

# AULA 8 – ANALYZE

PROFA. DRA. LEILA BERGAMASCO

CC6240 – Tópicos Avançados em Banco de Dados

# AGENDA

- Analyze

# RECAPITULANDO

- Otimização de Consulta
  - O usuário não escreve a consulta pensando em eficiência
  - Cabe ao SGBD elaborar diferentes estratégias para recuperar de forma eficiente os dados.
    - Aulas passadas vimos como partes específicas das consultas influenciam o resultado
      - Consultas utilizando atributos que sejam ou não chaves primárias
      - Consultas utilizando atributos que sejam ou não índices
      - Consultas utilizando =, < , >
      - Consultas que utilizam ORDER BY
      - Consultas que utilizam JOIN
        - Merge, Hash, Nested (Aninhado)
      - Como o arranjo das consultas influenciam o resultado final
    - Na aula de hoje: como aplicar isso em uma base de dados?

## CLÁUSULA EXPLAIN

- É comum escrevermos uma consulta no banco de dados e ela demorar mais do que o esperado.
  - O que é demorado?
    - + de 3 minutos desconfiem.
- Como identificar os pontos de gargalos?
  - EXPLAIN
    - Oracle, Redshift, MySQL, MongoDB

# EXPLAIN

## ■ Sintaxe

```
EXPLAIN [ ( option [, ...] ) ] statement  
EXPLAIN [ ANALYZE ] [ VERBOSE ] statement
```

where *option* can be one of:

```
ANALYZE [ boolean ]  
VERBOSE [ boolean ]  
COSTS [ boolean ]  
BUFFERS [ boolean ]  
TIMING [ boolean ]  
FORMAT { TEXT | XML | JSON | YAML }
```

# EXPLAIN

- O explain gera o nosso PEC → *query plan*
- Com todas as estatísticas e tempos de execução

```
EXPLAIN SELECT * FROM clientes WHERE cpf = '222.222.222-22';
```

## QUERY PLAN

```
-----  
Seq Scan on clientes (cost=0.00..208.23 rows=1 width=54)  
  Filter: ((cpf)::text = ' 222.222.222-22'::text)
```

Estratégia utilizada  
(I)

Condições

Tab

Campos

Custo

Tipo do campo

## EXPLAIN - PARÂMETROS

- COSTS
- FORMAT
- VERBOSE
- SUMMARY
- ANALYZE
  - TIMING
  - BUFFERS
  - SETTINGS
- WALL

# EXPLAIN - COSTS

- Vem habilitado por default
  - Para desabilitar

```
explain SELECT *  
  FROM public.cliente  
  ORDER BY numero_cliente
```

QUERY PLAN
text

Sort (cost=45.59..47.17 rows=630 width=100)
---

Sort Key: numero_cliente
--------------------------

-> Seq Scan on cliente (cost=0.00..16.30 rows=630 width=100)
--

```
explain (costs off)  
  SELECT *  
  FROM public.cliente  
  ORDER BY numero_cliente
```

QUERY PLAN
text

Sort
------

Sort Key: numero_cliente
--------------------------

-> Seq Scan on cliente
------------------------



# EXPLAIN

## ■ Custo

- Lembrem-se: é uma estimativa!
- Valores arbitrários -> não são “tempo”
- Formato: xx .. yy
  - xx: unidade de inicio
  - yy: unidade de fim

Dessas quatro  
operações qual  
demorou mais?

```
Nested Loop (cost=4.65..118.62 rows=10 width=488)
-> Bitmap Heap Scan on tenk1 t1 (cost=4.36..39.47 rows=10 width=244)
    Recheck Cond: (unique1 < 10)
    -> Bitmap Index Scan on tenk1_unique1 (cost=0.00..4.36 rows=10 width=0)
        Index Cond: (unique1 < 10)
-> Index Scan using tenk2_unique2 on tenk2 t2 (cost=0.29..7.91 rows=1 width=244)
    Index Cond: (unique2 = t1.unique2)
```

## EXPLAIN - FORMAT

- Diferentes formatos de saída
  - YAML, JSON, XML, TXT: default

```
explain SELECT *  
  FROM public.cliente  
  ORDER BY numero_cliente
```

QUERY PLAN  
text

Sort (cost=45.59..47.17 rows=630 width=100)

Sort Key: numero\_cliente

-> Seq Scan on cliente (cost=0.00..16.30 rows=630 width=100)

```
explain (format JSON) SELECT *  
  FROM public.cliente  
  ORDER BY numero_cliente
```

```
[  
  {  
    "Plan": {  
      "Node Type": "Sort",  
      "Parallel Aware": false,  
      "Startup Cost": 45.59,  
      "Total Cost": 47.17,  
      "Plan Rows": 630,  
      "Plan Width": 100,  
      "Sort Key": [  
        "numero_cliente"  
      ],  
      "Plans": [  
        {  
          "Node Type": "Seq Scan",  
          "Parent Relationship": "Outer",  
          "Parallel Aware": false,  
          "Relation Name": "cliente",  
          "Alias": "cliente",  
          "Startup Cost": 0,  
          "Total Cost": 16.3,  
          "Plan Rows": 630,  
          "Plan Width": 100  
        }  
      ]  
    }  
  }  
]
```

# EXPLAIN - FORMAT

- Diferentes formatos de saída
  - YAML, JSON, XML, TXT: default

```
explain SELECT *  
FROM public.cliente  
ORDER BY numero_cliente
```

QUERY PLAN  
text

Sort (cost=45.59..47.17 rows=630 width=100)

Sort Key: numero\_cliente

-> Seq Scan on cliente (cost=0.00..16.30 rows=630 width=100)

```
explain (format JSON) SELECT *  
FROM public.cliente  
ORDER BY numero_cliente
```

```
[  
  {  
    "Plan": {  
      "Node Type": "Sort",  
      "Parallel Aware": false,  
      "Startup Cost": 45.59,  
      "Total Cost": 47.17,  
      "Plan Rows": 630,  
      "Plan Width": 100,  
      "Sort Key": [  
        "numero_cliente"  
      ],  
      "Plans": [  
        {  
          "Node Type": "Seq Scan",  
          "Parent Relationship": "Outer",  
          "Parallel Aware": false,  
          "Relation Name": "cliente",  
          "Alias": "cliente",  
          "Startup Cost": 0,  
          "Total Cost": 16.3,  
          "Plan Rows": 630,  
          "Plan Width": 100  
        }  
      ]  
    }  
  }  
]
```

## EXPLAIN - VERBOSE

- Vem desabilitado por default
- Informações adicionais sobre triggers, schemas nome de campos
- Útil para consultas mais complexas que envolvem muitos campos

```
explain SELECT *  
  FROM public.cliente  
  ORDER BY numero_cliente
```

QUERY PLAN
text
Sort (cost=45.59..47.17 rows=630 width=100)
Sort Key: numero_cliente
-> Seq Scan on cliente (cost=0.00..16.30 rows=630 width=100)

```
explain (verbose) SELECT *  
  FROM public.cliente  
  ORDER BY numero_cliente
```

	QUERY PLAN	
	text	🔒
1	Sort (cost=45.59..47.17 rows=630 width=100)	
2	Output: cidade, nome, numero_cliente, agencia	
3	Sort Key: cliente.numero_cliente	
4	-> Seq Scan on public.cliente (cost=0.00..16.30 rows=630 width=1...	
5	Output: cidade, nome, numero_cliente, agencia	

## EXPLAIN - SUMMARY

- Mostra o tempo planejado/estimado para a execução da consulta

```
explain SELECT *  
  FROM public.cliente  
  ORDER BY numero_cliente
```

QUERY PLAN  
text

Sort (cost=45.59..47.17 rows=630 width=100)

Sort Key: numero\_cliente

-> Seq Scan on cliente (cost=0.00..16.30 rows=630 width=100)

```
explain (summary)  
  SELECT *  
  FROM public.cliente  
  ORDER BY numero_cliente
```

QUERY PLAN  
text

1 Sort (cost=45.59..47.17 rows=630 width=100)

2 Sort Key: numero\_cliente

3 -> Seq Scan on cliente (cost=0.00..16.30 rows=630 width=100)

4 Planning Time: 0.045 ms

# COISAS LEGAIS NO POSTGRES

- PgAdmin
  - Ver versão
  - Ver work\_mem
  - Mostrar exemplos de explain e analyze
  - Mostrar format
  - Mostrar verbose
  - Mostrar summary

# EXPLAIN ANALYZE

- Apresenta a estimativa, executa a query e mostra resultado final em milisegundos
  - Mais acurado que apenas o EXPLAIN → apenas estimativa
  - **CUIDADO COM INSERT E DELETE**

```
explain SELECT *  
  FROM public.cliente  
 ORDER BY numero_cliente
```

QUERY PLAN
text
Sort (cost=45.59..47.17 rows=630 width=100)
Sort Key: numero_cliente
-> Seq Scan on cliente (cost=0.00..16.30 rows=630 width=100)

```
explain (analyze)  
  SELECT *  
  FROM public.cliente  
 ORDER BY numero_cliente
```

## QUERY PLAN

text
Sort (cost=45.59..47.17 rows=630 width=100) (actual time=0.020..0.021 rows=3 loops=1)
Sort Key: numero_cliente
Sort Method: quicksort Memory: 25kB
-> Seq Scan on cliente (cost=0.00..16.30 rows=630 width=100) (actual time=0.009..0.011 rows=3 loops=1)
Planning Time: 0.087 ms
Execution Time: 0.044 ms

# EXPLAIN ANALYZE

## ■ Para INSERT/DELETE/UPDATE

**BEGIN;**

```
EXPLAIN ANALYZE UPDATE tenk1 SET hundred = hundred + 1 WHERE unique1 < 100;
```

### QUERY PLAN

```
-----
Update on tenk1  (cost=5.07..229.46 rows=101 width=250) (actual time=14.628..14.628 rows=0 loops=1)
->  Bitmap Heap Scan on tenk1  (cost=5.07..229.46 rows=101 width=250) (actual time=0.101..0.439 rows=100 loops=1)
      Recheck Cond: (unique1 < 100)
      ->  Bitmap Index Scan on tenk1_unique1  (cost=0.00..5.04 rows=101 width=0) (actual time=0.043..0.043 rows=100 loops=1)
            Index Cond: (unique1 < 100)
Planning time: 0.079 ms
Execution time: 14.727 ms
```

**ROLLBACK;**



# EXPLAIN ANALYZE

- `actual_time`: tempo em ms para encontrar o primeiro registro

```
explain SELECT *  
  FROM public.cliente  
  ORDER BY numero_cliente
```

```
explain (analyze)  
  SELECT *  
  FROM public.cliente  
  ORDER BY numero_cliente
```

## QUERY PLAN

text

Sort (cost=45.59..47.17 rows=630 width=100)

Sort Key: numero\_cliente

-> Seq Scan on cliente (cost=0.00..16.30 rows=630 width=100)

## QUERY PLAN

text

Sort (cost=45.59..47.17 rows=630 width=100) (actual time=0.020..0.021 rows=3 loops=1)

Sort Key: numero\_cliente

Sort Method: quicksort Memory: 25kB

-> Seq Scan on cliente (cost=0.00..16.30 rows=630 width=100) (actual time=0.009..0.011 rows=3 loops=1)

Planning Time: 0.087 ms

Execution Time: 0.044 ms

# EXPLAIN ANALYZE TIMING

- Vem habilitado por default
- Se desabilitar, retira o `actual_time` da análise

```
explain SELECT *  
  FROM public.cliente  
  ORDER BY numero_cliente
```

## QUERY PLAN

text

Sort (cost=45.59..47.17 rows=630 width=100)

Sort Key: numero\_cliente

-> Seq Scan on cliente (cost=0.00..16.30 rows=630 width=100)

```
explain (analyze, timing off)  
  SELECT *  
  FROM public.cliente  
  ORDER BY numero_cliente
```

## QUERY PLAN

text

Sort (cost=45.59..47.17 rows=630 width=100) (actual rows=3 loops=1)

Sort Key: numero\_cliente

Sort Method: quicksort Memory: 25kB

-> Seq Scan on cliente (cost=0.00..16.30 rows=630 width=100) (actual rows=3 loops=1)

Planning Time: 0.086 ms

Execution Time: 0.037 ms

# EXPLAIN - BUFFERS

- Só pode ser utilizado em conjunto com ANALYZE
- Mostra a informação de “*shared read*”
  - Número de blocos lidos em cache para determinada tabela

```
explain SELECT *
  FROM public.cliente
 ORDER BY numero_cliente
```

QUERY PLAN  
text

Sort (cost=45.59..47.17 rows=630 width=100)

Sort Key: numero\_cliente

-> Seq Scan on cliente (cost=0.00..16.30 rows=630 width=100)

```
explain (ANALYZE, BUFFERS )
  SELECT *
  FROM public.cliente
 ORDER BY numero_cliente
```

QUERY PLAN		text	
1	Sort	(cost=45.59..47.17 rows=630 width=100) (actual time=0.943..0.945 rows=3 loops=1)	
2	Sort Key:	numero_cliente	
3	Sort Method:	quicksort Memory: 25kB	
4	Buffers:	shared read=1	
5	-> Seq Scan on cliente	(cost=0.00..16.30 rows=630 width=100) (actual time=0.930..0.932 rows=3 loops=1)	
6	Buffers:	shared read=1	
7	Planning Time:	0.084 ms	
8	Execution Time:	0.966 ms	

# EXPLAIN - SETTINGS

- A partir da versão Postgres v12(2019)
  - Para ver o valor atual: `show version`
- Só pode ser utilizado em conjunto com ANALYZE
- Mostra a informação de quanta memória interna pode ser consumida (work\_mem)
  - Para ver o valor atual: `show work_mem`

Query

-----

```
explain (analyze, settings) select * from t order by c;
```

Query Plan

-----

```
Sort (actual rows=9664 loops=1)
  Sort Key: c
  Sort Method: quicksort  Memory: 1140kB
  -> Seq Scan on t (actual rows=9664 loops=1)
Settings: work_mem = '16MB'
```

## EXPLAIN - WAL

- A partir da versão Postgres v13(2020)

- Para ver o valor atual: `show version`

- Write-Ahead Logging (WAL)

- implica que qualquer alteração feita no banco de dados deve primeiro ser anexada ao arquivo de log, em seguida, o arquivo de registro deve ser liberado/inserido no disco.
- Qual é a consequência disso?
  - Garantir que quando houver uma falha no SO ou no PostgreSQL ou no hardware, o banco de dados possa ser recuperado

Query

-----

```
explain (analyze, wal, costs off, timing off, summary off)
delete from t where id = 1;
```

Query Plan

-----

```
Delete on t (actual rows=0 loops=1)
  WAL: records=1 bytes=54
  -> Seq Scan on test (actual rows=1 loops=1)
        Filter: (id = 1)
```

# COISAS LEGAIS NO POSTGRES

- PgAdmin
  - Mostrar analyze completo
  - Mostrar analyze com timing, costs off
  - Mostrar buffer

# EXPLAIN – OPERAÇÕES DE SCAN

- Seq Scan: leitura sequencial
  - Bom para tabelas pequenas
- Index Scan: leitura baseada em índices ou primary key
  - Bom quando você tem interesse em uma pequena porção dos dados
- Index Only Scan: leitura baseada **SOMENTE** em índices ou no **SELECT** os atributos também pertencem ao índice.

```
postgres=# explain analyze select count(*) from categories;
```

QUERY PLAN

```
-----
Aggregate  (cost=12.53..12.54 rows=1 width=0) (actual time=0.046..0.046 rows=1 loops=1)
->  Index Only Scan using categories_pkey on categories  (cost=0.00..12.49 rows=16 width=0) (actual time=0.018..0.038 rows=16 loops=1)
      Heap Fetches: 16
Total runtime: 0.108 ms
(4 rows)
```

Existem várias outras  
operações de SCAN

QUERY PLAN

text

Sort (cost=45.59..47.17 rows=630 width=100)

Sort Key: numero\_cliente

-> Seq Scan on cliente (cost=0.00..16.30 rows=630 width=100)

text

Index Scan using cliente\_pkey on cliente (cost=0.15..8.17 rows=1 width=100) (actual time=0.008..0.008 rows=0 loops=1)

Index Cond: (numero\_cliente = 1)

Buffers: shared hit=1

Planning Time: 0.065 ms

Execution Time: 0.019 ms

# EXPLAIN – OPERAÇÕES DE JOIN

- Nested Join: faz um-contra-todos
- Hash Join: utiliza uma tabela hash para fazer o join
  - Bom quando você faz o join por um atributo PK-FK, e a tabela hash criada CAIBA NA MEMÓRIA.
- Merge Join: aplicado quando ambas as tabelas envolvidas no join são grandes.  
Porque??

```
explain (analyze, buffers, timing)
  SELECT nome
    FROM public.cliente, emprestimo
 WHERE emprestimo.nome_agencia > cliente.agencia
```

QUERY PLAN	
text	
Nested Loop (cost=0.00..11379.88 rows=252000 width=32) (actual time=0.021..0.027 rows=2 loops=1)	
Join Filter: ((emprestimo.nome_agencia)::text > (cliente.agencia)::text)	
Rows Removed by Join Filter: 7	
Buffers: shared hit=2	
-> Seq Scan on emprestimo (cost=0.00..22.00 rows=1200 width=32) (actual time=0.006..0.007 rows=3 loops=1)	
Buffers: shared hit=1	
-> Materialize (cost=0.00..19.45 rows=630 width=64) (actual time=0.002..0.003 rows=3 loops=3)	
Buffers: shared hit=1	
-> Seq Scan on cliente (cost=0.00..16.30 rows=630 width=64) (actual time=0.005..0.006 rows=3 loops=1)	
Buffers: shared hit=1	
Planning Time: 0.080 ms	
Execution Time: 0.049 ms	



# COISAS LEGAIS NO POSTGRES

- PgAdmin
  - Mostrar join com  $> e =$
  - Mostrar Where com  $> e =$
  - Mostrar order by
- Mostrar gráficos explain e analyze

## BIBLIOGRAFIA

- ABRAHAM SILBERSCHATZ, HENRY F. KORTH, S. SUDARSHAN. Sistema de Banco de Dados. 6. Campus. 0. ISBN 9788535245356.
- ELMASRI, RAMEZ, SHAMKANT B. NAVATHE. Sistemas de banco de dados. Vol. 6. São Paulo: Pearson Addison Wesley, 2011.
- DATE, CHRISTHOPER J. Introdução a Sistemas de Bancos de Dados, 5ª. Edição. Campus, Rio de Janeiro (2004).

OBRIGADO E ATÉ A PRÓXIMA AULA!