

Análise de Dados Multidimensionais e Processo ETL

Aluno: Marco Leone Merini

Professor: Luiz C. Camargo

Disciplina: Análise Preditiva – ED

Curso: Engenharia de Software

1. Introdução

Este documento descreve a implementação de um sistema de análise multidimensional de dados e o processo ETL (Extração, Transformação e Carga) para dados de vendas. O projeto foi desenvolvido utilizando PostgreSQL e Python para geração e processamento de dados sintéticos.

2. Análise Multidimensional de Dados

2.1 Schema Implementado

Foi implementado um esquema Schema para análise de dados educacionais com as seguintes tabelas:

- **Estudante** (estudanteID, nome, curso)
- **Instrutor** (instrutorID, curso)
- **Aula** (aulaID, Instituicao, cidade, estado)
- **Aulas_assistidas** (estudanteID, instrutorID, aulaID, notas) - Tabela fato

2.2 Consultas Analíticas Implementadas

a) Alunos com notas acima de 70 em SC com instrutor de curso diferente

```

SELECT E.nome, A.instituicao, AA.notas
FROM Aulas_assistidas AA
JOIN Estudante E ON AA.estudanteID = E.estudanteID
JOIN Instrutor I ON AA.instrutorID = I.instrutorID
JOIN Aula A ON AA.aulaID = A.aulaID
WHERE A.estado = 'Santa Catarina'
AND E.curso <> I.curso
AND AA.notas > 70;

```

	nome text	instituicao text	notas numeric
1	Alice	UFSC	85
2	Daniel	UDESC	90

b) Médias por aluno e instrutor em Joinville

```

SELECT E.nome, I.instrutorID, AVG(AA.notas) AS media
FROM Aulas_assistidas AA
JOIN Estudante E ON AA.estudanteID = E.estudanteID
JOIN Instrutor I ON AA.instrutorID = I.instrutorID
JOIN Aula A ON AA.aulaID = A.aulaID
WHERE A.cidade = 'Joinville'
GROUP BY E.nome, I.instrutorID;

```

	nome text	instrutorid integer	media numeric
1	Bruno	4	72.0000000000000000
2	Daniel	3	90.0000000000000000

c) Rollup por instrutor

```

SELECT I.instrutorID, AVG(AA.notas) AS media
FROM Aulas_assistidas AA
JOIN Estudante E ON AA.estudanteID = E.estudanteID

```

```

JOIN Instrutor I ON AA.instrutorID = I.instrutorID
JOIN Aula A ON AA.aulaID = A.aulaID
WHERE A.cidade = 'Joinville'
GROUP BY ROLLUP (I.instrutorID);

```

	InstrutorID [PK] integer	media numeric
1	[default]	81.0000000000000000
2	3	90.0000000000000000
3	4	72.0000000000000000

d) Média por curso do estudante

```

SELECT E.curso, AVG(AA.notas) AS media
FROM Aulas_assistidas AA
JOIN Estudante E ON AA.estudanteID = E.estudanteID
GROUP BY E.curso;

```

	curso text	media numeric
1	Matemática	77.0000000000000000
2	Direito	70.3333333333333333
3	Medicina	74.6666666666666667
4	Engenharia	81.2000000000000000

e) Drill down por curso do instrutor e estudante

```

SELECT E.curso AS curso_estudante, I.curso AS curso_instrutor, AVG(A
A.notas) AS media
FROM Aulas_assistidas AA
JOIN Estudante E ON AA.estudanteID = E.estudanteID
JOIN Instrutor I ON AA.instrutorID = I.instrutorID
GROUP BY E.curso, I.curso;

```

	curso_estudante text	curso_instrutor text	media numeric
1	Direito	Medicina	74.0000000000000000
2	Matemática	Engenharia	77.0000000000000000
3	Engenharia	Matemática	83.0000000000000000
4	Medicina	Direito	70.0000000000000000
5	Medicina	Medicina	72.0000000000000000
6	Engenharia	Engenharia	92.0000000000000000
7	Direito	Direito	69.0000000000000000
8	Medicina	Matemática	82.0000000000000000
9	Engenharia	Direito	90.0000000000000000
10	Engenharia	Medicina	58.0000000000000000
11	Direito	Engenharia	68.0000000000000000

f) WITH ROLLUP por granularidades geográficas

```
SELECT A.estado, A.cidade, A.instituicao, AVG(AA.notas) AS media
FROM Aulas_assistidas AA
JOIN Aula A ON AA.aulaID = A.aulaID
GROUP BY ROLLUP (A.estado, A.cidade, A.instituicao);
```

	estado text	cidade text	instituicao text	media numeric
1	[null]	[null]	[null]	76.5000000000000000
2	Santa Catarina	Florianópolis	UFSC	76.5000000000000000
3	São Paulo	Bauru	UNESP	75.5000000000000000
4	Paraná	Londrina	UTFPR	75.5000000000000000
5	Santa Catarina	Joinville	UDESC	81.0000000000000000
6	Paraná	Curitiba	UFPR	67.5000000000000000
7	São Paulo	São Paulo	USP	83.0000000000000000
8	Paraná	Londrina	[null]	75.5000000000000000
9	Paraná	Curitiba	[null]	67.5000000000000000
10	Santa Catarina	Florianópolis	[null]	76.5000000000000000
11	Santa Catarina	Joinville	[null]	81.0000000000000000
12	São Paulo	São Paulo	[null]	83.0000000000000000
13	São Paulo	Bauru	[null]	75.5000000000000000
14	Paraná	[null]	[null]	71.5000000000000000
15	Santa Catarina	[null]	[null]	78.7500000000000000
16	São Paulo	[null]	[null]	79.2500000000000000

g) CUBE para análise multidimensional

```
SELECT A.estado, A.cidade, A.instituicao, AVG(AA.notas) AS media
FROM Aulas_assistidas AA
JOIN Aula A ON AA.aulaID = A.aulaID
GROUP BY CUBE (A.estado, A.cidade, A.instituicao);
```

	estado text	cidade text	institucao text	media numeric
1	[null]	[null]	[null]	76.500000000000000000
2	Santa Catarina	Florianópolis	UFSC	76.500000000000000000
3	São Paulo	Bauru	UNESP	75.500000000000000000
4	Paraná	Londrina	UTFPR	75.500000000000000000
5	Santa Catarina	Joinville	UDESC	81.000000000000000000
6	Paraná	Curitiba	UFPR	67.500000000000000000
7	São Paulo	São Paulo	USP	83.000000000000000000
8	Paraná	Londrina	[null]	75.500000000000000000
9	Paraná	Curitiba	[null]	67.500000000000000000
10	Santa Catarina	Florianópolis	[null]	76.500000000000000000

3. Implementação do Ambiente de Dados

3.1 Banco de Dados de Vendas

Foi criado um banco de dados PostgreSQL chamado "vendas" com a seguinte estrutura:

```
CREATE TABLE vendas (
  id_venda integer PRIMARY KEY,
  data_venda date,
  id_produto integer,
  categoria character varying(50),
  regioao character varying(50),
  quantidade integer,
  valor_unitario numeric(10,2),
  valor_total numeric(10,2),
  canal character varying(50)
);
```

3.2 Inserção de Dados

Foram inseridas 500 amostras de dados sintéticos de vendas, conforme verificado pela consulta:

```
SELECT COUNT(*) FROM vendas; -- Retorna 500 registros
```

```
vendas=# SELECT COUNT(*) FROM vendas;
count
-----
    500
(1 linha)
```

4. Processo ETL para Dados de Vendas

4.1 Extração (Extract)

Os dados brutos foram gerados sinteticamente através de um script Python que:

- Define categorias, regiões e canais de venda
- Gera datas aleatórias dentro de um período
- Simula uma queda nas vendas após o terceiro mês
- Gera valores unitários e quantidades aleatórias

```
def extract_data():
    categorias = ['Eletrônicos', 'Vestuário', 'Alimentos', 'Brinquedos']
    regioes = ['Sul', 'Sudeste', 'Norte', 'Centro-Oeste']
    canais = ['Loja Física', 'Online', 'Telefone']

    dados_brutos = []
    data_base = datetime(2024, 1, 1)

    for i in range(500):
        data_venda = data_base + timedelta(days=random.randint(0, 120))
        mes = data_venda.month
        quantidade = random.randint(5, 20) if mes <= 3 else random.randint(1, 10)
        valor_unitario = round(random.uniform(20.0, 300.0), 2)

        dados_brutos.append({
            'id_venda': i + 1,
```

```
        'data_venda': data_venda,  
        'id_produto': random.randint(100, 199),  
        'categoria': random.choice(categorias),  
        'regiao': random.choice(regioes),  
        'quantidade': quantidade,  
        'valor_unitario': valor_unitario,  
        'canal': random.choice(canais)  
    })  
  
    return dados_brutos
```

4.2 Transformação (Transform)

Os dados brutos foram transformados através de:

- Cálculo do valor total (quantidade × valor unitário)
- Formatação da data para o padrão ISO (YYYY-MM-DD)

```
def transform_data(dados_brutos):  
    dados_transformados = []  
  
    for registro in dados_brutos:  
        valor_total = round(registro['quantidade'] * registro['valor_unitario'], 2)  
        data_formatada = registro['data_venda'].strftime('%Y-%m-%d')  
  
        dados_transformados.append({  
            'id_venda': registro['id_venda'],  
            'data_venda': data_formatada,  
            'id_produto': registro['id_produto'],  
            'categoria': registro['categoria'],  
            'regiao': registro['regiao'],  
            'quantidade': registro['quantidade'],  
            'valor_unitario': registro['valor_unitario'],  
            'valor_total': valor_total,  
            'canal': registro['canal']  
        })
```



```
return dados_transformados
```

4.3 Carga (Load)

Os dados transformados foram carregados no PostgreSQL através de:

- Conexão com o banco de dados
- Inserção registro a registro
- Commit da transação

```
def load_data(dados_transformados):
    df = pd.DataFrame(dados_transformados)
    try:
        conn = psycopg2.connect(
            dbname="vendas",
            user="postgres",
            password="123456",
            host="localhost",
            port="5432"
        )
        cursor = conn.cursor()

        for _, row in df.iterrows():
            cursor.execute("""
                INSERT INTO vendas
                (id_venda, data_venda, id_produto, categoria, regioao, quantidade, va
                VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)
            """, tuple(row))

        conn.commit()
        print("Dados carregados com sucesso no PostgreSQL!")
    except Exception as e:
        print(f"Erro ao carregar dados: {e}")
    finally:
```

```
if conn:
    conn.close()
```

5. Análises Adicionais sobre os Dados de Vendas

Foram implementadas consultas analíticas sobre os dados de vendas:

5.1 Rollup por região e categoria

```
SELECT regioao, categoria, SUM(valor_total) AS total_vendas
FROM vendas
GROUP BY ROLLUP(regiao, categoria)
ORDER BY regioao, categoria;
```

	regiao character varying (50) 🔒	categoria character varying (50) 🔒	total_vendas numeric 🔒
1	Centro-Oeste	Alimentos	37798.82
2	Centro-Oeste	Brinquedos	56748.08
3	Centro-Oeste	Eletrônicos	36258.01
4	Centro-Oeste	Vestuário	76696.69
5	Centro-Oeste	[null]	207501.60
6	Norte	Alimentos	73975.39
7	Norte	Brinquedos	54792.55

5.2 Rollup por canal e mês

```
SELECT
    canal,
    EXTRACT(MONTH FROM data_venda) AS mes,
    SUM(valor_total) AS total_vendas
FROM vendas
GROUP BY ROLLUP(canal, mes)
ORDER BY canal, mes;
```

	canal character varying (50)	mes numeric	total_vendas numeric
1	Loja Física	1	105735.75
2	Loja Física	2	77866.48
3	Loja Física	3	93632.06
4	Loja Física	4	27555.88
5	Loja Física	[null]	304790.17
6	Online	1	97630.27
7	Online	2	59869.56

5.3 Top 5 produtos por receita

```
SELECT id_produto, SUM(valor_total) AS receita_total
FROM vendas
GROUP BY id_produto
ORDER BY receita_total DESC
LIMIT 5;
```

	id_produto integer	receita_total numeric
1	116	23730.66
2	134	21157.20
3	143	19802.15
4	186	19544.09
5	109	19349.51

5.4 CUBE por região e canal

```
SELECT regioao, canal, SUM(valor_total) AS total_vendas
FROM vendas
GROUP BY CUBE(regiao, canal)
ORDER BY regioao, canal;
```

	regiao character varying (50) 🔒	canal character varying (50) 🔒	total_vendas numeric 🔒
1	Centro-Oeste	Loja Física	76824.34
2	Centro-Oeste	Online	73694.95
3	Centro-Oeste	Telefone	56982.31
4	Centro-Oeste	[null]	207501.60
5	Norte	Loja Física	86231.19
6	Norte	Online	71642.07
7	Norte	Telefone	83225.27
8	Norte	[null]	241098.53
9	Sudeste	Loja Física	78927.18

6. Conclusão

Este projeto demonstrou:

1. A implementação do schema para análise multidimensional
2. A geração de dados sintéticos com padrões específicos (como queda nas vendas)
3. O processo completo ETL (Extração, Transformação e Carga)
4. A utilização de operadores OLAP como ROLLUP e CUBE
5. Técnicas de análise como drill-down e agregação