

PROJETO INTEGRADOR STEP BY STEP

Integrantes: André Luis dos Santos Fagundes, José Danrley da Silva, Luis Guilherme Belem de Sousa, Matheus Amauri de Jesus Campos, Washington Henrique Fernandes de Sousa

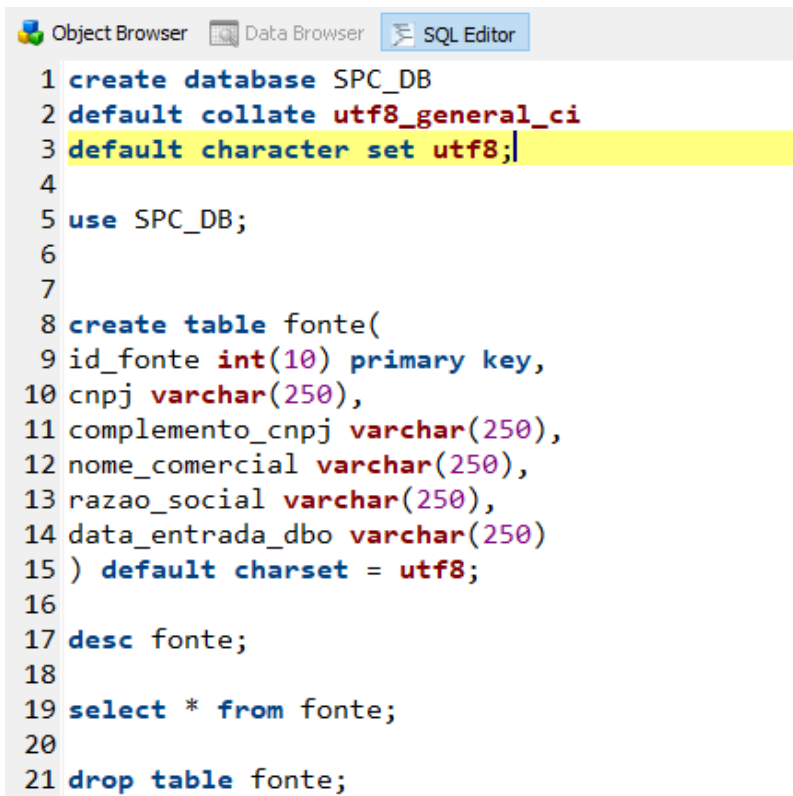
Descrição: todo processo realizado até o momento, com a remessa de dados enviada pelo SPC e todo tratamento realizado voltado para os indicadores solicitados sobre o Cadastro Positivo, como objetivo para a primeira entrega do projeto.

1. BANCO DE DADOS EM MYSQL: SQL FRONT

Visando a melhor análise dos dados, para o projeto foi utilizado o armazenamento e gerenciamento dos dados pela ferramenta MySQL, que teve os seguintes processos de desenvolvimento.

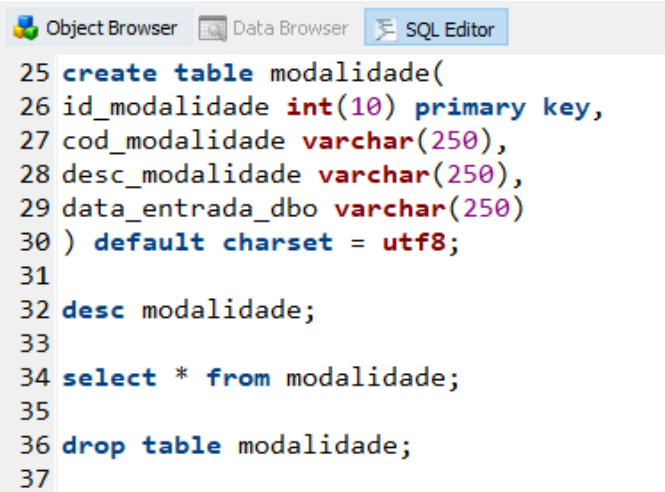
1.1 Criação das tabelas enviadas em SQL

→ tabela FONTE



```
1 create database SPC_DB
2 default collate utf8_general_ci
3 default character set utf8;
4
5 use SPC_DB;
6
7
8 create table fonte(
9 id_fonte int(10) primary key,
10 cnpj varchar(250),
11 complemento_cnpj varchar(250),
12 nome_comercial varchar(250),
13 razao_social varchar(250),
14 data_entrada_dbo varchar(250)
15 ) default charset = utf8;
16
17 desc fonte;
18
19 select * from fonte;
20
21 drop table fonte;
```

→ tabela MODALIDADE



The screenshot shows a SQL Editor window with three tabs: Object Browser, Data Browser, and SQL Editor. The SQL Editor tab is active and contains the following SQL code:

```
25 create table modalidade(  
26 id_modalidade int(10) primary key,  
27 cod_modalidade varchar(250),  
28 desc_modalidade varchar(250),  
29 data_entrada_dbo varchar(250)  
30 ) default charset = utf8;  
31  
32 desc modalidade;  
33  
34 select * from modalidade;  
35  
36 drop table modalidade;  
37
```

→ tabela MOVIMENTO



The screenshot shows a SQL Editor window with three tabs: Object Browser, Data Browser, and SQL Editor. The SQL Editor tab is active and contains the following SQL code:

```
40 create table movimento(  
41 id_movimento int(10) primary key,  
42 saldo_utilizado varchar(250),  
43 valor_total varchar(250),  
44 valor_minimo varchar(250),  
45 valor_parcelado varchar(250),  
46 quant_cli_cad_pos varchar(250),  
47 quant_mvt varchar(250),  
48 tipo_cliente varchar(250),  
49 id_fonte varchar(250),  
50 cod_modalidade varchar(250),  
51 data_ult_alteracao date,  
52 data_entrada_dbo varchar(250),  
53 foreign key (id_fonte) references fonte(id_fonte),  
54 foreign key (cod_modalidade) references modalidade(cod_modalidade)  
55 ) default charset = utf8;  
56  
57 desc movimento;  
58  
59 select * from movimento;  
60  
61 drop table movimento;  
62
```

→ tabela OPERAÇÃO

```
Object Browser Data Browser SQL Editor
64
65 create table operacao(
66 id_registro_base int(10) primary key,
67 valor_total_contrato varchar(250),
68 quant_parcelas_contratadas varchar(250),
69 valor_ainda_não_pago varchar(250),
70 quant_cli_cad_pos varchar(250),
71 quant_opr varchar(250),
72 id_fonte int(10),
73 cod_modalidade varchar(250),
74 tipo_cliente enum("F", "J"),
75 data_ult_alteracao date,
76 data_entrada_dbo varchar(250),
77 foreign key (id_fonte) references fonte(id_fonte),
78 foreign key (cod_modalidade) references modalidade(cod_modalidade)
79 ) default charset = utf8;
80
81 desc operacao;
82
83 select * from operacao;
84
85 drop table operacao;
```

→ tabela PAGAMENTO

```
Object Browser Data Browser SQL Editor
89 create table pagamento(
90 id_pagamento int(10) primary key,
91 valor_pago varchar(250),
92 data_vencimento date,
93 cod_modalidade varchar(250),
94 quant_cli_cad_pos varchar(250),
95 quant_pgt varchar(250),
96 id_fonte int(10),
97 tipo_cliente varchar(250),
98 data_ult_alteracao date,
99 data_entrada_dbo varchar(250),
100 foreign key (id_fonte) references fonte(id_fonte),
101 foreign key (cod_modalidade) references modalidade(cod_modalidade)
102 ) default charset = utf8;
103
104 desc pagamento;
105
106 select * from pagamento;
107
108 drop table pagamento;
```

→ Total de **5 TABELAS** exportadas, observações das planilhas enviadas:

Fonte (ID_STG_FNT_ITT / NUM_CNPJ / NUM_CMP_CNPJ / NOM_COM / NOM_RAZ_SCL / DAT_INC_DBO)

Modalidade (ID_STG_MDL / COD_MDL / DES_MDL / DAT_INC_DBO)

Movimento (ID_STG_MVT_CRD / VLR_SDO_UTZ_CRD_RTO / VLR_TOT_FAT / VLR_MIM_FAT / VLR_PCL_FAT / QTD_CLI_CAD_POS / QTD_MVT / DES_TIP_PSS / ID_FNT_ITT / COD_MDL / DAT_RSS_FNT_ITT / DAT_INC_DBO)

obs: no dicionário de dados atualizado não consta as 12 colunas

Operação (ID_STG_OPR_ITT / VLR_CTRD_CSC / QTD_PCL / VLR_SDO_DDR / QTD_CLI_CAD_POS / QTD_OPR / ID_FNT_ITT / ID_MDL / DES_TIP_PSS / ID_FNT_ITT / COD_MDL / DAT_RSS_FNT_ITT / DAT_INC_DBO)

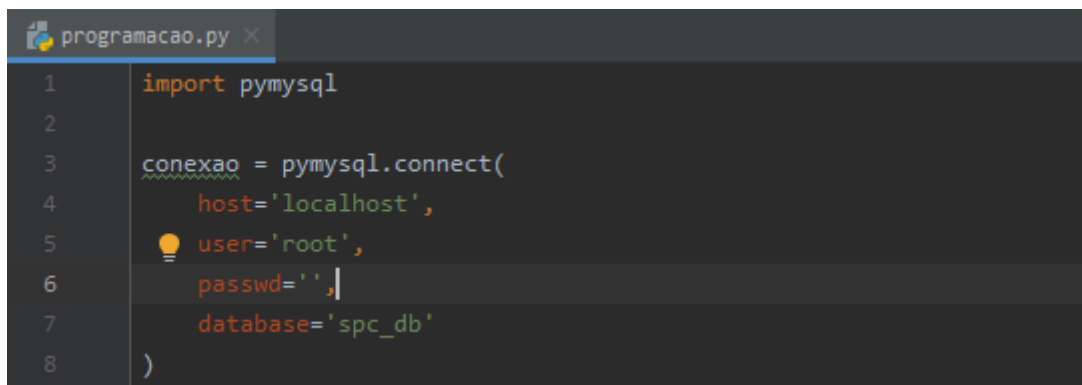
obs: COD_MDL aparece na tabela mas não aparece no novo dicionário.

Pagamento (ID_STG_PGT / VLR_PGT_FAT / DAT_VCT / COD_MDL / QTD_CLI_CAD_POS / QTD_PGT / ID_FNT_ITT / DES_TIP_PSS / DAT_RSS_FNT_ITT / DAT_INC_DBO)

→ Todas as planilhas exportadas pelo mySQL e gerenciadas por **python charm**.

2. APLICAÇÃO POR PYTHON (software: PYCHARM): aplicando a análise dos dados por programação.

2.1 importando biblioteca e criando link com o banco de dados



```
1 import pymysql
2
3 conexao = pymysql.connect(
4     host='localhost',
5     user='root',
6     passwd='',
7     database='spc_db'
8 )
```

2.2 criando cursos com comandos nativos do MySQL e adicionando as tabelas para uma lista simulação do banco de dados do SPC.

```
programacao.py x
10 cursor = conexao.cursor()
11 fonte = "select * from fonte"
12 modalidade = "select * from modalidade"
13 movimento = "select * from movimento"
14 operacao = "select * from operacao"
15 pagamento = "select * from pagamento"
16
17 lista_spc_db = [fonte, modalidade, movimento, operacao, pagamento]
18
```

2.3 criando funções para analisar os dados recebidos.

2.3.1 verificando a quantidade de dígitos do CNPJ

```
programacao.py x
20 # Funções:
21
22 # Função: Verifica a quantidade de dígitos do CNPJ:
23 def digitos_cnpj(lista):
24     CNPJ = []
25     for z in lista:
26         if len(z[1]) == 8:
27             if len(z[2]) <= 6:
28                 complemento = str(z[2])
29                 while len(complemento) < 6:
30                     complemento = '0' + complemento
31                 cnpj_pessoa = str(z[0]) + "|" + str(z[1]) + "-" + complemento
32                 CNPJ.append(cnpj_pessoa)
33     return CNPJ
```

2.3.2 contabilizando CNPJs duplicados

```
programacao.py x
37 # Função: Contabilizando os duplicados (Função específica para o CNPJ):
38 def duplicados_cnpj(lista):
39     duplicado_id = []
40     correto_id = []
41     correto = []
42
43     for z in lista:
44         CNPJ = z[4:]
45         if CNPJ not in correto:
46             correto.append(CNPJ)
47             correto_id.append(z)
48         else:
49             duplicado_id.append(z)
50     return duplicado_id
```

2.3.3 verificando duplicados

```
programacao.py x
54 # Função para verificar os duplicados:
55 def duplicados(lista):
56     duplicado = []
57     correto = []
58
59     for z in lista:
60         if z not in correto:
61             correto.append(z)
62         else:
63             duplicado.append(z)
64     return duplicado
65
```

2.3.4 verificando campos vazios

```
programacao.py x
68 # Função para verificar se o campo esta nulo:
69 def nulo(lista):
70     nulo = []
71
72     for z in lista:
73         for y in z:
74
75             # Programação para a tabela fonte, modalidade e pagamento:
76             if x == fonte or x == modalidade or x == pagamento:
77                 if y == "":
78                     nulo.append(str(z[0]))
79
80             # Programação para a tabela movimento:
81             if x == movimento:
82                 if y == "" and not y[9] == "E01" and not y[9] == "D01":
83                     nulo.append(str(z[0]))
84
85             # Programação para a tabela operação:
86             if x == operacao:
87                 if y == "" and not y[7] == "C01":
88                     nulo.append(str(z[0]))
89     return nulo
```

2.3.5 valores da tabela movimento e operação

```
programacao.py x
93 # Função para conta os valores da tabela movimento e operação:
94 def contar_quantidade(lista):
95     modalidade_dicionario = {
96         'A01': [0, 0],
97         'A02': [0, 0],
98         'A04': [0, 0],
99         'A05': [0, 0],
100        'A99': [0, 0],
101        'B01': [0, 0],
102        'B02': [0, 0],
103        'B03': [0, 0],
104        'B04': [0, 0],
105        'B05': [0, 0],
106        'B06': [0, 0],
107        'B07': [0, 0],
108        'B99': [0, 0],
109        'C01': [0, 0],
110        'D01': [0, 0],
111        'E01': [0, 0],
112        'E02': [0, 0],
113        'F01': [0, 0],
114        'G01': [0, 0],
115    }
116    if x == operacao:
117        for y in lista:
118            if modalidade_dicionario[y[7]]:
119                modalidade_dicionario[y[7]][0] += int(y[4])
120                modalidade_dicionario[y[7]][1] += int(y[5])
121    return modalidade_dicionario
```

2.3.6 quantidade de movimentações da tabela operação e pagamento

```
programacao.py x
124 # Somando a quantidade de movimentações da tabela operação e pagamento:
125 def somar_quantidade(dicionario, lista):
126     if x == pagamento:
127         for y in lista:
128             if dicionario[y[3]]:
129                 dicionario[y[3]][0] += int(y[4])
130                 dicionario[y[3]][1] += int(y[5])
131     return dicionario
```

2.4 iniciando análise dos dados

2.4.1 tabela FONTE

```
programacao.py x
135 # Início da análise:
136 print("\nAnálise referente a base de dados da SPC Brasil: \n\n")
137
138 # Verificando as tabelas:
139 for x in lista_spc_db:
140     cursor.execute(x)
141
142     # Tabela Fonte:
143     if x == fonte:
144
145         # Declarando variáveis:
146         linha_fonte = []
147         cnpj_correto = []
148         cnpj_duplicados_id = []
149         complemento = ""
150         fonte_nulo = []
151
152         for y in cursor:
153             linha_fonte.append(y)
154
155             # Verificando se o cnpj possui 14 dígitos
156             cnpj_correto = digitos_cnpj(linha_fonte)
157
158             # Verificando se o cnpj é duplicado:
159             cnpj_duplicados_id = duplicados_cnpj(cnpj_correto)
160
161             # Verificando se a tabela possui um campo nulo:
162             fonte_nulo = nulo(linha_fonte)
163
164         print(f'''Tabela Fonte:
165
166         Número de cnpj corretos: {len(cnpj_correto)}
167         Total de cnpj's fora do padrão: {len(linha_fonte) - len(cnpj_correto)}
168         Total de cnpj's duplicados: {len(cnpj_duplicados_id)}
169         Total de campos nulos: {len(fonte_nulo)}
170         Total de cnpj's: {len(linha_fonte)} \n\n''')
```


2.4.2 tabela MODALIDADE

```
programacao.py x
176     # Tabela Modalidade:
177     if x == modalidade:
178
179         # Declarando variaveis:
180         linha_modalidade = []
181         modalidade_nulo = []
182         modalidade_duplicado = []
183
184         for y in cursor:
185             linha_modalidade.append(y)
186
187         # Verificando se a tabela possui algum campo nulo:
188         modalidade_nulo = nulo(linha_modalidade)
189
190         # Verificando se a tabela possui valores duplicados:
191         modalidade_duplicado = duplicados(linha_modalidade)
192
193         print(f'''Tabela Modalidade:
194
195     Total de campos nulos: {len(modalidade_nulo)}
196     Total de valores duplicados: {len(modalidade_duplicado)}
197     Total de linhas verificadas: {len(linha_modalidade)} \n\n''')
```

2.4.3 tabela MOVIMENTO

```
programacao.py x
203     # Tabela Movimento:
204     if x == movimento:
205
206         # Declarando as Variaveis:
207         linha_movimento = []
208         movimento_nulo = []
209         movimento_duplicado = []
210
211         for y in cursor:
212             linha_movimento.append(y)
213
214         # Verificando se a tabela possui algum campo nulo:
215         movimento_nulo = nulo(linha_movimento)
216
217         # Verificando se a tabela possui valores duplicados:
218         movimento_duplicado = duplicados(linha_movimento)
219
220         print(f'''Tabela Movimento:
221
222     Total de Campos nulos: {len(movimento_nulo)}
223     Total de valores duplicados: {len(movimento_duplicado)}
224     Total de linhas verificadas: {len(linha_movimento)} \n\n''')
```

2.4.4 tabela OPERAÇÃO

```
programacao.py x
230     # Tabela Operacao:
231     if x == operacao:
232
233         # Declarando as variaveis:
234         operacao_nulo = []
235         operacao_duplicado = []
236         operacao_quantidade = {}
237         linha_operacao = []
238
239         for y in cursor:
240             linha_operacao.append(y)
241
242         # Verificando se a tabela possui algum campo nulo:
243         operacao_nulo = nulo(linha_operacao)
244
245         # Verificando se a tabela possui valores duplicados:
246         operacao_duplicado = duplicados(linha_operacao)
247
248         # Verificando a quantidade de movimentações:
249         operacao_quantidade = contar_quantidade(linha_operacao)
250
251         print(f'''Tabela Operação:
252
253         Total de Campos nulos: {len(operacao_nulo)}
254         Total de valores duplicados: {len(operacao_duplicado)}
255         Total de linhas verificadas: {len(linha_operacao)} \n\n''')
256
```

2.4.5 tabela PAGAMENTO

```
programacao.py x
261 # Tabela Pagamento:
262 if x == pagamento:
263
264     # Declarando as Variaveis:
265     pagamento_nulo = []
266     pagamento_duplicado = []
267     pagamento_quantidade = {}
268     linha_pagamento = []
269
270     for y in cursor:
271         linha_pagamento.append(y)
272
273     # Verificando se a tabela possui algum campo nulo:
274     pagamento_nulo = nulo(linha_pagamento)
275
276     # Verificando se a tabela possui algum campo duplicado:
277     pagamento_duplicado = duplicados(linha_pagamento)
278
279     # Verificando a quantidade movimentações:
280     pagamento_quantidade = somar_quantidade(operacao_quantidade, linha_pagamento)
281
282     print(f'''Tabela Pagamento:
283
284     Total de Campos nulos: {len(pagamento_nulo)}
285     Total de valores duplicados: {len(pagamento_duplicado)}
286     Total de linhas verificadas: {len(linha_pagamento)} \n\n''')
287
```