evita inconsistências e reduz o espaço de armazenamento necessário. Além disso, facilita a manutenção e o acesso aos dados por meio de consultas SQL.

Ao aplicar a normalização, é importante considerar o equilíbrio entre a eficiência e a simplicidade do esquema de bancos de dados.

A Terceira Forma Normal (3FN) é uma técnica de normalização do modelo de dados que visa eliminar as dependências transitivas entre os atributos, ou seja, quando um atributo depende de outro que não é chave primária.

Para exemplificar a 3FN, serão criadas duas tabelas com seus respectivos atributos e valores:

Tabela: Cliente

ID	Nome	Endereço
1	João Silva	Rua das Flores, 123
2	Maria Santos	Av. Brasil, 456

Tabela: Pedido

ID	Data	Cliente_ID	Valor
1	2023-05-01	1	100
2	2023-05-02	2	200

Na tabela Cliente, temos a chave primária ID e os atributos Nome e Endereço. Na tabela Pedido, temos a chave primária ID, os atributos Data e Valor, e o atributo Cliente_ID que referencia a tabela Cliente.

Neste caso, podemos ver que o atributo Endereço na tabela Cliente não depende diretamente da chave primária ID, mas sim do atributo Nome. Isso representa uma dependência transitiva e pode levar a problemas de redundância e inconsistência nos dados.

Para resolver esse problema, podemos aplicar a 3FN e separar a tabela Cliente em duas:

Tabela: Cliente normatizada **(3 FN)**

ID	Nome
1	Maria Santos
2	João Silva

Tabela: Endereço normatizada **(3 FN)**

ID	Cliente_ID	Endereço
1	1	Rua das Flores, 123

2	2	Av. Brasil, 456

Agora, a tabela Cliente possui apenas a chave primária ID e o atributo Nome, enquanto a tabela Endereço possui a chave primária ID, o atributo Cliente_ID que referencia a tabela Cliente e o atributo Endereço.

Dessa forma, eliminamos a dependência transitiva do atributo Endereço em relação à chave primária ID na tabela Cliente e tornamos o modelo de dados mais eficiente e livre de redundâncias.

4.3. Chaves primárias e estrangeiras

Chaves primárias e estrangeiras são conceitos importantes em bancos de dados relacionais.

Uma chave primária é uma coluna ou conjunto de colunas em uma tabela que identifica de forma exclusiva cada linha ou registro na tabela. Essa coluna ou conjunto de colunas é escolhida como a principal chave para a tabela e é usada para garantir a integridade dos dados. A chave primária não pode ter valores duplicados ou nulos.

Já a chave estrangeira é uma coluna em uma tabela que se refere à chave primária de outra tabela. Ela estabelece uma relação entre as duas tabelas, permitindo que os dados sejam vinculados. A chave estrangeira é usada para manter a consistência referencial entre as tabelas.

Por exemplo, se houver uma tabela "Pedidos" com uma chave primária "ID do Pedido", outra tabela "Itens do Pedido" poderia ter uma chave estrangeira "ID do Pedido" que se refere à chave primária da tabela "Pedidos". Isso permite que os itens do pedido estejam vinculados ao pedido correto e evita a inserção de dados inconsistentes.

Exemplo

Tabela de Clientes e Pedidos

Suponha que temos duas tabelas em um banco de dados: "Clientes" e "Pedidos". A tabela "Clientes" possui as seguintes colunas:

ID_Cliente (chave primária)	Nome	Endereço	Telefone

Já a tabela "Pedidos" possui as seguintes colunas:

ID_Pedido (chave primária)	ID_Cliente (chave estrangeira)	Data	Total

Observe que a coluna "ID_Cliente" na tabela "Pedidos" é uma chave estrangeira que faz referência à coluna "ID_Cliente" na tabela "Clientes". Essa relação permite vincular os pedidos aos clientes correspondentes.

Esses exemplos ilustram como as chaves primárias e estrangeiras podem ser usadas para relacionar dados em um banco de dados relacional. A chave primária identifica exclusivamente cada registro em uma tabela, enquanto a chave estrangeira estabelece uma relação entre as tabelas usando a chave primária de outra tabela.

Em resumo, a chave primária é usada para identificar exclusivamente registros em uma tabela, enquanto a chave estrangeira é usada para estabelecer relações entre tabelas.

4.4. Índices em bancos de dados

Os índices em bancos de dados são estruturas de dados adicionais que permitem uma melhoria significativa no desempenho das consultas. Eles funcionam como um mapa, apontando a localização dos registros na tabela que correspondem a um determinado valor ou conjunto de valores.

Quando uma consulta é executada com base em uma coluna indexada, o banco de dados pode usar o índice para localizar os registros relevantes muito mais rapidamente do que se tivesse que pesquisar toda a tabela.

Os índices podem ser criados em uma ou mais colunas de uma tabela, dependendo das consultas mais comuns que serão realizadas. Portanto, é importante equilibrar o número e a abrangência dos índices em relação ao desempenho geral do sistema e aos requisitos de armazenamento.

5. Linguagem SQL

SQL, ou Structured Query Language, é a linguagem padrão para gerenciamento de banco de dados relacionais. Ele permite que os usuários manipulem e consultem dados em um banco de dados com facilidade.

O SQL foi criado na década de 1970 por Donald D. Chamberlin e Raymond F. Boyce da IBM. Desde então, ele se tornou o padrão de fato para gerenciamento de banco de dados e é amplamente utilizado em todo o mundo.

Uma das principais características do SQL é sua capacidade de acessar e manipular dados em uma base de dados relacional. Isso inclui inserir, atualizar e excluir registros, bem como consultar informações específicas dentro da base de dados.

O SQL é uma linguagem declarativa, o que significa que os usuários fornecem instruções sobre o que eles querem que o banco de dados faça, em vez de como fazê-lo. Isso simplifica o processo de escrita de consultas complexas, já que os usuários não precisam se preocupar com as nuances técnicas da implementação.

Outra característica importante do SQL é sua capacidade de realizar operações em massa em grandes conjuntos de dados. Isso é feito por meio de instruções como SELECT, INSERT, UPDATE e DELETE.

Além disso, o SQL é altamente flexível e pode ser adaptado para atender às necessidades específicas de diferentes bancos de dados. Existem várias variantes do SQL, cada uma projetada para trabalhar com um tipo específico de banco de dados.

Principais ferramentas para a implementação de banco de dados SQL

Existem muitas ferramentas para implementação de banco de dados SQL, tanto para iniciantes quanto para usuários avançados. Abaixo estão algumas das principais ferramentas:

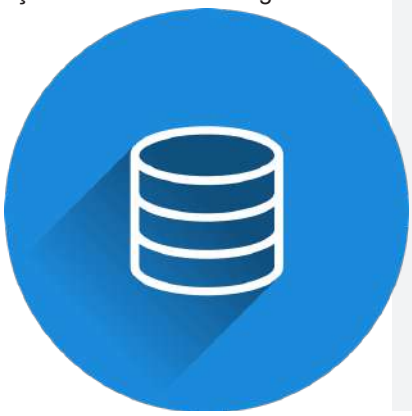


Imagem de Stephan por HYPERLINK "https://pixabay.com/pt/?utm_source=link-

MySQL Workbench

Esta é uma ferramenta gratuita e popular para gerenciamento de banco de dados MySQL. Ele oferece uma interface gráfica do usuário para criar e gerenciar bancos de dados, bem como recursos de modelagem de dados.

Oracle SQL Developer

Esta é uma ferramenta gratuita fornecida pela Oracle Corporation para desenvolver e gerenciar bancos de dados Oracle. Ela inclui recursos para escrever e executar consultas SQL, bem como recursos para gerenciar esquemas de banco de dados.

Microsoft SQL Server Management Studio

Esta é uma ferramenta gratuita para gerenciar bancos de dados SQL Server da Microsoft. Ela oferece uma ampla gama de recursos, desde a criação de bancos de dados até a execução de consultas complexas.

PostgreSQL

Este é um sistema de gerenciamento de banco de dados relacional de código aberto que suporta SQL. Ele é frequentemente usado em aplicações web e oferece recursos como transações ACID e indexação de texto completo.

SQLite

Este é um banco de dados SQL embutido, ou seja, ele é incorporado diretamente em aplicativos em vez de ser executado como um servidor separado. É frequentemente usado em aplicativos móveis e oferece uma boa compatibilidade com o SQL padrão.

Principais instruções do SQL

Existem diversas instruções na linguagem SQL que permitem aos usuários manipular e consultar dados em um banco de dados. Abaixo estão algumas das principais instruções do SQL:

SELECT: Esta instrução é usada para recuperar dados do banco de dados. É a instrução mais básica do SQL e permite que os usuários especifiquem as colunas das quais desejam recuperar dados, bem como filtros para restringir os resultados.

Exemplos de programação utilizando o comando **SELECT**

1. **Selecionando todas as colunas de uma tabela:**

SELECT * **FROM** tabela;
Comentário sobre o código: Neste exemplo, "tabela" é o nome da tabela que contém os dados a serem selecionados. O asterisco (*) é usado para selecionar todas as colunas na tabela.

2. **Selecionando colunas específicas de uma tabela:**

SELECT coluna1, coluna2, coluna3 **FROM** tabela;
Comentário sobre o código: Neste exemplo, "coluna1", "coluna2" e "coluna3" são os nomes das colunas que serão selecionadas na tabela especificada. Somente essas colunas serão retornadas na consulta.

3. **Selecionando colunas específicas com um filtro:**

```
SELECT coluna1, coluna2 FROM tabela WHERE coluna3 = 'valor';
```

Comentário sobre o código: Neste exemplo, estamos selecionando apenas as colunas "coluna1" e "coluna2" da tabela "tabela" onde o valor da coluna "coluna3" for igual a "valor".

INSERT: Esta instrução é usada para inserir novos dados em uma tabela. Os usuários podem especificar os valores que desejam inserir para cada coluna da tabela.

Exemplos de programação utilizando o comando **INSERT**

1. **Inserir um registro em uma tabela com valores específicos:**

```
INSERT INTO tabela_exemplo (coluna1, coluna2, coluna3)
VALUES ('valor1', 'valor2', 'valor3');
```

Comentário sobre o código: Nesse exemplo, estamos inserindo um novo registro na tabela "tabela_exemplo" com os valores "valor1", "valor2" e "valor3" nas colunas "coluna1", "coluna2" e "coluna3", respectivamente.

2. **Inserir vários registros em uma tabela ao mesmo tempo:**

```
INSERT INTO tabela_exemplo (coluna1, coluna2, coluna3)
VALUES ('valor1', 'valor2', 'valor3'),
       ('valor4', 'valor5', 'valor6'),
       ('valor7', 'valor8', 'valor9');
```

Comentário sobre o código: Neste exemplo, estamos inserindo três novos registros na tabela, "tabela_exemplo" ao mesmo tempo, cada um com seus próprios valores nas colunas especificadas.

3. **Inserir dados de outra tabela em uma nova tabela:**

```
INSERT INTO nova_tabela (coluna1, coluna2, coluna3)
SELECT coluna1, coluna2, coluna3 FROM tabela_original
WHERE condicao = 'algum_valor';
```

Comentário sobre o código: Nesse caso, estamos criando uma nova tabela chamada "nova_tabela" e inserindo nela todos os registros da tabela, "tabela_original" que atendem à condição especificada. As colunas da nova tabela são definidas na primeira linha do comando INSERT, e os valores delas são obtidos pelo SELECT na segunda linha.

UPDATE: Este comando é usado para atualizar valores existentes em uma tabela. Os usuários podem especificar os valores que desejam atualizar e podem adicionar cláusulas WHERE para restringir os registros que serão atualizados.
Exemplo de programação utilizando o comando **UPDATE**

1. **Atualização de um registro específico na tabela:**

```
UPDATE clientes
SET nome = 'João Silva', email = 'joao.silva@example.com'
WHERE id = 123;
```

Comentário sobre o código: Neste exemplo, estamos atualizando o registro da tabela "clientes" com o ID igual a 123. O comando **SET** define quais colunas serão atualizadas e seus novos valores.

DELETE: Este comando é usado para excluir registros de uma tabela. Os usuários podem adicionar cláusulas WHERE para restringir os registros que serão excluídos.
Exemplo de programação utilizando o comando **DELETE**

1. **Excluindo todos os registros da tabela:**

```
DELETE FROM nome_da_tabela;
```

Comentário sobre o código: Este comando exclui todos os registros da tabela especificada.

2. **Excluindo registros com uma condição específica:**

```
DELETE FROM nome_da_tabela WHERE coluna = valor;
```

Comentário sobre o código: Este comando exclui todos os registros da tabela que atendem à condição especificada na cláusula WHERE.

CREATE: é usada para criar novas tabelas, índices e outros objetos no banco de dados.

1. **Criando uma tabela:**

```
CREATE TABLE clientes (
id INT PRIMARY KEY,
nome VARCHAR(50),
email VARCHAR(50),
telefone VARCHAR(20)
);
```

Comentário sobre o código: Neste exemplo, estamos criando uma tabela chamada "clientes" com quatro colunas: "id", "nome", "email" e "telefone". Além disso, definimos a coluna "id" como chave primária usando a cláusula PRIMARY KEY. O tipo de dado VARCHAR significa que estamos armazenando strings (texto) de tamanho variável.

2. **Criando um índice:**

```
CREATE INDEX idx_clientes_email ON clientes (email);
```

Comentário sobre o código: Este comando cria um novo índice chamado "idx_clientes_email" na tabela "clientes" para a coluna "email". Isso ajuda a acelerar as consultas que envolvem essa coluna, pois o banco de dados pode usar o índice em vez de precisar examinar todas as linhas da tabela.

3. **Criando um usuário:**

```
CREATE USER 'joao'@'localhost' IDENTIFIED BY 'senha123';
```

Comentário sobre o código: Este comando cria um novo usuário chamado "joao" com senha "senha123". Usuários são usados para controlar o acesso ao banco de dados e seus objetos.

DROP: é usada para excluir tabelas, índices e outros objetos do banco de dados.

1. **Exemplo de DROP TABLE:**

```
DROP TABLE clientes;
```

Comentário sobre o código: Este comando irá excluir a tabela "clientes" do banco de dados.

2. **Exemplo de DROP INDEX:**

```
DROP INDEX idx_nome;
```

Comentário sobre o código: Este comando irá excluir o índice "idx_nome" do banco de dados.

3. **Exemplo de DROP VIEW:**

```
DROP VIEW produtos_vendidos;
```

Comentário sobre o código: Este comando irá excluir a visão "produtos_vendidos" do banco de dados.

4. **Exemplo de DROP DATABASE:**

```
DROP DATABASE minha_empresa;
```

Comentário sobre o código: Este comando irá excluir o banco de dados "minha_empresa" inteiro.

É importante ter cuidado ao utilizar a instrução **DROP**, pois ela pode excluir dados importantes do banco de dados permanentemente. Por isso, recomenda-se fazer backups regulares dos bancos de dados e testar os comandos **DROP** em ambientes de desenvolvimento ou de teste antes de aplicá-los em produção.

ALTER: é usada para alterar a estrutura de uma tabela existente. Por exemplo, os usuários podem adicionar ou remover colunas ou alterar o tipo de dados de uma coluna existente.

1. **Exemplo de ALTER TABLE - Adicionar uma Coluna:**

```
ALTER TABLE clientes
ADD COLUMN telefone VARCHAR(15);
```

Comentário sobre o código: Este comando irá adicionar uma nova coluna chamada "telefone" à tabela "clientes", com tipo de dados VARCHAR e tamanho máximo de 15 caracteres.

2. **Exemplo de ALTER TABLE - Modificar uma Coluna:**

```
ALTER TABLE clientes
MODIFY COLUMN nome_completo VARCHAR(50);
```

Comentário sobre o código: Este comando irá modificar o tipo de dados da coluna "nome_completo" da tabela "clientes" para **VARCHAR** e tamanho máximo de 50 caracteres.

3. **Exemplo de ALTER TABLE - Remover uma Coluna:**

```
ALTER TABLE clientes
DROP COLUMN endereco;
```

Comentário sobre o código: Este comando irá remover a coluna "endereco" da tabela "clientes".

Comentado [1]: Nalto bom dia, na hora da formatação ter atenção para essas palavras entre “”permanecerem sem acento e Ç, esta é a padronização do código de programação.

É importante lembrar que a instrução **ALTER** pode afetar diretamente os dados armazenados nas tabelas, então é recomendável realizar backup dos dados antes de realizar qualquer modificação na estrutura das tabelas. Além disso, é importante planejar bem as modificações antes de executá-las em ambientes de produção.

JOIN: é usada para combinar dados de duas ou mais tabelas em uma única consulta. Existem vários tipos de junções, incluindo **INNER JOIN, LEFT JOIN, RIGHT JOIN** e **FULL OUTER JOIN.**

1. **INNER JOIN:**

```
SELECT customers.name, orders.order_number, orders.total_price
FROM customers
INNER JOIN orders ON customers.customer_id = orders.customer_id;
```

Comentário sobre o código: Nesse exemplo, estamos pegando o nome do cliente e o número do pedido e total de preço de todos os pedidos que possuem um customer_id válido em ambas as tabelas.

2. **LEFT JOIN:**

```
SELECT customers.name, orders.order_number, orders.total_price
FROM customers
LEFT JOIN orders ON customers.customer_id = orders.customer_id;
```

Comentário sobre o código: Aqui, estamos selecionando os mesmos campos, mas dessa vez usando um **LEFT JOIN.** Isso significa que estamos pegando todos os registros da tabela "customers" e apenas os registros correspondentes da tabela "orders". Se não houver registros correspondentes na tabela "orders", os campos terão valores nulos.

GROUP BY: é usada para agrupar os resultados de uma consulta com base em uma ou mais colunas. Os usuários podem aplicar funções de agregação, como COUNT, SUM e AVG, aos grupos resultantes.

1. **Agrupando por uma coluna:**

```
SELECT country_code, COUNT(*) as num_of_customers
FROM customers
GROUP BY country_code;
```

Comentário sobre o código: Nesse exemplo, estamos agrupando os registros da tabela "customers" pela coluna "country_code" e contando o número de clientes para cada país. A função **COUNT()** é utilizada para contar o número de registros em cada grupo.

2. **Agrupando por mais de uma coluna:**

```
SELECT country_code, city, COUNT(*) as num_of_customers
FROM customers
GROUP BY country_code, city;
```

Comentário sobre o código: Aqui, estamos agrupando os registros da tabela "customers" por duas colunas - "country_code" e "city". Isso significa que a contagem será feita considerando tanto o país quanto a cidade de cada cliente.

5. Referências:

Oracle. What is a Database? Disponível em: https://www.oracle.com/database/what-is-a-database.html. Acesso em: 22 abr. 2023.

IBM. A Brief History of Databases. Disponível em: https://www.ibm.com/cloud/learn/database-a-brief-history-of-databases. Acesso em: 20 abr. 2023.

Oracle. The Evolution of Database Management Systems. Disponível em: https://www.oracle.com/database/what-is-a-database.html#the-evolution-of-database-management-systems. Acesso em: 20 abr. 2023.

Microsoft. O que é banco de dados? Disponível em: https://docs.microsoft.com/pt-br/sql/relational-databases/what-is-a-relational-database?view=sql-server-ver15. Acesso em: 30 abr. 2023.

W3schools. SQL Tutorial. Disponível em: https://www.w3schools.com/sql/default.asp. Acesso em: 30 abr. 2023.

"Introduction to Relational Databases" (https://www.orafaq.com/wiki/Introduction_to_Relational_Databases)

"Cloud Databases" (https://www.ibm.com/cloud/learn/cloud-databases)
"In-Memory Databases" (https://www.oracle.com/database/what-is-an-in-memory-database.html)

"Object-Oriented Database Systems" (https://www.springer.com/gp/book/9781461374089)

"NoSQL Overview" (https://dzone.com/articles/nosql-overview) Banco de dados orientado a objetos

Choosing the Right Database Management System, Dataversity (https://www.dataversity.net/choosing-right-database-management-system/)

How to Choose the Right Database for Your Application, Compose Articles (https://www.compose.com/articles/how-to-choose-the-right-database-for-your-application/)

Selecting a database for your application, IBM Cloud Education (https://cloud.ibm.com/docs/ibm-cloud-databases?topic=ibm-cloud-db-selecting-db)

ELMASRI, R.; NAVATHE, S. B. Sistemas de Banco de Dados. Pearson, 2011.

SILVA, F. N. Modelagem de Dados: Entendendo os conceitos fundamentais. Novatec Editora, 2015

Referência: JOHNSON, Richard. Desenvolvimento de Banco de Dados: planejamento, projeto e administração. Elsevier, 2005.

Referência: SILBERSCHATZ, Abraham; KORTH, Henry F.; SUDARSHAN, S. Sistema de Banco de Dados. Elsevier, 2012.

Referência: ELMASRI, Ramez; NAVATHE, Shamkant B. Fundamentals of database systems. Pearson Education India, 2014.

Referência: CONNOLLY, Thomas M.; BEGG, Carolyn E. Banco de Dados Modernos. Pearson Education do Brasil, 2011.

Referência: DATE, C. J. Introdução a sistemas de bancos de dados. Elsevier, 2004.

Referência: RAMAKRISHNAN, Raghu; GEHRKE, Johannes. Banco de Dados: Projeto, Implementação e Administração. McGraw-Hill Education, 2002.

Referência: TEOREY, Toby J.; LIGHTSTONE, Sam S.; NADEAU, Tom. Projetando um Banco de Dados Corporativo. Elsevier, 2010.

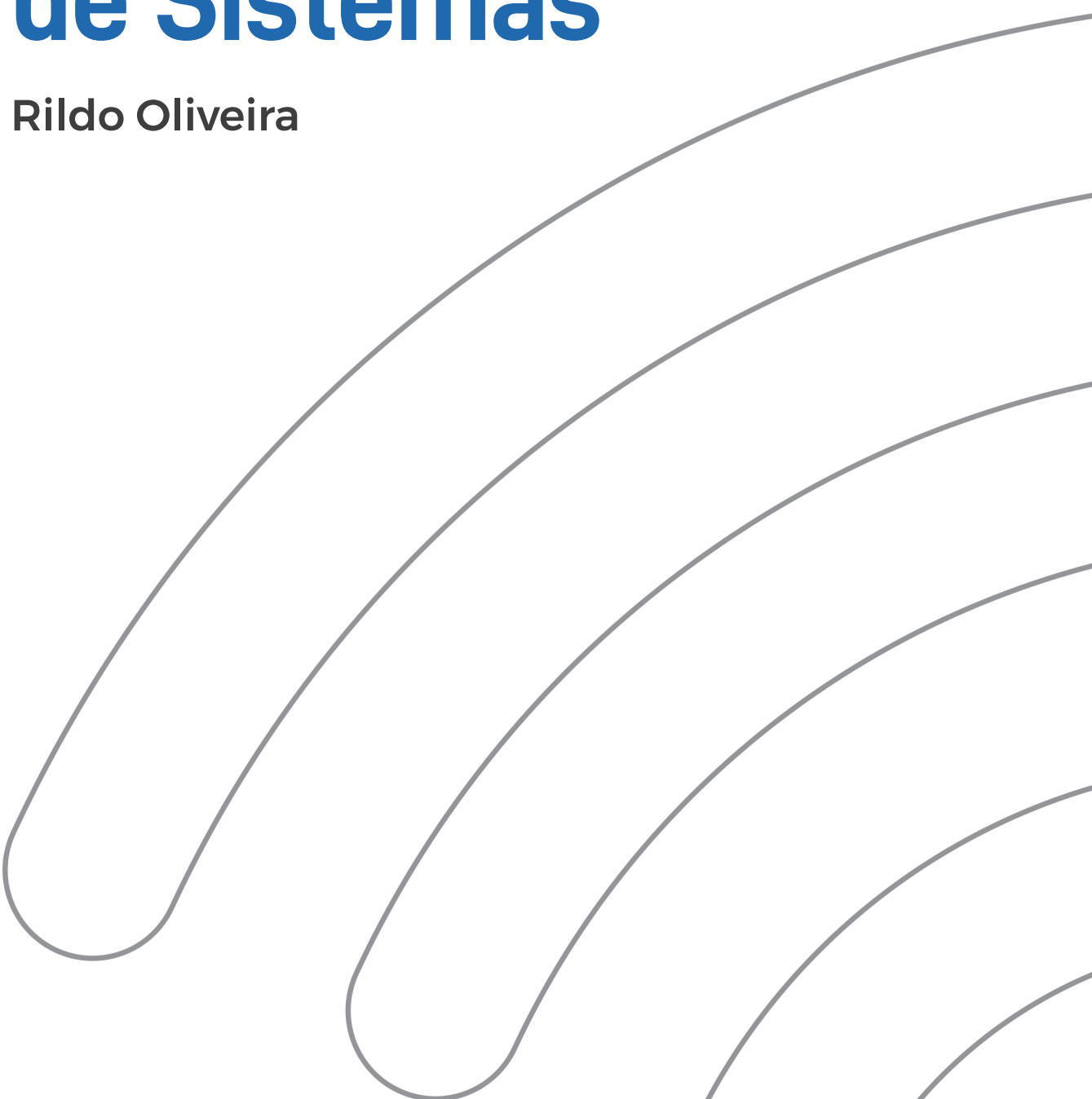
CURSO TÉCNICO EM

INFORMÁTICA

MÓDULO III

Projeto de Desenvolvimento de Sistemas

Rildo Oliveira



Prezado(a) estudante,

Seja bem-vindo à Projeto de Desenvolvimento de Sistemas. Nesta disciplina, você terá a oportunidade de estudar conceitos fundamentais como CSS, HTML, JAVASCRIPT, Apache, PHP, dentre outros, os quais farão parte do seu cotidiano como programador.

O estudo desta disciplina é de suma importância para ampliar seus conhecimentos sobre as linguagens de programação web, bem como para compreender o seu funcionamento. Com ela, você poderá aprender a utilizar a web e o conjunto de tecnologias que a formam, além de aplicar a linguagem HTML.

Desejo-lhe sucesso em sua jornada de aprendizado e principalmente no que tange ao aprimoramento de seus conhecimentos!

Bons estudos!

Atenciosamente,

Rildo Oliveira

Sumário

1.0 INTRODUÇÃO AOS CONCEITOS DE SISTEMAS PARA INTERNET

Os sistemas para internet são cada vez mais presentes em nosso dia a dia, desde as redes sociais até as plataformas de e-commerce. Eles são compostos por diversas tecnologias e ferramentas que trabalham juntas para oferecer uma experiência ao usuário. Neste contexto, é importante compreender a história da internet, o conceito de servidor e cliente, bem como os protocolos de comunicação utilizados na rede.

1.1 História e evolução da internet

A história da internet remonta à década de 1960, quando foi criada a ARPANET, uma rede que conectava universidades e centros de pesquisa financiados pelo Departamento de Defesa dos Estados Unidos. Com o passar dos anos, a rede evoluiu e se tornou a internet que conhecemos hoje. A evolução da internet permitiu o surgimento de novas tecnologias e serviços, como a World Wide Web (WWW), que revolucionou a forma como as informações são acessadas e compartilhadas.

1.2 Conceitos de servidor e cliente

Os sistemas para internet são compostos por dois elementos principais: o servidor e o cliente. O servidor é um computador que armazena e processa as informações que serão disponibilizadas na internet, enquanto o cliente é o dispositivo utilizado pelo usuário para acessar essas informações, como um computador ou um smartphone. É importante entender como esses dois elementos se relacionam para que seja possível desenvolver sistemas para internet eficientes e seguros.

1.3 Protocolos de comunicação na internet (TCP/IP, HTTP)

Os protocolos de comunicação são fundamentais para que os sistemas para internet funcionem corretamente. O TCP/IP (Transmission Control Protocol/Internet Protocol) é o protocolo mais utilizado na internet e é responsável por estabelecer a conexão entre o servidor e o cliente. Já o HTTP (Hypertext Transfer Protocol) é o protocolo utilizado para transferir dados entre o servidor e o cliente, permitindo que as informações sejam acessadas de forma rápida e eficiente. É essencial compreender como esses protocolos funcionam para desenvolver sistemas para internet de alta qualidade.

2.0 INTRODUÇÃO À LINGUAGEM HTML

A linguagem HTML (Hypertext Markup Language) é a base para a construção de páginas web. É uma linguagem de marcação, ou seja, é usada para definir a estrutura e o conteúdo de uma página, utilizando Tags e atributos. As Tags são códigos que indicam ao navegador como exibir o conteúdo, como por exemplo o cabeçalho, parágrafo, imagem, etc. Já os atributos são utilizados para definir características específicas das Tags, como a cor de um texto ou o tamanho de uma imagem.

2.1 Evolução da linguagem HTML

A primeira versão do HTML foi criada em 1990 por Tim Berners-Lee, criador da World Wide Web. Era uma linguagem simples, com poucas Tags e sem suporte a estilos ou imagens. Com o passar dos anos, foram lançadas novas versões do HTML, com mais recursos e funcionalidades.

A versão 2.0 foi lançada em 1995 e trouxe suporte para frames (páginas divididas em várias partes) e formulários. Já a versão 3.2, lançada em 1997, introduziu o suporte para tabelas e folhas de estilo em cascata (CSS). Em 1999, foi lançada a versão 4.0, que incluiu suporte para camadas (divisão de uma página em várias áreas) e scripts.

A partir da versão 5, conhecida como HTML5, a linguagem passou por uma grande evolução, incorporando novos recursos como suporte a áudio e vídeo, geolocalização, armazenamento local e muitos outros. Além disso, a nova versão trouxe melhorias no suporte a dispositivos móveis e acessibilidade.

2.2 Versões e características do HTML

Atualmente, a versão mais recente do HTML é a 5.2, lançada em maio de 2017. Além de novos recursos, essa versão traz melhorias na semântica do código, tornando mais fácil para os motores de busca e leitores de tela entenderem o conteúdo da página.

Uma das características importantes do HTML é a sua capacidade de ser interpretado pelos navegadores de internet, que conseguem exibir o conteúdo das páginas de acordo com as Tags e atributos utilizados. Isso

permite que as páginas sejam acessadas por usuários de diferentes dispositivos e sistemas operacionais.

Outra característica importante é a possibilidade de separar o conteúdo da apresentação visual da página. Isso é feito através do uso de folhas de estilo em cascata (CSS), que permitem definir as cores, fontes, tamanhos, entre outros aspectos visuais das páginas.

2.3 Estrutura básica de um documento HTML

HTML, sigla para Hypertext Markup Language, é uma linguagem de marcação utilizada para criar conteúdo na web. A estrutura básica de um documento HTML é composta por duas partes: o cabeçalho (head) e o corpo (body).

O cabeçalho é a primeira seção do documento e contém informações importantes sobre a página, como o título, metadados e referências a outros recursos. Já o corpo é a seção onde é colocado todo o conteúdo da página, incluindo texto, imagens, vídeos, etc.

Um exemplo de estrutura básica de um documento HTML pode ser visto abaixo:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Título da página</title>
    </head>
    <body>
        Conteúdo da página
    </body>
</html>
```

Neste exemplo, a primeira linha declara o tipo de documento (doctype) como HTML. Em seguida, a tag <html> inicia o documento e contém as Tags <head> e <body>. O cabeçalho contém as Tags <title> e <meta>, enquanto o corpo contém as Tags <h1> e <p>, que são usadas para criar um título e um parágrafo, respectivamente.

2.4 Comentários em HTML

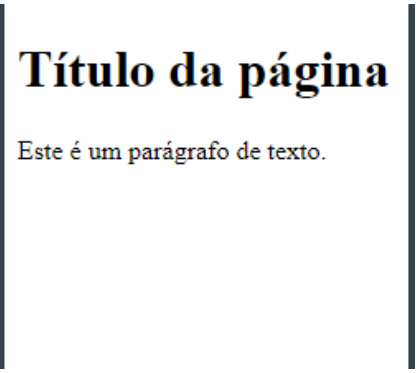
Os comentários são partes de um código que são ignoradas pelo interpretador da linguagem de programação e servem para documentar o código, explicar seu funcionamento e facilitar a sua manutenção.

Comentários bem escritos podem tornar o código mais legível e compreensível, principalmente quando outro programador precisa entender ou alterar o código. Além disso, eles ajudam a evitar erros de compreensão e a identificar possíveis bugs no código.

Em HTML, os comentários são definidos entre as Tags <!-- e -->. Tudo o que estiver entre essas duas Tags será ignorado pelo navegador. É importante lembrar que os comentários em HTML não podem ser aninhados, ou seja, um comentário dentro de outro comentário não é permitido.

Veja um exemplo de como criar um comentário em HTML:

```
<!DOCTYPE html>
<html>
<head>
    <title>Exemplo de comentário em HTML</title>
</head>
<body>
    <!-- Este é um comentário em HTML -->
    <h1>Título da página</h1>
    <p>Este é um parágrafo de texto.</p>
</body>
</html>
```



Nesse exemplo, o texto "Este é um comentário em HTML" é ignorado pelo navegador e não é exibido na página.

2.5 Tags HTML (parágrafos, títulos, links, imagens, listas)

A primeira linha do código HTML é chamada DOCTYPE e sua função é informar aos navegadores, mecanismos de busca, leitores de tela e outros dispositivos, qual é o tipo de documento que está sendo carregado. Isso é importante porque existem diferentes tipos de documentos que podem ser carregados, como o XML, por exemplo, e o Doctype avisa ao navegador como ele deve ler o código.

Logo após a tag DOCTYPE, começa o código HTML com a tag <html>, que indica que tudo que estiver entre as Tags <html></html> é escrito em HTML. Junto à palavra HTML há um atributo chamado "lang", que indica o idioma utilizado nos textos da página.

Em seguida, aparece a tag <head>, que contém informações importantes sobre a página, como o título do documento, especificado pela tag <title>, a tabela de caracteres que o navegador deve usar para renderizar o texto e outras informações que não são visíveis para o usuário.

O Google e outros mecanismos de busca utilizam a tag <title> para mostrar o título da página em seus resultados de busca, o que é muito importante para definir como o site será apresentado nos resultados de busca.

Após a tag de fechamento </head>, temos a tag <body>, que é onde todo o código HTML do restante do site é escrito e será visualizado pelo usuário.

É importante lembrar que as Tags HTML são apenas uma forma de marcar o conteúdo, e que a aparência final é definida por CSS. Portanto, é necessário ter uma boa compreensão de ambos para criar páginas web eficazes e esteticamente agradáveis.

2.6 Títulos e Subtítulos

Para gerar títulos e subtítulos em HTML, são utilizadas as Tags de cabeçalho (header) que variam de H1 a H6.

A tag H1 é a mais importante e geralmente é utilizada para o título principal da página, enquanto as Tags H2 a H6 são utilizadas como subtítulos e títulos secundários, em ordem decrescente de importância.

Os browsers tratam visualmente os títulos (headings) modificando seu tamanho. Exemplo:

```html
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <title>Exemplo de Título</title>
</head>
<body>
  <h1>Título principal da página</h1>
  <h2>Subtítulo</h2>
  <h3>Outro subtítulo</h3>
  <p>Conteúdo do parágrafo.</p>
  <h4>Título secundário</h4>
  <p>Outro conteúdo do parágrafo.</p>
</body>
</html>
```

Neste exemplo, o título principal da página é definido pela tag H1, enquanto os subtítulos são definidos pelas Tags H2 e H3. O conteúdo do site é escrito dentro da tag <body>.

2.7 Estilos de textos

Existem algumas Tags bastante úteis que permitem aplicar uma formatação a um trecho do texto, apenas adicionando as Tags de abertura e fechamento antes e depois do texto a ser formatado.

: envolve o texto em negrito (bold).

: envolve o texto em itálico (italics).

<u></u>: envolve o texto em sublinhado (underlined).

<del></del>: envolve o texto em riscado.

: envolve o texto em subscrito.

: envolve o texto em sobrescrito.

: centraliza o texto dentro de uma div.

Observe que as Tags <u> e foram descontinuadas na versão 5 da HTML e seu uso não é mais recomendado. Exemplo:

```
<!DOCTYPE html>
<html>
<body>
<p>Este é um texto <strong>em negrito</strong> e
<em>em itálico</em>.</p>
<p>Este é um texto <u>sublinhado</u> e
<del>riscado</del>.</p>
<p>Este é um texto <sub>subscrito</sub> e
<sup>sobrescrito</sup>.</p>
<div style="text-align:center;"><p>Este é um texto
centralizado.</p></div>
</body>
</html>
```

Este é um texto **em negrito** e *em itálico*.

Este é um texto sublinhado e ~~riscado~~.

Este é um texto $_{subscrito}$ e ^{sobrescrito}.

Este é um texto centralizado.

2.8 Listas

Em HTML, listas são usadas para organizar informações em formato de lista, permitindo que o conteúdo seja estruturado de forma mais clara e legível. Existem três tipos de listas em HTML: listas não ordenadas (ul), listas ordenadas (ol) e listas de definição (dl).

As listas não ordenadas são usadas quando a ordem dos itens não é importante, como em uma lista de tópicos. Para gerar uma lista não ordenada, utiliza-se a tag "ul" e, dentro dela, cada item da lista é criado com a tag "li". O exemplo abaixo mostra como criar uma lista não ordenada:

```
<ul>
  <li>Item 1</li>
  <li>Item 2</li>
  <li>Item 3</li>
</ul>
```

As listas ordenadas são usadas quando a ordem dos itens é importante, como em uma lista de etapas de um processo. Para gerar uma lista ordenada, utiliza-se a tag "ol" e, dentro dela, cada item da lista é criado com a tag "li". O exemplo abaixo mostra como criar uma lista ordenada:

```
<ol>
  <li>Passo 1</li>
  <li>Passo 2</li>
  <li>Passo 3</li>
</ol>
```

As listas de definição são usadas para definir termos ou palavras e seus significados. Para gerar uma lista de definição, utiliza-se a tag "dl". Dentro

dela, cada termo é definido com a tag "dt" e cada definição é criada com a tag "dd". O exemplo abaixo mostra como criar uma lista de definição:

```
<dl>
  <dt>HTML</dt>
  <dd>Linguagem de marcação usada para criar páginas web.</dd>
  <dt>CSS</dt>
  <dd>Linguagem de estilo usada para definir a aparência de páginas web.</dd>
</dl>
```

2.9 Imagens

Formatos de imagens

As imagens são elementos fundamentais na criação de sites, aplicativos e outros projetos de desenvolvimento de software. No entanto, existem diferentes tipos de arquivos de imagem, cada um com suas próprias características e formatos de arquivo. Neste texto, vamos explorar os tipos mais comuns de arquivos de imagem, suas extensões e alguns sites onde você pode encontrar imagens gratuitas para seus projetos.

JPEG: o formato JPEG é amplamente utilizado na web e é compatível com a maioria dos dispositivos e navegadores. Esse tipo de arquivo é otimizado para armazenar imagens com muitos detalhes e cores, mas com menor qualidade de imagem. A extensão de um arquivo JPEG é .jpg ou .jpeg.

PNG: o formato PNG é usado para imagens com áreas transparentes e é preferido quando você precisa de alta qualidade de imagem e pode lidar com tamanhos de arquivo maiores. É frequentemente usado para imagens de logotipos e gráficos vetoriais. A extensão de um arquivo PNG é .png.

GIF: o formato GIF é geralmente usado para imagens animadas e é compatível com quase todos os navegadores. É possível usar o formato GIF para criar pequenas animações ou ícones animados. A extensão de um arquivo GIF é .gif.

Além desses, existem outros tipos de arquivos de imagem, como BMP, TIFF e SVG, cada um com suas próprias características.

Existem vários sites de imagens gratuitas disponíveis na internet, incluindo:

Unsplash (https://unsplash.com/)

Pexels (https://www.pexels.com/)

Pixabay (https://pixabay.com/)

Freepik (https://www.freepik.com/)

Canva (https://www.canva.com/photos/free/)

Esses sites oferecem uma grande variedade de imagens em diferentes resoluções e formatos. É importante lembrar que algumas imagens podem ter restrições de uso, então é sempre bom verificar os termos e condições de cada site antes de usá-las em seus projetos.

Escolher o tipo correto de arquivo de imagem pode afetar a qualidade e o desempenho do seu projeto, portanto, é importante ter conhecimento dos diferentes tipos de arquivos disponíveis. Além disso, sempre procure por sites que ofereçam imagens gratuitas de alta qualidade para seus projetos.

Inserção de imagens em projeto

Para inserir uma imagem em uma página HTML, você precisa usar a tag HTML <img>. O atributo src é obrigatório e deve especificar o caminho para a imagem no seu servidor web ou em um servidor externo. O atributo alt também é recomendado e fornece um texto alternativo para a imagem, que é útil para pessoas com deficiência visual ou para usuários que têm problemas para carregar imagens.

Aqui está um exemplo básico de código HTML para inserir uma imagem:

```
<img src="caminho/para/sua/imagem.jpg" alt="Descrição da imagem">
```

Note que a extensão do arquivo (neste caso, ".jpg") deve ser incluída no caminho da imagem.

Além disso, você também pode usar os atributos opcionais como height e width para definir as dimensões da imagem na página. Por exemplo:

```
<img src="caminho/para/imagem.jpg"
alt="Descrição da imagem" height="300"
width="400">
```

Lembre-se de sempre especificar o tamanho correto da imagem para evitar problemas de carregamento da página e garantir a melhor experiência do usuário.

2.10 Links em Html

Os links em HTML são elementos que permitem a conexão entre páginas da web. Eles podem ser usados para direcionar o usuário para outra página, outra seção da mesma página ou até mesmo para abrir um e-mail ou arquivo para download. Os links são criados por meio da tag <a> (âncora), que envolve o texto que será clicável ou uma imagem que servirá como botão.

Existem três tipos principais de links em HTML:

Links de e-mail: são aqueles que abrem um programa de e-mail com o endereço do destinatário já preenchido. Para criar um link de e-mail é necessário atribuir o endereço de e-mail ao atributo href, precedido pelo prefixo mailto:. Por exemplo:

```
<a href="mailto:exemplo@exemplo.com">Enviar um e-mail</a>
```

Nesse caso, o link abre um programa de e-mail com o endereço exemplo@exemplo.com já preenchido.

Links internos: são aqueles que levam o usuário para outra seção da mesma página. Para criar um link interno, é necessário atribuir um valor ao atributo href com um identificador para a seção que será acessada. Por exemplo:

```
<a href="#sobre">Ir para a seção sobre</a>
```

Nesse caso, o link leva o usuário para a seção com o identificador #sobre. Essa seção deve ser criada com a tag <section> e ter o mesmo identificador atribuído a ela, como no exemplo abaixo:

```
<section id="sobre">
  <h2>Sobre nós</h2>
  <p>Aqui vai o conteúdo sobre a empresa.</p>
</section>
```

Links externos: são aqueles que levam o usuário para outra página na web. Para criar um link externo, é necessário atribuir o endereço da página ao atributo href. Por exemplo:

```
<a href="https://www.google.com">Ir para o Google</a>
```

Nesse caso, o link leva o usuário para a página do Google.

Existem algumas variações de tipos de links, como o com uso de imagem. Para criar um link para uma imagem em HTML, podemos utilizar a tag <img> dentro de uma tag <a>. O atributo src da tag <img> indica o caminho da imagem, e o atributo href da tag <a> indica o endereço para onde o usuário será redirecionado quando clicar na imagem. Exemplo:

```
<a href="https://www.exemplo.com/arquivo_imagem.jpg">
  <img src="https://www.exemplo.com/imagem.jpg" alt="Descrição da imagem">
</a>
```

Para criar um link para download, podemos utilizar a tag <a> com o atributo download. O valor desse atributo indica o nome do arquivo que será baixado pelo usuário. O atributo href indica o caminho do arquivo. Exemplo:

```
<a href="https://www.exemplo.com/arquivo.pdf" download="documento.pdf">
  Baixar documento
</a>
```

Além desses tipos, existem outras variações de links em HTML, como os links de download e os links de telefone, que abrem o discador com o número já preenchido.

Segundo o autor Jon Duckett, em seu livro "HTML and CSS: Design and Build Websites", os links são um dos elementos mais importantes da web, pois permitem a conexão entre páginas e a navegação dos usuários de maneira rápida e intuitiva.

3.0 ESTRUTURAÇÃO E FORMATAÇÃO DE CONTEÚDO COM HTML

O HTML é uma linguagem de marcação que permite estruturar e formatar conteúdo de forma organizada e semântica. Duas tags importantes para a divisão e organização do conteúdo são a div e a section.

3.1 Seções e Divisórias

A tag div é utilizada para dividir o conteúdo em seções genéricas, sem uma finalidade semântica específica. É comum utilizá-la para agrupar elementos HTML e aplicar estilos CSS a eles. Veja um exemplo:

```
<div>
  <h2>Título da seção</h2>
  <p>Este é um parágrafo de exemplo.</p>
  <ul>
    <li>Item 1</li>
    <li>Item 2</li>
  </ul>
</div>
```

Neste exemplo, estamos utilizando a tag div para agrupar um título (h2), um parágrafo (p) e uma lista (ul). Note que o conteúdo dentro da div não tem uma finalidade semântica específica, sendo utilizada apenas para a organização e formatação visual do conteúdo.

Já a tag section é utilizada para dividir o conteúdo em seções com uma finalidade semântica específica. É comum utilizar esta tag para dividir o conteúdo em seções como cabeçalho, conteúdo principal, rodapé, entre outros. Veja um exemplo:

```
<section>
  <h1>Cabeçalho da página</h1>
  <nav>
    <ul>
      <li><a href="#">Link 1</a></li>
      <li><a href="#">Link 2</a></li>
      <li><a href="#">Link 3</a></li>
    </ul>
  </nav>
</section>
<section>
  <h2>Conteúdo principal</h2>
  <p>Este é um parágrafo de exemplo.</p>
</section>
<section>
  <h2>Rodapé</h2>
  <p>Este é um parágrafo de exemplo para o rodapé.</p>
</section>
```

Neste exemplo, estamos utilizando a tag section para dividir o conteúdo em três seções com finalidades semânticas específicas: cabeçalho, conteúdo principal e rodapé. Note que cada section contém elementos HTML diferentes, de acordo com sua finalidade semântica.

A utilização correta das tags div e section ajuda a organizar e estruturar o conteúdo de uma página HTML, facilitando a manutenção e a aplicação de estilos CSS. Além disso, o uso de tags semânticas como a section ajuda na acessibilidade do conteúdo para pessoas que utilizam tecnologias assistivas, como leitores de tela.

3.2 Tabelas

As tabelas em HTML são uma forma de exibir dados em forma de grade, com linhas e colunas. A tag <table> é usada para definir uma tabela, e as células são definidas com as Tags <tr> (table row), <th> (table heading) e <td> (table data).

A tag <caption> é usada para definir um título para a tabela, que é exibido acima da tabela. Por exemplo:

```html
<table border="1">
  <tr>
    <th>Nome</th>
    <th>Sobrenome</th>
    <th>Idade</th>
  </tr>
  <tr>
    <td>João</td>
    <td>Silva</td>
    <td>30</td>
  </tr>
  <tr>
    <td>Maria</td>
    <td>Souza</td>
    <td>25</td>
  </tr>
</table>
```

Nome	Sobrenome	Idade
João	Silva	30
Maria	Souza	25

A tag <tr> é usada para definir uma nova linha na tabela. Cada linha é composta por células, que são definidas com as Tags <th> ou <td>. É importante lembrar que cada linha deve ser fechada com </tr>.

A tag <th> é usada para definir as células de cabeçalho da tabela, que geralmente são exibidas em negrito. Essas células podem ficar sozinhas em uma coluna ou linha, ou haver várias células de cabeçalho em uma mesma coluna ou linha.

A tag <td> é usada para definir as células de dados da tabela, que são exibidas em fonte normal e alinhadas à esquerda. Assim como com as células de cabeçalho, as células de dados também podem ficar sozinhas em uma coluna ou linha, ou haver várias células em uma mesma coluna ou linha. Exemplo completo de tabela em HTML:

```html
<table>
  <tr>
    <td>linha 1, coluna 1</td>
    <td>linha 1, coluna 2</td>
  </tr>
  <tr>
    <td>linha 2, coluna 1</td>
    <td>linha 2, coluna 2</td>
  </tr>
</table>
```

3.3 Bordas

As bordas em tabelas HTML são linhas que circundam as células e colunas de uma tabela para separá-las e torná-las mais legíveis. Elas podem ser adicionadas, removidas ou modificadas usando as propriedades CSS.

Para criar uma tabela com bordas, você precisa adicionar a propriedade "border" à tag <table>. O valor atribuído à propriedade border define a largura da borda em pixels. Por exemplo:

```
<table border="1">
  <tr>
    <td>linha 1, coluna 1</td>
    <td>linha 1, coluna 2</td>
  </tr>
  <tr>
    <td>linha 2, coluna 1</td>
    <td>linha 2, coluna 2</td>
  </tr>
</table>
```

Este exemplo cria uma tabela com bordas de 1 pixel de largura ao redor de cada célula.

As bordas podem ser estilizadas de várias maneiras usando CSS, incluindo diferentes estilos de linha, cores, larguras e cantos arredondados. A escolha do estilo de borda depende do design visual que você deseja para sua tabela.

Align: é uma propriedade que define o alinhamento horizontal do conteúdo dentro de uma célula da tabela. Pode ser definido como "left" (esquerda), "center" (centro) ou "right" (direita). Exemplo:

```
<table>
  <tr>
    <td align="left">Conteúdo alinhado à esquerda</td>
    <td align="center">Conteúdo centralizado</td>
    <td align="right">Conteúdo alinhado à direita</td>
  </tr>
</table>
```

Valign: é uma propriedade que define o alinhamento vertical do conteúdo dentro de uma célula da tabela. Pode ser definido como "top" (topo), "middle" (meio) ou "bottom" (fundo). Exemplo:

```
<table>
  <tr>
    <td valign="top">Conteúdo alinhado ao topo</td>
    <td valign="middle">Conteúdo alinhado ao meio</td>
    <td valign="bottom">Conteúdo alinhado ao fundo</td>
  </tr>
</table>
```

Rowspan e colspan: são propriedades que permitem unir células de uma tabela em linhas ou colunas. Rowspan é usado para mesclar células em uma mesma linha, enquanto colspan é usado para mesclar células em uma mesma coluna. O valor definido nessas propriedades é a quantidade de células que serão mescladas. Exemplo:

```
<table>
  <tr>
    <td rowspan="2">Célula mesclada</td>
    <td>Célula 1</td>
    <td>Célula 2</td>
  </tr>
  <tr>
    <td colspan="2">Células mescladas</td>
  </tr>
</table>
```

Largura da célula: é uma propriedade que define a largura de uma célula da tabela em pixels ou em porcentagem do tamanho da tabela. Pode ser definido usando a propriedade "width". Exemplo:

```
<table>
  <tr>
    <td width="100">Célula com largura de 100 pixels</td>
    <td width="50%">Célula com largura de 50% da tabela</td>
  </tr>
</table>
```

3.4 Formulários

Em HTML, é possível criar formulários que permitem aos usuários interagir com o servidor, inserindo informações em campos, clicando em botões e enviando informações. O elemento "form" é responsável por permitir essa interação. Ele permite a criação de formulários com janelas de entrada de textos, botões e outras opções.

Para criar um formulário, todos os elementos devem ser colocados entre as Tags <form> </form>. Dentro da tag "form", deve ser inserido o atributo "method", que determina como os dados serão enviados após o usuário salvar o formulário. Existem dois valores possíveis para o atributo "method":

GET: Envia os dados adicionados aos campos do formulário associados à URL da página, suportando até 128 caracteres.

POST: É o método mais utilizado e envia cada dado de forma separada da URL. Com este método, as informações adicionadas aos campos fazem parte do corpo da mensagem enviada para o servidor e podem transferir grandes quantidades de dados.

Também podem ser adicionados os atributos "id" e "name" ao formulário. Esses atributos podem ter os mesmos valores e servem para identificar o formulário na página e no servidor, respectivamente.

Exemplo de código HTML para um formulário simples com o método POST:

```
<form method="POST" id="myForm" name="myForm">
  <label for="name">Nome:</label>
  <input type="text" id="name" name="name"><br><br>

  <label for="email">Email:</label>
  <input type="email" id="email" name="email"><br><br>

  <input type="submit" value="Enviar">
</form>
```

No exemplo acima, o formulário possui dois campos para o usuário preencher: "Nome" e "Email". O atributo "for" no elemento "label" é usado para associar o texto da label ao campo correspondente. O elemento "input" é usado para criar os campos de entrada de texto. O atributo "type" especifica o tipo de entrada, "id" e "name" são usados para identificar os campos e "value" é usado para especificar o texto do botão de envio.

A TAG <input> é usada em HTML para criar um elemento de formulário interativo que permite ao usuário inserir dados ou selecionar uma opção. Existem vários tipos de elementos <input>, cada um com um propósito diferente. Abaixo estão alguns exemplos:

O elemento <input type="text"> cria uma caixa de texto na qual o usuário pode inserir um texto ou um número. Exemplo:

```
<label for="nome">Nome:</label>
<input type="text" id="nome" name="nome">
```

O elemento <input type="password"> cria uma caixa de texto que mascara o que o usuário está digitando, para que a senha fique oculta. Exemplo:

```
<label for="senha">Senha:</label>
<input type="password" id="senha" name="senha">
```

O elemento <input type="radio"> cria um botão de opção que permite que o usuário selecione uma opção dentre um conjunto de opções mutuamente exclusivas. Exemplo:

```
<label for="opcao1"><input type="radio"
id="opcao1"
name="opcao" value="1"> Opção 1</label>

<label for="opcao2"><input type="radio"
id="opcao2"
name="opcao" value="2"> Opção 2</label>
```

○ Opção 1 ○ Opção 2

O elemento <input type="checkbox"> cria uma caixa de seleção que permite que o usuário selecione uma ou mais opções dentre um conjunto de opções independentes. Exemplo:

```
<label for="opcao1"><input type="checkbox"
id="opcao1" name="opcao1" value="1"> Opção
1</label>
<label for="opcao2"><input type="checkbox"
id="opcao2" name="opcao2" value="2"> Opção
2</label>
```

☐ Opção 1 ☐ Opção 2

O elemento <input type="submit"> cria um botão que envia o formulário para o servidor. Exemplo:

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

O elemento <input type="reset"> cria um botão que redefine todos os campos do formulário para seus valores padrão. Exemplo:

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

Limpar

A **TAG <textarea>** é usada para criar uma área de texto multilinhas, onde o usuário pode inserir um texto mais longo. É útil quando é necessário receber informações que requerem mais espaço do que uma simples caixa de texto. Por exemplo:

```
<label for="comentarios">Comentários:
</label>
<textarea id="comentarios"
name="comentarios" rows="5" cols="50">
Escreva aqui seus comentários...
</textarea>
```

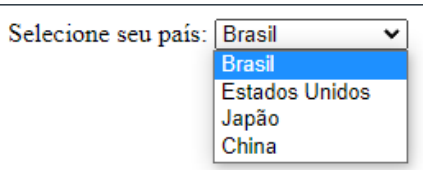
Comentários:

Escreva aqui seus comentários...

Nesse exemplo, a TAG <textarea> cria uma área de texto com 5 linhas e 50 colunas. O atributo id e name são usados para identificar o campo na página e no servidor.

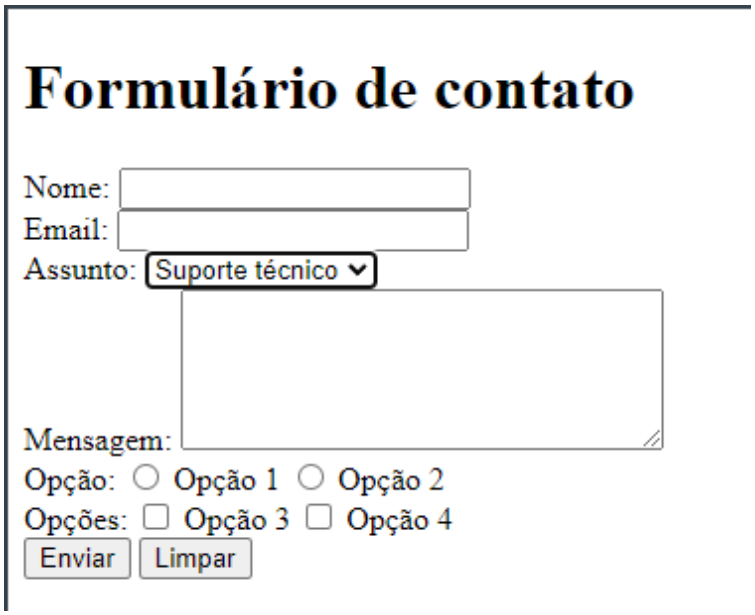
Já a **TAG <select>** é usada para criar uma lista suspensa de opções, onde o usuário pode selecionar uma única opção. É útil quando é necessário apresentar uma lista de opções para o usuário escolher. Por exemplo:

```
<label for="pais">Selecione seu país:
</label>
<select id="pais" name="pais">
  <option value="brasil">Brasil</option>
  <option value="eua">Estados
Unidos</option>
  <option value="japao">Japão</option>
  <option value="china">China</option>
</select>
```



Nesse exemplo, a TAG <select> cria uma lista suspensa com 4 opções. O atributo id e name são usados para identificar o campo na página e no servidor. Cada opção é criada com a TAG <option>, que contém o valor da opção e o texto que será exibido para o usuário.

Exemplo de Formulário Completo



```html
<!DOCTYPE html>
<html>
<head>
    <title>Exemplo de formulário em HTML</title>
</head>
<body>
    <h1>Formulário de contato</h1>
    <form action="#" method="POST">
        <label for="nome">Nome:</label>
        <input type="text" id="nome" name="nome" required><br>

        <label for="email">Email:</label>
        <input type="email" id="email" name="email" required><br>

        <label for="assunto">Assunto:</label>
        <select id="assunto" name="assunto">
            <option value="suporte">Suporte técnico</option>
            <option value="vendas">Vendas</option>
            <option value="financeiro">Financeiro</option>
        </select><br>

        <label for="mensagem">Mensagem:</label>
        <textarea id="mensagem" name="mensagem" rows="5"
cols="30"></textarea><br>
```

```html
        <label for="opcao">Opção:</label>
        <input type="radio" id="opcao1" name="opcao" value="1">
        <label for="opcao1">Opção 1</label>
        <input type="radio" id="opcao2" name="opcao" value="2">
        <label for="opcao2">Opção 2</label><br>

        <label for="opcoes">Opções:</label>
        <input type="checkbox" id="opcao3" name="opcoes[]"
value="3">
        <label for="opcao3">Opção 3</label>
        <input type="checkbox" id="opcao4" name="opcoes[]"
value="4">
        <label for="opcao4">Opção 4</label><br>

        <input type="submit" value="Enviar">
        <input type="reset" value="Limpar">
    </form>
</body>
</html>
```

4.0 A LINGUAGEM CSS

O CSS (Cascading Style Sheets) é uma linguagem utilizada para estilizar e formatar documentos HTML, tornando-os visualmente atraentes e mais acessíveis. O CSS evoluiu ao longo do tempo, adicionando novas funcionalidades e melhorando a experiência do usuário.

Inicialmente, o CSS foi criado em 1996 pelo W3C (World Wide Web Consortium) com o objetivo de separar o conteúdo da apresentação em

páginas da Web. Na época, a linguagem era bastante limitada, com poucas propriedades de estilo e compatibilidade limitada entre navegadores.

Com a evolução do CSS, surgiram novas versões da linguagem, adicionando novas propriedades de estilo e recursos mais avançados. Em 1998, o CSS1 foi lançado, seguido pelo CSS2 em 1999, que trouxe melhorias significativas em relação à versão anterior. Em 2011, o CSS3 foi lançado, adicionando ainda mais recursos, como animações, transições e transformações.

Atualmente, o CSS é amplamente utilizado na construção de sites e aplicativos web, permitindo a criação de layouts responsivos e acessíveis em diferentes dispositivos e plataformas. Com a evolução da linguagem, o CSS continua a crescer e a se adaptar às necessidades do mundo digital.

4.1 Versões e características do CSS:

O CSS (Cascading Style Sheets) é uma linguagem utilizada para estilizar documentos HTML, permitindo a criação de layouts e estilos visuais. Ao longo do tempo, várias versões do CSS foram lançadas, adicionando novas propriedades de estilo e recursos mais avançados.

O CSS1 foi a primeira versão oficial do CSS, lançada em 1996 pelo W3C. Essa versão introduziu propriedades de estilo básicas, como cor, fonte, margens e preenchimento. O CSS2, lançado em 1999, adicionou novas propriedades, como posicionamento e layout de página. O CSS3, lançado em 2011, trouxe ainda mais recursos, como transições, animações e transformações.

Algumas das principais características do CSS incluem:

Separação de conteúdo e apresentação: o CSS permite que o conteúdo e a apresentação de uma página sejam separados, facilitando a manutenção e atualização do site.

Flexibilidade e controle: o CSS oferece controle preciso sobre a aparência e o layout de uma página, permitindo que os desenvolvedores criem layouts responsivos e personalizados.

Compatibilidade com diferentes dispositivos e plataformas: o CSS é compatível com uma ampla variedade de dispositivos e plataformas, permitindo que os sites sejam acessíveis em diferentes dispositivos e navegadores.

Modularidade: o CSS permite que os estilos sejam organizados em diferentes arquivos, tornando a manutenção do código mais fácil e eficiente.

4.2 Para que serve o CSS?

O CSS é utilizado para separar a estrutura de um documento HTML da sua apresentação visual. Ele permite que os desenvolvedores criem regras de estilo que se aplicam a elementos específicos de uma página, como cor, fonte, tamanho, layout, entre outros.

Além disso, o uso do CSS torna a manutenção do código mais fácil e eficiente, já que permite alterar a aparência de um site inteiro apenas modificando uma folha de estilos em vez de ter que editar cada elemento individualmente.

4.3 Sintaxe CSS

Cada estilo criado é definido como uma regra CSS, e cada regra deve seguir uma estrutura específica:

elemento {atributo1: valor; atributo2: valor;}

elemento: Descreve o elemento de design que receberá o estilo. Ele é referenciado pelo nome do elemento HTML sem os sinais de maior e menor. Por exemplo, "h1" ou "div".

atributo: Especifica o aspecto do elemento que será modificado. Deve ser um nome de propriedade CSS válido, como "font-size" ou "background-color".

Valor: Define a configuração que será aplicada à propriedade. Deve ser uma configuração válida para a propriedade em questão, como "12px" para font-size ou "red" para color.

Atributo: Valor - É a combinação de uma propriedade e seu valor, separados por dois pontos. Pode-se atribuir múltiplas declarações e separá-las com ponto-e-vírgula (;).Exemplos:

Para definir o tamanho da fonte dos títulos de um site como 20 pixels, a regra CSS seria:

```
h1 {
font-size: 20px;
}

```

4.4 Valores e Tamanhos

Unidades de Medidas

As unidades de medidas são utilizadas na criação de sites para definir o tamanho e a posição dos elementos na página. As unidades mais comuns são pixels (px), em (em), rem (rem) e porcentagem (%).

O pixel (px) é a unidade de medida mais comum, que representa um ponto na tela. É uma medida fixa e não é afetada pelo tamanho da tela do dispositivo. É amplamente utilizado para definir tamanhos de fontes, margens, espaçamentos, larguras e alturas de elementos.

Já o em (em) é uma unidade de medida relativa que é baseada no tamanho da fonte do elemento pai. Por exemplo, se o tamanho da fonte do elemento pai for definido como 16px, um valor de 1em será igual a 16px. Essa unidade é muito utilizada para definir tamanhos de fontes, espaçamentos e larguras em relação ao tamanho da fonte.

O rem (rem) é semelhante ao em, mas é baseado no tamanho da fonte do elemento raiz (geralmente o elemento HTML). Essa unidade é especialmente útil para sites responsivos, pois permite que os elementos sejam dimensionados de forma consistente em todas as resoluções de tela.

Por fim, a porcentagem (%) é outra unidade relativa que é baseada no tamanho do elemento pai. Por exemplo, se o tamanho da largura do elemento pai for definido como 100%, um valor de 50% será igual a metade

da largura do elemento pai. Essa unidade é frequentemente usada para definir larguras, alturas e margens em relação ao tamanho do elemento pai.

Tipos de folhas de estilo

No desenvolvimento de páginas web, é possível definir regras de CSS em três lugares distintos, que afetam a forma como os estilos são aplicados à página. Esses três lugares são:

A. Estilos externos: as regras de CSS são definidas em um arquivo externo, que é vinculado ao documento HTML por meio do elemento <link> no cabeçalho da página. Esse arquivo contém todas as regras de estilo para o site e é referenciado em todas as páginas HTML que o compõem.

Um exemplo prático de estilos externos seria criar um arquivo CSS chamado "estilos.css" e colocar nele todas as regras de estilo para o site, como por exemplo:

```css
body {
  font-family: Arial, sans-serif;
  background-color: #f2f2f2;
}

h1 {
  color: #333;
  font-size: 36px;
}

p {
  line-height: 1.5;
}
```

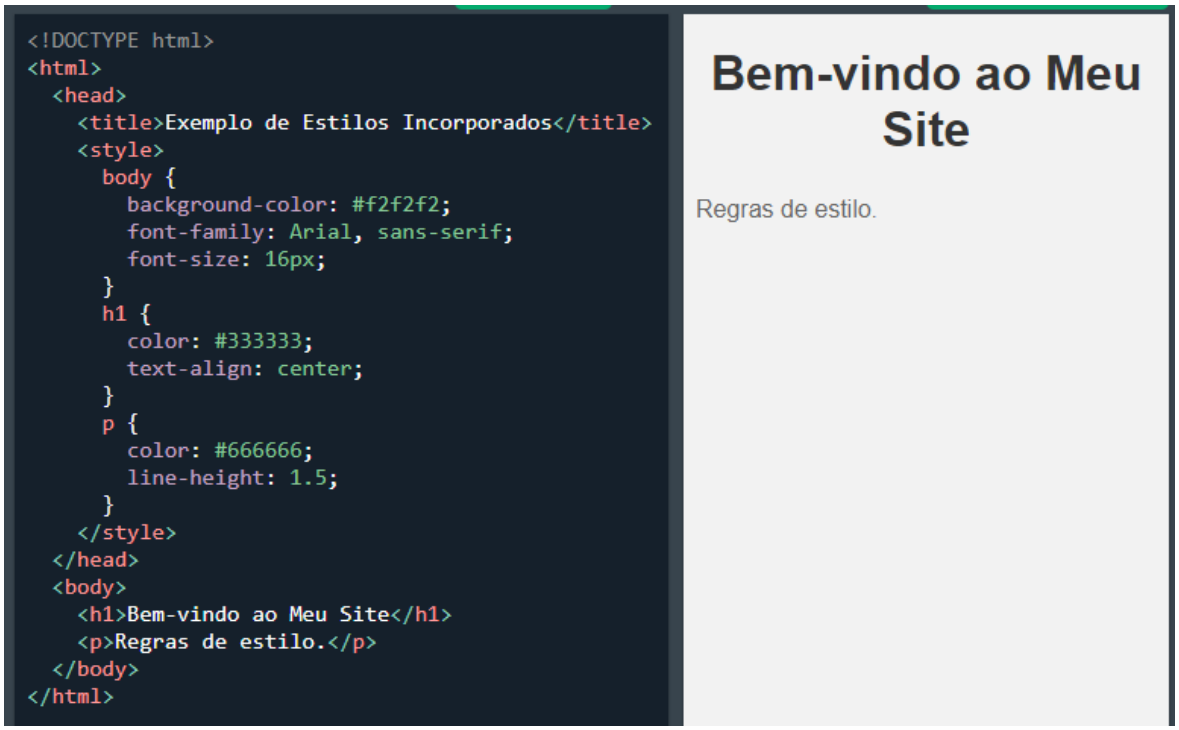
Em seguida, no cabeçalho de cada página HTML do site, seria adicionado o seguinte código para vincular o arquivo de estilo externo:

```html
<head>
  <meta charset="UTF-8">
  <title>Minha página</title>
  <link rel="stylesheet" href="estilos.css">
</head>
```

Dessa forma, todas as regras de estilo contidas no arquivo "estilos.css" seriam aplicadas a essa página HTML e a qualquer outra página que referencie esse arquivo externo. Isso torna a manutenção do estilo do site mais fácil e organizada, pois todas as regras estão contidas em um único arquivo.

B. Estilos incorporados: as regras de CSS são definidas dentro do cabeçalho de um documento HTML usando o elemento <style>. Esse método permite que as regras de estilo sejam aplicadas apenas ao documento HTML atual.

Um exemplo de estilos incorporados pode ser visto no seguinte código HTML:



```
<!DOCTYPE html>
<html>
  <head>
    <title>Exemplo de Estilos Incorporados</title>
    <style>
      body {
        background-color: #f2f2f2;
        font-family: Arial, sans-serif;
        font-size: 16px;
      }
      h1 {
        color: #333333;
        text-align: center;
      }
      p {
        color: #666666;
        line-height: 1.5;
      }
    </style>
  </head>
  <body>
    <h1>Bem-vindo ao Meu Site</h1>
    <p>Regras de estilo.</p>
  </body>
</html>
```


Neste exemplo, as regras de estilo estão definidas dentro do cabeçalho da página HTML, usando o elemento <style>. As regras aplicam estilos diferentes aos elementos body, h1 e p, como a cor do texto, tamanho da fonte, alinhamento de texto, entre outros. Essas regras serão aplicadas somente a esse documento HTML e não a outros documentos que possam ser vinculados ao mesmo arquivo CSS externo.

C. Estilos inline: as regras de CSS são definidas diretamente em uma tag HTML usando o atributo style. Esse método permite que as regras de estilo sejam aplicadas apenas ao elemento HTML em que foram definidas.

Um exemplo de estilo inline usando o atributo style em uma tag HTML seria o seguinte: Um exemplo de estilo inline usando o atributo style em uma tag HTML seria o seguinte:

```
<p style="color: blue; font-size:
18px;">Este é um parágrafo com estilo
inline.</p>
```

Este é um parágrafo com estilo inline.

Neste exemplo, a cor do texto é definida como azul e o tamanho da fonte como 18 pixels. Essas regras de estilo são aplicadas apenas ao elemento p em que foram definidas e não são afetadas por quaisquer estilos externos ou incorporados. Este método é útil para aplicar estilos a elementos individuais de uma página HTML. No entanto, ele pode se tornar trabalhoso se você tiver muitos elementos para estilizar, pois requer que você defina as regras de estilo individualmente para cada elemento.

A interação entre as regras de estilo definidas nos três lugares é regida pelo conceito de "cascata", que define a prioridade das regras quando há conflitos entre elas.

Por exemplo, suponha que haja uma regra de estilo para definir a cor do texto de um elemento <p>, definida no estilo externo e outra regra de estilo para definir a mesma propriedade, mas com uma cor diferente, definida no estilo inline. Nesse caso, a regra inline terá prioridade e será aplicada à página, ignorando a regra externa.

A definição de regras de CSS em diferentes lugares em uma página web permite maior flexibilidade e controle na estilização do site, possibilitando que diferentes elementos e documentos HTML tenham estilos específicos e personalizados.

4.5 Comentários CSS

Comentários em CSS são uma parte importante do processo de desenvolvimento de estilos em uma página da web. Eles permitem que os desenvolvedores documentem o código e descrevam o que está acontecendo em cada linha. Comentários podem ser usados para explicar as escolhas de design, descrever as soluções para problemas específicos ou simplesmente adicionar notas para si mesmos ou outros desenvolvedores que possam trabalhar com o mesmo código.

Segundo Lucas Mazza, autor de "HTML5 e CSS3 - Domine a Web do Futuro": "Comentários são uma ferramenta importante em qualquer linguagem de programação e é uma boa prática utilizá-los para documentar o seu código CSS".

Para adicionar um comentário em CSS, basta utilizar a sintaxe /* texto do comentário /. Tudo o que estiver dentro dos caracteres / e */ será considerado um comentário e ignorado pelo navegador.

Por exemplo, se quisermos comentar sobre o uso de uma determinada cor em uma regra CSS, poderíamos adicionar o seguinte comentário:

```css
/* Esta cor foi escolhida para aumentar a visibilidade do botão */
button {
  background-color: #FF5733;
  color: white;
  padding: 10px 20px;
  border: none;
}
```

Dessa forma, outros desenvolvedores que trabalham no mesmo projeto poderão entender facilmente o motivo pelo qual essa cor foi escolhida e os pensamentos por trás do design. Comentários em CSS são uma prática recomendada para garantir que o código seja fácil de entender e manter a longo prazo.

4.6 Classes de estilos

As classes de estilo em CSS servem para agrupar um conjunto de propriedades CSS e aplicá-las a vários elementos HTML sem a necessidade

de definir as propriedades repetidamente para cada elemento individualmente. Isso torna o processo de estilização muito mais eficiente e organizado, especialmente em projetos maiores.

Para criar uma classe de estilo, basta definir um seletor de classe usando um ponto antes do nome da classe e, em seguida, adicionar as propriedades CSS desejadas. Por exemplo, para criar uma classe de estilo chamada "destaque" que define a cor de fundo como amarelo e o tamanho da fonte como 18px, podemos escrever o seguinte código CSS:

```css
.destaque {
  background-color: yellow;
  font-size: 18px;
}
```

Em seguida, podemos aplicar essa classe de estilo a qualquer elemento HTML usando o atributo "class". Por exemplo, para aplicar a classe de estilo "destaque" a um elemento de parágrafo, podemos escrever o seguinte código HTML:

```html
<p class="destaque">Este parágrafo está destacado</p>
```

A importância das classes de estilo é destacada por M.E. Davis e J.A. Phillips em "Aprendendo PHP e MySQL" ao afirmar que "as classes são especialmente úteis em documentos com várias páginas ou seções que requerem formatação semelhante" (Davis e Phillips, 2013, p. 196). Lucas Mazza, em "HTML5 e CSS3 - Domine a Web do Futuro", também destaca a importância das classes de estilo na organização do código CSS, afirmando que "as classes são uma forma excelente de manter o CSS organizado e reutilizável" (Mazza, 2013, p. 163).

Além disso, as classes de estilo também permitem que os desenvolvedores apliquem estilos diferentes a elementos semânticos idênticos, como links e botões.

4.7 Seletor ID

O seletor ID é uma das formas de selecionar elementos HTML com CSS, permitindo estilizar um elemento de forma única. Ele é representado pelo

caractere "#" seguido do nome do ID. Ao utilizar o seletor ID, é possível aplicar estilos exclusivos para um único elemento, sem interferir nos demais elementos que possam ter estilos semelhantes.

De acordo com Mazza (2013), o seletor ID é uma forma eficiente de estilizar um elemento de forma única, principalmente quando se trata de elementos que possuem uma função específica em uma página. Por exemplo, se há uma seção de notícias em um site e deseja-se estilizar o título dessa seção, pode-se utilizar o seletor ID para selecionar o elemento que contém o título e aplicar um estilo exclusivo.

Para criar um seletor ID, basta atribuir um valor único ao atributo "id" do elemento HTML. Por exemplo:

```html
<div id="titulo-secao">
  <h1>Título da Seção</h1>
</div>
```

No CSS, o seletor ID é representado pelo caractere "#" seguido do valor do atributo "id". Por exemplo:

```css
#titulo-secao {
  font-size: 24px;
  color: #333;
  font-weight: bold;
}
```

Nesse exemplo, o seletor ID "titulo-secao" foi utilizado para aplicar um estilo exclusivo ao elemento que possui esse ID. O estilo aplicado inclui uma fonte de tamanho 24 pixels, uma cor de texto #333 e um peso de fonte em negrito.

É importante lembrar que, por ser um seletor muito específico, o uso excessivo de IDs pode tornar o código CSS difícil de manter e atualizar. Portanto, é recomendado utilizar os seletores IDs de forma moderada e estratégica, reservando-os para casos em que é realmente necessário aplicar um estilo único a um elemento específico.

4.8 Utilizando a TAG

A tag é uma das mais importantes na linguagem HTML e é frequentemente usada em conjunto com o CSS para estruturar o layout de uma página da web. O termo "div" é uma abreviação de "division" e representa uma divisão lógica do conteúdo de uma página.

No CSS, a tag pode ser usada como um seletor de estilo para aplicar estilos a todos os elementos que são marcados com essa tag. Isso é feito atribuindo uma classe ou um ID à tag e, em seguida, aplicando as regras de estilo correspondentes no arquivo CSS.

Ao usar a tag no CSS, é importante lembrar que seu principal propósito é criar contêineres para outros elementos. Isso significa que você pode colocar outros elementos HTML dentro de uma tag e, em seguida, aplicar regras de estilo para essa tag para modificar o layout e o visual desses elementos.

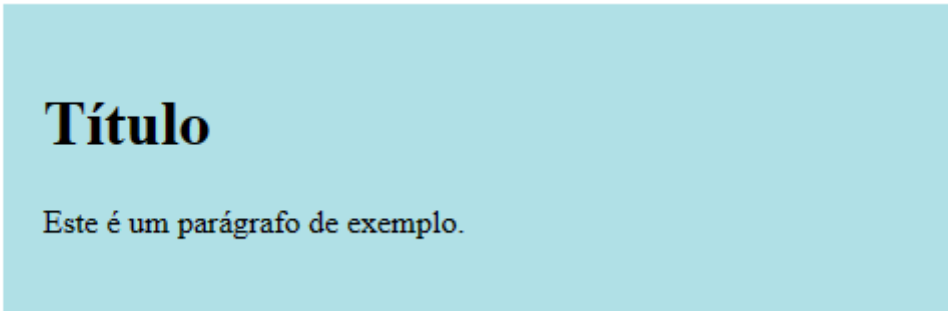
Um exemplo de uso da tag no CSS seria para criar uma seção em uma página da web que contém um conjunto de elementos que precisam ser estilizados juntos. Por exemplo, imagine que você tenha uma seção de depoimentos em seu site que contenha várias caixas de texto e imagens. Você poderia envolver todo esse conteúdo em uma tag e aplicar regras de estilo para essa tag, como tamanho da fonte, cor do texto, espaçamento entre elementos e margens.

É importante ressaltar que o uso excessivo da tag pode levar a um código HTML desnecessariamente complexo e difícil de manter. Por isso, é importante usar a tag com moderação e sempre com um propósito específico em mente.

Exemplo de como usar a tag no CSS:

```html
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" href="style.css">
</head>
<body>
<div class="container">
  <h1>Título</h1>
  <p>Este é um parágrafo de exemplo.</p>
</div>
</body>
</html>
```

```css
.container {
  width: 80%;
  margin: 0 auto;
  background-color: #B0E0E6;
  padding: 20px;
}
```



Neste exemplo, a tag é usada para envolver um conjunto de elementos HTML, no caso um título e um parágrafo. Em seguida, uma classe CSS chamada "container" é aplicada à tag . A classe "container" define a largura da div em 80%, adiciona margem automática para centralizá-la na página, adiciona uma cor de fundo e define um espaçamento interno de 20 pixels.

O uso da tag em conjunto com CSS permite agrupar elementos HTML relacionados e aplicar estilos a eles de maneira organizada e eficiente.

4.9 Utilizando a TAG

A tag é usada para aplicar estilos em partes específicas de um texto ou em elementos inline. Essa tag não cria um bloco de conteúdo, apenas aplica estilos em uma parte específica do texto ou elemento.

Para aplicar estilos em um elemento no CSS, você pode utilizar o seletor de classe ou de ID, assim como é feito com a tag . Por exemplo:

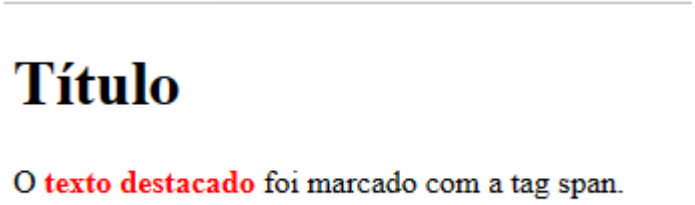
HTML:

```html
<p>O <span class="destaque">texto destacado</span> foi marcado com a tag span.</p>
```

CSS:

```css
.destaque {
   font-weight: bold;
   color: red;
}
```

Resultado:



Nesse exemplo, foi criada uma classe chamada "destaque" e aplicada no elemento que envolve o texto "texto destacado". Essa classe define um estilo de font-weight em negrito e cor vermelha. Dessa forma, apenas a parte destacada do texto receberá esses estilos.

Além disso, a tag também pode ser utilizada para agrupar elementos em um bloco inline. Por exemplo, você pode criar um bloco inline para mostrar uma imagem e uma legenda ao lado dela, como no exemplo abaixo:

HTML:

```html
<div class="imagem-com-legenda">
   <img src="imagem.jpg" alt="Imagem">
   <span class="legenda">Imagem de um belo pôr do sol.</span>
</div>
```

CSS:

```css
.imagem-com-legenda {
    display: inline-block;
    vertical-align: top;
}

.legenda {
    font-style: italic;
    font-size: 14px;
    margin-left: 5px;
}
```

Resultado:

Imagem de um belo pôr do sol.

Nesse exemplo, foi criada uma classe chamada "imagem-com-legenda" e aplicada no elemento que envolve a imagem e a legenda. Essa classe define um estilo de display em inline-block e vertical-align em top, para que a imagem e a legenda fiquem na mesma linha. Já a classe "legenda" define um estilo de font-style em itálico, font-size em 14px e margem à esquerda de 5px, para que a legenda fique ao lado da imagem.

4.10 Propriedades básicas

Cores

Cores são um aspecto fundamental do design visual de páginas web. Em CSS, é possível definir a cor de diversos elementos, como fundos, bordas, textos, entre outros.

Para definir uma cor em CSS, podemos usar algumas notações diferentes. A notação mais comum é a notação hexadecimal, que consiste em uma

combinação de seis dígitos (ou três dígitos se a cor for web-safe) que representam o valor de cada canal de cor (vermelho, verde e azul) em uma escala de 0 a 255. Por exemplo, o código hexadecimal #FF0000 representa a cor vermelha pura.

Outra notação comum é a notação RGB, que usa valores decimais para cada canal de cor. Por exemplo, a cor vermelha pode ser representada por rgb (255, 0, 0).

Além disso, é possível usar nomes de cores pré-definidas em CSS, como "red", "green", "blue", "yellow", entre outros.

Para aplicar uma cor em um elemento HTML, podemos usar a propriedade "color" para definir a cor do texto, e a propriedade "background-color" para definir a cor de fundo. Por exemplo:

```
p {
  color: #333333; /* cor do texto em tom de cinza */
  background-color: #FFFFFF; /* fundo branco */
}
```

Também é possível definir a cor de bordas e outros elementos usando as propriedades apropriadas em CSS.

É importante lembrar que o uso de cores deve ser feito com cuidado e considerando a acessibilidade. Pessoas com deficiência visual podem ter dificuldades em enxergar determinadas cores, por isso é importante garantir um bom contraste entre as cores usadas em um site. A W3C tem diretrizes específicas para o uso de cores em páginas web, como o WCAG (Web Content Accessibility Guidelines).

Plano de Fundo

O plano de fundo é uma das propriedades de estilo mais importantes do CSS, pois permite definir a cor ou imagem que aparecerá como fundo de uma página web. O uso adequado do plano de fundo pode contribuir significativamente para a aparência e experiência do usuário na navegação de um site.

Para definir o plano de fundo em CSS, podemos utilizar a propriedade background. Essa propriedade permite definir a cor de fundo, a imagem de fundo ou uma combinação dos dois. Existem diversas opções que podem ser utilizadas para personalizar o plano de fundo, tais como:

background-color: define a cor de fundo da página;

background-image: define uma imagem de fundo para a página;

background-repeat: define a forma como a imagem de fundo será repetida;

background-position: define a posição da imagem de fundo na página;

background-size: define o tamanho da imagem de fundo.

Para definir uma cor de fundo em CSS, podemos utilizar o valor de cor hexadecimal, que é representado por um código de seis caracteres que começa com o caractere "#" seguido por uma combinação de números e letras. Por exemplo, para definir um fundo vermelho, podemos utilizar o código hexadecimal #FF0000. Também é possível utilizar o nome da cor, como red, green e blue.

Para definir uma imagem de fundo, podemos utilizar a propriedade background-image, que permite especificar a URL da imagem. É possível utilizar imagens em diferentes formatos, como JPEG, PNG e GIF. Por exemplo, para definir uma imagem de fundo, podemos utilizar o seguinte código CSS:

```css
body {
  background-image: url("imagem-de-fundo.jpg");
}
```

Também é possível utilizar uma combinação de cores e imagens de fundo para criar efeitos mais elaborados. Por exemplo, podemos utilizar uma cor de fundo e uma imagem de fundo sobreposta, como no seguinte código CSS:

```css
body {
  background-color: #FFFFFF;
  background-image: url("imagem-de-fundo.png");
  background-repeat: no-repeat;
  background-position: center center;
  background-size: cover;
}
```

Esse código define um fundo branco com uma imagem de fundo sobreposta no centro da página, com tamanho ajustado automaticamente para cobrir todo o conteúdo do elemento body.

O plano de fundo é uma propriedade importante do CSS que permite definir a aparência e personalização de uma página web. É possível utilizar cores e imagens de fundo para criar efeitos visualmente atraentes e contribuir para uma melhor experiência do usuário na navegação do site.

5.0 ELEMENTOS ESTILIZADOS COM CSS

5.1 Textos em CSS

Em CSS, é possível alterar características do texto que não podem ser modificadas em HTML, como por exemplo, o espaçamento entre as linhas. Isso permite que textos em qualquer elemento do HTML sejam caracterizados usando propriedades específicas. Algumas das propriedades que podem ser utilizadas são:

color: define a cor do texto;

text-indent: define a distância de recuo do texto no início do parágrafo;

line-height: define o espaçamento entre as linhas;

text-align: define o alinhamento do texto, podendo ser ao centro, à direita, à esquerda ou no estilo justificado;

text-decoration: Define a decoração de um texto e é feita com os seguintes valores:

underline: sublinhado.

overline: sobrelinhado.

line-through: uma linha em cima do texto.

blink: faz piscar o texto.

text-transform: Define uma transformação ao texto, que podem ser as seguintes:

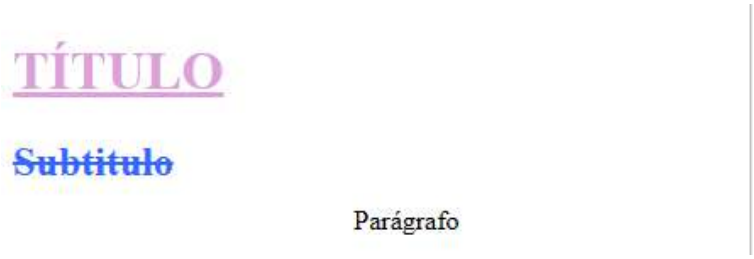
uppercase: torná-las todas maiúsculas.

lowercase: todas minúsculas.

capitalize: todas as primeiras letras maiúsculas.

```html
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" href="style.css">
</head>
<body>
<h1>Título</h1>
<h2>Subtitulo</h2>
<p>Parágrafo</p>

</body>
</html>
```

```css
h1 {
    color: #DDA0DD;
    text-decoration: underline;
    text-transform: uppercase;
}

h2 {
    color: #3366FF;
    text-decoration: line-through;
    text-transform: none;
}

p {
    text-indent: 1cm;
    line-height: 2px;
    text-align: center;
}
```



O uso de fontes de texto em HTML e CSS é uma forma importante de melhorar a apresentação visual de um site. A escolha da fonte pode influenciar diretamente na legibilidade do texto, transmitir uma sensação de estilo e personalidade, além de garantir a consistência visual entre as páginas do site.

Em HTML, é possível definir a fonte de texto de um elemento usando a tag <font>. Por exemplo:

```html
<font face="Arial, sans-serif">Este texto usa a fonte Arial</font>
```

No entanto, essa forma de definir a fonte é considerada obsoleta e não recomendada. É mais comum usar o CSS para definir a fonte de um texto.

Em CSS, a propriedade font-family é usada para definir a fonte de um elemento. Por exemplo:

```
body {
  font-family: Arial, sans-serif;
}
```

Essa regra define que todo o texto dentro do elemento body será exibido com a fonte Arial. A opção sans-serif é usada como uma opção de fallback, caso a fonte Arial não esteja disponível no computador do usuário.

Outras propriedades relacionadas às fontes incluem:

font-size: define o tamanho da fonte em pontos ou em pixels;

font-weight: define a espessura da fonte, que pode ser normal ou bold;

font-style: define o estilo da fonte, que pode ser normal, italic ou oblique;

font-variant: define a variação da fonte, como small-caps (todas as letras maiúsculas, mas com tamanho reduzido). Por exemplo:

```
h1 {
  font-family: "Times New Roman", serif;
  font-size: 32px;
  font-weight: bold;
  font-style: italic;
  font-variant: small-caps;
}
```

Essa regra define que todos os títulos h1 serão exibidos com a fonte Times New Roman, em negrito, itálico e com todas as letras maiúsculas em tamanho reduzido.

5.2 Bordas em CSS

As bordas em HTML e CSS são utilizadas para decorar elementos da página, como caixas de texto, imagens, botões, entre outros. Elas podem ser

aplicadas utilizando diversas propriedades, tais como: border-width (largura da borda), border-style (estilo da borda) e border-color (cor da borda).

```
<style>
  table {
    border-collapse: collapse;
    border: 1px solid red;
  }
  td {
    border: 1px solid red;
  }
</style>
```

Este exemplo define um estilo de borda sólido vermelho para a tabela e para cada célula da tabela.

O border-width define a largura da borda, que pode ser em pixels, em pontos, em porcentagem ou até mesmo com valores predefinidos, como "thin", "medium" e "thick". Por exemplo, para definir uma borda com largura de 1 pixel, podemos utilizar a propriedade border-width: 1px;.

O border-style define o estilo da borda, que pode ser sólido (solid), tracejado (dashed), pontilhado (dotted), duplo (double), entre outros. Por exemplo, para definir uma borda sólida, podemos utilizar a propriedade border-style: solid;.

O border-color define a cor da borda, que pode ser qualquer cor em RGB, hexadecimal ou nome da cor. Por exemplo, para definir uma borda vermelha, podemos utilizar a propriedade border-color: red;.

Além dessas propriedades básicas, também é possível combinar as três propriedades em uma única propriedade border, que segue a ordem largura, estilo e cor, separadas por espaços. Por exemplo, border: 1px solid red; define uma borda com largura de 1 pixel, estilo sólido e cor vermelha.

Podemos aplicar as bordas em diferentes elementos HTML, como , <img>, <button>, entre outros. Abaixo segue um exemplo de como aplicar bordas em uma imagem utilizando CSS:

```
<img src="exemplo.jpg" style="border: 1px solid black;">
```

Neste exemplo, a imagem exemplo.jpg terá uma borda com largura de 1 pixel, estilo sólido e cor preta.

Além disso, a propriedade border-radius permite criar bordas arredondadas em elementos, definindo um raio para cada canto do elemento. Isso pode ser útil para criar botões, caixas de mensagens, entre outros elementos com uma aparência mais suave. Exemplo:

Suponha que queremos criar uma caixa com uma borda sólida de cor preta e uma largura de 2 pixels. Para isso, utilizamos a seguinte propriedade CSS:

```
div {
  border: 2px solid black;
}
```

Isso criará uma borda sólida ao redor da caixa div com uma largura de 2 pixels e cor preta. Para criar bordas diferentes em cada lado, podemos utilizar as propriedades border-top, border-right, border-bottom e border-left.

Para criar bordas arredondadas, podemos utilizar a propriedade border-radius. Por exemplo, para criar uma caixa com bordas arredondadas de 10 pixels, utilizamos a seguinte propriedade CSS:

```
div {
  border: 2px solid black;
  border-radius: 10px;
}
```

Isso criará uma borda com cantos arredondados de 10 pixels ao redor da caixa div.

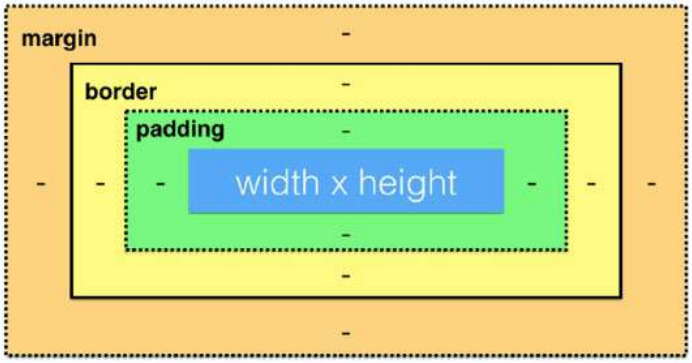
5.3 Margin e Padding

As propriedades margin e padding são utilizadas em CSS para controlar o espaçamento e a distância entre os elementos em uma página web.

A propriedade margin define a área externa de um elemento, ou seja, o espaço entre o elemento e o próximo elemento ao seu redor. Essa propriedade pode ser definida para cada lado do elemento (superior, inferior, esquerdo e direito) de forma individual, ou para todos os lados juntos através da propriedade margin sem especificar qual lado.

Já a propriedade padding define a área interna de um elemento, ou seja, o espaço entre o conteúdo do elemento e sua borda. Assim como a propriedade margin, a propriedade padding pode ser definida para cada lado do elemento ou para todos os lados juntos através da propriedade padding sem especificar qual lado.

As propriedades margin e padding aceitam valores em pixels, em porcentagens ou em unidades relativas, como em e rem.



Valores para margens:

margin-top: Define a margem superior.

margin-right: Define a margem direita.

margin-bottom: Define a margem inferior.

margin-left: Define a margem esquerda.

Margin: Define os valores das quatro margens na seguinte ordem: top, right, bottom, left.

Valores para Paddings:

padding-top: Define o padding superior.

padding-right: Define o padding direita.

padding-bottom: Define o padding inferior.

padding-left: Define o padding esquerda.

padding: Define os valores dos quatro paddings na seguinte ordem: top, right, bottom, left.

A seguir, um exemplo de como utilizar as propriedades margin e padding em CSS:

```css
div {
  background-color: #eee;
  border: 1px solid #ccc;
  margin: 20px; /* Define a margem para todos os lados */
  padding: 10px; /* Define o preenchimento para todos os lados */
}
```

Nesse exemplo, todas as Tags div da página terão uma margem de 20 pixels e um preenchimento de 10 pixels em todos os lados. O valor da propriedade margin afetará a distância entre as Tags div e os elementos ao seu redor, enquanto o valor da propriedade padding afetará a distância entre o conteúdo da tag div e sua borda.

5.4 Listas em CSS

Em CSS, é possível personalizar as listas HTML para melhor se adequarem ao estilo do seu site. Existem várias propriedades que podem ser aplicadas a listas, como margens, preenchimento, estilos de marcadores, entre outras. Abaixo estão algumas das propriedades mais comuns para personalização de listas em CSS:

list-style-type: define o estilo do marcador da lista, como círculos, quadrados, números, letras, entre outros. Alguns exemplos:

```css
ul {
  list-style-type: disc; /* círculos */
}

ol {
  list-style-type: decimal; /* números */
}
```

list-style-position: define a posição do marcador em relação ao texto. O valor padrão é "outside", que coloca o marcador à esquerda do texto. O valor "inside" coloca o marcador dentro do espaço reservado para o texto. Exemplo:

```
ul {
  list-style-position: inside; /* marcadores dentro do espaço do texto */
}
```

list-style-image: define uma imagem personalizada como marcador. Exemplo:

```
ul {
  list-style-image: url("marcador.png");
}
```

margin: define as margens externas da lista. Exemplo:

```
ul {
  margin: 20px;
}
```

padding: define o preenchimento interno da lista. Exemplo:

```
ul {
  padding: 10px;
}
```

Além disso, é possível personalizar a aparência dos itens da lista individualmente usando o seletor "li". Por exemplo:

```
li {
  font-weight: bold;
  color: red;
}
```

Essa regra definiria todos os itens da lista em negrito e na cor vermelha. No geral, a personalização de listas em CSS permite que você crie listas mais estilizadas e que combinem com o restante do seu site.

5.5 Tabelas

As propriedades de tabelas em CSS permitem que você personalize a aparência de tabelas em seu documento HTML. Aqui estão algumas das propriedades mais comuns:

border: define a largura e estilo da borda de uma tabela. Por exemplo, border: 1px solid black; define uma borda sólida preta com uma largura de 1 pixel.

border-collapse: define como as bordas da tabela devem ser tratadas quando se encontram. Por exemplo, border-collapse: collapse; combina as bordas das células adjacentes.

width: define a largura da tabela. Por exemplo, width: 100%; define a largura da tabela como 100% da largura do elemento pai.

text-align: define o alinhamento horizontal do conteúdo da tabela. Por exemplo, text-align: center; centraliza o conteúdo da tabela.

background-color: define a cor de fundo da tabela. Por exemplo, **background-color:** lightgrey; define um fundo cinza claro.

padding: define o espaçamento entre as bordas da tabela e o conteúdo das células. Por exemplo, padding: 10px; adiciona um espaçamento de 10 pixels em todas as quatro bordas da tabela.

border-spacing: define o espaçamento entre as células de uma tabela. Por exemplo, border-spacing: 5px; adiciona um espaçamento de 5 pixels entre as células.

Aqui está um exemplo de CSS para uma tabela:

```css
table {
  border: 1px solid black;
  border-collapse: collapse;
  width: 100%;
  text-align: center;
  background-color: lightgrey;
  padding: 10px;
  border-spacing: 5px;
}

th, td {
  border: 1px solid black;
  padding: 10px;
}
```

```html
<table>
  <tr>
    <td>Coluna 1, linha 1</td>
    <td>Coluna 2, linha 1</td>
  </tr>
  <tr>
    <td>Coluna 1, linha 2</td>
    <td>Coluna 2, linha 2</td>
  </tr>
</table>
```

Coluna 1, linha 1	Coluna 2, linha 1
Coluna 1, linha 2	Coluna 2, linha 2

Este CSS aplica uma borda sólida preta de 1 pixel à tabela e células, combina as bordas das células adjacentes, define a largura da tabela como 100% da largura do elemento pai, centraliza o conteúdo da tabela, define um fundo cinza claro, adiciona um espaçamento de 10 pixels em todas as quatro bordas da tabela e adiciona um espaçamento de 5 pixels entre as células. As células também têm uma borda sólida preta de 1 pixel e um espaçamento de 10 pixels em todas as quatro bordas.

6.0 SERVIDOR DE ACESSO LOCAL WAMP

6.1 Introdução ao WAMP: o que é e para que serve?

O WAMP (Windows, Apache, MySQL e PHP) é um software livre que possibilita a criação de um servidor de acesso local. Ele é muito utilizado para o desenvolvimento e teste de aplicações web, pois permite a simulação de um ambiente semelhante ao de um servidor de hospedagem, mas no próprio computador do desenvolvedor.

Segundo Davis e Phillips (2012), o WAMP é uma opção bastante popular para o desenvolvimento de aplicações web em ambiente Windows, uma vez que ele já vem com os principais componentes necessários para o desenvolvimento e execução de sites e aplicações web.

6.2 O que compõe o WAMP?

Windows é um sistema operacional desenvolvido pela Microsoft, utilizado em computadores pessoais e servidores. Ele fornece uma interface gráfica do usuário (GUI) e é compatível com vários softwares, incluindo servidores web.

Apache é um servidor web de código aberto amplamente utilizado para hospedar sites na internet. Ele é altamente configurável, possui suporte para múltiplos módulos e é compatível com várias plataformas, incluindo o Windows.

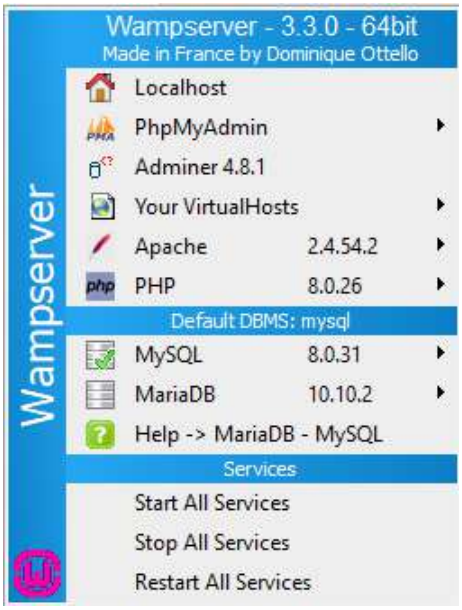
MySQL é um sistema de gerenciamento de banco de dados relacional (RDBMS) de código aberto, amplamente utilizado em aplicativos web. Ele oferece alta escalabilidade, suporte a transações ACID e é compatível com várias linguagens de programação, incluindo PHP.

PHP é uma linguagem de programação de código aberto amplamente utilizada para desenvolver aplicativos web. Ela é executada no lado do servidor e é compatível com vários bancos de dados, incluindo o MySQL. Ela é utilizada para criar sites dinâmicos e aplicativos web interativos.

6.3 Instalação do WAMP no Windows

Para instalar o WAMP no Windows, é necessário fazer o download do pacote de instalação no site oficial (www.wampserver.com). Após o

download, basta executar o arquivo e seguir as instruções apresentadas na tela.



Mazza (2014) recomenda que, durante a instalação, seja verificado se as portas de conexão estão configuradas corretamente, para evitar conflitos com outros programas. Além disso, é importante escolher a versão adequada para o sistema operacional utilizado e seguir as instruções apresentadas na tela para a instalação dos módulos adicionais, como o PHPMyAdmin.

Com o WAMP instalado é possível criar e testar aplicações web localmente antes de publicá-las em um servidor de hospedagem. Nieradka (2014) destaca que isso permite um desenvolvimento mais rápido e seguro, já que é possível identificar e corrigir erros antes da publicação.

6.4 Configuração do Apache e do PHP no WAMP

O WAMP é uma ferramenta que permite a instalação local de um servidor Apache, PHP e MySQL no Windows. Após a instalação do WAMP, é necessário configurar o Apache e o PHP para que possam ser usados corretamente.

A configuração do Apache pode ser feita através do arquivo httpd.conf, localizado na pasta do Apache dentro do diretório do WAMP. Neste arquivo, é possível definir diversas configurações, como a porta em que o servidor irá rodar e os diretórios que serão usados como raiz dos sites.

Já a configuração do PHP é feita através do arquivo php.ini, que também pode ser encontrado na pasta do Apache. Neste arquivo, é possível definir diversas configurações relacionadas ao PHP, como o caminho para as extensões e o limite de memória que o PHP pode usar.

6.5 Configuração do MySQL no WAMP

Além da configuração do Apache e do PHP, também é necessário configurar o MySQL no WAMP. Para isso, é necessário acessar o PHPMyAdmin, que é uma interface web para gerenciamento do MySQL.

No PHPMyAdmin, é possível criar bancos de dados, tabelas e usuários, além de realizar diversas outras tarefas relacionadas ao MySQL. É importante lembrar de definir uma senha forte para o usuário root do MySQL, a fim de garantir a segurança do sistema.

Utilizando o PHPMyAdmin para gerenciar o MySQL

O PHPMyAdmin é uma ferramenta web gratuita usada para gerenciar bancos de dados MySQL. Com ele, é possível criar, editar e excluir bancos de dados, tabelas, colunas, índices e muito mais. É uma ferramenta essencial para desenvolvedores que precisam lidar com bancos de dados MySQL em seus projetos.

Criando um banco de dados e tabelas no MySQL

Para criar um banco de dados e tabelas no MySQL usando o PHPMyAdmin, basta acessar o painel de controle do PHPMyAdmin, selecionar a opção de criar um novo banco de dados e, em seguida, criar as tabelas necessárias. As tabelas são criadas especificando o nome da tabela e suas colunas, bem como seus tipos de dados e outras propriedades.

É importante destacar que é fundamental ter um bom conhecimento em SQL para utilizar o PHPMyAdmin de forma eficiente e eficaz. Davis e Phillips (2013) afirmam que o PHPMyAdmin é "uma ferramenta muito poderosa, mas que requer um conhecimento sólido em SQL para tirar o máximo proveito dela". Além disso, Mazza (2015) destaca que o PHPMyAdmin é "uma ferramenta essencial para qualquer desenvolvedor web que precise gerenciar bancos de dados MySQL em seus projetos".

Conexão PHP ao banco de dados no WAMP:

Para realizar a conexão do PHP ao banco de dados no WAMP, é necessário estabelecer alguns parâmetros, como nome do banco de dados, usuário e senha. Uma das formas de realizar essa conexão é utilizando a extensão

PDO, que oferece suporte para diversos tipos de banco de dados, incluindo o MySQL. É importante lembrar que a conexão com o banco de dados deve ser segura, evitando vulnerabilidades como SQL injection.

A conexão feita entre o PHP e o Banco de dados será visto posteriormente pois necessita de conhecimentos da linguagem PHP.

Utilizando o PHP para realizar operações no banco de dados:

Após estabelecer a conexão com o banco de dados, é possível realizar diversas operações com o PHP, como inserção, atualização, exclusão e seleção de dados. Para isso, é necessário utilizar comandos SQL, como INSERT, UPDATE, DELETE e SELECT, que serão executados através do PHP. Além disso, é importante validar e sanitizar os dados inseridos no banco de dados para garantir a segurança da aplicação.

Criação de páginas web com HTML e CSS no WAMP.

O primeiro passo é instalar o WAMP no computador, seguindo as instruções do instalador. Uma vez instalado, o servidor Apache será iniciado automaticamente e pode ser acessado pelo navegador em "http://localhost". É possível verificar se o PHP também foi instalado corretamente, criando um arquivo PHP em um diretório do servidor web e acessando-o através do navegador.

Para criar uma página web, é necessário criar um arquivo HTML que contenha a estrutura básica do documento, como a tag <html>, <head>, <body>, entre outras. O HTML é uma linguagem de marcação que define a estrutura do conteúdo de uma página web, como cabeçalhos, parágrafos, imagens, links e tabelas.

O CSS é utilizado para estilizar o layout da página, alterando as cores, fontes, tamanhos e posições dos elementos HTML. O CSS pode ser incluído no documento HTML usando a tag <style>, que pode ser inserida no <head> do documento ou em um arquivo separado.

A criação de páginas web com HTML e CSS envolve a aprendizagem e a prática de diversos conceitos, incluindo o uso de Tags HTML, atributos, seletores CSS, propriedades, unidades de medida e modelos de box. É

importante também entender os princípios de design responsivo e acessibilidade web, que permitem que as páginas se adaptem a diferentes tamanhos de tela e possam ser acessadas por usuários com diferentes necessidades.

A criação de páginas web com HTML e CSS no WAMP envolve a criação de um arquivo HTML que define a estrutura do conteúdo da página e a inclusão de um arquivo CSS que estiliza o layout da página.

7.0 INTRODUÇÃO AO PHP

O PHP (acrônimo recursivo para "PHP: Hypertext Preprocessor") é uma linguagem de programação de código aberto, interpretada e principalmente utilizada para desenvolvimento web. Foi criada em 1994 por Rasmus Lerdorf e desde então vem sendo constantemente atualizada e aprimorada por uma comunidade ativa de desenvolvedores.

O objetivo principal do PHP é gerar conteúdo dinâmico para a web, como páginas, formulários, fóruns, entre outros. Por ser uma linguagem interpretada, é possível executar o código PHP em diferentes sistemas operacionais e servidores web.

7.1 Evolução da linguagem PHP

Ao longo dos anos, a linguagem PHP passou por diversas mudanças e evoluções, incorporando novas funcionalidades e melhorando a performance. A versão mais antiga do PHP ainda em uso é a 5.6, lançada em 2014, enquanto a versão atual é a 8.0, lançada em 2020.

Uma das grandes evoluções do PHP foi a introdução do paradigma de orientação a objetos na versão 5.0, em 2004. Com essa mudança, a linguagem passou a permitir o desenvolvimento de aplicações mais complexas e escaláveis, além de tornar o código mais organizado e fácil de manter.

Outra evolução importante foi a adição do mecanismo de gerenciamento de dependências Composer, que permite o fácil gerenciamento e instalação de bibliotecas e pacotes externos.

7.2 Versões e características do PHP

Cada versão do PHP traz novas funcionalidades e melhorias na performance e segurança. A partir da versão 5.0, a linguagem passou a suportar o paradigma de orientação a objetos e desde então vem incorporando novos recursos, como namespaces, traits e closures.

Na versão 7.0, lançada em 2015, houve um grande avanço na performance, com a introdução do mecanismo de compilação JIT (Just-In-Time), que torna a execução do código até duas vezes mais rápida do que na versão anterior.

Já na versão 8.0, lançada em 2020, algumas das principais novidades incluem a introdução de atributos e construtores de delegação, melhorias na sintaxe e na tipagem, além de uma significativa melhora na performance.

7.3 A Principal função do PHP

PHP é utilizada para desenvolvimento web do lado do servidor, ou seja, é processado no servidor antes de ser enviado ao cliente, permitindo que o conteúdo dinâmico seja criado em um site. Com o PHP, é possível criar páginas web dinâmicas e interativas que podem exibir conteúdo diferente com base nas ações do usuário, como preencher um formulário ou clicar em um botão.

Além disso, o PHP é frequentemente utilizado para desenvolver aplicativos web complexos, como sistemas de gerenciamento de conteúdo, sistemas de comércio eletrônico e aplicativos de rede social. O PHP também tem uma grande comunidade de desenvolvedores e uma vasta coleção de bibliotecas e frameworks que facilitam o desenvolvimento web.

Vamos explorar algumas das principais funcionalidades do PHP.

Processamento de formulários: O PHP é muito útil para o processamento de formulários em páginas web. Ele pode receber e manipular dados enviados através de formulários HTML, validando e processando esses dados para armazenamento em um banco de dados.

Acesso a banco de dados: O PHP oferece suporte a muitos bancos de dados populares, como MySQL, PostgreSQL e Oracle. Ele permite que você

se conecte a esses bancos de dados e execute consultas SQL para manipular dados.

Gerenciamento de sessão: As sessões do PHP permitem que você armazene e recupere dados entre várias solicitações do usuário. Isso é muito útil para armazenar informações de login, preferências do usuário e carrinhos de compras.

Trabalhar com arquivos: O PHP permite a manipulação de arquivos no servidor, como upload e download de arquivos. Também permite a leitura e gravação de dados em arquivos de texto e CSV.

Manipulação de imagens: O PHP oferece suporte a manipulação de imagens, como redimensionamento, recorte e aplicação de filtros. Isso é útil para a criação de galerias de imagens, avatares e muito mais.

Envio de e-mails: O PHP permite o envio de e-mails a partir de um servidor web. Isso pode ser útil para envio de notificações, confirmações de pedidos e muito mais.

Criação de gráficos: O PHP pode ser usado para gerar gráficos, como gráficos de barras, de pizza e de linhas. Isso é útil para visualizar dados de relatórios.

Essas são apenas algumas das funcionalidades mais comuns do PHP. Com o PHP, você pode fazer muito mais para desenvolver aplicativos web dinâmicos e interativos.

7.4 Fundamentos da linguagem PHP

O PHP é uma linguagem de programação open source utilizada para o desenvolvimento de aplicações web. Com ela, é possível criar desde simples páginas dinâmicas até sistemas complexos e robustos. O PHP foi criado em 1994 por Rasmus Lerdorf e, desde então, tem evoluído constantemente, se tornando uma das linguagens mais populares para desenvolvimento web.

7.5 Sintaxe básica do PHP

A sintaxe básica do PHP é relativamente simples e segue a mesma estrutura de outras linguagens de programação, como C e Java. Para começar um

código PHP, é necessário utilizar a tag "<?php" e, em seguida, escrever o código dentro desse bloco. Um exemplo básico seria:

```php
<?php
    echo "Olá, mundo!";
?>
```

Variáveis, constantes e tipos de dados:

Assim como outras linguagens de programação, o PHP permite a utilização de variáveis e constantes para armazenar valores e informações que serão utilizados no decorrer do código. Para declarar uma variável no PHP, basta utilizar o símbolo "$" seguido do nome da variável e atribuir um valor a ela. Já as constantes são definidas utilizando a função "define()".

O PHP possui diferentes tipos de variáveis e constantes que podem ser utilizados para armazenar e manipular dados dentro de um programa. Esses tipos incluem:

Variáveis escalares: são aquelas que armazenam apenas um valor por vez, como números, strings e booleanos. Algumas das variáveis escalares mais comuns incluem:

Integer: armazena números inteiros, positivos ou negativos.

Float: armazena números decimais, com precisão limitada.

String: armazena texto, podendo incluir letras, números e caracteres especiais.

Boolean: armazena valores verdadeiro ou falso.

Variáveis compostas: são aquelas que permitem armazenar vários valores em uma única variável, como arrays e objetos.

Array: armazena uma coleção de valores relacionados, podendo ser indexado ou associativo.

Object: armazena um objeto, que pode ter propriedades e métodos.

Constantes: são valores que não podem ser alterados durante a execução do programa. Elas são definidas usando a função define() e geralmente são escritas em letras maiúsculas.

Exemplo de definição de constante: define("PI", 3.14159);

É importante escolher o tipo de variável ou constante adequado para o tipo de dado que será armazenado, garantindo a integridade dos dados e a eficiência do programa. Além disso, é necessário considerar o escopo das variáveis, ou seja, onde elas são declaradas e podem ser acessadas no programa.

Segundo Davis e Phillips (2012), "o conhecimento dos fundamentos da linguagem é fundamental para se tornar um programador PHP competente e capaz de desenvolver sistemas complexos e eficientes". Por isso, é importante dedicar um tempo para estudar e praticar os conceitos básicos antes de partir para projetos mais avançados.

Operadores e expressões

Em programação, operadores são símbolos ou palavras reservadas que executam operações em dados. As expressões, por sua vez, são combinações de operadores e operandos que resultam em um valor. Em PHP, há diversos tipos de operadores e expressões que podem ser utilizados para realizar operações matemáticas, lógicas e de comparação.

Os operadores aritméticos em PHP incluem adição (+), subtração (-), multiplicação (*), divisão (/), resto da divisão (%), incremento (++) e decremento (--). Por exemplo, a expressão 10 + 5 resultaria em 15, enquanto a expressão $x++ incrementaria o valor da variável $x em 1.

Os operadores lógicos são utilizados para combinar expressões condicionais e retornar valores booleanos (verdadeiro ou falso). Os operadores lógicos em PHP incluem negação (!), E lógico (&&) e OU lógico (||). Por exemplo, a expressão !(2 == 2) retornaria falso, enquanto a expressão (2 == 2) && (3 > 1) retornaria verdadeiro.

Os operadores de comparação são utilizados para comparar valores e retornar valores booleanos. Os operadores de comparação em PHP incluem

igual (==), diferente (!=), maior (>), menor (<), maior ou igual (>=) e menor ou igual (<=). Por exemplo, a expressão 2 == "2" retornaria verdadeiro, enquanto a expressão 2 === "2" retornaria falso porque o operador de igualdade estrita verifica também o tipo de dado.

As expressões em PHP também podem ser agrupadas utilizando parênteses para especificar a ordem das operações. Por exemplo, a expressão (5 + 3) * 2 resultaria em 16, já que a adição seria resolvida antes da multiplicação.

Além dos operadores e expressões básicas, o PHP também suporta operadores de atribuição, operadores ternários e operadores de incremento/decremento pós-fixados. É importante entender como utilizar esses operadores e expressões de forma adequada para realizar operações complexas em programas em PHP.

Segue abaixo um exemplo de código em PHP utilizando alguns dos operadores e expressões mencionados acima:

```php
<?php
// operadores aritméticos
$x = 10;
$y = 5;
echo $x + $y; // 15
echo $x % $y; // 0
echo ++$x; // 11
```

```php
// operadores lógicos
$a = true;
$b = false;
echo !$a; // false
echo $a && $b; // false
```

```php
// operadores de comparação
$c = 3;
$d = "3";
var_dump($c == $d); // true
var_dump($c === $d); // false
```

```php
// agrupamento de expressões
echo (5 + 3) * 2; // 16
?>
```

8.0 ESTRUTURAS DE CONTROLE DE FLUXO EM PHP

As estruturas de controle de fluxo são essenciais para que um programa possa executar diferentes tarefas dependendo das condições encontradas durante a execução. No PHP, existem três principais estruturas de controle de fluxo: if/else, switch/case e loops.

Vamos recordar um pouco sobre estas estruturas:

As estruturas condicionais e de repetição são fundamentais na programação. Duas estruturas de controle de fluxo muito comuns em PHP são o "if" e o "while".

A estrutura "if" é usada para executar um bloco de código somente se uma condição for verdadeira. O formato básico é o seguinte:

```php
if (condição) {
  // código a ser executado se a condição for verdadeira
}
```

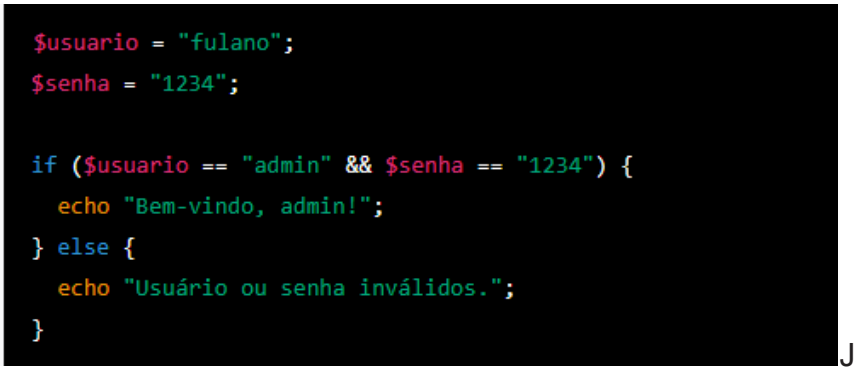
Por exemplo, o código abaixo verifica se uma variável "idade" é maior que 18 e exibe uma mensagem na tela:

```php
$idade = 20;

if ($idade > 18) {
  echo "Você é maior de idade.";
}
```

Já a estrutura "ifelse" permite que outro bloco de código seja executado caso a condição do "if" seja falsa. O formato é:

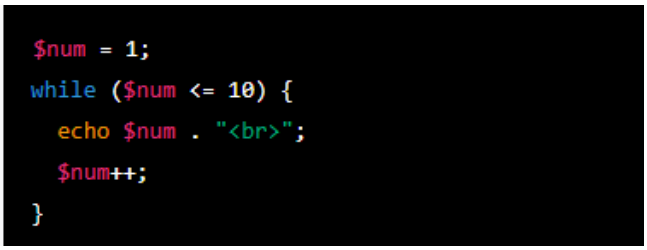
```php
if (condição) {
  // código a ser executado se a condição for verdadeira
} else {
  // código a ser executado se a condição for falsa
}
```

Por exemplo, o código abaixo verifica se um usuário tem permissão de acesso e exibe uma mensagem de acordo:



```php
$usuario = "fulano";
$senha = "1234";

if ($usuario == "admin" && $senha == "1234") {
  echo "Bem-vindo, admin!";
} else {
  echo "Usuário ou senha inválidos.";
}
```

Já a estrutura "while" é usada para repetir um bloco de código enquanto uma condição for verdadeira. O formato é:



```php
$num = 1;
while ($num <= 10) {
  echo $num . "<br>";
  $num++;
}
```

Nesse exemplo, a variável $num é inicializada com o valor 1. Em seguida, o laço "while" é utilizado para repetir o bloco de código enquanto a condição $num <= 10 for verdadeira. Dentro do laço, o valor atual de $num é impresso na tela utilizando a função echo e, em seguida, a variável é incrementada em 1. Esse processo se repete até que $num atinja o valor 11, momento em que a condição deixa de ser verdadeira e o laço é interrompido.

O resultado desse código seria a impressão dos números de 1 a 10, um em cada linha. Vale ressaltar que a estrutura "while" é especialmente útil quando o número de repetições não é conhecido antecipadamente, ou quando a condição de parada não pode ser expressa facilmente em uma única instrução "for".

8.1 Lógica em Estruturas

A estrutura if/else permite que o programa execute um bloco de código se uma condição for verdadeira e outro bloco de código se a condição for falsa. A sintaxe básica da estrutura if/else é a seguinte:

```
if (condição) {
  // bloco de código se a condição for verdadeira
} else {
  // bloco de código se a condição for falsa
}
```

Por exemplo, o código abaixo verifica se uma variável $idade é maior ou igual a 18 anos e, caso seja verdadeiro, imprime uma mensagem de boas-vindas para um site de conteúdo adulto:

```
if ($idade >= 18) {
  echo "Bem-vindo ao site de conteúdo adulto!";
} else {
  echo "Você não tem idade suficiente para acessar este site.";
}
```

A estrutura switch/case é semelhante à estrutura if/else, mas é usada quando há várias condições a serem avaliadas. Ela permite que o programa execute diferentes blocos de código dependendo do valor de uma variável. A sintaxe básica da estrutura switch/case é a seguinte:

```
switch (variável) {
  case valor1:
    // bloco de código se a variável for igual a valor1
    break;
  case valor2:
    // bloco de código se a variável for igual a valor2
    break;
  default:
    // bloco de código se a variável não for igual a nenhum valor anterior
}
```

Por exemplo, o código abaixo verifica o valor de uma variável $opcao e, dependendo do valor, executa uma determinada operação matemática:

```php
switch ($opcao) {
  case 1:
    $resultado = $valor1 + $valor2;
    break;
  case 2:
    $resultado = $valor1 - $valor2;
    break;
  case 3:
    $resultado = $valor1 * $valor2;
    break;
  case 4:
    $resultado = $valor1 / $valor2;
    break;
  default:
    echo "Opção inválida.";
}
```

Os loops são estruturas de controle de fluxo que permitem que um bloco de código seja repetido várias vezes. No PHP, existem três tipos de loops: while, do/while e for. O loop while executa um bloco de código enquanto uma condição for verdadeira. A sintaxe básica do loop while é a seguinte:

```php
while (condição) {
  // bloco de código a ser repetido enquanto a condição for verdadeira
}
```

Por exemplo, o código abaixo imprime os números de 1 a 10:

```php
$i = 1;
while ($i <= 10) {
  echo $i;
  $i++;
}
```

9.0 FUNÇÕES PREDEFINIDAS DO PHP

O PHP fornece uma ampla variedade de funções predefinidas que podem ser usadas em diferentes situações. Essas funções podem ser agrupadas em várias categorias, como funções de string, de array, de matemática, de tempo, de arquivo, de rede, de banco de dados, entre outras.

Algumas das funções mais comuns são:

Funções de String:

strlen(): retorna o tamanho de uma string.

```php
<!DOCTYPE html>
<html>
<body>
<?php
$nome = "Maria";
$tamanho = strlen($nome);
echo "O tamanho da string é: " . $tamanho; //
?>
</body>
</html>
```

O tamanho da string é: 5

substr(): retorna uma parte de uma string.

str_replace(): substitui uma substring por outra em uma string.

strpos(): retorna à posição de uma substring em uma string.

strtolower(): converte uma string para minúsculas.

strtoupper(): converte uma string para maiúsculas.

```php
<!DOCTYPE html>
<html>
<body>
<?php
$texto = "exemplo de texto em maiúsculas";
$texto_maiusculo = strtoupper($texto);
echo $texto_maiusculo;
?>
</body>
</html>
```

EXEMPLO DE TEXTO EM MAIúSCULAS

Funções de Array:

count(): retorna o número de elementos em um array.

array_push(): adiciona um ou mais elementos no final de um array.

```php
<!DOCTYPE html>
<html>
<body>
<?php
$frutas = array("maçã", "banana");
array_push($frutas, "laranja", "uva");
print_r($frutas);
?>
</body>
</html>
```

```
Array ( [0] => maçã [1] => banana [2] => laranja [3] => uva )
```

array_pop(): remove e retorna o último elemento de um array.

array_shift(): remove e retorna o primeiro elemento de um array.

array_unshift(): adiciona um ou mais elementos no início de um array.

array_merge(): mescla dois ou mais arrays em um único array.

Funções de Matemática:

abs(): retorna o valor absoluto de um número.

ceil(): arredonda um número para cima.

floor(): arredonda um número para baixo.

round(): arredonda um número para o valor mais próximo.

rand(): gera um número aleatório.

Funções de Tempo:

time(): retorna o número de segundos desde 1º de janeiro de 1970.

date(): formata uma data e hora.

```php
<!DOCTYPE html>
<html>
<body>
<?php
$data = date("d/m/Y H:i:s");
echo "A data atual é: " . $data;
?>
</body>
</html>
```

```
A data atual é: 04/04/2023 21:19:05
```

strtotime(): converte uma string em uma data e hora.

Funções de Arquivo:

fopen(): abre um arquivo.

fclose(): fecha um arquivo aberto.

fread(): lê um arquivo aberto.

fwrite(): escreve em um arquivo aberto.

file_exists(): verifica se um arquivo existe.

Funções de Rede:

fopen(): abre uma conexão com um URL.

fread(): lê dados de uma conexão com um URL.

fsockopen(): abre uma conexão com um servidor remoto.

fgets(): lê uma linha de uma conexão com um servidor remoto.

Funções de Banco de Dados:

mysqli_connect(): estabelece uma conexão com um servidor MySQL.

mysqli_query(): executa uma consulta SQL em um banco de dados.

mysqli_fetch_assoc(): busca uma linha de resultado como um array associativo.

As funções predefinidas do PHP são muito úteis, pois permitem que você escreva código mais rápido e eficiente. No entanto, é importante lembrar que o uso excessivo de funções predefinidas pode tornar o código mais lento e difícil de manter. Por isso, é sempre recomendável usar as funções personalizadas sempre que possível, pois elas são mais específicas para o seu projeto e podem ser otimizadas de acordo com suas necessidades.

9.1 Criação e utilização de funções personalizadas

O PHP é uma linguagem de programação que oferece diversas funcionalidades e recursos para a criação de aplicações web. Uma dessas funcionalidades é a capacidade de criar e utilizar funções personalizadas. As funções personalizadas são blocos de código que podem ser reutilizados em diferentes partes do código, permitindo uma maior organização e facilidade na manutenção do projeto.

Para criar uma função personalizada em PHP, é necessário definir seu nome, os parâmetros de entrada (se houver) e o código a ser executado. Por exemplo, podemos criar uma função que receba dois números como entrada e retorne a soma entre eles:

```php
function soma($num1, $num2){
   $resultado = $num1 + $num2;
   return $resultado;
}
```

Nesse exemplo, a função "soma" recebe dois parâmetros, $num1 e $num2, e retorna o resultado da soma entre eles.

As funções personalizadas em PHP podem ser chamadas em qualquer parte do código, desde que tenham sido definidas anteriormente. Por exemplo, podemos chamar a função "soma" em outra parte do código da seguinte maneira:

```php
echo soma(2, 3); // imprime 5
```

```php
<!DOCTYPE html>
<html>
<body>
<?php
function soma($num1, $num2){
   $resultado = $num1 + $num2;
   return $resultado;
}
echo soma(2, 3); // imprime 5
?>
</body>
</html>
```

```
5
```

Além disso, as funções personalizadas também podem ser utilizadas em conjunto com outras funções predefinidas do PHP. Por exemplo, podemos utilizar a função "strlen", que retorna o tamanho de uma string, em conjunto com uma função personalizada que recebe uma string como entrada e retorna seu tamanho quadrado:

```php
<!DOCTYPE html>
<html>
<body>
<?php
function tamanho_quadrado($texto){
   $tamanho = strlen($texto);
   $resultado = $tamanho * $tamanho;
   return $resultado;
}

echo tamanho_quadrado("ola"); //
imprime 9
?>
</body>
</html>
```

```
9
```

Nesse exemplo, a função "tamanho_quadrado" recebe um texto como entrada, calcula seu tamanho e retorna o tamanho ao quadrado.

As funções personalizadas são uma funcionalidade importante do PHP, permitindo que blocos de código sejam reutilizados em diferentes partes do projeto, aumentando a organização e a facilidade na manutenção do código. A criação e utilização de funções personalizadas é uma habilidade essencial para qualquer desenvolvedor PHP.

9.2 Manipulação de strings e arrays

Manipulação de Strings e Arrays é uma parte importante da programação em PHP, pois é comum a necessidade de realizar operações com textos e listas de dados. Existem diversas funções predefinidas no PHP para realizar essas tarefas.

Para manipulação de strings, podemos utilizar a função strlen(), que retorna o tamanho da string passada como argumento, e a função str_replace(), que substitui uma parte da string por outra. Por exemplo:

```php
<!DOCTYPE html>
<html>
<body>
<?php
$string = "Olá mundo!";
$tamanho = strlen($string); // retorna
11
echo $string;
$nova_string = str_replace("mundo",
"pessoal", $string); // retorna "Olá
pessoal!"
echo $nova_string;
?>
</body>
</html>
```

Olá mundo!Olá pessoal!

Já para a manipulação de arrays, podemos utilizar funções como array_push(), que adiciona um elemento ao final do array, e array_pop(), que remove o último elemento do array. Também podemos utilizar a função count(), que retorna à quantidade de elementos do array, e a função in_array(), que verifica se um valor está presente no array. Por exemplo:

```php
$lista = array("maçã", "banana", "laranja");
array_push($lista, "morango"); // adiciona "morango" ao final do array
array_pop($lista); // remove "laranja" do array
$quantidade = count($lista); // retorna 3
$tem_maca = in_array("maçã", $lista); // retorna true
```

Além das funções predefinidas, também podemos criar nossas próprias funções personalizadas para manipulação de strings e arrays. Isso pode ser útil quando precisamos realizar tarefas específicas que não são contempladas pelas funções predefinidas. Por exemplo, podemos criar uma função que recebe um array de números e retorna a soma desses números:

```php
function somarArray($array) {
  $soma = 0;
  foreach ($array as $numero) {
    $soma += $numero;
  }
  return $soma;
}

$meu_array = array(1, 2, 3, 4, 5);
$total = somarArray($meu_array); // retorna 15
```

Esses são apenas alguns exemplos de como podemos manipular strings e arrays em PHP. Com essas funcionalidades, podemos realizar uma infinidade de operações e criar programas cada vez mais complexos e úteis.

9.3 Manipulação de datas e horas

A manipulação de datas e horas é uma tarefa importante em muitas aplicações web, como em sistemas de reservas, agendas e gerenciamento de projetos. Em PHP, existem diversas funções predefinidas para realizar operações com datas e horas de forma fácil e eficiente.

A primeira etapa para manipular datas em PHP é a criação de um objeto da classe DateTime, que representa uma data e hora específica. Para isso, podemos usar a função date_create, que recebe uma string contendo a data no formato desejado:

```php
$data = date_create('2022-03-29');
```

Com o objeto $data criado, podemos utilizar diversas funções predefinidas para manipular as informações contidas nele. Por exemplo, para exibir a data no formato "dd/mm/yyyy", podemos utilizar a função format:

```php
echo $data->format('d/m/Y');
```

Além disso, é possível realizar diversas operações com datas e horas, como adicionar ou subtrair dias, horas, minutos e segundos. Para isso, podemos utilizar as funções add e sub:

```php
$data->add(new DateInterval('P1D')); // adiciona 1 dia
$data->sub(new DateInterval('PT2H30M')); // subtrai 2 horas e 30 minutos
```

Outra função bastante útil é a diff, que permite calcular a diferença entre duas datas em anos, meses, dias, horas, minutos e segundos:

```php
$data1 = date_create('2022-03-29');
$data2 = date_create('2023-06-15');

$diferenca = $data1->diff($data2);
echo $diferenca->format('%y anos, %m meses e %d dias');
```

Além das funções predefinidas, também é possível criar funções personalizadas para manipulação de datas em PHP. Por exemplo, podemos criar uma função que calcula o próximo dia útil a partir de uma determinada data:

```php
function proximo_dia_util($data) {
    $data = date_create($data);

    // adiciona um dia até encontrar um dia útil
    do {
        $data->add(new DateInterval('P1D'));
    } while ($data->format('N') >= 6); // sábado ou domingo

    return $data->format('d/m/Y');
}

echo proximo_dia_util('2022-03-29'); // 30/03/2022
```

10.0 FORMULÁRIOS EM HTML USANDO PHP

O uso de formulários é essencial na criação de páginas web interativas, permitindo que os usuários possam enviar informações e realizar ações na página. No entanto, para que essas informações possam ser processadas e armazenadas, é necessário trabalhar com uma linguagem de programação do lado do servidor, como o PHP.

10.1 Trabalhando com Formulários HTML

Os formulários HTML consistem em vários elementos, incluindo campos de entrada de texto, caixas de seleção, botões de opção e botões de envio. Esses elementos são definidos dentro de uma tag form, que especifica o

método de envio (GET ou POST) e a URL do script PHP que irá processar os dados do formulário.

Por exemplo, o seguinte código HTML cria um formulário que solicita ao usuário seu nome e sobrenome e um botão de envio:

```
<form method="POST" action="processa_formulario.php">
  <label for="nome">Nome:</label>
  <input type="text" id="nome" name="nome">
  <br>
  <label for="sobrenome">Sobrenome:</label>
  <input type="text" id="sobrenome" name="sobrenome">
  <br>
  <input type="submit" value="Enviar">
</form>
```

Ao clicar no botão de envio, o formulário será enviado ao script processa_formulario.php, que irá processar os dados enviados e realizar a ação desejada.

O PHP pode receber esses dados do formulário usando as superglobais $_GET ou $_POST, dependendo do método de envio especificado no formulário. Por exemplo, o seguinte código PHP recebe os dados enviados pelo formulário acima via POST:

```
$nome = $_POST['nome'];
$sobrenome = $_POST['sobrenome'];
```

Esses dados podem ser usados para realizar várias ações, como atualizar um banco de dados ou exibir uma mensagem personalizada na página.

Exemplo prático:

Suponha que você queira criar um formulário para que o usuário insira seu nome e sobrenome e exiba uma mensagem personalizada na página. Para isso, você pode criar um arquivo HTML com o código acima e um arquivo PHP com o seguinte código para processar os dados:

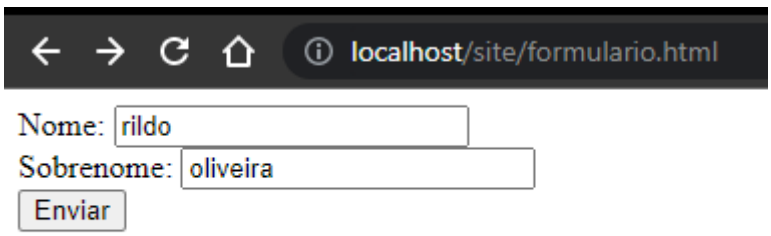
```php
<?php
  $nome = $_POST['nome'];
  $sobrenome = $_POST['sobrenome'];

  echo "Olá, " . $nome . " " . $sobrenome . "! Bem-vindo(a)!";
?>
```

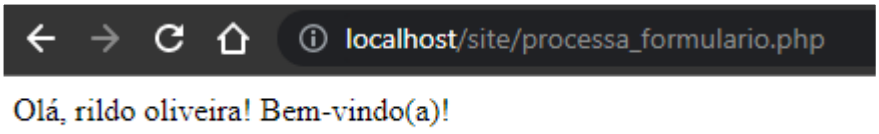
Ao preencher e enviar o formulário, a página exibirá a mensagem "Olá, [nome] [sobrenome]! Bem-vindo(a)!", onde [nome] e [sobrenome] são os valores inseridos pelo usuário.

```html
<!DOCTYPE html>
<html>
<head></head>
<body>
<form method="POST" action="processa_formulario.php">
  <label for="nome">Nome:</label>
  <input type="text" id="nome" name="nome">
  <br>
  <label for="sobrenome">Sobrenome:</label>
  <input type="text" id="sobrenome" name="sobrenome">
  <br>
  <input type="submit" value="Enviar">
</form>
</body>
</html>
```

processa_formulario.php

```php
<!DOCTYPE html>
<html>
<head></head>
<body>
<?php
  $nome = $_POST['nome'];
  $sobrenome = $_POST['sobrenome'];

  echo "Olá, " . $nome . " " . $sobrenome . "! Bem-vindo(a)!";
?>
</body>
</html>
```





Enviando dados de formulários via método GET e POST

vamos discutir sobre o envio de dados de formulários através dos métodos GET e POST.

Ao enviar dados de formulários de uma página HTML para uma página PHP, é importante entender os diferentes métodos de envio. Os dois métodos mais comuns são GET e POST.

O método GET é usado quando o formulário envia dados para a URL como parâmetros. Esses dados podem ser acessados através da variável global $_GET. Por exemplo, se um formulário de pesquisa envia dados usando o método GET, os parâmetros de pesquisa serão adicionados à URL.

O método POST é usado quando os dados de formulário são enviados para o servidor em uma solicitação HTTP separada. Esses dados podem ser acessados através da variável global $_POST. É importante usar o método POST para enviar dados confidenciais, como senhas, pois eles não serão visíveis na URL.

Aqui está um exemplo de um formulário HTML simples que envia dados via POST para uma página PHP:

```
<form method="post" action="process.php">
  <label for="name">Nome:</label>
  <input type="text" id="name" name="name">
  <label for="email">Email:</label>
  <input type="email" id="email" name="email">
  <button type="submit">Enviar</button>
</form>
```

E aqui está um exemplo de como processar os dados enviados no arquivo "process.php":

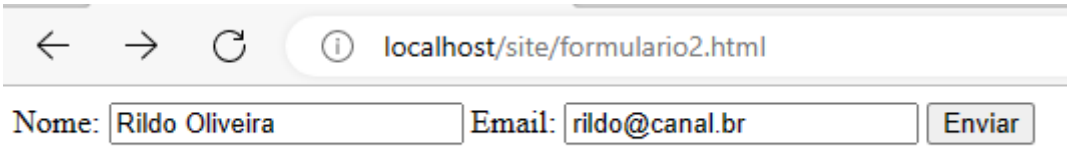
```php
<?php
  $name = $_POST['name'];
  $email = $_POST['email'];
  echo "Nome: $name<br>";
  echo "Email: $email";
?>
```

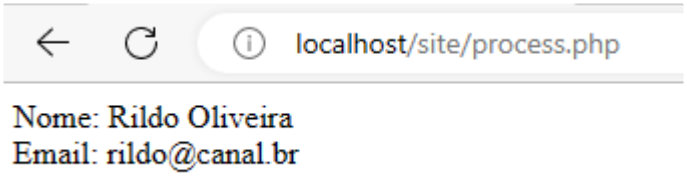
Note que estamos acessando os valores dos campos do formulário usando a variável global $_POST e depois imprimindo os valores na tela.

```html
<!DOCTYPE html>
<html>
<head></head>
<body>
<form method="post" action="process.php">
  <label for="name">Nome:</label>
  <input type="text" id="name" name="name">
  <label for="email">Email:</label>
  <input type="email" id="email" name="email">
  <button type="submit">Enviar</button>
</form>

</body>
</html>
```

process.php

```php
<!DOCTYPE html>
<html>
<head></head>
<body>
<?php
  $name = $_POST['name'];
  $email = $_POST['email'];
  echo "Nome: $name<br>";
  echo "Email: $email";
?>
</body>
</html>
```





É importante validar e sanitizar os dados de entrada do usuário antes de usá-los em seu código. Isso pode ajudar a evitar vulnerabilidades de segurança e erros de código.

10.2 Validando e sanitizando entradas de usuário

Quando os usuários interagem com um site, é importante validar e sanitizar as entradas de dados que eles fornecem para evitar problemas de segurança, como ataques de injeção de SQL, XSS (cross-site scripting) e outros tipos de vulnerabilidades.

O PHP oferece várias funções para validar e sanitizar entradas de usuário, e é importante que os desenvolvedores web saibam como usá-las corretamente.

A validação de entrada refere-se a garantir que as informações fornecidas pelos usuários sejam do tipo esperado, como números, endereços de e-mail ou datas. O PHP possui funções predefinidas que podem ser usadas para validar esses tipos de dados, como filter_var() e preg_match().

Já a sanitização refere-se a remover qualquer tipo de código malicioso que possa ter sido inserido nas entradas de usuário, como scripts JavaScript ou comandos SQL. O PHP possui funções para sanitizar entradas, como htmlentities() e addslashes().

É importante lembrar que a validação e sanitização de entradas de usuário não devem ser as únicas medidas de segurança em um site. Outras medidas, como criptografia de senhas e proteção contra ataques de CSRF (cross-site request forgery), também devem ser implementadas.

Exemplo de validação de um endereço de e-mail:

```php
$email = $_POST['email'];

if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
  echo "Endereço de e-mail inválido";
} else {
  echo "Endereço de e-mail válido";
}
```

Exemplo de sanitização de um campo de entrada:

```php
$senha = $_POST['senha'];

// Sanitiza a senha para evitar injeção de SQL
$senha = addslashes($senha);

// Sanitiza a senha para remover código malicioso
$senha = htmlentities($senha, ENT_QUOTES);
```

10.3 Criando e utilizando cookies e sessões

O uso de cookies e sessões em PHP é uma forma de armazenar informações do usuário e personalizar a experiência do usuário. Um cookie é um pequeno arquivo de texto que é armazenado no computador do usuário e é usado para armazenar informações como preferências do usuário, nome de usuário, senhas etc. As sessões são usadas para armazenar informações temporárias do usuário enquanto ele estiver navegando no site.

Para criar um cookie em PHP, podemos usar a função setcookie(). A função recebe vários parâmetros, como nome, valor, data de expiração etc. O seguinte exemplo mostra como criar um cookie com o nome "username" e o valor "joao":

```php
setcookie("username", "joao", time() + 3600);
```

Neste exemplo, o cookie expirará em uma hora (3600 segundos após a hora atual). Para acessar o valor do cookie, podemos usar a variável $_COOKIE. O seguinte exemplo mostra como recuperar o valor do cookie "username":

```
echo $_COOKIE["username"];
```

Para criar uma sessão em PHP, precisamos usar a função session_start(). Essa função inicia uma sessão no servidor e cria um ID de sessão exclusivo para o usuário. Podemos armazenar informações na sessão usando a variável $_SESSION. O seguinte exemplo mostra como armazenar o valor "joao" na sessão:

```
session_start();
$_SESSION["username"] = "joao";
```

Podemos acessar o valor da sessão a qualquer momento usando a variável $_SESSION. O seguinte exemplo mostra como recuperar o valor da sessão "username":

```
session_start();
echo $_SESSION["username"];
```

É importante lembrar que as sessões têm um tempo limite de inatividade. Por padrão, o tempo limite é de 24 minutos. Podemos alterar o tempo limite usando a configuração session.gc_maxlifetime no arquivo php.ini.

O uso de cookies e sessões em PHP pode ajudar a personalizar a experiência do usuário e armazenar informações do usuário. No entanto, é importante lembrar que as informações armazenadas em cookies e sessões podem ser acessadas pelo usuário, portanto, é importante não armazenar informações confidenciais, como senhas, nesses mecanismos de armazenamento.

11.0 INTEGRAÇÃO COM BANCOS DE DADOS

A integração com bancos de dados é fundamental para a maioria das aplicações web. É a partir do banco de dados que as informações são armazenadas e recuperadas. O PHP tem suporte a diversos bancos de dados, como MySQL, PostgreSQL, Oracle, entre outros. Para integrar o PHP

com um banco de dados, é necessário realizar a conexão com o banco de dados e executar consultas SQL.

11.1 Conexão com bancos de dados MySQL

O MySQL é um banco de dados relacional muito utilizado em aplicações web. Para estabelecer a conexão com um banco de dados MySQL, é necessário informar os dados de acesso, como o nome do servidor, o nome do usuário, a senha e o nome do banco de dados. Veja um exemplo de conexão com um banco de dados MySQL em PHP:

```php
$servername = "localhost";
$username = "root";
$password = "senha";
$dbname = "banco_de_dados";

// Cria a conexão
$conn = new mysqli($servername, $username, $password, $dbname);

// Verifica se a conexão foi estabelecida com sucesso
if ($conn->connect_error) {
    die("Conexão falhou: " . $conn->connect_error);
}
```

11.2 Executando consultas SQL com o PHP

Após estabelecer a conexão com o banco de dados, é possível executar consultas SQL em PHP. As consultas SQL são utilizadas para recuperar informações do banco de dados, inserir, atualizar ou excluir dados. Veja um exemplo de como executar uma consulta SQL em PHP:

```php
$sql = "SELECT * FROM usuarios";
$resultado = $conn->query($sql);

if ($resultado->num_rows > 0) {
    // Loop pelos resultados
    while($row = $resultado->fetch_assoc()) {
        echo "Nome: " . $row["nome"] . " - Email: " . $row["email"];
    }
} else {
    echo "0 resultados";
}
```

11.3 Utilizando PDO para acessar diferentes tipos de banco de dados

O PDO (PHP Data Objects) é uma extensão do PHP que fornece uma interface para acessar diferentes tipos de bancos de dados. O PDO permite que o mesmo código seja utilizado para acessar bancos de dados diferentes, basta apenas alterar a string de conexão. Veja um exemplo de como utilizar o PDO para conectar em um banco de dados MySQL:

```php
$dsn = "mysql:host=localhost;dbname=banco_de_dados";
$username = "root";
$password = "senha";

try {
    $conn = new PDO($dsn, $username, $password);
    // Configura o modo de erro para exceções
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    echo "Conexão realizada com sucesso";
} catch(PDOException $e) {
    echo "Conexão falhou: " . $e->getMessage();
}
```

A integração com bancos de dados é um dos principais recursos do PHP para o desenvolvimento de aplicações web. É possível conectar com diversos bancos de dados e executar consultas SQL para recuperar, inserir, atualizar ou excluir informações. O uso do PDO permite acessar diferentes tipos de bancos de dados com o mesmo código.

Conexão com bancos de dados MySQL

O acesso a bancos de dados é uma das tarefas mais comuns em desenvolvimento de aplicativos web, e PHP possui recursos nativos para lidar com essa tarefa. No entanto, é necessário compreender os conceitos básicos de conexão e manipulação de dados em bancos de dados.

MySQL é um dos bancos de dados mais utilizados no mundo e a integração com o PHP é muito fácil. A conexão com um banco de dados MySQL pode ser feita utilizando a função "mysqli_connect()". A sintaxe é a seguinte:

```php
$mysqli = mysqli_connect("hostname", "username", "password", "database_name");
```

$mysqli = mysqli_connect("hostname", "username", "password", "database_name");

Os parâmetros são o nome do servidor de banco de dados (hostname), o nome de usuário do banco de dados (username), a senha do usuário (password) e o nome do banco de dados que você deseja conectar (database_name).

Após estabelecer a conexão, podemos executar consultas SQL para recuperar, inserir, atualizar ou excluir dados. Por exemplo, para recuperar todos os registros de uma tabela, podemos usar a função "mysqli_query()":

```
$result = mysqli_query($mysqli, "SELECT * FROM tabela");
```

O resultado é um objeto "mysqli_result" que pode ser percorrido utilizando a função "mysqli_fetch_assoc()":

```
while ($row = mysqli_fetch_assoc($result)) {
    echo $row['campo1'] . ' ' . $row['campo2'];
}
```

No entanto, é importante lembrar que a manipulação de dados em bancos de dados pode ser vulnerável a ataques de segurança, como SQL injection. Por isso, é importante utilizar as funções de escape e sanitização de entrada de usuário. Davis e Phillips destacam a importância da validação de entrada e recomendam o uso de funções como "mysqli_real_escape_string()" para evitar a injeção de SQL em consultas.

Além disso, a utilização do PDO (PHP Data Object) é uma alternativa interessante para acessar diferentes tipos de banco de dados. O PDO é uma camada de abstração de banco de dados que permite utilizar as mesmas funções para diferentes tipos de bancos de dados, facilitando a portabilidade do código entre diferentes plataformas de banco de dados.

11.4 Executando consultas SQL com o PHP

Neste tópico, abordaremos o tema de executar consultas SQL com PHP. Quando se trata de interação com bancos de dados, é comum precisar buscar informações específicas ou realizar ações em conjunto com o banco

de dados. O PHP é uma linguagem que permite executar consultas SQL em bancos de dados de várias formas, desde as mais simples até as mais complexas.

Para executar uma consulta SQL em PHP, é necessário criar uma conexão com o banco de dados. Isso pode ser feito utilizando a extensão mysqli ou a extensão PDO. A extensão mysqli é específica para o banco de dados MySQL, enquanto a PDO oferece suporte a vários bancos de dados, incluindo o MySQL, Oracle e PostgreSQL.

Uma consulta SQL pode ser dividida em três partes: SELECT, FROM e WHERE. O SELECT é responsável por selecionar as colunas que você deseja exibir, o FROM indica a tabela na qual você está executando a consulta e o WHERE é usado para filtrar os resultados com base em determinados critérios.

Vejamos um exemplo de consulta SQL simples utilizando a extensão mysqli:

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";
```

```php
// Cria conexão
$conn = mysqli_connect($servername, $username, $password, $dbname);

// Verifica conexão
if (!$conn) {
  die("Conexão falhou: " . mysqli_connect_error());
}
```

```php
// Executa a consulta SQL
$sql = "SELECT * FROM customers WHERE city='São Paulo'";
$result = mysqli_query($conn, $sql);

// Verifica se há resultados
if (mysqli_num_rows($result) > 0) {
  // Exibe os dados de cada linha
  while($row = mysqli_fetch_assoc($result)) {
    echo "Nome: " . $row["name"]. " - Email: " . $row["email"]. "<br>";
  }
} else {
  echo "0 resultados";
}

// Fecha conexão
mysqli_close($conn);
?>
```

Neste exemplo, a consulta SQL seleciona todos os dados da tabela "customers" onde a cidade é "São Paulo". O resultado é então exibido em um loop enquanto há linhas de resultado. Caso não haja resultados, é exibida a mensagem "0 resultados".

11.5 Utilizando PDO para acessar diferentes tipos de banco de dados

O PDO (PHP Data Objects) é uma extensão do PHP que fornece uma interface genérica para acessar diferentes tipos de bancos de dados, como MySQL, SQLite, Oracle, PostgreSQL e outros. Ele oferece uma maneira segura e eficiente de trabalhar com bancos de dados usando o PHP.

Uma das principais vantagens do PDO é que ele fornece uma camada de abstração entre o PHP e o banco de dados, permitindo que o código seja portado facilmente para outros bancos de dados sem precisar alterar o código da aplicação. Além disso, o PDO oferece suporte a recursos avançados, como transações, prepared statements e fetch modes.

Para utilizar o PDO, é necessário criar uma conexão com o banco de dados desejado. Isso pode ser feito utilizando o construtor da classe PDO, que recebe como parâmetros o nome do driver do banco de dados, o nome do banco de dados, o nome do usuário e a senha de acesso:

```php
$dsn = 'mysql:host=localhost;dbname=mydatabase';
$username = 'myusername';
$password = 'mypassword';

try {
    $pdo = new PDO($dsn, $username, $password);
} catch (PDOException $e) {
    echo 'Connection failed: ' . $e->getMessage();
}
```

O exemplo acima cria uma conexão com o banco de dados MySQL utilizando o driver PDO_MYSQL. Caso a conexão seja bem sucedida, uma instância da classe PDO é retornada. Caso contrário, uma exceção do tipo PDOException é lançada.

Uma vez criada a conexão, é possível executar consultas SQL utilizando o método query da classe PDO:

```php
$sql = 'SELECT * FROM users';
$stmt = $pdo->query($sql);

while ($row = $stmt->fetch()) {
    echo $row['name'] . '<br />';
}
```

O exemplo acima executa uma consulta SQL para selecionar todos os registros da tabela users e exibe o nome de cada usuário na tela.

Além do método query, o PDO oferece suporte a prepared statements, que permitem que as consultas SQL sejam preparadas uma vez e executadas várias vezes com diferentes parâmetros. Isso pode melhorar significativamente o desempenho da aplicação e também torná-la mais segura contra ataques de SQL injection.

Para utilizar prepared statements com o PDO, é necessário utilizar o método prepare em vez do método query:

```php
$sql = 'SELECT * FROM users WHERE email = :email';
$stmt = $pdo->prepare($sql);
$stmt->execute(['email' => 'john@example.com']);

while ($row = $stmt->fetch()) {
    echo $row['name'] . '<br />';
}
```

O exemplo acima utiliza um prepared statement para selecionar todos os registros da tabela users que possuem o endereço de e-mail 'john@example.com'. O valor do parâmetro email é passado através de um array associativo para o método execute.

O PDO é uma ferramenta poderosa para acessar diferentes tipos de bancos de dados com o PHP. Ele oferece uma maneira segura e eficiente de trabalhar com bancos de dados, além de fornecer recursos avançados como prepared statements e fetch modes. Com o uso correto do PDO, é possível criar aplicações seguras, eficientes e portáveis entre diferentes bancos de dados.

12.0 ORIENTAÇÃO A OBJETOS EM PHP

A orientação a objetos (OO) é um paradigma de programação que permite modelar objetos do mundo real em um programa de computador. Em PHP, a OO é uma característica essencial que permite criar código mais organizado, reutilizável e modular.

Segundo Davis e Phillips (2014), a OO em PHP se baseia em quatro pilares: encapsulamento, herança, polimorfismo e abstração. Cada um desses conceitos ajuda a estruturar o código de forma a torná-lo mais fácil de entender, manter e expandir.

O encapsulamento é o conceito que permite esconder a implementação de um objeto, expondo somente a interface pública. Isso significa que outras partes do código não precisam saber como um objeto foi implementado, apenas como utilizá-lo. Isso reduz a complexidade do código e torna mais fácil sua manutenção.

A herança é a capacidade de uma classe herdar propriedades e métodos de outra classe. Isso permite criar classes mais específicas a partir de classes mais genéricas. Segundo Mazza (2014), a herança é um dos conceitos mais importantes da OO e é amplamente utilizada em PHP.

O polimorfismo é a capacidade de um objeto assumir diferentes formas e comportamentos, dependendo do contexto em que é utilizado. Isso significa que uma mesma função pode ser aplicada a diferentes objetos, com resultados diferentes. Isso torna o código mais flexível e genérico.

A abstração é o conceito que permite criar classes abstratas, que não podem ser instanciadas diretamente, mas podem ser utilizadas como base para outras classes. Isso permite criar classes mais genéricas e reutilizáveis.

12.1 Conceitos básicos de OOP

A programação orientada a objetos é uma forma de desenvolvimento de software que se baseia na criação de objetos que interagem entre si. Esses objetos possuem características (propriedades) e ações (métodos) que são definidas por meio de classes.

Em PHP, assim como em outras linguagens de programação orientadas a objetos, as classes são a base para a criação de objetos. As classes podem ser entendidas como um molde para a criação de objetos. Segundo Davis e Phillips (2009), "Uma classe é uma estrutura que define as propriedades e métodos que um objeto do tipo dessa classe terá". As propriedades são as variáveis que definem as características do objeto, enquanto os métodos são as funções que definem as ações que o objeto pode executar.

A utilização de classes e objetos permite a criação de um código mais organizado e reutilizável, além de oferecer maior segurança e facilidade na manutenção do software. No entanto, para utilizar esses conceitos, é necessário entender a estrutura básica de uma classe em PHP, que é composta por:

A declaração da classe com a palavra-chave "class";

O nome da classe, que deve ser iniciado por uma letra maiúscula;

A definição das propriedades e métodos da classe;

A criação de objetos a partir da classe com a palavra-chave "new".

O conceito de classes e objetos está diretamente relacionado com o paradigma de orientação a objetos em PHP. A programação orientada a objetos oferece diversas vantagens para o desenvolvimento de software, como a encapsulação, herança e polimorfismo. Esses conceitos são essenciais para o desenvolvimento de sistemas complexos e de grande porte.

12.2 Criando e utilizando classes e objetos em PHP

As classes são um elemento fundamental da programação orientada a objetos (OOP) e permitem que os desenvolvedores definam estruturas de dados personalizadas com suas próprias propriedades e métodos. Os objetos são instâncias dessas classes e possuem seus próprios valores de propriedade exclusivos.

Para criar uma classe em PHP, utilizamos a palavra-chave "class" seguida pelo nome da classe. Vamos criar um exemplo de classe simples para entendermos melhor:

```php
class Pessoa {
    public $nome;
    public $idade;

    public function imprimirNome() {
        echo "O nome da pessoa é " . $this->nome;
    }
}
```

Nesse exemplo, criamos uma classe chamada "Pessoa" com duas propriedades (nome e idade) e um método chamado "imprimirNome". A propriedade "public" indica que as propriedades e métodos podem ser acessados de fora da classe.

Para criar um objeto dessa classe, utilizamos a palavra-chave "new" seguida pelo nome da classe e parênteses vazios:

```php
$pessoa1 = new Pessoa();
```

Agora temos um objeto chamado $pessoa1 que podemos utilizar para acessar as propriedades e métodos da classe. Por exemplo, podemos definir o nome da pessoa e imprimir seu nome utilizando os seguintes comandos:

```
$pessoa1->nome = "João";
$pessoa1->imprimirNome();
```

Esse código irá imprimir "O nome da pessoa é João" no navegador.

A utilização de classes e objetos permite que os desenvolvedores criem código mais modular, reutilizável e fácil de manter. Por exemplo, podemos criar uma classe "Produto" com métodos para adicionar, remover e exibir produtos, e depois utilizar essa classe em diferentes partes de nosso site ou aplicativo.

Segundo Davis e Phillips (2005), "A utilização de classes permite que os desenvolvedores se concentrem em criar estruturas de dados e algoritmos reutilizáveis em vez de lidar com detalhes de implementação em cada parte do código."

A criação e utilização de classes e objetos é uma parte fundamental da programação orientada a objetos em PHP e pode ajudar a tornar nosso código mais modular e fácil de manter.

Herança, encapsulamento e polimorfismo em PHP

Vejamos alguns exemplos de como utilizar herança, encapsulamento e polimorfismo em PHP.

Herança:

A herança permite que uma classe filha herde propriedades e métodos da classe pai. Veja um exemplo:

```php
class Animal {
    public $name;
    public function __construct($name) {
        $this->name = $name;
    }
    public function makeSound() {
        echo "Animal making sound";
    }
}

class Dog extends Animal {
    public function makeSound() {
        echo "Bark";
    }
}

$dog = new Dog("Buddy");
echo $dog->name . " says ";
$dog->makeSound();
```

Nesse exemplo, a classe Dog herda a propriedade name e o método makeSound() da classe Animal. Ao criar um novo objeto Dog, passamos o nome Buddy como parâmetro para o construtor da classe pai. Ao chamar o método makeSound(), o método da classe filha é executado e a mensagem Bark é exibida.

Encapsulamento:

```php
class Person {
    private $name;
    public function setName($name) {
        $this->name = $name;
    }
    public function getName() {
        return $this->name;
    }
}
```

```php
$person = new Person();
$person->setName("John");
echo $person->getName();
```

O encapsulamento permite controlar o acesso aos membros de uma classe, definindo-os como públicos, privados ou protegidos.

No exemplo anterior, a propriedade $name é definida como privada, o que significa que só pode ser acessada de dentro da própria classe. Para acessar e modificar o valor da propriedade, utilizamos os métodos setName() e getName(), que são públicos e podem ser acessados de fora da classe. Assim, garantimos o encapsulamento da propriedade $name.

Polimorfismo:

O polimorfismo permite que um método tenha diferentes comportamentos em classes diferentes. Veja um exemplo:

```php
interface Animal {
    public function makeSound();
}

class Dog implements Animal {
    public function makeSound() {
        echo "Bark";
    }
}

class Cat implements Animal {
    public function makeSound() {
        echo "Meow";
    }
}

function makeAnimalSound(Animal $animal) {
    $animal->makeSound();
}

$dog = new Dog();
$cat = new Cat();
makeAnimalSound($dog); // Bark
makeAnimalSound($cat); // Meow
```

Nesse exemplo, a interface Animal define o método makeSound(). As classes Dog e Cat implementam essa interface e fornecem suas próprias implementações do método makeSound(). O método makeAnimalSound()

recebe um objeto que implementa a interface Animal e chama o método makeSound() desse objeto. Assim, podemos chamar o mesmo método para diferentes objetos e obter comportamentos diferentes, graças ao polimorfismo.

13.0 O QUE É MVC?

MVC é uma arquitetura de software que consiste em dividir uma aplicação em três componentes distintos: Modelo, Visão e Controlador. O Modelo é responsável por lidar com os dados e regras de negócio, a Visão é responsável pela apresentação da interface gráfica ao usuário e o Controlador é responsável por gerenciar a interação entre o Modelo e a Visão.

Um dos principais objetivos do uso de MVC é a separação de responsabilidades. Com a separação dos componentes, é possível desenvolver cada um deles de forma independente, permitindo uma maior flexibilidade na manutenção do código. Além disso, o uso de MVC torna o código mais organizado e de fácil entendimento, facilitando a implementação de novas funcionalidades.

Outro benefício do uso de MVC é a melhoria da escalabilidade. Como cada componente é independente, é possível escalar apenas o que é necessário, evitando a necessidade de escalar toda a aplicação.

Segundo Martin Fowler, um dos principais nomes da literatura de programação, "o objetivo da arquitetura é minimizar a dor do desenvolvimento a longo prazo, mantendo a flexibilidade no curto prazo". O uso de MVC é uma forma de alcançar esse objetivo, pois permite a separação de responsabilidades e a melhoria da escalabilidade, tornando o desenvolvimento e a manutenção do código mais fácil e flexível.

Um exemplo prático do uso de MVC em linguagem PHP é o framework Laravel. O Laravel é um dos frameworks mais populares de PHP e utiliza o padrão de arquitetura MVC para desenvolver aplicações web de forma organizada e escalável. Com o Laravel, é possível separar as responsabilidades do Modelo, Visão e Controlador de forma clara e objetiva, facilitando o desenvolvimento e manutenção do código.

O uso de MVC em linguagem PHP é uma forma de organizar e estruturar o código de forma clara e objetiva, facilitando a manutenção e o desenvolvimento de novas funcionalidades. Além disso, o uso de MVC permite uma melhoria na escalabilidade da aplicação, evitando a necessidade de escalar toda a aplicação de forma desnecessária.

13.1 O que é Model?

O Model é uma das três partes do padrão de arquitetura de software MVC (Model-View-Controller). Ele é responsável por lidar com a lógica de negócios da aplicação, incluindo a comunicação com o banco de dados e a manipulação dos dados. O Model é a camada que representa a estrutura de dados da aplicação.

Responsabilidades do Model

As responsabilidades do Model incluem a validação de dados, a lógica de negócios e a manipulação de dados. A validação de dados é importante para garantir que as informações inseridas na aplicação sejam válidas e coerentes. A lógica de negócios é a parte da aplicação que define as regras que determinam como os dados serão manipulados e como a aplicação deve se comportar em diferentes situações. A manipulação de dados inclui a leitura, gravação, atualização e exclusão de dados do banco de dados.

Criação de Models em PHP

Para criar um Model em PHP, primeiro é necessário definir a estrutura de dados da aplicação. Isso envolve a criação de classes e propriedades que representam as tabelas e campos do banco de dados. Em seguida, é preciso definir os métodos que irão manipular esses dados, incluindo operações de leitura, gravação, atualização e exclusão. É importante seguir boas práticas de programação e padronizar a nomenclatura dos métodos e propriedades, a fim de manter a consistência e facilitar a manutenção do código.

Exemplo prático de criação de um Model

Um exemplo prático de criação de um Model em PHP seria a criação de um sistema de gerenciamento de produtos em uma loja virtual. Nesse caso,

seria necessário criar uma classe Produto que representasse a tabela de produtos do banco de dados, com propriedades como id, nome, descrição, preço, etc. Em seguida, seriam criados métodos para realizar operações como listar todos os produtos, buscar um produto por id, adicionar um novo produto, atualizar um produto existente e excluir um produto.

13.2 O que é View?

View é uma parte essencial do padrão de projeto MVC (Model-View-Controller). Ela é responsável por representar os dados do Model na interface do usuário, ou seja, é a camada que cuida da exibição dos dados e interação do usuário com a aplicação. É através da View que o usuário pode ver e interagir com os dados do sistema.

Responsabilidades da View:

A View é responsável por receber os dados do Model e exibi-los na tela para o usuário. Ela é responsável por apresentar os dados em uma forma amigável para o usuário, ou seja, com uma interface gráfica atrativa e de fácil utilização. A View também pode ser responsável por receber a interação do usuário e repassá-la para o Controller.

Criação de Views em PHP:

Para criar uma View em PHP, é preciso criar um arquivo com extensão .php que irá conter a estrutura HTML da página. É possível usar tags PHP para adicionar código PHP que irá processar os dados e exibi-los na tela. É importante ressaltar que a View não deve conter nenhuma regra de negócio ou acesso ao banco de dados, essas responsabilidades devem ser tratadas pelo Model e pelo Controller.

Exemplo prático de criação de uma View:

Suponha que temos um sistema de cadastro de clientes. A View será responsável por exibir na tela as informações dos clientes cadastrados. Para criar uma View em PHP que irá exibir os dados, podemos criar um arquivo chamado "clientes.php" e inserir a estrutura HTML da página. Em seguida, podemos adicionar código PHP que irá processar os dados e exibi-los na tela. Por exemplo:

```
<!DOCTYPE html>
<html>
<head>
    <title>Lista de Clientes</title>
</head>
<body>
    <h1>Lista de Clientes</h1>
    <table>
        <tr>
            <th>Nome</th>
            <th>E-mail</th>
            <th>Telefone</th>
        </tr>
        <?php foreach ($clientes as $cliente): ?>
        <tr>
            <td><?= $cliente->nome ?></td>
            <td><?= $cliente->email ?></td>
            <td><?= $cliente->telefone ?></td>
        </tr>
        <?php endforeach; ?>
    </table>
</body>
</html>
```

Nesse exemplo, temos a estrutura HTML da página com uma tabela que irá exibir os dados dos clientes. Usamos um loop foreach para percorrer a lista de clientes e exibi-los na tabela.

13.3 O que é Controller?

O Controller é uma das partes do padrão arquitetural MVC (Model-View-Controller) utilizado em programação web com o objetivo de separar as responsabilidades e melhorar a organização do código fonte. O Controller é responsável por receber as requisições do usuário e manipular os dados para que sejam exibidos corretamente na View.

Responsabilidades do Controller

O Controller é responsável por processar as informações enviadas pelo usuário por meio de formulários ou outras formas de entrada de dados. Ele tem a tarefa de validar esses dados, realizar as operações necessárias no Model e retornar o resultado final para a View.

Criação de Controllers em PHP

Em PHP, a criação de um Controller envolve a criação de uma classe que estende a classe Controller do framework utilizado. Essa classe deve conter métodos que correspondam às ações que o usuário pode realizar na aplicação. Cada ação deve ser responsável por receber os dados enviados pelo usuário, validar esses dados, executar as operações necessárias no Model e retornar o resultado para a View.

Exemplo prático de criação de um Controller

Vamos supor que estamos criando um sistema de cadastro de usuários em PHP utilizando o padrão arquitetural MVC. O Controller será responsável por receber os dados de cadastro enviados pelo usuário, validar esses dados e executar as operações necessárias no Model. Para isso, podemos criar uma classe UserController que estende a classe Controller do framework utilizado. Essa classe terá um método register, responsável por receber os dados de cadastro e executar as operações necessárias no Model, e um método login, responsável por receber os dados de login e autenticar o usuário.

13.4 Integração entre Model, View e Controller

A arquitetura MVC é dividida em três camadas distintas: Model, View e Controller. Cada camada possui suas responsabilidades específicas, mas é a integração dessas camadas que permite que a aplicação funcione de maneira eficiente. Neste texto, vamos abordar como as três camadas se comunicam e o fluxo de dados em uma aplicação MVC, além de um exemplo prático de integração entre Model, View e Controller.

Como as três camadas se comunicam?

A comunicação entre as três camadas ocorre de forma organizada e estruturada, conforme o fluxo de dados da aplicação. O Controller recebe as requisições do usuário e as direciona para o Model, que processa as informações e retorna os dados solicitados para o Controller. Em seguida, o Controller passa esses dados para a View, que é responsável pela exibição da informação para o usuário.

Fluxo de dados em uma aplicação MVC

O fluxo de dados em uma aplicação MVC é baseado no padrão de requisição e resposta. Quando o usuário realiza uma requisição, o Controller recebe essa requisição e a trata, geralmente enviando-a para o Model. O Model processa a informação recebida, acessando o banco de dados, fazendo cálculos ou executando outras tarefas. Em seguida, o Model retorna as informações processadas para o Controller, que passa esses dados para a View. A View então exibe a informação processada para o usuário.

Exemplo prático de integração entre Model, View e Controller

Para exemplificar a integração entre Model, View e Controller, podemos pensar em uma aplicação de cadastro de usuários. O usuário preenche um formulário com suas informações e clica em um botão para enviar as informações. Esse botão envia uma requisição para o Controller, que recebe essas informações e as envia para o Model. O Model recebe as informações e verifica se o usuário já existe no banco de dados. Caso o usuário não exista, o Model cria um novo usuário com as informações recebidas e retorna essa informação para o Controller. O Controller então passa essas informações para a View, que exibe uma mensagem de sucesso para o usuário. Caso o usuário já exista no banco de dados, o Model retorna essa informação para o Controller, que passa para a View, que exibe uma mensagem de erro informando que o usuário já está cadastrado.

Segue abaixo um exemplo em PHP de integração entre Model, View e Controller usando o padrão MVC:

Model (modelo) - responsável por gerenciar os dados da aplicação:

```php
class UserModel {
    private $users = array(
        1 => array("id" => 1, "name" => "João", "email" => "joao@teste.com"),
        2 => array("id" => 2, "name" => "Maria", "email" => "maria@teste.com"),
        3 => array("id" => 3, "name" => "Pedro", "email" => "pedro@teste.com"),
    );
```

```
    public function getUsers() {
        return $this->users;
    }

    public function getUserById($id) {
        return isset($this->users[$id]) ? $this->users[$id] : null;
    }

    public function addUser($user) {
        $this->users[$user['id']] = $user;
    }

    public function deleteUser($id) {
        unset($this->users[$id]);
    }
}
```

View (visão) - responsável por exibir os dados da aplicação:

```
class UserView {
    public function output($data) {
        echo "<table>";
        foreach($data as $user) {
            echo "<tr><td>".$user['id']."</td><td>".$user['name']."</td><td>".$user['email']."</td></tr>";
        }
        echo "</table>";
    }
```

```
    public function form() {
        echo "<form method='post' action=''>";
        echo "ID: <input type='text' name='id'><br>";
        echo "Name: <input type='text' name='name'><br>";
        echo "Email: <input type='text' name='email'><br>";
        echo "<input type='submit' value='Add'>";
        echo "</form>";
    }
}
```

Controller (controlador) - responsável por gerenciar as requisições do usuário:

```php
class UserController {
    private $model;
    private $view;

    public function __construct($model, $view) {
        $this->model = $model;
        $this->view = $view;
    }

    public function index() {
        $data = $this->model->getUsers();
        $this->view->output($data);
    }
```

```php
    public function add() {
        if(isset($_POST['id']) && isset($_POST['name']) && isset($_POST['email'])) {
            $user = array(
                "id" => $_POST['id'],
                "name" => $_POST['name'],
                "email" => $_POST['email']
            );
            $this->model->addUser($user);
        }

        $this->view->form();
    }

    public function delete($id) {
        $this->model->deleteUser($id);
    }
}
```

Para integrar as três camadas, podemos criar um arquivo index.php que instancie as classes Model, View e Controller e chame seus respectivos métodos:

```php
require_once("model/UserModel.php");
require_once("view/UserView.php");
require_once("controller/UserController.php");

$model = new UserModel();
$view = new UserView();
$controller = new UserController($model, $view);

$action = isset($_GET['action']) ? $_GET['action'] : 'index';

switch($action) {
   case 'index':
      $controller->index();
      break;
   case 'add':
      $controller->add();
      break;
   case 'delete':
      $id = isset($_
```

14.0 FRAMEWORKS MVC

Frameworks MVC são ferramentas de software que fornecem uma estrutura pré-definida para construir aplicativos usando a arquitetura MVC. Eles ajudam os desenvolvedores a manter um código organizado e escalável, permitindo a separação clara de responsabilidades entre as camadas Model, View e Controller.

Um Framework MVC típico possui uma estrutura de diretórios que separa os arquivos do modelo, visão e controlador. Além disso, o framework geralmente fornece um conjunto de bibliotecas e funções que facilitam a criação de aplicativos da web.

Alguns exemplos populares de frameworks MVC em PHP incluem Laravel, CodeIgniter, Symfony e CakePHP. Esses frameworks fornecem recursos para tornar a programação em PHP mais eficiente, fornecendo recursos pré-construídos para manipulação de banco de dados, autenticação, segurança e muito mais.

O uso de um framework MVC oferece muitos benefícios para o desenvolvimento de aplicativos da web em PHP. O primeiro benefício é a economia de tempo e esforço para criar um aplicativo do zero. Com o uso de um framework MVC, grande parte da estrutura e funcionalidade básica já está pronta, permitindo que os desenvolvedores se concentrem em desenvolver recursos mais complexos e específicos para o aplicativo.

Além disso, o uso de um framework MVC ajuda a garantir que o código seja organizado e fácil de manter. A separação clara de responsabilidades entre o modelo, visão e controlador ajuda a minimizar a confusão no código e garante que cada camada esteja se concentrando em seu próprio conjunto de tarefas.

Para utilizar um Framework MVC, é necessário seguir a documentação fornecida pelo próprio Framework. Normalmente, é necessário instalar o framework em um servidor web, configurá-lo para funcionar com um banco de dados e iniciar a criação do aplicativo seguindo a estrutura do framework.

Os Frameworks MVC em PHP oferecem uma estrutura organizada para o desenvolvimento de aplicativos web, economizando tempo e esforço, além de ajudar na manutenção do código. Os desenvolvedores podem escolher o framework que melhor se adapta às suas necessidades e seguir a documentação fornecida para construir aplicativos complexos e escaláveis.

15.0 REFERÊNCIAS GERAIS

CSS Tricks - A Complete Guide to Border: https://css-tricks.com/border/

MDN Web Docs. HTML div tag. Disponível em: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/div

DUCKETT, Jon. HTML and CSS: Design and Build Websites. John Wiley & Sons, 2011.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design patterns: elements of reusable object-oriented software. Addison-Wesley.

FOWLER, Martin. Patterns of Enterprise Application Architecture. Addison-Wesley Professional, 2002.

MORRISON, Michael. Head First HTML and CSS: A Learner's Guide to Creating Standards-Based Web Pages. O'Reilly Media, 2012.

ZELDMAN, Jeffrey. Designing with Web Standards. New Riders, 2003.

MAZZA, Lucas. HTML5 e CSS3 - Domine a Web do Futuro. São Paulo: Casa do Código, 2014.

NIERADKA, Itamar Pena: Javascript+CSS+DOM, desenvolvimento para web. Nova Terra, 2014.

DAVIS M.E., PHILLIPS, J. A. Aprendendo PHP e MySQL. 1 ed Alta Books, 2019.

16 SUGESTÕES DE ATIVIDADES

Capítulo 01 - Pesquisa

Crie um texto com mínimo de 20 linhas sobre o seguinte tópico de discussão:

A informação tornou-se mais valiosa com a chegada da internet?

Capítulo 02 - Projeto de produção

Crie um pequeno projeto de um site

regras:

Deverão existir 03 páginas;

As páginas deverão estar linkadas entre elas;

Deverá ser inserida imagens e listas em pelo menos uma página;

Todo o código das páginas deverá manter uma estrutura em html;

Capítulo 03 - Projeto de produção

Ao projeto feito no capítulo anterior, crie uma nova página que deverá ter um formulário.

O formulário deverá ter os seguintes campos: Nome, sobrenome e sugestão (caixa de texto).

Capítulo 04 - Projeto de produção

Adicione um arquivo de estilo ao site criado;

Crie uma classe que transforme a fonte de todo parágrafo do site em tipo calibri.

Capítulo 05 - Projeto de produção

Ao projeto do site adicione:

Margens de 10% em todo corpo do site;

Padding de 2px para todos os lados nas imagens do site;

Faça uma página nova onde deverá existir uma tabela 2x2 com bordas tracejadas.

Capítulo 06 - Projeto de produção

Coloque todo o projeto do site dentro do servidor Wamp e faça ficar acessível por um ip dentro de uma rede local.

Capítulo 07 - Pesquisa

Crie um texto com mínimo de 20 linhas explicando o que são linguagens que funcionam somente do lado do servidor e qual a importância dessa estratégia?

Capítulo 08 - Pesquisa

Crie um texto com mínimo de 20 linhas explicando qual a importância do controle de fluxo na programação.

Capítulo 09 - Projeto de produção

Faça um código em php que tenha uma variável que receberá a palavra chocolate.

Deverá ser mostrada através do comando Echo o tamanho da palavra e também sua escrita em maiúscula.

Capítulo 10 - Projeto de produção

Crie uma nova página em nosso site que tenha um formulário que peça a digitação de uma palavra e um botão enviar.

Ao pressionar o botão enviar deverá aparecer em tela quantas letras tem a palavra.

Capítulo 11 - Projeto de produção

Crie uma base de dados com os seguintes campos: nome, sobrenome e idade.

Faça uma página com um formulário que receba esses valores e os coloque no banco.

Capítulo 12 - Pesquisa

Crie um texto com mínimo de 20 linhas explicando o que significa linguagem orientada a objeto.

Capítulo 13 - Pesquisa

Crie um texto com mínimo de 20 linhas explicando a importância do uso do MVC em projetos.

Capítulo 14 - Pesquisa

Crie um texto com mínimo de 20 linhas explicando qual a utilidade de um framework MVC, qual usaria em um projeto e o porquê dessa escolha.

CURSO TÉCNICO EM

INFORMÁTICA

MÓDULO III

Estrutura de Dados

Amanda Souza

1. **Introdução**
- Estruturas de dados
- Importância das estruturas de dados

2. **Arrays/Vetores**
- O que são arrays/vetores?
  Quando os arrays devem ser utilizados ?
- Criação e manipulação em código com a linguagem C
- Vantagens e desvantagens
- Exercício prático: criar um programa em C que utilize arrays/vetores para armazenar informações

3. **Listas Encadeadas**
- O que são listas encadeadas?
- Criação e manipulação em código com a linguagem C
- Vantagens e desvantagens
- Exercício prático: criar um programa em C que utilize listas encadeadas para armazenar informações

4. **Pilhas**
- O que é uma pilha?
- Criação e manipulação em código com a linguagem C
  Vantagens e desvantagens
- Exercício prático: criar um programa em C que utilize pilhas

5. **Filas**
- O que é uma fila?
- Criação e manipulação em código com a linguagem C
- Exercício prático: criar um programa em C que utilize filas para simular o atendimento a clientes em uma loja

6. **Árvores**
- O que são árvores?
- Tipos de árvores (binária, balanceada, B)
- Criação e manipulação em código com a linguagem C
- Exercício prático: criar um programa em C que utilize árvores binárias para armazenar informações e realizar buscas

Introdução

A ciência da computação é uma disciplina que envolve o desenvolvimento de software, hardware e tecnologias relacionadas. Uma das bases da programação são os algoritmos e estruturas de dados.

Os algoritmos são conjuntos de instruções bem definidas e organizadas logicamente para executar uma tarefa ou resolver um problema específico. Eles são essenciais na programação, pois permitem que artistas de software criem soluções computacionais eficientes e confiáveis.

As estruturas de dados, por sua vez, fornecem uma maneira organizada de armazenar e gerenciar informações em um programa. Elas são a base do design de software escalonável, permitindo que os desenvolvedores trabalhem com grandes conjuntos de dados e minimizando a complexidade do código.

As estruturas de dados mais comuns incluem arrays/vetores, listas encadeadas, filas, pilhas, árvores e grafos. Cada uma dessas estruturas tem suas próprias vantagens e desvantagens, e é importante escolher a estrutura correta para cada aplicação.

Uma das principais funções dos algoritmos é manipular as estruturas de dados para realizar operações específicas. Por exemplo, um algoritmo de busca pode procurar um item específico em uma lista ou array, enquanto um algoritmo de ordenação pode reorganizar os elementos de uma lista ou array em ordem crescente ou decrescente.

O uso correto de algoritmos e estruturas de dados é fundamental para o desenvolvimento de programas eficientes e escalonáveis. A habilidade de escolher a estrutura de dados correta para cada aplicação e implementar

algoritmos eficientes é uma das principais habilidades que um programador deve dominar.

Nesta apostila, vamos explorar em detalhes os conceitos de algoritmos e estruturas de dados, suas aplicações na programação e como você pode utilizá-los para melhorar a performance do seu código e desenvolver soluções robustas e escalonáveis para problemas computacionais.

Importância na programação

A programação é uma das disciplinas mais importantes e em rápida evolução da ciência da computação. É através da programação que desenvolvemos softwares, aplicativos e sistemas que são essenciais para a nossa vida cotidiana.

Algoritmos e estruturas de dados são fundamentais para a programação. Eles são usados para resolver problemas computacionais complexos, realizar tarefas específicas e gerenciar grandes quantidades de dados. São conceitos que fornecem as bases teóricas e práticas para o desenvolvimento de programas eficientes, escaláveis e seguros.

A escolha do algoritmo correto pode fazer a diferença entre um programa rápido e eficiente e um programa lento e ineficiente. Por exemplo, um algoritmo simples de busca linear pode funcionar bem para uma pequena quantidade de dados, mas pode ser muito lento quando se trata de grandes conjuntos de dados. Nesses casos, um algoritmo mais sofisticado, como a busca binária, pode ser mais adequado.

Da mesma forma, a escolha da estrutura de dados correta pode ter um grande impacto no desempenho de um programa. Uma lista encadeada pode ser mais adequada do que um array/vetor para armazenar dados que precisam ser inseridos ou removidos com frequência, enquanto um array/vetor pode ser

mais adequado para armazenar dados que precisam ser acessados com frequência.

Além disso, os algoritmos e estruturas de dados são essenciais para a resolução de problemas computacionais complexos e para o desenvolvimento de soluções inovadoras. Eles são frequentemente usados em áreas como análise de dados, inteligência artificial, aprendizado de máquina e sistemas distribuídos.

Em suma, os algoritmos e estruturas de dados são fundamentais para a programação. Eles fornecem as bases teóricas e práticas para o desenvolvimento de programas eficientes e escaláveis, permitindo que os desenvolvedores criem soluções computacionais inovadoras e resolvam problemas complexos.

Nesta apostila, vamos explorar em detalhes esses conceitos e como utilizá-los para melhorar a performance do seu código e desenvolver soluções robustas e escalonáveis para problemas computacionais.

1.1. Estruturas de dados

Estruturas de dados são elementos fundamentais da programação e da computação em geral. Elas se referem a maneiras específicas de organizar e armazenar dados para que possam ser acessados, lidos, modificados e excluídos de forma eficiente.

Em outras palavras, as estruturas de dados definem a forma como os dados são organizados e manipulados em um programa de computador. Por exemplo, uma lista de compras pode ser organizada de diferentes formas, como em um papel solto, em um bloco de notas ou em um aplicativo de celular. Cada uma dessas opções representa uma estrutura diferente para armazenar a mesma informação.

Existem diversas estruturas de dados disponíveis, cada uma com suas próprias vantagens e desvantagens. A escolha da estrutura correta pode fazer uma grande diferença no desempenho e funcionalidade do programa.

A seguir, vamos explorar algumas das principais estruturas de dados utilizadas na programação, incluindo arrays/vetores, listas encadeadas, pilhas, filas, árvores, grafos e tabelas de dispersão (hash tables). Cada uma dessas estruturas tem suas próprias características, aplicações e formas de implementação, mas todas têm o objetivo comum de organizar os dados de forma eficiente e escalável.

As estruturas de dados são essenciais para a programação, fornecendo as bases teóricas e práticas para o desenvolvimento de programas eficientes e escaláveis. Nesta apostila, vamos explorar em detalhes as principais estruturas de dados utilizadas na programação e como utilizá-las para criar soluções eficientes e inovadoras.

1.2. A importância das estruturas de dados

As estruturas de dados são elementos fundamentais da programação, permitindo que os programadores manipulem grandes quantidades de dados de forma organizada, eficiente e escalável. Nesta apostila, vamos explorar a importância das estruturas de dados na programação e na computação em geral.

Organização eficiente de dados

Uma das principais vantagens das estruturas de dados é a sua capacidade de organizar dados de forma eficiente. Isso significa que as estruturas de dados permitem que os dados sejam armazenados de tal forma que seja fácil acessá-los e manipulá-los em um programa de computador. Sem uma estrutura adequada para armazenamento de dados, o programa pode sofrer desperdício de recursos de processamento e levar muito tempo para ser executado.

Algoritmos eficientes

A escolha da estrutura de dados correta também é importante para construir algoritmos eficientes. Algoritmos são procedimentos lógicos que resolvem problemas específicos. Uma estrutura de dados inadequada pode levar a algoritmos ineficientes e isso pode impactar significativamente a performance do programa. Por outro lado, uma estrutura de dados bem escolhida pode facilitar a criação de algoritmos que economizam tempo e recursos de processamento.

Reutilização de código

Outra vantagem das estruturas de dados é que elas podem ser reutilizadas em diferentes programas, tornando a programação mais eficiente. Em vez de criar uma nova estrutura de dados para cada problema diferente, um programador pode utilizar estruturas de dados já existentes e adaptá-las às suas necessidades específicas. Isso economiza tempo e esforço, além de facilitar o processo de desenvolvimento de softwares.

Compreensão da teoria da computação

O estudo das estruturas de dados também é importante para a compreensão da teoria da computação e da ciência da computação em geral. Aprender sobre as diferentes estruturas de dados ajuda os programadores a entender como os computadores funcionam, como os programas são executados e como a informação é processada.

A importância das estruturas de dados na programação

As estruturas de dados são elementos fundamentais da programação, permitindo que os programadores manipulem grandes quantidades de dados de forma organizada, eficiente e escalável. Além disso, elas são importantes para a construção de algoritmos eficientes, reutilização de código e compreensão da teoria da computação. Em resumo, o conhecimento das estruturas de dados é

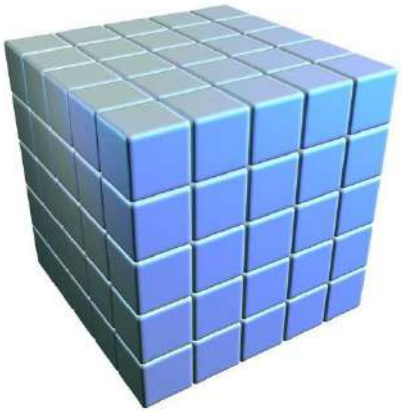
essencial para qualquer programador que deseje criar soluções inovadoras e eficientes.

2. Arrays/Vetores

Arrays, também conhecidos como vetores, são estruturas de dados que permitem armazenar uma coleção de elementos do mesmo tipo. Os elementos do array são organizados em um conjunto ordenado e cada elemento é acessado por um índice numérico.

Em outras palavras, um array é uma sequência contínua de variáveis do mesmo tipo, todas armazenadas consecutivamente na memória. O acesso a um elemento individual do array é feito através de um índice inteiro que especifica a posição do elemento na sequência.

As vantagens dos arrays incluem sua simplicidade de implementação, facilidade de utilização e eficiência na busca de elementos. Por outro lado, as desvantagens incluem seu tamanho fixo (que pode limitar a capacidade de armazenamento) e a necessidade de copiar todo o array se for necessário adicionar ou remover um elemento no meio da sequência.



Fonte: Pixabay

A declaração de um array em C é feita usando o seguinte formato:

```
tipo nome_do_array [tamanho];
```

Onde "tipo" é o tipo de dado dos elementos do array, "nome_do_array" é o nome escolhido para o array e "tamanho" é o número de elementos que serão armazenados no array.

Por exemplo, para declarar um array de números inteiros com 5 elementos, usamos:

```
int numeros[5];
```

O acesso aos elementos individuais do array é feito usando o índice do elemento entre colchetes:

```
números[0] = 10; // atribui o valor 10 ao primeiro elemento do array
int x = números[2]; // lê o valor do terceiro elemento do array e atribui a x
```

Os arrays são utilizados quando você precisa armazenar e acessar múltiplos valores do mesmo tipo em uma única variável. Eles são particularmente úteis quando você precisa lidar com uma grande quantidade de dados repetitivos, como uma lista de números ou nomes.

Existem muitas situações em que os arrays podem ser usados, como:

1. **Armazenar uma lista de elementos:** Os arrays podem ser usados para armazenar uma lista de elementos de um mesmo tipo, como uma lista de números ou nomes.
2. **Acesso rápido aos elementos:** Uma vez que os elementos são armazenados em um array, é fácil acessá-los por meio de seus índices.
3. **Manipulação de dados:** Os arrays permitem a manipulação eficiente dos dados armazenados, como ordenação, pesquisa e filtragem.
4. **Passagem de parâmetros:** Os arrays também são usados para passar múltiplos valores como parâmetros para uma função ou método.

Portanto, os arrays são uma ferramenta importante na programação e são amplamente utilizados em muitas linguagens de programação.

2.1. Criação e manipulação em código com a linguagem C

Em C, a criação e manipulação de arrays é bastante simples. Para criar um array, você precisa especificar o tipo de dados que ele armazenará e o número de elementos que terá.

Por exemplo, para criar um array de inteiros com 5 elementos, você pode fazer o seguinte:

```
int meuArray[5];
```

Você também pode inicializar os valores do array no momento da criação:

```
int meuArray[5] = {1, 2, 3, 4, 5};
```

Para acessar os elementos do array, você usa seu índice, que começa em zero para o primeiro elemento e aumenta em 1 para cada elemento subsequente.

Por exemplo, para acessar o terceiro elemento do array "meuArray", você faria o seguinte:

```
int terceiroElemento = meuArray[2];
```

Você também pode alterar os valores de um elemento do array fazendo uma atribuição direta:

```
meuArray[0] = 10;
```

Para percorrer todos os elementos de um array, você pode usar um loop for:

```
for (int i = 0; i < 5; i++) {
  printf("%d ", meuArray[i]);
}
```

Isso imprimirá todos os elementos do array no console. É importante lembrar que, ao acessar elementos do array, você deve garantir que está dentro dos limites do array, caso contrário, poderá ocorrer um erro de acesso inválido à memória.

2.2. Vantagens e desvantagens dos arrays

Arrays são estruturas de dados que permitem armazenar e manipular um conjunto de elementos em uma única variável. Abaixo, seguem algumas vantagens e desvantagens dos arrays:

Vantagens

- Fornecem acesso rápido aos elementos, pois os elementos são indexados e podem ser acessados diretamente;
- São úteis para armazenar grandes quantidades de dados do mesmo tipo;
- Permitem fazer operações em lote em todos os elementos do array, o que pode tornar o processamento mais eficiente;
- São fáceis de usar em loops, permitindo iterar sobre todos os elementos do array.

Desvantagens

- O tamanho do array é fixo e não pode ser alterado após a sua criação. Isso significa que, se você precisar adicionar ou remover elementos do array, precisará criar um novo array e copiar os elementos para ele;
- Os índices do array começam em zero, o que pode ser confuso para alguns usuários;
- Arrays ocupam espaço na memória, então se você tiver muitos arrays grandes, pode ter problemas de desempenho devido ao uso excessivo de memória;
- Se o array for mal gerenciado, pode ocorrer um estouro de buffer, o que pode causar crashes ou vulnerabilidades de segurança em seu código.

Segue abaixo um exemplo simples de programa em C que utiliza arrays para calcular a média de notas de um aluno:

```c
#include <stdio.h>

int main() {
    // Declaração do array com 5 elementos do tipo float
    float notas[5];

    // Preenchimento do array com as notas informadas pelo usuário
    printf("Digite as 5 notas do aluno:\n");
    for (int i = 0; i < 5; i++) {
        scanf("%f", &notas[i]);
    }

    // Cálculo da média das notas
    float media = 0.0;
    for (int i = 0; i < 5; i++) {
        media += notas[i];
    }
    media /= 5;

    // Impressão da média das notas
    printf("A média das notas é: %.2f\n", media);

    return 0;
}
```

Neste programa, é declarado um **array de notas** com capacidade para armazenar 5 notas do aluno. Em seguida, o programa solicita ao usuário que digite as 5 notas, e as notas são armazenadas no **array** através de um loop for.

Depois disso, é realizado o cálculo da média das notas, somando todos os valores do **array** e dividindo pela quantidade de elementos. Por fim, a média é impressa na tela utilizando a função **printf.**

Exemplos de implementação de arrays

1. **Declaração e inicialização de um array de inteiros com 5 elementos:**

```
int numeros[5] = {10, 20, 30, 40, 50};
```

2. **Atribuição de um valor a um elemento específico do array:**

```
numeros[2] = 35;
```

3. **Utilização de um loop para percorrer todos os elementos do array e imprimi-los na tela:**

```
for(int i = 0; i < 5; i++) {
    printf("%d ", numeros[i]);
}
}
```

4. **Declaração e inicialização de um array de caracteres (string):**

```
char nome[20] = "João da Silva";
```

5. **Utilização da função strlen() para obter o tamanho da string armazenada no array:**

```
int tamanho = strlen(nome);
printf("O nome tem %d caracteres\n", tamanho);
```

6. **Declaração e inicialização de uma matriz (array multidimensional) com 3 linhas e 4 colunas:**

```
int matriz[3][4] = {
    {1, 2, 3, 4},
    {5, 6, 7, 8},
    {9, 10, 11, 12}
};
```

7. **Acesso a um elemento específico da matriz:**

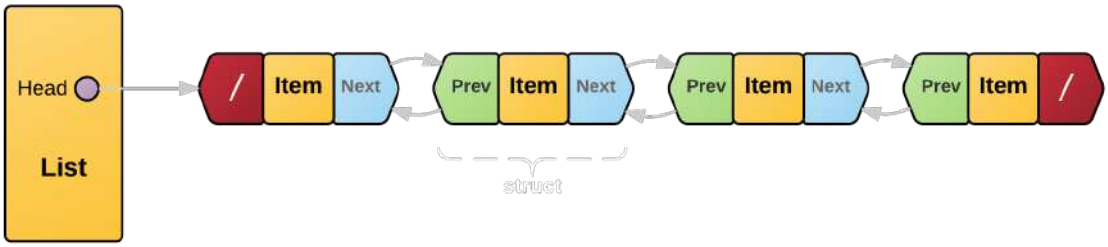
```
int valor = matriz[1][2]; // valor = 7
```

8. **Utilização de dois loops aninhados para percorrer todos os elementos da matriz e imprimi-los na tela:**

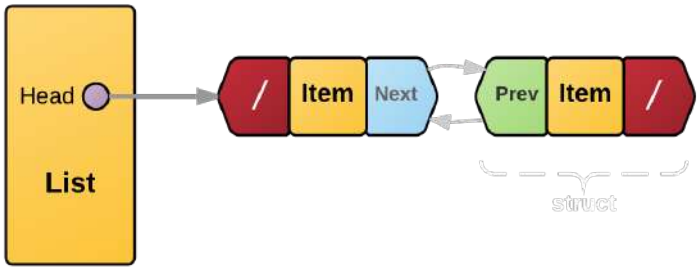
```
for(int i = 0; i < 3; i++) {
    for(int j = 0; j < 4; j++) {
        printf("%d ", matriz[i][j]);
    }
    printf("\n");
}
```

3. Listas Encadeadas

As listas encadeadas são uma estrutura de dados na qual cada elemento (ou nó) contém um valor e um ponteiro para o próximo elemento na lista. A seguir, apresento algumas informações importantes sobre as listas encadeadas:



As listas encadeadas podem ser simplesmente encadeadas (cada nó tem apenas um ponteiro para o próximo elemento) ou duplamente encadeadas (cada nó tem dois ponteiros, um para o próximo e outro para o elemento anterior).



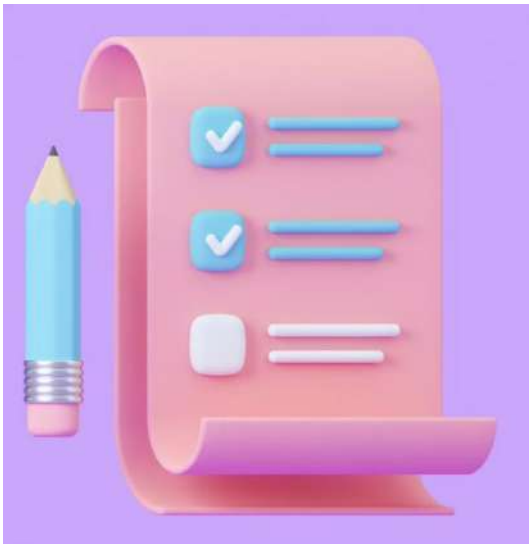
Uma das principais vantagens das listas encadeadas é que elas permitem a inserção e a remoção de elementos em qualquer posição da lista com eficiência O(1) (tempo constante), desde que se tenha acesso ao nó anterior (no caso de inserção) ou ao nó a ser removido.

No entanto, as operações de busca (para encontrar um elemento específico na lista) têm eficiência O(n) (tempo proporcional ao tamanho da lista). Por isso, as listas encadeadas são mais adequadas para cenários em que a inserção e a remoção de elementos são mais frequentes do que a busca.

Algumas outras operações comuns em listas encadeadas incluem a criação de uma nova lista vazia, a adição de elementos ao final da lista e a impressão dos elementos na lista.

3.1. O que são listas encadeadas?

Listas encadeadas são estruturas de dados que consistem em uma sequência de elementos chamados "nós", onde cada nó contém um valor e um ponteiro que aponta para o próximo elemento da lista. Essa estrutura é diferente das listas convencionais, pois os elementos não precisam estar armazenados em posições adjacentes na memória.



Fonte: Microsoft Office

Essa estrutura de dados permite inserções e remoções de elementos de forma mais eficiente do que as listas convencionais, pois não é necessário realocar todos os elementos após cada operação. Além disso, as listas encadeadas podem ser usadas para implementar outras estruturas de dados, como pilhas, filas e árvores binárias.

3.2. Criação e manipulação em código com a linguagem C

As listas encadeadas são uma estrutura de dados dinâmica que permite armazenar um conjunto de elementos em sequência. Cada elemento da lista, chamado de nó, contém um valor e um ponteiro que indica o próximo elemento da lista.

Abaixo apresento um exemplo de implementação de uma lista encadeada em C:

```c
#include <stdio.h>
#include <stdlib.h>

// Definição da estrutura do nó da lista
typedef struct node {
    int value;
    struct node *next;
} Node;

// Função para inserir um novo nó no início da lista
void insert(Node **head, int value) {
    // Aloca memória para o novo nó
    Node *new_node = (Node*) malloc(sizeof(Node));
    new_node->value = value;

    // O novo nó aponta para a cabeça da lista
    new_node->next = *head;

    // O novo nó se torna a nova cabeça da lista
    *head = new_node;
}
```

```c
// Função para imprimir os valores da lista
void print_list(Node *head) {
    printf("Lista: ");
    while (head != NULL) {
        printf("%d ", head->value);
        head = head->next;
    }
    printf("\n");
}

int main() {
    Node *head = NULL; // Inicializa a lista vazia

    // Insere alguns nós na lista
    insert(&head, 3);
    insert(&head, 5);
    insert(&head, 2);

    // Imprime a lista
    print_list(head);

    return 0;
}
```

Comentários do código

Neste exemplo, a função **insert** insere um novo nó no início da lista, enquanto a função **print_list** percorre a lista e imprime os valores dos nós. Para inserir um novo nó em qualquer posição da lista, é necessário alterar os ponteiros do nó anterior e do nó posterior ao novo nó.

É importante lembrar que as listas encadeadas não possuem um tamanho fixo e podem ser facilmente modificadas durante a execução do programa, o que as torna uma estrutura de dados bastante flexível.

No entanto, a manipulação de listas encadeadas pode ser mais complexa do que outras estruturas de dados, como **arrays**, por exemplo, devido à necessidade de gerenciar os ponteiros entre os nós.

3.3. Vantagens e desvantagens das listas encadeadas

As listas encadeadas são uma estrutura de dados comumente usada em programação. Algumas das vantagens e desvantagens dessas listas são:

Vantagens

1. **Flexibilidade:** as listas encadeadas permitem inserção e remoção de elementos sem a necessidade de alterar a posição dos outros elementos da lista, tornando-as flexíveis.
2. **Uso eficiente da memória:** essa estrutura de dados usa apenas o espaço de memória necessário para armazenar os elementos, evitando o desperdício de memória.
3. **Rapidez na inserção e remoção:** inserir ou remover um elemento no início de uma lista encadeada é muito rápido, pois não há a necessidade de realocar todos os outros elementos.

Desvantagens

1. **Acesso lento:** encontrar um elemento específico em uma lista encadeada pode ser demorado, pois a busca precisa ser feita sequencialmente a partir do início da lista.
2. **Espaço adicional para o ponteiro:** cada nó em uma lista encadeada contém um ponteiro para o próximo nó, o que requer mais espaço de memória do que uma matriz ou vetor.
3. **Dificuldades com ordenação:** ordenar uma lista encadeada pode ser uma tarefa difícil, pois é necessário reorganizar os ponteiros entre os nós da lista.

Exemplo em C que utiliza uma lista encadeada para armazenar informações de funcionários, incluindo nome e salário.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Definição da estrutura do nó da lista encadeada
struct Funcionario {
    char nome[50];
    float salario;
    struct Funcionario *proximo;
};
```

Nesse programa, criamos uma estrutura Funcionario que contém o nome e salário do funcionário, além de um ponteiro para o próximo nó da lista encadeada.

A função inserirFuncionario é responsável por criar um novo nó na lista com os dados fornecidos e atualizar a cabeça da lista.

Já a função imprimirFuncionarios percorre a lista encadeada a partir da cabeça e imprime os dados de cada funcionário.

No main, criamos a cabeça da lista e adicionamos três funcionários. Em seguida, imprimimos todos eles usando a função imprimirFuncionarios.

```c
// Função para imprimir todos os funcionários na lista encadeada
void imprimirFuncionarios(struct Funcionario *cabeca) {
    struct Funcionario *atual = cabeca;

    while (atual != NULL) {
        printf("Nome: %s | Salario: %.2f\n", atual->nome, atual->salari
        atual = atual->proximo;
    }
}

int main() {
    // Cria a cabeça da lista encadeada
    struct Funcionario *cabeca = NULL;

    // Insere alguns funcionários na lista
    inserirFuncionario(&cabeca, "Joao", 1500.00);
    inserirFuncionario(&cabeca, "Maria", 2000.00);
    inserirFuncionario(&cabeca, "Pedro", 1800.00);

    // Imprime todos os funcionários da lista
    imprimirFuncionarios(cabeca);

    return 0;
}
```

```c
// Função para inserir um novo funcionário na lista encadeada
void inserirFuncionario(struct Funcionario **cabeca, char nome[], float
    // Aloca memória para o novo nó
    struct Funcionario *novo = (struct Funcionario*)malloc(sizeof(struc

    // Copia o nome e salário para o novo nó
    strcpy(novo->nome, nome);
    novo->salario = salario;

    // Define o próximo nó como o atual primeiro da lista
    novo->proximo = *cabeca;

    // Atualiza a cabeça da lista para o novo nó
    *cabeca = novo;
}
```

4. **Pilhas**

Uma pilha é uma estrutura de dados que organiza e armazena um conjunto de elementos de forma ordenada, seguindo o princípio LIFO (Last In, First Out). Isso significa que o último elemento a ser adicionado na pilha será o primeiro a ser removido.

A pilha é um tipo de estrutura de dados linear, ou seja, seus elementos são organizados em uma linha, com cada elemento conectado apenas ao seu vizinho imediato.

A diferença das pilhas em relação às outras estruturas lineares, como as listas e filas, é que na pilha os elementos são adicionados e removidos apenas no topo da estrutura.

Em geral, as operações básicas em pilhas incluem "push", para adicionar um elemento ao topo da pilha, e "pop", para remover o elemento do topo da pilha.

Há também outras operações úteis para manipulação de pilhas, como "peek", que permite visualizar o elemento no topo da pilha sem removê-lo, e "isEmpty", que verifica se a pilha está vazia.

As pilhas são bastante utilizadas em algoritmos de processamento de expressões, como na avaliação de expressões matemáticas e lógicas, por exemplo.

Também são amplamente empregadas na implementação de programas de computador, onde podem ser usadas para gerenciamento de chamadas de funções, controle de histórico de navegação em navegadores web, entre outras aplicações.

É importante destacar que, assim como outras estruturas de dados, as pilhas têm suas vantagens e desvantagens.

Entre as principais vantagens estão a simplicidade de implementação e a eficiência em algumas operações, como inserção e remoção de elementos.

Já entre as desvantagens, está a limitação de acesso aos elementos internos da pilha, já que só é possível acessar o elemento no topo da pilha.

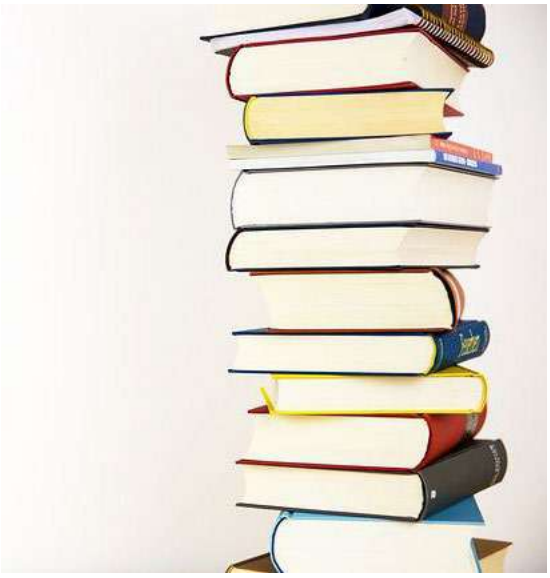


Imagem de Hermann Traub por Pixabay

4.2 Criação e manipulação em código com a linguagem C

Para criar e manipular uma pilha em C, é possível utilizar um array e um ponteiro que indica o topo da pilha. O array armazena os elementos da pilha e o ponteiro aponta para o elemento mais recentemente adicionado.

Para implementar a pilha em C, podemos definir uma struct com dois campos: o array de elementos da pilha e um inteiro que representa a posição do topo da pilha. Por exemplo:

```
struct Stack {
    int items[MAX_SIZE];
    int top;
};
```

Aqui, MAX_SIZE é uma constante que define o tamanho máximo da pilha. É possível também usar um typedef para renomear a struct para um nome mais simples de ser usado, por exemplo:

```
typedef struct Stack stack;
```

A primeira operação que precisamos implementar é a inicialização da pilha. Isso pode ser feito criando uma função que zera o valor do campo top. Por exemplo:

```
void init(stack *s) {
    s->top = -1;
}
```

Outra operação útil é verificar se a pilha está vazia. Para isso, basta verificar se o campo top é igual a -1 (que indica que a pilha não tem nenhum elemento):

```
int isEmpty(stack *s) {
    return s->top == -1;
}
```

Também é possível verificar se a pilha está cheia, comparando o campo top com o valor MAX_SIZE-1 (que indica que a pilha está completamente preenchida):

```
int isFull(stack *s) {
    return s->top == MAX_SIZE-1;
}
```

Para adicionar um elemento à pilh é necessário incrementar o valor do campo top e atribuir o novo elemento à posição correspondente no array.

```
void push(stack *s, int item) {
    if (isFull(s)) {
        printf("Erro: Pilha cheia.\n");
    } else {
        s->top++;
        s->items[s->top] = item;
    }
}
```

Para remover um elemento da pilha é necessário retornar o elemento no topo da pilha (que corresponde ao valor na posição top do array), decrementar o valor do campo top e fazer com que o ponteiro aponte para a posição do novo elemento no topo da pilha.

```
int pop(stack *s) {
    if (isEmpty(s)) {
        printf("Erro: Pilha vazia.\n");
        return -1;
    } else {
        int removedItem = s->items[s->top];
        s->top--;
        return removedItem;
    }
}
```

Por fim, é possível obter o elemento no topo da pilha sem removê-lo usando a função peek. Esta função retorna o valor na posição top do array.

```
int peek(stack *s) {
    if (isEmpty(s)) {
        printf("Erro: Pilha vazia.\n");
        return -1;
    } else {
        return s->items[s->top];
    }
}
```

No main(), podemos criar uma pilha, inicializá-la, adicionar elementos, remover elementos e exibir o elemento no topo da pilha.

É importante destacar que as variáveis passadas como argumento nas chamadas das funções são ponteiros, para que as alterações feitas dentro das funções afetem diretamente a pilha original.

4.2. Vantagens e desvantagens de uma pilha

As pilhas possuem diversas vantagens e desvantagens como estrutura de dados. Abaixo seguem as principais:

Vantagens

- **Simplicidade de implementação:** a estrutura de pilha é relativamente simples de ser implementada, o que a torna uma opção viável em muitas situações;
- **Eficiência em operações específicas:** a inserção e remoção de elementos em uma pilha são operações muito eficientes. Elas têm tempo de execução constante (O(1)), independentemente do tamanho da pilha;
- **Facilita a resolução de determinados problemas:** as pilhas podem ser utilizadas para resolver diversos problemas, tais como avaliação de expressões matemáticas, histórico de navegação em um navegador web e gerenciamento de chamadas de funções em programas.

Desvantagens

- **Acesso limitado aos elementos:** uma das principais desvantagens das pilhas é que só é possível acessar o elemento no topo da pilha. Isso limita bastante o uso das pilhas em certas aplicações;
- **Espaço de armazenamento:** a implementação de uma pilha pode requerer mais espaço de armazenamento do que outras estruturas de dados, especialmente se os elementos da pilha forem grandes;
- **Não suporta acesso aleatório:** ao contrário de outras estruturas de dados, as pilhas não suportam o acesso aleatório aos seus elementos. Se você precisa acessar um elemento específico na pilha, é necessário remover todos os elementos acima dele primeiro.

É importante lembrar que a escolha da estrutura de dados adequada deve ser feita com base nas características do problema que se deseja resolver. Em alguns casos, as pilhas podem ser a melhor opção, enquanto em outros casos outras estruturas de dados, como as listas, as filas ou as árvores, podem ser mais apropriadas.

4.3. Exercícios

Escreva um programa em C que implemente uma pilha e permita realizar as seguintes operações:

- Inserir um elemento na pilha
- Remover um elemento da pilha
- Verificar o elemento no topo da pilha
- Verificar se a pilha está vazia

```c
#include <stdio.h>
#include <stdlib.h>

#define MAX 10

int stack[MAX], top = -1;

void push(int element) {
    if(top == MAX-1) {
        printf("Stack Overflow\n");
        return;
    }
    stack[++top] = element;
    printf("%d pushed to stack\n", element);
}

void pop() {
    if(top == -1) {
        printf("Stack Underflow\n");
        return;
    }
    printf("%d popped from stack\n", stack[top--]);
}
```

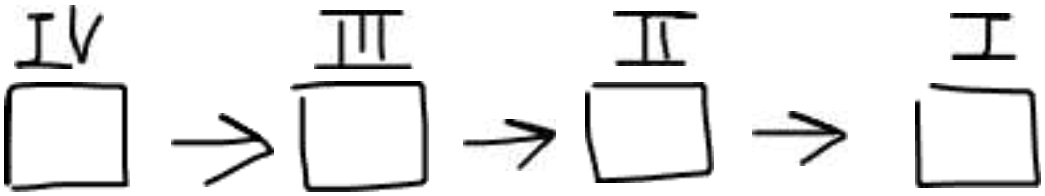
```c
int peek() {
    if(top == -1) {
        printf("Stack is empty\n");
        return -1;
    }
    return stack[top];
}

int isEmpty() {
    return top == -1;
}

int main() {
    push(1);
    push(2);
    push(3);
    printf("Top element is %d\n", peek());
    pop();
    printf("Top element is %d\n", peek());
    return 0;
}
```

5. Filas

As filas são uma estrutura de dados linear que seguem o princípio FIFO (First-In-First-Out), ou "o primeiro a entrar é o primeiro a sair". Isso significa que os elementos que são inseridos na fila são organizados em uma ordem sequencial, e sempre o elemento que estiver na frente da fila será o próximo a ser removido.

A fila é um tipo de coleção de objetos que pode ter operações de enfileiramento (inserção) e desenfileiramento (remoção).

Quando um elemento é adicionado à fila, ele vai para o final da fila. E quando um elemento é removido da fila, ele é retirado do início da fila. A fila pode ter um número limitado de elementos ou pode ser infinita.



(Fonte: Criação própria)

As filas são amplamente utilizadas em sistemas computacionais, como em algoritmos de busca em largura, gerenciamento de tarefas em sistemas operacionais, escalonamento de processos em servidores, controle de acesso ao recurso compartilhado em redes, entre outras aplicações.

Fonte: Pixabay

As operações fundamentais em uma fila são a inserção de elementos na parte de trás ("enqueue") e a remoção de elementos na parte da frente ("dequeue"). Além dessas operações, existem outras operações que podem ser realizadas em filas, como por exemplo:

- **Verificar se a fila está vazia:** essa verificação é importante antes de realizar a operação de desenfileiramento.
- **Verificar se a fila está cheia:** essa verificação é importante para controlar o número de elementos na fila quando a fila tem um tamanho fixo.
- **Obter o tamanho atual da fila:** essa operação retorna o número de elementos na fila.
- **Obter o elemento na frente da fila sem removê-lo:** essa operação permite que se obtenha o elemento que está na frente da fila sem retirá-lo. Essa operação é útil quando se deseja apenas verificar qual é o próximo elemento a ser removido.

As filas podem ser implementadas de diversas formas, como por exemplo usando um vetor, uma lista encadeada ou uma pilha. A escolha da implementação mais adequada para cada caso específico depende das características do problema que se deseja resolver e dos recursos disponíveis no sistema computacional em questão.

5.1. Criação e manipulação em código com a linguagem C

Uma fila é uma estrutura de dados linear que segue o princípio FIFO (First In, First Out), ou seja, o primeiro elemento inserido na fila é o primeiro a ser removido. Em C, podemos criar e manipular filas utilizando ponteiros para nós.

Abaixo, vou explicar em detalhes como criar e manipular uma fila utilizando a linguagem C:

Criando uma fila

Para criar uma fila precisamos de uma estrutura que define um nó da fila com os seguintes campos:

```
struct Node {
    int data;
    struct Node* next;
};
```

O campo data armazena o dado do nó, enquanto o campo next é um ponteiro para o próximo nó da fila.

Em seguida, criamos uma estrutura para representar a fila com dois ponteiros: um para o início da fila (chamado de frente ou front) e outro para o final da fila (chamado de trás ou rear):

```
struct Queue {
    struct Node *front, *rear;
};
```

Agora, podemos criar uma fila vazia chamando a função initializeQueue, que inicializa os ponteiros front e rear com NULL:

```
void initializeQueue(struct Queue* q) {
    q->front = q->rear = NULL;
}
```

Inserindo elementos na fila

Para adicionar um novo elemento à fila, utilizamos a função enqueue. Essa função recebe como parâmetro um ponteiro para a fila (q) e o valor a ser inserido (value).

A função cria um novo nó com o valor passado, atualiza o ponteiro next do último nó da fila para apontar para o novo nó, e atualiza o ponteiro rear da fila para apontar para o novo nó:

```
void enqueue(struct Queue* q, int value) {
    struct Node* newNode = (struct Node*) malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = NULL;

    if (isEmpty(q)) { // Se a fila estiver vazia, o novo nó será o prim
        q->front = q->rear = newNode;
        return;
    }

    q->rear->next = newNode; // Atualiza o ponteiro next do último nó c
    q->rear = newNode; // Atualiza o ponteiro rear da fila para apontar
}
```

Removendo elementos da fila

Para remover um elemento da fila, utilizamos a função dequeue. Essa função recebe como parâmetro um ponteiro para a fila (q) e retorna o valor do primeiro nó da fila.

```
int isEmpty(struct Queue* q) {
    return (q->front == NULL);
}
```

A função remove o primeiro nó da fila, atualiza o ponteiro front para apontar para o próximo nó da fila e libera a memória alocada para o nó removido:

Verificando se a fila está vazia

Para verificar se a fila está vazia, podemos utilizar a função isEmpty, que retorna 1 se a fila estiver vazia e 0 caso contrário:

```
int dequeue(struct Queue* q) {
    if (isEmpty(q)) { // Se a fila estiver vazia, retorna erro
        printf("A fila está vazia!\n");
        return -1;
    }

    struct Node* temp = q->front; // Armazena o endereço do primeiro nó
    int value = temp->data; // Armazena o valor do primeiro nó

    q->front = q->front->next; // Atualiza o ponteiro front da fila par

    if (q->front == NULL) { // Se a fila ficar vazia, atualiza também o
        q->rear = NULL;
    }

    free(temp); // Libera a memória alocada para o primeiro nó

    return value; // Retorna o valor do primeiro nó
}
```

5.2. Vantagens e desvantagens de uma fila

As filas, como qualquer outra estrutura de dados, possuem suas vantagens e desvantagens. Abaixo, listo as principais:

Vantagens

- As filas são muito eficientes em lidar com elementos que precisam ser processados na ordem em que foram adicionados.
- As operações básicas de enfileiramento e desenfileiramento têm complexidade O(1), ou seja, são muito rápidas em comparação com outras estruturas de dados, como árvores binárias de busca. As filas são muito fáceis de implementar e entender.

Desvantagens:

- Acesso aleatório aos elementos da fila não é possível. É necessário remover todos os elementos anteriores para acessar um elemento específico.
- As filas podem ficar cheias, o que pode levar a problemas de estouro de memória. Para evitar isso, é necessário limitar o tamanho máximo da fila ou implementar uma fila dinâmica que possa crescer ou diminuir conforme necessário.
- As operações de inserção e remoção são relativamente lentas em filas implementadas usando vetores, já que é necessário mover todos os elementos após a posição do elemento removido ou inserido. Isso pode ser evitado ao se usar uma lista encadeada para implementar a fila.
- As filas não são muito eficientes em termos de uso de memória, já que cada nó na fila requer espaço adicional para manter a referência ao próximo nó.

Apesar das desvantagens, as filas são uma estrutura de dados extremamente útil e versátil, sendo amplamente utilizadas em diversos tipos de algoritmos e aplicações computacionais.

6.0. Árvores

As árvores são uma estrutura de dados fundamental em ciência da computação e engenharia de software. Elas são frequentemente usadas para armazenar e organizar informações hierárquicas, como a estrutura de um arquivo no sistema de arquivos de um computador ou a organização de elementos em uma página web.

Uma árvore é composta por um conjunto de nós conectados por arestas. Cada nó pode ter zero ou mais nós filhos, exceto o nó raiz, que não tem nenhum pai. Os nós sem filhos são chamados de folhas. A altura da árvore é definida como o comprimento do caminho mais longo da raiz até uma folha.

Existem vários tipos de árvores, cada um com suas próprias características e aplicações. Algumas das árvores mais comuns incluem:

- **Árvore binária:** Uma árvore binária é uma árvore em que cada nó tem no máximo dois filhos. O filho esquerdo sempre representa um valor menor do que o nó pai, enquanto o filho direito representa um valor maior.

- **Árvore de busca binária:** Uma árvore de busca binária é uma árvore binária em que os valores estão organizados de forma que a esquerda seja menor e a direita seja maior. Isso permite que a árvore seja pesquisada de forma eficiente em tempo logarítmico.

- **Árvore balanceada:** Uma árvore balanceada é uma árvore em que a diferença de altura entre as subárvores esquerda e direita de qualquer nó é no máximo 1. Isso mantém o tempo de operações de inserção e exclusão em logarítmico.

- **Árvore Trie:** Uma árvore Trie é uma árvore em que cada nó representa um prefixo de uma palavra. Cada aresta representa um caractere e a raiz representa o prefixo vazio.

As árvores oferecem muitas vantagens em relação às estruturas de dados lineares, como listas e pilhas. Por exemplo, elas permitem a busca eficiente, a inserção e a exclusão em tempo logarítmico, dependendo da estrutura da árvore. Além disso, as árvores podem ser usadas para implementar algoritmos de classificação, como o algoritmo quicksort, tornando-os uma ferramenta poderosa para programadores.

No entanto, assim como todas as estruturas de dados, as árvores têm suas desvantagens. Uma das principais desvantagens é que a implementação incorreta pode levar a uma árvore não balanceada, o que pode aumentar o tempo de execução de operações críticas. Além disso, as árvores são mais complexas do que algumas outras estruturas de dados, o que pode dificultar o desenvolvimento e manutenção do código.

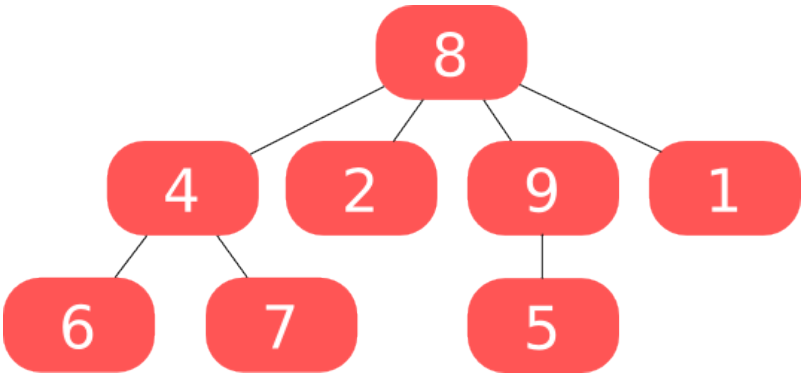
As árvores são uma estrutura de dados poderosa e versátil que é amplamente utilizada em ciência da computação e engenharia de software. Compreender os diferentes tipos de árvores e como eles são usados pode ajudar a criar código mais eficiente e robusto.

6.1. Tipos de árvores

6.1.1 Árvores Binárias

Uma árvore binária é uma estrutura de dados em que cada nó pode ter no máximo dois filhos. O filho esquerdo representa um valor menor do que o nó pai, enquanto o filho direito representa um valor maior. A árvore começa com um nó raiz, que não tem pai.

Cada nó contém uma chave e opcionalmente outros dados relacionados a essa chave. As chaves são usadas para comparar os nós durante a busca ou inserção na árvore. A forma como as chaves são organizadas na árvore afeta seu desempenho.



A altura da árvore binária é definida como o número máximo de arestas entre a raiz e qualquer folha na árvore. Uma árvore binária com n nós tem altura máxima n-1 e altura mínima log2(n+1).

A árvore binária é frequentemente usada para criar estruturas de dados mais complexas, como a árvore de busca binária, que é uma árvore binária ordenada. Em uma árvore de busca binária, cada nó deve conter uma chave única e a chave do nó à esquerda é sempre menor do que a chave do nó à

direita. Isso permite que a árvore seja pesquisada de forma eficiente em tempo logarítmico.

Uma operação comum em árvores binárias é a inserção de um novo nó. Para inserir um novo nó, você começa a partir da raiz e compara a chave do nó a ser inserido com a chave do nó atual. Se a chave for menor do que a do nó atual, você segue para o filho esquerdo. Caso contrário, você segue para o filho direito. Quando chegar a um nó que não tem filhos, você cria um novo nó com a chave desejada e adiciona-o como filho do nó atual.

Outra operação comum é a exclusão de um nó. A exclusão pode ser complexa porque deve manter a ordenação da árvore após a remoção. Se o nó a ser excluído não tiver filhos, ele será removido facilmente. Se tiver apenas um filho, esse filho substituirá o nó excluído. Se tiver dois filhos, deve-se encontrar o menor nó à direita ou o maior nó à esquerda e usá-lo para substituir o nó excluído.

Em resumo, a árvore binária é uma estrutura de dados simples e versátil que é amplamente utilizada em ciência da computação e engenharia de software. Compreender como as chaves são organizadas na árvore afeta seu desempenho e eficiência em operações críticas. A árvore binária pode ser usada para criar estruturas de dados mais complexas, como a árvore de busca binária.

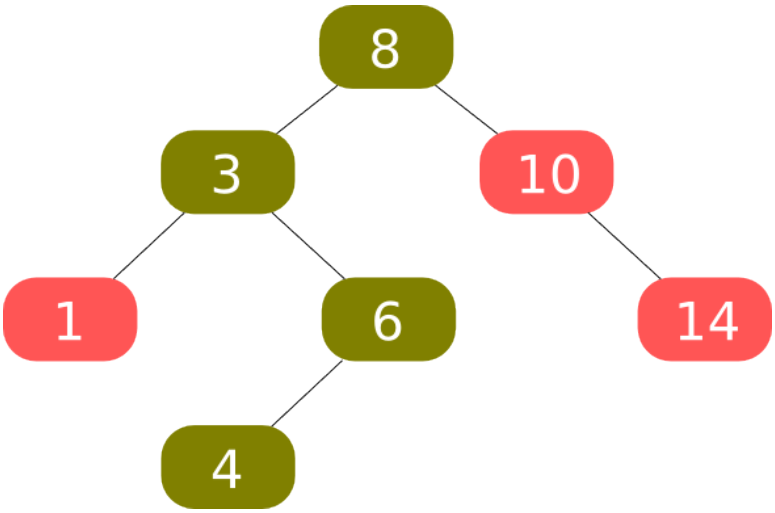
Classificação das Árvores Binárias

As árvores binárias podem ser classificadas de diferentes maneiras, dependendo das características que são levadas em consideração. Algumas das classificações comuns incluem:

Árvores Binárias Simples vs. Árvores Binárias de Busca

As árvores binárias simples são aquelas em que não há nenhuma restrição quanto à ordem dos valores armazenados nos nós. Já as árvores binárias de busca são aquelas em que cada nó tem um valor associado e a ordem desses valores segue uma regra específica (por exemplo, todos os nós à

esquerda do nó pai têm valores menores que o valor do nó pai, enquanto todos os nós à direita têm valores maiores).



Árvores binárias simples e árvores binárias de busca são duas estruturas de dados muito utilizadas em ciência da computação.

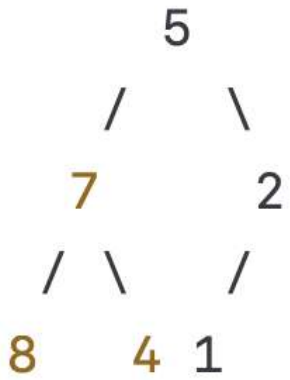
Uma árvore binária simples é uma estrutura de dados em que cada nó pode ter no máximo dois filhos. Geralmente, as árvores binárias são usadas para representar estruturas hierárquicas, como por exemplo em algoritmos de pesquisa e ordenação.

Já a árvore binária de busca é uma árvore binária especial em que cada nó tem um valor associado e a propriedade dos nós da subárvore esquerda possuírem valores menores ou iguais ao seu nó pai, enquanto os nós da subárvore direita possuem valores maiores que o seu nó pai. Dessa forma, essa estrutura de dados permite a realização de buscas eficientes, pois é possível reduzir a área de busca pela metade em cada passo.

Embora ambas sejam árvores binárias, a principal diferença entre elas é que a árvore binária de busca possui uma organização especial nos seus nós, que permite uma busca mais eficiente, enquanto na árvore binária simples os nós não apresentam essa organização.

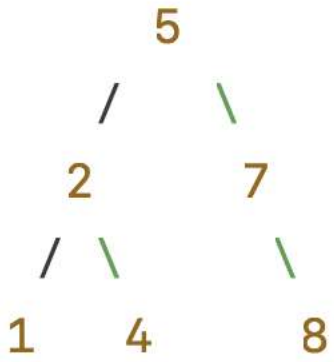
Exemplo de Árvores Binárias Simples vs. Árvores Binárias de Busca

- Vamos supor que temos os seguintes valores: 5, 7, 2, 8, 4 e 1.
- Uma possível representação em árvore binária simples seria a seguinte:



Nessa estrutura, cada nó possui no máximo dois filhos, mas não há uma organização especial entre eles. Essa árvore pode ser utilizada para representar uma expressão aritmética, por exemplo.

Já em uma árvore binária de busca, os nós são organizados de forma que cada nó da subárvore esquerda possui um valor menor ou igual ao seu nó pai, e cada nó da subárvore direita possui um valor maior do que o seu nó pai. A representação da mesma sequência de valores como uma árvore binária de busca seria a seguinte:

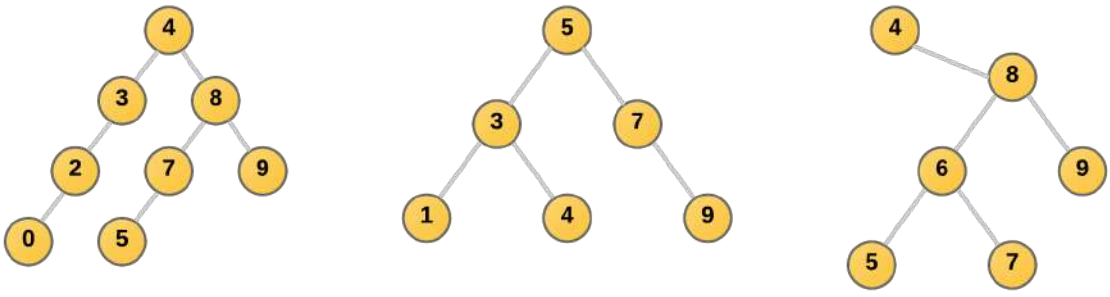


Essa estrutura permite que buscas sejam realizadas de forma eficiente, pois é possível reduzir a área de busca pela metade em cada passo. Por exemplo, se quisermos encontrar o valor 4 na árvore, podemos começar verificando o nó raiz, que possui o valor 5. Como o valor que buscamos é menor do que 5, podemos seguir pela subárvore esquerda, onde encontraremos o valor 2. Novamente, como o valor procurado é maior do que 2, seguimos pela subárvore direita, onde encontraremos o valor 4.

Árvores Binárias Balanceadas vs. Não-Balanceadas

As árvores binárias balanceadas são aquelas em que a altura das subárvores esquerda e direita de cada nó difere no máximo em uma unidade (por exemplo, nas árvores AVL e vermelho-preto). Já nas árvores binárias não-balanceadas, essa diferença pode ser maior, o que pode levar a piores desempenhos em certas operações.

Uma árvore binária é considerada balanceada se a diferença entre as alturas de suas subárvores esquerda e direita for, no máximo, 1. Por exemplo, uma árvore binária perfeitamente balanceada teria todas as folhas no mesmo nível, enquanto uma árvore binária desbalanceada pode ter um lado da árvore muito mais profundo do que o outro.



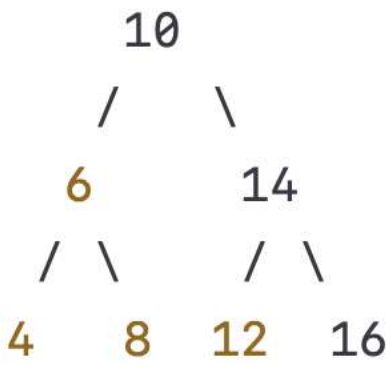
As árvores binárias balanceadas têm a vantagem de garantir que o tempo de busca, inserção e remoção dos nós seja O(log n), onde n é o número de elementos na árvore. Isso ocorre porque a altura da árvore é mantida baixa e,

portanto, o número de operações necessárias para encontrar, inserir ou remover um elemento é limitado.

Por outro lado, árvores binárias não-balanceadas podem ter alturas maiores e, como resultado, as operações de busca, inserção e remoção podem levar mais tempo para serem executadas. No entanto, elas podem ser mais fáceis de implementar e, em algumas situações, podem ser suficientemente eficientes para as operações necessárias.

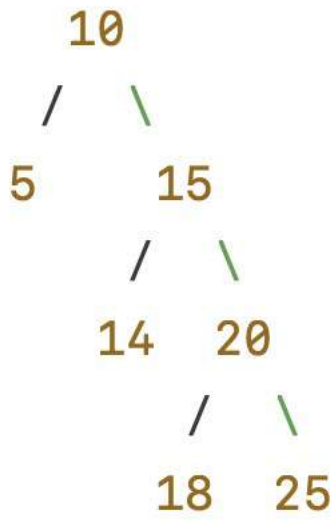
As árvores binárias balanceadas são preferíveis quando o desempenho é uma prioridade e espera-se que a árvore cresça significativamente em tamanho. As árvores binárias não-balanceadas podem ser adequadas para situações em que a simplicidade da implementação é uma prioridade e a árvore não deve crescer muito.

Árvores Binárias Balanceadas



Nesta árvore, a diferença entre a altura da subárvore esquerda (que tem altura 2) e a subárvore direita (que também tem altura 2) é no máximo 1. Portanto, esta árvore é considerada balanceada.

Árvore Binária Não-Balanceada:



Nesta árvore, a altura da subárvore esquerda é 1, enquanto a altura da subárvore direita é 3. A diferença entre as alturas das subárvores é maior que 1, portanto, esta árvore é considerada desbalanceada.

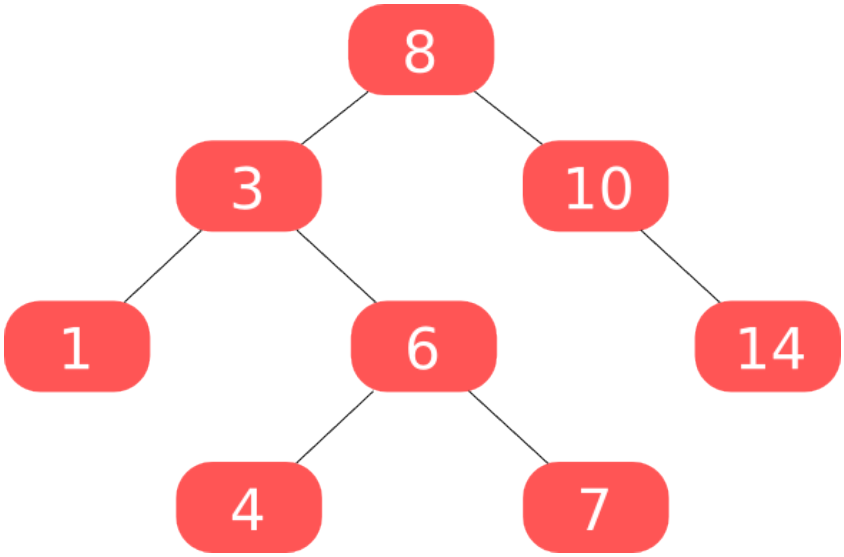
As operações de busca, inserção e remoção nesta árvore podem ser mais lentas do que em uma árvore balanceada, pois as operações precisam percorrer mais nós na subárvore mais profunda.

Árvores Binárias Completas vs. Incompletas

As árvores binárias completas são aquelas em que todos os níveis da árvore estão completamente preenchidos, exceto possivelmente o último nível, que pode estar incompleto. Já nas árvores binárias incompletas, alguns dos níveis podem estar totalmente vazios, o que pode resultar em uma estrutura de dados menos eficiente em termos de espaço.

Uma árvore binária completa é uma árvore binária em que todos os níveis estão preenchidos, exceto possivelmente o último, que é preenchido a partir da esquerda para a direita. Em outras palavras, todas as folhas estão no mesmo nível e cada nó pai tem exatamente dois filhos.

Já uma árvore binária incompleta é uma árvore em que alguns nós podem ter apenas um filho ou nenhum filho. Isso significa que as folhas não necessariamente estão no mesmo nível e não há garantia de que cada nó pai tenha exatamente dois filhos.



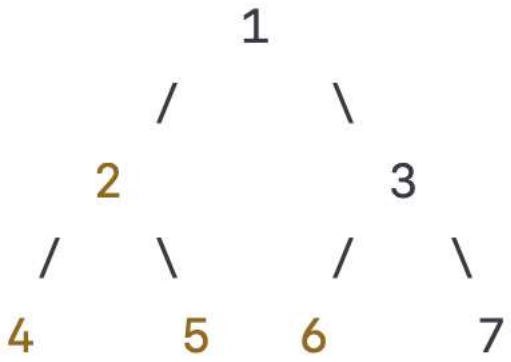
As árvores binárias completas têm algumas propriedades importantes que as tornam úteis em certos algoritmos e estruturas de dados. Por exemplo, a altura de uma árvore binária completa com N nós é log2(N), o que significa que as operações na árvore podem ser realizadas de forma mais eficiente do que em árvores incompletas, especialmente quando se trata de busca e inserção de elementos.

Além disso, a representação dos elementos em uma árvore binária completa pode ser feita de forma mais simples e compacta em comparação com árvores incompletas.

No entanto, nem sempre é necessário utilizar uma árvore binária completa, e muitas vezes uma árvore incompleta pode ser suficiente para atender às necessidades do problema em questão. O importante é escolher a estrutura de dados mais adequada para a aplicação específica.

Árvores Binárias Completas

Observe que todos os níveis da árvore estão preenchidos, exceto possivelmente o último, que está preenchido da esquerda para a direita. Cada nó pai tem exatamente dois filhos.



Árvores Binárias Completas Incompletas

Observe que o nó 2 tem apenas um filho e não há nenhum filho para a esquerda do nó 6. Essa árvore não é uma árvore binária completa, pois nem todas as folhas estão no mesmo nível e nem todos os nós pais têm exatamente dois filhos.

Árvores Binárias Cheias vs. Não-Cheias: As árvores binárias cheias são aquelas em que cada nó tem exatamente dois filhos ou nenhum filho. Já as árvores binárias não-cheias (ou parcialmente cheias) são aquelas em que alguns nós podem ter apenas um filho.

Essas são algumas das maneiras comuns de classificar as árvores binárias. Vale lembrar que essas classificações não são mutuamente exclusivas e uma mesma árvore pode ser classificada em mais de uma categoria (por exemplo, uma árvore AVL é tanto balanceada quanto uma árvore binária de busca).

Para criar e manipular uma árvore em C, podemos utilizar uma estrutura de dados denominada "nó". Um nó contém um valor e ponteiros para seus nós filhos. Segue abaixo um exemplo de código para criação e manipulação de uma árvore simples em C:

```c
#include <stdio.h>
#include <stdlib.h>

// Definição da estrutura do nó da árvore
struct no {
    int chave;
    struct no *esq;
    struct no *dir;
};

// Função para criação de um novo nó
struct no* novoNo(int chave) {
    struct no* no = (struct no*) malloc(sizeof(struct no));
    no->chave = chave;
    no->esq = NULL;
    no->dir = NULL;
    return no;
}
```

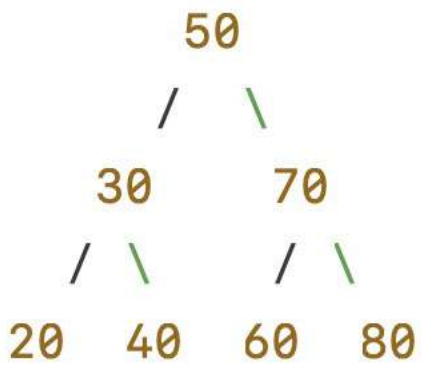
```c
// Função para inserção de um novo nó na árvore
struct no* inserir(struct no* raiz, int chave) {
    if (raiz == NULL) {
        return novoNo(chave);
    }
    if (chave < raiz->chave) {
        raiz->esq = inserir(raiz->esq, chave);
    } else {
        raiz->dir = inserir(raiz->dir, chave);
    }
    return raiz;
}

// Função para busca de um nó na árvore
struct no* buscar(struct no* raiz, int chave) {
    if (raiz == NULL || raiz->chave == chave) {
        return raiz;
    }
    if (chave < raiz->chave) {
        return buscar(raiz->esq, chave);
    } else {
        return buscar(raiz->dir, chave);
    }
}
```

```c
// Função para impressão em ordem da árvore
void imprimirEmOrdem(struct no* raiz) {
    if (raiz != NULL) {
        imprimirEmOrdem(raiz->esq);
        printf("%d ", raiz->chave);
        imprimirEmOrdem(raiz->dir);
    }
}

// Função principal
int main() {
    struct no* raiz = NULL;

    // Inserção dos nós na árvore
    raiz = inserir(raiz, 50);
    inserir(raiz, 30);
    inserir(raiz, 20);
    inserir(raiz, 40);
    inserir(raiz, 70);
    inserir(raiz, 60);
    inserir(raiz, 80);
```

```
    // Busca de um nó na árvore
    struct no* resultadoBusca = buscar(raiz, 60);
    if (resultadoBusca != NULL) {
        printf("Nó encontrado: %d\n", resultadoBusca->chave);
    } else {
        printf("Nó não encontrado!\n");
    }

    // Impressão em ordem dos nós da árvore
    imprimirEmOrdem(raiz);
    printf("\n");

    return 0;
}
```

Nesse exemplo, temos as funções novoNo, que cria um novo nó; inserir, que insere um novo nó na árvore de acordo com a sua chave; buscar, que busca um nó na árvore a partir de sua chave; e imprimirEmOrdem, que imprime os nós da árvore em ordem crescente.

A árvore criada possui os seguintes nós:



A saída do programa será:

```
Nó encontrado: 60
20 30 40 50 60 70 80
```

Referência:

TANENBAUM, Andrew S.; LANGSAM, Yedidyah; AUGENSTEIN, Moshe J. Estruturas de dados usando C. São Paulo: Pearson Education, 2007.

REZENDE, Solon Venâncio de; FERREIRA, Janaína Gonçalves. Estruturas de dados com implementações em C e Java. São Paulo: Thomson Learning, 2004.

ZIVIANI, Nivio. Projeto de algoritmos com implementações em Pascal e C. São Paulo: Thomson Learning, 1999.

GONNET, Gaston H.; BAEZA-YATES, Ricardo. Algoritmos e estruturas de dados. Rio de Janeiro: LTC, 2013.

SZWARCFITER, Jayme Luiz; MARKENZON, Lilian. Estrutura de dados e seus algoritmos. 3ª ed. Rio de Janeiro: LTC, 2010.

GUIMARÃES, Angelo de Moura; LAGES, Newton Alberto de Castilho. Estruturas de dados em C. Rio de Janeiro: LTC, 2002.

FORBELLONE, André Luiz Villar; EBERSPÄCHER, Henri Frederico. Lógica de programação: a construção de algoritmos e estruturas de dados. 3ª ed. Porto Alegre: Bookman, 2005.

PEREIRA, Silvio do Lago. Estruturas de dados fundamentais: conceitos e aplicações em C. São Paulo: Érica, 2005.

WIRTH, Niklaus. Algoritmos e estruturas de dados. Rio de Janeiro: LTC, 1989.

DEMO, Gerson da Penha; MACEDO, Mauro Lopes de. Estruturas de dados com técnicas de programação em C. São Paulo: Makron Books, 1996.

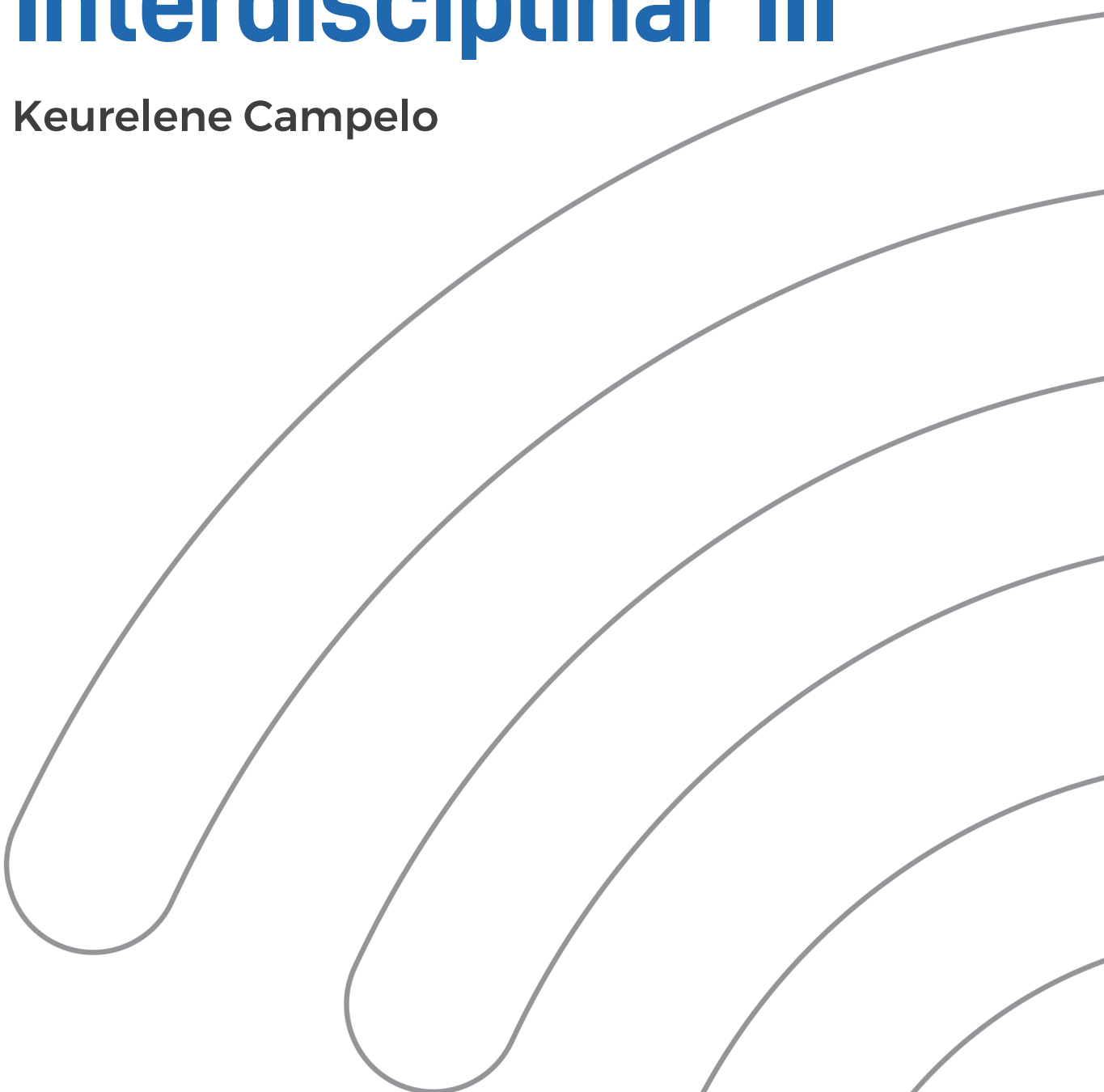
CURSO TÉCNICO EM

INFORMÁTICA

MÓDULO III

Projeto de Aprendizagem Interdisciplinar III

Keurelene Campelo



Sumário

1. EDUCAÇÃO E TECNOLOGIA

1.1 Autonomia na educação à distância

Historicamente, a sociedade sempre buscou – por questões de sobrevivência – a propagação de sua espécie, dando a ela qualidade de vida e construindo mecanismos que pudessem atender suas necessidades. Temendo sua extinção, atividades coletivas, desenvolvimento da agricultura, formação de agrupamentos sociais, faziam parte dos métodos e técnicas para a continuidade daqueles grupos na terra.

Inicialmente agrícola, a sociedade buscava aprimorar seus instrumentos primários para o plantio, colheita e confecção de manufaturas. Já na sociedade industrial, o foco era elevar a produção de objetos industrializados, utilizando um saber tecnicista para uma melhor produção. Apesar do salto histórico entre a Revolução Agrícola e a Revolução Industrial, vemos aqui o uso da técnica e da tecnologia como ferramentas para a formação de uma sociedade, formando também culturas e saberes.

Disponível em: www.pixabay.com/Acesso em 22 abr. 2023

Atualmente, a rapidez da globalização exige de si uma sociedade onde o conhecimento pede uma atualização quase imediata, tendo o saber como elemento essencial para a sobrevivência dos novos tempos. Em tempos de rede, vida social, profissional e educacional se ampliam, se misturam e exigem de nós habilidades e conhecimentos em tempos mais curtos e mais específicos, mantendo sempre a qualidade desse processo de conhecimento.

Desse modo, a aprendizagem à distância surge aqui como uma possibilidade de ampliar nossos saberes, possibilitando um novo olhar para a educação, aqui agora ajustada com as exigências da globalização, do conhecimento, do trabalho, personalizando estudos e formando uma sociedade cada vem mais atenta a si.

A Educação à Distância é um modelo educacional que utiliza os mecanismos tecnológicos disponíveis a sua época para alcançar um determinado espaço, um determinado grupo. Amadurecida, hoje a EaD já é nacionalmente reconhecida como uma modalidade de aprendizagem e está inserida, formalmente, no contexto educacional, apresentando uma merecida expansão no cenário educacional mundial.

Tal fato pode ser compreendido ao analisar as novas demandas políticas e sociais, visto que a necessidade e a exigência do aperfeiçoamento profissional no mercado de trabalho pedem uma continuidade do processo de formação educacional.

No campo tecnológico, é importante ressaltar que as inovações da tecnologia possibilitam novas situações de aprendizagem. Na esfera pedagógica, observa-se a educação à distância como uma modalidade flexível e que corresponde como um modelo de autoformação, algo alternativo e necessário no atual contexto, consolidando uma autonomia ao indivíduo durante o seu processo de aprendizagem.

PARA IR ALÉM:

Black Mirror: (Direção: Charles Brooker, Netflix, 2011, 5 temporadas): a série possui cinco temporadas e em cada episódio o público conhece uma história diferente que aborda alguns dos vários malefícios do uso excessivo da tecnologia. A série de ficção científica e thriller psicológico envolve o telespectador e apresenta reviravoltas e reflexões, tais como as questões abaixo:

Se você pudesse, bloquearia uma pessoa na vida real?

Disponível em: www.pixabay.com/Acesso em 22 abr. 2023

A educação tem buscado acompanhar as transformações sociais ocorridas ao longo dos anos, sem perder sua função social, possibilitando assim o pleno desenvolvimento humano. Logo, a modalidade à distância de ensino acaba por atender às novas demandas educacionais e profissionais socialmente impostas.

Ao longo do seu processo escolar, o indivíduo acaba por procurar e localizar formas preferenciais de aprendizagens. Nesse período, o ensino à distância acaba por ganhar visibilidade e preferência pois oportuniza e viabiliza

várias linguagens de aprendizagem para o conteúdo desejado em tempo acessível ao aluno, muitas vezes já inserido no mercado de trabalho. A aula passa a ser direcionada ao aluno, que à sua maneira, no seu tempo, participa através de debates, conferências, fóruns e videoaulas oferecidos, tornando ele um ser ativo no seu processo de formação e desenvolvimento intelectual.

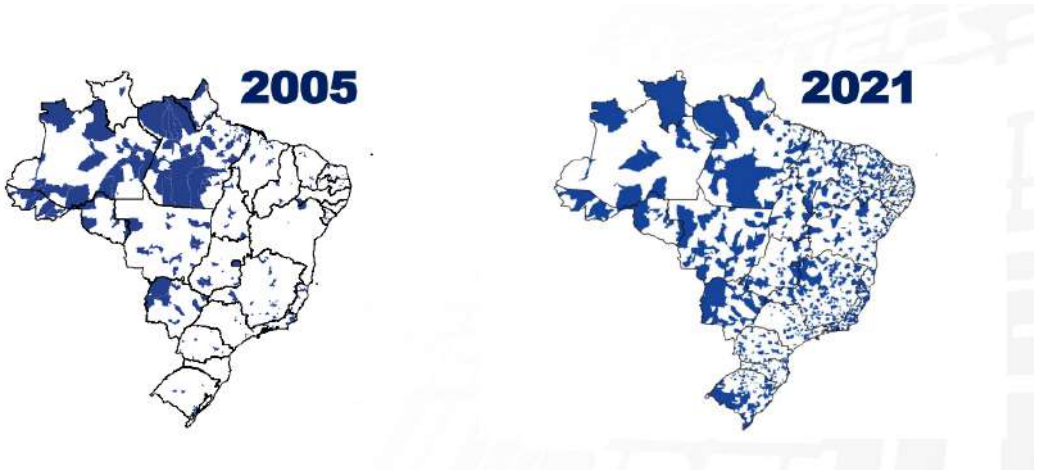
Disponível em: www.pixabay.com/Acesso em 22 abr. 2023

No mundo digital a qual estamos inseridos, com um smartphone na mão, já é possível realizar qualquer curso on-line. Além disso, sem a necessidade de manter espaços físicos para comportar seus alunos, diversas universidades acabam por investir na expansão de campus digitais. Com uma única aula gravada, um professor pode alcançar milhares de alunos no Brasil e no mundo, dando a todos eles as mesmas condições de aprendizagem, barateando assim custos que exigiriam se estivesse em um espaço físico.

Com esse barateamento de custos, as instituições privadas repassam valores ainda mais acessíveis a seus alunos, que se interessam em obter diplomas de graduação, pós e cursos técnicos, uma vez que isso passa a caber no seu orçamento.

Entre os outros elementos que fazem a educação à distância manter o êxito e a procura estão a flexibilização de horários que ela pode promover para o estudante, atendimento personalizado, inovação das metodologias de ensino, aperfeiçoamento e novas possibilidades de avaliação da aprendizagem, assim como a ampliação de relacionamentos interpessoais construídos. Além disso, o EaD respeita o período de concentração e interesse do aluno para o estudo, promovendo assim a continuidade da construção de sua autonomia.

Nos últimos anos, o número de municípios que vêm aderindo à modalidade educacional à distância vem aumentando, promovendo um crescimento nunca antes visto no campo educacional brasileiro.

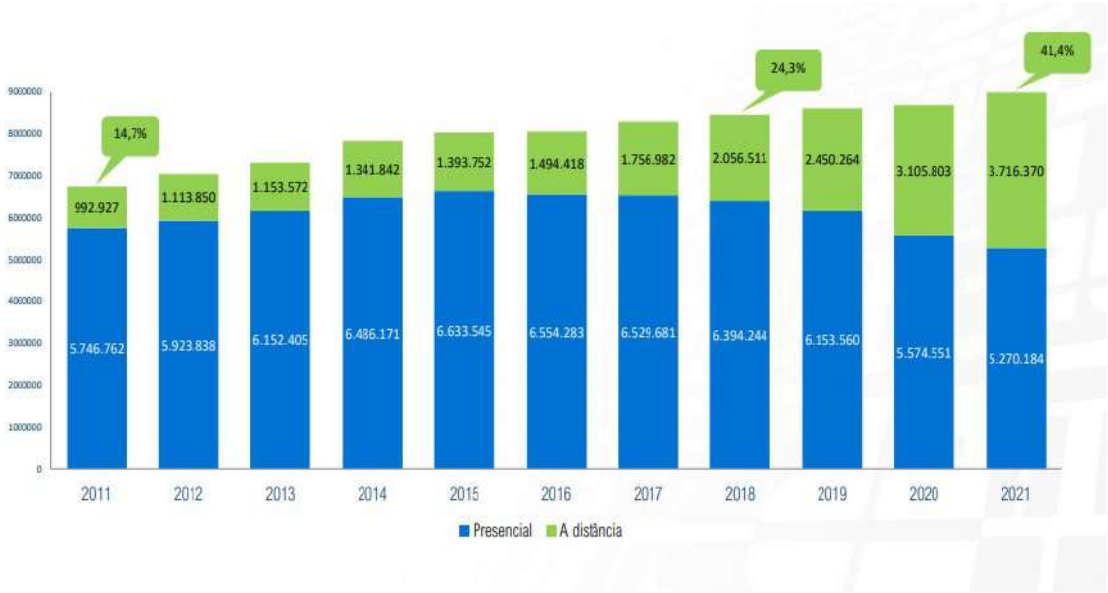


Disponível em: www.inep.gov.br/Acesso em 21 abr. 2023

Segundo dados do INEP, em 2021, a matrícula na rede federal estava presente em 931 municípios brasileiros, por meio de campi com cursos presenciais ou de polos EaD. Sendo 101 deles nos municípios na região Norte; 315, no Nordeste; 249, no Sudeste; 174, no Sul; e 92, no Centro-Oeste do país.

Apesar da procura e da visibilidade que mantém nos últimos anos, o Ensino à Distância também é alvo de críticas referentes à abordagem qualitativa do ensino, ao alcance construído na aprendizagem dos educandos e na superação do estigma que a caracteriza como uma “facilitadora de diplomas”. Essas críticas podem aprimorar ainda mais a metodologia adotada, assim como construir formas de solucionar problemas que surgem.

Historicamente desigual, a formação da sociedade brasileira se apoiou na educação como a forma mais democrática e segura de ascensão política e social, promovendo inclusão e possibilidade a populações historicamente excluídas. O ensino à distância, como parte desse histórico educacional, também acaba por ter sua função social.

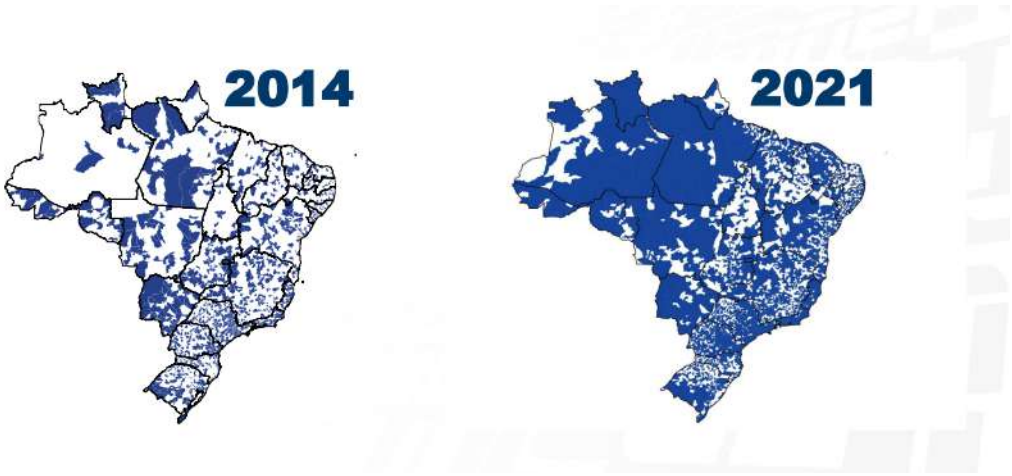


Disponível em: www.inep.gov.br/Acesso em 21 abr. 2023

Segundo dados do INEP, desde o ano de 2016, a matrícula em cursos presenciais na rede privada de educação superior tem diminuído e esse comportamento é acompanhado pelo aumento do ritmo de crescimento dos cursos EaD. Em 2021, o número de matrículas em cursos à distância da rede privada acaba ultrapassando o número de matrículas em cursos presenciais.

O ensino à distância respeita os interesses e as necessidades da população brasileira, rompendo assim com os desejos da classe dominante do país que se enriquece com a pobreza e a desvalorização do outro, elevando possibilidades e democratizando o acesso à educação.

A formação continuada oferecida pelo ensino à distância eleva a motivação e o interesse para a continuidade da aprendizagem, aperfeiçoando e favorecendo habilidades profissionais e educativas. Vale ressaltar que o sucesso das EaD´s vem dos resultados concretos que seu ensino oferece percebidos na redução da disparidade econômica e educacional entre os brasileiros, promovendo o exercício da cidadania a muitos brasileiros, elevando assim a compreensão política e cultural do brasileiro, além de promover o desenvolvimento do país.



Disponível em: www.inep.gov.br/Acesso em 21 abr. 2023

Segundo dados do INEP, em 2021, a matrícula na modalidade EaD estava presente em 2.968 municípios brasileiros, por meio de campus das IES ou de polos EaD. Um aumento de quase 120% quando comparado com o ano de 2014.

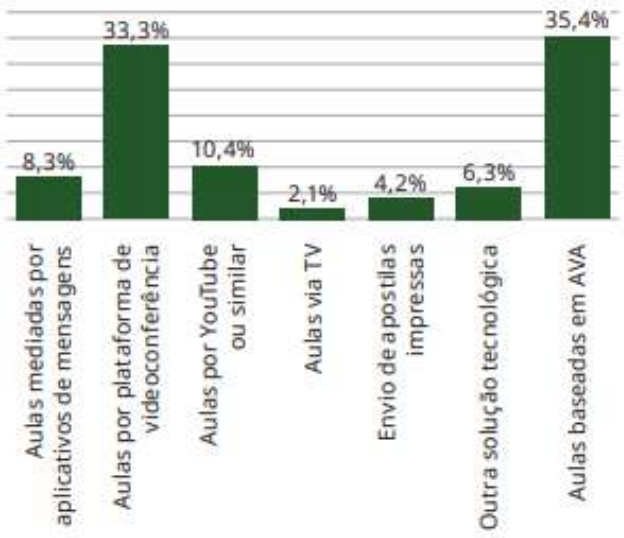
1.2 Cenários escolares em tempo de COVID-19:

Em 11 de março de 2020 o mundo foi agitado pela declaração da Organização Mundial da Saúde (OMS) de que estaríamos inseridos em uma pandemia, provocada pela COVID-19, doença causada pelo novo coronavírus (Sars-Cov-2), atingindo mais de 180 países. Por se tratar de uma doença altamente contagiosa para a qual - à época - não havia tratamento nem vacina, o mundo passou a adotar medidas de distanciamento físico, adquirindo novos hábitos como o *home office* e a suspensão das aulas nos espaços escolares físicos, migrando as interações escolares para ambientes virtuais até então utilizadas no Brasil, prioritariamente, pelo método de Educação à Distância.

Com a defesa do distanciamento físico e do isolamento social como melhores práticas de prevenção, passamos a forjar outras relações de convívio social e, no campo educacional, práticas pedagógicas urgentes tiveram que ser acionadas para lidarmos com a pandemia. Essas práticas acabaram por desvelar desafios e tensões que a educação já vinha enfrentando nos últimos anos.

Disponível em: www.pixabay.com/Acesso em 22 abr. 2023

A pandemia acabou por amplificar crises econômicas, políticas e sociais, tornando-as maiores e mais complexas e, ao mesmo tempo, denunciantes. Na educação não foi diferente. Com a busca por soluções imediatas, muitas escolas adotaram estratégias alternativas, modificando suas rotinas pedagógicas, para que seu ano letivo não fosse suspenso e a manutenção dos vínculos na comunidade escolar fosse garantida, promovendo uma aceleração de muitas delas às exigências tecnológicas do século XXI, inclusive alterando sua infraestrutura física e tecnológica.



Soluções tecnológicas para continuidade das aulas durante a pandemia

Disponível em: www.abed.org.br/Acesso em 22 abr. 2022

Conforme indica o gráfico acima, a cultura estabelecida na EAD clássica, baseada em ambientes virtuais de aprendizagem (AVA), apresentou-se, inicialmente, como a principal alternativa em 35,4% dos casos. No entanto, os dados indicam uma proximidade em relação à tendência que se consolidou com o prolongamento da urgência social: as aulas por plataformas de videoconferência.

Isso pode ser constatado com a crescente utilização dos serviços de videoconferência – como Zoom e o Google Meet – na educação, em todos os níveis e modalidades, a partir do segundo trimestre de 2020. Essa tendência é seguida por aulas em plataformas de compartilhamento de vídeos, como YouTube, e pela utilização complementar de aplicativos de mensagens instantâneas, por exemplo, Telegram e WhatsApp, como alternativas de atendimento.

Disponível em: www.pixabay.com/Acesso em 22 abr. 2023

Vale ressaltar que muitos nomes foram dados à modalidade de educação não presencial, divulgados na grande mídia como se pertencessem a um “pacote único” de ensino, e não é por aí. Educação a distância, ensino híbrido, educação online e ensino remoto foram os termos mais utilizados e propositalmente embaralhados como se a metodologia, as competências e habilidades utilizadas fossem a mesma. E não são.

Presente na Legislação Educacional desde a promulgação da Lei de Diretrizes e Bases (LDB) nº 9394/96, com oferta em instituições de ensino,

especialmente de nível superior, a *Educação a Distância* – EaD adquiriu uma visibilidade nas demandas e preocupações da sociedade brasileira a partir do distanciamento social imposto pela pandemia de COVID-19. Emergencialmente as instituições de ensino, a mídia e a sociedade em geral, passaram a falar de EaD em uma confusão de nomes que denuncia o desconhecimento da modalidade, utilizando o termo como sinônimo de outras modalidades de ensino não presencial.

. Contemplada especialmente no artigo 80 da LDB, a EaD tem hoje um decreto nº 9.057/2017, que a define em seu 1º artigo como uma modalidade educacional na qual a mediação didático-pedagógica nos processos de ensino e aprendizagem ocorra com a utilização de meios e tecnologias de informação e comunicação, com docentes qualificados, com políticas de acesso, com acompanhamento e avaliação compatíveis e desenvolva atividades educativas por estudantes e profissionais da educação que estejam em lugares e tempos diversos (BRASIL, 2017).

Tal modalidade de ensino acumula construções sobre formatos de desenvolvimento curricular, material didático multimídia, procedimentos avaliativos específicos, demandas de conteúdos e práticas a serem trabalhadas em formato presencial e/ou a distância, além de ambientes e ferramentas de mediação virtual. Vale ressaltar que a ausência física do professor em sala de aula, substituída por uma tela de transmissão, não caracteriza a modalidade como EaD.

Disponível em: www.pixabay.com/Acesso em 22 abr. 2023

O *ensino híbrido*, por sua vez, já surge como uma modalidade que consegue somar as dinâmicas educacionais presencial e a distância de forma igualitária, promovendo um encontro de saberes e práticas que não se sobrepõe à outra, mas sim promove uma mediação, desenvolvendo habilidades, conteúdos e produções.

Atualmente, no Brasil, este formato é regulado pelo Ministério da Educação que oferta, por Instituições de Educação Superior (IES), disciplinas na modalidade a distância em cursos de graduação presencial. De acordo com esta regulamentação as IES podem ofertar entre 20% e 40% da carga horária total dos seus cursos na modalidade a distância.

Vale destacar, no entanto, que a perspectiva de ensino híbrido pressupõe a realização de atividades com presença física, o que só foi possibilitado em 2021 com as políticas de vacinação e o uso de medidas preventivas como o acesso às máscaras e o uso do álcool em gel.

A *educação on-line*, muitas vezes confundida com as modalidades já citadas, é um conceito amplo e multifacetado, mas sem regulamentação no Brasil. Por apresentar um caráter complexo, a educação on-line é compreendida como um conjunto de ações de ensino-aprendizagem mediadas por tecnologias digitais que fortalecem práticas interativas.

Enquanto fenômeno nascido da cibercultura, a Educação On-line, portanto, não é sinônimo de EaD. Acaba por se definir apenas como uma perspectiva pedagógica que pode ser assumida como potencializadora de situações de aprendizagem mediadas por encontros presenciais, a distância ou em processos híbridos.

Deste modo, não são os encontros mediados por tecnologias e o uso de ambientes virtuais de aprendizagem que caracterizam essa perspectiva de ensino, mas sim a cibercultura enquanto fenômeno social associado à maneira como anônimos ou personalidades se apropriam das tecnologias digitais e do ciberespaço por meio de processos interativos e tutoriais de ensino, como exemplo, poderemos citar youtubers, coachs e até digital influencers.

Disponível em: www.pixabay.com/Acesso em 22 abr. 2023

A crítica feita a essa modalidade é que, muitas vezes, o processo de ensino adotado é centrado em pressupostos pedagógicos sem embasamento teórico, com fins de visibilidade, engajamento e lucro pautados no sucesso da sua transmissão, adotando lógicas massivas das mídias de massa.

Em razão da pandemia de COVID-19, um outro termo ganha repercussão e visibilidade na sociedade: o *ensino remoto*. A legislação educacional construída em razão da pandemia de COVID-19, não contempla o ensino remoto como modalidade de ensino. No entanto, esse termo se popularizou na mídia, nas redes sociais digitais e entre gestores públicos na tentativa de nomear as ações pedagógicas criadas para atender às regulamentações emergenciais referentes à educação escolar em tempos de pandemia.

Embora a legislação não conceitue o ensino remoto nem o adote como categoria de ensino, seu uso em situações atípicas e emergenciais como pandemias e outras catástrofes é eficaz e possível para a continuidade das atividades pedagógicas com o objetivo de diminuir os prejuízos derivados da suspensão das aulas presenciais e acaba evitando que o vínculo pedagógico planejado pela escola não seja rompido totalmente.

Vale destacar que nomear as modalidades, tipologias e práticas de ensino feitas aqui é promover a valoração das modalidades feitas em outro espaço escolar, evitando assim a precarização do sistema de ensino e o enfraquecimento e fragilização dessas áreas educacionais.

A pandemia da COVID-19 evidenciou as fragilidades da educação, expondo uma necessária mudança nos modos de ensino e aprendizagem no século XXI. A educação no contexto da cibercultura evoca o princípio de Paulo Freire de que educar não pode se resumir a práticas de transmissão de conteúdo.

1.2 Experiências Pedagógicas e COVID-19:

Não há dúvidas que a pandemia de COVID-19 acabou gerando perdas para a educação e para aprendizagem dos mais de 47.000.000 de estudantes matriculados no país[1]. Na rede pública, esse abismo ainda foi maior, cabendo às secretarias propor alternativas para a suspensão das atividades presenciais físicas e um outro planejamento para o retorno dos estudantes.

Segundo dados do CIEB (Centro de Inovação para a Educação Brasileira) muitas escolas acabaram por adotar em caráter emergencial a mudança de seus calendários pedagógicos adotando a suspensão, o adiantamento de férias, recessos e até a permanência das atividades escolares. Esse tempo fez com que outras estratégias de ensino fossem valorizadas como o uso da modalidade EaD e até mesmo – na eficácia do programa de vacinação – a abertura para o ensino híbrido.

Fonte: Centro de Inovação para a Educação Brasileira.

Com o início da pandemia de COVID-19, muito se alterou nas casas dos estudantes brasileiros. Houve um aumento nas taxas de desemprego fazendo

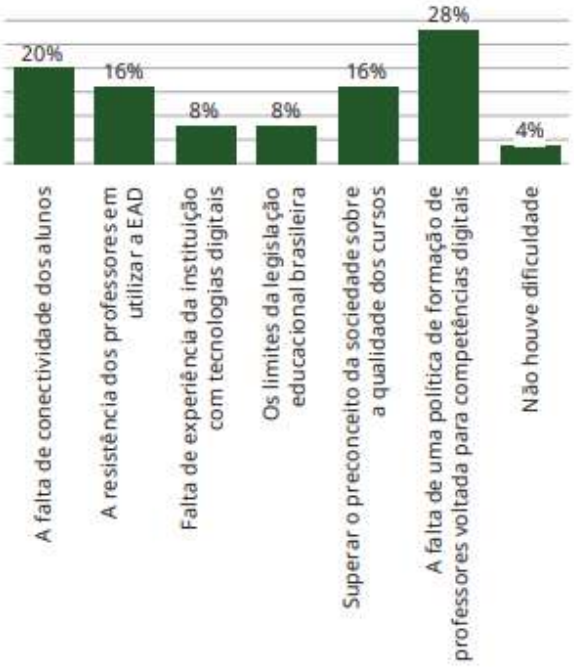
[1] Dados do Censo Escolar 2019 produzido pelo Instituto Nacional de Estudos e Pesquisas Educacionais Anísio Teixeira (INEP) (BRASIL, 2020).

com que, todo orçamento do brasileiro passasse por alterações, já que o valor da faculdade era incompatível com a renda que possuía e isso é um importante fator que diz muito sobre o perfil dos alunos da modalidade EAD.

Com a crise sanitária, o número de evasões e de desistentes também teve seu aumento. Se antes as pessoas deixavam seus cursos por diversos motivos, indo da falta de identificação com a escolha até mudanças de planos em suas vidas, vemos, hoje, o fator econômico mais presente.

Segundo dados do censo da ABED (Associação Brasileira de Educação a Distância) de 2020 no modelo EAD, grande parte dos estudantes cursam licenciatura, o que acaba por contribuir com o aumento de mestres e doutores lecionando nas graduações.

Vale ressaltar que há fatores que levam ao afastamento de muitos alunos do ensino à distância, que segundo dados da ABED, vão da resistência das instituições e de professores ao uso de plataformas digitais, assim como ausência de capacitação para isso, devido à falta de conectividade dos alunos.



Maiores dificuldades para a ampliação da oferta da EAD

Disponível em: www.abed.org.br/Acesso em 22 abr. 2022

Ainda segundo dados do último ano, a grande maioria das pessoas que fazem cursos à distância são mulheres, que, por vezes, no meio de suas jornadas triplas entre trabalho, casa e maternidade, veem na EAD uma oportunidade de melhorar seus currículos sem sair de casa.

Porém, quando o recorte racial é feito, a pesquisa revela que a população negra não compõe a maioria em nenhuma modalidade de ensino. Na região Sudeste, as instituições presenciais e públicas têm conseguido igualar a quantidade de negros e brancos nas salas de aulas graças à implementação das cotas. A única região onde é possível notar uma quantidade de pessoas negras maior que a de pessoas brancas é o Nordeste do país, que respeita a relação demográfica de seus estados, compostos, em grande parte, por negros.

Disponível em: www.pixabay.com/Acesso em 22 abr. 2023

Quanto à classe social, as últimas pesquisas apontam que, na modalidade EAD, a maioria dos estudantes pertencem às classes C, D e E, enquanto na modalidade presencial, as classes A e B ainda ocupam bastante espaço, tanto em universidades públicas quanto em privadas, mesmo que estas correspondam a uma pequena faixa da população brasileira.

A possibilidade de realizar uma graduação à distância usando a internet expandiu o acesso e a chance de muitas pessoas realizarem seus sonhos, crescerem em suas carreiras e, até mesmo, mudarem suas realidades. Vale ressaltar que o fato do estudante poder escolher a que horas e de que maneira

estudar abre as portas da graduação àqueles que fazem parte da grande mão de obra trabalhadora do país.

Essa mudança é fundamental no combate às desigualdades sociais e econômicas e prova que a educação é um caminho necessário para o exercício da cidadania e para a correção de falhas estruturais em um Brasil que luta para eliminar sua histórica desigualdade.

2. BRASIL, BRASIS:

2.1 Formação do Povo Brasileiro

A formação do Brasil e da sua população não pode ser definida por palavras-chave ou por teorias, pois trata-se de uma experiência e um processo histórico, repleto de contradições, conflitos e desigualdades e ainda em construção. No percurso escolar que temos, somos levados a entender o Brasil como o país da diversidade cultural, que devido a mistura de raças originou um "povo novo", tolerável, livre de preconceitos, um país diferente, onde todos cabiam, todos podiam.

Para a filósofa Marilena Chauí (2001) existe uma crença generalizada de que o Brasil é um país pacífico, generoso, alegre e até sensual; que apesar de sofredor, é um país sem preconceitos e acolhedor; que seus contrastes regionais, geram um país diverso e plural, faltando ao país apenas investimentos para a sua modernização e assim ser valorizado como os demais. Mas não é bem por aí.

Disponível em: www.pixabay.com/Acesso em 22 abr. 2023

Essa ideia de um país diverso – logo sem preconceitos - vem da falsa ideia de uma formação do povo brasileiro. Segundo Darcy Ribeiro (1995) surgimos do choque do invasor português com índios silvícolas e com negros africanos, estes dois últimos, povos que acabaram sendo escravizados. Ou seja, o povo brasileiro surge a partir de uma “mistura” ocasionada pelos povos responsáveis por sua formação, em uma violenta miscigenação de culturas.

Após o primeiro contato com os povos nativos, com violências que iam da invasão territorial ao etnocídio e genocídio de tribos indígenas, o plantio da cana de açúcar e a necessidade da mão de obra para sua execução fez com que inúmeros africanos fossem extraídos de suas terras para viverem na condição de escravizados. Tal fato é significativo na formação do Brasil. Estes foram trazidos brutalmente a um lugar até então desconhecido, distante de suas referências culturais e familiares, cabendo a eles recriar sua maneira de ver e agir no continente novo.

Muitos acabavam não suportando o sofrimento causado pela travessia, e como resistência recorriam ao suicídio, à violência e a formação de quilombos para se livrar da exploração e elaborar uma cultura à parte da ordem colonial. Outros buscavam meios de comprar a sua própria liberdade ou, mesmo sendo vistos como escravizados, conquistaram funções e redes de relacionamentos que lhes concediam uma vida com maiores possibilidades.

Vale ressaltar que o processo de miscigenação não se limitou ao contato entre o português e o indígena; a mistura de povos se estendeu também à

população negra. Com o passar do tempo, o reconhecimento desses sujeitos miscigenados passou a ser limitado pela cor da pele, distinguindo-os de outros grupos sociais.

PARA PENSAR O BRASIL:

O Censo Demográfico 2022 iniciou em agosto a coleta de dados em domicílios por todo o Brasil. Além de atualizar as estatísticas - a edição anterior foi a de 2010 - a pesquisa do IBGE tem relevância específica para populações como indígenas e quilombolas. "Primeiro, estar nas estatísticas oficiais já garante a esses grupos a existência perante o Estado. Nos materiais didáticos, na informação que circula na sociedade como um todo, muitas vezes eles estão como algo do passado. Tem esse papel de visibilizar a diversidade étnica que há no Brasil em termos de povos e comunidades tradicionais", explicou a Ecoa a coordenadora do Censo de povos e comunidades tradicionais do IBGE Marta Antunes.

Fonte: www.uol.com.br/ecoa/Acesso em: 23 abr. 2023

De forma lenta a agricultura e o uso da mão de obra escravizada foram sendo substituídos pela modernização capitalista. O trabalho escravo acabou abrindo espaço para a entrada de outros povos. Italianos, alemães, poloneses e portugueses não suportando os abalos causados pelas tensões políticas dos seus países, acabam vendo no Brasil em desenvolvimento a forma mais segura de buscar novas oportunidades contribuindo para a formação do operariado nas primeiras jornadas de trabalho em ambiente fabril.

Ao analisarmos a formação do povo brasileiro e seu processo de miscigenação, poderíamos pensar que as influências dos diversos povos promoveriam uma sociedade multiétnica, porém, ocorre o contrário, já que tais sujeitos foram condicionados a reproduzirem valores do país colonizador, sendo condicionado a adotar um papel receptor nesse processo de formação. A resistência desses povos fez com que o efeito da colonização fosse menos evidente, daí a leitura de um país miscigenado, diverso.

Todo esse processo formador deu origem a uma distância social no Brasil que separa e opõe diferentes grupos sociais, além de uma discriminação

que pesa sobre negros e indígenas, com imposições que anulam a manifestação de suas culturas de forma qualitativa e sem repressão.

Vale ressaltar que o primeiro desafio cultural imposto ao negro e ao indígena foi aprender a língua portuguesa em prol de sua comunicação e sobrevivência. Posteriormente, o domínio da língua portuguesa ocasionou uma difusão em todo o território, originando uma nova língua que contava tanto com dialetos tupi-guarani como de povos iorubás e bantos.

Apesar do Brasil ser um país que tem todo seu processo de construção relacionado à mistura de raças, onde índios e negros tiveram um maior destaque na sua formação cultural, a sociedade ainda se mostra resistente em assumir e reconhecer seu processo de formação, mantendo atitudes e pensamentos discriminatórios e preconceituosos, a favor de um embranquecimento de sua história, elencando a culinária e a linguagem como únicas formas de transmissão cultural dos povos nativos e africanos.

Assim, é urgente o reconhecimento desses povos na formação cultural brasileira a partir das suas práticas, representações, símbolos e rituais, demarcando identidades, proporcionando o fortalecimento da autoestima, e na reconstrução de nossa história.

2.2 Uma identidade para o Brasil

Enquanto sujeitos sociais que somos conduzimos nossas experiências por representações de quem somos, de quem podemos ou desejamos ser. Por envolver interação social, afetos, autoestima, a identidade é uma categoria social construída, expressa e percebida por diferentes linguagens: escritas, corporais, gestuais, imagéticas e midiáticas.

Fruto da ideia de pertencimento a um grupo, a um povo, a um movimento, produzindo significados, a identidade brasileira está ligada à formação cultural de um ou mais grupos humanos e está em constante transformação. Essa construção se apoia, também, de diversas matérias-primas fornecidas pela nossa história, pela geografia local, pela biologia, pela memória.

Disponível em: www.pixabay.com/Acesso em 22 abr. 2023

Ao se pensar em apresentar o Brasil e o brasileiro, acabam, de certa maneira, por apresentá-lo, a partir de seu caráter "identitário", ainda que por uma identidade construída por meio da diferença, da miscigenação, da dominação. No entanto, embora podendo surpreender pela força da expressão, é possível se encontrar essa identidade nacional no brasileiro?

Pensemos na dimensão e na diversidade geográfica e territorial do Brasil, na variedade de costumes, culinárias, sotaques, na diversidade de seus valores, ideias, práticas, ritos, religiões, manifestações, tradições, expressões, sentimentos, que podemos verificar nas ruas, praças, casas, famílias, trabalhos, igrejas, empresas, entidades e associações, localidades, estados e regiões, conforme nos demonstra Roberto DaMatta em *O que faz o brasil, Brasil?* (1997).

O que nos faz então sermos reconhecidos como "brasileiros" quando somos vistos pelo olhar do outro, pelo olhar estrangeiro? Como brasileiros das mais diferentes regiões, costumes e crenças se reúnem, se unificam, se identificam e são reconhecidos como tal, ou seja, como "brasileiros”?

Podemos apontar os fatores dessa unidade, como a língua falada e escrita que, apesar dos sotaques e usos regionais, apresenta sua semelhança; algumas comidas que já se tornaram nacionais; determinados “gostos” nacionais e certos comportamentos; algumas festas e manifestações; alguns sentimentos e expressões, que nos moldam de acordo com um "jeito de ser” brasileiro. Vale

ressaltar que a nossa identidade não nasceu pronta, ela foi planejada, construída.

A identidade brasileira, apesar de ser construída pela formação de povos nativos, africanos e europeus, se assumiu como “brasileira” apenas após o seu processo de independência, apenas durante o Segundo Reinado, quando o Brasil temia se fragmentar, e acabou buscando elementos que unificavam o território e o seu povo.

Disponível em: www.pixabay.com/Acesso em 22 abr. 2023

Um projeto de construção da identidade nacional brasileira utilizou códigos de reconhecimento mútuo no país, com elementos narrativos de uma história que focava nas virtudes da jovem nação, dando orgulho a seus habitantes, e assim, confiança em relação ao seu futuro – aqui início do Segundo Período Monárquico.

Naquele momento, o Brasil não precisava de uma História que falasse das tensões e sim de uma união, de um povo só. Mas afinal, o que tornava alguém brasileiro no século XIX? Quais elementos originais o Brasil necessitava para marcar seu lugar entre as grandes nações?

Literatura, história e geografia passaram a se tornar, cada uma à sua maneira, campos de ponto de união do Brasil e de diferenciação em relação aos europeus, afinal, destacando a exuberância do país. Porém, apenas a natureza não era suficiente para apresentar a originalidade do Brasil, era preciso apresentar o fruto daquele habitat.

O homem brasileiro, representado principalmente pelos elementos indígenas e mestiços, foi utilizado para materializar o exuberante país, realizando a ideia da mestiçagem das três raças, construindo um mito de uma democracia racial no país. Vale ressaltar que as três raças não foram assumidas na sua “forma natural”, logo caberia também o papel de aperfeiçoar essas raças – através do branqueamento de sua população e a civilização do indígena – para o desenvolvimento da nação.

Estavam então determinados os elementos que definiriam a identidade nacional brasileira: a natureza e sua gente. Também já se sabia o modelo civilizacional que o Brasil deveria seguir e o modo como ele se destacaria entre essas nações, era um país unido, democrático, para todos.

Chegamos, enfim, à questão: mas e o povo brasileiro, quem é? Dentre tantos autores que procuram responder isto, podemos citar Sérgio Buarque de Hollanda, Darcy Ribeiro e suas respectivas obras. Dessas leituras, podemos extrair uma conclusão: os brasileiros são tantos, tão variados e diversificados, tão múltiplos em suas origens e trajetórias que talvez nunca venhamos saber de fato quem ele é.

Poderemos falar em fatores de unidade que construídos ao longo da história nos aproximam enquanto brasileiros. Porém pensemos também na diversidade que segue, indo da formação étnica, nas diferenças por fatores como a região geográfica, gênero, geração e classes sociais.

3. O TRABALHO NO SÉCULO XXI:

3.1 Tenha um bom trabalho precarizado

Desde as primeiras décadas do século XIX, ao ganhar força no processo industrial, o trabalho assalariado, ao mesmo tempo que se expandiu, promoveu e acompanhou as grandes mudanças sociais do seu tempo, revelando-se como um novo fator de estruturação de desigualdades sociais.

Foi com a Revolução Industrial, na Inglaterra, que se tornou evidente essa realidade social, mediada pela produção e pelo capital, além claro da força crescente do movimento operário que acabou por, em suas reivindicações, ajudar a mostrar a natureza da economia capitalista, além das desigualdades econômicas e sociais.

A histórica luta travada pela classe operária foi marcada por conflitos, mas também por progressos; podemos destacar entre eles a conquista das 8

horas de trabalho diárias, o direito à greve, à sindicalização e a CLT. Porém, tais conquistas se desfazem quando ao entrarmos no século XXI, assistimos à precarização do trabalho em nome de uma modernização fria e excludente.

Este processo de precarização não é recente, nem veio com a virada do século. Promovendo desigualdades dentro do mercado de trabalho, a precarização se manifesta na imposição de uma economia informal ao trabalho flexível (combinando diferentes rotinas de trabalho), do trabalho de gerações mais jovens (de caráter temporário, parcial e que não valoriza suas qualificações escolares) até a adaptações aos setores tecnológicos e esse processo veio com a adoção do neoliberalismo nos países desenvolvidos.

A entrada do século XXI acabou por marcar novas medidas para as relações trabalhistas. O modo de produção exigido e a acumulação capitalista submetem o trabalhador às suas regras, mutáveis pelas exigências do mercado fazendo-o empenhar-se por inteiro ao trabalho, como se fosse parte da empresa. As leis de mercado, agora determinadas pelo neoliberalismo, enaltecem a mercadoria e o consumo, buscando, como resultado, um lucro de curto prazo, independente das condições do trabalhador.

COM A PALAVRA, O CONCEITO:

Neoliberalismo é uma teoria econômica que defende a livre-iniciativa, ou seja, o direito de um empresário fazer investimentos no ramo que considerar mais lucrativo, opondo-se à presença do Estado na economia.

O neoliberalismo acaba por produzir um quadro desolador para os trabalhadores, que se veem impedidos de ter acesso a um emprego, dificultado com a exigência de qualificação técnica para as novas tecnologias que surgem com o novo modelo de acumulação de capital. Os altos índices de desemprego produzem, o que Marx chamaria de “exército de reserva” , pondo a classe trabalhadora em condições precárias de trabalho.

Tal precarização é materializada nos processos de terceirização, subcontratação e trabalho informal. O discurso neoliberal também se materializa com as propostas de privatizações de instituições públicas, redução de gastos

públicos e ataque aos movimentos sindicais, a defesa de uma flexibilização dos direitos trabalhistas, redução de salários e redução de benefícios previdenciários. O pensamento neoliberal também considera que os sindicatos, historicamente, criaram uma rigidez nas relações de trabalho, dificultando sua flexibilidade.

QUESTÕES PARA REFLETIR:

- Qual será o destino dos trabalhadores diante do neoliberalismo?
- Qual será o destino dos trabalhadores que não conseguem se engajar nas novas tecnologias exigidas no mundo do trabalho?

O pensamento neoliberal também tem como característica justificar os graves problemas sociais, entre eles a pobreza e a miséria. Tais problemas são entendidos pelo neoliberalismo como naturais, já que são esses "miseráveis" os próprios culpados por estarem nessa condição, pelo simples motivo de que se todos têm as mesmas oportunidades por mérito e esforço, só os mais hábeis teriam sucesso e por conseguirem não teriam responsabilidade pela miséria alheia. Para o pensamento neoliberal, não existiria exploração, pois todos são livres para escolher o seu próprio caminho, o de quem serve e o de quem é servido.

Disponível em: www.pixabay.com/Acesso em 23 abr. 2023

Outro discurso criado pelo neoliberalismo é que essas mudanças no campo trabalhista devem ser entendidas pelos trabalhadores como normal para

o momento atual, pois são necessários sacrifícios para que haja o progresso e o desenvolvimento da sociedade.

Para que as políticas neoliberais sejam implantadas com sucesso, além do consentimento do operário da “sua cooperação para o sucesso da empresa”, é necessária também a “cultura do medo”. Esta coerção está presente, por exemplo, no medo do desemprego, da miséria, da inadimplência, representando uma forma eficaz de controle capitalista do trabalho nas condições de vida do trabalhador. O medo do desemprego, por sua vez, precariza o trabalhador, comprometendo possíveis reivindicações suas em atividades sindicais.

PARA IR ALÉM:

Dois dias, uma noite: (Bélgica, 2015. Direção de Jean-Pierre Dardenne e Luc Dardenne. Duração: 95 min)

O filme retrata a história de Sandra, que acaba sendo vítima de uma votação realizada na empresa em que trabalha: os demais funcionários decidiram que ela deveria ser demitida para que eles pudessem ter seu aumento salarial. O resultado do voto só será legitimado na segunda-feira, o que dá a Sandra a chance de reverter a situação ao longo do fim de semana.

Assista ao filme e identifique como os diretores trabalharam a questão do neoliberalismo e a precarização do trabalho.

3.2 Terceirização do Trabalho

Em condições precárias impostas, o trabalho é cotidianamente associado à insatisfação, sofrimento e até tortura, além de outros adjetivos que o associam sempre a um grande esforço. O próprio termo trabalho historicamente acaba por ter significados distintos ao longo da sua história. Segundo João Bosco Santos[2], o termo surgiu no século XI, originado da palavra latina *tripalium* que significava um instrumento de três pés utilizado para torturas.

[2] SANTOS, J. **O avesso da maldição do Gênesis:** a saga de quem não tem trabalho. São Paulo: Annablume; Fortaleza: Secretaria da Cultura e Desporto do Governo do Estado do Ceará, 2000.

Outro termo indicado como surgimento da palavra é *trabaculu*, da mesma raiz da palavra trava ou travar, tendo no latim o significado da corda que amarrava escravizados, forçando-os ao trabalho. Assim, o termo trabalho teve como significado primeiro o de castigo, tortura, tormento e sofrimento.

Em alguma medida, esses significados permanecem até nossos dias, principalmente quando não se estabelece uma relação de prazer do trabalhador com o resultado de seu trabalho. Segundo Santos, algumas expressões que concebem o trabalho como negação ou sofrimento ainda são utilizadas pelo senso comum: “dar trabalho”, no sentido de exigir esforço ou atenção; causar transtorno ou preocupação; uma “coisa trabalhosa”, algo fatigante, difícil, demorado.

Esses significados passam uma imagem de trabalho que se distancia do prazer e se coloca apenas como uma necessidade misturada à obrigação de sobrevivência dentro de um sistema capitalista, tendo como base o tripé trabalho–salário–consumo.

Disponível em: www.pixabay.com/Acesso em 23 abr. 2023

Com a implementação do novo modo de produção e de acumulação capitalista, a noção de trabalho se amplia, reduz a classe trabalhadora, mas mantém sua imagem de esforço e tortura. A terceirização, a contratação temporária, a contratação precária sem o registro em carteira e a subcontratação são elementos do neoliberalismo e demarcam os espaços ocupados pela classe trabalhadora.

A terceirização acaba sendo posta em prática porque as organizações a veem como necessária para o crescimento e desenvolvimento de suas áreas, promovendo eficiência, redução de custos, maior rentabilidade, otimização do tempo e melhora na qualidade do serviço.

A lei de mercado, imposta pelo neoliberalismo, faz com que direitos trabalhistas adquiridos ao longo do tempo acabem fragilizados. Com a flexibilização das leis trabalhistas, a precarização do trabalho acabou se estabelecendo, dando espaço para que empresas desrespeitem durante a contratação a garantia de direitos aos trabalhadores e o trabalhador refém da "cultura do medo" do desemprego acabam aceitando as condições impostas.

Então, os empregadores sentem-se autorizados a também não cumprirem outras normas obrigatórias. Com isso, pouco a pouco, o direito do trabalho vai desaparecendo, se distanciando daqueles princípios de proteção ao trabalhador.

O neoliberalismo e consequentemente a terceirização do trabalho apresentam realidades difusas que confundem o trabalhador, dando a ele a falsa ideia de que mais opções de trabalho surgiriam e que ele estaria mais seguro se fizesse seu trabalho.

Por um lado, assistimos às novas tecnologias na indústria e avanços do setor terciário produzindo empregos mais sofisticados, tanto na área técnica quanto profissional, exigindo um conhecimento teórico maior; por outro, vê-se o crescimento da precarização e surgimento do trabalho em domicílio e à distância. A terceirização surge então como uma forma de burlar as leis trabalhistas, reduzindo os custos com o trabalho. O trabalhador continuaria empregado, mas sem vínculo com empresas.

Com o isolamento do trabalhador, distanciando-o da empresa, um efeito dominó surge, já que a terceirização o desmobiliza politicamente e enfraquece sua participação em movimentos sindicais, tornando-o vulnerável à fácil redução de salários, flexibilização da sua jornada de trabalho e sem garantias de direitos trabalhistas. Vale ressaltar que uma outra defasagem no campo trabalhista imposto pela terceirização é o distanciamento do trabalhador à sua formação de

consciência de classe, fragmentando-o à mera função que ocupa naquela empresa.

Disponível em: www.pixabay.com/Acesso em 23 abr. 2023

Destaca-se que com a precarização das relações trabalhistas acaba-se por reduzir também a classe trabalhadora. O espaço da política, representado pelos sindicatos, aos poucos começa a ser corroído. O abalo dos direitos trabalhistas e as rápidas transformações impostas ao mundo do trabalho, fazem o trabalhador perder sua identidade. Tudo parece se tornar supérfluo, substituível, inclusive o trabalhador.

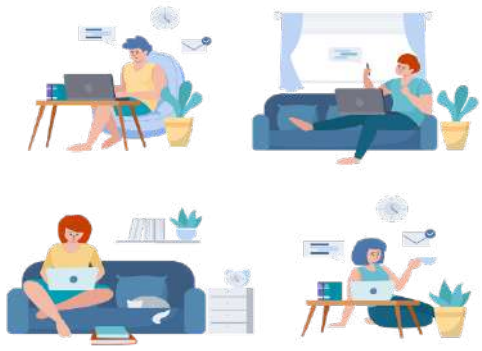
Com o processo de terceirização, a subjetividade do trabalhador passa a não lhe pertencer mais, sendo deslocada para o empregador, que a utiliza, absorvendo não só o trabalho bruto, mas também os desejos e os medos daquele funcionário. Os avanços tecnológicos que surgiram, ao invés de libertar o homem de seu trabalho repetitivo, acaba por penalizá-lo pelo medo de ficar desempregado. E é preciso pensar nisso; pensar no futuro do trabalho é pensar também no futuro das classes e das desigualdades que ele impõe. Um outro mundo é possível.

3.3 Trabalho: conquistas e desafios

A pandemia do novo coronavírus acelerou diversas mudanças em diferentes áreas trabalhistas, ampliando a utilização da tecnologia para a execução das atividades laborais em ambientes remotos. No mercado de

trabalho ocorreu uma ampliação do trabalho remoto por meio da possibilidade de utilização de funcionalidades tais como reuniões e colaboração remotas.

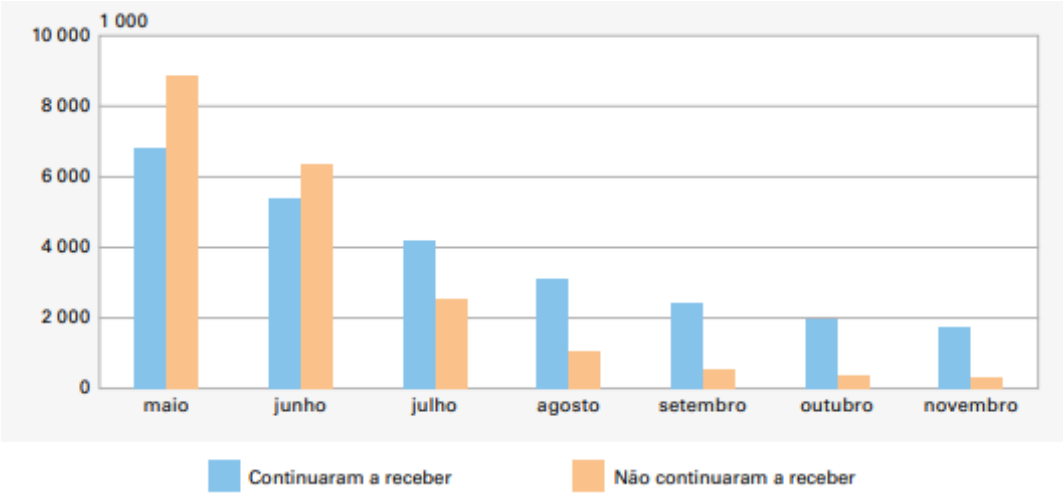
De acordo com Góes (2020), esse fenômeno é mundial, havendo uma forte associação positiva entre a renda do país e a presença do trabalho remoto. Mas isso não quer dizer muita coisa aqui no Brasil, já que grande parte da população ainda não se encontra totalmente inserida no campo digital e ao ser analisada pelo recorte de gênero e raça, os dados acabam reduzindo.



Disponível em: www.pixabay.com/Acesso em 23 abr. 2023

Nesse contexto, de acordo com a PNAD COVID-19, em maio de 2020 10,3% de pessoas ocupadas trabalharam de forma remota, sendo a maior proporção dentre as que possuíam o nível superior completo (31,1%). Nos demais níveis de instrução, a presença dessa modalidade de trabalho foi inferior, sendo de 6,0% entre os que possuíam do ensino médio completo ao superior incompleto, em torno de 1% na categoria até o ensino médio incompleto e em torno de 0,5% na categoria sem instrução ao fundamental incompleto. Houve também maior participação de mulheres (12,9%) do que de homens (8,4%), o que pode estar associado aos maiores níveis de escolaridade e às atividades econômicas onde atuavam.

Número de pessoas ocupadas afastadas pelo distanciamento social
que continuaram ou não a receber os rendimentos - Brasil - 2020



Fonte: IBGE, Pesquisa Nacional por Amostra de Domicílios COVID-19.

Nota: Dados referentes aos meses de maio a novembro de 2020.

São sinalizados por esses dados que as ocupações de profissionais acadêmicos são mais passíveis de serem exercidas de forma remota, ou seja, as que exigem maior qualificação, recortando aí uma parcela da população que não ocupa esses espaços por questões que vão da etnia à escolaridade.

Segundo dados do IBGE, ao contrário do observado para o afastamento por distanciamento social, a participação entre as pessoas brancas (14,1%) foi mais do que o dobro das pessoas pretas ou pardas (6,8%). Por grupos de idade, as pessoas entre 30 e 49 anos foram as que mais adotaram essa forma de trabalho (11,4%), enquanto nas demais faixas a participação foi inferior a 10%. Veja a tabela abaixo:

Proporção de pessoas ocupadas de forma remota, segundo características selecionadas - Brasil - 2020

Características selecionadas	Proporção de pessoas ocupadas de forma remota (%)						
	maio	jun.	jul.	ago.	set.	out.	nov.
Brasil	**10,3**	**10,4**	**10,3**	**10,2**	**9,7**	**9,0**	**8,7**
Sexo							
Homem	8,4	8,1	7,8	7,5	7,1	6,7	6,3
Mulher	12,9	13,6	13,7	14,0	13,4	12,4	12,0
Cor ou raça (1)							
Branca	14,1	14,3	14,3	14,0	13,4	12,4	11,9
Preta ou parda	6,8	6,8	6,7	6,7	6,3	5,9	5,6
Grupos de idade							
14 a 29 anos	9,1	9,1	9,0	8,6	8,1	7,3	7,1
30 a 49 anos	11,4	11,4	11,2	11,1	10,7	9,9	9,5
50 a 59 anos	9,1	9,4	9,5	9,5	9,1	8,7	8,3
60 anos ou mais	9,9	10,4	10,5	10,7	10,1	9,5	9,1
Nível de instrução							
Sem instrução ao fundamental incompleto	0,5	0,3	0,4	0,5	0,3	0,3	0,3
Fundamental completo ao médio incompleto	1,3	1,2	1,1	1,1	1,0	0,9	0,9
Médio completo ao superior incompleto	6,0	5,9	5,9	5,5	5,0	4,6	4,4
Superior completo ou pós-graduação	31,1	31,9	31,1	31,0	30,2	28,3	27,1

Fonte: IBGE, Pesquisa Nacional por Amostra de Domicílios COVID-19.

Nota: Dados referentes aos meses de maio a novembro de 2020.

Não estão apresentados os resultados para cor ou raça amarela, indígena e pessoas sem declaração de cor ou raça.

Góes (2021) mostra, a partir das informações da PNAD COVID-19, que mais de 60% das pessoas ocupadas em trabalho remoto estavam no setor privado. Em maio, correspondiam a 68,1% do total, chegando em novembro a 61,1%. Eram 4,5 milhões no setor privado e 2,9 milhões no setor público. Além disso, o autor ressalta a força do setor formal nesse tipo de trabalho, com participações relativas em torno de 84% durante todo o período da pesquisa, o que representa cerca de 6,2 milhões de trabalhadores.

Tais dados revelam a precarização do trabalho que já existia antes da pandemia e que foi evidenciada com ela, empurrando uma grande parcela da população ou ao desemprego ou ao trabalho informal. Causa muito comum para a escolha do caminho da informalidade é a falta de preparação para encarar o mercado de trabalho e a fuga do desemprego. Quantas pessoas não passaram a fazer doces, por exemplo? Quantas pessoas não começaram a trabalhar com carros de aplicativo?

Essas informações ao serem cruzadas com os estados do país, revelam disparidades que precisam ser apresentadas. Em 2022, pouco mais de um quarto (27,4%) dos jovens piauienses de 15 a 29 anos encontravam-se sem

estudar e sem uma ocupação no mercado de trabalho. Tal indicador foi superior ao registrado para o país, que foi de 25,8%, mas inferior ao observado para a região Nordeste, que atingiu 33%.

No Piauí, há cerca de 220 mil jovens nessa situação e no país esse quantitativo chegou a cerca de 12,7 milhões de pessoas. Em termos de faixa etária no Piauí, estavam sem estudar nem trabalhar: 7,6% dos jovens de 15 a 17 anos; 31,6% dos jovens de 18 a 24 anos e 38,2% dos jovens de 25 a 29 anos. Segundo dados do IBGE, os motivos informados pelos jovens por não conseguirem trabalho, foram o de não haver trabalho na localidade em que moravam e o de ter que cuidar dos afazeres domésticos, dos filhos ou de outros parentes.

Em 2022, a média anual de trabalhadores sem carteira de trabalho assinada atingiu 12,9 milhões. O número é recorde para o indicador desde o início da série histórica da Pesquisa Nacional por Amostra de Domicílio Contínua (PNAD), em 2012. O número de pessoas nessa situação aumentou 14,9% em relação a 2021, quando havia 11,2 milhões de trabalhadores sem carteira assinada.

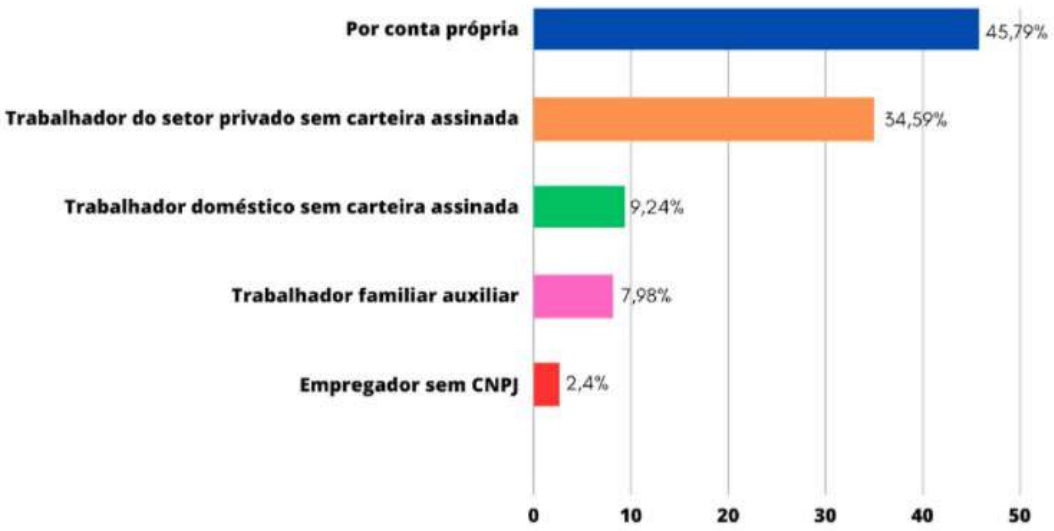
Os trabalhadores por conta própria - formais ou informais - somaram 25,5 milhões no ano, alta de 2,6% em relação ao ano anterior. O estudo divide os trabalhadores informais em quatro categorias: os informais de subsistência, os informais com potencial produtivo, os informais por opção e os formais frágeis.

Os informais de subsistência (60,5%), inclui os profissionais de baixa ou nenhuma qualificação e que oferecem serviços de demanda instável, conhecidos como ‘bicos’. Os informais com potencial produtivo (16,1%) representam os trabalhadores que não são formalizados por conta dos custos implicados ou pela falta de oportunidades.

Os informais por opção (2,3%) são aqueles que têm condições de se formalizarem, mas pretendem se manter dessa forma para ampliar seu orçamento. Os profissionais classificados como formais frágeis (21,1%) têm CNPJ ou carteira de trabalho assinada, mas com contratos intermitentes, redução dos direitos formais e ameaça de voltar à informalidade total.

A pesquisa também traçou o perfil do trabalhador informal brasileiro de subsistência. Ele é homem, jovem, preto e de baixa escolaridade. Cerca de 75% têm o ensino fundamental incompleto ou inferior. Na faixa etária de 14 a 17 anos, o grupo representa mais de 80% e nas idades de 18 a 24 anos, os informais de subsistência são 64% do total. Na análise por regiões, a presença desse grupo é especialmente expressiva nas regiões Norte (49%) e Nordeste (45,5%). A maioria deles trabalha com serviços ligados a comércio, reparação de veículos e construção.

O trabalho informal surge aqui como uma realidade de tempos pós-pandêmicos, já que a pandemia ao afetar também o processo educacional no Estado acabou fragilizando a permanência desses jovens nas escolas e assim ampliou as estatísticas de evasão escolar e de trabalho informal. No Piauí, os dados não repetem a estatística nacional, mas seguem em evidência.



Ocupação na informalidade do Piauí em 2022 (em %)

O maior número de pessoas ocupadas na informalidade no estado do Piauí é constituído por trabalhadores por conta própria sem registro no CNPJ, com 327 mil pessoas (45,79%). Na sequência, vem as pessoas ocupadas no setor privado sem carteira assinada, com um contingente de 247 mil pessoas

(34,59%); trabalhadores domésticos sem carteira assinada, com 66 mil pessoas ocupadas (9,24%); trabalhador familiar auxiliar, com 57 mil pessoas (7,98%); e empregadores sem CNPJ, com 17 mil pessoas (2,4%).

Vale ressaltar que a informalidade não é a primeira opção da maioria. A falta de emprego e de capacitação para entrar no mercado são os principais fatores que levam as pessoas a recorrerem ao trabalho informal. Há quem defenda a ocupação informal justificando a decisão pela liberdade que adquiriu de definir suas próprias horas e escolher seus próprios projetos. Apesar de não serem acobertados pelo seguro-desemprego e INSS, essa defesa é a marca dos tempos neoliberais que estamos vivendo.

4. REFERÊNCIAS BIBLIOGRÁFICAS:

ALVES, G. **O novo (e precário) mundo do trabalho:** reestruturação produtiva e crise do sindicalismo. São Paulo: Boitempo, 2000.

BRASIL. Ministério da Educação. Decreto nº 9.057, de 25 de maio de 2017. Regulamenta o art. 80 da Lei nº 9.394, de 20 de dezembro de 1996, que

estabelece as diretrizes e bases da educação nacional. **Diário Oficial da União**, Brasília, 25 maio 2017.

BRASIL. Ministério da Educação. Portaria nº 544/2020. Dispõe sobre a substituição das aulas presenciais por aulas em meios digitais, enquanto durar a situação de pandemia do novo coronavírus - Covid-19, e revoga as Portarias MEC nº 343, de 17 de março de 2020, nº 345, de 19 de março de 2020, e nº 473, de 12 de maio de 2020. **Diário Oficial da União**, Brasília, 16 jun. 2020.

CHAUÍ, M. **Brasil:** mito fundador e sociedade autoritária. São Paulo: Fundação Perseu Abramo, 2001.

DA MATTA, R. **Carnavais, malandros e heróis:** para uma sociologia do dilema brasileiro. Rio de Janeiro: Rocco, 1997.

_____. **O que faz o brasil, Brasil?** Rio de Janeiro: Rocco, 1993.

DECCA, E. Cidadão, mostre-me a identidade! *In*: **Caderno Cedes**, Campinas, V.22, nº 58, pp. 7-20, 2002.

GÓES, G.; MARTINS, F.; NASCIMENTO, J. **Potencial de teletrabalho na pandemia: um retrato no Brasil e no mundo**. Carta de Conjuntura, Rio de Janeiro: Instituto de Pesquisa Econômica Aplicada - Ipea, n. 47, p. 1-10, 2020. Disponível em: www.ipea.gov.br/Acesso em: abr. 2023.

HOLANDA, S. **Raízes do Brasil**. São Paulo: Companhia das Letras, 1995.

IBGE. [Pesquisa Nacional por Amostra de Domicílios Contínua - PNAD]. Acesso em: abr. 2023.

LARAIA, R. **Cultura:** um conceito antropológico. Rio de Janeiro: Jorge Zahar, 1992.

LEMOS, A. **Isso (não) é muito Black Mirror:** passado, presente e futuro das tecnologias de comunicação e informação. Salvador: EDUFBA, 2018.

PRETI, O. **Autonomia do Aprendiz na Educação a Distância**: significados e dimensões. Cuiabá: UFMT/NEAD, 2005.

RIBEIRO, D. **O povo brasileiro:** a formação e o sentido do Brasil. São Paulo: Companhia das Letras, 1995.

SANTOS, J. **O que é cultura.** São Paulo: Brasiliense, 1992.

Escola Estadual com muito orgulho

SEDUC TEC

Canal Educação
PROGRAMA DE MEDIAÇÃO TECNOLÓGICA

SECRETARIA DA **EDUCAÇÃO** - SEDUC

GOVERNO DO PIAUÍ
AQUI TEM **TRABALHO**.
AQUI TEM **FUTURO**.