Persistência de Arquivos: texto, binário

QXD0099 - Desenvolvimento de Software para Persistência

Universidade Federal do Ceará - *Campus* Quixadá

Prof. Francisco Victor da Silva Pinheiro
victorpinheiro@ufc.br

Slides baseados no material do Professor Régis Magalhães

Agenda

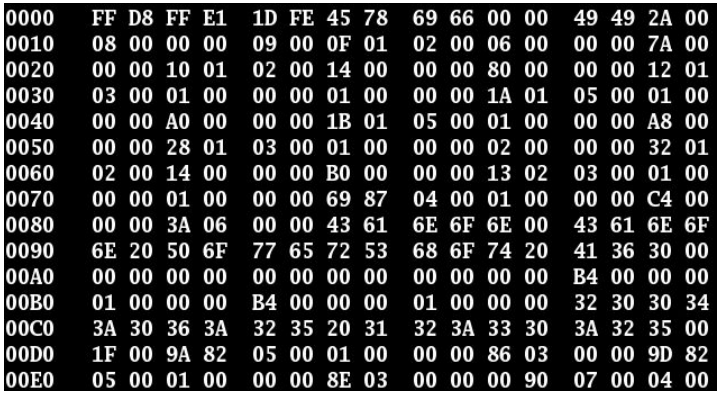
- Tipos de Arquivos
- Diferenças entre Arquivos Texto e Binário
- Modos de abertura de arquivos
- Fluxos
- Gerenciamento com with
- Leitura de arquivo
- Codificações Comuns
- Traduzindo de determinada codificação para unicode
- Conversor de codificação
- Funções Úteis
- Tratamento de Erros
- BOM - Byte Order Mark
- Código Python para lidar com BOM
- Detecção manual da BOM (sem utf-8-sig)
- Lendo uma linha inteira
- Lendo o arquivo inteiro
- Lendo Strings do Teclado
- Escrita em arquivo
- Escrita em arquivo utilizando print
- Lendo Strings do Teclado e salvando em um arquivo

Tipos de Arquivos

- Texto
  - Texto plano
  - Propriedades
  - CSV
  - XML
  - JSON
  - Código fonte

Tipos de Arquivos

- Binário
  - Imagem
  - Vídeo
  - Áudio
  - Arquivo compactado
  - Código compilado: Executável / Bytecode.
  - PDF



0000	FF	D8	FF	E1	1D	FE	45	78	69	66	00	00	49	49	2A	00
0010	08	00	00	00	09	00	0F	01	02	00	06	00	00	00	7A	00
0020	00	00	10	01	02	00	14	00	00	00	80	00	00	00	12	01
0030	03	00	01	00	00	00	01	00	00	00	1A	01	05	00	01	00
0040	00	00	A0	00	00	00	1B	01	05	00	01	00	00	00	A8	00
0050	00	00	28	01	03	00	01	00	00	00	02	00	00	00	32	01
0060	02	00	14	00	00	00	B0	00	00	00	13	02	03	00	01	00
0070	00	00	01	00	00	00	69	87	04	00	01	00	00	00	C4	00
0080	00	00	3A	06	00	00	43	61	6E	6F	6E	00	43	61	6E	6F
0090	6E	20	50	6F	77	65	72	53	68	6F	74	20	41	36	30	00
00A0	00	00	00	00	00	00	00	00	00	00	00	00	B4	00	00	00
00B0	01	00	00	00	B4	00	00	00	01	00	00	00	32	30	30	34
00C0	3A	30	36	3A	32	35	20	31	32	3A	33	30	3A	32	35	00
00D0	1F	00	9A	82	05	00	01	00	00	00	86	03	00	00	9D	82
00E0	05	00	01	00	00	00	8E	03	00	00	00	90	07	00	04	00

Diferenças entre Arquivos Texto e Binário

- **Texto:**
  - Armazena caracteres (letras, números, símbolos) codificados (ex: UTF-8).
  - Pode ser aberto diretamente com editores de texto.
    - Ex: .txt, .csv, .json

- **Binário:**
  - Armazena bytes brutos, sem codificação legível.
    - Ex: .jpg, .pdf, .exe

Arquivo Texto | Arquivo Binário

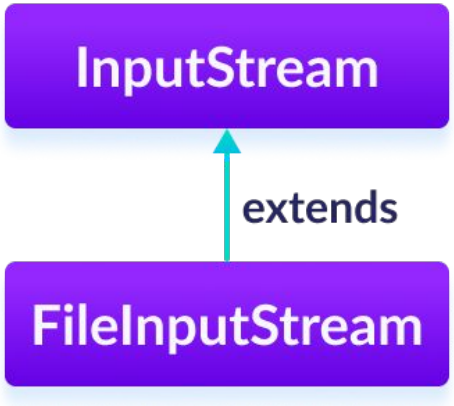
Modos de abertura de arquivos

- 'r' leitura (erro se não existir)
- 'w' escrita (sobrescreve o arquivo)
- 'a' append (adiciona no final)
- 'x' criação exclusiva (erro se existir)
- 'b' modo binário
- 't' modo texto (padrão)
- '+' leitura e escrita

- **Combinações:**
  - 'rb' – leitura binária
  - 'wb' – escrita binária
  - 'a+' – leitura + escrita, adicionando ao final

Fluxos

- **Fluxo de Entrada**
  - [Python] - file-like objects ou stream objects - Ler bytes.
  - [Java] - InputStream
- **Fluxo de Saída**
  - [Python] - file-like objects ou stream objects – Escrever bytes.
  - [Java] - OutputStream
- **Servem para operações sobre:**
  - Arquivos.
  - Conexão remota via Socket.
  - Entrada e saída padrão (teclado e console).



Gerenciamento com with

- O with em Python é chamado de context manager (gerenciador de contexto).
- Ele serve para abrir e garantir o fechamento automático de recursos, como arquivos, conexões de banco de dados, sockets, etc.
- Usando with (forma segura)

```
with open('arquivo.txt', 'rb') as file:
    b = file.read(1)
```

Gerenciamento com with

- **O que acontece:**
  - O Python abre o arquivo **dados.txt** no modo leitura ('r') e o associa à variável f.
  - Assim que o bloco **with** termina (quando o código sai da identação), o Python automaticamente fecha o arquivo, mesmo que ocorra um erro dentro do bloco.
  - Isso evita vazamento de recursos e comportamentos indesejados.

Gerenciamento com with

- Sem with (precisa fechar manualmente)

```
f = open('dados.txt', 'r')
print(f.read())
f.close()
```

- **O que acontece:**
  - O arquivo é aberto normalmente.
  - Mas você é responsável por fechar o arquivo manualmente com f.close().
  - Se ocorrer um erro entre o open e o close, o f.close() talvez nunca seja executado, deixando o arquivo aberto na memória do sistema.

Leitura de arquivo

```
# Exemplo de leitura de arquivo em Python

# Abre o arquivo "arquivo.txt" para leitura em modo binário
with open('arquivo.txt', 'rb') as file: # 'rb' significa leitura em modo binário
    b = file.read(1) # Lê o primeiro byte do arquivo

# Exibe o valor do byte lido
print(b)
```

Codificações Comuns

- **'utf-8' (mais comum)**
  - é um padrão de codificação de caracteres que permite representar todos os caracteres Unicode. É a codificação de caracteres mais utilizada na web e é o padrão para plataformas baseadas em *nix.

- **'utf-16' (Windows, Excel)**
  - é um sistema de codificação de caracteres que usa unidades de 16 bits para representar pontos de código Unicode.

- **'latin-1' (ISO-8859-1, América Latina)**
  - é um padrão de codificação de caracteres do alfabeto latino. Foi desenvolvido pela ISO e é a base de outros mapeamentos, como o Windows-1252.

Traduzindo de determinada codificação para unicode

```
# Exemplo de leitura de um arquivo com codificação específica e conversão para Unicode

# Abre o arquivo "arquivo.txt" para leitura com a codificação UTF-8 (ou outra se necessário)
with open('arquivo.txt', 'r', encoding='utf-8') as file:  # Aqui 'utf-8' pode ser substituído por 'iso-8859-1' se necessário
    c = file.read(1)  # Lê o primeiro caractere do arquivo

# Exibe o caractere lido
print(c)
```

- **open('arquivo.txt', 'r', encoding='utf-8')**: Em Python, ao abrir arquivos de texto, você pode especificar a codificação com o argumento **encoding**. Aqui, usamos **'utf-8'**, mas pode ser substituído por **'iso-8859-1'** ou qualquer outra codificação desejada.
- **file.read(1)**: Lê o primeiro caractere do arquivo. Ao usar uma codificação, os dados são automaticamente convertidos para Unicode internamente em Python.
- Python já trata a manipulação de **char** (caracteres) de maneira nativa, diferente do Java, que precisa de classes específicas como **InputStreamReader**.

Conversor de codificação

```
with open('original.txt', 'r', encoding='latin-1') as origem:
    texto = origem.read()

with open('convertido.txt', 'w', encoding='utf-8') as destino:
    destino.write(texto)
```

- Esse código lê um arquivo de texto com codificação latin-1 (ISO-8859-1) e cria um novo arquivo com o mesmo conteúdo, mas salvo com codificação utf-8.
- Em outras palavras: é uma conversão de codificação de caracteres de um arquivo para outro.

Funções Úteis

- f.read(size)
  - Lê até 'size' caracteres
- f.readline()
  - Lê uma linha
- f.readlines()
  - Retorna lista de linhas
- f.seek(offset)
  - Move o cursor
- f.tell()
  - Mostra posição do cursor

Tratamento de Erros

- Boa prática: Sempre tratar exceções ao lidar com arquivos.

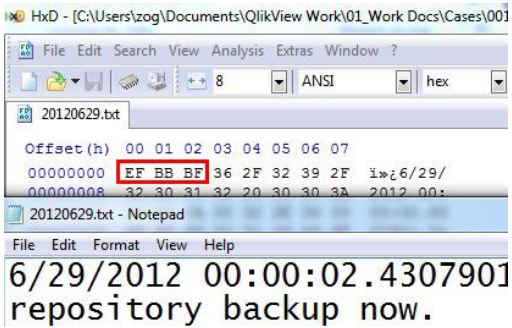
```
try:
    with open('arquivo_inexistente.txt', 'r') as f:
        conteudo = f.read()
except FileNotFoundError:
    print("Arquivo não encontrado.")
except PermissionError:
    print("Sem permissão.")
```

BOM - Byte Order Mark

- É um uso particular do caractere Unicode especial, U FEFF ZERO WIDTH NO-BREAK SPACE, cuja aparência como um número mágico no início de um fluxo de texto pode sinalizar várias coisas para um programa que lê o texto:
  - A ordem de bytes, ou endianness, do fluxo de texto nos casos de codificações de 16 e 32 bits;
  - O fato de a codificação do fluxo de texto ser Unicode, com alto nível de confiança;
  - Qual codificação de caracteres Unicode é usada.

- A representação UTF-8 da BOM é a sequência de bytes (hexadecimal) EF BB BF.

- O Padrão Unicode permite a BOM em UTF-8, mas não exige nem recomenda seu uso.



Código Python para lidar com BOM

```
# Abrindo o arquivo que pode conter uma BOM, utilizando 'utf-8-sig' para lidar com a BOM
automaticamente
with open('arquivo.txt', 'r', encoding='utf-8-sig') as file:
    conteudo = file.read()

# Exibe o conteúdo do arquivo, sem a BOM
print(conteudo)
```

- **encoding='utf-8-sig'**: Ao abrir um arquivo com essa codificação, o Python remove automaticamente o BOM se ele estiver presente. Isso é útil ao trabalhar com arquivos UTF-8 que contêm a BOM.
- **file.read()**: Lê o conteúdo do arquivo como texto, ignorando a BOM.

Detecção manual da BOM (sem utf-8-sig)

```python
# Abrindo o arquivo no modo binário para verificar os primeiros bytes manualmente
with open('arquivoBom.txt', 'rb') as file:
    primeiro_bytes = file.read(3)  # Lê os primeiros 3 bytes

    # Verifica se a BOM está presente
    if primeiro_bytes == b'\xef\xbb\xbf':
        print("BOM detectada no arquivo!")
        # Ler o restante do arquivo, agora sem a BOM
        conteudo = file.read().decode('utf-8')
    else:
        # Se não houver BOM, volta ao início e lê o arquivo inteiro
        file.seek(0)
        conteudo = file.read().decode('utf-8')

# Exibe o conteúdo do arquivo
print(conteudo)
```

- O código verifica se o arquivo contém uma BOM UTF-8 no início.
- Se a BOM for encontrada, ela é ignorada, e o restante do arquivo é lido e decodificado.
- Se a BOM não estiver presente, o arquivo é lido desde o início.
- O conteúdo do arquivo é então exibido, sem a BOM.

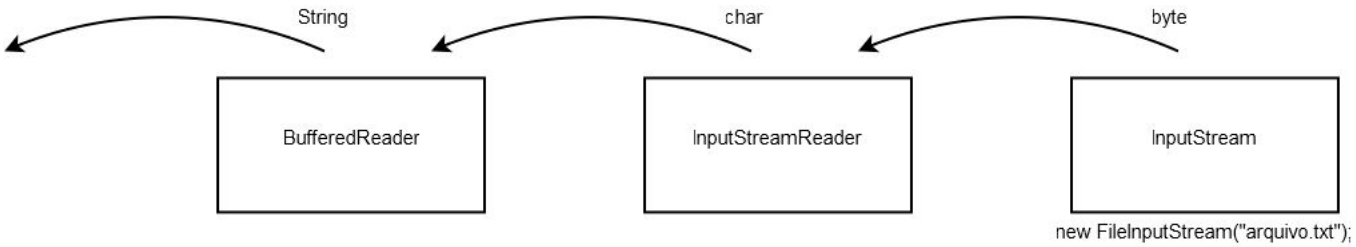
Lendo uma linha inteira

```
# Abrindo o arquivo para leitura de uma linha
with open('arquivo.txt', 'r', encoding='utf-8') as file:
    s = file.readline()   # Lê uma linha do arquivo

# Exibe a linha lida
print(s)
```

- **open('arquivo.txt', 'r', encoding='utf-8')**: Abre o arquivo arquivo.txt em modo de leitura ('r'). O parâmetro encoding='utf-8' garante que o arquivo seja lido corretamente como texto UTF-8.
- **file.readline()**: Lê uma linha inteira do arquivo. A função readline() lê até encontrar um caractere de nova linha (\n) ou até o final do arquivo.
- **print(s)**: Exibe o conteúdo da linha lida.

Lendo uma linha inteira

Leitura do Reader por pedaços, usando o Buffer, para evitar muitas chamadas ao SO.



Padrão de composição chamado Decorator Pattern.

Lendo o arquivo inteiro

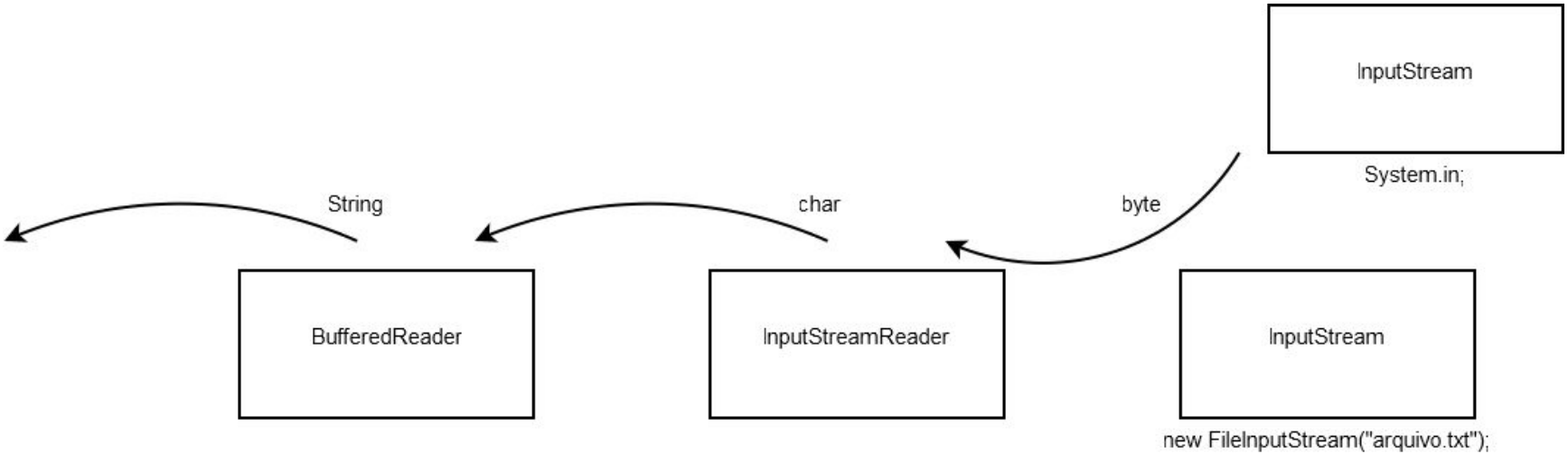
```
# Abrindo o arquivo para leitura
with open('arquivo.txt', 'r', encoding='utf-8') as file:
    # Lê a primeira linha
    s = file.readline()

    # Enquanto houver linhas para ler
    while s:
        # Imprime a linha atual
        print(s.strip())  # .strip() remove os espaços em branco no final da linha, incluindo o \n
        # Lê a próxima linha
        s = file.readline()
```

Lendo Strings do Teclado

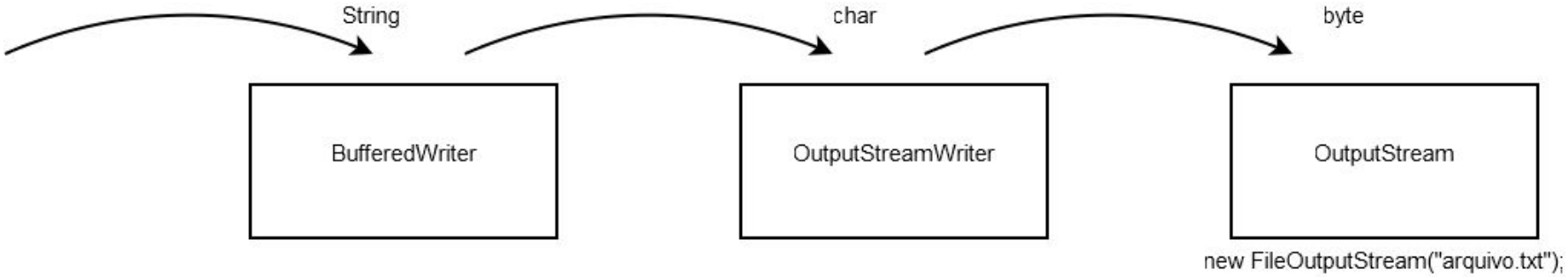
```
# Leitura da entrada do usuário (console)
import sys

# Lê a primeira linha da entrada padrão
s = sys.stdin.readline().strip()   # .strip() remove espaços em branco e nova linha

# Enquanto houver algo na entrada
while s:
    # Exibe a linha lida
    print(s)
    # Lê a próxima linha da entrada padrão
    s = sys.stdin.readline().strip()
```

- **sys.stdin.readline()**: Equivalente ao br.readLine() no Java, lê uma linha da entrada padrão (no caso, o console). O .strip() remove qualquer nova linha ou espaços em branco no final.
- **while s:**: O loop continua até que não haja mais nada a ser lido da entrada, similar ao while (s != null) em Java.
- **print(s)**: Exibe a linha lida no console.

Lendo Strings do Teclado



Escrita em arquivo



Escrita em arquivo

```
# Abrindo o arquivo para escrita
with open('saida.txt', 'w', encoding='utf-8') as file:
    # Escreve "Java" no arquivo
    file.write("Java\n")
```

- **open('saida.txt', 'w', encoding='utf-8')**: Abre o arquivo saida.txt para escrita ('w'), criando o arquivo se ele não existir. O argumento encoding='utf-8' garante que o arquivo seja escrito usando a codificação UTF-8.
- **file.write("Java\n")**: Escreve a string "Java" no arquivo, seguida por uma nova linha (\n), equivalente ao bw.newLine() em Java.
- **with**: O uso de with garante que o arquivo seja fechado automaticamente após a escrita, eliminando a necessidade de bw.close().

Escrita em arquivo utilizando print

```
# Abre o arquivo "saida.txt" para escrita
with open('saida.txt', 'w', encoding='utf-8') as file:
    # Usa a função print para escrever "Java" no arquivo com uma nova linha automaticamente
    print("Java", file=file)
```

- **io.StringIO()**: Cria um buffer em memória que simula um arquivo de texto. Ele age como um stream onde você pode "imprimir" dados.
- **print("Java", file=buffer)**: Escreve a string "Java" no buffer, da mesma forma que faria em um arquivo ou com PrintStream no Java.
- **file.write(buffer.getvalue())**: Escreve o conteúdo do buffer no arquivo real.
- **buffer.close()**: Fecha o buffer, embora não seja estritamente necessário para StringIO.

Lendo Strings do Teclado e salvando em um arquivo

```
# Abrindo o arquivo para escrita
with open('arquivo.txt', 'w', encoding='utf-8') as file:
    # Lendo strings do teclado
    while True:
        try:
            # Lê uma linha do teclado
            line = input()
            # Escreve a linha no arquivo
            print(line, file=file)
        except EOFError:
            # Termina o loop quando não houver mais
entrada (Ctrl+D no Linux/macOS ou Ctrl+Z no Windows)
            break
```

- **with open('arquivo.txt', 'w', encoding='utf-8')**: Abre o arquivo arquivo.txt em modo de escrita, usando a codificação UTF-8.
- **input()**: Lê uma linha de entrada do teclado.
- **print(line, file=file)**: Escreve a linha no arquivo. Isso é equivalente ao PrintStream.println() do Java.
- **EOFError**: Interrompe o loop quando o usuário envia um sinal de fim de entrada (Ctrl+D em Linux/macOS ou Ctrl+Z no Windows).

Exercício 1 – Lista de Tarefas

- **Crie um programa que:**
  - Pergunte ao usuário se ele deseja (1) adicionar uma tarefa ou (2) visualizar as tarefas.
  - Se a opção for adicionar, o programa deve pedir a descrição da tarefa e salvá-la no arquivo tarefas.txt.
  - Se a opção for visualizar, deve exibir todas as tarefas cadastradas, uma por linha.

- **Requisitos:**
  - Use with open(...).
  - Use o modo 'a' para adicionar tarefas.
  - Use o modo 'r' para visualizar.
  - O arquivo deve ser salvo com codificação 'utf-8'.

- **Dica:** Trate o erro caso o arquivo ainda não exista ao tentar visualizar as tarefas.

Exercício 2 – Gerenciador de Tarefas com Remoção e Busca

- **Crie um programa que permita ao usuário:**
  - Adicionar uma nova tarefa, informando a descrição e a prioridade (baixa, média ou alta).
  - Visualizar todas as tarefas.
  - Buscar tarefas por palavra-chave.
  - Remover uma tarefa pelo número.
  - Sair do programa.

- **Requisitos:**
  - Use with open(...) sempre.
  - Codificação 'utf-8'.
  - Utilize try/except para tratar erros como FileNotFoundError.
  - Para remover, reescreva o arquivo sem a tarefa excluída.

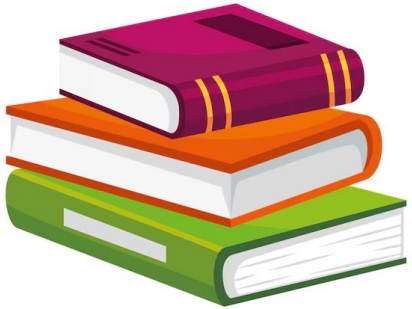
- **Dica:**
  - Use split(" | ") para separar os campos ao ler o arquivo.
  - Para remover, carregue as tarefas em uma lista, exclua o item e salve tudo de novo.

Bibliografia Básica

- SADALAGE, P. J. E FOWLER, M. NoSQL Essencial. Editora Novatec, São Paulo, 2013.
- REDMOND, E.; WILSON, J. R. Seven Databases in Seven Weeks: A Guide to Modern Databases and the NoSQL Movement. 1ª edição, 2012. The Pragmatic Programmers.
- ULLMAN, J.D.; WIDOW, J. First Course in Database Systems. 3a edição, 2007. Prentice Hall.
- HAMBRICK, G. et al. Persistence in the Enterprise: A Guide to Persistence Technologies; 1ª edição, 2008. IBM Press.
- ELMASRI, R.; NAVATHE, S. B. Sistemas de banco de dados. 4ª edicao, 2009. Pearson/Addison-Wesley.

Bibliografia Complementar

- WHITE, Tom. Hadoop: the definitive guide. California: O´Reilly, 2009. xix, 501 p. ISBN 9780596521974 (broch.).
- AMBLER, S.W., SADALAGE, P.J. Refactoring Databases: Evolutionary Database Design. 1a edição, 2011. Addison Wesley.
- SILBERSCHATZ, A.; SUDARSHAN, S. Sistema de banco de dados. 2006. Campus.
- LYNN, B. Use a cabeça! SQL. 1ª edição, 2008. ALTA BOOKS.
- SMITH, Ben. JSON básico: conheça o formato de dados preferido da web. São Paulo: Novatec, 2015. 400 p. ISBN 9788575224366 (broch.).
- HITZLER, P., KRÖTZSCH, M., and RUDOLPH, S. (2009). Foundations of Semantic Web Technologies. Chapman & Hall/CRC.
- ANTONIOU, G. and HARMELEN, F. (2008). A Semantic Web Primer. Second Edition,Cambridge, MIT Press, Massachusetts.
- HEATH, T. and BIZER, C. (2011). Linked Data: Evolving the Web into a Global Data Space. Morgan & Claypool, 1st edition.



Obrigado!

Dúvidas?

Universidade Federal do Ceará - *Campus* Quixadá

Prof. Francisco Victor da Silva Pinheiro
victorpinheiro@ufc.br