Boas práticas e desempenho no PostgreSQL

Francisco Summa Netto
DBA PostgreSQL

CELEPAR
Tecnologia da Informação e Comunicação do Paraná

PgDay Curitiba na Celepar
Março de 2016

PostgreSQL

Agenda

- **Boas práticas**
  - Padrões
  - Char
  - Tipo
  - Quantidade de registros
  - Autocomplete
  - Quantidade de colunas
  - Chave primária
  - Joins
- **Desempenho**
  - Índices
  - Explain
  - Explain analyze
  - Like
  - Busca textual
  - Índice Parcial
  - Clusterização
  - Particionamento de tabelas
  - Replicação
- **pgRelatórios**
- **pgBadger**

Boas Práticas
Nomenclatura

- **Seguir padrões**
  - Facilita e muito para quando alguém esta vendo os objetos no banco, identificar o que é este objeto
    - Tabela → tb_
    - Índice → idx_
    - View → vw_
    - Função → fn_
    - Trigger → tg_
    - Sequence → _seq
    - Tablespace → tbs_

Boas Práticas

Tipos de dados: CHAR

- **VARCHAR != CHAR**
  - varchar lê até encontrar o null
  - char busca o tamanho fixo
  - colunas até 10, 12 caracteres é recomendado o char
  - tipos definidos é melhor usar o char
    - ex: uf, cpf, sexo(enum)

Boas Práticas

Tipos de dados: campo sexo

- **Campo sexo em um servidor corporativo**

	sexo1	sexo2	sexo3	sexo4	sexo5	sexo6	sexo7	sexo8
Tipo	varchar(1)	varchar(10)	varchar(4)	varchar	char	integer	smallint	text
Dados	M	Masculino	Masc	Masculino	M	1	1	Masculino
Bytes	2	10	5	10 - N	2	4	2	10 - N

- **CREATE TYPE sexo AS ENUM ('Feminino', 'Masculino', 'Não definido');**

Boas Práticas
Quantidade de registros

- **Utilize filtros: WHERE**

- **LIMIT x OFFSET y**
  - Limit → Quantidade de registros
  - Offset → Começo dos registros
    - LIMIT 10 → 1 ao 10
    - LIMIT 10 OFFSET 10 → 11 ao 20
    - LIMIT 20 OFFSET 70 → 71 ao 90
  - Muito utilizado para **paginação**

Boas Práticas
Autocomplete

- **Utilizado no HTML para apresentar sugestões de preenchimento em um campo**
- **NÃO utilizar ORDER BY na consulta**
- **Se necessário ordenar os dados faça:**
  - via aplicação (melhor)
  - criar índice ordenado
- **Utilizar LIMIT**

Boas Práticas
Quantidade de colunas

- **SELECT * FROM tabela**
  - Mas na aplicação apresenta apenas o nome
  - Ele trará todas as colunas (de todas as tabelas envolvidas)

- **SELECT c1, c2, ..., c233 FROM tabela**
  - Serão utilizadas as 233 colunas??

Boas Práticas
Chave primária

- **Chave primária numérica**
  - Otimiza a junção entre tabelas
  - Comparação entre números é muito mais rápido que comparar texto

- **Utilize chave natural**
  - Informações únicas são excelentes para ser a chave da tabela. Ex: CPF, Matricula, ISSN, ...

Boas Práticas
Chave primária

- **Chave primária pode ser criada com SERIAL**
  - Cria automaticamente uma sequence a coluna e define como valor default o próximo valor da sequence
    - smallserial → smallint → 2 bytes → 32.767
    - serial → integer → 4 bytes → 2.147.483.647
    - bigserial → bigint → 8 bytes → 9.223.372.036.854.775.807

```
CREATE TABLE teste (id serial PRIMARY KEY, nome varchar(60));

 Column |       Type          |                    Modifiers                    |  Storage  | Stats target | Description
--------+---------------------+-------------------------------------------------+-----------+--------------+-------------
 id     | integer             | not null default nextval('teste_id_seq'::regclass) | plain     |              |
 nome   | character varying(60) |                                                 | extended  |              |
Indexes:
    "teste_pkey" PRIMARY KEY, btree (id)
```

Boas Práticas

Junção com outras tabelas

- **Deixar explícito a condição de junção nos joins e não colocá-las no where**

```
select t1.col1, t2.col2
from tabela_1 t1
inner join tabela_2 t2 ON t1.coluna_id = t2.id
...
```

- **Aconselhável utilizar no máximo 8 tabelas, ou seja, 7 joins.**

  O tempo de escolha do plano de execução aumenta exponencialmente após a oitava tabela.

O banco esta lento
O que eu faço?

- **O banco esta lento**
  O que eu faço?
  - Pesquisar onde estão os gargalos
  - Verificar as consultas demoradas e sua utilização dos índices
  - Utilização do LIKE
  - Índice parcial
  - Índice clusterizado
  - Particionamento de tabela
  - Hot-Standby

Desempenho
Utilização de índices

- **Prós:**
  - São bons para consultas
  - Fundamentais para JOINS
  - Podem ser criados com condições ou funções
- **Contras:**
  - Ruins para UPDATE, INSERT e DELETE
  - Gasta espaço
- **Pode retornar os dados diretamente do índice**
- **Índice composto**
- Criar utilizando o **CONCURRENTLY**, não bloqueia a tabela.

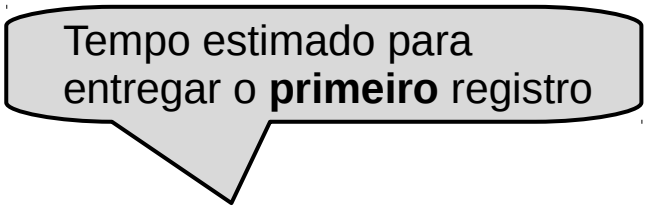
Desempenho
Utilização de índices

- **Verificar o plano de execução de uma consulta com o EXPLAIN**
  - Se fizer a utilização do índice esta legal
    - Index Scan, Index Only Scan, Index Only Scan Backward, Bitmap index scan, Bitmap heap scan, ...
  - Se fizer **Seq Scan** ele varre toda a tabela = demorado
    - Para tabelas 'pequenas' sempre irá fazer Seq Scan
- **EXPLAIN ANALYZE**
  - CUIDADO com DELETE, UPDATE e INSERT, pois irá executar a consulta

Desempenho
Explain

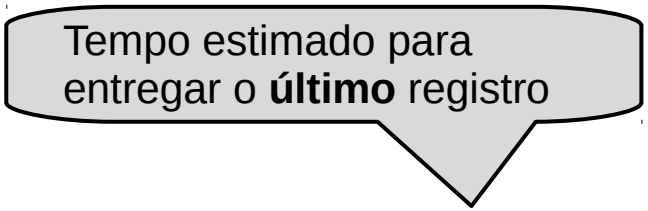

```
                                                                QUERY PLAN
Unique  (cost=1131.50..1131.55 rows=1 width=144)
  ->  Sort  (cost=1131.50..1131.51 rows=1 width=144)
        Sort Key: a.descricao, a.cod_almox, a.cod_user_manutencao, a.cod_user_responsavel,
a.cod_orgao, a.logradouro, a.complemento, a.bairro, a.uf, a.cep, a.fone, a.fax, a.ts_criacao
        ->  Nested Loop  (cost=12.95..1131.49 rows=1 width=144)
              ->  Nested Loop  (cost=12.95..95.36 rows=3 width=152)
                    ->  Hash Semi Join  (cost=12.95..70.11 rows=37 width=148)
                          Hash Cond: (a.cod_almox = ao.cod_almox)
                          ->  Seq Scan on tb_almox a  (cost=0.00..55.44 rows=482 width=144)
                                Filter: (cod_status = 17510)
                          ->  Hash  (cost=12.45..12.45 rows=40 width=4)
                                ->  Seq Scan on tb_almox_orgao ao  (cost=0.00..12.45 rows=40 width=4)
                                      Filter: (cod_orgao = 59)
                    ->  Index Only Scan using tb_user_almox_pk on tb_user_almox ua (cost=0.00..0.67
rows=1 width=4)
                          Index Cond: ((cod_almox = a.cod_almox) AND (cod_user = 11532))
              ->  Index Only Scan using tb_fechamento_mensal_entrada_index47600 on
tb_fechamento_mensal_entrada f  (cost=0.00..345.37 rows=1 width=4)
                    Index Cond: (cod_almox = a.cod_almox)
                    Filter: ((date_part('month', (mes_ano_ref)) = 1) AND (date_part('year', (mes_ano_ref)) =
2016))
```

Desempenho
Explain



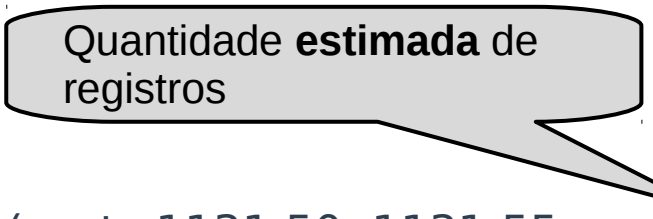

```
                                                QUERY PLAN
Unique  (cost=1131.50..1131.55 rows=1 width=144)
  ->  Sort  (cost=1131.50..1131.51 rows=1 width=144)
        Sort Key: a.*
        ->  Nested Loop  (cost=12.95..1131.49 rows=1 width=144)
              ->  Nested Loop  (cost=12.95..95.36 rows=3 width=152)
                    ->  Hash Semi Join  (cost=12.95..70.11 rows=37 width=148)
                          Hash Cond: (a.cod_almox = ao.cod_almox)
                          ->  Seq Scan on tb_almox a  (cost=0.00..55.44 rows=482 width=144)
                                Filter: (cod_status = 17510)
                          ->  Hash  (cost=12.45..12.45 rows=40 width=4)
                                ->  Seq Scan on tb_almox_orgao ao (cost=0.00..12.45 rows=40 width=4)
                                      Filter: (cod_orgao = 59)
                    ->  Index Only Scan using tb_user_almox_pk on tb_user_almox ua
(cost=0.00..0.67 rows=1 width=4)
...
```

Desempenho
Explain



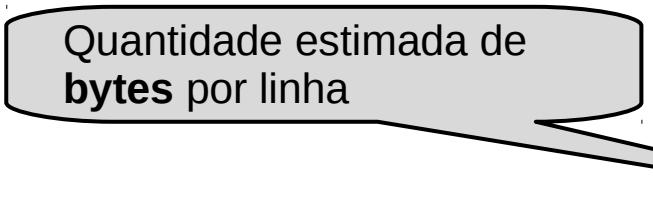

```
                                QUERY PLAN
Unique  (cost=1131.50..1131.55 rows=1 width=144)
  ->  Sort  (cost=1131.50..1131.51 rows=1 width=144)
        Sort Key: a.*
        ->  Nested Loop  (cost=12.95..1131.49 rows=1 width=144)
              ->  Nested Loop  (cost=12.95..95.36 rows=3 width=152)
                    ->  Hash Semi Join  (cost=12.95..70.11 rows=37 width=148)
                          Hash Cond: (a.cod_almox = ao.cod_almox)
                          ->  Seq Scan on tb_almox a  (cost=0.00..55.44 rows=482 width=144)
                                Filter: (cod_status = 17510)
                          ->  Hash  (cost=12.45..12.45 rows=40 width=4)
                                ->  Seq Scan on tb_almox_orgao ao (cost=0.00..12.45 rows=40 width=4)
                                      Filter: (cod_orgao = 59)
                    ->  Index Only Scan using tb_user_almox_pk on tb_user_almox ua
(cost=0.00..0.67 rows=1 width=4)
...
```

Desempenho
Explain



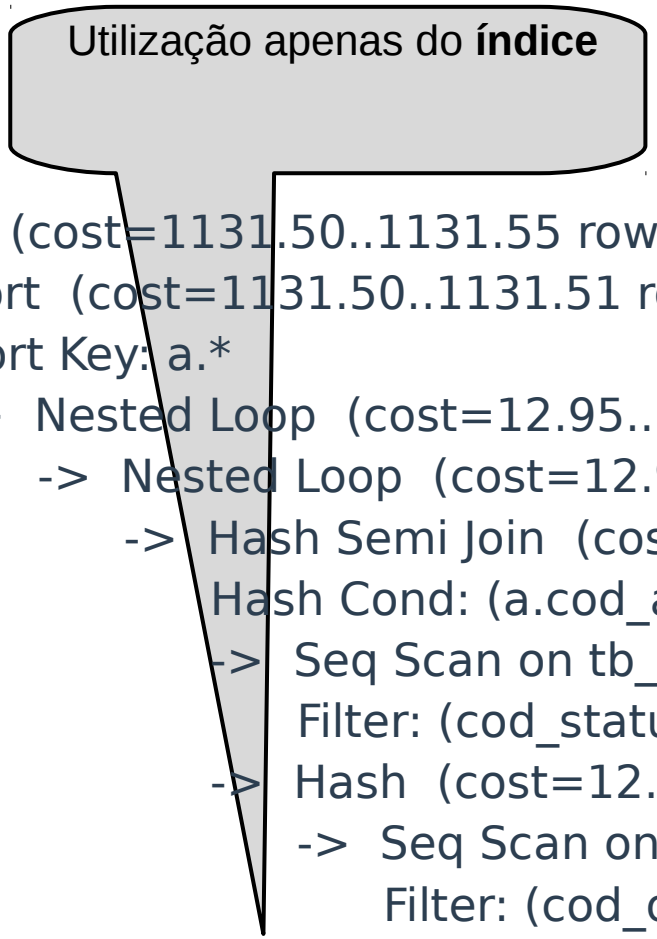

```
                                QUERY PLAN
Unique  (cost=1131.50..1131.55 rows=1 width=144)
  ->  Sort  (cost=1131.50..1131.51 rows=1 width=144)
        Sort Key: a.*
        ->  Nested Loop  (cost=12.95..1131.49 rows=1 width=144)
              ->  Nested Loop  (cost=12.95..95.36 rows=3 width=152)
                    ->  Hash Semi Join  (cost=12.95..70.11 rows=37 width=148)
                          Hash Cond: (a.cod_almox = ao.cod_almox)
                          ->  Seq Scan on tb_almox a  (cost=0.00..55.44 rows=482 width=144)
                                Filter: (cod_status = 17510)
                          ->  Hash  (cost=12.45..12.45 rows=40 width=4)
                                ->  Seq Scan on tb_almox_orgao ao (cost=0.00..12.45 rows=40 width=4)
                                      Filter: (cod_orgao = 59)
                    ->  Index Only Scan using tb_user_almox_pk on tb_user_almox ua
(cost=0.00..0.67 rows=1 width=4)
...
```

Desempenho
Explain




```
                                QUERY PLAN
Unique  (cost=1131.50..1131.55 rows=1 width=144)
  ->  Sort  (cost=1131.50..1131.51 rows=1 width=144)
        Sort Key: a.*
        ->  Nested Loop  (cost=12.95..1131.49 rows=1 width=144)
              ->  Nested Loop  (cost=12.95..95.36 rows=3 width=152)
                    ->  Hash Semi Join  (cost=12.95..70.11 rows=37 width=148)
                          Hash Cond: (a.cod_almox = ao.cod_almox)
                          ->  Seq Scan on tb_almox a  (cost=0.00..55.44 rows=482 width=144)
                                Filter: (cod_status = 17510)
                          ->  Hash  (cost=12.45..12.45 rows=40 width=4)
                                ->  Seq Scan on tb_almox_orgao ao (cost=0.00..12.45 rows=40 width=4)
                                      Filter: (cod_orgao = 59)
                    ->  Index Only Scan using tb_user_almox_pk on tb_user_almox ua
(cost=0.00..0.67 rows=1 width=4)
...
```

Desempenho
Explain




```
                                                QUERY PLAN
Unique  (cost=1131.50..1131.55 rows=1 width=144)
  ->  Sort  (cost=1131.50..1131.51 rows=1 width=144)
        Sort Key: a.*
        ->  Nested Loop  (cost=12.95..1131.49 rows=1 width=144)
              ->  Nested Loop  (cost=12.95..95.36 rows=3 width=152)
                    ->  Hash Semi Join  (cost=12.95..70.11 rows=37 width=148)
                          Hash Cond: (a.cod_almox = ao.cod_almox)
                          ->  Seq Scan on tb_almox a  (cost=0.00..55.44 rows=482 width=144)
                                Filter: (cod_status = 17510)
                          ->  Hash  (cost=12.45..12.45 rows=40 width=4)
                                ->  Seq Scan on tb_almox_orgao ao (cost=0.00..12.45 rows=40 width=4)
                                      Filter: (cod_orgao = 59)
                    ->  Index Only Scan using tb_user_almox_pk on tb_user_almox ua
(cost=0.00..0.67 rows=1 width=4)
...
```

Desempenho
Explain

Seq Scan: Irá varrer a tabela inteira.
Falta índice ?

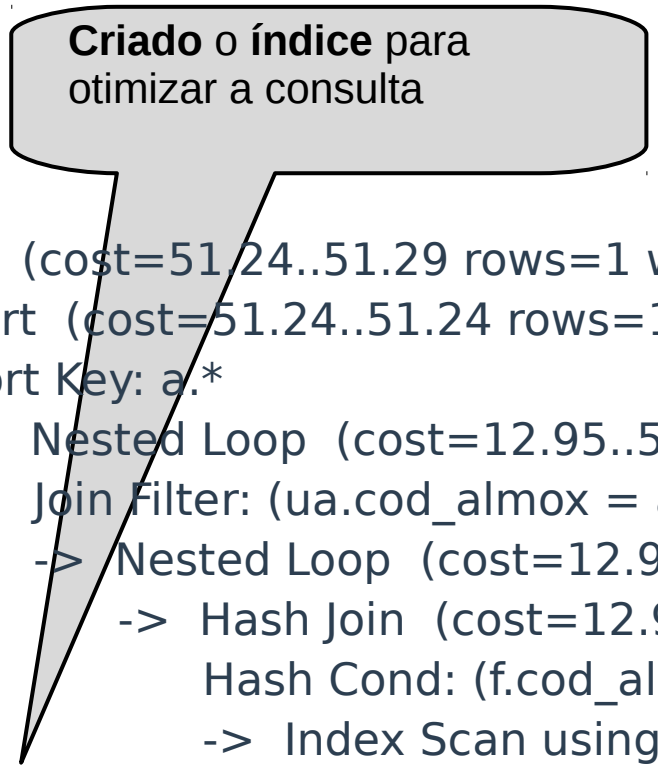
```
                                                QUERY PLAN
Unique  (cost=1131.50..1131.55 rows=1 width=144)
  ->  Sort  (cost=1131.50..1131.51 rows=1 width=144)
        Sort Key: a.*
        ->  Nested Loop  (cost=12.95..1131.49 rows=1 width=144)
              ->  Nested Loop  (cost=12.95..95.36 rows=3 width=152)
                    ->  Hash Semi Join  (cost=12.95..70.11 rows=37 width=148)
                          Hash Cond: (a.cod_almox = ao.cod_almox)
                          ->  Seq Scan on tb_almox a  (cost=0.00..55.44 rows=482 width=144)
                                Filter: (cod_status = 17510)
                          ->  Hash  (cost=12.45..12.45 rows=40 width=4)
                                ->  Seq Scan on tb_almox_orgao ao (cost=0.00..12.45 rows=40 width=4)
                                      Filter: (cod_orgao = 59)
                    ->  Index Only Scan using tb_user_almox_pk on tb_user_almox ua
(cost=0.00..0.67 rows=1 width=4)
...
```

Desempenho
Explain

Seq Scan: Tabela pequena, sempre irá fazer a varredura dela completa

```
                                                QUERY PLAN
Unique  (cost=1131.50..1131.55 rows=1 width=144)
  ->  Sort  (cost=1131.50..1131.51 rows=1 width=144)
        Sort Key: a.*
        ->  Nested Loop  (cost=12.95..1131.49 rows=1 width=144)
              ->  Nested Loop  (cost=12.95..95.36 rows=3 width=152)
                    ->  Hash Semi Join  (cost=12.95..70.11 rows=37 width=148)
                          Hash Cond: (a.cod_almox = ao.cod_almox)
                          ->  Seq Scan on tb_almox a  (cost=0.00..55.44 rows=482 width=144)
                                Filter: (cod_status = 17510)
                          ->  Hash  (cost=12.45..12.45 rows=40 width=4)
                                ->  Seq Scan on tb_almox_orgao ao (cost=0.00..12.45 rows=40
width=4)
                                      Filter: (cod_orgao = 59)
                    ->  Index Only Scan using tb_user_almox_pk on tb_user_almox ua
(cost=0.00..0.67 rows=1 width=4)
...
```

Desempenho
Explain




```
                                                    QUERY PLAN
 Unique  (cost=51.24..51.29 rows=1 width=144)
   ->  Sort  (cost=51.24..51.24 rows=1 width=144)
         Sort Key: a.*
         ->  Nested Loop  (cost=12.95..51.23 rows=1 width=144)
               Join Filter: (ua.cod_almox = a.cod_almox)
               ->  Nested Loop  (cost=12.95..48.13 rows=1 width=12)
                     ->  Hash Join  (cost=12.95..25.97 rows=8 width=8)
                           Hash Cond: (f.cod_almox = ao.cod_almox)
                           ->  Index Scan using
idx_fechamentomensalentrada_mesanoref_2016 on tb_fechamento_mensal_entrada f
(cost=0.00..12.68 rows=74 width=4)
                           ->  Hash  (cost=12.45..12.45 rows=40 width=4)
                                 ->  Seq Scan on tb_almox_orgao ao  (cost=0.00..12.45 rows=40
width=4)
                                       Filter: (cod_orgao = 59)
...
```

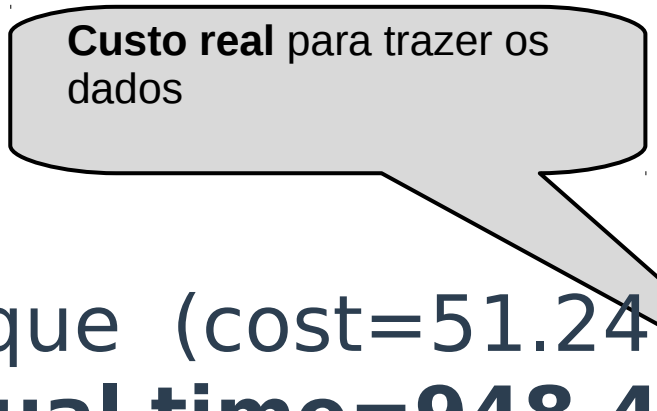
Desempenho
Explain - Comparação

- **Estimativa do plano de execução**
  - Original
    - Unique (cost=1131.50..1131.55 rows=1 width=144)
  - Após criado o índice
    - Unique (cost=51.24..51.29 rows=1 width=144)

- Redução na estimativa de execução em **22x**

Desempenho

Explain Analyze - Executa de fato a consulta



QUERY PLAN

Unique (cost=51.24..51.29 rows=1 width=144) (**actual time=948.453..979.903** rows=40 loops=1)

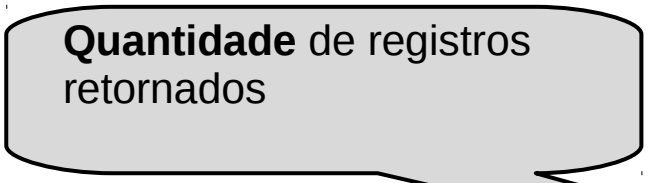
-> Sort (cost=51.24..51.24 rows=1 width=144) (actual time=948.452..950.037 rows=15016 loops=1)

Sort Key: a.*

Sort Method: quicksort Memory: 4373kB

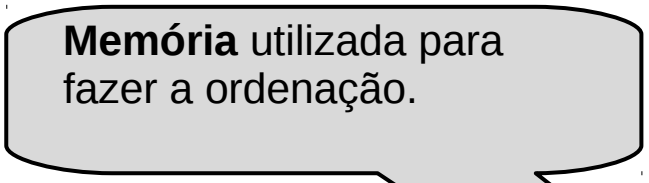
...

Desempenho
Explain Analyze




```
                                QUERY PLAN
 Unique  (cost=51.24..51.29 rows=1 width=144)
(actual time=948.453..979.903 rows=40 loops=1)
   ->  Sort  (cost=51.24..51.24 rows=1 width=144)
(actual time=948.452..950.037 rows=15016
loops=1)
         Sort Key: a.*
         Sort Method: quicksort  Memory: 4373kB
...
```

Desempenho
Explain Analyze




```
                                QUERY PLAN
 Unique  (cost=51.24..51.29 rows=1 width=144)
(actual time=948.453..979.903 rows=40 loops=1)
   ->  Sort  (cost=51.24..51.24 rows=1 width=144)
(actual time=948.452..950.037 rows=15016
loops=1)
         Sort Key: a.*
         Sort Method: quicksort  Memory: 4373kB
...
```

Desempenho
Explain Analyze

Tempo estimado: **51ms**

Tempo real gasto: **979ms**

- **Diferença pode ocorrer por:**
  - Os dados não estavam em memória e o plano de execução esperava que estivessem
  - Má configuração dos custos (valores padrões)
    - seq_page_cost = 1.0 (disco)
    - random_page_cost = 4.0 (disco)
    - cpu_tuple_cost = 0.01 (cada linha)
    - cpu_index_tuple_cost = 0.005 (cada entrada no índice)
    - cpu_operator_cost = 0.0025 (operador ou função)
  - Má configuração do autovacuum que tem a função de limpeza das dead tuples e atualização das estatísticas
  - Falta de atualização das estatísticas: ANALYZE ou VACUUM ANALYZE

Desempenho
Like

- **%LIKE%**
  - Não consegue utilizar o índice, por não saber por onde iniciar a leitura = Lento
  - Similaridade: pg_trgm
- **LIKE%**
  - Criar índice a ser utilizado no like%
    - varchar → varchar_pattern_ops
    - char → bpchar_pattern_ops
    - text → text_pattern_ops

```
CREATE INDEX idx_tab_nome ON tab (nome varchar_pattern_ops);
```

SELECT id, nome FROM tab WHERE nome LIKE 'Francisco%'; → Utiliza o índice

Desempenho
Busca Textual – FTS - Tsearch2

- **Pesquisa semelhante ao %LIKE%**
- **Pesquisa por radicais**
  - Palestra: palestr
  - Francisco: francisc
- **Faz pesquisa em texto e pode utilizar índices, tornando rápida a consulta**
- **Índices do tipo GIN/GIST**

Desempenho
Índice parcial

- **É criado em relação a um subconjunto de dados da tabela baseado em uma expressão condicional.**

```
CREATE INDEX idx_data_2015 ON tb_tabela (data)
  WHERE ((data >= '2015-01-01 00:00:00')
     AND (data <= '2015-12-31 23:59:59'));

CREATE INDEX idx_data_2016 ON tb_tabela (data)
  WHERE ((data >= '2016-01-01 00:00:00')
     AND (data <= '2016-12-31 23:59:59'));
```

  - Se utilizado no *where data = '2016-03-03'* pode utilizar o índice idx_data_2016

Desempenho
Clusterização

- **Organiza fisicamente uma tabela de acordo com o índice**

```
CLUSTER tabela USING idx_tabela_id3;
```

- **Os novos dados ou atualizações que ocorrerem após clusterizado a tabela serão adicionados ao final da tabela.**
- **Necessário refazer o cluster de tempos em tempo para reordenar os dados fisicamente e melhorar o desempenho.**

Desempenho
Particionamento de tabelas

- Dividir tabelas enormes em tabelas grandes
- Utiliza herança de tabelas
- Indicado para tabelas com mais de 10GB

Desempenho
PT: Vantagens

- **O índice e a tabela podem ser carregados em memória**
- **Transparente para a aplicação**
  - ➔ Continua chamando a mesma tabela
- **Separa dados que raramente são usados**
- **Economia de espaço no backup**
- **Desempenho em leitura**
- **Barato para fazer expurgo**

Desempenho
PT: Desvantagens

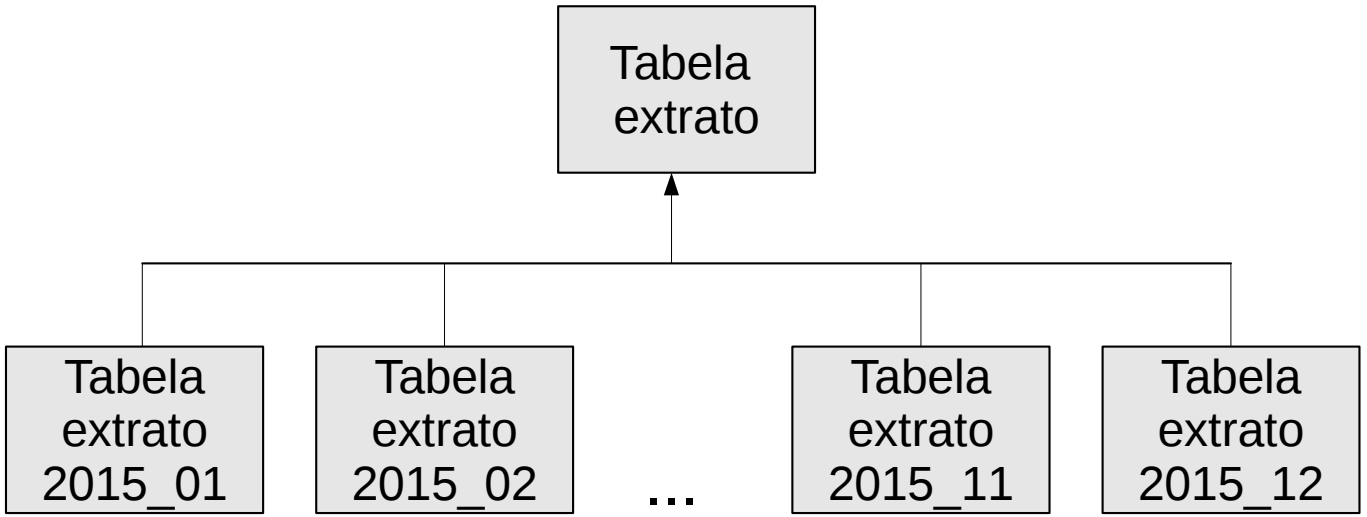
- **Para ter a melhora no desempenho é bom que a chave seja utilizada na grande maioria das consultas**
- **Problema: Ao inserir um registro na tabela retorna “INSERT 0 0”**
  - O registro foi inserido na tabela filha
  - Pode ser necessário tratar na aplicação o valor de retorno
- **Limitação: uma tabela não particionada não pode ter uma FK apontando para uma tabela particionada**

Desempenho
PT: Chave

- **Exemplos de chaves**
  - Por data:
    - data BETWEEN DATE ’2015-11-01′ AND DATE’2015-11-30′
    - data BETWEEN DATE’2015-12-01′ AND DATE’2015-12-31′
  - Por localização geográfica:
    - uf = ‘SP’
    - uf_pedido IN (‘AM’, ‘AC’, ‘RO’, ‘PA)
  - Por loja:
    - cod_loja = 1
    - cod_loja IN (3, 4, 5)
  - Por ano fiscal:
    - cod_ano_fis = 2014

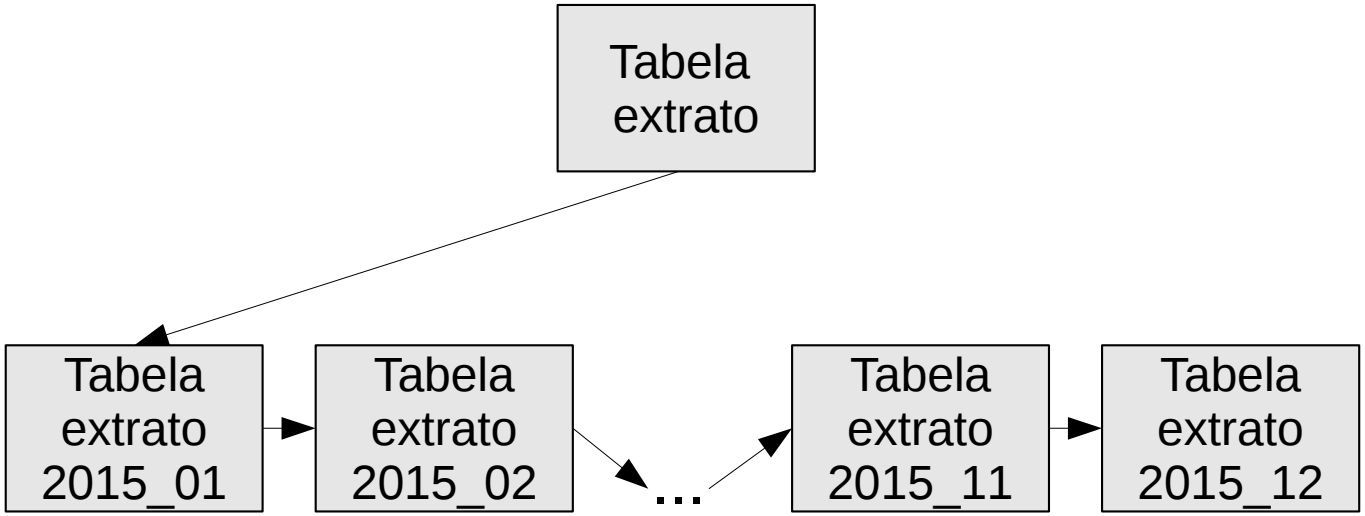
Desempenho
PT: Exemplo

- **Exemplo de uso**
  - Uma tabela com 12 milhões de registros em 2015
  - 12 tabelas com 1 milhão cada



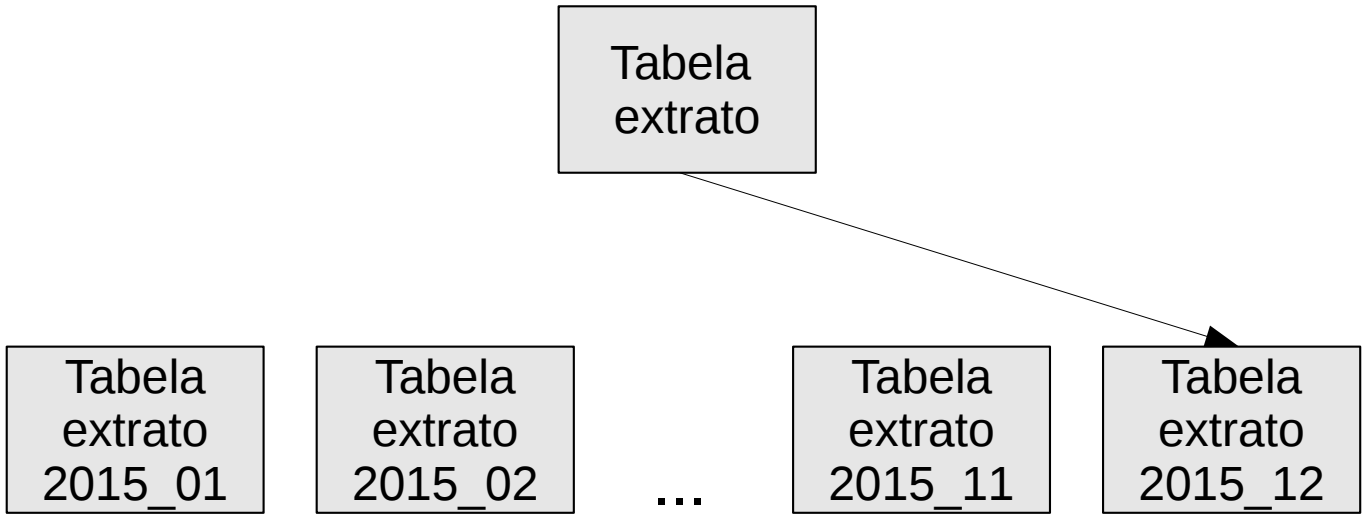
Desempenho
PT: Exemplo – sem chave

- **Consulta sem chave**
  - select data, saldo from tb_extrato;



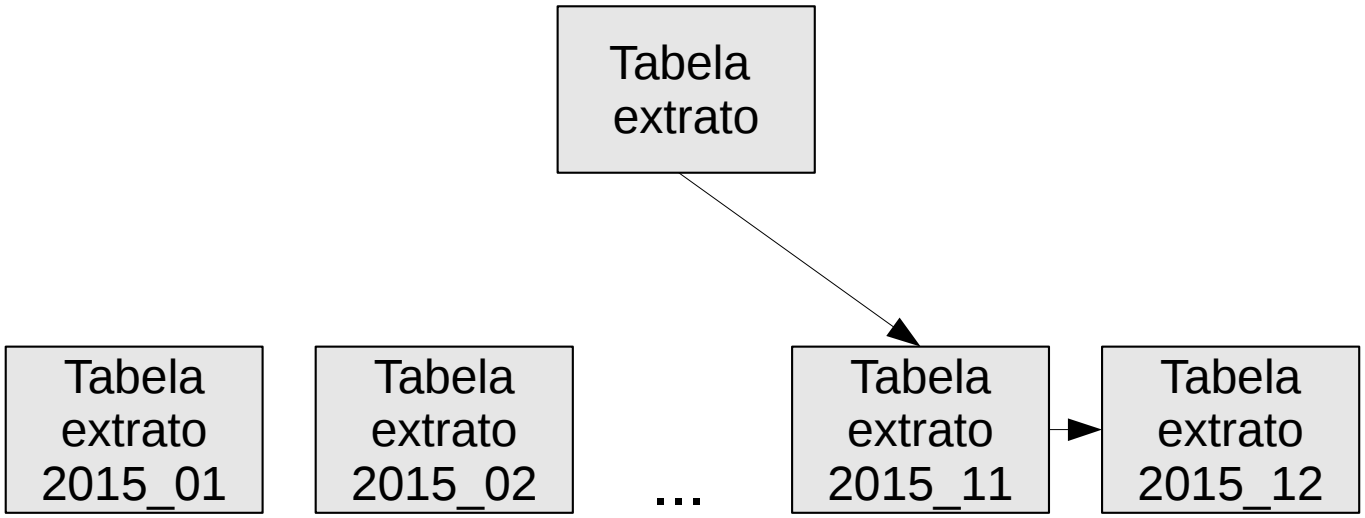
Desempenho
PT: Exemplo – específica

- **Consulta específica**
  - select data, saldo from tb_extrato where data = '2015-12-09';



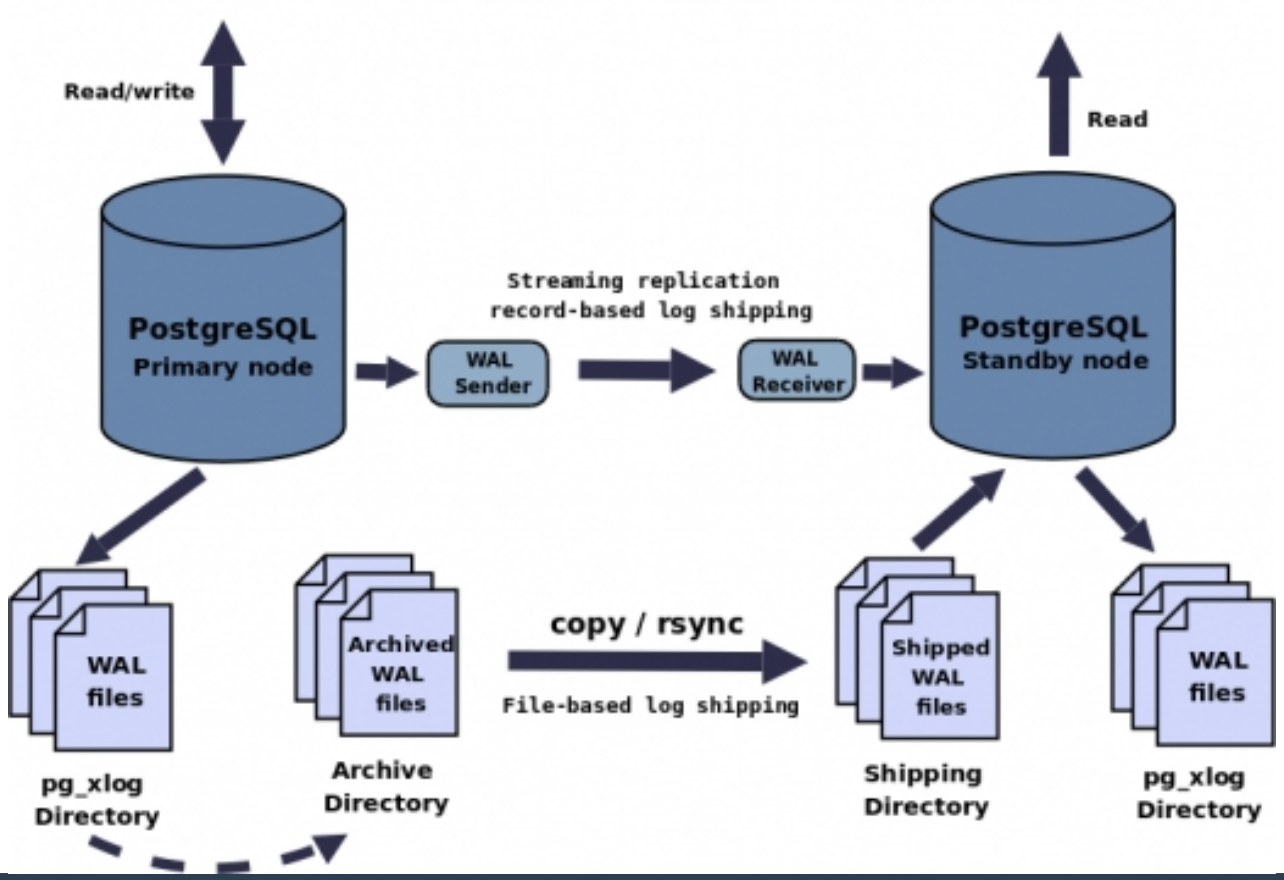
Desempenho
PT: Exemplo – 2 tabelas

- **Consulta em duas tabelas**
  - select data, saldo from tb_extrato where data between '2015-11-09' and '2015-12-09';



Desempenho
Hot Standby

- **Replicação com o Hot Standby**
  - Um servidor de leitura e escrita
  - Demais servidores de leitura



pgRelatórios

- **Centralizar a geração de relatórios de todos os servidores PostgreSQL em um único lugar.**
- **Atualizado diariamente.**
- **Retenção dos relatórios em semanas.**
- **Relatórios diários ou semanais.**

pgRelatórios

www.pgrelatorios.celepar.parana

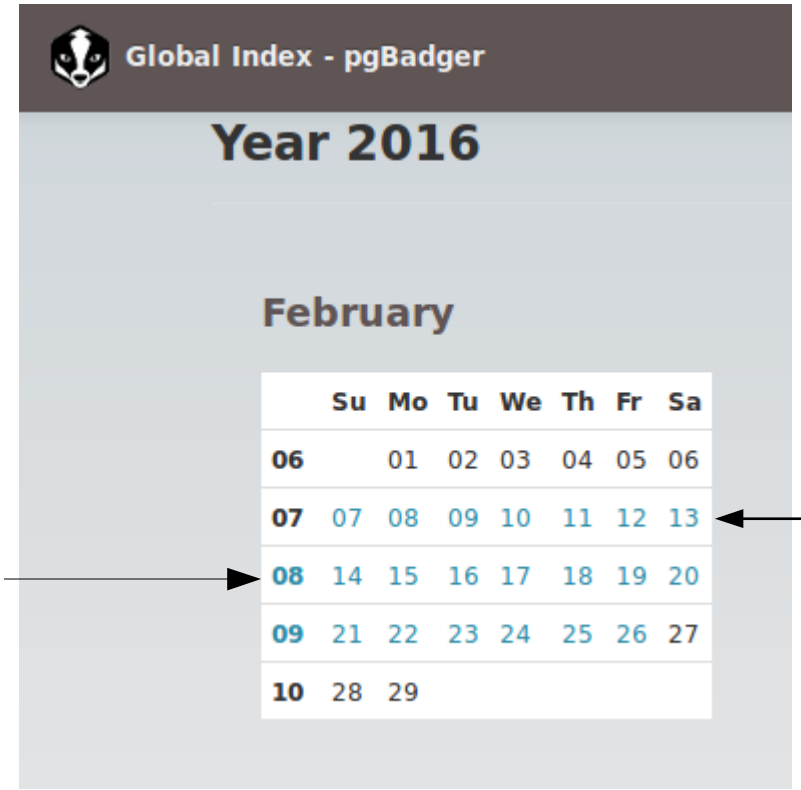
Relatórios do PostgreSQL para a Celepar.

Ocultar Todos / Expandir Todos

- Servidor: **bancos.corporativo.celepar.parana** - Relatório do servidor
  - cep
  - monitor
  - corporativo
- Servidor: **bancos.educacao.celepar.parana** - Relatório do servidor
- Servidor: **bancos.email.celepar.parana** - Relatório do servidor
- Servidor: **bancos.secseguranca.celepar.parana** - Relatório do servidor
- Servidor: **bancos.detranpr.celepar.parana** - Relatório do servidor
  - banco1
  - banco2
  - banco3
- Servidor: **bancos.sefa.celepar.parana** - Relatório do servidor
- Servidor: **bancos.sesa.celepar.parana** - Relatório do servidor
- Servidor: **bancos.geo.celepar.parana** - Relatório do servidor
- Servidor: **bancos.bi.celepar.parana** - Relatório do servidor
- Servidor: **bancos.dw.celepar.parana** - Relatório do servidor
- Servidor: **bancos.porto.celepar.parana** - Relatório do servidor
  - bancoA
  - bancoB
  - bancoC
  - bancoD
- Servidor: **bancos.telefonia.celepar.parana** - Relatório do servidor
- Servidor: **bancos.telecentros.celepar.parana** - Relatório do servidor
- Servidor: **bancos.saude2.celepar.parana** - Relatório do servidor
- Servidor: **bancos.sesp.celepar.parana** - Relatório do servidor
- Servidor: **bancos.corporativo2.celepar.parana** - Relatório do servidor
- Servidor: **bancos.terceiros.celepar.parana** - Relatório do servidor

Desenvolvido por ChicO - Novembro de 2015 - v1.1

pgRelatórios



pgBadger

- **Gerador de relatórios baseado nos logs do PostgreSQL.**
- **O relatório é baseado no que foi logado e não em tudo o que aconteceu no servidor.**
- **CUIDADO: Se ativar para logar todas as consultas, o consumo de processamento e espaço em disco cresce muito. Pode transformar o servidor de banco banco de dados em um gerador de logs.**

PgBadger
Menu e Overview

pgBadger | Overview | Connections | Sessions | Checkpoints | Temp Files | Vacuums | Locks | Queries | Top | Events

Global Stats

Queries | Events | Vacuums | Temporary files | Sessions | Connections

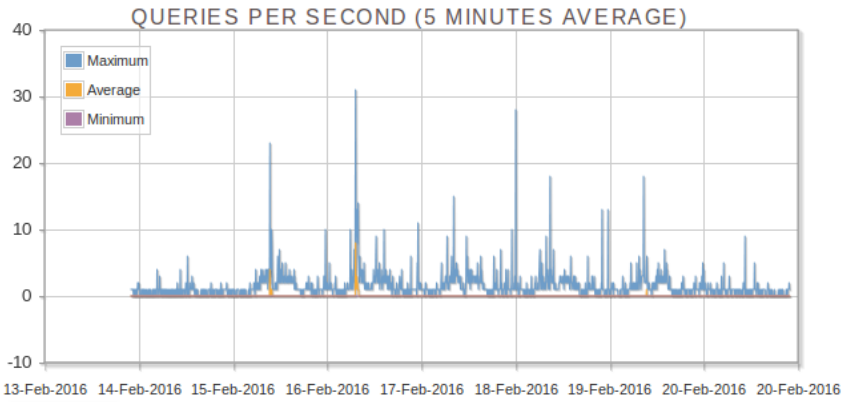
1,524 Number of unique normalized queries	**35,950** Number of queries	**4d10h10m6s** Total query duration	**2016-02-14 00:00:13** First query	**2016-02-20 23:59:56** Last query	**31 queries/s at 2016-02-16 09:18:04** Query peak

SQL Traffic

KEY VALUES

31 queries/s
Query Peak

2016-02-16 09:18:04
Date



Download

PgBadger
Queries

Queries by type

KEY VALUES

35,127
Total read queries

790
Total write queries

Chart | Table

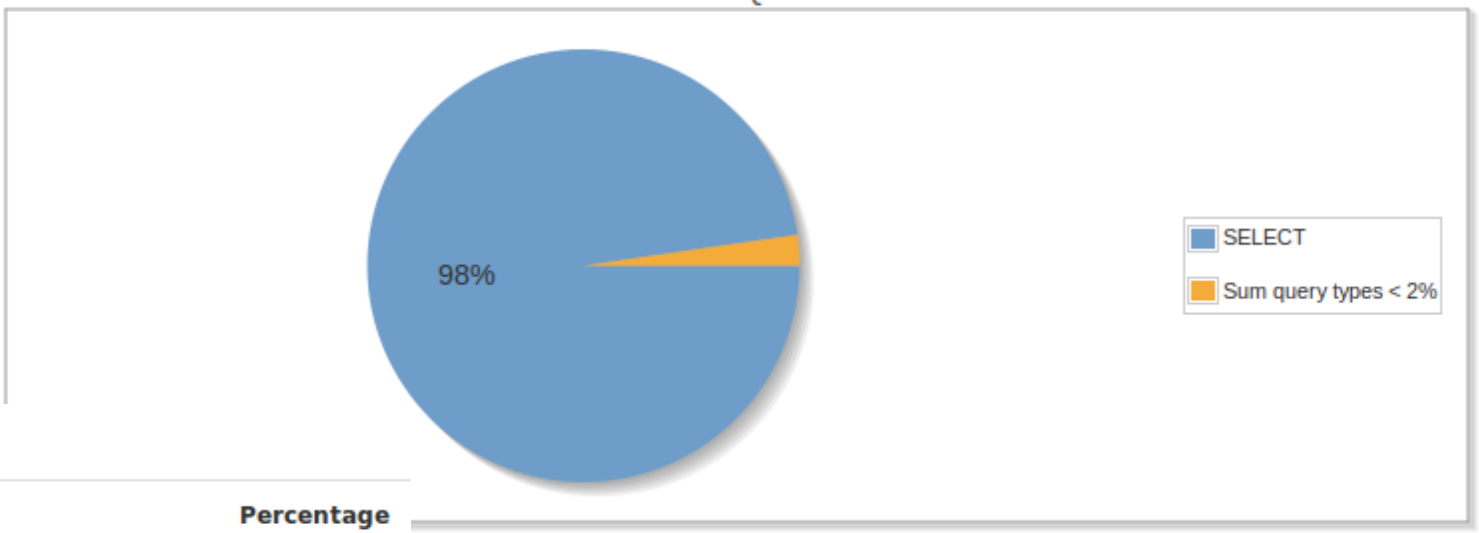


Chart | Table

Type	Count	Percentage
SELECT	35,127	97.71%
INSERT	187	0.52%
UPDATE	82	0.23%
DELETE	521	1.45%
OTHERS	33	0.09%

PgBadger
Time consuming queries

Time consuming queries

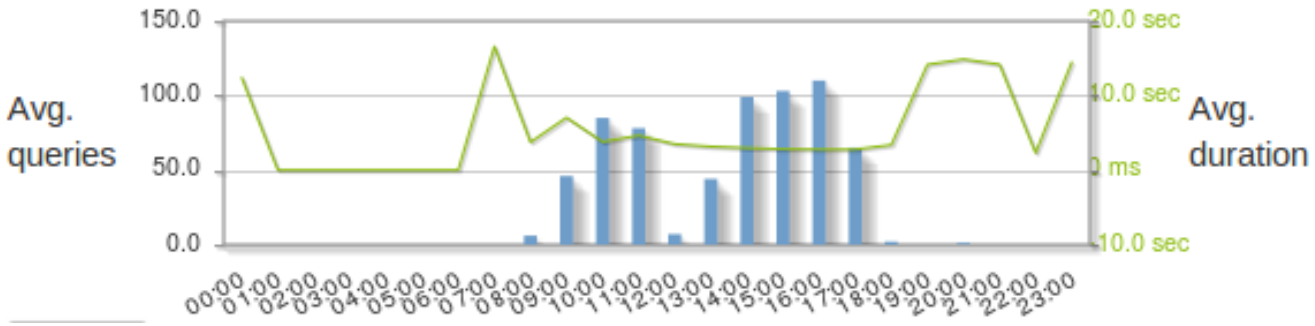
Rank	Total duration	Times executed	Min duration	Max duration	Avg duration	Query
1	19h7m42s	132 Details	1m25s	19m39s	8m41s	SELECT *count* (*) AS col_0_0_ FROM sa_sistema.tb_agentetransito agentetran0_ LEFT OUTER JOIN sa_sistema.tb_agenteautuador agentesaut1_ ON agentetran0_.codagentetransito = agentesaut1_.codagentetransito LEFT OUTER JOIN sa_sistema.tb_unidadefiscalizacao unidadefis2_ ON agentesaut1_.codunidadefiscalizacao = unidadefis2_.codunidadefiscalizacao WHERE agentetran0_.codorgao = '' AND (agentesaut1_.dthrinicioooperacao IS NULL OR agentesaut1_.dthrinicioooperacao = (SELECT max (agenteautu3_.dthrinicioooperacao) FROM sa_sistema.tb_agenteautuador agenteautu3_ WHERE agenteautu3_.codagentetransito = agentetran0_.codagentetransito)); Examples User(s) involved
2	18h36m10s	131 Details	1m21s	19m51s	8m31s	SELECT (agentetran0_.rgagente || agentetran0_.ufrgagente) AS col_0_0_, agentetran0_.tipoagente AS col_1_0_, agentesaut1_.dthrinicioooperacao AS col_2_0_, unidadefis2_.numunidadefiscalizacao AS col_3_0_, unidadefis2_.nomeunidadefiscalizacao AS col_4_0_, agentetran0_.codagentetransito AS col_5_0_, unidadefis2_.codunidadefiscalizacao AS col_6_0_, agentetran0_.nomeagente AS col_7_0_, agentetran0_.identificacaoagente AS col_8_0_, agentesaut1_.dthrfimoperacao AS col_9_0_, agentetran0_.cpfagente AS col_10_0_, agentesaut1_.observacaosituacao AS col_10_0_ FROMsa_sistema.tb_agentetransito agentetran0_ LEFT OUTER JOIN sa_sistema.tb_agenteautuador agentesaut1_ ON agentetran0_.codagentetransito = agentesaut1_.codagentetransito LEFT OUTER JOIN sa_sistema.tb_unidadefiscalizacao unidadefis2_ ON agentesaut1_.codunidadefiscalizacao = unidadefis2_.codunidadefiscalizacao WHERE agentetran0_.codorgao = '' AND (agentesaut1_.dthrinicioooperacao IS NULL OR agentesaut1_.dthrinicioooperacao = (SELECT max (agenteautu3_.dthrinicioooperacao) FROM sa_sistema.tb_agenteautuador agenteautu3_ WHERE agenteautu3_.codagentetransito = agentetran0_.codagentetransito)) ORDER BY agentetran0_.nomeagente LIMIT '';

PgBadger
Most frequent queires

Most frequent queries (N)

Rank	Times executed	Total duration	Min duration	Max duration	Avg duration	Query
1	4,557 Details	4h31m1s	2s	33s949ms	3s568ms	SELECT this_.codsubstitutoficha AS codsubst1_30_10_, this_.dataho this_.codusuario AS codusuario30_10_, this_.codhistoricoprocuradorf this_.codsubstitutosindisponibilidade AS codsubst7_30_10_, usuario3 usuario3_.cartorio AS cartorio40_0_, usuario3_.tabelionato_codtabel usuario3_.usuarioatualizacao AS usuarioa6_40_0_, usuario3_.dataatua comarca4_.codprocuradoria AS codprocu2_5_1_, comarca4_.nome AS nome comarca4_.codtabeltj AS codtabeltj5_1_, comarca4_.virtual AS virtua numeropr9_5_1_, comarca4_.pro_ultimoseqremessa_h8 AS pro10_5_1_, co

TIMES REPORTED MOST FREQUENT QUERIES #1



PgBadger
Most frequent error

⚠ Most Frequent Errors/Events

KEY VALUES

166
Max number of times the same event was reported

1,145
Total events found

Rank	Times reported	Error
1	166 Details	`ERROR: missing FROM-clause entry for table "..."` Examples
2	142 Details	`ERROR: canceling statement due to statement timeout` Examples
3	137 Details	`ERROR: null value in column "..." violates not-null constraint` Examples
4	86 Details	`ERROR: current transaction is aborted, commands ignored until end of transaction block` Examples
5	76 Details	`ERROR: update or delete on table "..." violates foreign key constraint "..." on table "..."` Examples
6	75 Details	`ERROR: value too long for type character varying(...)` Examples

Perguntas??

Francisco Summa Netto
franciscosummma@celepar.pr.gov.br