

PostgreSQL

Aula 08



Presença

- Linktree: Presente na bio do nosso instagram
- Presença ficará disponível até 1 hora antes da próxima aula
- É necessário 70% de presença para obter o certificado

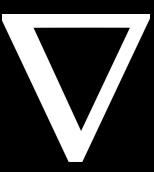


Presença





Processo de ETL: Conceitos





Processo de ETL

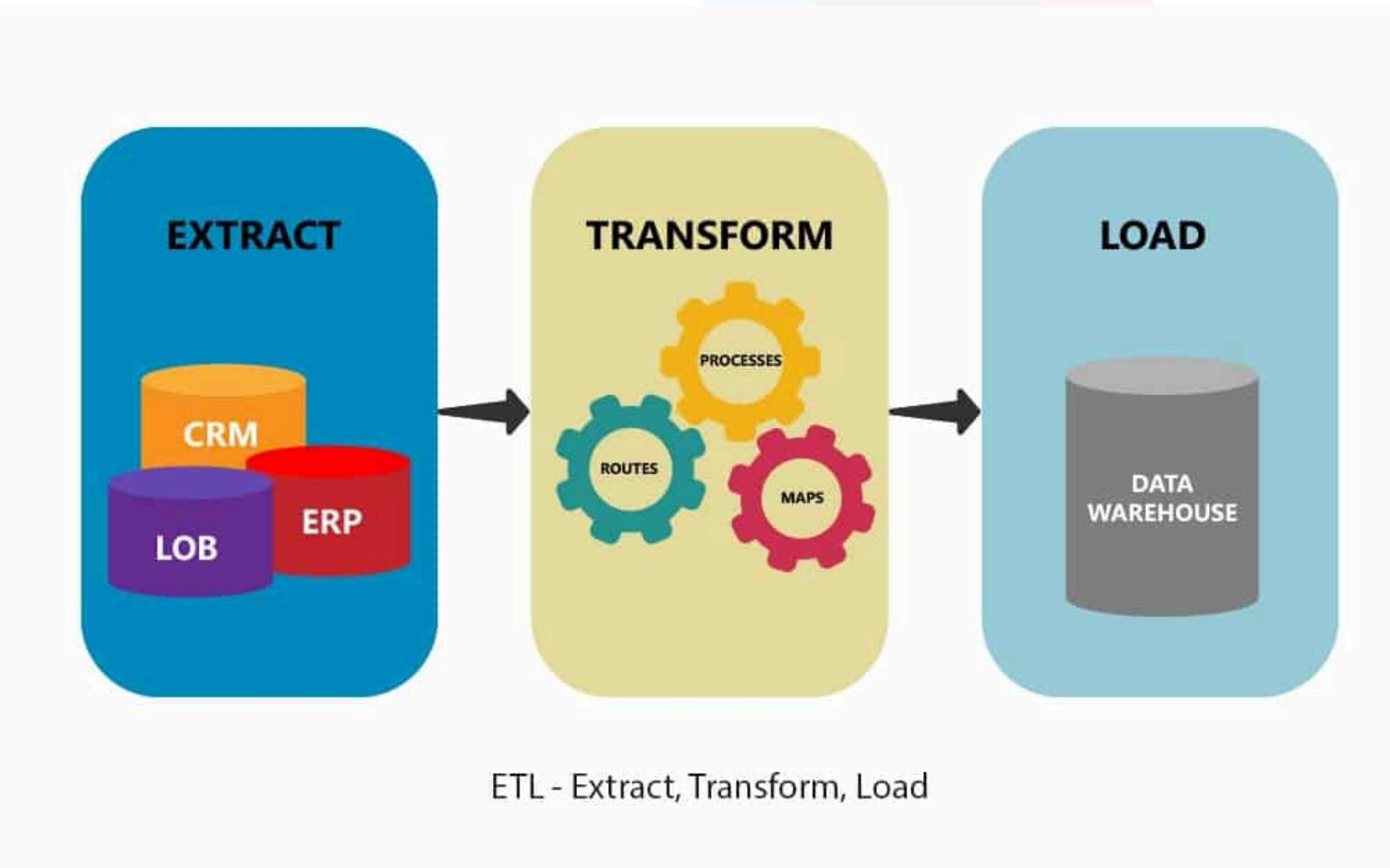
ETL (Extract, Transform, Load) é um processo de captação, limpeza e integração de dados brutos de diferentes fontes

Objetivo

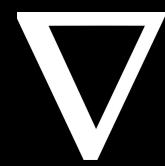
- Preparar conjuntos de fontes diversas de dados para possibilitar análises posteriores



Processo de ETL

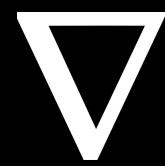


Processo de coleta até o armazenamento é realizado em intervalos periódicos de tempo, normalmente em um determinado horário



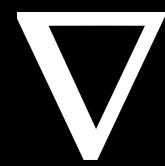
Processo de ETL

- **Extract**
 - Coletar dados de diferentes fontes e armazená-los temporariamente numa área de preparação
 - Banco de dados transacionais, arquivos (CSV), APIs de sistemas web, etc



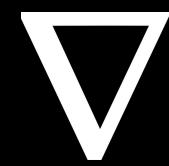
Processo de ETL

- **Transform**
 - Limpeza, padronização, agregação dos dados copiados presentes na área de preparação
 - Tem por objetivo deixar os dados limpos, consistentes e no formato da análise



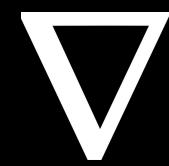
Processo de ETL

- **Transform - Query Ad Hoc x ETL**
 - Lógica de transformação (query) pode ser realizada de duas formas
 - **SQL Sob Demanda:** lógica de transformação é feita no momento da consulta do analista
 - Flexível e rápida a mudanças de lógica de transformação



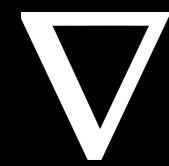
Processo de ETL

- **Transform - Query Ad Hoc x ETL**
 - Lógica de transformação (query) pode ser realizada de duas formas
 - **SQL Sob Demanda:** lógica de transformação é feita no momento da consulta do analista
 - Flexível e rápida a mudanças de lógica de transformação
 - Lenta (JOINS e alto volume de dados) e normalmente presente apenas no computador do analista (menor visibilidade)



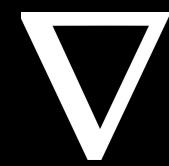
Processo de ETL

- **Transform - Query Ad Hoc x ETL**
 - Lógica de transformação (query) pode ser realizada de duas formas
 - **Pipeline ETL:** lógica de transformação é feita em background em tempos específicos (não é rodada sob demanda pelo analista) e os resultados são salvos em tabelas persistentes
 - Mais rápido (usuário consulta a tabela de resultado e a query é feita em segundo plano), maior visibilidade e confiabilidade (códigos inseridos no Github, por exemplo)



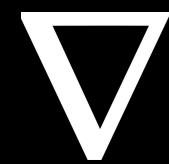
Processo de ETL

- **Transform - Query Ad Hoc x ETL**
 - Lógica de transformação (query) pode ser realizada de duas formas
 - **Pipeline ETL:** lógica de transformação é feita em background em tempos específicos (não é rodada sob demanda pelo analista) e os resultados são salvos em tabelas persistentes
 - Mais rápido (usuário consulta a tabela de resultado e a query é feita em segundo plano), maior visibilidade e confiabilidade (códigos inseridos no Github, por exemplo)
 - Latência da ETL para mostrar resultados novos e menos flexível a mudanças (teste e validação do novo pipeline)



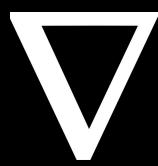
Processo de ETL

- **Load**
 - Armazenamento dos dados preparados para análise no **Data Warehouse**
 - Repositório Central com uma grande quantidade de dados padronizados de diversas origens



Processo de ETL

- **Load**
 - Armazenamento dos dados preparados para análise no **Data Warehouse**
 - Repositório Central com uma grande quantidade de dados padronizados de diversas origens
 - Dados podem ser armazenados por completo, substituindo os dados já presentes (**Load Completo**), ou podem ser armazenados apenas os dados modificados em relação ao último load (**Load Incremental**)



Processo de ETL

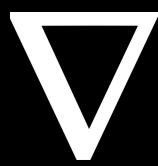
- Na fase de transform (principalmente), podemos criar “funções” que compactam uma query em um determinado nome. Esse nome pode ser reutilizado em códigos seguintes como uma tabela
 - **View:** consulta armazenada. Ao instanciá-la, a consulta é rodada novamente
 - **Materialized View:** armazena o resultado (tabela física) da consulta referenciada
- Materialized view tem maior performance, mas precisam de Refresh periódico

Execução de um Processo de ETL

Cenário de Exemplo

Suponha que temos um cenário de um sistema de clientes e pedidos

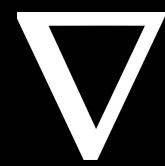
- Fase de Extract: geração e população das tabelas
- Fase de Transform: manipulação dos dados brutos, criando atributos e views relevantes para análise
- Fase de Load: Carregamento dos dados trabalhados



Extract

Suponha que temos um cenário de um sistema de clientes e pedidos

```
1 CREATE TABLE customers (
2     id SERIAL PRIMARY KEY,
3     name VARCHAR(100),
4     email VARCHAR(100),
5     created_at TIMESTAMP DEFAULT NOW()
6 );
7
8 CREATE TABLE orders (
9     id SERIAL PRIMARY KEY,
10    customer_id INT REFERENCES customers(id),
11    order_date TIMESTAMP DEFAULT NOW(),
12    total NUMERIC(10,2)
13 );
```

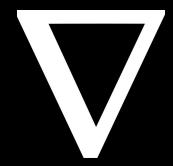


Extract

Suponha que temos um cenário de um sistema de clientes e pedidos

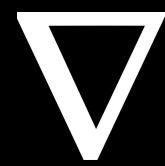
	id [PK] integer	name character varying (100)	email character varying (100)	created_at timestamp without time zone
1	1	Cliente 1	cliente1@exemplo.com	2025-10-31 08:30:11.407207
2	2	Cliente 2	cliente2@exemplo.com	2025-10-31 08:30:11.407207
3	3	Cliente 3	cliente3@exemplo.com	2025-10-31 08:30:11.407207

	id [PK] integer	customer_id integer	order_date timestamp without time zone	total numeric (10,2)
1	1	4	2025-09-02 08:30:11.407207	1939.41
2	2	26	2025-08-10 08:30:11.407207	606.84
3	3	14	2025-09-17 08:30:11.407207	1013.17



Transform

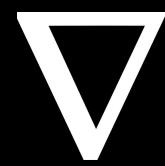
Criação das métricas “total de pedidos” e “valor gasto por cliente”



Transform

Criação das métricas “total de pedidos” e “valor gasto por cliente”

```
1  SELECT
2      c.id AS customer_id,
3      c.name,
4      COUNT(o.id) AS total_pedidos,
5      SUM(o.total) AS valor_total
6  FROM customers c
7  LEFT JOIN orders o ON c.id = o.customer_id
8  GROUP BY c.id, c.name
9  ORDER BY valor_total DESC;
```

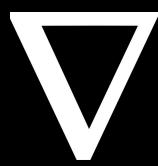


Transform

Criação das métricas “total de pedidos” e “valor gasto por cliente”

```
1  SELECT
2      c.id AS customer_id,
3      c.name,
4      COUNT(o.id) AS total_pedidos,
5      SUM(o.total) AS valor_total
6  FROM customers c
7  LEFT JOIN orders o ON c.id = o.customer_id
8  GROUP BY c.id, c.name
9  ORDER BY valor_total DESC;
```

	customer_id integer	name character varying (100)	total_pedidos bigint	valor_total numeric
1	27	Cliente 27	11	17580.68
2	4	Cliente 4	9	13898.06
3	22	Cliente 22	10	12370.12
4	9	Cliente 9	10	11831.37
5	11	Cliente 11	10	10756.00



Transform

Podemos também criar uma materialized view, caso essa query seja frequentemente usada:



Transform

Podemos também criar uma materialized view, caso essa query seja frequentemente usada:

```
1  -- Criação da Materialized View
2  CREATE MATERIALIZED VIEW customer_summary_mv AS
3  SELECT
4      c.id AS customer_id,
5      c.name,
6      COUNT(o.id) AS total_pedidos,
7      SUM(o.total) AS valor_total
8  FROM customers c
9  LEFT JOIN orders o ON c.id = o.customer_id
10 GROUP BY c.id, c.name;
11
12 -- Atualizar a materialized view quando necessário
13 REFRESH MATERIALIZED VIEW customer_summary_mv;
```

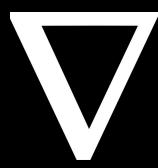


Transform

Podemos também criar uma materialized view, caso essa query seja frequentemente usada:

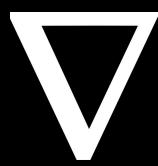
```
1  -- Criação da Materialized View
2  CREATE MATERIALIZED VIEW customer_summary_mv AS
3  SELECT
4      c.id AS customer_id,
5      c.name,
6      COUNT(o.id) AS total_pedidos,
7      SUM(o.total) AS valor_total
8  FROM customers c
9  LEFT JOIN orders o ON c.id = o.customer_id
10 GROUP BY c.id, c.name;
11
12 -- Atualizar a materialized view quando necessário
13 REFRESH MATERIALIZED VIEW customer_summary_mv;
```

	customer_id integer	name character varying (100)	total_pedidos bigint	valor_total numeric
1	22	Cliente 22	9	12716.16
2	11	Cliente 11	1	1346.90
3	9	Cliente 9	6	5167.70
4	15	Cliente 15	7	8183.14
5	26	Cliente 26	7	7801.60
6	19	Cliente 19	4	6040.48
7	30	Cliente 30	3	4314.22
8	21	Cliente 21	4	3954.96
9	3	Cliente 3	10	13988.25
10	17	Cliente 17	6	5616.47
11	28	Cliente 28	6	8264.21
12	5	Cliente 5	11	10422.17
13	29	Cliente 29	8	9412.51



Transform

Uma outra view que podemos fazer é sobre métricas mensais por cliente,



Transform

Uma outra view que podemos fazer é sobre métricas mensais por cliente,

```
1 CREATE VIEW customer_monthly AS
2 SELECT
3     customer_id,
4     DATE_TRUNC('month', order_date) AS month,
5     COUNT(*) AS orders_count,
6     SUM(total) AS total_spent,
7     AVG(total) AS avg_order_value
8 FROM orders
9 GROUP BY customer_id, DATE_TRUNC('month', order_date);
```



Transform

Uma outra view que podemos fazer é sobre métricas mensais por cliente,

```
1 CREATE VIEW customer_monthly AS
2 SELECT
3     customer_id,
4     DATE_TRUNC('month', order_date) AS month,
5     COUNT(*) AS orders_count,
6     SUM(total) AS total_spent,
7     AVG(total) AS avg_order_value
8 FROM orders
9 GROUP BY customer_id, DATE_TRUNC('month', order_date)
```

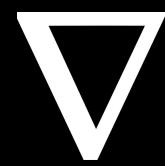
	customer_id	month	orders_count	total_spent	avg_order_value
1	30	2025-10-01 00:00:00	2	3114.57	1557.28500000000000000000
2	19	2025-09-01 00:00:00	2	3042.09	1521.04500000000000000000
3	11	2025-10-01 00:00:00	1	1346.90	1346.90000000000000000000
4	7	2025-10-01 00:00:00	2	1811.05	905.52500000000000000000
5	2	2025-09-01 00:00:00	3	1536.05	512.016666666666666667
6	29	2025-10-01 00:00:00	2	2057.98	1028.99000000000000000000
7	29	2025-09-01 00:00:00	3	3204.34	1068.1133333333333333
8	27	2025-10-01 00:00:00	3	2857.38	952.46000000000000000000
9	21	2025-10-01 00:00:00	3	2633.92	877.9733333333333333
10	24	2025-09-01 00:00:00	1	1192.78	1192.78000000000000000000
11	1	2025-09-01 00:00:00	5	4619.79	923.95800000000000000000
12	19	2025-08-01 00:00:00	2	2998.39	1499.19500000000000000000
13	5	2025-08-01 00:00:00	4	2699.03	674.75750000000000000000



Load

Agora nossos dados estão prontos para o consumo.

Por exemplo, poderíamos usar bibliotecas de python para análise de dados, plotagem, etc.



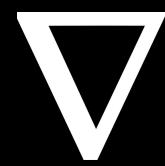
Load

Um exemplo de carregamento do banco de dados em python:

```
# Notebook Python: ler a materialized view
import pandas as pd
import sqlalchemy

engine = sqlalchemy.create_engine("postgresql://usuario:senha@localhost:5432/mydb")

# Ler a MV
df = pd.read_sql("SELECT * FROM customer_summary_mv", engine)
df.head()
```



Load

Podemos fazer, por exemplo, uma plotagem simples:

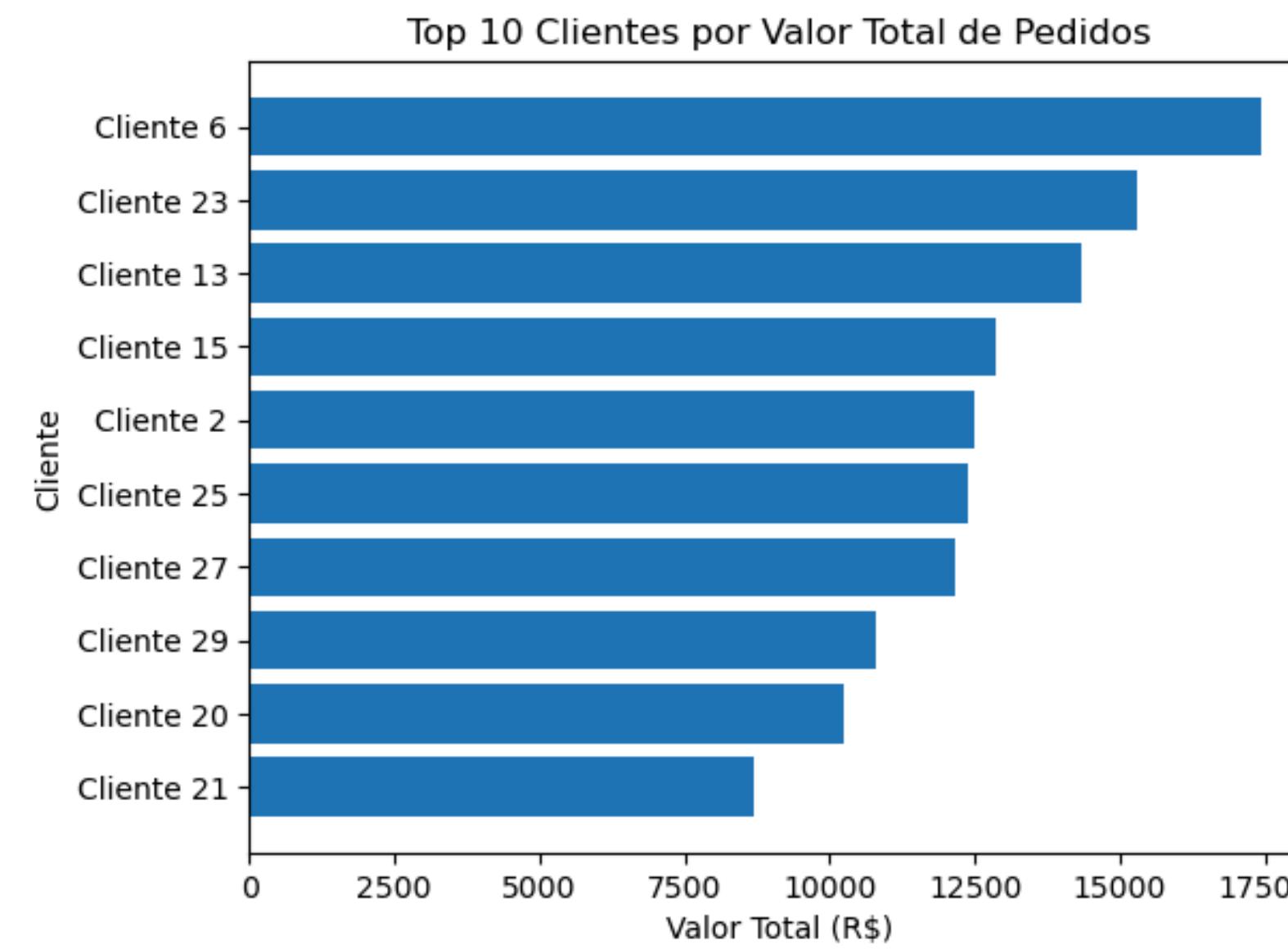
```
import matplotlib.pyplot as plt

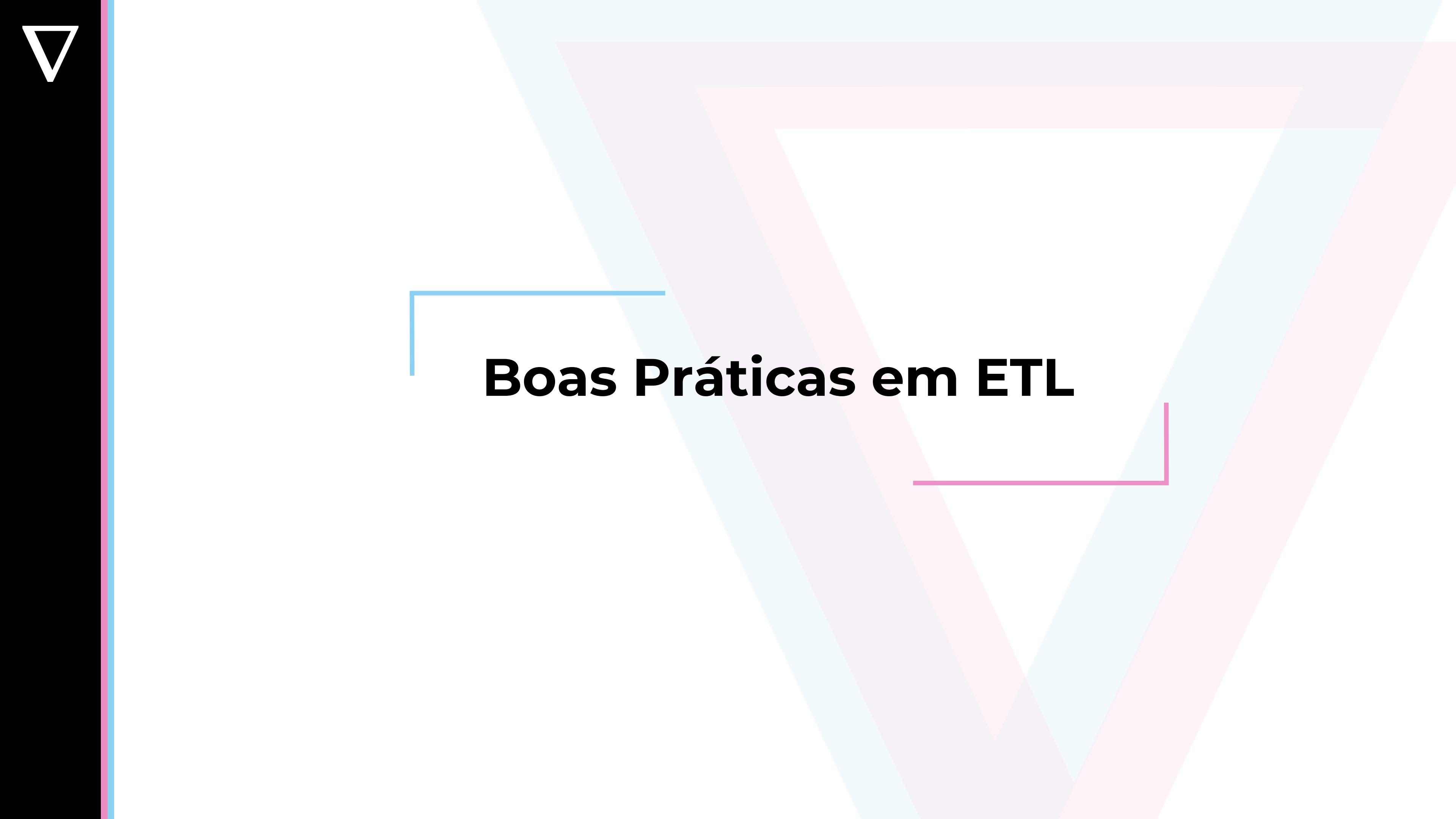
top_customers = df.sort_values("valor_total", ascending=False).head(10)

plt.barh(top_customers["name"], top_customers["valor_total"])
plt.xlabel("Valor Total (R$)")
plt.ylabel("Cliente")
plt.title("Top 10 Clientes por Valor Total de Pedidos")
plt.gca().invert_yaxis()
plt.show()
```

Load

Podemos fazer, por exemplo, uma plotagem simples:





Boas Práticas em ETL



Boas práticas

- **Auditoria:** mantenha colunas de `created_at` e `updated_at` nas tabelas e logs de execução do ETL
- **Idempotência:** scripts que possam rodar várias vezes sem duplicar dados.
- **Backfill:** estratégia para importar dados históricos.
- **Atualizações incrementais:** evite recriar toda a base; atualize apenas registros novos/alterados.
- **Documentação:** comente queries complexas e mantenha um changelog do pipeline.



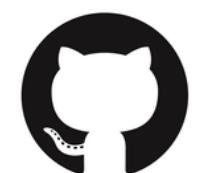
data@icmc.usp.br



[@data.icmc](https://www.instagram.com/@data.icmc)



[/c/DataICMC](https://www.youtube.com/c/DataICMC)



[/icmc-data](https://github.com/icmc-data)



data.icmc.usp.br

|| obrigado!