



ESCOLA POLITÉCNICA

PCS3623 - Banco de Dados I

Relatório Final - Projeto Prático

Professor Dr. Jorge Rady de Almeida Junior

Engenharia da Computação - EPUSP

Enzo Cardeal Neves 11257522

Felipe Bagni 11257571

Gustavo Azevedo Corrêa 11257693

Leticia Cohen Ferrari 10792240

São Paulo

Abril de 2021

Sumário

1	Introdução	2
2	Descrição da Atividade	2
2.1	Motivação	2
2.2	Modelagem	2
2.3	Levantamento dos dados	5
2.4	Desenvolvimento do Banco de Dados	6
3	Consultas	16
4	Manual do Usuário	18
5	Conclusão	18

1 Introdução

O objetivo deste documento é relatar a proposta, desenvolvimento e resultados obtidos para o projeto final da disciplina de PCS3623 - Banco de Dados I.

2 Descrição da Atividade

2.1 Motivação

Para desenvolver um projeto relacionado a algum tema da disciplina, o grupo optou por abordar a questão do desempenho e da otimização de bancos de dados. Esse tema foi trabalhado em cima de um banco de dados cuja temática é a vacinação contra o vírus da COVID-19 no estado de São Paulo.

A motivação para a escolha desse assunto é a quantidade imensa de dados gerada diariamente que traz a necessidade da utilização de um banco de dados otimizado para possibilitar a análise desses dados tão relevantes na atualidade.

2.2 Modelagem

Tendo como objetivo o desenvolvimento de um banco de dados útil e funcional para análises relacionadas à saúde pública, levantaram-se algumas entidades e atributos que julgaram-se como relevantes:

- Vacina(IdVacina, Nome);
- Laboratório(IdLaboratório, Nome, País);
- Pessoa(IdPessoa, Nome, Gênero, Data de nascimento, Etnia, Ocupação);
- Município(IdMunicípio, Nome, População);
- UBS(IdUBS, Nome, Endereço);
- Dose(IdDose, Número, Data de validade, Status).

A partir da definição das entidades, definiram-se os seguintes relacionamentos entre elas:

- Produzida_por(IdVacina, IdLaboratório);

- Habita_em(IdMunicípio, IdPessoa);
- Aplicada_em(IdPessoa, IdDose, Data);
- Fica_no(IdUBS, IdMunicípio);
- Do_tipo(IdDose, IdVacina);
- Enviada_para(IdVacina, IdUBS);
- Tem(IdDose, IdUBS).

Com todas as entidades e relacionamentos definidas, desenvolveu-se o Diagrama de Entidades e Relacionamentos da Figura 1

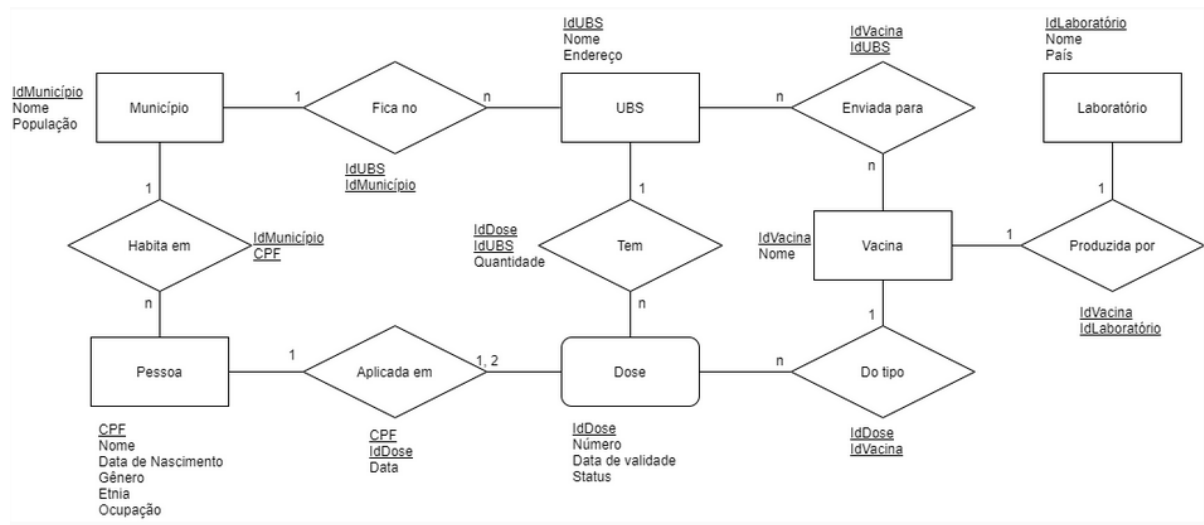


Figura 1: Diagrama Entidade-Relacionamento

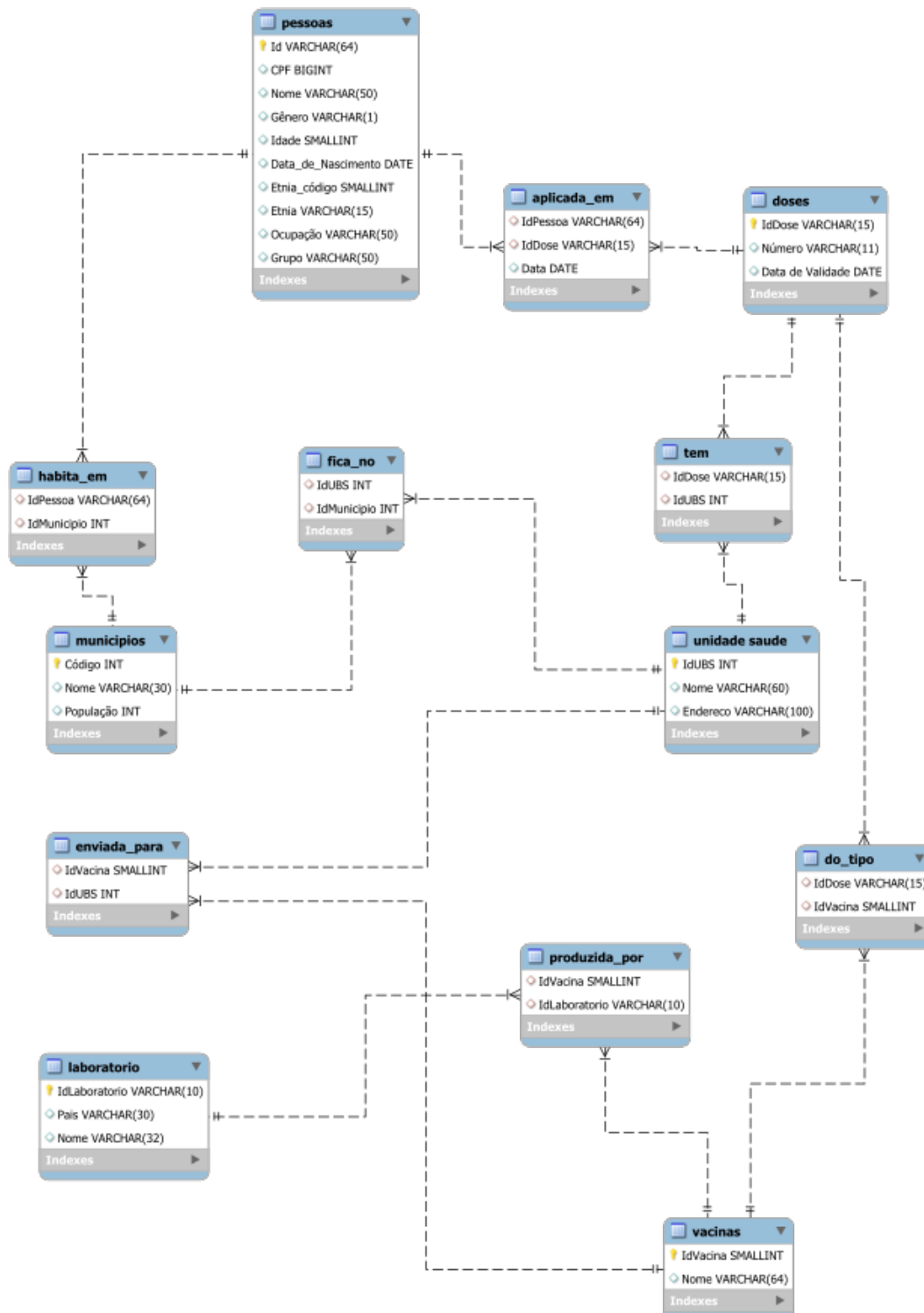


Figura 2: Diagrama Entidade-Relacionamento pelo MySQL Workbench

2.3 Levantamento dos dados

Feita a modelagem, iniciou-se o trabalho de coleta dos dados necessários para o banco de dados. Felizmente, a maior parte deles está disponível de forma aberta em sites governamentais, sendo eles o do *IBGE* [2] e o *openDataSUS* [3].

Contudo, nem todos os dados necessários para a construção do banco de dados estavam disponíveis. Os dados referentes às identificações das pessoas, como nome e CPF, não estão presentes nos dados fornecidos pelo *openDataSUS*, provavelmente, por questões relacionadas à LGPD. Além disso, dados como os endereços das unidades de saúde e a data de validade das doses de cada vacina não foram encontrados pelos integrantes do grupo de forma simples.

Por isso, para fins de prova de conceito, optou-se por gerar de forma sintética os dados faltantes com o auxílio de uma biblioteca do *Python* de geração aleatória de dados chamada *faker*.

```
1 import numpy as np
2 from faker import Faker
3 from random import *
4
5 fake = Faker('pt_BR') # Setting the data language as PT-BR
6
7 """The following process was necessary because running fake.name()
8     would sometimes return Sr. or Dr. before the actual name"""
9
10 # FAKE NAMES
11
12 def fake_names(n):
13     names = [fake.first_name() for i in range(n)] # Generate fake first
14     last_names = [fake.last_name() for i in range(n)] # Generate fake
15     complete_names = [names[i] + " " + last_names[i] for i in range(n)] # Generate fake complete names
16     return complete_names
```

Listing 1: Geração de dados sintéticos

2.4 Desenvolvimento do Banco de Dados

Para desenvolver o banco de dados, utilizou-se o *Python* com o auxílio de algumas bibliotecas para tratar e transferir os dados para o software gerenciador MySQL.

Com o intuito de apresentar as partes mais relevantes do código, disponibilizado no repositório do *GitHub* [1], optou-se por utilizar os trechos relativos à criação da tabela de Pessoas como exemplo.

No arquivo principal *pass_df_to_mysql.py*, apresentado parcialmente no Listing 2, desenvolve-se um *DataFrame* a partir do arquivo *CSV* disponibilizado diariamente na plataforma *openDataSUS* a respeito da vacinação. Feito isso, insere-se esse *DataFrame* como parâmetro na função *fill_table*, que é apresentada na Listing 3. Finalmente, com um novo *DataFrame* formado especificamente para as informações das pessoas, realiza-se a conversão dele para uma tabela no *MySQL*.

```
1 import sqlalchemy
2 from generate_df import *
3 import real_inf
4 from sqlalchemy import create_engine
5 import pymysql
6 import pandas as pd
7
8 #cria munic pio
9 print("Lendo arquivo de munic pios")
10 arquivo = 'c3d64a3788342bbdd97d01ef7694f1a0.xlsx'
11 municipio = Municipio("Munic pio")
12 municipio.fill_table(arquivo)
13 municipio.create_df()
14 df_municipio = municipio.out_df()
15 # removendo ltimo digito do c digo do municipio
16 df_municipio = df_municipio.astype({"C digo" : str})
17 df_municipio['C digo'] = df_municipio['C digo'].str[:-3]
18
19 #[...]
20
21 # cria pessoa
22 print("Gerando DataFrame Pessoa")
23 pessoa = Pessoa("Pessoa")
24 pessoa.fill_table(df_other_info)
25 pessoa.create_df()
26 df_pessoa = pessoa.out_df()
27 df_pessoa = df_pessoa.drop_duplicates(keep='first', subset=['Id'])
```

```

28
29 #[...]
30
31 # Convert dataframe to sql table
32 print("Criando tabela Pessoa")
33 df_pessoa['Data_de_Nascimento'] = pd.to_datetime(df_pessoa['
    Data_de_Nascimento'], format='%Y-%m-%d')
34 df_pessoa.to_sql('pessoas', engine, index=False,
35                 dtype={
36                     'Id' : sqlalchemy.types.VARCHAR(length=64),
37                     'CPF' : sqlalchemy.types.BIGINT,
38                     'Nome' : sqlalchemy.types.VARCHAR(length=50),
39                     'G nero' : sqlalchemy.types.VARCHAR(length=1),
40                     'Idade' : sqlalchemy.types.SMALLINT,
41                     'Data_de_Nascimento' : sqlalchemy.types.DATE,
42                     'Etnia_c digo' : sqlalchemy.types.SMALLINT,
43                     'Etnia' : sqlalchemy.types.VARCHAR(length=15),
44                     'Ocupa o' : sqlalchemy.types.VARCHAR(length=50),
45                     'Grupo' : sqlalchemy.types.VARCHAR(length=50)
46
47                 })
48 print("Tabela Pessoa criada")

```

Listing 2: Geração de DataFrame e criação de tabela no MySQL

O método *fill_table* da classe *Pessoa* utiliza todos os parâmetros retornados da função *pessoa_info*, cujo código consta na Listing 3. Todos esses parâmetros são alocados em forma de dicionário, como pode-se observar na Listing 3.

```

1 class Pessoa(Table):
2     def fill_table(self, df):
3         vetor_pessoa_info = pessoa_info(df)
4         self.dic = {
5             'Id' : vetor_pessoa_info[0],
6             'CPF' : vetor_pessoa_info[1],
7             'Nome' : vetor_pessoa_info[2],
8             'G nero' : vetor_pessoa_info[3],
9             'Idade' : vetor_pessoa_info[4],
10            'Data_de_Nascimento' : vetor_pessoa_info[5],
11            'Etnia_c digo' : vetor_pessoa_info[6],
12            'Etnia' : vetor_pessoa_info[7],
13            'Ocupa o' : vetor_pessoa_info[8],
14            'Grupo' : vetor_pessoa_info[9]

```



```
15     }
```

Listing 3: Classe Pessoa e método de preenchimento de tabela

A função *peessoa_info* separa as colunas do *DataFrame* criado, convertendo-as em listas que serão retornadas. Além disso, na Listing 4 pode-se observar o uso de funções geradoras de dados sintéticos, como a da Listing 1, para completar os dados faltantes da entidade em questão.

```
1 def pessoa_info(df):
2     pessoa_id = list(df["paciente_id"]) # novo atributo
3     pessoa_cpf = fake_cpf(len(pessoa_id))
4     pessoa_nome = fake_names(len(pessoa_id))
5     pessoa_enumSexoBiologico = list(df["paciente_enumSexoBiologico"])
6     pessoa_idade = list(df["paciente_idade"]) # novo atributo
7     pessoa_dataNascimento = list(df["paciente_dataNascimento"])
8     pessoa_racaCor_codigo = list(df["paciente_racaCor_codigo"])
9     pessoa_racaCor_valor = list(df["paciente_racaCor_valor"])
10    pessoa_ocupacao = fake_oc(len(pessoa_id))
11    pessoa_grupo = list(df["vacina_categoria_nome"]) # novo atributo
12    return (pessoa_id, pessoa_cpf, pessoa_nome,
13            pessoa_enumSexoBiologico, pessoa_idade,
14            pessoa_dataNascimento, pessoa_racaCor_codigo,
15            pessoa_racaCor_valor,
16            pessoa_ocupacao, pessoa_grupo)
```

Listing 4: Pessoa Info

Ferramentas utilizadas e justificativa de sua escolha

No desenvolvimento da interface gráfica (GUI) foi usado a biblioteca PyQt5 do Python em conjunto com as bibliotecas usadas na geração das tabelas para conectar o programa ao banco de dados. A escolha desse método partiu do princípio que a biblioteca utilizada tinha um bom desempenho em criar protótipos de interface com facilidade mas ao mesmo tempo permitir sua customização de forma livre, coisa que seria dificultada caso usasse-se Tkinter, onde seria fácil a criação da interface mas necessitaria de muito trabalho para deixá-la visualmente agradável.

Utilização das ferramentas

Com fins de organização foi dividido a produção da GUI em duas partes: o visual (frontend) e o funcional (backend).

Para a criação do frontend foi utilizado uma ferramenta chamada Qt Designer, ela permite que o usuário crie por meio de "drag and drop" as janelas das suas aplicações e seus widgets (e.g. botões, listas e fotos). A partir dela foram inseridas a parte gráfica das ferramentas utilizadas na interface.

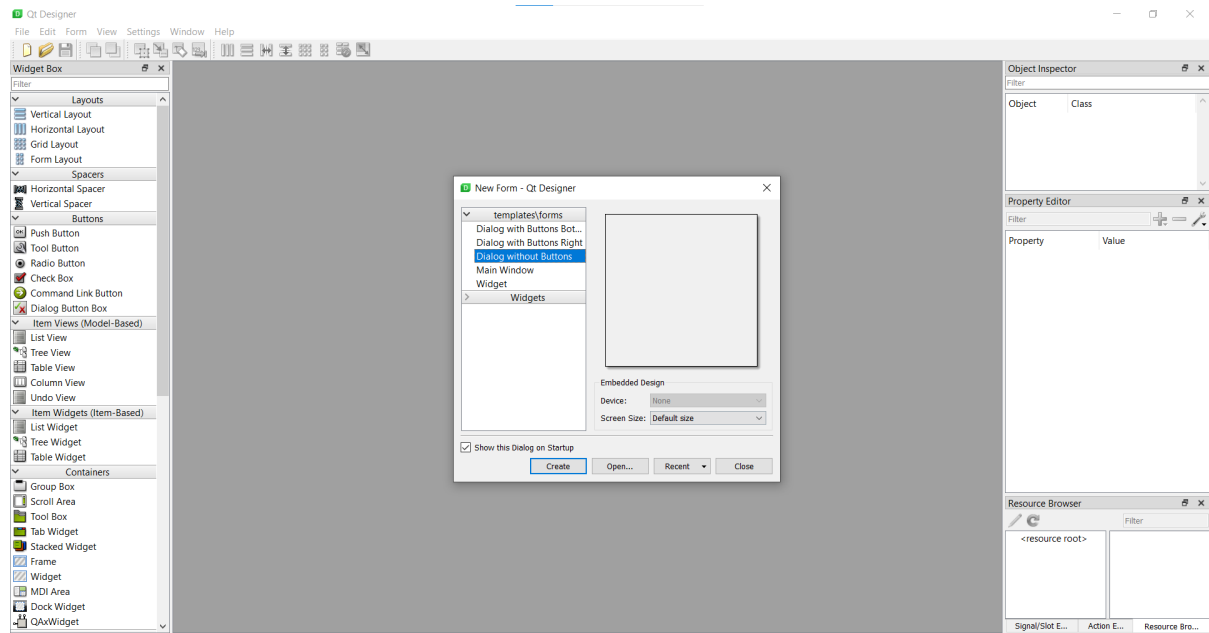
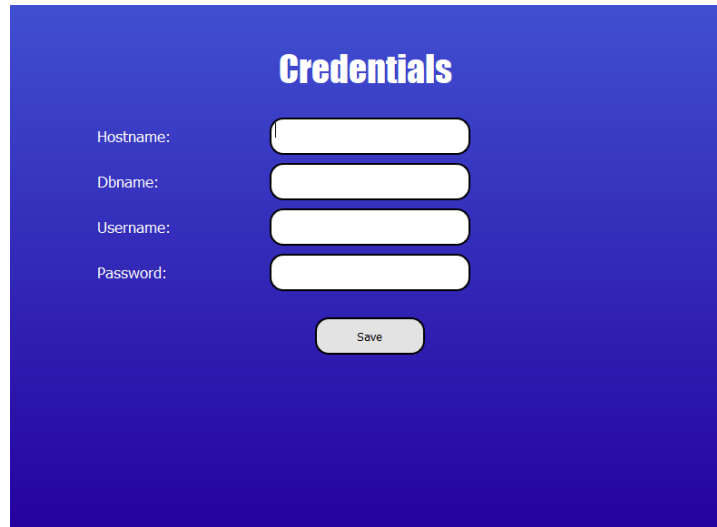


Figura 3: Software utilizado no frontend

A partir disso montou-se uma tela de login, uma tela de busca e salvamento das buscas feitas no banco de dados e uma tela de buscas úteis prontas indexadas pelo nome do que seria buscado. O resultado final ficou da seguinte forma:

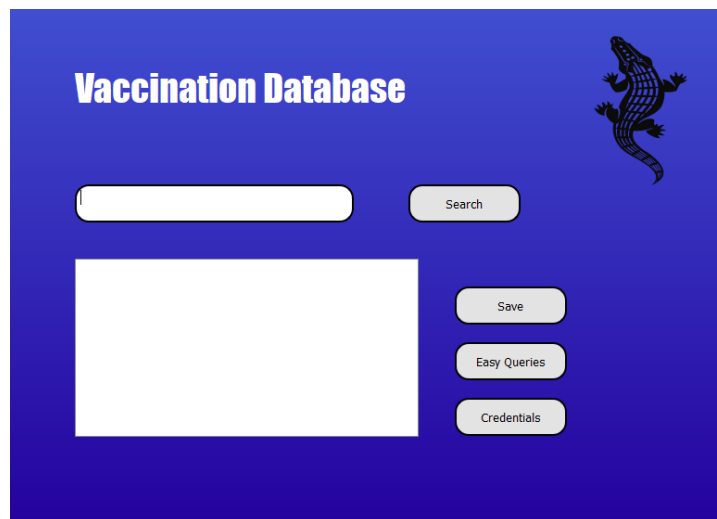
Tela de Login



The login form is titled "Credentials" in bold white text. It features four input fields on the left, each with a label: "Hostname:", "Dbname:", "Username:", and "Password:". To the right of these labels are four corresponding white input boxes. Below the input fields is a single "Save" button.

Figura 4: Tela de Login

Tela de Busca



The search form is titled "Vaccination Database" in bold white text. In the top right corner, there is a black silhouette of a lizard. Below the title, there is a white input field for search terms and a "Search" button. Below the input field is a large, empty white rectangular box. To the right of this box are three buttons: "Save", "Easy Queries", and "Credentials".

Figura 5: Tela de Busca

Tela de Buscas Prontas

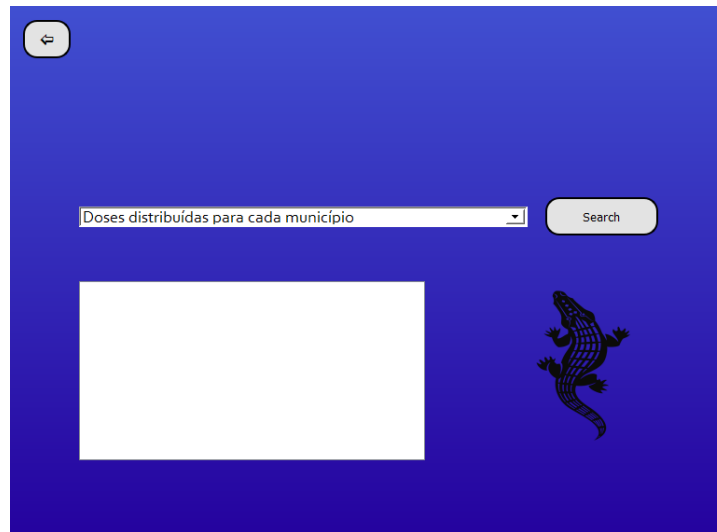


Figura 6: Tela de Buscas Prontas

Depois das interfaces estarem satisfatórias, gera-se um arquivo em python para cada janela com as descrições estruturais e definição dos widgets.

Para fazer o backend do programa, escreve-se um programa em python que acessa as descrições estruturais dos outros arquivos e descreve seu funcionamento.

A partir do backend as telas ganham as funcionalidades. A tela de login pega as credenciais do usuário e as salva. A tela de busca pega o input escrito, realiza o comando no banco de dados e salva o que foi buscado em um arquivo .csv . A tela de buscas prontas pega o que foi escolhido pelo usuário na caixa de seleção e puxa os resultados da busca.

O código de cada tela ficou da seguinte forma:

Tela de Login

```
1 class Dialog_cred(QtWidgets.QDialog):
2
3     def __init__(self, *args, **kwargs):
4         super().__init__(*args, **kwargs)
5
6         self.ui = Ui_Dialog_cred()
7         self.ui.setupUi(self)
8         self.ui.search.clicked.connect(self.save_bt)
```

```

9
10
11     def save_bt(self):
12         global hostname
13         global dbname
14         global username
15         global pwd
16         global connection
17         global cursor
18         hostname = self.ui.plainTextEdit.toPlainText()
19         dbname = self.ui.plainTextEdit_2.toPlainText()
20         username = self.ui.plainTextEdit_3.toPlainText()
21         pwd = self.ui.lineEdit.text()
22         print(hostname)
23         print(dbname)
24         print(username)
25         print(pwd)
26
27         connection = pymysql.connect(
28             host=hostname,
29             user=username,
30             password=pwd,
31             db=dbname
32         )
33         cursor = connection.cursor()
34         widget.setCurrentIndex(widget.currentIndex() + 1)

```

Listing 5: Funcionalidade da tela de Login

Tela de Busca

```

1 class MainWindow(QMainWindow):
2
3     def __init__(self, *args, **kwargs):
4         super().__init__(*args, **kwargs)
5
6         self.ui = Ui_MainWindow()
7         self.ui.setupUi(self)
8         self.ui.pushButton.clicked.connect(self.exec_quer)
9         self.ui.pushButton_2.clicked.connect(self.save_quer)
10        self.ui.pushButton_3.clicked.connect(self.go_to_easy)
11        self.ui.pushButton_4.clicked.connect(self.go_to_cred)

```

```

12
13
14     def exec_quer(self):
15         text = self.ui.plainTextEdit.toPlainText()
16         df = pd.read_sql(text, connection)
17         model = pandasModel(df)
18         self.ui.tableView.setModel(model)
19
20     def save_quer(self):
21         text = self.ui.plainTextEdit.toPlainText()
22         df = pd.read_sql(text, connection)
23         print(df)
24         df.to_csv(r'querie.csv')
25
26     def go_to_easy(self):
27         widget.setCurrentIndex(widget.currentIndex() + 1)
28         widget.setFixedHeight(widget.currentWidget().frameGeometry().
height())
29         widget.setFixedWidth(widget.currentWidget().frameGeometry().
width())
30
31     def go_to_cred(self):
32         widget.setCurrentIndex(widget.currentIndex() - 1)
33         widget.setFixedHeight(widget.currentWidget().frameGeometry().
height())
34         widget.setFixedWidth(widget.currentWidget().frameGeometry().
width())

```

Listing 6: Funcionalidade da Tela de Busca

```

1 class Dialog(QtWidgets.QDialog):
2
3     def __init__(self, *args, **kwargs):
4         super().__init__(*args, **kwargs)
5
6         self.ui = Ui_Dialog()
7         self.ui.setupUi(self)
8         self.ui.back.clicked.connect(self.back)
9         self.ui.search.clicked.connect(self.fav_search)
10
11
12     def back(self):

```

```

13         widget.setCurrentIndex(widget.currentIndex() - 1)
14         widget.setFixedHeight(widget.currentWidget().frameGeometry().
height())
15         widget.setFixedWidth(widget.currentWidget().frameGeometry().
width())
16
17     def fav_search(self):
18         quer_dic = {
19             "Doses distribu das para cada munic pio" : "SELECT m.Nome
, count(d.IdDose) FROM municipios m, `unidade saude` u, tem t, doses
d, fica_no f WHERE m.C digo=f.IdMunicipio AND f.IdUBS=u.IdUBS AND
u.IdUBS=t.IdUBS AND t.IdDose=d.IdDose GROUP BY m.Nome;",
20             "Rela o Vacina e Origem": "SELECT lab.Nome AS
Laborat rio , lab.Pais, v.Nome AS Vacina, count(d.IdDose) AS `Doses
aplicadas` FROM parcial.laboratorio lab, parcial.produzida_por p,
parcial.vacinas v, parcial.do_tipo dt, parcial.doses d WHERE lab.
IdLaboratorio=p.IdLaboratorio AND v.IdVacina=p.IdVacina AND v.
IdVacina=dt.IdVacina AND d.IdDose=dt.IdDose GROUP BY v.Nome;",
21             "Doses aplicadas por munic pios" : "SELECT m.Nome, count(d
.IdDose) FROM municipios m, habita_em h, pessoas p, aplicada_em a,
doses d WHERE m.C digo=h.IdMunicipio AND h.IdPessoa=p.Id AND p.Id=a
.IdPessoa AND a.IdDose=d.IdDose GROUP BY m.Nome;",
22             "Pessoas vacinadas por faixa et ria" : "SELECT p1.Nome, p1
.Data_de_Nascimento FROM pessoas p1 WHERE p1.Data_de_Nascimento <
\"1950-01-01\" AND EXISTS(SELECT * FROM aplicada_em a WHERE a.
IdPessoa=p1.Id);",
23             "N mero de doses importadas da China" : "SELECT l.Pais, l.
Nome, v.Nome, count(dt.idDose) FROM laboratorio l, produzida_por pp,
vacinas v, do_tipo dt, doses d WHERE l.IdLaboratorio=pp.
IdLaboratorio AND v.IdVacina=pp.IdVacina AND v.IdVacina=dt.IdVacina
AND d.IdDose=dt.IdDose AND l.Pais=\"China\" GROUP BY l.Nome;",
24             "N mero de pessoas que receberam a 2 dose por munic pio
" : "SELECT m.Nome, count(p.Id) FROM parcial.pessoas p, parcial.
aplicada_em ae, parcial.doses d, parcial.habita_em h, parcial.
municipios m WHERE p.Id=ae.IdPessoa AND d.IdDose=ae.IdDose AND d.
N mero=' 2 Dose' AND p.Id = h.IdPessoa AND h.IdMunicipio = m.
C digo GROUP BY m.C digo;"
25         }
26         pesquisa = self.ui.box.currentText()
27         querie = quer_dic[pesquisa]
28         df = pd.read_sql(querie, connection)
29         model = pandasModel(df)

```

```
30 self.ui.table.setModel(model)
```

Listing 7: Funcionalidade da Tela de Buscas Prontas

Para fazer a conexão com a base de dados cria-se uma classe que pega um o dataframe devolvido pelo querie da busca e transforma em uma tabela para o PyQt5.

```
1
2 class pandasModel(QAbstractTableModel):
3
4     def __init__(self, data):
5         QAbstractTableModel.__init__(self)
6         self._data = data
7
8     def rowCount(self, parent=None):
9         return self._data.shape[0]
10
11    def columnCount(self, parent=None):
12        return self._data.shape[1]
13
14    def data(self, index, role=Qt.DisplayRole):
15        if index.isValid():
16            if role == Qt.DisplayRole:
17                return str(self._data.iloc[index.row(), index.column()
18    ])
19
20    def headerData(self, col, orientation, role):
21        if orientation == Qt.Horizontal and role == Qt.DisplayRole:
22            return self._data.columns[col]
23        return None
```

Listing 8: Classe para criação da tabela

Há ainda a parte do código que permite a alternância das telas pela criação de uma lista com elas e também executa a aplicação como um todo.

```
1
2 if __name__ == '__main__':
3     app = QtWidgets.QApplication(sys.argv)
4     widget = QtWidgets.QStackedWidget()
5     main_win = MainWindow()
6     easy = Dialog()
```



```

7 cred = Dialog_cred()
8 widget.addWidget(cred)
9 widget.addWidget(main_win)
10 widget.addWidget(easy)
11 widget.setFixedHeight(main_win.frameGeometry().height())
12 widget.setFixedWidth(main_win.frameGeometry().width())
13 widget.show()
14
15 sys.exit(app.exec())

```

Listing 9: Código que roda o programa

Assim, a junção do frontend e backend constroem uma GUI funcional para visualizar o banco de dados de vacinação.

3 Consultas

As consultas prontas escolhidas para serem feitas na GUI foram as que seriam consideradas mais utilizadas repetidamente por alguém interessado no banco de dados, essas seriam:

- Doses distribuídas para cada município

```

1 SELECT m.Nome, count(d.IdDose) FROM municipios m, `unidade
saude` u, tem t, doses d, fica_no f WHERE m.C digo=f.
IdMunicipio AND f.IdUBS=u.IdUBS AND u.IdUBS=t.IdUBS AND t.
IdDose=d.IdDose GROUP BY m.Nome;
2

```

- Relação Vacina e Origem

```

1 SELECT lab.Nome AS Laborat rio, lab.Pais, v.Nome AS Vacina,
count(d.IdDose) AS `Doses aplicadas` FROM parcial.laboratorio
lab, parcial.produzida_por p, parcial.vacinas v, parcial.
do_tipo dt, parcial.doses d WHERE lab.IdLaboratorio=p.
IdLaboratorio AND v.IdVacina=p.IdVacina AND v.IdVacina=dt.
IdVacina AND d.IdDose=dt.IdDose GROUP BY v.Nome;
2

```

- Doses aplicadas por municípios

```

1      SELECT m.Nome, count(d.IdDose) FROM municipios m, habita_em h,
      pessoas p, aplicada_em a, doses d WHERE m.C d igo=h.
      IdMunicipio AND h.IdPessoa=p.Id AND p.Id=a.IdPessoa AND a.
      IdDose=d.IdDose GROUP BY m.Nome;
2

```

- Pessoas vacinadas por faixa etária

```

1      SELECT p1.Nome, p1.Data_de_Nascimento FROM pessoas p1 WHERE
      p1.Data_de_Nascimento < \"1950-01-01\" AND EXISTS(SELECT *
      FROM aplicada_em a WHERE a.IdPessoa=p1.Id);
2

```

- Número de doses importadas da China

```

1      SELECT l.Pais, l.Nome, v.Nome, count(dt.idDose) FROM
      laboratorio l, produzida_por pp, vacinas v, do_tipo dt, doses d
      WHERE l.IdLaboratorio=pp.IdLaboratorio AND v.IdVacina=pp.
      IdVacina AND v.IdVacina=dt.IdVacina AND d.IdDose=dt.IdDose AND
      l.Pais=\"China\" GROUP BY l.Nome;
2

```

- Número de pessoas que receberam a 2ª dose por município

```

1      SELECT m.Nome, count(p.Id) FROM parcial.pessoas p, parcial.
      aplicada_em ae, parcial.doses d, parcial.habita_em h, parcial.
      municipios m WHERE p.Id=ae.IdPessoa AND d.IdDose=ae.IdDose AND
      d.N mero='      2      Dose' AND p.Id = h.IdPessoa AND h.
      IdMunicipio = m.C d igo GROUP BY m.C d igo;
2

```

4 Manual do Usuário

O manual do usuário pode ser encontrado na íntegra no README.md do repositório do GitHub, cujo acesso pode ser feito por meio deste [link](#).

5 Conclusão

Por meio do projeto prático, foi possível por em prática vários dos conceitos vistos nas aulas teóricas durante o curso. Indo desde a obtenção dos dados até uma Interface Gráfica para o usuário. O grande desafio foi manipular a grande quantidade de dados, pelo qual aprendemos diversas técnicas e ferramentas usadas no mercado para a realização de tais manipulações e testes.

Referências

- [1] NEVES, ENZO C.; BAGNI, FELIPE; CORRÊA, GUSTAVO A.; FERRARI, LETICIA C. *Link do repositório do Projeto no GitHub*. https://github.com/guazco/Vaccination_Database Grupo 7, Banco de Dados I, Departamento de Engenharia da Computação Sistemas digitais, USP, São Paulo, SP, 2021
- [2] IBGE. *Cidades e Estados - São Paulo*. <https://www.ibge.gov.br/cidades-e-estados/sp.html>
- [3] OPENDATASUS. *Registros de Vacinação COVID19*. <https://opendatasus.saude.gov.br/dataset/covid-19-vacinacao/resource/ef3bd0b8-b605-474b-9ae5-c97390c197a8>