dim_year

```python
import pandas as pd
# initialize list of lists
data = [[1, 2018], [2, 2019], [3, 2020], [4, 2021]]

# Create the pandas DataFrame
dim_year = pd.DataFrame(data, columns=['year_key', 'year'])
dim_year
```

Python

dim_month

```python
import pandas as pd
import random
from datetime import datetime, timedelta
from math import ceil

start_dt = datetime.strptime('01/31/18', '%m/%d/%y')
end_dt = datetime.strptime('12/31/21', '%m/%d/%y')
# list of dates
dates = []
# delta ot time to advanced on each loop
delta = timedelta(days=30)
loop_date = start_dt

while loop_date <= end_dt:
    # add loop_date with all attributes to list
    row = {
        "month": loop_date.month,
        "month_name": loop_date.strftime('%B'),
        "quarter": ceil(loop_date.month / 3),
        "year": loop_date.year
          }
    dates.append(row)
    # increment start date by timedelta
    loop_date += delta
df_dim_year = pd.DataFrame(dates)
df_dim_year.insert(0, 'month_key', range(1, 1 + len(df_dim_year)))
df_dim_year
```

Python

SQL_LOAD_FT_PIB_MUN_SP_Y

```
SELECT m.municipio_key,
y.year_key,
u.valor_pib,
current_date() as load_date
FROM tb_5938_pib_mun_sp_unpivot u
inner join dim_municipio m
 on(u.Cod_Mun = m.id)
inner join dim_year y
on(y.year=u.ano)
```

DuckDB

O **SQL_LOAD_FT_PIB_MUN_SP_Y** prepara a tabela fato

jccorrea

Tabela **FATO** tradicional com PIB DOS MUNICIPIOS

jccorrea

tb_5938_pib_mun_sp_unpivot

```
SELECT CAST("Cod_mun" AS INTEGER) cod_mun,
"EnteFederativo" as Nome,
ano,
cast(valor_pib as decimal) valor_pib
 FROM
(
UNPIVOT tb_5938_pib_mun_sp
ON 2018, 2019, 2020, 2021
INTO
    NAME ano
    VALUE valor_pib
)
```

cod_mun	Nome	ano	valor_pib
3500105	Adamantina (SP)	2018	1070910
3500105	Adamantina (SP)	2019	1191134
3500105	Adamantina (SP)	2020	1243768
3500105	Adamantina (SP)	2021	1370866
3500204	Adolfo (SP)	2018	92203
3500204	Adolfo (SP)	2019	112493

DuckDB 129 ms (Just now) 4 columns · 2,580 rows

Unpivot para facilitar a criação da tabela fato

jccorrea

Tabela intermediária ou temporária para carga de dados

jccorrea

tb_5938_pib_mun_sp

```
SELECT replace("Cod",'"','') Cod_mun,
 replace(t."EnteFederativo",'"','') EnteFederativo,
 replace(t."2018",'"','') as "2018",
 replace(t."2019",'"','') as "2019",
 replace(t."2020",'"','') as "2020",
 replace(t."2021",'"','') as "2021"
FROM tabela5938_clean_file t
where t.Nivel= '"MU"'
and substring(t."EnteFederativo",-4,2) ='SP'
```

Cod_mun	EnteFederativo	2018	2019	2020	2021
3500105	Adamantina (SP)	1070910	1191134	1243768	1370866
3500204	Adolfo (SP)	92203	112493	136758	115490
3500303	Aguaí (SP)	972613	1046725	1143458	1314737

DuckDB 15 ms (Just now) 6 columns · 645 rows

Realizei uma segunda limpeza via SQL para depois criar uma espécie de **tabela fato ou totalizadora/agg.**

jccorrea

PIB por Mesorregião - SP

GRAFICO PIB POR MESORRGIÃO

MOSTRA A SOMA DO PIB POR ANO PARA DA

jccorrea

TOT_PIB_MUN_SP_MESORREGIAO_LAST_4_Y

```
SELECT me.nome,
u.ano,
sum(u.valor_pib) valor_pib
FROM tb_5938_pib_mun_sp_unpivot u
inner join dim_municipio m
 on(u.Cod_Mun = m.id)
inner join dim_microrregiao mi
 on(mi.id = m.microrregiao_id)
inner join dim_mesorregiao me
 on(me.id = mi.mesorregiao_id)
 group by me.nome, u.ano
```

DuckDB

TABELA **TOTALIZADORA** DE PIB por MESO MUNICIPIOS EM NOS ULTIMOS 4 ANOS

jccorrea

Relacionando informações de População com Localidades. Dados de APIs diferentes integrados

jccorrea

TB_MUN_MAIS_POP_MESO

```
SELECT municipio, mesorregiao,populacao_2022
FROM "TB_MUN_RANK_BY_POP"
WHERE "RANK_POP_MUN" = 1
Order by 3 desc
```

DuckDB

TB_MUN_RANK_BY_POP

```
SELECT m.nome as municipio,
       me.nome as mesorregiao,
       cast(serie_2022 as integer) as populacao_2022,
      RANK() OVER (PARTITION BY me.nome ORDER BY cast(serie_2022 as integer) DESC)
AS RANK_POP_MUN
 FROM populacao_residente p
 inner join dim_municipio m on (p.localidade_id = m.id)
 inner join dim_microrregiao mi on (m.microrregiao_id = mi.id )
 inner join dim_mesorregiao me on ( mi.mesorregiao_id = me.id)
```

DuckDB

Existem APIs para região, macroregião e etc, mas o intuito aqui foi demonstrar atraves de uma única API a questão da **abstração na modelagem de dados snow flake**

jccorrea

dim_municipio

```python
import pandas as pd
import json
# create pandas df from municipios_sp object
# id, nome, microrregiao, regiao-imediata

#define source to DIM_MUNICIPIO
municipios_df = pd.DataFrame(municipios_sp_data)
df_municipios_sp = municipios_df[["id","nome","microrregiao"]]
df_municipios_sp = pd.json_normalize(municipios_sp_data, errors='ignore',
max_level=1)
df_municipios_sp = df_municipios_sp[["id","nome","microrregiao.id"]]
df_municipios_sp.columns = ["id","nome", "microrregiao_id"]
#generate a simple blind key
df_municipios_sp.insert(0, 'municipio_key', range(1, 1 + len(df_municipios_sp)))
#show
df_municipios_sp
```

Python

dim_microrregiao

```python
import pandas as pd
# create pandas df from municipios_sp object
# id, nome, microrregiao, regiao-imediata

#define source to DIM_MICRORREGIAO
municipios_df = pd.DataFrame(municipios_sp_data)
df_microregiao = pd.json_normalize(municipios_df.microrregiao,max_level=1)
df_microregiao = df_microregiao[["id","nome","mesorregiao.id"]]
df_microregiao.columns = ["id","nome", "mesorregiao_id"]
df_microregiao = df_microregiao.drop_duplicates()
#generate a simple blind key
df_microregiao.insert(0, 'microrregiao_key', range(1, 1 + len(df_microregiao)
))
#show
df_microregiao
```

Python

dim_mesorregiao

```python
import pandas as pd
# create pandas df from municipios_sp object
# id, nome, microrregiao, regiao-imediata

#define source to DIM_MICRORREGIAO
municipios_df = pd.DataFrame(municipios_sp_data)
df_mesorregiao = pd.json_normalize(municipios_df.microrregiao,max_level=2)
df_mesorregiao = df_mesorregiao[["mesorregiao.id","mesorregiao.nome","mesorregiao.UF.
id"]]
df_mesorregiao.columns = ["id","nome","uf_id"]
df_mesorregiao = df_mesorregiao.drop_duplicates()
#generate a simple blind key
df_mesorregiao.insert(0, 'mesorregiao_key', range(1, 1 + len(df_mesorregiao)))
#show
df_mesorregiao
```

Python

Essa dimensão por conter dados apenas de SP, vai ficar apenas com uma linha. Mas aqui podemos armazenar os dados das outras UFs

jccorrea

dim_uf

```python
import pandas as pd
# create pandas df from municipios_sp object
# id, nome, microrregiao, regiao-imediata

#define source to DIM_MICRORREGIAO
municipios_df = pd.DataFrame(municipios_sp_data)
df_uf = pd.json_normalize(municipios_df.microrregiao,max_level=1)
df_uf = pd.json_normalize(df_uf["mesorregiao.UF"],max_level=1)
df_uf = df_uf[["id","sigla","nome","regiao.id"]]
df_uf.columns = ["id","sigla","nome","regiao_id"]
df_uf = df_uf.drop_duplicates()
df_uf.insert(0, 'uf_key', range(1, 1 + len(df_uf)))
#show
```

Python

dim_regiao

```python
import pandas as pd
# create pandas df from municipios_sp object
# id, nome, microrregiao, regiao-imediata

#define source to dim_regiao

municipios_df = pd.DataFrame(municipios_sp_data)
df_regiao = pd.json_normalize(municipios_df.microrregiao,max_level=1)
df_regiao = pd.json_normalize(df_regiao["mesorregiao.UF"],max_level=1)
df_regiao = df_regiao[["regiao.id","regiao.sigla","regiao.nome"]]
df_regiao.columns = ["id","sigla","nome"]
df_regiao = df_regiao.drop_duplicates()
df_regiao.insert(0, 'regiao_key', range(1, 1 + len(df_regiao)))
```

Python

populacao_residente

```python
import pandas as pd
import json
from flatten_json import flatten
# create pandas df from censo demografico (json object)

#define source to Pop Residente
df_master_populacao_residente = pd.DataFrame(pop_residente_mun_sp_data)
df_populacao_residente = df_master_populacao_residente[["id","variavel","unidade","resultados"]]
#json normalize
#working v1
#cria  column classificacoes e series(dados)
#df1_populacao_residente = pd.json_normalize(pop_residente_mun_sp_data, record_path=['resultados'],
errors="ignore", max_level=1)

df1_populacao_residente = pd.json_normalize(pop_residente_mun_sp_data, ['resultados','series'],
errors="ignore" )
df1_populacao_residente.columns = ["localidade_id","nivel_id","nivel_nome","localidade_nome",
"serie_2022"]
df1_populacao_residente
```

Python

Transformação na esrtutura dos dados em json para flat , para ser usados no SQL

jccorrea

municipios_sp

```python
#modules
import requests
import json

# define OIBGE URL
url_localidade_mun_sp = "https://servicodados.ibge.gov.br/api/v1/localidades/estados/SP/
municipios"

#calling IBGE URL
headers = {'Content-Type': "application/json"}
body = json.dumps({})

response = None
try:
  response = requests.get(url_localidade_mun_sp, data=body, headers=headers)
except Exception as e:
  print(e)
if response != None and response.status_code == 200:
  print(response.url)
  municipios_sp_data = json.loads(response.text)
  #municipios_sp_json_data = response.json()
  print(municipios_sp_data)
  #print(municipios_sp_json_data)
```

</> Python

tabela5938_clean_file

tabela5938_clean_file.csv

column0	Nivel	Cod	EnteFederativo	2018 ↑	2019	2020	2021
837	"MU"	"2207108"	"Olho D'Água do Piauí (PI)"	"20204"	"23544"	"26002"	"28731"
1334	"MU"	"2504850"	"Coxixola (PB)"	"20235"	"20762"	"22527"	"24017"
460	"MU"	"1720150"	"São Félix do Tocantins (TO)"	"20810"	"23269"	"27244"	"36541"
448	"MU"	"1718659"	"Rio da Conceição (TO)"	"21080"	"23178"	"25567"	"29142"
776	"MU"	"2203859"	"Floresta do Piauí (PI)"	"21555"	"22023"	"24381"	"25258"
708	"MU"	"2200954"	"Aroeiras do Itaim (PI)"	"21585"	"22661"	"25690"	"24625"
1167	"MU"	"2405900"	"João Dias (RN)"	"21749"	"24159"	"26179"	"27991"
896	"MU"	"2210383"	"São Miguel da Baixa Grande (PI)"	"21933"	"23399"	"25349"	"27272"
1341	"MU"	"2505303"	"Curral Velho (PB)"	"22134"	"23362"	"24919"	"26783"
2848	"MU"	"3149408"	"Pedro Teixeira (MG)"	"22265"	"22934"	"26480"	"30268"
806	"MU"	"2205540"	"Lagoinha do Piauí (PI)"	"22299"	"24643"	"37321"	"40519"
1425	"MU"	"2512606"	"Quixaba (PB)"	"22604"	"23064"	"23491"	"26746"
3046	"MU"	"3165800"	"Senador José Bento (MG)"	"22675"	"22609"	"28497"	"35496"

DuckDB · 11 ms (Just now) · 8 columns · 5,598 rows

populacao_residente_sp_2022

```python
#modules
import requests
import json

# define URL IBGE
url_pop_mun_sp = "https://servicodados.ibge.gov.br/api/v3/agregados/9520/periodos/-6/variaveis/93?localidades=N6[N3[35]]"

#calling IBGE URL
headers = {'Content-Type': "application/json"}
body = json.dumps({})

response = None
try:
  response = requests.get(url_pop_mun_sp, data=body, headers=headers)
except Exception as e:
  print(e)
if response != None and response.status_code == 200:
  print(response.url)
  pop_residente_mun_sp_data = json.loads(response.text)
  print(pop_residente_mun_sp_data)
```

Python

Ingestão de dados via API usando python(requests,pandas, json). API: https://servicodados.ibge.gov.br/api/v3/agregados/9520/periodos/-6/variaveis/93?localidades=N6[N3[35]]

jccorrea

Censo demográfico

Período: 2022

População residente (1522)

jccorrea

Em um outro código, os parametros da URL podem ser dinâmicos para servir para reutilização de código. Aqui está hardcoded apenas para estudo. Poderiamos ter um local armazenado com as URLs e sua identificação.

jccorrea