

Consultas Aproximadas

Euler Taveira de Oliveira
Diogo Biazus

PostgreSQL Brasil

26 de setembro de 2008

Resumo

- 1 Introdução
- 2 PostgreSQL
- 3 Pesquisa

Consulta Aproximada

- banco de dados trabalha com um modelo de consultas "exatas" (chave = valor)
- queremos fazer consultas *quase* exatas (chave \approx valor)
- similaridade: $\phi > \text{limiar}$

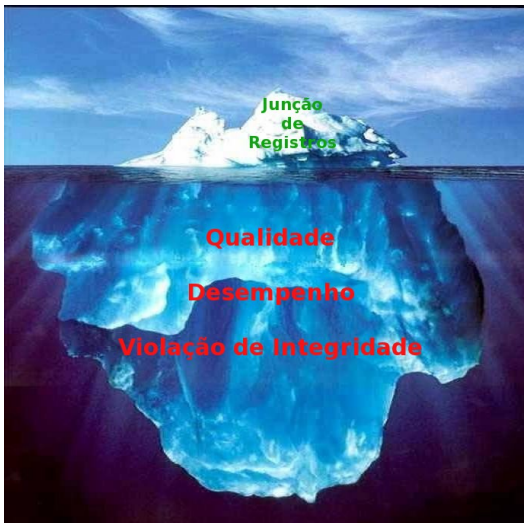
Problemas

- humano
 - entrada incorreta
 - informação imprecisa (abreviaturas, omissão, ...)
 - desinformação
- aplicação
 - aplicações com problemas
 - modelo de dados é falho
 - restrições inexistentes ou imprecisas
 - chaves estrangeiras inexistentes
- obsolência
 - mundo real é dinâmico!

Consequências

- pobre qualidade dos dados
 - 2% dos registros estão obsoletos em registros de consumidores após 1 mês [DWI02]
 - Bill Clinton, William Jefferson Clinton, William Jefferson Blythe III: mesma pessoa?
- grande impacto financeiro
 - U\$ 611 bi/ano perdido nos USA devido a pobre qualidade dos dados de consumidores [DWI02]
 - U\$ 2,5 bi/ano perdido devido a preços incorretos nos BDs de lojas de varejo [E00]

Solução?



Resumo

- 1 Introdução
- 2 PostgreSQL
- 3 Pesquisa

O que utilizar?

- expressão regular
- LIKE e ILIKE
- fuzzystrmatch
- text search
- indexação
- pg_trgm

Expressão Regular

- casar conjunto de dados que estão em um mesmo padrão
- problema: não casa inversões e erros ortográficos

Exemplo

```
meubd=# select * from foo where nome ~ '^euler';  
meubd=# select * from foo where nome ~ '^Euler';  
meubd=# select * from foo where nome ~* '^Euler';
```

[I]LIKE

- casamento parcial de dados
- problema: não casa inversões e erros ortográficos

Exemplo

```
meubd=# select * from foo where nome LIKE 'euler%';  
meubd=# select * from foo where nome LIKE 'Euler%';  
meubd=# select * from foo where nome ILIKE 'Euler%';
```

fuzzystrmatch

- levenshtein
- metaphone
- soundex
- problema: funções específicas do domínio do problema
- problema: custo computacional alto

Exemplo

```
meubd=# select * from foo where levenshtein(nome, 'eulerr  
taaveira') ≤ 3;
```

text search

- técnicas de recuperação de informações (motores de busca)
- resolver problemas de inversão (separar em termos)
- resolver problemas de flexão verbal e singular x plural (stemming)
- remover palavras irrelevantes (stopwords)
- problema: erros ortográficos

Exemplo

```
meubd=# select * from foo where nome @@ 'euler & oliveira'  
meubd=# select * from foo where nome @@ 'euler |diogo'
```

text search: índices

- acelerar consultas
- índices B-Tree e Hash não resolvem!

Índices

- GIN (Generalized Inverted Index)
- GiST (Generalized Search Trees)

pg_trgm

- funções e operadores
- casar texto baseado em trígama
- euler = ("e", "eu", eul, ule, ler, "er ")

Utilizando o módulo

```
CREATE TABLE foo (nome text);  
CREATE INDEX foo_idx ON foo USING gin (nome gin_trgm_ops);  
SELECT nome,similarity(nome, 'euler') AS sim FROM foo;
```

Resumo

- 1 Introdução
- 2 PostgreSQL
- 3 Pesquisa**

Minha Pesquisa

Idéias

- tempo de resposta baixo
- não utilizar pré-processamento (online)
- não utilizar estrutura auxiliar
- conservar qualidade dos resultados
- extensível

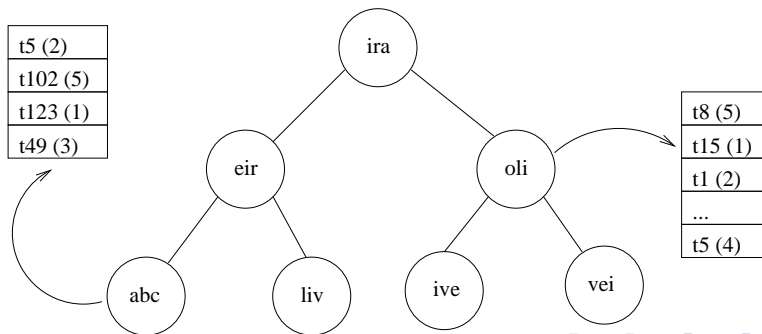
Proposta

- índice: filtragem
- operador: resolver falso positivos

Índice

Vantagens

- processamento: diminui o uso da função no operador
- extensibilidade: heurísticas de filtragem



Operador

Vantagens

- simplicidade: usuário não precisará implementar uma técnica
- extensibilidade: cada função de similaridade terá um operador

SELECT a, b, c FROM foo, bar WHERE foo.nome ∽ bar.nome (1)

Perguntas

?

Euler Taveira de Oliveira
euler@timbira.com
<http://www.timbira.com/>