NPRSQL - Normalização de Pesquisas e Respostas SQL
Otimizando Desempenho de Aplicações com Grande Volume de Dados

José Roberto Mininel
Centro Estadual de Educação Tecnológica Paula Souza (CEETEPS) – São Paulo – Brasil
jmininel@yahoo.com.br

Marília Macorin de Azevedo
Centro Estadual de Educação Tecnológica Paula Souza (CEETEPS) – São Paulo – Brasil
mmacorin@terra.com.br

Resumo - Com o alto volume de informações gerado pelas empresas e a necessidade de recuperação freqüente de tais informações têm ocorrido gargalos de processamento nos sistemas de informação. Este estudo apresenta uma abordagem mais performática e não trivial na arquitetura de recuperação de informações de banco de dados em aplicações que necessitam de alta performance e fazem uso de alto volume de dados, armazenados ou a ser recuperados.

Palavras-chave: Performance, Desempenho, Banco de Dados, Arquitetura, SQL.

1. Introdução:

1.1. Situação Problema

Em uma pesquisa SQL é comum que na resposta haja dados redundantes, resultantes de junções de tabelas, onde uma ou mais colunas tenham o mesmo conteúdo para todas as linhas de resposta ou várias delas.

Esta redundância de dados pode ser significativa e expõe a aplicação a tráfego de dados maior que o necessário. Quando esta redundância é pequena no todo, ou seja, não há volume significativo de dados, não causará problemas, mas quando o volume de informações pesquisados ou trafegados é grande, qualquer redundância que puder ser eliminada para cada linha de resposta, pode gerar ganhos significativos no tempo de processamento no Servidor, na Rede e na Estação Cliente.

1.1.1. Impacto no Servidor

a. Por não haver a necessidade de fazer a junção no SGBD não haverá armazenamento de dados em memória, dependendo do caso podendo até fazer gravação em disco (SWAP);
b. Eliminação do processamento para a junção dos dados;
c. Diminuição do tempo de transferência dos dados para a estação que requisitou os dados, minimizando o estrangulamento dos recursos do SGBD.

1.1.2. Impacto na Rede

a. Menor tráfego de dados;
b. Menor concorrência de processamentos.

1.1.3. Impacto no Client

a. Menor uso de memória para armazenar a resposta;

b. Menor uso ou eliminação SWAP em disco;
c. Maior processamento para junção das tabelas.

1.2. Resultados com Grandes Volumes

Várias situações fazem uso de pesquisas em volumes grandes de dados e/ou com resultados grandes. Citamos aqui alguns exemplos mais comuns, que de alguma forma o leitor já tenha tido algum conhecimento ou mesmo vivenciado:

a. Impressão de contas ou extratos (água, energia elétrica, telefones, conta corrente, fatura de cartão, etc.);
b. Malas diretas e propagandas personalizadas;
c. Confecção de arquivos de troca de dados entre sistemas e/ou empresas;
d. Datawarehouse para sistemas de BI.

Temos ainda situações em que o resultado é pequeno, mas com grande quantidade de concorrência.

a. Entrada de dados pela Internet;
b. Sistemas de CALL CENTER;
c. Serviços remotos como transmissão de declaração de Imposto de Renda pela internet.

1.3. Redundâncias no Resultado

Vamos exemplificar: consideremos uma pesquisa que retorne os logradouros e os seus respectivos estados:

```
SELECT NMLOGRADOURO, NMESTADO
FROM LOGRADOURO L JOIN ESTADO E ON L.CODESTADO = E.CODESTADO
```

Nome do logradouro	Descrição do Estado
BARAO DE PARANAPIACABA	SÃO PAULO
BENJAMIM CONSTANT	SÃO PAULO
CHA, DO	SÃO PAULO
DIREITA	SÃO PAULO
FILIPE DE OLIVEIRA	SÃO PAULO
JOSE BONIFACIO	SÃO PAULO
OUVIDOR PACHECO E SILVA	SÃO PAULO
PATRIARCA, DO	SÃO PAULO
QUINTINO BOCAIUVA	SÃO PAULO
SE, DA	SÃO PAULO
PRIMEIRO DE MARCO	RIO DE JANEIRO
QUINZE DE NOVEMBRO	RIO DE JANEIRO
SAO JOSE	RIO DE JANEIRO
TINOCO	RIO DE JANEIRO
TOCANTINS	RIO DE JANEIRO

Tabela 1: Tabela resposta com redundância de dados

Observamos nesta resposta que a coluna de “Estado” tem conteúdo redundante em várias linhas. Esta redundância é necessária porque precisamos saber o estado de cada logradouro individualmente.

2. Metodologia:

2.1. Solução Proposta

No exemplo acima observamos que temos uma junção de tabelas na pesquisa SQL, sendo que a tabela de logradouros tem 15 linhas e a tabela de estados tem 2 linhas. Se fizermos uma pesquisa para cada tabela, evitamos a redundância da coluna "Descrição do Estado". Vejamos como seria:

```
SELECT NOME, CODESTADO FROM LOGRADOURO ORDER BY CODESTADO
```

Nome do logradouro	Código do Estado
BARAO DE PARANAPIACABA	1
BENJAMIM CONSTANT	1
CHA, DO	1
DIREITA	1
FILIPE DE OLIVEIRA	1
JOSE BONIFACIO	1
OUVIDOR PACHECO E SILVA	1
PATRIARCA, DO	1
QUINTINO BOCAIUVA	1
SE, DA	1
PRIMEIRO DE MARCO	2
QUINZE DE NOVEMBRO	2
SAO JOSE	2
TINOCO	2
TOCANTINS	2

Tabela 2: Tabela de Logradouros

```
SELECT CODESTADO, NMESTADO FROM ESTADO ORDER BY CODESTADO
```

Código do Estado	Descrição do Estado
1	SÃO PAULO
2	RIO DE JANEIRO

Tabela 3: Resposta da tabela de Estados

E agora a aplicação assume a função de traduzir o "Código do Estado" em "Descrição do Estado". Para executar esta função pode-se utilizar diferentes formas. Algumas destas formas estão descritas logo a seguir.

2.2. Divisão da Resposta

Para fazer a divisão da resposta é necessário primeiro identificar na pesquisa a ser realizada, a tabela origem que terá seus dados replicados na resposta, em função da junção (join) realizada com outra tabela. No exemplo acima, esta tabela está exemplificada pela tabela ESTADO. Esta tabela é retirada da pesquisa SQL original e é feita a sua recuperação em outra pesquisa SQL, procurando não realizar nenhum vínculo com a pesquisa principal, para não acrescentar uma nova fase de processamento inexistente na pesquisa original, o que pode acarretar um tempo de processamento maior.

Forma inapropriada (Pesquisa com relacionamento na pesquisa principal)

```
SELECT CODESTADO, NMESTADO FROM ESTADO E WHERE EXISTS( SELECT *
FROM LOGRADOURO L WHERE L.CODESTADO = E.CODESTADO)
```

Formas apropriadas (Pesquisa total da tabela)

```
SELECT CODESTADO, NMESTADO FROM ESTADO
```

Ou se houver necessidade de restringir o retorno dos dados:

```
SELECT CODESTADO, NMESTADO FROM ESTADO WHERE CODESTADO IN ( 1, 2)
```

2.3. Resposta no Servidor

Como pode ser observada, a solução proposta tem duas pesquisas SQL que resultarão em duas respostas separadas para a aplicação. As pesquisas poderão ser disparadas por um mesmo objeto de acesso ao banco de dados, mas o mesmo não acontece para as respostas que estarão sendo utilizadas concomitantemente, portanto precisam ser armazenadas em objetos distintos.

Vale ressaltar que a pesquisa dos dados da tabela auxiliar não deve ser realizada linha a linha, dentro do processamento, pois isto gera uma alta perda de desempenho.

2.4. Junção na Estação Cliente

A junção das respostas ocorrerá na estação cliente, através do processamento da aplicação. O objeto com a resposta principal será percorrido e para cada linha deste deverá ser identificada o seu correspondente no elemento auxiliar, provendo os dados complementares à resposta principal. Para prover este mecanismo de junção vamos apresentar três diferentes opções, que devem ser aplicadas em situações distintas de acordo com a quantidade de linhas na resposta auxiliar.

2.5. Uso de Vetor

O uso de vetor consiste em utilizar um vetor de n elementos texto, onde n é o código máximo da resposta e cada elemento texto armazena a sua respectiva descrição.

```
Vetor[1] := 'SÃO PAULO'
Vetor[2] := 'RIO DE JANEIRO'
```

Assim, para recuperar a descrição, basta acessar o vetor de índice igual ao código desejado. Abaixo segue o algoritmo descrevendo o funcionamento do método:

1) Recuperar dados da tabela auxiliar (Estado);
2) Para cada código da tabela auxiliar alimentar o conteúdo do respectivo índice do vetor com a descrição da tabela auxiliar;
3) Recuperar dados da tabela principal (Logradouro);
4) Fazer o processamento desejado, percorrendo a tabela principal, recuperando a descrição do Estado no vetor através do índice correspondente ao código do Estado do Logradouro.

Esta solução apresenta um desempenho excepcional, por outro lado é necessário reservar todas as posições do vetor de 1 ao código máximo o que pode representar uso excessivo de memória. Por exemplo, se o código do 'RIO DE JANEIRO' fosse 20, seria necessário criar um vetor de 20 posições, onde estariam sendo utilizadas apenas as posições 1 e 20, e as posições de 2 a 19 estariam com espaço reservado na memória sem utilização efetiva.

2.6. Tabela em Memória

Outra opção é o uso na aplicação de um objeto, onde os dados auxiliares possam ser armazenados e recuperados. Este objeto pode ser do mesmo tipo utilizado para fazer a pesquisa SQL. A pesquisa deve ser ordenada pelo atributo código e no processamento da tabela principal deverá ser feita uma busca ordenada para um bom desempenho. Veja abaixo o algoritmo descrevendo o funcionamento deste método:

1) Recuperar dados da tabela auxiliar ordenado pelo atributo código (Estado);
2) Recuperar dados da tabela principal (Logradouro);
3) Processar os dados, percorrendo a tabela principal, recuperando a descrição do Estado na tabela auxiliar através de busca ordenada pelo código do Estado.

Esta solução tem um bom desempenho e utiliza o espaço de memória estritamente necessário para armazenar os dados, independente do código ser seqüencial ou alternado. Outra vantagem desta solução é que o código (atributo identificador) não precisa ser numérico. Mas o desempenho é inferior ao uso da solução por vetor, pois é necessário fazer uma busca ordenada, enquanto que por vetor a localização é imediata.

2.7. Tabela Ordenada em Memória

Esta solução diferencia-se da solução anterior somente na busca da tabela auxiliar, que nesta solução faz com que a mudança de linha da tabela auxiliar não seja por busca ordenada e sim por um avanço progressivo na mesma de forma sincronizada com a tabela principal que precisará ser recuperada também ordenada pelo mesmo atributo.

Esta solução tem praticamente as mesmas características da solução anterior, mas com uma diferença básica: a resposta da tabela auxiliar somente terá navegação em um sentido. Em tabelas pequenas o resultado não é vantajoso, mas quando a tabela auxiliar está com muitas linhas, por não exigir busca indexada e não percorrer várias vezes registros desnecessariamente, passa a ser uma solução com resultados melhores.

É necessário ponderar em cada situação qual solução é mais apropriada. Deve-se considerar a possibilidade de mesclar soluções quando existir mais de uma tabela auxiliar que passará por “NPRSQL - Normalização de Pesquisa e Respostas SQL”.

3. Resultados:

3.1. Cálculo dos Dados Reduzidos

Vamos considerar a pesquisa apresentada anteriormente: Logradouro e Estado. Neste caso temos a seguinte estrutura de dados:

Atributo	Tipo	Tamanho em bytes
Nome do Logradouro	Caractere	40
Código do Estado	Inteiro	1
Nome do Estado	Caractere	40

Tabela 4: Atributos da resposta solicitada

E que o volume de dados seja de 10 logradouros em São Paulo e 5 no Rio de Janeiro. Para a solução trivial em que é feita a junção das tabelas teremos a seguinte quantidade de bytes na resposta da pesquisa:

Bytes por linha	40 + 40 =	80
Quantidade de linhas		15
Total de bytes da pesquisa	80 * 15 =	**1200**

Tabela 5: Total de bytes na Pesquisa SQL Trivial

Já na solução-proposta temos duas respostas de dados a serem calculadas:

Bytes por linha da pesquisa de logradouros	40 + 1 =	41
Quantidade de linhas		15
Total de bytes da pesquisa de logradouros	41 * 15 =	**615**
Bytes por linha da pesquisa de estados	1 + 40 =	41
Quantidade de linhas		2
Total de bytes da pesquisa de estados	41 * 2 =	**82**
Total de bytes das pesquisas	615 + 82 =	**697**

Tabela 6: Total de bytes na Pesquisa SQL Proposta

Podemos agora então calcular a redução dos dados:

Bytes economizados	1200 – 697 =	503
Economia percentual sobre o resultado trivial	503 / 1200 =	**42%**
Proporção de 4 bytes economizados para cada 10 pesquisados		
Economia percentual sobre o resultado proposto	503 / 697 =	**72%**
Proporção de 7 bytes economizados para cada 10 pesquisados		

Tabela 7: Redução de dados

3.2. Redução do Tempo

Estimar o tempo que será reduzido envolve ponderar várias características do ambiente e da aplicação, entre elas vamos citar as mais críticas:

1) Tempo de resposta do SGBD, rede e estação cliente;
2) Capacidade de processamento da estação cliente;
3) Concorrência de acesso ao SGBD e rede;
4) Distribuição dos dados nas tabelas;
5) Modelagem dos dados e índices em uso.

Compor uma fórmula que considere todos estes fatores torna-se uma tarefa complexa e pouco prática, pois temos variação de alguns fatores como por exemplo a concorrência de acesso.

Mas como o que nos interessa é se de fato ocorre uma economia significativa de tempo, podemos fazer um teste comparativo em uma ambiente isolado, para obtermos uma noção do ganho do desempenho.

3.3. Medições de Tempo

Apresentamos agora algumas medições que foram efetuadas em três situações distintas, utilizando dois PC desktop Pentium 3 Core Duo (M1 e M2) e um servidor (M3). O primeiro teste de cada execução foi descartado, tomando sempre o tempo do segundo teste. As tabelas e atributos dos testes são as mesmas utilizadas nos exemplos acima, modificando somente a quantidade de linhas e conteúdo dos atributos.

Aplicação e SGBD em M1:

Linhas	Solução Trivial	Solução Vetor	Solução Tabela	Tab. Ordenada	Ganho
10.000	1,66 segundos	1,56 segundos	1,66 segundos	1,64 segundos	6 %
20.000	6,02 segundos	5,63 segundos	5,78 segundos	5,78 segundos	3 %
50.000	45,25 segundos	43,47 segundos	41,72 segundos	43,66 segundos	8 %

Tabela 8: Medições de tempo de processamento local

Aplicação em M1 e o SGDB em M2 com acesso por rede local de 10 Mbps:

Linhas	Solução Trivial	Solução Vetor	Solução Tabela	Tab. Ordenada	Ganho
10.000	2,56 segundos	2,06 segundos	2,19 segundos	2,20 segundos	20 %
20.000	8,70 segundos	6,61 segundos	6,67 segundos	7,16 segundos	24 %
50.000	42,61 segundos	35,39 segundos	38,67 segundos	38,99 segundos	17 %

Tabela 9: Medições de tempo em rede Local

Aplicação em M1 e o SGDB em M3 com acesso internet speedy de 250 Kbps:

Linhas	Solução Trivial	Solução Vetor	Solução Tabela	Tab. Ordenada	Ganho
10.000	30,3 segundos	16,5 segundos	17,0 segundos	16,7 segundos	45 %
20.000	62,8 segundos	35,3 segundos	36,5 segundos	35,7 segundos	44 %
50.000	180,6 segundos	105,9 segundos	105,5 segundos	106,0 segundos	42%

Tabela 10: Medições de tempo em rede Internet

4. Discussão e Considerações

Nos últimos anos a necessidade de melhorar o desempenho de sistemas de informações tem ganhado destaque especial em vista do aumento do volume das informações armazenadas. Por vezes, este aumento tem sido exponencial.

Estas aplicações geralmente utilizam ambientes de processamento CLIENT/SERVER ou multicamada, e como pôde ser visto, apresentam alto índice de redundância na transferência de dados entre aplicação e SGBD, contribuindo para perda de desempenho das aplicações e maior tráfego nas redes.

Neste sentido este estudo apresentou um novo conceito de recuperação de dados minimizando esta redundância, que embora exija um pouco mais de análise da arquitetura e codificação da aplicação, tem a recompensa de perfazer ganhos significativos no desempenho e minimizar tráfego dos dados em rede de aplicações que utilizam grandes volumes de dados.

É importante salientar ainda que é possível calcular previamente a redução dos dados trafegados, com relativa facilidade e a partir desta informação analisar e tomar a decisão de utilizar um modelo mais eficiente de recuperação de dados: **NPRSQL – Normalização de Pesquisas e Respostas SQL.**

5. Agradecimentos

A minha orientadora Marília Macorin de Azevedo pelo apoio e incentivo na realização deste trabalho.

6. Contato

José Roberto Mininel
Coordenador e Arquiteto de Desenvolvimento de Sistemas
(11) 9264-9289 (cel) (11) 4238-2937 (res)
jmininel@uol.com.br