MAC 5701 - Tópicos em Ciência da Computação

O Uso de Visões Materializadas em Data Warehouses

Isabel Cristina Italiano
Orientador: Prof. João Eduardo Ferreira

Dezembro de 2000

Resumo. Um data warehouse pode ser definido como um conjunto de visões materializadas, obtidas a partir de uma ou mais fontes de dados, com o propósito de implementar, de forma eficiente, as consultas dos sistemas de suporte à decisão. O uso de visões materializadas nos data warehouses tem como objetivo minimizar o tempo de resposta destas consultas e os custos de manutenção das visões. Neste trabalho serão descritas algumas das formas de utilização das visões materializadas nos data warehouses, destacando alguns aspectos importantes como a seleção das visões mais adequadas, dado um conjunto de pesquisas comuns e as características dos processos de manutenção das visões de forma dinâmica e incremental.

1 Introdução

Warehousing é uma técnica utilizada para recuperação e integração de dados a partir de fontes distribuídas, autônomas e, possivelmente, heterogêneas [ZGM+94].

Estes dados são armazenados em um um grande depósito chamado de *Data Warehouse*[1]. Um data warehouse sumariza os dados que são organizados em dimensões, disponibilizando-os para consultas e análises através de aplicações OLAP (On-Line Analytical Processing) e sistemas de suporte à decisão [GM96].

Os data warehouses vêm sendo muito utilizados pelas empresas do mundo todo, já que proporcionam um alicerce sólido de integração de dados corporativos e históricos para a realização de análises gerenciais. Sua construção e implementação é feita de uma maneira passo a passo, organizando e armazenando os dados sob uma perspectiva de longo prazo. Assim, partindo-se dos dados históricos básicos, pode-se realizar análises de tendências [WIRH97].

[1] O termo *Data Warehouse*, que pode ser traduzido para *armazém de dados*, é sempre encontrado na literatura escrita em português, na sua forma original, em inglês.

Por sua característica básica, que é a integração de dados provenientes de várias fontes diferentes, a etapa mais complexa na implementação de um data warehouse é o processo de carga. Neste processo, os dados distribuídos pelos vários ambientes operacionais (bases de dados de produção, que contêm os dados utilizados pelos vários sistemas transacionais de uma empresa) devem ser selecionados, trabalhados com o objetivo de padronização e limpeza, transferidos para o novo ambiente e finalmente carregados, sempre atendendo ao padrão da modelagem utilizada para o data warehouse. Este processo é feito periodicamente, sendo que sua freqüência depende de vários fatores relacionados ao modelo de negócios utilizado pela empresa e, normalmente, não é menor que 24 horas. Desta forma, podemos dizer que os dados armazenados no data warehouse são, para todos os propósitos práticos, uma longa série de fotografias, tiradas ao longo do tempo. Uma vez que os dados são armazenados no data warehouse, eles não mais sofrem atualizações, sendo, portanto, um ambiente apenas de carga e acesso.

Após sua criação e primeira carga, o data warehouse passa a sofrer cargas incrementais que devem refletir o ambiente operacional ao longo do tempo tornando-o uma imensa base de dados para os sistemas de apoio à decisão.

A abordagem mais utilizada pelas empresas é a de manter o data warehouse disponível para leituras, através de consultas, durante o dia, sendo que durante a noite, permanece indisponível, para que seja realizado o processo de manutenção.

Torna-se claro, portanto, que o data warehouse, na grande maioria dos casos, tem características de um ambiente estático, que só reflete as alterações ocorridas no ambiente operacional após períodos pré-definidos. Este fato em si não é tão grave, já que em várias áreas de negócio, as análises são feitas baseadas em resumos mensais, por exemplo. Um dos fatores críticos é a alta complexidade do processo de carga que se transforma em um ponto muito suscetível à introdução de erros, que podem levar ao colapso de todo o processo de tomada de decisão. Outro fator crítico da abordagem citada é que as empresas, ao passarem por processos de globalização, necessitam do data warehouse disponível o maior tempo possível. Com isto, torna-se necessária a implementação de alternativas que reduzam, ao máximo, o tempo necessário para a manutenção ou que permitam que o processo de manutenção seja executado de forma concorrente às consultas dos usuários, garantindo, porém, a consistência dos acessos ao data warehouse [DQ97].

Como o data warehouse sofre periodicamente novas cargas, aumentando constantemente seu tamanho, surge mais um grande problema em sua utilização, que é a dificuldade em responder às consultas dos usuários de forma rápida e eficiente. As otimizações nos componentes que envolvem este processo, visando agilizar as respostas às consultas, vêm sendo projetadas em vários níveis:

1. mudanças nas estruturas físicas que compõem a base de dados;
2. ferramentas de análise mais eficientes e
3. melhorias no modelo de dados implementado no data warehouse;

Os itens 1 e 2 acima não fazem parte do escopo deste estudo, sendo que apenas o item 3 será considerado. Todos os problemas citados anteriormente, ou seja, a alta complexidade do processo de carga, a prolongada indisponibilidade do data warehouse, aliadas às suas características estáticas e à necessidade de melhorar o modelo de dados implementado, são

a motivação para este trabalho, que busca analisar algumas das alternativas disponíveis para uma implementação mais eficiente e dinâmica do data warehouse.

Uma outra forma de implementar um data warehouse é defini-lo como um conjunto de *visões materializadas*[2] que integram os dados a partir de múltiplas fontes heterogêneas, e eventualmente distribuídas, de informação [DEB95].

Uma visão é uma relação derivada, definida em termos de relações base, que é computada todas as vezes em que uma referência a ela é feita. Uma visão é dita materializada quando ela é realmente armazenada na base de dados em vez de ser computada a partir das relações base em resposta a consultas [QGMW97]. Uma visão materializada pode ser vista como um cache – uma cópia dos dados que pode ser acessada rapidamente

Além de definir o próprio data warehouse como um conjunto de visões materializadas baseadas nos dados dos ambientes operacionais [QGMW97, HG97], estas visões também podem ser utilizadas com o objetivo de otimizar consultas complexas [DQ97], sendo que, para isto, devemos definir um conjunto compartilhado de visões, criteriosamente escolhidas a partir de uma análise das consultas mais freqüentes executadas no data warehouse. Estas visões, ao serem materializadas, agilizariam o acesso aos dados necessários para a realização das consultas ao data warehouse[YKL96].

2 A utilização de visões materializadas nos data warehouses

Como já citado, uma visão materializada é uma consulta cujo resultado já está computado e armazenado na base de dados [DQ97]. As consultas que puderem utilizar as visões já armazenadas podem ser executadas de forma muito mais rápida, sendo que, para consultas complexas envolvendo grandes volumes de dados, esta alternativa favorece dramaticamente os resultados: de horas ou dias para segundos ou minutos. De fato, as visões materializadas são vistas como uma das principais opções para o controle do desempenho de um data warehouse [RK96].

A desvantagem das visões materializadas é que as alterações feitas aos dados base, a partir dos quais uma visão materializada é definida, torna a visão desatualizada. Para que a visão possa estar novamente sincronizada aos dados base, será necessário recriar a visão a partir dos dados de origem ou então, atualizá-la incrementalmente [DQ97].

Outro aspecto importante é que estas atualizações podem ser executadas imediatamente, tão logo a alteração é recebida, ou podem ser proteladas, de tal forma que todas as alterações ocorridas nos dados base serão reunidas e aplicadas em processos batch, que muitas vezes podem ser demorados.

O uso de visões materializadas nos data warehouses, conforme descrito nos parágrafos acima, requer, portanto, a análise de alguns tópicos relacionados, como:

[2] O termo *visões materializadas* foi traduzido do inglês *materialized views*.

1. a seleção das visões mais adequadas levando-se em conta uma análise dos custos de manutenção e os benefícios de cada visão e os algoritmos disponíveis;
2. aspectos da materialização e utilização das visões para responder às consultas;
3. os mecanismos que permitem a propagação correta quando da atualização das fontes de dados base, para vários tipos de visões, preferencialmente de forma online;
4. a manutenção das visões de forma dinâmica e incremental;
5. a criação de algoritmos que permitam a manutenção das visões de maneira autônoma, utilizando o conceito de *self maintainable views*.

A seguir descreveremos as visões materializadas e alguns aspectos referentes à sua definição, seleção, utilização e manutenção.

2.1 Selecionando as visões a serem materializadas

[GM95] define uma visão como uma relação derivada, em termos das relações base armazenadas. Uma visão é, portanto, uma função que parte de um conjunto de tabelas base para uma tabela derivada, sendo que esta função é recalculada todas as vezes em que é referenciada.

Uma visão pode ser materializada, através do armazenamento das tuplas da visão na base de dados. Com esta materialização, é possível criar índices que tornarão o acesso a estas visões materializadas muito mais rápido que o recálculo, que é executado todas as vezes em que as visões são referenciadas.

Conforme descrito em [YKL96], um data warehouse, devido aos diferentes tipos de análises, pode conter múltiplas visões. Quando estas visões estão relacionadas umas com as outras, isto é, quando estão definidas sobre partes sobrepostas dos dados base, então pode ser mais eficiente materializar apenas certas visões compartilhadas, ou porções dos dados base, em vez de materializar todas as visões. Assim, surge o primeiro aspecto a ser analisado, que se refere à escolha das visões que serão materializadas. Esta escolha se baseia na determinação de um conjunto de visões, de uso compartilhado no data warehouse, de modo a combinar bom desempenho com baixo custo de manutenção. O objetivo é selecionar um conjunto apropriado de visões que minimize o tempo total de reposta das consultas e o custo de manutenção das visões selecionadas, dado um certo limite de recursos, como por exemplo, tempo de materialização, espaço para armazenamento etc.

Vários trabalhos tratam deste tema, como por exemplo, [HRU96] que apresenta e analisa algoritmos para a seleção das visões em um caso especial de “cubos de dados”. [GHRU96] estende os resultados para a seleção de visões e índices em cubos de dados, porém ambos ignoram os custos de manutenção das visões. Uma estrutura teórica é definida em [HG97] para o problema geral de seleção de visões, apresentando uma heurística para alguns casos especiais importantes de problemas que ocorrem na prática. [YKL96] define aspectos de uma metodologia para desenho das visões materializadas, por exemplo, como selecionar um conjunto de resultados intermediários das consultas a serem materializados de forma que o custo total seja mínimo. Apresenta também um algoritmo para escolha deste conjunto, levando em conta as freqüências das consultas e as freqüências das atualizações nos dados base. Todo o processo de escolha das visões está baseado em um plano de

processamento das visões envolvidas (MVPP – Multiple View Processing Plan), gerado por um algoritmo, que parte dos planos individuais de cada consulta envolvida. O trabalho apresentado por [YKL96] analisa o desenho da visão materializada em termos de desempenho, levando em conta também os recursos utilizados na manutenção da visão. A discussão é apresentada em termos do modelo relacional com operações *select*, *project* e *join*, sendo que a abordagem utilizada pode ser estendida para operações mais complexas, como consultas com agregações e consultas recursivas.

2.1.1 Uma descrição resumida do trabalho de [YKL96]

Nesta seção serão apresentados os conceitos do trabalho realizado por [YKL96]. A apresentação destes conceitos estará baseada em uma base de dados exemplo, que contém as seguintes relações:

Product[3] (Pid, name, Did)

Division (Did, name, city)

Order (Pid, Cid, quantity, date)

Customer (Cid, name, city)

Part (Tid, name, Pid, supplier)

Para simplificar os diagramas, **Pd, Div, Ord, Cust** e **Pt** representam, respectivamente, as relações acima. Além disso, assume-se que estas relações encontram-se todas no mesmo local e, portanto, serão desconsiderados os custos de comunicação de dados para os cálculos que se seguirão.

Supondo que temos as seguintes consultas, freqüentemente utilizadas no acesso ao data warehouse:

Consulta 1:

```
Select  Pd.name
From    Pd, Div
Where   Div.city = "LA" and Pd.Did=Div.Did
```

Consulta 2:

```
Select  Pt.name
From Pd, Pt, Div
Where   Div.city = "LA" and Pd.Did=Div.Did
        And Pt.Pid=Pd.Pid
```

Para cada uma das consultas será gerado um grafo de processamento, que representa seu plano de acesso individual, conforme representado na figura abaixo:

[3] Os nomes dos atributos e das relações envolvidas no exemplo serão mantidos no idioma original.

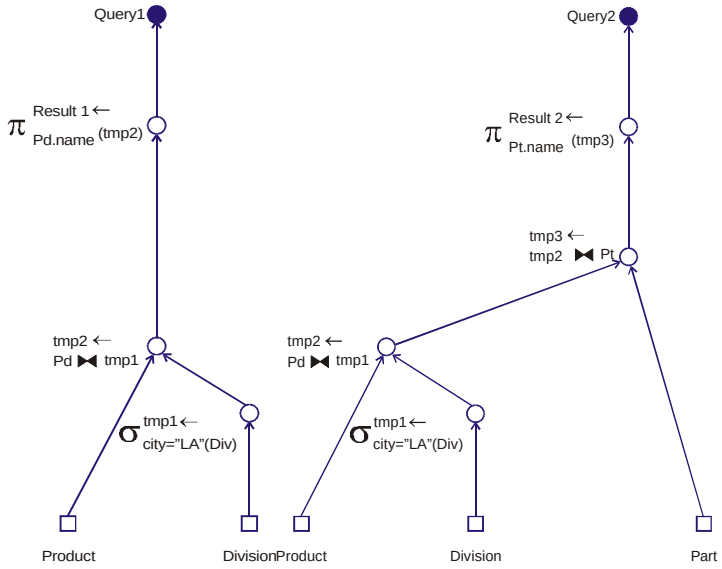

Figura 1 – Plano de acesso individual para as consultas 1 e 2 (query1 e query2).

Uma das alternativas para obter um rápido tempo de respostas para as consultas acima, seria materializar alguns dos nós intermediários de cada plano de acesso individual, sendo que, para o cálculo do custo total, os custos de manutenção das visões também deveriam ser levados em conta. O nó intermediário `tmp2` da `query1` é equivalente ao da `query2`, na figura 1 e são chamados de sub-expressões comuns. Os dois planos individuais podem ser combinados de modo a formar um único plano conforme mostra a figura 2, abaixo:

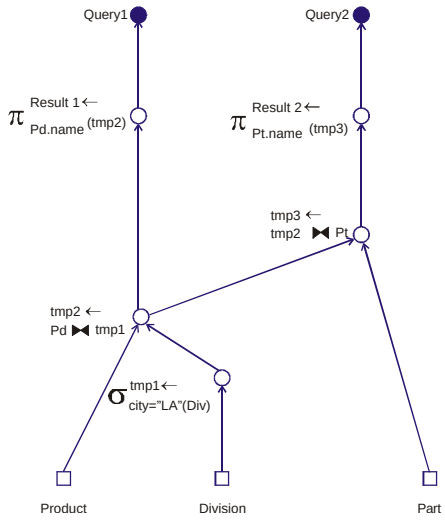

Figura 2 – Plano de acesso combinado para query1 e query2.

Fica claro que, se o nó identificado como `tmp1` for materializado, pode ser utilizado pelas duas consultas, em vez do acesso às relações base `Product`, `Division` e `Part`, diminuindo assim seus custos de execução. Além disso, o custo de manutenção de apenas um nó `tmp1` será menor que o de dois `tmp1`s. Desta forma, haverá ganhos em termos de custo total do acesso global e da manutenção da visão.

Agora, vamos supor duas outras consultas freqüentes ao data warehouse:

Consulta 3:

```
Select   Cust.name, Pd.name, quantity
From     Pd, Div, Ord, Cust
Where    Div.city = "LA" and Pd.Did=Div.Did
         and Pd.Pid=Ord.Pid and Ord,Cid=Cust.Cid
         and date>7/1/96
```

Consulta 4:

```
Select   Cust.city, date
From Ord, Cust
Where    quantity>100 and Ord.Cid=Cust.Cid
```

A figura 3 representa uma plano de acesso global para as quatro consultas, de forma que os planos de acesso locais ou individuais foram combinados com base nas operações compartilhadas dos dados comuns. Este plano global é chamado de **Multiple View Processing Plan (MVPP)**. Após a criação deste plano global, pode-se decidir quais são os nós a serem materializados, de forma que o custo da consulta de o da manutenção da visão sejam mínimos.

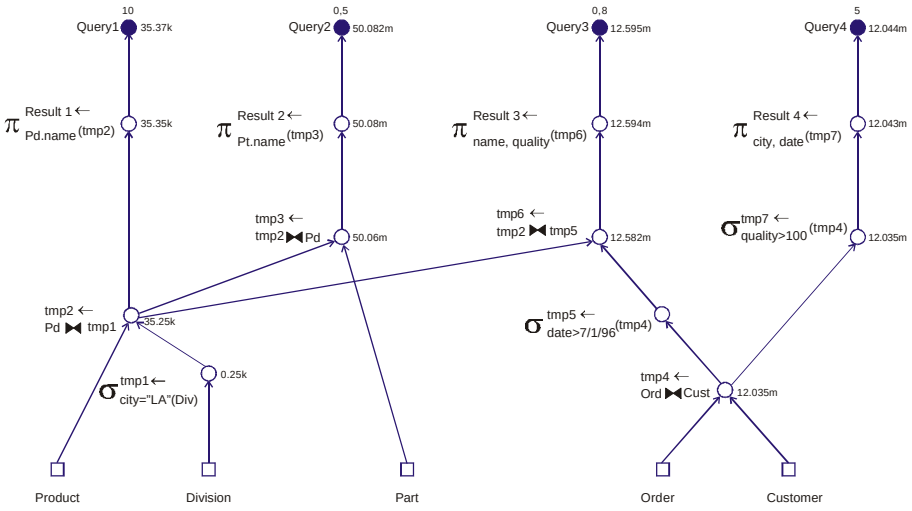

Figura 3 – Um MVPP para o exemplo.

Obviamente, no grafo, existem várias opções para a escolha do conjunto de visões materializadas, a saber: (1) materializar todas as consultas; (2) materializar alguns dos nós intermediários, por exemplo, tmp1, tmp2, tmp4 etc.; (3) manter virtuais todos os nós que não sejam folhas. Para a decisão do melhor conjunto, o custo de cada alternativa deve ser calculado em termos do processamento da consulta e da manutenção da visão.

O trabalho apresentado em [YKL96] define dois algoritmos para implementar uma solução para o problema:

1. **Algoritmo para a definição de múltiplos MVPP:**

   Normalmente para uma consulta existem vários planos de processamento, sendo que, entre eles existe um plano considerado ótimo. Da mesma forma, pode-se ter múltiplos MVPPs baseados em diferentes combinações destes planos individuais. Para reduzir o espaço da pesquisa, o trabalho parte de planos individuais ótimos e os ordena, baseado nas freqüências das consultas e seus custos. Uma vez que a ordem está definida, o algoritmo trabalha com os vários planos individuais, combinando suas sub-expressões comuns chegando a um conjunto de *k* planos globais ou *k* MVPPs. Cada MVPP gerado tem um custo total que será analisado pelo segundo algoritmo descrito em seguida.

2. **Algoritmo para selecionar os nós intermediários a serem materializados:**

   Dado um MVPP, o objetivo é definir um conjunto de visões materializadas tal que o custo total do processamento da consulta e da manutenção da visão seja mínimo, tentando comparar os custos de cada combinação de nós possível. O trabalho apresentado propõe um algoritmo que compara o custo total de cada MVPP gerado, selecionando o de menor custo.

2.2 Caracterizando alguns aspectos da manutenção das visões

Por se comportar como um cache, uma visão materializada oferece acesso rápido aos dados, sendo que esta velocidade pode ser crítica em aplicações onde a quantidade de consultas é alta e a complexidade das visões levam à impossibilidade de recálculo da visão em todo acesso.

Assim, como um cache que se torna obsoleto quando seus dados base sofrem atualização, as visões materializadas também ficam desatualizadas quando as bases de dados de origem são modificadas. Neste caso, é necessária a execução de um processo de manutenção sendo que, na maioria dos casos, é desperdício de recursos proceder com seu recálculo a partir do zero [GM95]. Isto porque apenas uma parte da visão muda em resposta às alterações das bases originais e, portanto, é possível alterar o conteúdo da visão materializada, de forma incremental. Obviamente, isto é apenas heurística. Se tivermos, por exemplo, a eliminação da relação base como um todo, será mais barato recalcular a visão, que depende desta relação eliminada, de forma total, do que apenas aplicar as alterações. Vários trabalhos descrevem algoritmos que permitem a manutenção incremental das visões materializadas, com diferentes domínios de aplicabilidade, como [SI84, CW91, HD92, GMS93].

[GM95] propõe uma classificação do problema de manutenção das visões, através de quatro dimensões, ao longo das quais este problema pode ser analisado, a saber:

- Dimensão da Informação: Refere-se à quantidade de informação disponível para a manutenção da visão. Na análise sob esta dimensão, pergunta-se se o acesso é possível a toda ou apenas parte das relações base e se existe acesso à visão materializada, se são conhecidas regras de integridade e chaves. A quantidade de informação utilizada é ortogonal à “incrementabilidade” da manutenção da visão. “Incrementabilidade” se refere à capacidade de se computar ou calcular apenas aquela parte da visão que foi alterada. Nesta dimensão, portanto, analisamos os dados utilizados para computar as alterações na visão.

- Dimensão da Modificação: Nesta dimensão são analisadas as modificações que o algoritmo de manutenção da visão pode suportar. Questiona-se, então, se inserções ou eliminações de tuplas nas tabelas base são suportadas pelo algoritmo, se as atualizações nas tuplas são tratadas diretamente ou se são modeladas como eliminações seguidas por inserções e aspectos relacionados às alterações na definição da visão.
- Dimensão da Linguagem: Esta dimensão está relacionada a questões sobre a forma com que a visão está expressa, ou seja, foi definida como uma consulta do tipo *select-project-join* (também conhecida como visão SPJ ou como consulta conjuntiva) ou baseada em algum outro subconjunto da álgebra relacional, se utiliza a SQL ou um subconjunto da SQL, se a visão possui duplicidades e se utiliza agregações e recursões.
- Dimensão das Instâncias: Esta dimensão divide-se em dois tipos: informações relativas às instâncias da base de dados e às instâncias da modificação. É importante saber se o algoritmo de manutenção da visão funciona para todas as instâncias da base de dados ou apenas para algumas e se funciona para todas as instâncias da modificação ou apenas para algumas.

Para esclarecer a utilização desta classificação, [GM95] propõe alguns exemplos, que serão transcritos abaixo:

Exemplo 1: Dimensões da Informação e Modificação

Considere a relação:

$$\texttt{part(part\_no, part\_cost, contract)}$$

que lista o custo de uma peça, negociado em um contrato. Note que uma peça pode ter preços diferentes por contrato. Considere também a visão expensive_parts definida como:

$$\texttt{expensive\_parts(part\_no)} = \Pi_{\texttt{part\_no}}\, s_{\ \texttt{part\_cost} > 1000}(\texttt{part})$$

Esta visão contém números de peça **distintos** para as peças que custam mais que $1000 presentes em, ao menos, um contrato (a operação de projeção descarta as duplicidades). Considere a manutenção da visão quando uma tupla for inserida na relação part. Se a tupla inserida tiver part_cost ≤ 1000, então a visão não sofrerá alterações. Entretanto, suponha que part(*p*1, 5000, *c*1) foi inserida, tendo custo>1000. Algoritmos distintos podem ser definidos, dependendo da informação disponível para determinar se *p*1 deve ser ou não inserido na visão:

- Apenas a visão materializada está disponível: utiliza-se a antiga visão materializada para determinar se a peça de número part_no já está presente na visão. Se já estiver, não há alteração na materialização, caso contrário, deve-se inserir a peça *p*1 na materialização.
- Apenas a relação base part está disponível: utiliza-se a relação part para verificar se alguma tupla existente na relação tem o mesmo número part_no porém com custo igual ou maior. Se tal tupla já existir, então a nova tupla inserida não contribui para a visão.
- É conhecido que part_no é a chave: infere-se que a peça de part_no não está na visão e, portanto, deve ser inserida.

Um outro problema de manutenção das visões está relacionado a eliminações utilizando apenas a visão materializada. Considere-se a eliminação da tupla part(*p*1, 2000, *c*12). Está claro que a peça *p*1 tem que estar na materialização, porém não se pode eliminar *p*1 da visão, já que alguma outra tupla, como part(*p*1, 3000, *c*13), pode ter contribuído em levar a peça *p*1 para a visão. A existência (ou inexistência) desta tupla não pode ser provada utilizando-se apenas a visão. Desta forma, não existe um algoritmo que possa resolver este problema de manutenção para eliminações, utilizando apenas a visão materializada. É importante notar que, se a relação part estivesse disponível, ou se fosse conhecida a restrição da chave ou ainda, se a quantidade de tuplas derivadas estivessem disponíveis, então a visão poderia ser mantida.

Com respeito à dimensão da informação, deve-se notar que tanto a definição da visão quanto a alteração real, estão sempre disponíveis para o processo de manutenção e, em relação à dimensão da modificação, as atualizações são tratadas, tipicamente, como uma eliminação seguida por uma inserção.

Exemplo 2: Dimensões da Linguagem e das Instâncias

O exemplo anterior considerou uma visão, cuja definição consiste de operações de seleção e projeção. Para este segundo exemplo, a linguagem de definição da visão será estendida para suportar também a operação *join* , sendo que a visão supp_parts é definida como um *equijoin* entre as relações supp(supp_no, part_no, price) e part($\times_{part\_no}$ representa um *equijoin* no atributo part_no):

$$supp\_parts(part\_no) = \Pi part\_no(supp \times_{part\_no} part)$$

A visão contém os números distintos das peças que são fornecidas por, no mínimo, um fornecedor (a operação de projeção descarta as duplicidades). Para o processo de manutenção deve-se considerar apenas a utilização do conteúdo antigo da visão supp_parts. A inserção da peça part(*p*1, 5000, *c*15), será analisada neste exemplo. Se supp_parts já contiver a peça com part_no *p*1, então esta inserção não afetará a visão existente. Entretanto, se supp_parts não contiver *p*1, então o efeito da inserção não pode ser determinado utilizando-se apenas a visão.

É importante lembrar que a visão expensive_parts, do exemplo anterior, sofreu o processo de manutenção em decorrência das inserções em part, utilizando apenas a visão. Diferentemente, a operação *join* torna impossível a manutenção de supp_parts, como consequência das inserções em part, utilizando-se apenas a visão. Com isso, a visão supp_parts pode ser mantida se já contiver a peça com part_no *p*1, caso contrário, não. Portanto, a capacidade de sofrer manutenção de uma visão depende também de instâncias particulares da base de dados e da modificação.

2.3 Mantendo as visões de forma incremental

Como uma visão materializada é uma relação derivada e definida a partir de relações base, operações de inserção, eliminação e atualização nestas bases de origem fazem com que a visão se torne desatualizada. Recriar a visão como um todo, principalmente em data warehouses onde as relações existentes contém um número muito grande de elementos, pode ser um processo caro, demorado e ineficiente, dado que, na maior parte dos casos, a janela de tempo disponível para a manutenção do data warehouse é limitada. Uma

alternativa bastante utilizada é aplicar na visão, apenas as alterações. Este tipo de manutenção é chamado de manutenção incremental das visões.

A maioria das técnicas de manutenção define as visões como uma fórmula matemática e obtém uma expressão que representa o processo de manutenção. O exemplo abaixo ilustra estas técnicas, conforme [GM95]:

Exemplo 3:

Seja a relação base link(*S, D*), de tal forma que link(*a, b*) é verdadeiro, se existe uma ligação que parte do nó *a* e chega no nó *b*. A visão hop é definida de tal forma que hop(*c, d*) é verdadeiro, se *c* está conectado a *d* utilizando duas ligações, através de um nó intermediário:

$$D: \quad \text{hop}(X, Y) = \Pi_{X,Y}(\text{link}(X, V) \times_{V=W} \text{link}(W,Y))$$

Seja um conjunto de tuplas Δ(link), inserido na relação link. As inserções Δ(hop) que devem ser efetuadas na visão hop, de forma correspondente, podem ser computadas matematicamente, pela definição da diferenciação *D*, para se obter a seguinte expressão:

$$\begin{aligned}\Delta(\text{hop}) = \Pi_{X,Y} \quad & ((\Delta(\text{link})(X, V) \times_{V=W} \text{link}(W,Y)) \cup \\ & (\text{link}(X, V) \times_{V=W} \Delta(\text{link})(W,Y)) \cup \\ & (\Delta(\text{link})(X, V) \times_{V=W} \Delta(\text{link})(W,Y)))\end{aligned}$$

No exemplo 3 citado acima, se as tuplas forem eliminadas da relação link, a mesma expressão pode computar as eliminações da visão hop. Se as tuplas forem inseridas e eliminadas da relação link, então Δ(hop) pode ser obtido através do processamento do conjunto de eliminações Δ^{-}(hop) e do conjunto de inserções Δ^{+}(hop), separadamente [QW91, HD92]. Em [GMS93], para casos especiais, inserções e eliminações podem ser tratados em um único passo.

O trabalho apresentado em [GMS93] apresenta dois algoritmos para manutenção incremental que aplica nas visões materializadas as alterações (inserções, eliminações e atualizações) ocorridas nas relações base, sendo que as visões podem utilizar uniões, negações, agregações (como SUM, MIN) e recursões. Os dois algoritmos utilizam a definição da visão para produzir regras que implementarão as alterações nas visões utilizando, para isto, as alterações ocorridas nas relações base e as visões materializadas na situação anterior às modificações.

O primeiro algoritmo descrito em [GMS93] é um algoritmo de *counting*. Este algoritmo pode ser utilizado em visões com ou sem linhas duplicadas. Sua idéia básica é manter um contador do número de derivações para cada tupla da visão. Para descrevê-lo de forma mais detalhada, dada sua importância, será utilizado como base, o exemplo 3 citado acima, que utiliza a relação base link e a visão hop.

Exemplo 4: A definição da visão hop, em SQL é:

```
CREATE VIEW hop(S,D) as
    (select distinct l1.S, l2.D from link l1, link l2 where l1.D = l2.S)
```

Seja a relação link = {(*a*,*b*), (*b*,*c*), (*b*,*e*), (*a*,*d*), (*d*,*c*)} e, como resultado, a visão hop {(*a*,*c*), (*a*,*e*)}. A tupla hop (*a*,*e*) tem ocorrência única, ou seja, é derivada uma única vez de link, sendo que, a tupla hop (*a*,*c*) possui duas derivações. Se a visão tivesse duplicidade, ou seja, não tivesse sido definida com o operador **distinct**, então hop (*a*,*e*) teria ***count*** igual a 1 e hop (*a*,*c*) teria ***count*** igual a 2. O algoritmo de *counting* simula a duplicidade na visão e armazena estes contadores.

Vamos supor que a tupla link (*a*,*b*) seja eliminada. Neste caso hop seria recomputada como {(*a*,*c*)}. O algoritmo de *counting* infere que uma derivação de cada uma das tuplas hop (*a*,*c*) e hop (*a*,*e*) deve ser eliminada. Baseado nos contadores armazenados, infere também que hop (*a*,*c*) possui ainda uma derivação e portanto, apenas hop (*a*,*e*), que não possui mais nenhuma derivação, deve ser eliminada.

O segundo algoritmo descrito em [GMS93] não pode ser utilizado em visões com duplicidade de tuplas, ou seja, em visões definidas sem o operador **distinct**. Este algoritmo, de nome **DRed** (*Deletion and Rederivation*) calcula as alterações nas visões em três passos. No primeiro passo, o algoritmo faz uma super estimativa das tuplas derivadas de devem ser eliminadas. Uma tupla *t* está nesta super estimativa se as alterações feitas nas relações base invalidam qualquer ocorrência de *t*. Em um segundo passo, esta super estimativa sofre a remoção daquelas tuplas que tenham uma ocorrência ou derivação alternativa na nova base de dados. Finalmente, as novas tuplas que devem ser inseridas são computadas utilizando-se a visão materializada parcialmente atualizada e as inserções feitas nas relações base. Para ilustrar este algoritmo, vamos considerar o seguinte exemplo:

Exemplo 5:

Seja a visão hop, definida no exemplo 4 e a eliminação da tupla link (*a*,*b*). O algoritmo **DRed** primeiramente elimina as tuplas hop (*a*,*c*) e hop (*a*,*e*), já que ambas dependem da tupla eliminada. O algoritmo procura, então, pelas derivações alternativas de cada uma das tuplas eliminadas. Assim, no segundo passo, a tupla hop (*a*,*c*) é novamente derivada e inserida na visão materializada. O terceiro passo do algoritmo é vazio, para este exemplo, já que não existem tuplas a serem inseridas na tabela link.

O processo de manutenção incremental das visões dependem da quantidade de informações disponíveis. Conforme descrito em [GM95], vários trabalhos foram desenvolvidos apresentando algoritmos para a manutenção das visões em função da quantidade de informações disponíveis e do tipo de visão.

Foram analisados os casos onde toda a informação está disponível para o processo de manutenção, ou seja, todas as relações base e as visões materializadas estão disponíveis durante o processo de manutenção. Estes trabalhos consideram também, características da formação das visões. Segue abaixo uma breve descrição dos vários algoritmos definidos nestes trabalhos, agrupados pela quantidade de informação disponível e pelo tipo de visão suportada.

1. **Utilizando informação completa:** A maioria dos trabalhos em manutenção de visões assume que todas as relações base e as visões materializadas estão disponíveis para o processo de manutenção, considerando características como agregações, duplicidades, recursão e *outer join*. As técnicas diferem quanto à linguagem de definição utilizada, quanto ao uso das chaves e regras de integridade e quanto à capacidade de tratar inserções e eliminações de forma separada ou em apenas um passo. As atualizações são

modeladas e tratadas como uma eliminação seguida de uma inserção. Todas as técnicas funcionam para todas as instâncias da base de dados tanto para inserções como para eliminações de tuplas.

1.1. **Visões não recursivas:** as técnicas apropriadas neste caso incluem o algoritmo de *counting* (descrito acima) apresentado por [GMS93], além de outros algoritmos que também utilizam contadores, como [SI84] que cria estruturas com ponteiros que partem da tupla de origem para suas tuplas derivadas. [BLT86] utiliza os contadores exatamente como o algoritmo *counting*, porém apenas para visões do tipo SPJ, computando inserções e eliminações separadamente. O algoritmo de manutenção por diferenciação algébrica, introduzido por [Pai84] e utilizado posteriormente em [QW91] para manutenção de visões, diferencia expressões algébricas para derivar a expressão relacional que determina a alteração em uma visão SPJ. Este tipo de algoritmo gera duas expressões para cada visão: uma para definir as inserções na visão e outro, para definir as eliminações. O algoritmo Ceri-Widom [CW91] define regras de produção para manter as visões SQL que não possuam duplicidades, agregações e negação e aquelas onde os atributos das visões determinam funcionalmente a chave da relação base que está sendo atualizada.

1.2. **Visões com *outer-join*:** a manutenção de visões com *outer-join* é discutida em [GJM94], cujo algoritmo redefine a visão, obtendo duas instruções (uma com *left-outer-join* e outra com *right-outer-join*) para calcular as alterações a serem implementadas.

1.3. **Visões recursivas:** Os algoritmos citados nesta seção aplicam-se também a visões não recursivas. O primeiro deles é o algoritmo DRed [GMS93], descrito acima. Vários outros algoritmos se aplicam a este tipo de visão, estando entre eles: o algoritmo PF (Propagation/Filtration) definido em [HD92], que é muito similar ao DRed, sendo que, para visões não recursivas, o DRed sempre funciona melhor; o algoritmo Kuchenhoff [Kuc91] que cria regras para calcular a diferença entre estados consecutivos da base de dados e o algoritmo Urpi-Olive [UO92] que cria regras de transição mostrando como cada modificação na relação base se traduz em uma modificação em cada relação derivada.

2. **Utilizando informação parcial:** As visões podem ser mantidas utilizando-se somente um subconjunto das relações envolvidas na definição da visão. De forma diferente da utilização da informação completa, quando a manutenção utiliza apenas informação parcial, nem sempre é possível realizar o processo de manutenção da visão. A manutenção vai depender de aspectos como o tipo de modificação, ou seja, se é uma inserção, uma eliminação ou atualização. Portanto, os algoritmos neste caso devem se preocupar se a visão pode ser mantida e então, como efetuar a manutenção, sendo que em alguns casos, mesmo que não exista o algoritmo para eliminação+inserção, é possível existir o algoritmo para atualização.

2.1. **Nenhuma informação disponível:** Muitos trabalhos têm sido feitos para determinar quando uma modificação na base não acarreta alteração na visão. Esta situação é conhecida como “o problema da atualização irrelevante”. [BLT86, BCL89, Elk90, LS93] determinam atualizações irrelevantes em várias condições diferentes.

2.2. **Utilizando a visão materializada: *Self-Maintenance.*** As visões podem ser mantidas utilizando-se apenas a visão materializada e as regras implementadas através das chaves. Estas visões são tratadas por [GJM94], que apresenta vários resultados em *self-maintenance* em visões do tipo SPJ e com *outer-join*, respondendo a inserções, eliminações e atualizações. Este trabalho define uma visão *self-maintainable* com respeito ao tipo de modificação, não considerando as visões com *self-joins* ou *outer-joins*, as visões que não usam informações das chaves e as que não consideram a capacidade de se auto manter com respeito a todas as instâncias de modificações.

3. **Utilizando a visão materializada e algumas relações base: Referência parcial**. O problema da manutenção com referência parcial é manter a visão, dados apenas um subconjunto das relações base e a visão materializada. Dois subproblemas interessantes surgem desta abordagem: o primeiro é quando estão disponíveis a visão e todas as relações base, com exceção da relação que sofreu a modificação e o segundo, quando estão disponíveis a visão e a relação modificada. Vários trabalhos, como [JMS95, GB95, Gup94], tratam de variações desta situação.

3 Conclusão

Um data warehouse integra dados provenientes de fontes heterogêneas, distribuídas e autônomas, sumarizando-os e disponibilizando-os para consultas e análises através de aplicações OLAP (On-Line Analytical Processing) e sistemas de suporte à decisão [GM96].

Os data warehouses vêm sendo muito utilizados pelas empresas do mundo todo, já que proporcionam um alicerce sólido de integração de dados corporativos e históricos para a realização de análises gerenciais. Sua construção e implementação é feita de uma maneira passo a passo, organizando e armazenado os dados sob uma perspectiva de longo prazo. Assim, partindo-se dos dados históricos básicos, pode-se realizar análises de tendências [WIRH97].

Por sua característica básica, que é a integração de dados provenientes de várias fontes diferentes, a etapa mais complexa na implementação de um data warehouse é o processo de carga. Neste processo, os dados distribuídos pelos vários ambientes operacionais (bases de dados de produção, que contêm os dados utilizados pelos vários sistemas transacionais de uma empresa) devem ser selecionados, trabalhados com o objetivo de padronização e limpeza, transferidos para o novo ambiente e finalmente carregados, sempre atendendo ao padrão da modelagem utilizada para o data warehouse. Este processo é feito periodicamente, sendo que sua freqüência depende de vários fatores relacionados ao modelo de negócios utilizado pela empresa e, normalmente, não é menor que 24 horas. Desta forma, podemos dizer que os dados armazenados no data warehouse são, para todos os propósitos práticos, uma longa série de fotografias, tiradas ao longo do tempo. Uma vez que os dados são armazenados no data warehouse, eles não mais sofrem atualizações, sendo, portanto, um ambiente apenas de carga e acesso.

Após sua criação e primeira carga, o data warehouse passa a sofrer cargas incrementais que devem refletir o ambiente operacional ao longo do tempo tornando-o uma imensa base de dados para os sistemas de apoio à decisão.

Torna-se claro, portanto, que o data warehouse tem características de um ambiente estático, que só reflete as alterações ocorridas no ambiente operacional após períodos pré-definidos. Este fato em si não é tão grave, já que em várias áreas de negócio, as análises são feitas baseadas em resumos mensais, por exemplo. O que é um fator crítico é a alta complexidade do processo de carga que se transforma em um ponto muito suscetível à introdução de erros, que podem levar ao colapso de todo o processo de tomada de decisão.

Como o data warehouse sofre periodicamente novas cargas, aumentando constantemente seu tamanho, surge mais um grande problema em sua utilização, que é a dificuldade em responder às consultas dos usuários de forma rápida e eficiente.

Com o objetivo de conferir ao data warehouse uma característica mais dinâmica e otimizar o acesso aos dados, vários estudos têm sido feitos indicando o uso de *visões materializadas* como alternativa viável.

Uma visão é uma relação derivada, definida em termos de relações base, que é computada todas as vezes em que uma referência a ela é feita. Uma visão é dita materializada quando ela é realmente armazenada na base de dados em vez de ser computada a partir das relações base em resposta a consultas. Uma visão materializada pode ser vista como um cache – uma cópia dos dados que pode ser acessada rapidamente. Os data warehouses podem, portanto, armazenar estas visões materializadas com o objetivo de possibilitar acesso rápido à informação que está integrada a partir de diferentes fontes de dados distribuídas [DEB95].

O uso de visões materializadas traz vários aspectos que devem ser cuidadosamente analisados e resolvidos, entre eles:

1. a seleção das visões mais adequadas levando-se em conta uma análise dos custos de manutenção e os benefícios de cada visão e os algoritmos disponíveis;
2. os mecanismos que permitem a propagação correta quando da atualização das fontes de dados base para vários tipos de visões, preferencialmente de forma online e incremental;
3. a manutenção das visões de forma dinâmica e preferencialmente autônoma utilizando o conceito de *self maintainable views*.

Se confrontado com as características estáticas dos data warehouses atuais, a abordagem mais dinâmica implementada através das visões materializadas, resulta na definição de diretrizes para metodologia de modelagem e implementação do data warehouse. O resultado obtido será uma classificação dos domínios de aplicações de acordo com os paradigmas do data warehouse citados acima.

O levantamento bibliográfico realizado por este trabalho teve como objetivo analisar os aspectos relacionados à implementação das visões materializadas nos data warehouses. Esta análise irá complementar a base de uma dissertação de mestrado que tem por motivação a apresentação das abordagens estáticas e dinâmicas para construção de data warehouses em contextos diferenciados já que, em muitos casos, não se pode ter avaliar a eficiência de uma determinada abordagem em função de outra. A dissertação, que tratará dos aspectos da

metodologia do uso de visões materializadas em data warehouse, objetivará melhor caracterizar as situações em que uma abordagem pode ser utilizada em detrimento da outra.

Após a análise da abordagem estática, discutindo os modelos atualmente utilizados na implantação dos data warehouses, serão apresentadas, na dissertação, as características de um ambiente mais dinâmico utilizando as visões materializadas, incluindo todos os aspectos analisados neste trabalho. Uma classificação dos domínios de aplicação de acordo com os paradigmas do data warehouse, será definida a partir do levantamento de várias características dos métodos e aplicações, buscando-se construir elementos de uma taxonomia capaz de fornecer subsídios para os projetistas de data warehouse. Um modelo de simulação será criado para validar esta taxonomia, evidenciando os elementos e conceitos definidos através da implementação de um modelo de simulação. A conclusão da dissertação deverá dar subsídios para definição de diretrizes na construção de data warehouses híbridos, considerando que a proporção entre data warehouse estático e dinâmico pode ser alterado em função da evolução das aplicações.

Referências

[BCL89] J.A. Blakeley, N. Coburnm, P. Larson. “Updating derived relations: detecting irrelevant and autonomously computable updates”. *AM Transactions on Database Systems*, 14(3):369-400, 1989.

[BLT86] J.A. Blakeley, P. Larson, F. Tompa. *Efficiently updating materialized views.* SIGMOD, 1986..

[CW91] S. Ceri, J. Widom. *Deriving incremental production rules for incremental view maintenance. 17th VLDB*, 1991.

[DEB95] *IEEE Data Engineering Bulletin, Special Issue on Materialized Views and Data Warehousing*, 18(2), junho 1995.

[DQ97] D. Quass. *Materialized views in data warehouses.* Tese de doutoramento do Departamento de Ciência da Computação da Stanford University. Agosto 1997.

[Elk90] C. Elkan.”Independence of logic database queries and updates”. *9th PODS*, pág. 154-160, 1990.

[GB95] A. Gupta, J. A. Blakeley. *Maintaining views using materialized views.* Documento não publicado.

[GHRU96] H. Gupta, V. Harinarayan, A. Rajaraman, J. Ullman. *Index selection in OLAP*. 1996.

[GM95] A. Gupta, I. S. Mumick. *Maintenance of materialized views: problems, techniques, and applications.* 1995.

[GM96] A. Gupta e I. S. Mumick. “What is the data warehousing problem? (Are materialized views the answer?)”. *Proceedings of the 22nd VLDB*

Conference, Mumbai (Bombay), India, 1996.

[GMS93] A. Gupta, I. S. Mumick, V. S. Subrahmanian. *Maintaining views incrementally*. 1993

[Gup94] A. Gupta. *Partial information based integrity constraint checking*. Tese de doutoramento da Stanford University (CS-TR-95-1534).

[JMS95] H. V. Jagadish, I. S. Mumick, A. Silberschatz. "View maintenance issues in the chronicle data model". *14th PODS*, pág. 113-124, 1995.

[LS93] A.Y. Levy, Y. Sagiv. "Queries independent of updates". *19th VLDB*, pág. 171-181, 1993.

[HD92] J. Harrison, S. Dietrich. "Maintenance of materialized views in a deductive database: an update propagation approach". *Workshop on Deductive Databases, JICSLP*, 1992.

[HG97] H. Gupta. *Selection of views to materialize in a Data Warehouse*, 1997.

[HRU96] V. Harinarayan, A. Rajaraman, J. Ullman. "Implementing data cubes efficiently". *ACM Sigmod Intl. Conf. on Mngt. of Data*, 1996.

[Pai84] R. Paige. "Applications of finite differencing to database integrity control and query/transaction optimization". *Advances in Databse Theory*, pág. 170-209, Plenum Press, New York, 1984.

[QGMW97] D. Quass, A. Gupta, I. S. Mumick, J. Widom. *Making views self-maintainable for data warehousing.* 1997.

[QW91] X. Qian, G. Wiederhold. "Incremental recomputation of active relational expressions". *IEEE TKDE*, 3(1991), pág. 337-341.

[RK96] R. Kimball. *The data warehouse toolkit.* John Wiley & Sons, 1996.

[SI84] O. Shmueli, A. Itai. *Maintenance of views*. *Sigmod Record*, 14(2):240-255, 1984.

[WIRH97] W. H. Inmon, R. D. Hackathorn. *Como usar o Data Warehouse*. Infobook, Rio de Janeiro, 1997.

[YKL96] J. Yang, K. Karlapalem, Q. Li. *A framework for designing materialized views in data warehousing environment.* Technical Report HKUST-CS96-35. Outubro 1996.

[ZGM+94] Y. Zhuge, H. Garcia-Molina, J. Hammer e J. Widom. *View maintenance in a warehousing environment*. Technical report, Stanford University. Disponível: ftp://db.stanford.edu como pub/zhuge/1994/anomaly-full.ps. Outubro de 1994.