Projeto de Treinamento em Laparoscopia com Realidade Virtual

Introdução

Este projeto tem como objetivo auxiliar o desenvolvimento de um MVP (Minimum Viable Product) para aprimorar o treinamento de residentes em cirurgia laparoscópica utilizando tecnologias de Realidade Virtual (VR). A proposta busca solucionar desafios enfrentados pelo LEPIC/USP, proporcionando uma plataforma de treinamento acessível e eficiente.

Texto alternativo

Metodologia

Para alcançar os objetivos propostos, seguimos os seguintes passos:

1. **Conexão com o Banco de Dados Oracle:** Utilizamos a biblioteca `oracledb` para estabelecer a conexão e manipular os dados existentes nas tabelas `usuario`, `fase` e `executa`.
2. **Estruturas de Dados:** Selecionamos diferentes estruturas de dados (listas, dicionários, DataFrames do Pandas e arrays do NumPy) para armazenar resultados intermediários e compará-las em termos de eficiência e facilidade de uso.
3. **Desenvolvimento de Funções em Python:** Criamos funções para processar os dados, calcular estatísticas de desempenho dos residentes e implementar a lógica da solução no contexto do treinamento em VR.
4. **Análise dos Resultados:** Avaliamos os dados obtidos e as funcionalidades implementadas, destacando os benefícios e limitações de cada abordagem.

Implementação

Preparação do Ambiente

Instalamos as bibliotecas necessárias:

In [10]:
```
!pip install oracledb pandas numpy
```

```
Requirement already satisfied: oracledb in c:\users\guilherme\anaconda3\lib\site-packages (2.4.1)
Requirement already satisfied: pandas in c:\users\guilherme\anaconda3\lib\site-packages (2.1.4)
Requirement already satisfied: numpy in c:\users\guilherme\anaconda3\lib\site-packages (1.26.4)
Requirement already satisfied: cryptography>=3.2.1 in c:\users\guilherme\anaconda3\lib\site-packages (from oracledb) (42.0.2)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\guilherme\anaconda3\lib\site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\guilherme\anaconda3\lib\site-packages (from pandas) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in c:\users\guilherme\anaconda3\lib\site-packages (from pandas) (2023.3)
Requirement already satisfied: cffi>=1.12 in c:\users\guilherme\anaconda3\lib\site-packages (from cryptography>=3.2.1->oracledb) (1.16.0)
Requirement already satisfied: six>=1.5 in c:\users\guilherme\anaconda3\lib\site-packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
Requirement already satisfied: pycparser in c:\users\guilherme\anaconda3\lib\site-packages (from cffi>=1.12->cryptography>=3.2.1->oracledb) (2.21)
```

Importação das Bibliotecas

In [12]:
```
import oracledb
import pandas as pd
import numpy as np
import time
import timeit
```

Conexão com o Banco de Dados Oracle

In [14]:
```
# Conectar em modo Thin (sem necessidade do Instant Client)
connection = oracledb.connect(
    user='RM98604',
    password='300402',
    dsn='oracle.fiap.com.br:1521/orcl'
)
print("Conexão estabelecida com sucesso!")
```

```
Conexão estabelecida com sucesso!
```

Execução de Consultas e Recuperação de Dados

Recuperar Dados dos Usuários

In [17]:
```
# Criar um DataFrame com os dados dos usuários
query_usuarios = "SELECT * FROM usuario"
df_usuarios = pd.read_sql(query_usuarios, con=connection)
df_usuarios
```

```
C:\Users\Guilherme\AppData\Local\Temp\ipykernel_2008\2841585385.py:3: UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or database string URI or sqlite3 DBAPI2 connection. O
ther DBAPI2 objects are not tested. Please consider using SQLAlchemy.
  df_usuarios = pd.read_sql(query_usuarios, con=connection)
```

Out[17]:

	ID_USUARIO	NM_USUARIO	EMAIL	SENHA	TP_USUARIO	GERIDO_POR
0	1	Guilherme Nobre Bernardo	rm98604@fiap.com.br	34eeb309cc726bdb79f674b0e0912e5eae468045748230...	0	1
1	2	Bobson da Sílva Santos	son.bob@gmail.com.br	aec04198e7d486b7a2823d7518e9804e8fa0232eac95ab...	1	1
2	3	Caíque Walter Silva	rm550693@fiap.com.br	50a4f9cc73f428a42116ce6790c30ee8f24b1aaefabc32...	2	2
3	4	Gabriela Marsiglia	rm551237@fiap.com.br	d8a753c6533a75203d931626917fc1d7bffbe8c361ef52...	2	2
4	5	Matheus José de Lima Costa	rm551157@fiap.com.br	38a0963a6364b09ad867aa9a66c6d009673c21e1820154...	2	2
5	21	Roberson	roberson123@garai	123	1	1
6	43	senha123	Nome Teste	email@teste.com	1	1
7	22	Cleison	cleison@garai	123	1	1
8	41	teste	teste@teste	123	1	1
9	42	mat	mat	123	1	1

Recuperar Dados das Fases

In [19]:
```
# Criar um DataFrame com os dados das fases
query_fases = "SELECT * FROM fase"
df_fases = pd.read_sql(query_fases, con=connection)
df_fases
```

```
C:\Users\Guilherme\AppData\Local\Temp\ipykernel_2008\510870044.py:3: UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or database string URI or sqlite3 DBAPI2 connection. Ot
her DBAPI2 objects are not tested. Please consider using SQLAlchemy.
  df_fases = pd.read_sql(query_fases, con=connection)
```

Out[19]:

	ID_FASE	NM_FASE	LV_DIFICULDADE	DESCRICAO
0	21	FaseTeste	2	Fase onde se opera AIDs
1	23	FASE TESTE EBOLA	3	OPERANDO EBOLA
2	24	FASE LEGAL	2	FASE MUITO LEGAL
3	29	Nível Teste	3	Descrição Teste
4	30	Nível Teste	3	Descrição Teste
5	1	Fase 01	1	Fase numero 1 do programa
6	2	Fase 02	1	Fase numero 2 do programa
7	3	Fase 03	2	Fase numero 3 do programa
8	4	Fase 04	2	Fase numero 4 do programa
9	5	Fase 05	3	Fase numero 5 do programa
10	25	Nível Teste	3	Descrição Teste
11	26	Nível Teste	3	Descrição Teste
12	22	faseTESTE	2	FASE DE AIDS
13	27	Nível Teste	3	Descrição Teste
14	28	Nível Teste	3	Descrição Teste
15	31	Test Level	3	This is a test level.

Recuperar Dados das Execuções

In [21]:
```
# Criar um DataFrame com os dados das execuções
query_executa = "SELECT * FROM executa"
df_executa = pd.read_sql(query_executa, con=connection)
df_executa
```

```
C:\Users\Guilherme\AppData\Local\Temp\ipykernel_2008\209434149.py:3: UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or database string URI or sqlite3 DBAPI2 connection. Ot
her DBAPI2 objects are not tested. Please consider using SQLAlchemy.
  df_executa = pd.read_sql(query_executa, con=connection)
```

Out[21]:

	ID_USUARIO	ID_FASE	DT_TREINAMENTO	TM_EXECUCAO	PRECISAO
0	3	1	2024-09-15 18:53:41	0 days 00:10:30.500000	0.2296
1	4	2	2024-09-15 18:57:10	0 days 00:12:15.985000	0.1654
2	5	3	2024-09-15 18:57:10	0 days 00:05:46.452000	0.9851
3	3	2	2024-09-15 18:57:10	0 days 00:09:37.458000	0.7546
4	4	3	2024-09-15 18:57:10	0 days 00:22:19.454000	1.0000
5	5	1	2024-09-15 18:57:38	0 days 00:01:59.165000	0.0000

Estruturas de Dados para Armazenamento Intermediário

Utilizando Listas de Dicionários

In [24]:
```
# Converter DataFrames em listas de dicionários
usuarios_lista = df_usuarios.to_dict('records')
fases_lista = df_fases.to_dict('records')
executa_lista = df_executa.to_dict('records')
```

Utilizando Dicionários de Dicionários

In [26]:
```
# Criar dicionários com IDs como chaves
usuarios_dict = df_usuarios.set_index('ID_USUARIO').to_dict('index')
fases_dict = df_fases.set_index('ID_FASE').to_dict('index')
executa_dict = df_executa.set_index(['ID_USUARIO', 'ID_FASE']).to_dict('index')
```

Utilizando DataFrames do Pandas

Os DataFrames já foram criados anteriormente e serão utilizados diretamente.

Utilizando Arrays do NumPy

In [30]:
```
# Converter colunas numéricas em arrays NumPy
execucao_tempo = np.array(df_executa['TM_EXECUCAO'])
precisao_array = np.array(df_executa['PRECISAO'])
```

Comparação das Estruturas

Para comparar as diferentes estruturas de dados, implementamos funções que calculam a média de precisão por residente utilizando cada uma delas. Medimos o tempo de execução de cada função para determinar qual abordagem é mais eficiente.

Função utilizando Lista de Dicionários

In [34]:
```
def media_precisao_lista(executa_lista):
    resultados = {}
    contagem = {}
    for item in executa_lista:
        id_usuario = item['ID_USUARIO']
        precisao = item['PRECISAO']
        if id_usuario in resultados:
            resultados[id_usuario] += precisao
            contagem[id_usuario] += 1
        else:
            resultados[id_usuario] = precisao
            contagem[id_usuario] = 1
    media = {k: resultados[k]/contagem[k] for k in resultados}
    return media
```

Função utilizando Dicionário de Dicionários

In [36]:
```
def media_precisao_dict(executa_dict):
    resultados = {}
    contagem = {}
    for (id_usuario, _), valores in executa_dict.items():
        precisao = valores['PRECISAO']
        if id_usuario in resultados:
            resultados[id_usuario] += precisao
            contagem[id_usuario] += 1
        else:
            resultados[id_usuario] = precisao
            contagem[id_usuario] = 1
    media = {k: resultados[k]/contagem[k] for k in resultados}
    return media
```

Função utilizando DataFrame do Pandas

In [38]:
```
def media_precisao_pandas(executa_df):
    media = executa_df.groupby('ID_USUARIO')['PRECISAO'].mean().to_dict()
    return media
```

Função utilizando Arrays do NumPy

In [40]:
```
def media_precisao_numpy(executa_df):
    ids = executa_df['ID_USUARIO'].values
    precisao = executa_df['PRECISAO'].values
    unique_ids = np.unique(ids)
    media = {}
    for uid in unique_ids:
        media[uid] = precisao[ids == uid].mean()
    return media
```

Medindo o Tempo de Execução

Utilizamos o módulo `timeit` para medir o tempo de execução de cada função.

Tempo de Execução com Lista de Dicionários

In [44]:
```
tempo_lista = timeit.timeit(lambda: media_precisao_lista(executa_lista), number=1000)
print(f"Tempo de execução com Lista de Dicionários: {tempo_lista:.6f} segundos")
```

```
Tempo de execução com Lista de Dicionários: 0.001460 segundos
```

Tempo de Execução com Dicionário de Dicionários

In [46]:
```
tempo_dict = timeit.timeit(lambda: media_precisao_dict(executa_dict), number=1000)
print(f"Tempo de execução com Dicionário de Dicionários: {tempo_dict:.6f} segundos")
```

```
Tempo de execução com Dicionário de Dicionários: 0.001218 segundos
```

Tempo de Execução com DataFrame do Pandas

In [48]:
```
tempo_pandas = timeit.timeit(lambda: media_precisao_pandas(df_executa), number=1000)
print(f"Tempo de execução com Pandas DataFrame: {tempo_pandas:.6f} segundos")
```

```
Tempo de execução com Pandas DataFrame: 0.114621 segundos
```

Tempo de Execução com NumPy Arrays

In [50]:
```
tempo_numpy = timeit.timeit(lambda: media_precisao_numpy(df_executa), number=1000)
print(f"Tempo de execução com NumPy Arrays: {tempo_numpy:.6f} segundos")
```

```
Tempo de execução com NumPy Arrays: 0.017650 segundos
```

Resultados dos Tempos de Execução

In [52]:
```
print("Comparação dos Tempos de Execução:")
print(f"Lista de Dicionários: {tempo_lista:.6f} segundos")
print(f"Dicionário de Dicionários: {tempo_dict:.6f} segundos")
print(f"Pandas DataFrame: {tempo_pandas:.6f} segundos")
print(f"NumPy Arrays: {tempo_numpy:.6f} segundos")
```

```
Comparação dos Tempos de Execução:
Lista de Dicionários: 0.001460 segundos
Dicionário de Dicionários: 0.001218 segundos
Pandas DataFrame: 0.114621 segundos
NumPy Arrays: 0.017650 segundos
```

Análise dos Resultados

Com base nos tempos medidos, podemos comparar a eficiência de cada estrutura:

- **Dicionário de Dicionários:** Apresentou o menor tempo de execução (0.001218 segundos), sendo a opção mais eficiente entre as testadas. Isso ocorre porque os dicionários permitem acesso rápido aos elementos por chave, reduzindo o tempo de busca e agregação sem a necessidade de loops complexos.
- **Lista de Dicionários:** Teve um tempo de execução ligeiramente maior (0.001460 segundos) em comparação com o dicionário de dicionários. Embora seja simples de implementar, a iteração explícita sobre a lista adiciona um pequeno overhead, especialmente quando comparado ao acesso direto por chave nos dicionários.
- **NumPy Arrays:** Apresentou um tempo de execução intermediário (0.017650 segundos). Apesar de o NumPy ser altamente otimizado para operações em grandes volumes de dados numéricos, neste caso específico, o processo de filtragem e cálculo da média para cada usuário adiciona complexidade ao código e aumenta o tempo de execução em comparação com estruturas mais simples.
- **Pandas DataFrame:** Teve o maior tempo de execução (0.114621 segundos), sendo significativamente mais lento que as outras abordagens. Isso pode ser atribuído ao overhead associado aos métodos internos do Pandas para operações de agrupamento e agregação. O Pandas é mais eficiente em conjuntos de dados maiores, onde suas otimizações podem superar o overhead inicial.

Observação: Os tempos de execução foram medidos em um conjunto de dados relativamente pequeno. Em cenários com volumes de dados maiores, as vantagens das otimizações internas do Pandas e do NumPy podem se tornar mais evidentes, potencialmente reduzindo o tempo de execução em relação às estruturas básicas.

Assim, para conjuntos de dados pequenos, como o utilizado neste projeto, estruturas simples como dicionários oferecem melhor desempenho e simplicidade de código. Em contrapartida, para conjuntos de dados maiores e operações mais complexas, o uso de bibliotecas como Pandas ou NumPy pode ser mais adequado, apesar do overhead inicial, devido às suas funcionalidades avançadas e capacidade de lidar com grandes volumes de dados de forma eficiente.

Análise da Média de Precisão por Residente

Utilizando as funções implementadas, obtivemos as seguintes médias de precisão para cada residente:

In [59]:
```
media_precisao = media_precisao_pandas(df_executa)
print("Média de Precisão por Residente:")
for id_usuario, media in media_precisao.items():
    nome = df_usuarios[df_usuarios['ID_USUARIO'] == id_usuario]['NM_USUARIO'].values[0]
    print(f"{nome} (ID {id_usuario}): {media:.4f}")
```

```
Média de Precisão por Residente:
Caíque Walter Silva (ID 3): 0.4921
Gabriela Marsiglia (ID 4): 0.5827
Matheus José de Lima Costa (ID 5): 0.4925
```

Observamos que o residente com `Gabriela Marsiglia (ID 4)` possui a maior média de precisão.

Fechar a Conexão com o Banco de Dados

In [55]:
```
connection.close()
print("Conexão encerrada.")
```

```
Conexão encerrada.
```

Conclusão

Os resultados indicam que, para conjuntos de dados pequenos, como os utilizados neste projeto, estruturas simples como Dicionários de Dicionários são mais eficientes tanto em termos de tempo de execução quanto em simplicidade de código. Elas permitem acesso rápido e direto aos dados, facilitando operações como agregações e cálculos estatísticos sem a necessidade de bibliotecas externas.

Embora o Pandas DataFrame seja uma ferramenta poderosa para manipulação e análise de dados, seu uso pode não ser justificado em cenários com pequenos volumes de dados, devido ao overhead de performance observado. No entanto, em situações com conjuntos de dados maiores e operações mais complexas, as otimizações internas do Pandas podem superar esse overhead inicial, tornando-o mais eficiente.

O uso de NumPy Arrays também pode ser vantajoso em operações numéricas intensivas e com grandes volumes de dados, mas requer um código mais elaborado para manipulações que são mais diretas com dicionários.

Referências

- Documentação do oracledb
- Documentação do Pandas
- Documentação do NumPy
- Documentação do timeit