3
Heurísticas para sintonia de Visão Materializada na Arquitetura

Nesse capítulo apresentamos a arquitetura para auto-sintonia de visões materializadas (VM) em banco de dados relacionais e as heurísticas necessárias para implementação dessa arquitetura. A estratégia proposta utiliza a arquitetura apresentada em [Monteiro08], seguindo a abordagem não-intrusiva, ou seja, desacoplada do código do SGBD.

3.1.Arquitetura não-intrusiva para criação de visões materializadas

Como apresentado inicialmente em [Weikum94] e utilizado no trabalho de [Monteiro08], esta dissertação segue o ciclo de auto-sintonia clássico: Observação, Predição e Reação. Essas fases podem ser continuamente executadas sem necessariamente sobrecarregar o SGBD, uma vez que não são intrusivas e podem ser executadas em uma máquina distinta.

Desta forma as fases podem ser executadas em uma máquina apenas acessando a máquina com o servidor de banco de dados apenas para consultas rápidas aos metadados, sem atrapalhar o funcionamento do banco de dados na outra máquina. As execuções de cada fase têm a finalidade de monitorar o comportamento corrente do sistema, sugerir decisões de mudança sobre o comportamento futuro e aplicar essas mudanças, caso necessário.

Como apresentado no capítulo 2, os pontos importantes para seleção de visões materializadas podem ser implementados nas fases do ciclo de auto-sintonia clássico.

- Observação (Monitoramento da Carga de Trabalho e Seleção de VM)

  Na fase de observação é onde são tratados quatro pontos importantes para seleção automática de visões materializadas. Nesta fase é capturada a carga de trabalho submetida ao SGBD alvo e armazenada em uma metabase local. Cada cláusula é capturada e analisada, observando a frequência dos comandos

executados, para definir quais consultas serão usadas para definir a visão. Como a carga é capturada periodicamente, a definição das visões é feita à medida que a carga é capturada, dessa forma atendendo as variações da carga de trabalho. É nesta fase que as visões materializadas, que não existem fisicamente no banco de dados, mas podem diminuir potencialmente o tempo da consulta em questão, são eleitas, chamada de visões materializadas hipotéticas.

- Predição (Seleção Final de Visões Materializadas)

  Com intuito de determinar o momento de materialização da visão, utilizamos a heurística de benefícios. A idéia de heurística de benefícios já foi usada em outros trabalhos de auto-sintonia do projeto físico de banco de dados [Salles04 e Monteiro06], mas sobre estruturas de índices. Nesta fase são utilizadas heurísticas para atribuir benefícios às visões materializadas hipotéticas. A heurística proposta nesta dissertação, e apresentada na figura 17, procura dar benefícios a uma visão materializada hipotética cada vez que se estima que uma consulta otimize o desempenho se a visão existisse fisicamente no banco de dados. A heurística de seleção final de visão materializada define que uma visão hipotética está na hora de ser materializada quando seu benefício acumulado chega a um limite definido. O limite escolhido nessa dissertação é o custo de criação da visão materializada. Na seção 3.1.2 discutiremos a escolha do limite.

- Reação (Criação de Visões Materializadas)

  Ao final da predição, caso sejam detectadas visões hipotéticas a serem materializadas, a fase de reação é iniciada. Nesta fase são fisicamente criadas todas as visões que foram definidas pela fase de predição e cujo benefício acumulado atingiu o limite estipulado.

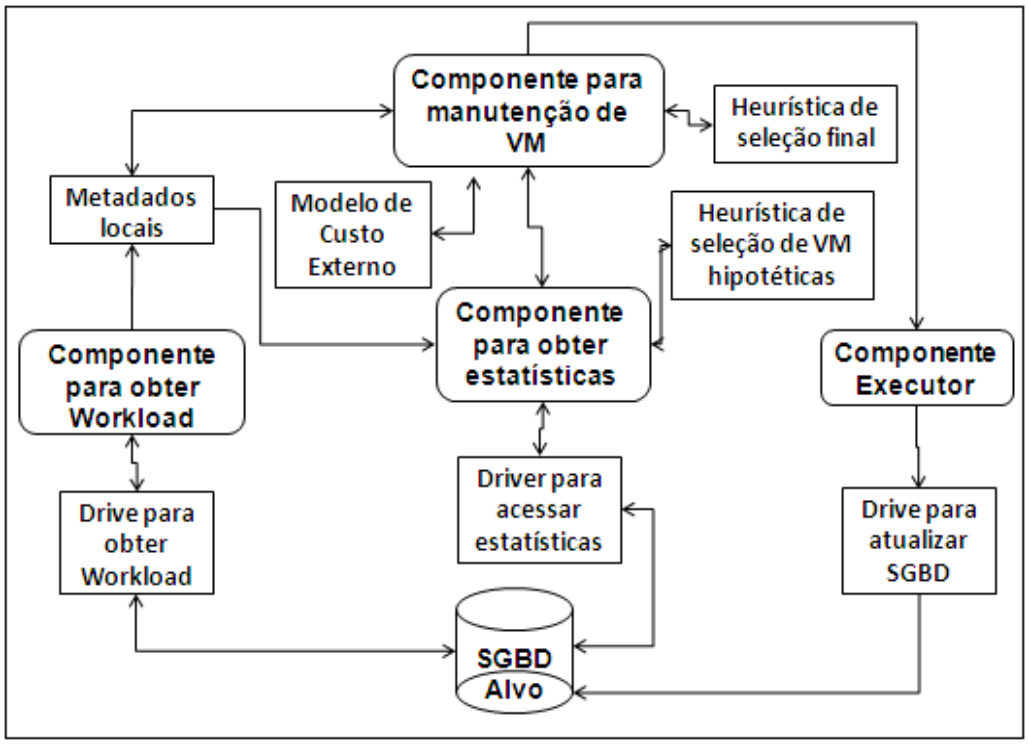

Figura 7 - Arquitetura para gerência de Visão Materializada

A figura 7 exibe a arquitetura utilizada para criação automática de visões materializadas utilizando o ciclo de auto-sintonia. Cada componente desempenha um papel especifico na arquitetura. Nas subseções seguintes são detalhadas as funções de cada componente.

3.1.1. Monitoramento da Carga de Trabalho e de seleção de Visões Materializadas Hipotéticas

Esta é a fase de observação, nela são feitas consultas periódicas à metabase do banco de dados alvo para capturar a carga de trabalho que lhe é submetida por intermédio do componente CWO (*Component for Workload Obtainment*). O CWO armazena as consultas capturadas junto dos seus respectivos planos na metabase local para que o componente *Component for Statistics Obtainment* (CSO) possa consultá-las depois.

Cada cláusula SQL capturada é analisada a fim de selecionar visões materializadas interessantes, ou seja, que podem diminuir o tempo de resposta das cláusulas executadas. Esta análise é feita pelo CSO. Este componente é responsável por acessar as informações capturadas pelo CWO para obter novas estatísticas tais como, custos estimados das consultas capturadas, subconjuntos de tabelas TS interessante, ou seja, que obedecem às métricas *TS-Cost* e *TS-Weight* -

que serão definidas ainda nesta seção - e a quantidade de tuplas estimadas de uma visão.

O CSO é responsável por selecionar as visões materializadas candidatas. Para tanto utiliza um algoritmo para encontrar o conjunto TS interessante proposto em [Agrawal et al. 00].

Este algoritmo foi escolhido, pois ele considera a frequência em que as consultas são submetidas e o tamanho das tabelas referenciadas por essas consultas. Entende-se por tamanho de tabela o tamanho do espaço que ela ocupa em disco. O tamanho da tabela é importante, pois uma consulta sobre uma tabela de 2GB é mais oneroso que uma consulta sobre uma tabela de 100KB.

Considere Q o conjunto de cláusulas submetidas ao banco, T um subconjunto de tabelas, e as seguintes métricas utilizadas pelo algoritmo:

- *TS-Cost(T)*

  TS-Cost(T) é o custo total de todas as cláusulas na carga de trabalho onde o subconjunto de tabelas T ocorre. Este custo pode ser obtido pela soma das estimativas de custo de cada cláusula.

- *TS-Weight(T)*

  TS-Weight é a métrica responsável pela importância relativa de um subconjunto de tabelas. Considere que *sizeTables*(T) é a soma dos tamanhos das tabelas em T e que *sizeTablesClause*(Q_i) é a soma de todas as tabelas referenciadas em Q_i. Então, *TS-Weight* é definida como:

$$\sum_i Cost(Qi) * \frac{sizeTables(T)}{sizeTablesClause(Qi)}$$

O somatório de *TS-Weight* é apenas sobre cláusulas Q_i onde T ocorre. O algoritmo retorna TS que obedeçam às duas métricas definidas acima. O algoritmo pode ser visto na figura 8.

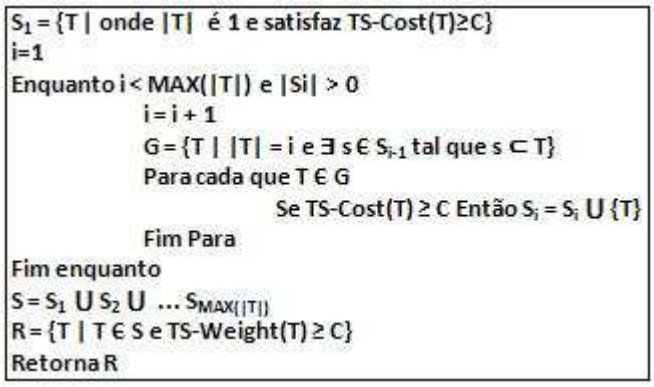

Figura 8 - Algoritmo para encontrar subconjunto de tabelas interessantes (*InterestingTableSubsets*)

No algoritmo da figura 8, |T| representa a quantidade de tabelas em *T*, e *C* uma porcentagem do total da carga de trabalho submetida. Para o SGBD SQL Server, [Agrawal et al. 00] utilizou C igual a 10% do total da carga de trabalho. *MAX(|T|)* representa a quantidade máxima de tabelas referenciadas em uma consulta na carga de trabalho.

A métrica *TS-Cost(T)* permite levantar algoritmos eficientes que propõem identificar conjunto de itens frequentes. Já a métrica *TS-Weight(T)* é responsável pela importância relativa de cada subconjunto de tabelas de acordo com os tamanhos das tabelas presentes em *T*.

O algoritmo de selecionar TS interessantes tem 2 passos: (a) eliminar subconjuntos de tabelas que não satisfazem a métrica *TS-Cost(T)* e (b) eliminar subconjuntos de tabelas obtido em (a) que não satisfaçam a métrica *TS-Weight(T)*.

A utilização deste algoritmo reduz o espaço de enumeração de visões materializadas, pois elimina subconjuntos de tabelas sobre as quais não é interessante materializar visões.

Em seguida são criadas visões materializadas interessantes sobre cada consulta que referencia um item deste conjunto, gerando o conjunto *V*. Para isso, criamos um algoritmo para definição da visão. Tomemos a consulta B sobre um subconjunto de tabela interessante encontrado {Venda} e a visão C em construção:

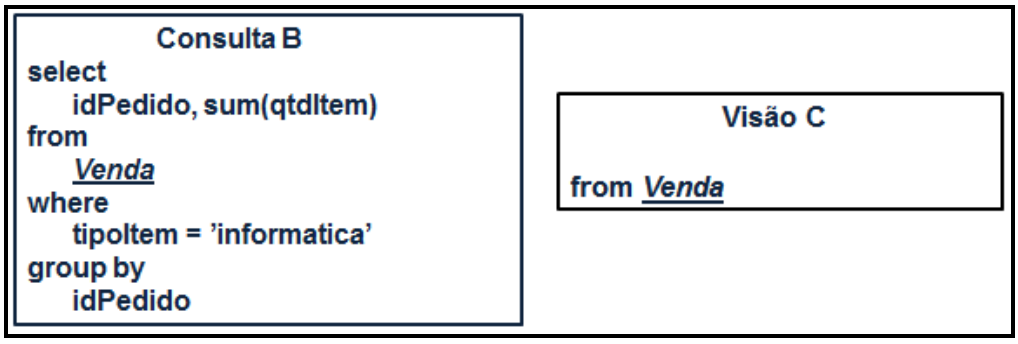

Figura 9 - Passo 1 para definição de Visão Materializada

Como mostrado na seção 2.7, notou-se que o otimizador utiliza uma visão materializada quando ela for compatível com uma consulta por inteiro. Entende-se por compatível quando todos os campos de seleção e agregação da consulta estiverem presentes na visão e todas as junções da consulta estiverem presentes na visão.

Desta forma, um ponto importante a observar são as tabelas que irão ser referenciadas no *from* da visão materializada. Como definimos visões sobre consultas, para que a visão em construção seja útil para essa consulta, ela também deverá referenciar o subconjunto de tabela interessante. Desta forma, o subconjunto de tabela {Venda} é referenciado pela visão C em construção. (conforme figura 9).

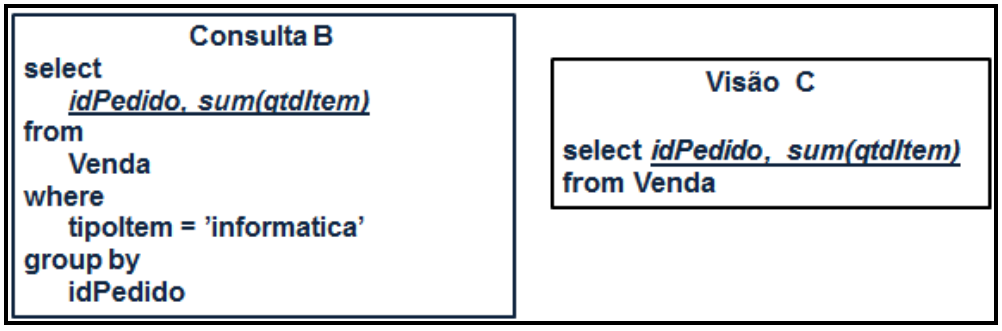

Figura 10 - Passo 2 para definição de Visão Materializada

É, também, importante observar os campos de seleção da consulta, pois para que a consulta utilize a visão materializada, é necessário que a visão contenha todos os campos necessários para a consulta. Desta forma os campos de seleção da consulta devem estar presentes nos campos de seleção da visão, como pode ser vista na figura 10.

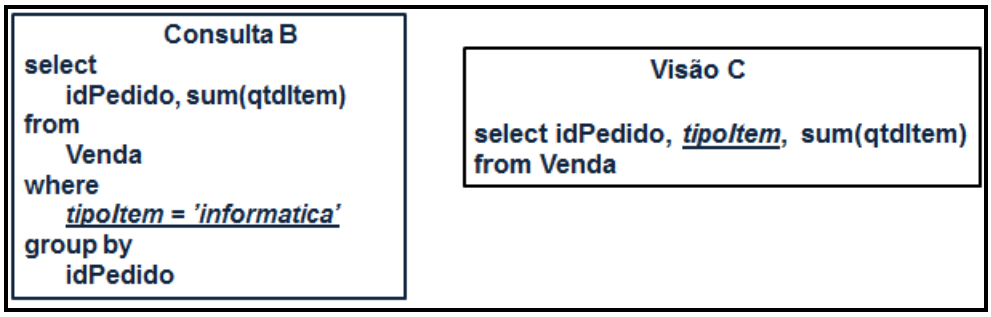

Figura 11 - Passo 3 para definição de Visão Materializada

Não deve deixar de ser observado os filtros das consultas, pois da mesma forma que a visão deve conter todos os campos de seleção da consulta para que seja útil, ela também deve conter os filtros da consulta. Nesta dissertação, os filtros foram definidos como campos de seleção da visão conforme figura 11.

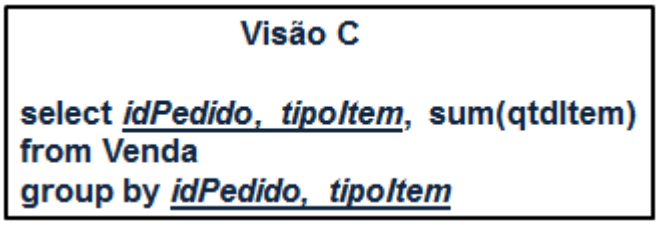

Figura 12 - Passo 4 para definição de Visão Materializada

Faz-se necessário também, definir o agrupamento sobre a consulta que define a visão. Para tanto, o agrupamento é feito sobre os campos de seleção da visão que não sejam funções de agregamento como mostrado na figura 12.

Outro ponto é observar se a consulta tem uma condição *having*. Para que a visão seja útil para a consulta, os campos *having* da consulta devem estar presentes nos campos de seleção da visão. Observe a consulta D e a visão E definida sobre ela como exemplo de tratamento do campo *having*.

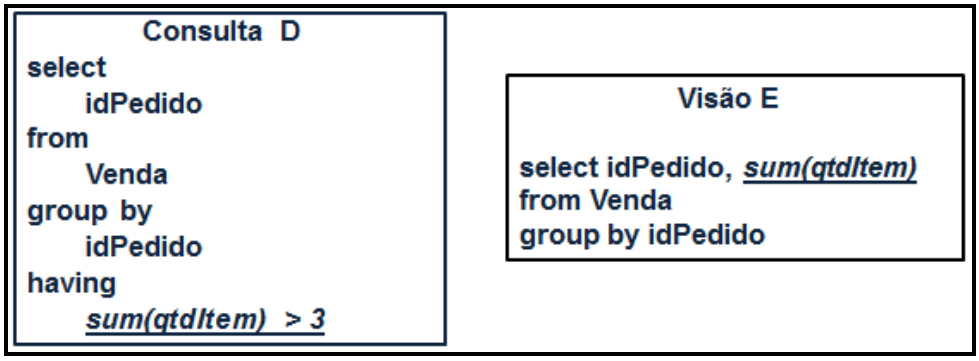

Figura 13 - Passo 5 para definição de Visão Materializada

Neste trabalho focamos em montar visões sobre consultas fazendo com que o campo de filtro ou de *having da* consulta seja um campo de seleção da visão.

Isto porque a carga de trabalho varia ao longo do tempo e a faixa de valores do filtro e a condição do *having* da visão pode vir a não ser compatível com a consulta conforme figura 13

Considerando os pontos importantes a observar para definir uma visão, propomos um algoritmo para definição de visão a partir de uma consulta apresentado na figura 14. Esta consulta, utilizada para definir a visão, pode ser uma subconsulta de uma consulta aninhada, porém ela deve ser uma consulta básica, para garantir que o otimizador utilize a visão materializada que é compatível com ela.

1. Tornar tabelas referenciadas na consulta em tabelas referenciadas na visão
2. Tornar campos de projeção da consulta em campos de projeção da visão
3. Tornar campos de filtro da consulta em campos de projeção da visão
4. Tornar campos de *having* da consulta em campos de projeção da visão
5. Definir *group by* através dos campos de projeção não agregados da visão
6. Retorna Visão

Figura 14 - Algoritmo DefineView

Com um algoritmo de definição de visão definido, propomos um algoritmo interessante para criar visões. Este deve (a) selecionar consultas que utilizem subconjunto de tabelas interessantes encontrados e (b) definir consultas de visões sobre essas consultas através do algoritmo de definição de visão. Este algoritmo, *MountVM*, é apresentado na figura 15.

```
1.  T = subconjunto de tabelas interessantes
2.  Q = carga de trabalho
3.  i = 0
4.  V = {}
5.  Enquanto i < |Q|
6.      Q' = Todas consultas básicas de Qi
7.      i = i + 1
8.      k = 0
9.      Enquanto k < |Q'|
10.         Se o conjunto de tabelas referenciadas em Q'k está em T
11.             v = DefineView(Q'k)
12.         Fim Se
13.         V = V U {v}
14.         k = k + 1
15.     Fim Enquanto
16. Fim Enquanto
```

Figura 15 - Algoritmo *MountVM* para criar as definições de consultas de visões materializadas

Na fase de observação são selecionadas as visões materializadas hipotéticas, ou seja, selecionam-se visões que não existem fisicamente no banco de dados e que podem diminuir potencialmente o tempo de resposta a uma consulta.

1. N = novas consultas analisadas por CSO
2. P = P ⋃ N
3. T = InterestingTableSubsets(P)
4. Para cada t em T
5. $V_{VM} = V_{VM} \cup$ MountVM(t, P)
6. Fim para
7. H_{VM} = visões hipotéticas anteriores
8. $V_{VM} = V_{VM} \cup H_{VM}$
9. H_{VM} = Merge(V_{VM})
10. Retorna H_{VM}

Figura 16 - Heurística para selecionar visões materializadas hipotéticas

Na figura 16 pode ser visto como é feita a utilização do algoritmo para obter subconjunto de tabelas (TS) interessante de [Agrawal et al. 00], com o algoritmo *MountVM* proposto nessa dissertação. Este algoritmo pode ser usado de forma *on-the-fly* para retornar o conjunto de visões materializadas hipotéticas.

Neste algoritmo só são analisadas consultas capturadas que não foram analisadas ainda. Para definir o subconjunto de tabelas interessantes são utilizadas as métrica *TS-Cost* e *TS-Weight* considerando toda a carga de trabalho executada até o momento. A frequência de captura da carga de trabalho é um valor configurável.

O conjunto *N* representa o conjunto das últimas consultas que foram capturadas pelo CWO e analisadas pelo componente CSO em busca do subconjunto de tabelas na consulta e subconsultas. O conjunto *P* representa o conjunto de todas as consultas analisadas pelo componente CSO das iterações anteriores. Ao utilizar o algoritmo para obter o subconjunto de tabelas interessantes é avaliada a carga de trabalho como um todo, pois assim obtemos *TS-Weight* e *TS-Cost*.

Em resumo, é na fase de observação que são selecionadas as visões materializadas hipotéticas, ou seja, seleciona-se visões que não existem fisicamente no banco de dados e que podem diminuir potencialmente o tempo de resposta a uma consulta.

3.1.2.Seleção Final de Visões Materializadas proposta

Esta é a fase de predição, nela cada visão materializada hipotética é avaliada e seu benefício é calculado. Para tal são utilizadas as heurísticas de benefícios propostas neste capítulo e que podem ser visualizadas na figura 17.

A execução da heurística de benefício é realizada pelo *Component for Materialized View Management* (CMVM), que é o componente responsável pelo acúmulo do beneficio de cada visão materializada hipotética.

Para seleção final de visões materializadas (VM) definimos alguns fatores para serem utilizados pelas heurísticas de avaliação de consultas e de criação de visão materializada que são apresentados a seguir:

- $Clause_{VM}$: é a definição da consulta de uma VM
- Cost(q): é o custo estimado pelo otimizador de executar a consulta q na ausência de visões hipotéticas
- HyCost(q): é o custo estimado de executar a consulta q na presença de visões hipotéticas
- Hy_{VM}: São VMs hipotéticas obtidas a partir do algoritmo de seleção de visões hipotéticas
- $HyGain_{VM}$: É o ganho hipotético de uma visão a ser materializada ao executar uma consulta que pode vir a utilizar esta VM hipotética.
- $HyGainAC_{VM}$: É o ganho hipotético acumulado da VM.
- $HyRow_{VM}$: Quantidade de tuplas da Hy_{VM} (estimado pelo otimizador).
- $SizeRow_{VM}$: Tamanho máximo de uma tupla da VM.
- $HyPg_{VM}$: Quantidade de páginas estimadas da VM hipotética.
- $CreationCost_{VM}$: É o custo de criação da VM.

Os fatores $HyGain_{VM}$, $HyGainAC_{VM}$ e $CreationCost_{VM}$ são fatores semelhantes aos B_K, BA_{CK} e CC_K respectivamente, proposto por [Monteiro06]. Os outros fatores foram criados por esta dissertação para obtenção dos fatores $HyGain_{VM}$, $HyGainAC_{VM}$ e $CreationCost_{VM}$.

Informações como quantidade de tuplas de uma tabela, tamanha máximo da tupla são informações que podem ser obtidas nos metadados do SGBD e o tamanho de página de um SGBD pode ser encontrado em sua documentação. Desta forma, os fatores propostos nessa dissertação podem ser calculados para qualquer SGBD.

O modelo de custo externo utilizado pelo componente CMVM é um modelo para estimar custos hipotéticos independente de SGBDs. O CMVM implementa o

modelo de forma que chama métodos do componente de estatísticas para obter custos hipotéticos.

Sabe-se que uma visão materializada armazena dados já calculados provenientes de uma ou mais tabelas. Desta forma, o custo de uma consulta na qual existe uma visão que é compatível com a consulta, é o custo de um *full scan* nesta visão materializada. Considerando que ainda não foi criado nenhum índice sobre a visão materializada, será feito um *full scan* para fazer uma busca em uma visão materializada.

Desta forma, o custo hipotético (*HyCost(q)*) de executar uma consulta que na presença de uma visão materializada hipotética que é compatível com ela é o custo de um *full scan* na visão hipotética. Para estimarmos o custo de consultar uma visão materializada, devemos considerar que ela é uma tabela proveniente de dados calculados de outras tabelas.

Em [Hose et al. 09] é proposta uma métrica para estimar o custo de uma varredura simples em uma tabela. Estima-se que o custo de uma varredura (C_T) em uma tabela é o (número de tuplas * tamanho máximo da tupla * *fillfactor* / tamanho da página). Considerando que uma visão materializada pode ser tratada como uma tabela, podemos estimar o custo de uma varredura em uma visão materializada como:

- $HyCost(q) = HyRow_{VM} * SizeRow_{VM} * fillfactor / tamanho\ da\ página$

Precisamos estimar o tamanho máximo de uma tupla da visão materializada ($SizeRow_{VM}$) e a quantidade estimada de tuplas da visão. O tamanho máximo de cada tupla é a soma dos tamanhos de cada coluna. O tamanho máximo da coluna é definido pelo tipo de dados da coluna. Para saber o tipo[1] é necessário acessar a metabase do SGBD alvo.

A quantidade de tuplas em uma visão materializada pode ser estimada pelo otimizador. Para tanto é necessário obter o plano de execução da consulta que define a visão materializada, que exibe a quantidade estimada de tuplas da consulta. Com essas duas medidas já podemos estimar o custo de uma varredura em uma visão caso ela existisse fisicamente.

Como já mencionado, o custo de uma consulta na qual existe uma visão compatível com a consulta é o custo de um *full scan* nesta visão materializada. Então podemos estimar o benefício ($HyGain_{VM}$) de uma visão materializada *vm*

pela diferença entre o custo de executar uma consulta *q*, cuja visão *vm* seja compatível com a consulta *q*, e o custo estimado da consulta (q_{vm}) que define a visão materializada *vm*.

- $HyGain_{VM} = Cost(q) – HyCost(q)$

As heurísticas propostas nessa dissertação são baseadas no acúmulo de benefícios de uma visão materializada hipotética e no custo de criação desta. A idéia dessa heurística é não permitir materializar visões por conta de uma consulta pontual, pois a VM tem que ser útil para a carga como todo, e não apenas para uma consulta que é submetida uma vez e pode nunca mais ser submetida. Na figura 17 está uma descrição da heurística de avaliação de consultas onde são acumulados os benefícios das visões materializadas hipotéticas. Este acúmulo é realizado até que se atinja um determinado limite que é o custo estimado de criação da visão materializada.

```
1.  F = {}
2.  Para cada visão materializada hipotética (vmi)
3.     HyGain_VMi = 0
4.     Se q_VMi é compatível com q
5.        HyGain_VMi = Cost(q) – HyCost(q)
6.     Fim se
7.     HyGainAC_VMi = HyGainAC_VMi + HyGain_VMi
8.     Se HyGainAC_VMi > CreationCost_VMi
9.        F = F ⋃ {VM_i}
10.    Fim se
11. Fim Para
```

Figura 17 - Heurística para seleção final de Visão Materializada

Para definir o custo de criação da visão materializada, é levado em consideração que a consulta que define a visão será calculada. Então, a primeira parte do custo de criação da visão materializada é considerar o custo estimado pelo otimizador da consulta que a define ($Cost(Clause_{VM})$).

A segunda parte do custo de criação é considerar que haverá o custo de E/S para armazenar os dados. Em [Monteiro08, Morelli06a, Salles04] estima-se o custo de criação de um índice como o dobro do número de páginas que ele pode ter (para representar o custo de entrada e saída). Assim, é necessário estimar a quantidade de páginas de uma visão materializada ($HyPg_{VM}$).

O número de páginas ocupado por uma tabela em um banco de dados é proporcional ao tamanho da tabela e ao tamanho da página de um determinado

[1]O tamanho de cada tipo é definido em seu manual como pode ser visualizado em [SQLServerColumn e OracleColumn]

SGBD. A seguir podemos ver os tamanhos das tabelas do banco de dados TPC-H e a quantidade de páginas que ocupam no SGBD SQL Server 2008. Na Tabela 1 estão especificados os tamanhos da tabela, a quantidade de páginas reservadas e o tamanho da tabela dividido pelo tamanho da página (Page -8KB) como considerado no SQL Server.

Tabela	Tamanho (KB)	Páginas ocupadas	Tamanho/Page
Customer	45896	5737	5737
Lineitem	1003104	125388	125388
Nation	16	2	2
Orders	181880	22735	22735
Part	44928	5616	5616
PartSupp	232952	29119	29119
Region	16	2	2
Supplier	2632	329	329

Tabela 1 - Tamanhos de tabelas versus quantidade de páginas

Foi verificado também que uma tabela com poucos dados ocupa no mínimo *k* páginas. Como uma visão materializada pode ser comparada a uma tabela, de tal forma que, em uma consulta se possa fazer uma busca na visão materializada, podemos estimar a quantidade de páginas de uma visão materializada pelo tamanho estimado da mesma, dividido pelo tamanho de uma página do SGBD.

Partindo dessa idéia é preciso estimar o tamanho de uma visão materializada. Sabemos que o tamanho máximo das tuplas da visão materializada hipotética ($SizeRow_{VM}$) pode ser calculado e que a quantidade de tuplas de uma visão materializada ($HyRow_{VM}$) pode ser estimada pelo otimizador. Portanto, o tamanho estimado da visão pode ser estimado por multiplicar o tamanho máximo da tupla pela quantidade de tuplas estimada.

Obtendo o tamanho estimado da visão materializada hipotética podemos obter a quantidade de páginas ocupadas dividindo o tamanho estimado pelo tamanho da página do SGBD alvo. Desta forma temos a métrica para estimar a quantidade de páginas de uma visão materializada hipotética: $HyVMPg = HyRow_{VM} * SizeRow_{VM} / tamanho\ da\ página$. Assim é possível definir uma métrica para estimar o custo de criação de uma visão materializada como: $CreationCost_{VM} = Cost(\text{Clause}_{VM}) + 2 * HyPg_{VM}$.

3.1.3.Materialização de Visões

Ao final da fase de predição, caso seja detectada a necessidade de se materializar visões, a fase de reação é iniciada. Neste caso, deve-se materializar o conjunto de visões cujos benefícios acumulados são maiores que os respectivos custos de criação.

Esta fase inicia-se quando o componente executor CE é chamado para materializar visões hipotéticas. É preciso ajustar algumas configurações do banco de dados para permitir a criação de uma visão materializada. O CE é responsável por montar os conjuntos de configurações necessárias e executá-los junto com a cláusula de criação da visão materializada.

3.2. Resumo do capítulo

O processo de criação automática deste trabalho é baseado no ciclo de observação, predição e reação. Este capítulo apresentou as tarefas desempenhadas por este trabalho em cada fase, e apresentou as heurísticas necessárias para criação de visão materializada.

As medidas necessárias para criação das heurísticas de seleção de visão materializada foram apresentadas neste capítulo. Algumas medidas foram propostas por este trabalho como uma varredura sequencial em uma visão que não existe fisicamente, estimar quantidade de páginas e tamanho de uma visão hipotética.

No próximo capítulo serão apresentados resultados experimentais da instanciação dessas heurísticas. Estes testes avaliam a eficácia das heurísticas propostas neste trabalho.