

Você está em

DevMedia

Artigo

Otimização de consultas no PostgreSQL

Artigo da Revista SQL Magazine - Edição 39.



Marcado como lido



Anotar

Artigos



Banco de Dados



Otimização de consultas no PostgreSQL

A todo o momento, quando se trabalha no desenvolvimento de aplicações e/ou administração de dados, busca-se sempre uma melhora de desempenho. Ao trabalhar com consultas em bancos de dados deve-se ter uma atenção maior com relação à sua eficiência, uma vez que consultas mal elaboradas podem degradar consideravelmente o desempenho do sistema como um todo.



2



no PostgreSQL que auxiliam nesta tarefa de otimização e, também, fornecer uma base de conhecimento para que seja possível criar consultas mais inteligentes, refinadas, objetivando o ganho de performance.

Entendendo o plano de execução

Quando se executa uma operação no banco de dados, seja ela um `SELECT`, `INSERT`, ou outra qualquer, o PostgreSQL, assim como outros SGBDs, possui um mecanismo interno chamado planejador (ou otimizador), que reescreve a consulta com a intenção de aperfeiçoar os resultados, gerando um Plano de Execução.

O Plano de Execução, ou de Consulta, é uma seqüência de passos que serão executados pelo SGBD para executar uma consulta, ou seja, quais os tipos de processamento que serão feitos diretamente nos registros ou em estruturas de índices, bem como informações como o tempo de entrada, o tempo de resposta e o total de registros percorridos. O planejador precisa então fazer uso de estatísticas como o número total de registros da tabela, o número de blocos de disco ocupados por cada tabela e se há a presença de índices ou não.

Criando um banco de dados

Para exemplificar os trabalhos do plano de execução, será criado o banco de dados `ACADEMICO`, utilizando a ferramenta de interface do PostgreSQL, `pgAdmin III` (ver



Você está em

DevMedia

**Figura 1.** Criação do banco de dados Academico

Para a criação deste banco de dados foi utilizado o usuário Administrador como proprietário, WIN1252 como tipo de caractere padrão e o template sendo o default. O PostgreSQL possui uma propriedade (tablespace) que permite ao proprietário do banco definir onde os objetos da base de dados (como tabelas e índices) irão residir. No nosso caso foi usado o caminho padrão.

Na **Figura 2**, pode-se visualizar, em linguagem SQL, o script de criação do banco de dados. A **Listagem 1** apresenta o script de criação das tabelas do banco. Para os exemplos deste artigo, foram inseridos alguns poucos registros em cada uma destas

Você está em

DevMedia**Figura 2.** Script de criação do banco de dados

```
1  /* Criação da tabela Aluno (Aluno)
2  */
3  CREATE TABLE Aluno (
4      matricula_aluno          INTEGER          NOT NULL,
5      nome_aluno               VARCHAR(30)      NOT NULL,
6      end_logradouro           VARCHAR(30),
7      end_numero               INTEGER,
8      end_bairro               VARCHAR(20),
9      telefone_residencial     VARCHAR(15),
10     data_nascimento           DATE,
11     cod_curso                 INTEGER          NOT NULL);
12
13  /* Criação da tabela Curso (Curso)
14  */
```



2





Você está em

DevMedia

```
23      matricula_professor      INTEGER      NOT NULL,
24      nome_professor           VARCHAR(30)      NOT NULL,
25      titulacao_maxima         VARCHAR(10)      NOT NULL);
26
27  /* Criação da tabela Curso_Professor (Curso_Professor)
28  */
29  CREATE TABLE Curso_Professor (
30      cod_curso                 INTEGER      NOT NULL,
31      matricula_professor      INTEGER      NOT NULL);
32
33  /* Criação da tabela Turma (Turma)
34  */
35  CREATE TABLE Turma (
36      cod_curso                 INTEGER      NOT NULL,
37      ano_turma                 INTEGER      NOT NULL,
38      semestre_turma           INTEGER      NOT NULL,
39      desc_turma                VARCHAR(10)   NOT NULL,
40      matricula_professor      INTEGER      NOT NULL);
41
42  /* Criação da tabela Turma_Aluno (Turma_Aluno)
43  */
44  CREATE TABLE Turma_Aluno (
45      matricula_aluno           INTEGER      NOT NULL,
46      cod_curso                 INTEGER      NOT NULL,
47      ano_turma                 INTEGER      NOT NULL,
48      semestre_turma           INTEGER      NOT NULL,
49      desc_turma                VARCHAR(10)   NOT NULL);
50
51  /* Criação de chave primária PK_Aluno (PK_Aluno) da tabela Aluno (Alu
52  */
53  ALTER TABLE Aluno ADD CONSTRAINT PK_Aluno PRIMARY KEY(matricula_aluno
54
55  /* Criação de chave primária PK_Curso (PK_Curso) da tabela Curso (Cu
```





Você está em

DevMedia

```
63  /* Criação de chave primária PK_Curso_Professor (PK_Curso_Professor)
64  tabela Curso_Professor (Curso_Professor)
65  */
66  ALTER TABLE Curso_Professor ADD CONSTRAINT PK_Curso_Professor PRIMARY
67  KEY(cod_curso,matricula_professor);
68
69  /* Criação de chave primária PK_Turma (PK_Turma) da tabela Turma (Tu
70  */
71  ALTER TABLE Turma ADD CONSTRAINT PK_Turma PRIMARY KEY(cod_curso,ano_t
72  semestre_turma,desc_turma);
73
74  /* Criação de chave primária PK_Turma_Aluno (PK_Turma_Aluno) da tabe
75  Turma_Aluno (Turma_Aluno)
76  */
77  ALTER TABLE Turma_Aluno ADD CONSTRAINT PK_Turma_Aluno PRIMARY
78  KEY(matricula_aluno,cod_curso,ano_turma,semestre_turma,desc_turma);
79
80  /* Criação das chaves estrangeiras da tabela Aluno
81  */
82  /* Criação da chave estrangeira FK_Aluno_01 (FK_Aluno_01)
83  */
84  ALTER TABLE Aluno ADD CONSTRAINT FK_Aluno_01 FOREIGN KEY(cod_curso) R
85  Curso;
86
87  /* Criação da chave estrangeira FK_Curso_01 (FK_Curso_01)
88  */
89  ALTER TABLE Curso ADD CONSTRAINT FK_Curso_01 FOREIGN KEY(matricula_pr
90  REFERENCES Professor;
91
92  /* Criação das chaves estrangeiras da tabela Curso_Professor
93  */
94  /* Criação da chave estrangeira FK_Curso_Professor_01 (FK_Curso_Prof
95  */
```





Você está em

DevMedia

```
103
104  /* Criação da chave estrangeira FK_Turma_02 (FK_Turma_02)
105  */
106  ALTER TABLE Turma ADD CONSTRAINT FK_Turma_02 FOREIGN KEY(matricula_pr
107  REFERENCES Professor;
108
109  /* Criação da chave estrangeira FK_Turma_Aluno_01 (FK_Turma_Aluno_01
110  */
111  ALTER TABLE Turma_Aluno ADD CONSTRAINT FK_Turma_Aluno_01 FOREIGN
112  KEY(cod_curso,ano_turma,semestre_turma,desc_turma) REFERENCES Turma;
```

Listagem 1. Script de criação das tabelas

Após a criação do banco de dados e da inserção de dados nas tabelas, serão apresentados conceitos e exemplos que interferem no desempenho de uma consulta. O desempenho dos comandos pode ser afetado por vários fatores, por exemplo, a atualização das tabelas internas utilizadas pelo planejador para formular os planos de consulta, a existência de índices e, também, a má elaboração de junções (JOINS) entre tabelas. Todos estes fatores podem ser manipulados pelos próprios usuários.

A escolha correta do planejador por um bom plano de consulta, visando o que possui menor custo, maior eficiência e melhor tempo de resposta; estruturar o comando SQL de forma planejada e; analisar as propriedades dos dados são de fundamentais para ganho de performance.



Você está em

DevMedia

comando recebido. O plano de execução, gerado pelo SGBD, pode ser visualizado com a utilização do comando EXPLAIN. Este mostra o plano de consulta que o planejador do PostgreSQL gera para o comando fornecido, como foi feita a varredura pelas tabelas referenciadas, se houve ordenação e qual algoritmo de junção foi utilizado, no caso de junção de tabelas. A **Figura 3** apresenta um exemplo de utilização do Explain.

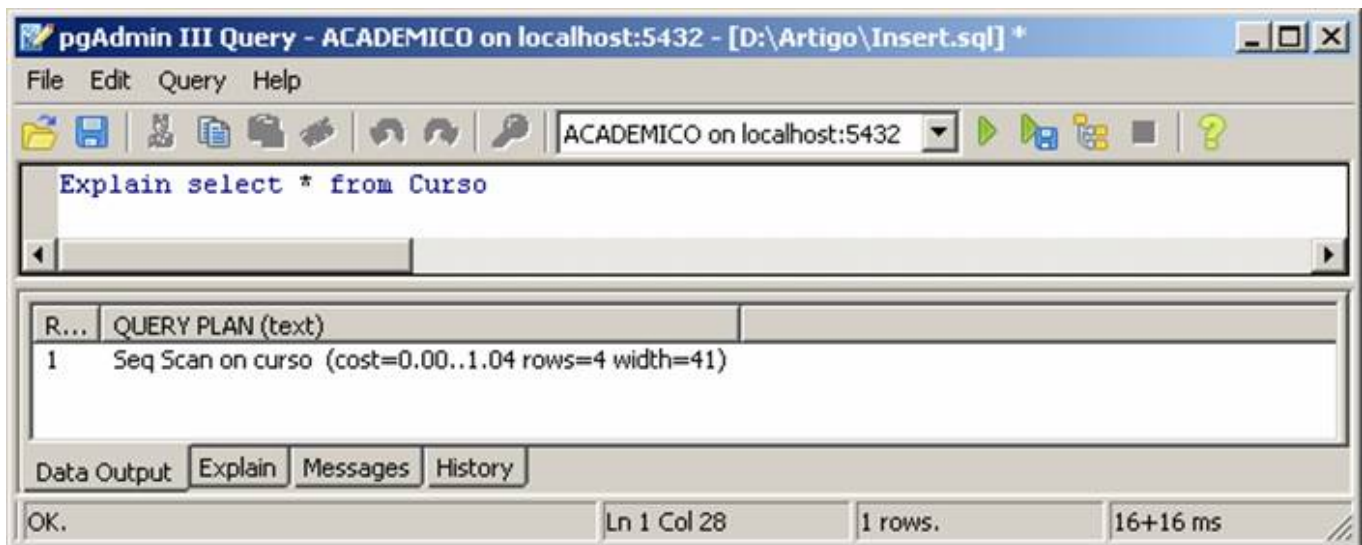


Figura 3. SELECT na tabela Curso utilizando o comando EXPLAIN

Traduzindo o resultado apresentado:

- Seq Scan on curso : indica que foi feita uma busca sequencial simples pela tabela curso;





Você está em

DevMedia

executar o comando. Para calcular este tempo é analisada a quantidade de páginas de disco acessadas somada à quantidade de linhas percorridas (o número total de linhas percorridas varia de acordo com fatores estipulados, denominados constantes de custo), número que também depende do tipo de varredura.

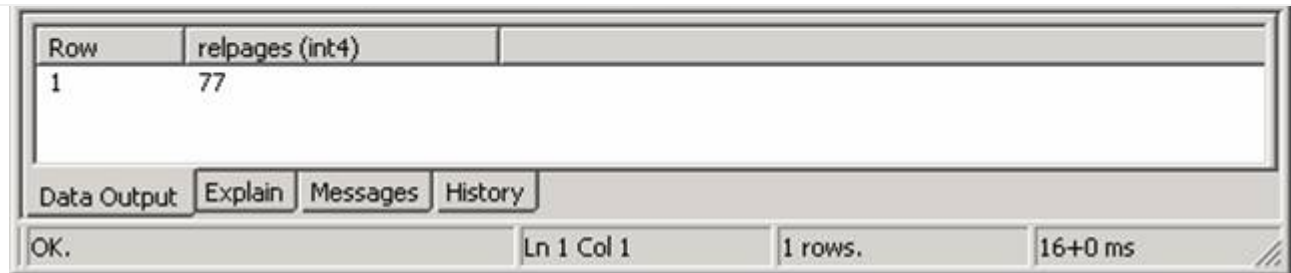
Há ainda o custo de partida que é o tempo gasto para começar a varrer a saída, ou seja, o tempo para fazer a primeira classificação, e o custo total, que estima o valor se todas as linhas fossem recuperadas. Geralmente, o tempo total terá maior importância, mas haverá casos em que o tempo inicial, ao invés do final, será enfatizado, como é o caso de uma consulta com a cláusula `EXIST`, já que a execução pára após ter obtido uma linha que atenda a esta condição.

Para conhecer o número de páginas em que a tabela está distribuída, utiliza-se a tabela `pg_class`, que é uma das tabelas de sistema do PostgreSQL. Ela possui diversas informações, como nome de todas as tabelas, número de páginas ocupadas pelas mesmas, índices e visões (ver **Figura 4**).



Você está em

DevMedia



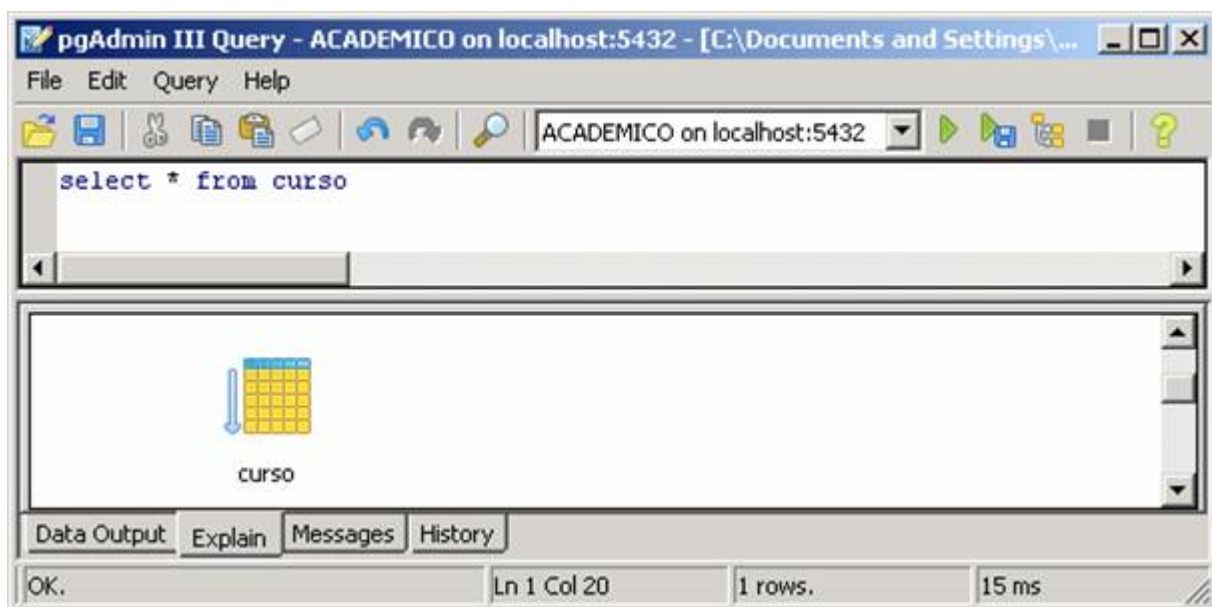
Row	relpages (int4)
1	77

Data Output Explain Messages History

OK. Ln 1 Col 1 1 rows. 16+0 ms

Figura 4. Número de páginas de disco ocupadas pela tabela Aluno

O PostgreSQL possui uma ferramenta gráfica que possibilita visualizar a execução do comando `EXPLAIN`. Para executá-la, é necessário criar a consulta e selecionar a aba `EXPLAIN`, como pode ser observado na **Figura 5**.

**Figura 5.** Representação gráfica do comando `EXPLAIN`

Você está em

DevMedia

contrário, ainda aumentam, pois além de ter que ler todas as linhas sequencialmente, gasta-se um tempo a mais na verificação da cláusula `WHERE`, como pode ser verificado no resultado da **Figura 3** (sem a cláusula `WHERE`) em comparação com a **Figura 6** (com a cláusula `WHERE`).

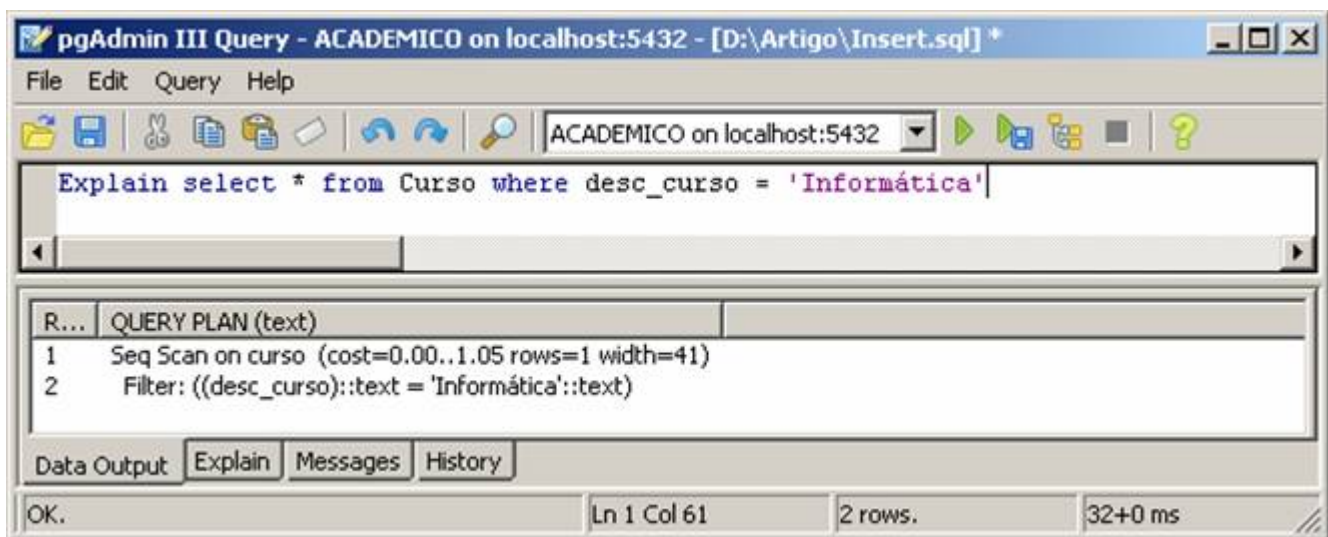


Figura 6. Exemplo de consulta com a utilização da cláusula `WHERE`

Observa-se que apesar de diminuir a estimativa de linhas a serem retornadas e manter a largura média, o custo de execução teve uma relativa mudança. Isso ocorreu porque a varredura ainda precisa percorrer todas as linhas da tabela e fazer o filtro pela cláusula `WHERE`, aumentando o custo, refletindo no tempo maior gasto pela CPU verificando as condições da cláusula `WHERE`. Esta diferença pode ser significativamente maior em função da quantidade de linhas a serem recuperadas e



Você está em

DevMedia

páginas estão todas as linhas que o PostgreSQL irá ler para retornar o resultado. Esta varredura pelas páginas pode ser feita de maneira sequencial, onde todas as linhas são lidas, ou com a utilização de índices.

Os índices são usados somente quando são bastante seletivos, ou seja, o resultado trazido deve possuir no máximo 5% do total de linhas armazenadas. Toda essa análise é feita internamente pelo SGBD para que se obtenha um ganho no custo estimado pela execução do comando.

Para melhor entendimento, será criado um índice na tabela Aluno (ver **Figura 7**).

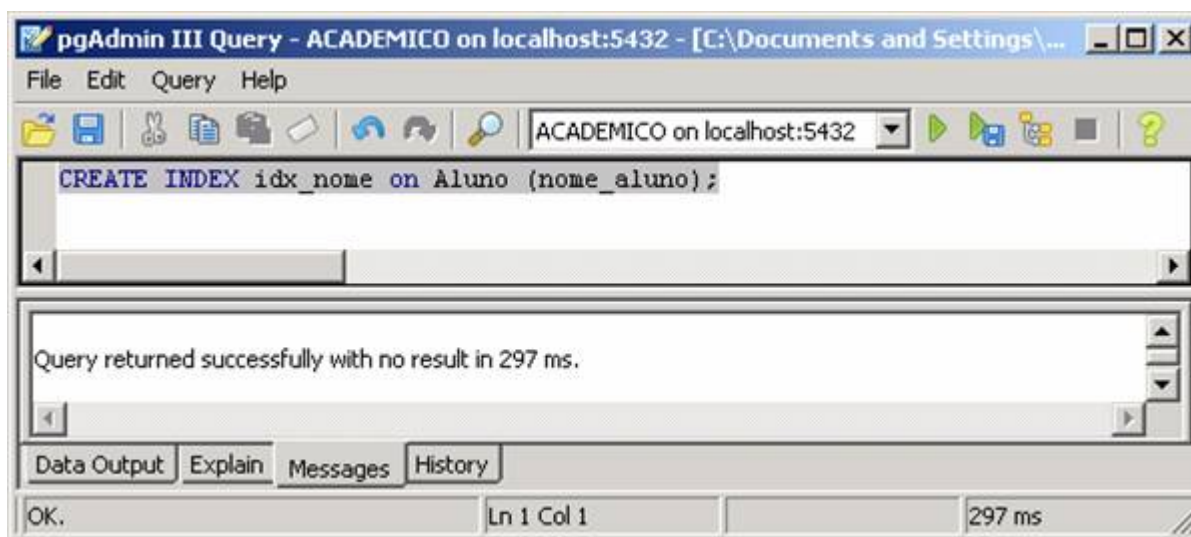


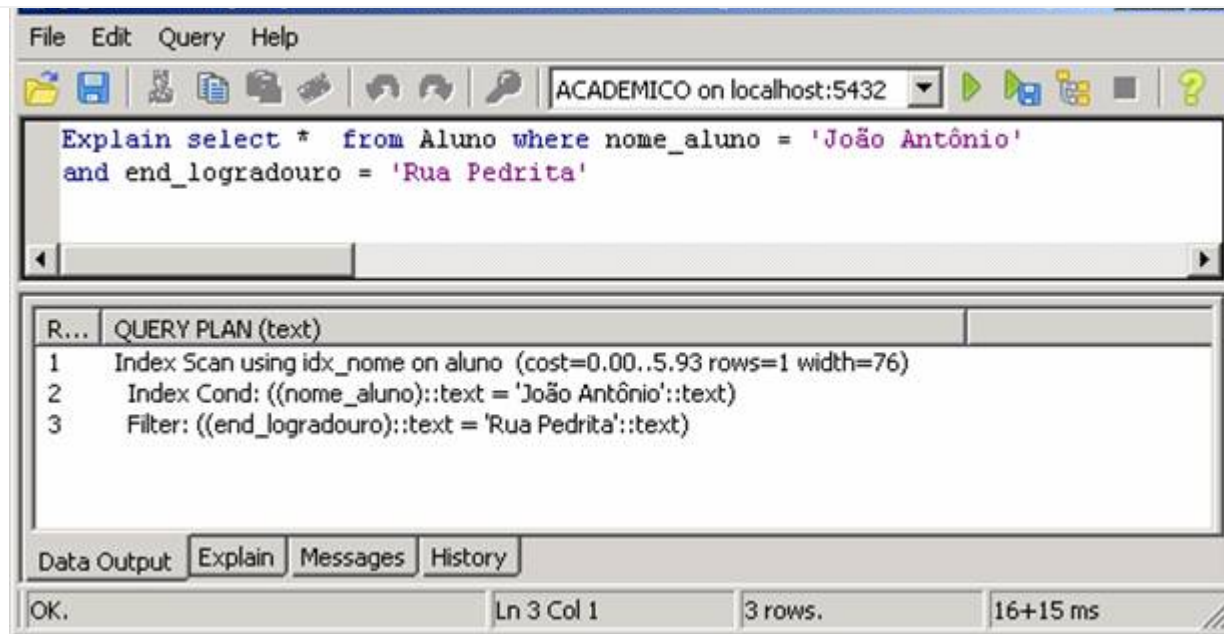
Figura 7. Criação do índice idx_nome na tabela Aluno

Será observado que após a criação do índice e da utilização de uma cláusula `WHERE`



Você está em

DevMedia

**Figura 8.** Varredura utilizando índice

Observa-se também que apesar de haver duas condições na cláusula WHERE, o campo NOME_ALUNO é utilizado como condição de índice e o outro, END_LOGRADOURO, como filtro.

O fato de uma consulta ser altamente seletiva, ou seja, fazer uso de índices quando estes existem, não significa que a mesma obterá um ganho de custo. Em muitos casos, ao tentar cercar todas alternativas, esquece-se de avaliar o número de linhas retornadas, resultando na não diminuição do custo.

Utilizando junções



Você está em

DevMedia

planejamento quanto para direcionar o planejador para um bom plano de execução.

A **Figura 9** apresenta uma junção entre duas tabelas. Neste exemplo, pode-se avaliar se existe algum ganho em fazer as operações de filtro (`c.matricula_professor = 100`) e junção (`c.matricula_professor = cp.matricula_professor`) numa mesma consulta ou fazê-las separadamente. A **Figura 10** mostra a mesma consulta em modo gráfico.

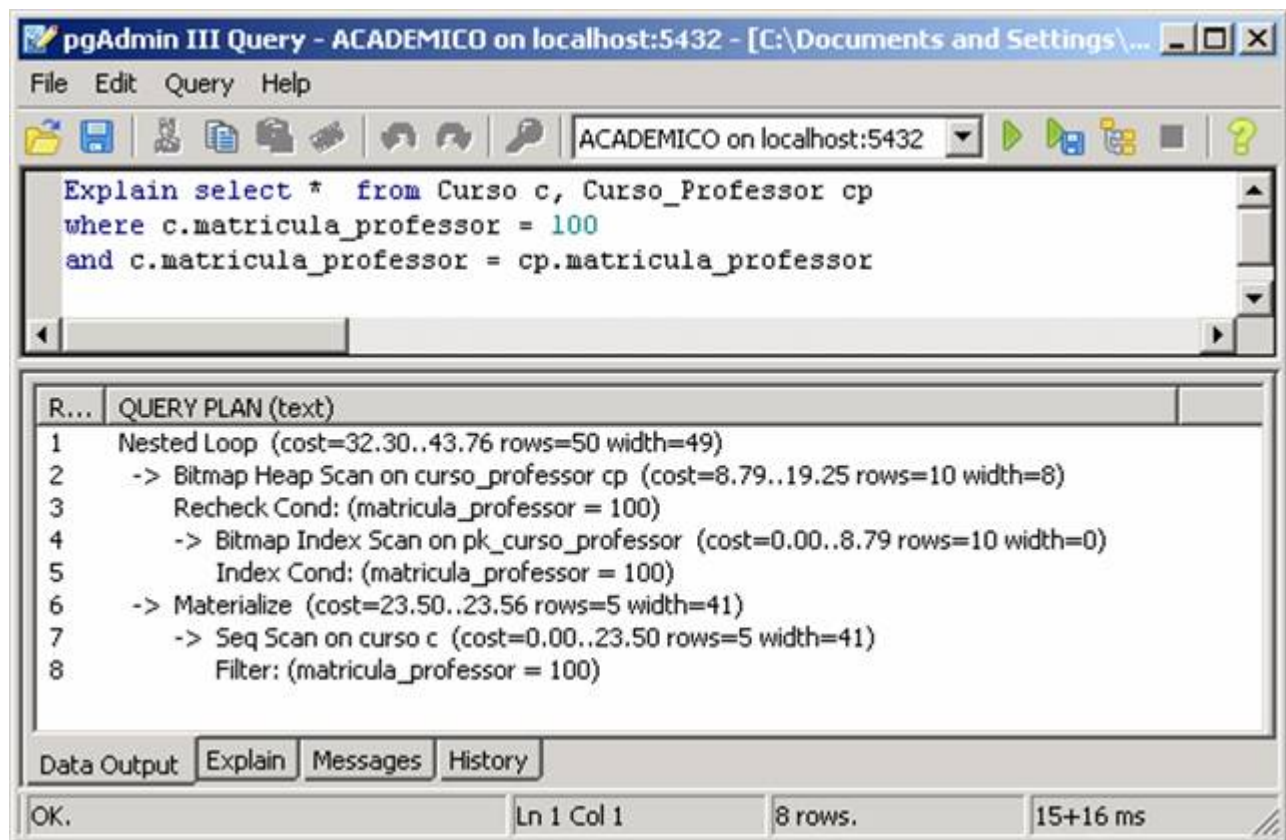


Figura 9. Exemplo do comando EXPLAIN em junções



Você está em

DevMedia

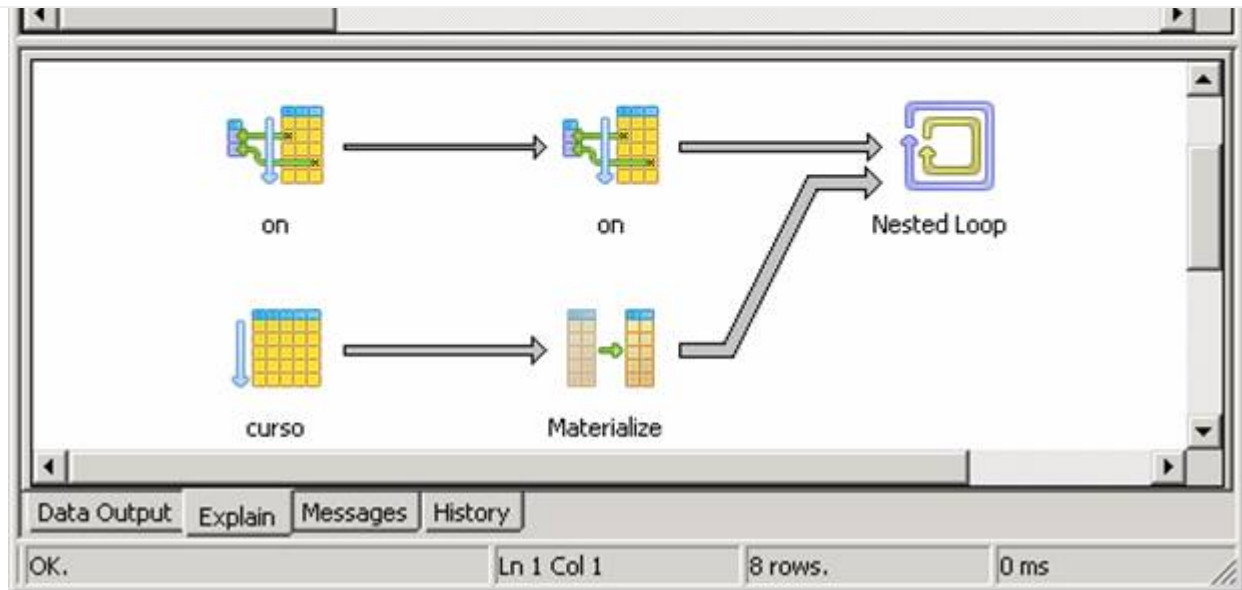


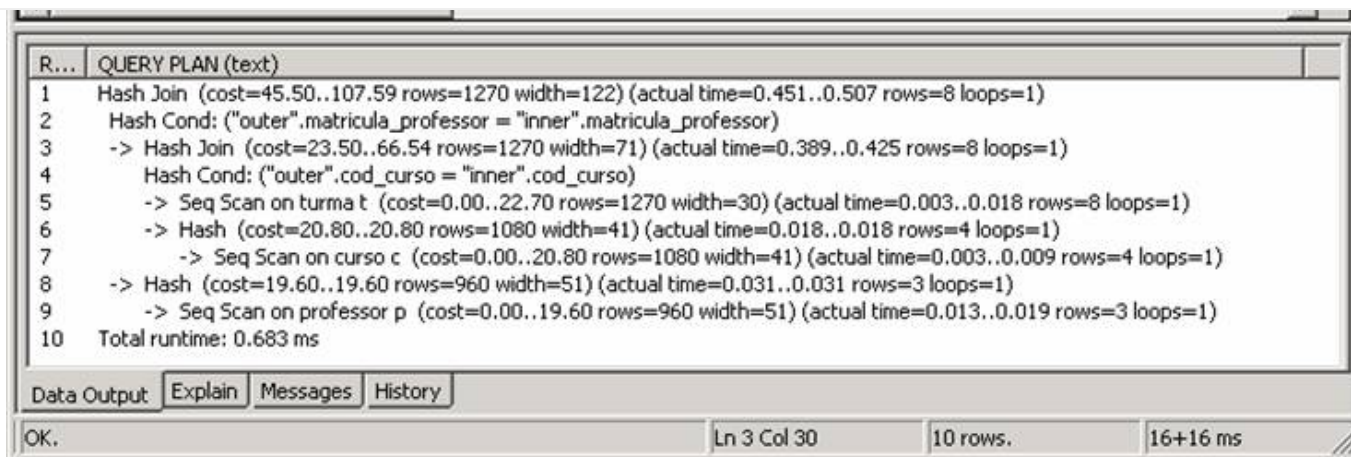
Figura 10. Exemplo do comando EXPLAIN em junções, em modo gráfico

Uma técnica de induzir o planejador para um bom plano é explicitando quais JOINS devem ser feitos primeiramente, em consultas envolvendo várias tabelas. Pois, se o planejador respeitar a ordem de JOINS, as consultas seguintes podem levar menos tempo para serem planejadas. Esta situação é muito interessante, embora possa não ser notado um ganho de desempenho quando utilizada com poucas tabelas ou tabelas com poucos registros, mas faz grande diferença quando se tem muitas tabelas envolvidas.

Observa-se nas **Figuras 11 e 12** que enquanto uma consulta comum pode levar 683 ms para ser executada, a mesma consulta melhor elaborada obtém um tempo total de execução de apenas 278 ms. Esta diferença de 405 ms é bastante considerável.



Você está em
DevMedia



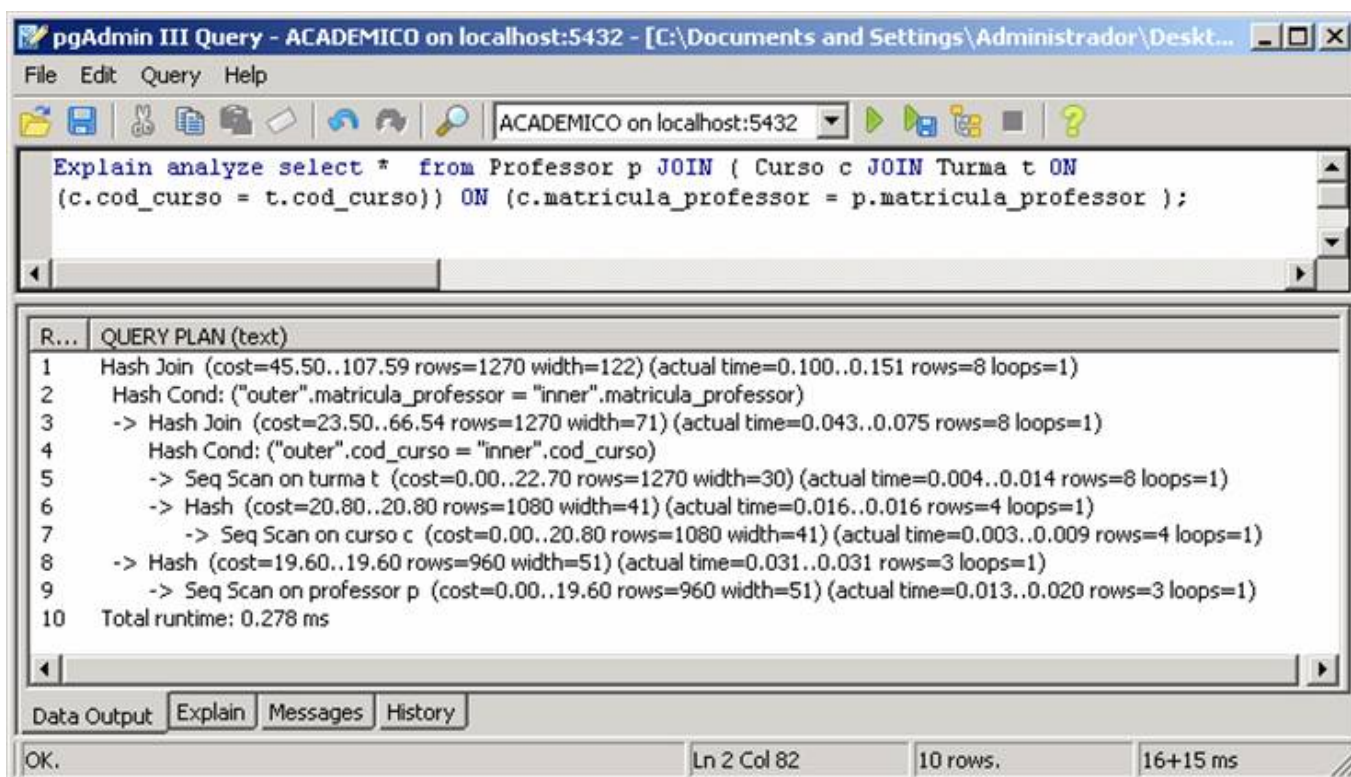
QUERY PLAN (text)

1	Hash Join (cost=45.50..107.59 rows=1270 width=122) (actual time=0.451..0.507 rows=8 loops=1)
2	Hash Cond: ("outer".matricula_professor = "inner".matricula_professor)
3	-> Hash Join (cost=23.50..66.54 rows=1270 width=71) (actual time=0.389..0.425 rows=8 loops=1)
4	Hash Cond: ("outer".cod_curso = "inner".cod_curso)
5	-> Seq Scan on turma t (cost=0.00..22.70 rows=1270 width=30) (actual time=0.003..0.018 rows=8 loops=1)
6	-> Hash (cost=20.80..20.80 rows=1080 width=41) (actual time=0.018..0.018 rows=4 loops=1)
7	-> Seq Scan on curso c (cost=0.00..20.80 rows=1080 width=41) (actual time=0.003..0.009 rows=4 loops=1)
8	-> Hash (cost=19.60..19.60 rows=960 width=51) (actual time=0.031..0.031 rows=3 loops=1)
9	-> Seq Scan on professor p (cost=0.00..19.60 rows=960 width=51) (actual time=0.013..0.019 rows=3 loops=1)
10	Total runtime: 0.683 ms

Data Output Explain Messages History

OK. Ln 3 Col 30 10 rows. 16+16 ms

Figura 11. Consulta com junções normalmente utilizadas



pgAdmin III Query - ACADEMICO on localhost:5432 - [C:\Documents and Settings\Administrador\Desktop...

File Edit Query Help

ACADEMICO on localhost:5432

Explain analyze select * from Professor p JOIN (Curso c JOIN Turma t ON (c.cod_curso = t.cod_curso)) ON (c.matricula_professor = p.matricula_professor);

QUERY PLAN (text)

1	Hash Join (cost=45.50..107.59 rows=1270 width=122) (actual time=0.100..0.151 rows=8 loops=1)
2	Hash Cond: ("outer".matricula_professor = "inner".matricula_professor)
3	-> Hash Join (cost=23.50..66.54 rows=1270 width=71) (actual time=0.043..0.075 rows=8 loops=1)
4	Hash Cond: ("outer".cod_curso = "inner".cod_curso)
5	-> Seq Scan on turma t (cost=0.00..22.70 rows=1270 width=30) (actual time=0.004..0.014 rows=8 loops=1)
6	-> Hash (cost=20.80..20.80 rows=1080 width=41) (actual time=0.016..0.016 rows=4 loops=1)
7	-> Seq Scan on curso c (cost=0.00..20.80 rows=1080 width=41) (actual time=0.003..0.009 rows=4 loops=1)
8	-> Hash (cost=19.60..19.60 rows=960 width=51) (actual time=0.031..0.031 rows=3 loops=1)
9	-> Seq Scan on professor p (cost=0.00..19.60 rows=960 width=51) (actual time=0.013..0.020 rows=3 loops=1)
10	Total runtime: 0.278 ms

Data Output Explain Messages History

OK. Ln 2 Col 82 10 rows. 16+15 ms

Figura 12. Consulta com junções elaboradas



Você está em

DevMedia

1 | EXPLAIN [ANALYZE] [VERBOSE]

O comando `EXPLAIN ANALYZE` tem uma grande diferença do comando `EXPLAIN` simples, que apenas apresenta qual plano será utilizado pelo planejador. Esta sintaxe realmente executa o comando `SQL`, como observado na **Figura 13**.

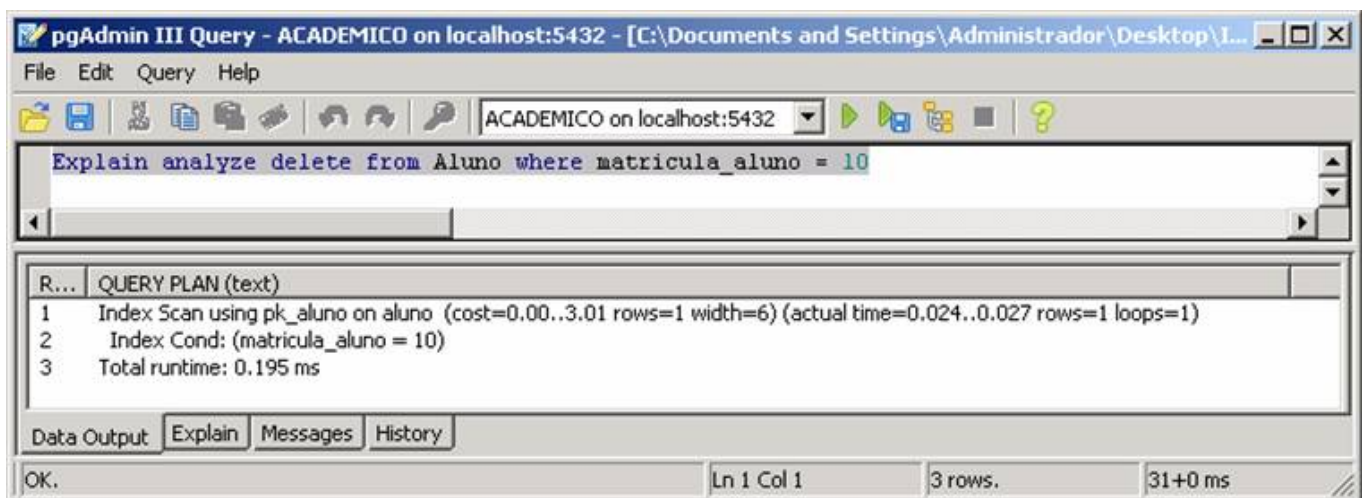


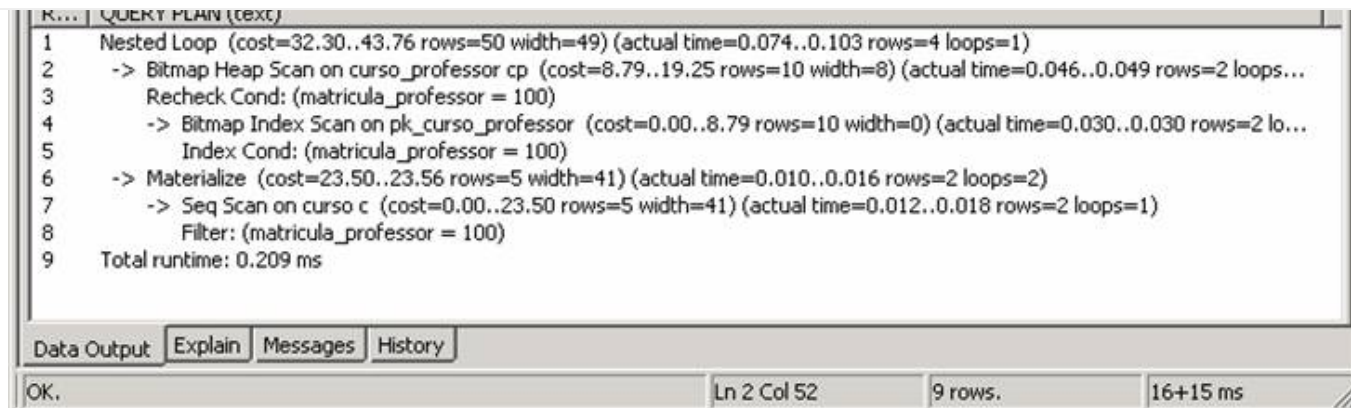
Figura 13. Execução do comando Explain Analyze para um comando Delete

Assim, o uso do `ANALYZE` possibilita uma verificação mais precisa dos resultados, pois além de apresentar os custos estimados, o tempo inicial e o tempo total que o comando `EXPLAIN` simples mostraria, apresenta o tempo real de cada etapa do plano.



Você está em

DevMedia



```
1  Nested Loop (cost=32.30..43.76 rows=50 width=49) (actual time=0.074..0.103 rows=4 loops=1)
2    -> Bitmap Heap Scan on curso_professor cp (cost=8.79..19.25 rows=10 width=8) (actual time=0.046..0.049 rows=2 loops=1)
3        Recheck Cond: (matricula_professor = 100)
4    -> Bitmap Index Scan on pk_curso_professor (cost=0.00..8.79 rows=10 width=0) (actual time=0.030..0.030 rows=2 loops=1)
5        Index Cond: (matricula_professor = 100)
6    -> Materialize (cost=23.50..23.56 rows=5 width=41) (actual time=0.010..0.016 rows=2 loops=2)
7        -> Seq Scan on curso c (cost=0.00..23.50 rows=5 width=41) (actual time=0.012..0.018 rows=2 loops=1)
8            Filter: (matricula_professor = 100)
9  Total runtime: 0.209 ms
```

Figura 14. Exemplo do comando Explain Analyze

Observa-se que os valores “actual time” são em milissegundos de tempo real, enquanto as estimativas de custo (cost) são expressas em unidades de busca em disco.

O tempo total de execução (Total runtime) apresentado pelo `ANALYZE` inclui os tempos de inicialização, de finalização e de processamento do resultado, não considerando o tempo de reescrita e nem análise da consulta.

Para os comandos `INSERT`, `UPDATE` e `DELETE`, o tempo de execução pode ser um pouco maior do que para um comando `SELECT`, uma vez que para executar um destes comandos, normalmente antes deve ser feita uma consulta para encontrar os dados a serem modificados (ver **Figura 13**).

O comando `ANALYZE` deve ser sempre executado para que as tabelas responsáveis por armazenar informações que serão utilizadas pelo planejador do PostgreSQL



Você está em

DevMedia

Por sua vez, o comando `EXPLAIN VERBOSE` mostra a representação interna completa da árvore do plano de consulta, em vez de apenas um resumo. Geralmente, esta opção é útil para finalidades especiais de depuração. A **Figura 15** apresenta a saída produzida utilizando a opção `VERBOSE`.

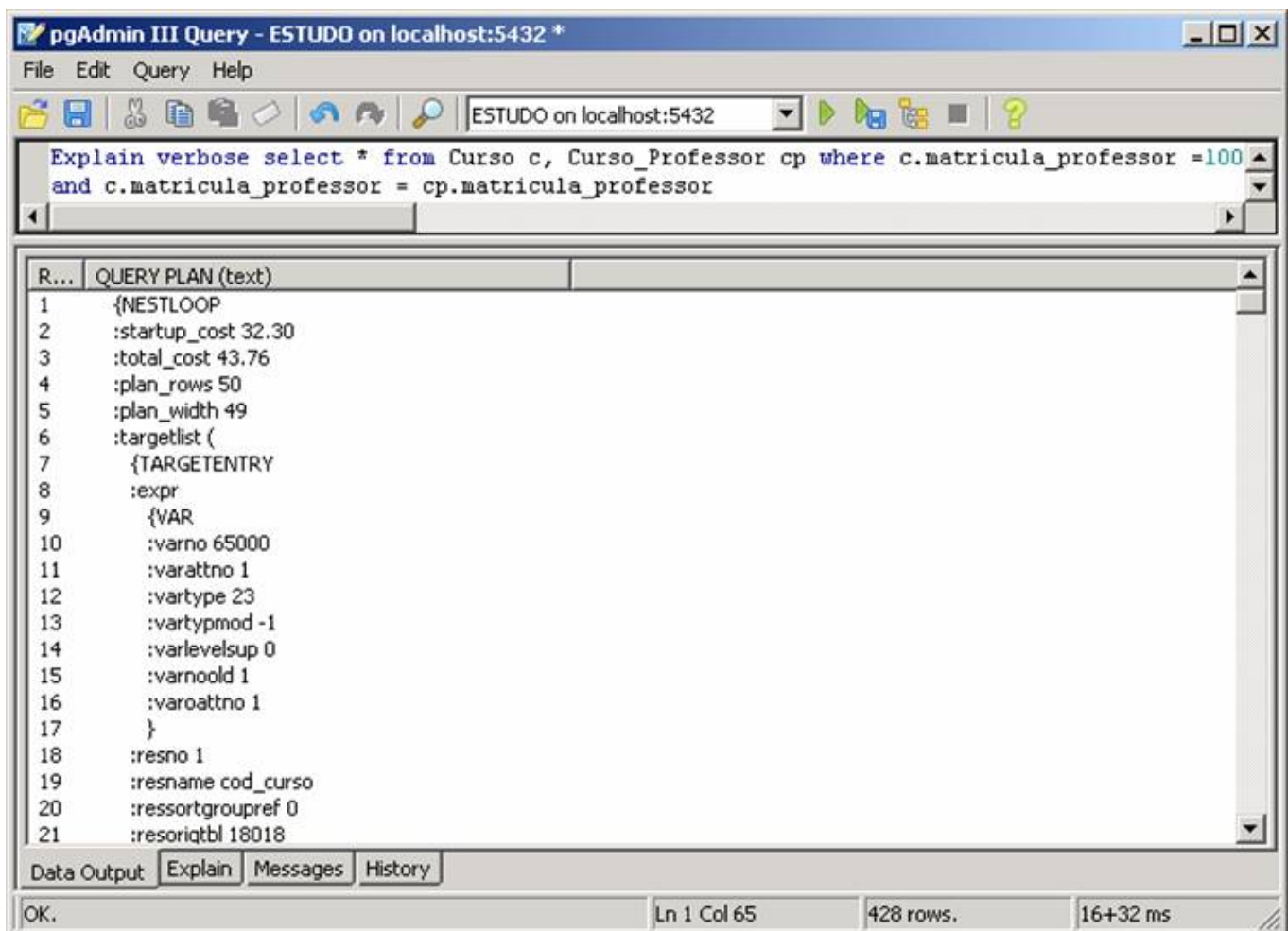


Figura 15. Exemplo do comando `EXPLAIN VERBOSE`

Você está em

DevMedia

considerar a estratégia que sairia vencedora. Isto é feito habilitando-se e desabilitando-se sinalizadores de cada tipo de plano.

Por exemplo, o PostgreSQL possui comandos como `SET ENABLE_SEQSCAN`, `SET ENABLE_INDEXSCAN` e `SET ENABLE_NESTLOOP`, os quais colocados em `OFF` desabilitam uma futura consulta do planejador utilizando consulta sequencial, consulta com índices ou com laço aninhado, respectivamente, como pode ser visto na **Figura 16**.

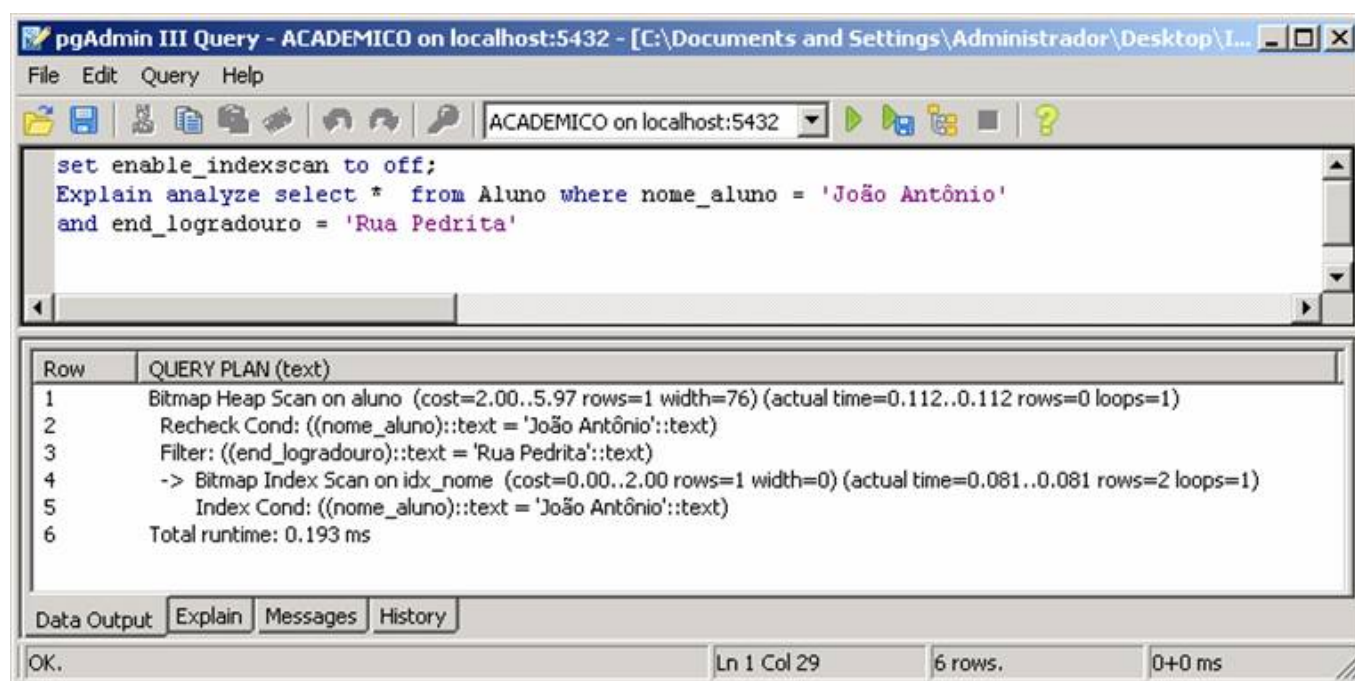


Figura 16. Desabilita varredura por índice

Neste caso, a consulta feita anteriormente na **Figura 8** que utilizava índices, utilizou-se de outros métodos para fazer a varredura



Você está em

DevMedia

anterior, para forçar o planejador a executar uma busca sequencial ao invés de fazer a leitura por índices. A consulta, neste caso, se torna mais eficiente uma vez que evitaria a leitura de páginas adicionais na procura por índices.

Finalizando, pode-se observar na **Tabela 1** um resumo das principais operações do PostgreSQL utilizadas em planos de consulta.

Operação	Descrição
Seq Scan	É a operação mais básica e representa uma leitura seqüencial da tabela. Quando ocorre uma Seq Scan, significa que a tabela foi lida da primeira à última linha.
Index Scan	Ocorre quando há uma leitura em alguma estrutura de índice a fim de se evitar uma leitura inteira da tabela, ou quando o planejador deseja aproveitar a ordem do índice para evitar uma operação de Sort.
Sort	Ocorre principalmente para satisfazer a cláusulas de ORDER BY, mas também é utilizada quando outra operação necessita de conjunto de dados ordenado para ser executada.
Unique	Essa operação é utilizada principalmente para satisfazer a cláusulas de DISTINCT, eliminando repetições.
Nested Loop	É utilizada para realizar JOIN entre duas tabelas. Ocorre principalmente em INNER JOINs, LEFT JOINs e UNIONs. Não processa completamente a tabela mais interna.
Hash e	As operações de Hash e Hash Join trabalham juntas para fazer JOINs



Você está em

DevMedia

Conclusão

Há muitas outras maneiras de se manipular consultas para encontrar um resultado satisfatório. Neste artigo foram enfatizados importantes recursos do PostgreSQL que podem ser muito úteis em grandes e pequenas aplicações.

Para melhor compreensão destes recursos, torna-se interessante conhecer as tabelas de sistema e os benefícios que podem ser retirados delas, além de conhecer outros comandos que podem proporcionar a criação de aplicações com melhor desempenho.

Tecnologias:

Banco de Dados

PostgreSQL

SQL



Marcado como lido



Anotar

Por **Lucas**

Em 2007



2





Você está em

DevMedia[Enviar dúvida](#)[Tecnologias](#)[Exercicios](#)[Cursos](#)[Artigos](#)[Revistas](#)
[Fale conosco](#)[Trabalhe conosco](#)[Assinatura para empresas](#)[Assine agora](#)[Hospedagem web por Porta 80 Web Hosting](#)

