Meus casos e cases com Particionamento

PostgreSQL 9.2, 9.6

PostgreSQL 10 e 11?

Cronograma

- Eu
- Particionamento
- Prós
- Contras
- CASE: O que sofri!?!?!?
- Perguntas

Eu

- Fernando Franquini
  - Capin
- DBA @ DragonBD
- DBA SRE Sênior @ Senior Sitemas
- Mais de XX anos experiência e XX anos atuando com Banco de Dados
- Pai de duas lindas meninas, Catarinense que mora em Floripa/SC

Particionamento

Existe desde a versão 8.1 do Postgresql utilizando herança e nas consultas exclusão por restrição;

Lembrando que são percalços que passei, podem e devem existir outros mais;

Os cases nessa apresentação tratam de algumas versões Postgresql:

- 9.2 (sim ainda existe isso em produção)
- 9.6
- 10
- 11

Particionamento

Particionamento nada mais é que uma forma ‘simples’ de dividirmos nossos dados

Utiliza herança entre a tabela Pai e Filhas

Alterações na pai reflete nas filhas

Na versão 9.6 ainda é da mesma forma. São utilizadas funções e triggers para ‘enviar’ os dados as tabelas corretas;

Na versão 10 a melhoria significativa que foi o particionamento declarativo, não sendo mais necessárias funções e triggers;

Na versão 11 foi adicionado o particionamento por HASH, somando aos RANGE e LIST das versões anteriores. Nesta versão o HASH pode receber UPDATE;

Prós

- Melhoria performance consulta (se utilizar a chave de partição no where), tamanhos menores de tabelas;
- Facilita a manutenção das tabelas (vacuum ... analyze ...);
- Pode remover tabela antiga sem impacto;
- Podem ser feitos backups de tabelas de histórico (ex.: dados que precisam ser armazenados por muitos anos);
- A engine do Postgresql tem melhorada ganhando performance nas buscas e nos inserts;

Contras

- Falta ‘PK’ Global
- Algumas dificuldades com ORMs (como Hibernate)
- Índices e constraints podem varias de partição a partição;
- Não exporta a PK como FK;
- Criação de índices por partição;


CASES:
Dividir para conquistar!

CASE Bilhetagem de Telefonia

- Ambiente RDS – AWS – 9.6
- Tenant: ID
- Desenvolvimento: Java + ORM Hibernate
- POC:
  - Cenário do particionamento por DATA
  - Cenário do particionamento por ID
- Analise das quantidades de partições e volume de dados
- Criação das partições e triggers
- Migrar os dados e testes (performance das consultas problemáticas)

CASE Bilhetagem de Telefonia

ORM Hibernate:

- Problemas com a Trigger na tabela:
  - *Batch update returned unexpected row count from update [0]; actual row count: 0; expected: 1*
  - Na função: RETURN NULL
- Solução:
  - Criar uma View (Select * from TABELA_PAI_PARTICIONADA)
  - Criar trigger na view (INSTEAD OF)
  - Na função: RETURN NEW

Obs.: Não obtive este problema em cliente que utiliza PHP, pois não utiliza ORM.

CASE Bilhetagem de Telefonia

- O resultado final do particionamento por ID em produção
  - Resultado: 332 partições
  - Performance: boa – muito boa (melhora analisada via SQL entre 40% e 75%)
  - Extra: criada uma nova trigger para gerar a partição a cada novo ID
  - Umas 2 partições ficaram maiores uma com 18 milhões e outra 24 milhões
- A implantação em produção é praticamente 0 down time – rename table.

CASE Transação Cartões

- Ambiente RDS – AWS – 9.6
- Tenat: Schema
- Desenvolvimento: Java + ORM Hibernate
- Particionamento por DATA (ano_mes)
- Funções e trigger;
- Criação da view (INSTEAD OF) para contornar o ‘erro’ do Hibernate
- Validação e testes com sucesso

CASE Transação Cartões

- Entrou em produção? **Não**

Motivos do porque não foi implementado:

- Desconfiança na “*sequence*” para geração da PK (falta de PK ‘golbal’)
- Por ser Tenant por schema não havia ‘ainda’ problema de lentidão
- Exportar FK

Ob.: Produto desenvolvido do ZERO

CASE Dados Jurídicos

- PostgreSQL – ON PRIMISE – 9.2
- Desenvolvimento: ASPNet (projeto de 2009/2010)
- Particionamento por DATA
- Funções e trigger;
- Tabela A (partição por ANO_QUARTO):
  - Numero partições: 68
  - As maiores tabelas tem 31Gb, 23Gb e 21Gb
  - Tamanho tabela: 445GB e 582.641.014 de linhas

CASE Dados Jurídicos

- Tabela NE (partição por ANO):
  - Número partições: 17
  - As maiores tabelas tem 7GB e 2 de 5GB
  - Tamanho tabela: 94GB e 229.561.157 de linhas
- Tamanho total da base: 1.3Tb
- Esse ambiente tem picos de 7 a 10 mil usuários simultâneos;

CASE Dados Jurídicos

- Partições mesmo grandes desempenhavam bem;
- Manutenção (vacuum ... analyze ... reindex ..) era feitas frequentemente;
- Problema desse tipo de informação por DATA é que sempre tem dado novo em várias partições;
- **Sem constraints;**

CASE Buscas por Robos

- PostgreSQL – Nuvem Privada – 9.6 (migrado para EC2 – AWS – 11)
- Desenvolvimento: Java
- Particionamento? ‘Na mão’
- Script *crontab* movimenta os registros mensalmente (primeiro domingo);
- Números:
  - Número de ‘partições’: 5
  - Fila: 25.166.663
  - Fila_Cancelada: 9.873.976
  - Fila_Completa: 42.059.350
  - Fila_Interna: 24.817.731

CASE Buscas por Robos

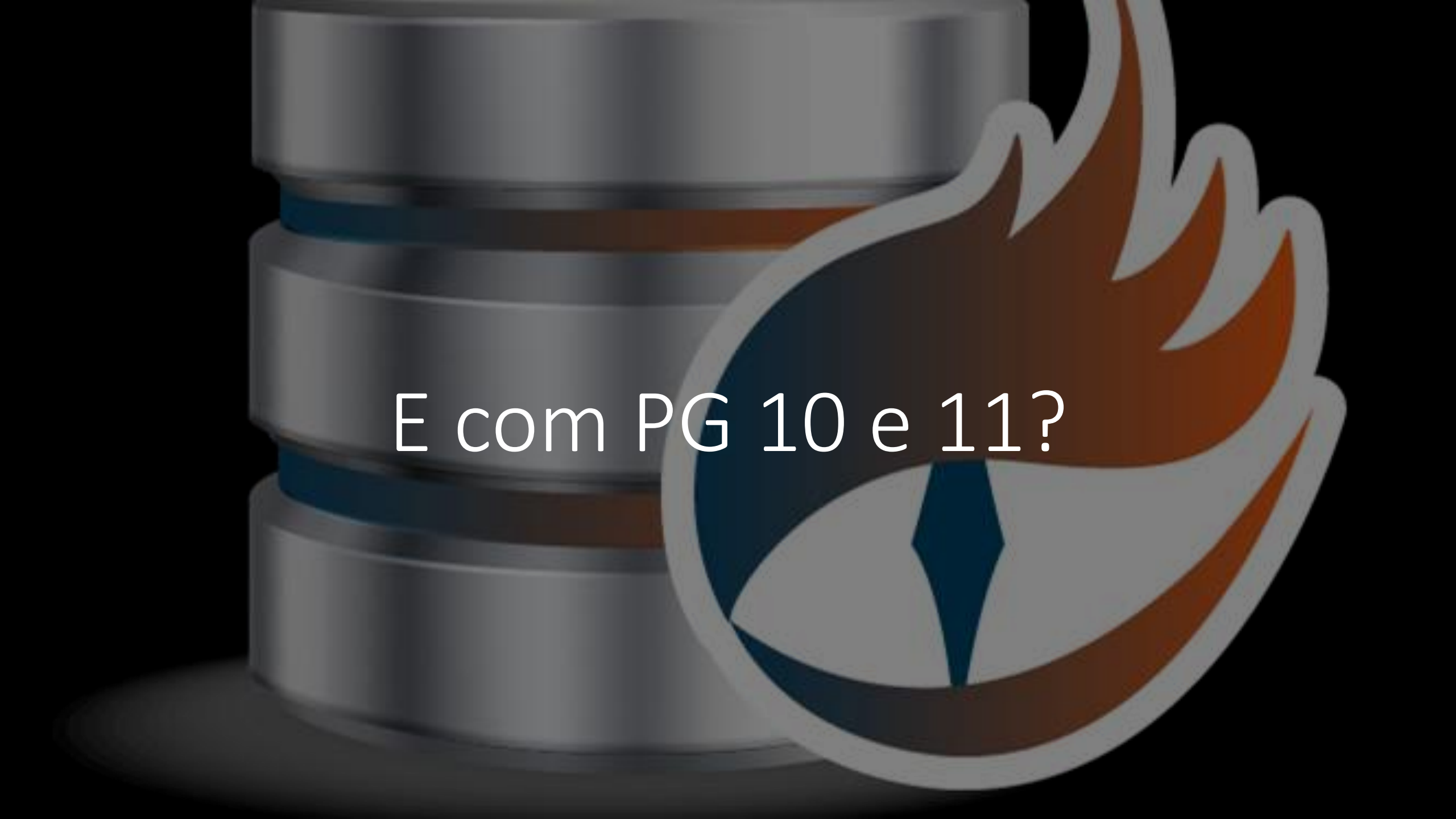
- Percebemos que não são tabelas imensas, porém a quantidade de requisições é alta (robôs);
- Criadas com o conceito de herança;
- As consultas são em cima de funções com regra de negócio (pouco lentas);
- Ambiente de captura de dados controlado podendo ter downtime;
- Ao fazer a movimentação roda vacuum full e reindex (downtime);
- Índices diferentes entre partições;
- E agora, porque ‘diabos’ foi feito na mão?

CASE Tenant Schema

- PostgreSQL – AWS – 9.6
- Desenvolvimento: Java + Python
- Particionamento? *Schemas*
- Números:
  - Número de ‘partições’: + 25mil
- Tabelas pequenas, cada schema atende cada cliente
- Velocidade de acesso a

CASE Que vi por aí

- Backup de dados (dados estáticos)
- Expurgo de dados via flag


E com PG 10 e 11?

2ndQuadrant PostgreSQL

Feature	PG9.6	PG10	PG11
Declarative Partitioning	✗	✓	✓
Auto Tuple Routing – INSERT	✗	✓	✓
Auto Tuple Routing – UPDATE	✗	✗	✓
Optimizer Partition Elimination	✓[1]	✓[1]	✓
Executor Partition Elimination	✗	✗	✓[2]
Foreign keys	✗	✗	✓[3]
Unique indexes	✗	✗	✓[4]
Default Partitions	✗	✗	✓
Hash Partitions	✗	✗	✓
FOR EACH ROW triggers	✗	✗	✓
Partition-level joins	✗	✗	✓[5]
Partition-level aggregation	✗	✗	✓
Foreign partitions	✗	✗	✓
Parallel Partition Scans	✗	✗	✓

https://www.2ndquadrant.com/en/blog/partitioning-evolution-postgresql-11/

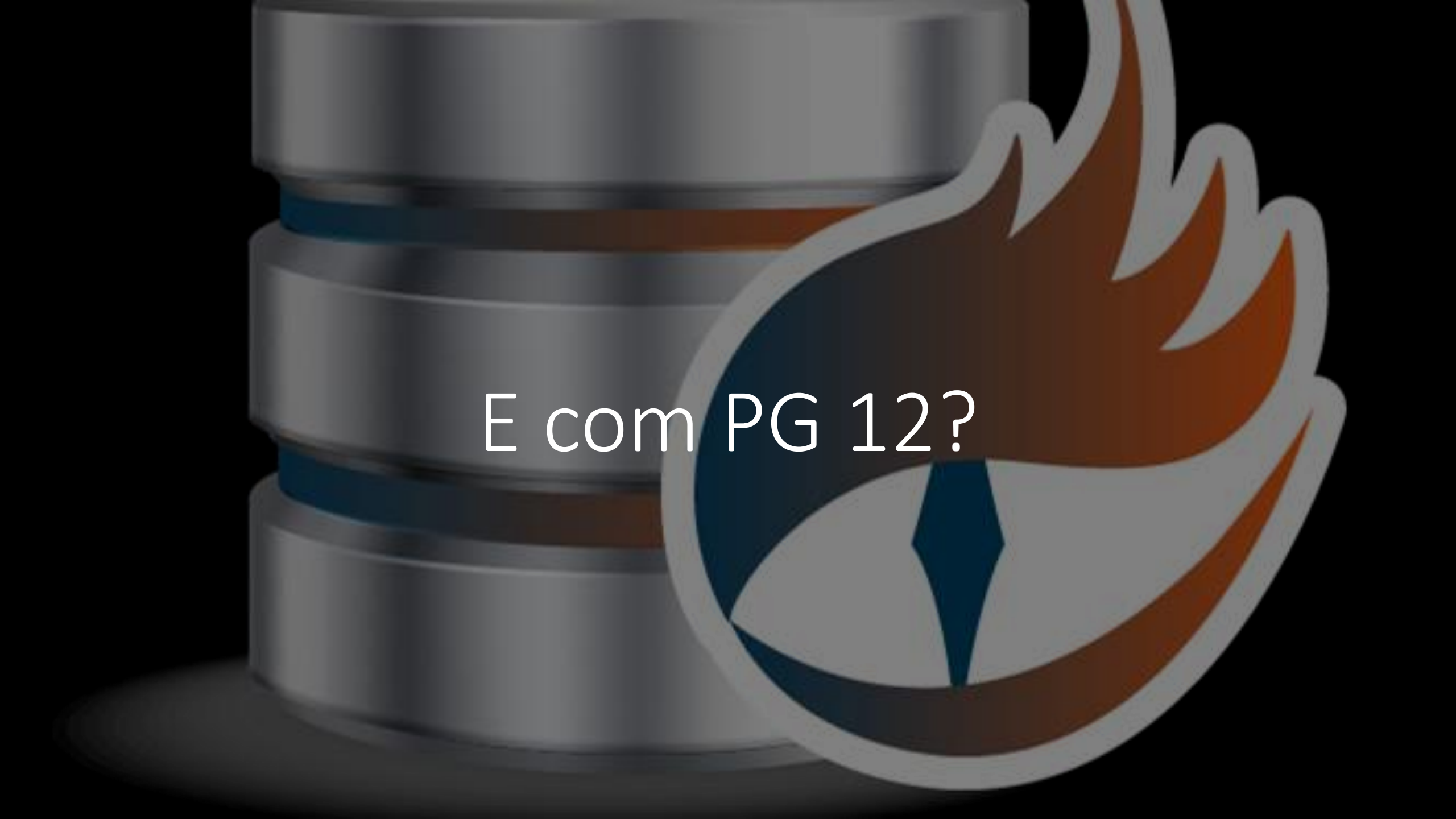
1. Using constraint exclusion
2. Append nodes only
3. On partitioned table referencing non-partitioned table only
4. Indexes must contain all partition key columns
5. Partition constraint on both sides must match exactly

E com PG 10 e 11?

- Da versão 9.X para 10:
  - Ainda teria o problema do ORM Hibernate;
  - Problema da ‘PK global’;
  - Sem exportar FK;
  - Ainda criação de índices por partição, podendo ficar diferentes;

E com PG 10 e 11?

- Da versão 9.X para 11:
  - **Melhoria de performance;**
  - Temos PK! :D
  - Poderemos ter também as Uks;
  - Criação de índices na tabela pai aplicados automaticamente nas filhas;
  - Update no registro move de partição;
  - Sem exportar FK :(


E com PG 12?

E com PG 12?

- Da versão 9.X para 12:
  - Teremos como exportar FK :)
  - **Melhoria de performance;**
  - **Melhoria de performance;**

Perguntas

- **Contato:**
- **fernando.franquini@gmail.com**
- **http://linkedin.com/in/capin**