4
Uma Abordagem Não-Intrusiva para a Manutenção Automática do Projeto Físico

Neste capítulo, apresentaremos uma abordagem para a manutenção automática do projeto físico de bancos de dados relacionais. A estratégia proposta é completamente desacoplada do SGBD (não-intrusiva) e autônoma.

A seguir, mostramos como este capítulo está organizado. A seção 4.1 descreve em linhas gerais a abordagem proposta. Nas seções 4.2, 4.3 e 4.4, são descritas as fases de observação, predição e reação da abordagem proposta. A seção 4.5 propõe estratégias para assegurar a qualidade das configurações selecionadas. A subseção 4.5.1 traz o conceito de relevância e discute como utilizar esta definição na atividade de manutenção automática do projeto físico. Já a subseção 4.5.2 trata do conceito de estabilidade de uma configuração de projeto físico.

4.1
Visão Geral

A abordagem proposta neste trabalho segue o ciclo clássico de autosintonia: Observação, Predição e Reação, como descrito inicialmente em (Weikum94). Estas fases são continuamente executadas[1] com a finalidade de: monitorar o comportamento corrente do sistema; derivar decisões sobre o comportamento futuro; e, caso mudanças sejam necessárias, aplicar estas mudanças, alterando as propriedades e o comportamento do sistema.

De acordo com a estratégia adotada, consideramos as seguintes fases:

– **Observação:** nesta fase, a carga de trabalho submetida ao SGBD é capturada e armazenada em uma metabase local (*Local Metabase - LM*), por intermédio do agente *AWO* (*Agent for Workload Obtainment*). A metabase local (*LM*) e o agente *AWO* são descritos no Capítulo 6. Em seguida, cada tarefa capturada é analisada a fim de se determinar as melhores estruturas de acesso (por exemplo, índices, visões materializadas,

[1] O fato do ciclo ser continuamente executado não impõe grande sobrecarga ao SGBD, uma vez que a abordagem não-intrusiva proposta pode ser executada em uma máquina distinta daquela utilizada para hospedar o SGBD.

etc) e seus respectivos benefícios para carga de trabalho como um todo. Esta análise é realizada pelos agentes que compõem o núcleo (*core*) da arquitetura apresentada no Capítulo 6, a partir de heurísticas baseadas em regras (*Heuristic Set*). Caso, durante a análise de uma tarefa seja detectado o fato que uma determinada estrutura, que não existe fisicamente, pode, potencialmente, diminuir o tempo da consulta em questão, essa estrutura é criada como hipotética. Neste caso, esse estrutura existirá apenas na metabase local (*LM*). Além disso, acompanha-se a utilização das estruturas de acesso fisicamente existentes (reais) a fim de monitorar a sua utilidade e a necessidade de reorganização (ou recriação).

- **Predição:** esta fase busca obter uma configuração de estruturas de acesso ótima, levando em consideração as restrições de espaço físico. Vale ressaltar que esta nova configuração pode envolver tanto estruturas reais quanto hipotéticas.
- **Reação:** ao final da fase de predição, caso seja detectada a necessidade de mudanças na configuração corrente ou a existência de estruturas reais que necessitem ser recriadas (índices fragmentados, por exemplo) a fase de reação é iniciada. Assim, obtida a nova configuração do projeto físico, torna-se necessário: (i) criar fisicamente as estruturas hipotéticas que pertencem à esta configuração, mas que ainda não existem fisicamente; (ii) remover as estruturas reais que não pertencem à nova configuração; e, (iii) reorganizar as estruturas reais que pertencem à nova configuração mas que necessitam ser reorganizadas ou recriadas.

A seguir, cada uma dessas fases será discutida detalhadamente.

4.2 Fase de Observação: Monitoramento da Carga de Trabalho e Atualização do Benefício das Estruturas de Acesso

4.2.1 Idéia Básica

Nesta fase, através de consultas à metabase do SGBD, captura-se, periodicamente, a carga de trabalho (conjunto de tarefas, Definição 2) submetida ao sistema de banco de dados (utilizando-se para isso o agente *AWO*, descrito no Capítulo 6). A idéia é obter as expressões SQL executadas, juntamente com seus respectivos planos de execução e estimativas de custo. Estas informações são armazenadas na metabase local (*LM*, Capítulo 6).

Em seguida, analisa-se cada tarefa capturada com a finalidade de se identificar as estruturas mais adequadas e seus respectivos benefícios para carga de

trabalho como um todo. Esta análise é realizada pelos agentes que compõem o núcleo da arquitetura (*Core Agents*) apresentada no Capítulo 6. Estes agentes utilizam heurísticas baseadas em regras, as quais são periodicamente[2] executadas. O conjunto das heurísticas utilizadas é denominado *Heuristic Set* (Capítulo 6).

As informações armazenadas para cada tarefa permitem fazer inferências sobre as características e custos de planos de execução alternativos e equivalentes que poderiam ser obtidos caso variássemos o projeto físico, por exemplo, criando ou removendo determinadas estruturas de acesso (índices, v. g.). A equivalência entre dois planos de execução foi formalmente apresentada na Definição 10.

As estimativas de custos dos planos alternativos são obtidas utilizando-se o modelo de custos externo discutido no Apêndice A. Vale destacar que esta análise é realizada sem que seja necessário fazer chamadas adicionais ao otimizador[3] de consultas, podendo até mesmo ser executada em uma máquina distinta daquela utilizada para hospedar o SGBD.

A idéia básica consiste em substituir um determinado sub-plano p, pertencente ao plano de execução original, por um sub-plano alternativo e equivalente p', obtido mediante a utilização de estruturas hipotéticas, porém com custo menor. Um plano alternativo obtido a partir da utilização de estruturas hipotéticas é denominado plano hipotético (Definição 11). O processo de manipular um plano de execução original e obter planos hipotéticos equivalentes ao plano de execução original é denominado otimização hipotética (Definição 12).

Vale destacar a diferença entre otimização de consultas e otimização hipotética. Otimização de consulta é um processo executado pelo SGBD, no qual diferentes planos de execução equivalentes (para uma determinada consulta `q`) são analisados e um dentre esses planos é escolhido, a fim de ser utilizado durante a execução da consulta `q`. A escolha do plano de execução a ser utilizado pode basear-se nos custos de execução dos planos (otimização baseada em custos) ou na aplicação de regras (otimização baseada em heurísticas). Nesse sentido, a otimização de consultas busca encontrar um plano de execução com custo menor do que os demais planos equivalentes. Já a otimização hipotética parte de um plano de execução real e procura alterar este plano, com a utilização de estruturas hipotéticas, buscando assim obter um plano hipotético de custo inferior ao plano real original.

[2]A periodicidade com que as heurísticas são executadas pode ser parametrizada pelo DBA.

[3]Chamadas adicionais ao otimizador impõem sobrecarga ao SGBD.

A otimização hipotética baseia-se no fato de que o custo do sub-plano original p pode ser obtido a partir da metabase local (LM) e o custo do sub-plano alternativo p' pode ser calculado utilizando-se o modelo de custos externo (ECM), definido no Capítulo 6 e descrito no Apêndice A. Logo, pode-se estimar quanto o desempenho do plano de execução original pode ser melhorado (ou degradado) se substituíssemos um determinado sub-plano p por um outro sub-plano hipotético p'. A partir da diferença entre o custo do sub-plano original p e o custo do sub-plano hipotético p', é possível inferir o benefício que as estruturas hipotéticas utilizadas no sub-plano hipotético p' trariam para o processamento da consulta analisada.

4.2.2 Heurística Integrada para Seleção e Acompanhamento das Estruturas de Acesso

A seguir iremos comentar uma heurística integrada para a seleção e acompanhamento das estruturas de acesso que compõem o projeto físico de um banco de dados relacional. Vale destacar que a abordagem proposta gerencia tanto as estruturas reais quanto as hipotéticas. Assim, para cada estrutura de acesso, real ou hipotética, existe uma entrada na metabase local (LM). A diferença é que as estruturas hipotéticas existem somente na metabase local, enquanto as estruturas reais existem fisicamente. O acompanhamento destas estruturas, reais e hipotéticas, baseia-se no conceito de "benefício" (Cos02).

A utilização do conceito de "benefícios" foi inicialmente proposta em (Cos02), sendo implementada em (Sal04, Cha04, Morelli06a, Luh07). A idéia de "benefício" consiste em acompanhar as estruturas de acesso, atribuindo-lhes benefícios à medida que estas contribuam (no caso de estruturas reais) ou que potencialmente possam contribuir (no caso de estruturas hipotéticas) de forma positiva (ou seja, ajudem a diminuir o custo de execução) nos comandos executados pelo SGBD. Assim, o benefício de uma determinada estrutura de acesso é incrementado quando detectado que a existência desta estrutura reduz ou poderia reduzir o custo de execução de um comando SQL. Mais precisamente, o benefício de uma estrutura é quantificado como o número de páginas que deixariam de ser transferidas do disco para a memória principal caso a estrutura fosse utilizada no processamento de um determinado comando SQL.

Na abordagem proposta, verifica-se periodicamente[4] se novas tarefas foram capturadas e adicionadas na metabase local. Em caso afirmativo, uma

[4]Observe que esta periodicidade pode ser implementada como um parâmetro a ser definido pelo DBA.

heurística genérica baseada em regras, denominada *HISAE* (Heurística Integrada para Seleção e Acompanhamento de Estruturas de Acesso), é executada. Vale ressaltar que a instanciação desta heurística depende dos tipos de estruturas de acesso que se pretende utilizar. No próximo capítulo, discutiremos uma instanciação da heurística *HISAE*, denominada heurística HISAI (Heurística Integrada para Seleção e Acompanhamento de Índices), aplicada ao problema da manutenção automática das estruturas de índices. A heurística HISAI baseia-se no conceito de "benefício" e nas idéias discutidas nesta seção.

É importante observar que as operações de atualização ("update") também necessitam ser analisadas. Uma operação de atualização sobre uma determinada tabela t pode implicar na necessidade de atualizar as estruturas de índices, e visões materializadas definidas sobre t. Neste caso, o custo de atualização da estrutura de índice ou mesmo de uma visão pode ser considerado como um "malefício" (ou seja, um benefício negativo).

4.2.3 Exemplo de Aplicação

Seja a consulta apresentada na Figura 4.1[5]:

```
select *
from lineitem l, orders o
where l.l_orderkey = o.o_orderkey
  and l.l_partkey = 10
  and o.o_orderkey > 3000
```

Figura 4.1: Cláusula SQL Exemplo 1.

Considere que nenhuma estrutura de índice tenha sido criada no sistema de banco de dados até este momento. Neste caso, a Figura 4.2 mostra um possível plano de execução para a consulta em questão (Figura 4.1).

Assuma agora que desejamos inferir o que aconteceria se criássemos um novo índice secundário $i_1 = (l\_partkey)$ na tabela lineitem. Neste caso, o sub-plano p composto pela operação "Seq Scan on Lineitem" poderia ser substituído por um sub-plano alternativo $p^{'}$ composto por uma operação "Index Scan" sobre a coluna $i_1.l\_partkey$ (figura 4.3).

Seja EC_{SS} a estimativa de custo da operação "Seq Scan" [6] e $P_{lineitem}$ o número de páginas da tabela lineitem. Podemos estimar o custo do sub-plano p como sendo:

[5] As tabelas utilizadas pertencem ao *benchmark* TPC-H e os planos foram extraídos do PostgreSQL.

[6] As estimativas de custo utilizam o modelo de custos externo definido no Apêndice A.

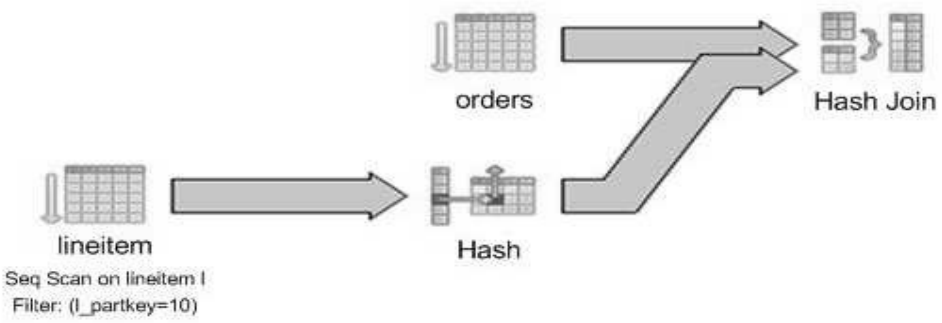

Figura 4.2: Plano de Execução da Cláusula SQL Exemplo 1.

$$EC_p = EC_{FS} = P_{lineitem}$$

Seja $EC_{S_{IS}}$ a estimativa de custo de uma operação de seleção utilizando um índice secundário ("Index Scan"), HT_{i_1} a altura do índice i_1 (árvore B^+) e $FS(l\_partkey, lineitem)$ o número médio de *tuplas* que satisfazem uma condição de igualdade sobre o atributo l_partkey na relação lineitem. O custo do sub-plano $p^{'}$ pode ser calculado como:

$$EC_{p'} = EC_{S_{IS}} = HT_{i_1} + FS(l\_partkey, lineitem)$$

Por intermédio de consultas à metabase do SGBD utilizado, podemos obter uma estimativa dos valores de: $P_{lineitem}$, HT_{I_1} e $FS(l\_partkey, lineitem)$. Neste caso particular, os valores obtidos foram: $P_{lineitem} = 132273$, $HT_{I_1} = 3$ e $FS(l\_partkey, lineitem) = 27$. Daí temos que, $EC_p = 132273$ e $EC_{p'} = 3 + 27 = 30$. Logo, podemos concluir que o plano alternativo apresenta uma economia de, no mínimo, $EC_p - EC_{p'} = 132273 - 30 = 132243$ $E/S's$, sendo, portanto, mais eficiente do que o plano original. No Exemplo 1, $i_1 = (l\_partkey)$ é o melhor índice secundário para otimizar o sub-plano p.

Desta maneira, utilizando esta estratégia iterativamente, podemos obter um plano de execução mais eficiente, sem termos que gerar todos os planos de execução possíveis para a consulta analisada. Observe que isto será alcançado, substituindo cada sub-plano do plano original por um sub-plano alternativo que seja o mais eficiente possível. Logicamente, não se pode afirmar que o plano de execução alternativo gerado por este processo é o plano ótimo, dado um conjunto de índices hipotéticos. Em vez disso, exploramos um conjunto

limitado de planos alternativos sem que seja necessário otimizar a consulta pela segunda vez (considerando a utilização dos índices hipotéticos). O custo do plano de execução resultante é portanto uma aproximação do plano global ótimo que seria gerado pelo otimizador de consultas caso os índices hipotéticos existissem fisicamente.

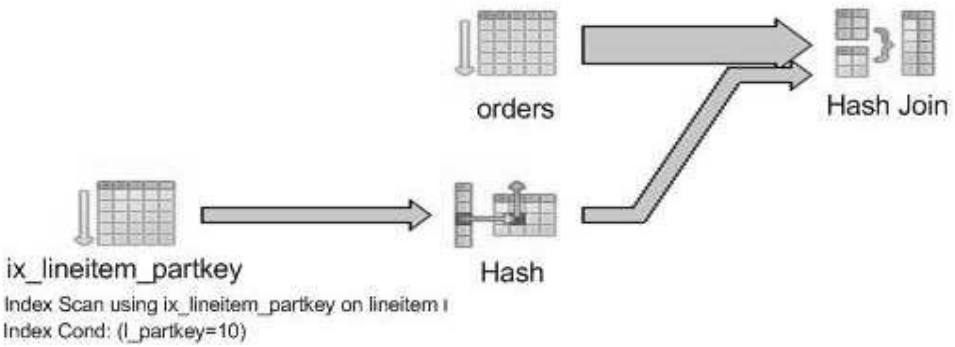

Figura 4.3: Plano de Execução da Cláusula SQL Exemplo 1 Considerando a Existência do Índice Secundário i_1.

Considere agora que tencionamos inferir o que aconteceria se criássemos um índice primário $i_2 = (l\_partkey)$ na tabela lineitem no lugar do índice secundário i_1. Neste caso, o sub-plano p composto pela operação "Seq Scan on Lineitem" poderia ser substituído por um sub-plano alternativo p'' composto por uma operação "Index Scan" sobre a coluna $i_2.l\_partkey$ (figura 4.4). Observe que, neste caso, a operação "Index Scan" utiliza um índice primário em vez de um índice secundário.

O custo do sub-plano p já foi estimado anteriormente como sendo:

$$EC_p = P_{lineitem}$$

Seja $F_{lineitem}$ o fator de página da relação lineitem. O custo do sub-plano p'' pode então ser calculado como:

$$EC_{p''} = EC_{S_{IP}} = HT_{i_2} + \lceil FS(l\_partkey, lineitem)/F_{lineitem} \rceil$$

A estimativa dos valores de $P_{lineitem}$, HT_{i_2} e $FS(l\_partkey, lineitem)$ já foi mencionada anteriormente. Podemos estimar o valor de $F_{lineitem}$ como sendo: $F_{lineitem} = \lfloor N_{lineitem}/P_{lineitem} \rfloor$. A partir da metabase do SGBD,

pode-se descobrir que $N_{lineitem} = 6001215$ e que $P_{lineitem} = 132273$. Logo, $F_{lineitem} = 45$.

Daí temos que $EC_p = 132273$ e $EC_{p'} = 3 + \lceil 27/45 \rceil = 4$.

Desta forma, podemos concluir que o plano alternativo produzido com base na permutação de p por p'', o qual utiliza o índice primário i_2, apresenta uma economia de $EC_p - EC_{p'} = 132273 - 4 = 132269$ $E/S's$, sendo, portanto, mais eficiente do que o plano original e mais eficiente do que o plano alternativo gerado substituindo-se p por p', o qual utiliza o índice secundário i_1.

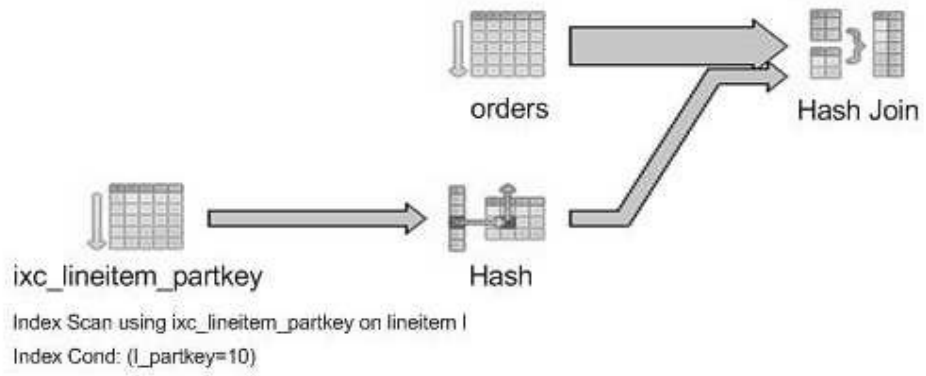

Figura 4.4: Plano de Execução da Cláusula SQL Exemplo 1 Considerando a Existência do Índice Primário i_2.

Devemos observar, entretanto, que cada tabela pode ter apenas um índice primário, restrição esta que não se aplica aos índices secundários. Além disso, em muitos SGBDs, o índice primário obrigatoriamente deve coincidir com a chave primária. Assim, a escolha do índice primário deve ser bastante criteriosa. No próximo capítulo, discutiremos como tratar esta restrição.

Como segundo exemplo, observe a consulta ilustrada na Figura 4.5:

```
select l_orderkey
from lineitem
```

Figura 4.5: Cláusula SQL Exemplo 2.

Considere que nenhuma estrutura de índice foi criada no sistema de banco de dados. Neste caso, a figura 4.6 mostra o plano de execução[7] originado para a consulta em questão (figura 4.5).

[7]As Figuras 4.6 e 4.7 foram obtidas utilizando-se o SQL Server 2005.

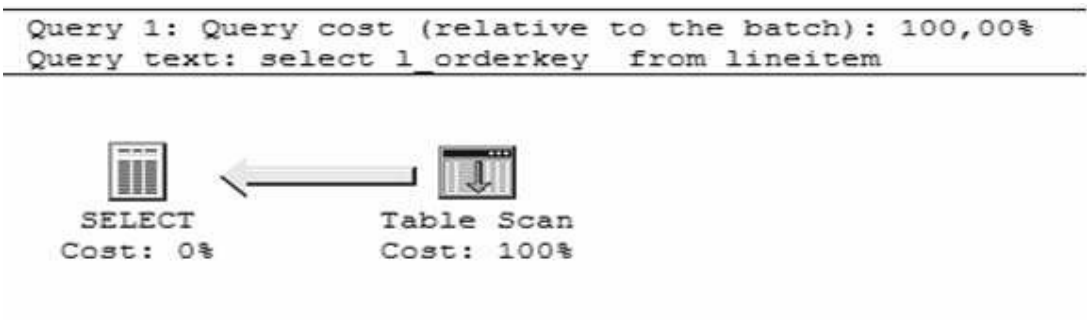

Figura 4.6: Plano de Execução da Cláusula SQL Exemplo 2.

Assuma agora que desejamos inferir o que aconteceria se criássemos um novo índice secundário $i_3 = (l\_orderkey)$ na tabela lineitem. Neste caso, o sub-plano p composto pela operação "Table Scan on Lineitem" poderia ser substituído por um sub-plano alternativo $p^{'}$ composto por uma operação "Index Scan" sobre a coluna $i_3.l\_orderkey$ (Figura 4.7).

O custo do sub-plano p já foi estimado anteriormente como sendo:

$EC_p = P_{lineitem}$

Seja HT_{i_3} a altura do índice i_3 (árvore B^+) e $P_{NF_{i_3}}$ o número de páginas que armazenam os nós folha da estrutura de índice i_3. O custo do sub-plano $p^{'}$ pode ser calculado como:

$EC_{p'} = HT_{i_3} + P_{NF_{i_3}}$

Mediante consultas à metabase do SGBD utilizado, podemos inferir que: $P_{lineitem} = 132273$ e que $HT_{i_3} = 3$. O valor de $P_{NF_{i_3}}$ pode ser estimado da seguinte forma: $P_{NF_{i_3}} \geq \lceil N_{lineitem}/O \rceil$. Sabemos que $N_{lineitem} = 6001215$ e que, no PostgreSQL, por exemplo, tipicamente $O = 8192$. Logo, $P_{NF_{i_3}} \geq 8800$.

Daí, temos que, $EC_p = 132273$ e $EC_{p'} = 3 + 8800 = 8803$.

Assim, podemos concluir que o plano alternativo apresenta uma economia de $EC_p - EC_{p'} = 132273 - 8803 = 123470E/S's$, sendo, portanto, mais eficiente do que o plano original. Observe que $i_3 = (l\_orderkey)$ é o melhor índice secundário para otimizar o sub-plano p. Vale ressaltar ainda que, neste caso (figura 4.5), a criação de um índice primário sobre o atributo $l\_orderkey$ não altera a estimativa de custo para o sub-plano $p^{'}$, uma vez que, independentemente do índice ser primário ou secundário, será necessário percorrer todos os nós folha do índice.

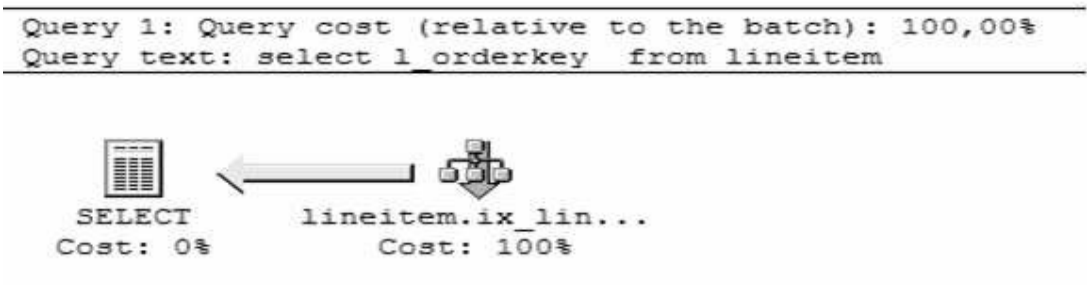

Figura 4.7: Plano de Execução da Cláusula SQL Exemplo 2 Considerando a Existência do Índice i_3.

4.3 Fase de Predição: Seleção *On-Line* de Estruturas de Acesso

4.3.1 Idéia Básica

O problema a ser resolvido nesta fase consiste em obter uma configuração de projeto físico ótima, levando em consideração as restrições de espaço físico. Para isso, utiliza-se as informações coletadas durante a fase de observação, as quais foram armazenadas na metabase local (definida no Capítulo 6).

Esta fase é iniciada sempre que o benefício acumulado de uma estrutura hipotética k ultrapassar um valor limite, o qual pode ser parametrizado pelo DBA. Neste sentido, duas estratégias são possíveis: agressiva e conservadora. Na estratégia agressiva, a fase de predição será iniciada sempre que o benefício acumulado de uma estrutura hipotética k qualquer ultrapassar o seu custo estimado de criação ($BA_k > EC_{C_k}$)[8], indicando que a existência desta estrutura contribuiria para diminuir o custo de processamento da carga de trabalho submetida ao SGBD, ou quando o benefício acumulado de uma estrutura real k' alcança um valor negativo, indicando que a existência de k' não está contribuindo para diminuir o custo de processamento da carga de trabalho. Já na estratégia conservadora, a fase de predição será iniciada sempre que o benefício acumulado de uma estrutura hipotética k qualquer ultrapassar o seu custo estimado de criação ($BA_k > EC_{C_k}$) ou quando o benefício acumulado de uma estrutura real k' alcança um valor negativo que supera, em módulo, o seu custo de criação ($BA_{k'} < 0$ e $|BA_{k'}| > EC_{C_k}$).

[8]O custo estimado de criação depende do tipo da estrutura de acesso.

4.3.2 Um Algoritmo Guloso

A seguir, apresentamos um algoritmo guloso para a seleção *on-thy-fly* de estruturas de acesso. Neste algoritmo, a variável E representa o conjunto das estruturas de acesso (incluindo estruturas hipotéticas e reais). Para cada estrutura $e \in E$, iremos utilizar: BA_e (benefício acumulado da estrutura e, Definição 7), P_e (número de páginas utilizadas para armazenar a estrutura e) e $state(e)$ (função que indica se a estrutura é hipotética ou real). Além disso, definimos *storage_constraint* como o espaço disponível para a materialização das estruturas de acesso. A solução adotada utiliza um algoritmo guloso adaptado de (Luh07). Observe que todas as estruturas (reais e hipotéticas) são ordenadas de acordo com o seu benefício relativo (Definição 8).

Ao ser executado, o algoritmo apresentado na figura 4.8 produz como saída uma nova configuração $\bar{C}$, ou seja, um novo conjunto de estruturas de acesso que maximiza o benefício para a carga de trabalho submetida ao SGBD, levando em consideração as restrições de espaço físico. Com a finalidade de evitar que as mesmas estruturas sejam freqüentemente adicionadas e removidas, uma nova configuração $\bar{C}$ somente substituirá a configuração atual C se o benefício total de $\bar{C}$ for superior ao benefício total de C multiplicado por um fator constante k (definido pelo DBA e tipicamente > 1.0, como sugerido em (Luh07)). Seja B_C o benefício total da configuração corrente C. Podemos definir a variável "*threshold*" da seguinte forma:

$$\text{threshold} = B_C \times k,$$

onde, B_C representa o benefício total da configuração C (Definição 17).

4.3.3 Um Algoritmo Baseado em Programação Dinâmica

A Figura 4.8 descreve um algoritmo guloso para a seleção de uma configuração de estruturas de acesso. Este algoritmo, apesar de produzir boas soluções, não resolve o problema da mochila booleana de forma exata (conforme discutido na Seção 2.8).

Para ilustrar esta asserção, considere o seguinte exemplo:

Seja W a capacidade da mochila, ou seja, a quantidade de espaço em disco disponível para a materialização das estruturas de acesso. Assuma que $W = 50$. Considere a existência de três estruturas candidatas: e_1, e_2 e e_3. Vamos assumir que: $BA_{e_1} = 840$, $BA_{e_2} = 600$, $BA_{e_3} = 400$, $P_{e_1} = 40$,

```
E'[1···n] ← ordenar(E) através do benefício relativo
C̄ ← φ
espaço_disponível ← storage_constraint
benefício_total ← 0
Para todo k ← 1···n faça
   Se espaço_disponível - P_E'[k] > 0 então
      C̄ ← C̄ ⋃ E'[k]
      espaço_disponível ← espaço_disponível - P_E'[k]
      benefício_total ← benefício_total + BA_E'[k]
   Fim Se
Fim Para
Se benefício_total < threshold então
   C̄ ← C
Fim Se
retorne C̄
```

Figura 4.8: Algoritmo Guloso para Seleção de uma Configuração de Estruturas de Acesso.

$P_{e_2} = 30$ e $P_{e_3} = 20$. Daí, podemos calcular o benefício relativo: $BR_{e_1} = 21$, $BR_{e_2} = 20$, $BR_{e_3} = 20$. Desta forma, se ordenamos as estruturas de acesso pelo benefício relativo, teremos: e_1, e_2 e e_3, nesta ordem. Logo, o algoritmo guloso irá, primeiramente, colocar a estrutura e_1 na mochila. Em seguida, o espaço disponível na mochila passa a ser $W - P_{e_1} = 50 - 40 = 10$. Observe que, neste caso, nenhuma das estruturas restantes (e_1 e e_2) serão adicionados na mochila, uma vez que o espaço disponível é insuficiente. Logo, a configuração encontrada ($\bar{C}$) é formada somente pela estrutura e_1. Note que o benefício total de $\bar{C}$ é igual a 40, contudo, o algoritmo guloso não foi capaz de maximizar o benefício relativo, pois existe uma configuração alternativa $\bar{C}'$, formada pelas estruturas e_1 e e_2 cujo benefício total é igual a 50, sendo, portanto, melhor do que $\bar{C}$.

Esta seção apresenta e discute um algoritmo exato baseado em programação dinâmica. A versão booleana do problema da mochila pode ser resolvida por programação dinâmica, uma técnica que pode ser definida como "recursão com apoio de uma tabela", o qual possui complexidade $O(nW)$, onde n é o número de objetos que podem ser adicionados na mochila, ou seja, o número de estruturas candidatas, e W é o tamanho da mochila, ou seja, a quantidade de espaço em disco disponível para a criação de estruturas de acesso. Vale ressaltar que este algoritmo impõe a seguinte restrição: os pesos dos objetos (ou seja, tamanho das estruturas de acesso) devem ser todos inteiros, não negativos. Observe que o nosso problema particular (ISP) [9] atende essa restrição uma vez que o tamanho das estruturas de acesso é medido em

[9] ISP - Index Selection Problem (Seção 5.2.1).

números de páginas (blocos), os quais são sempre inteiros não negativos.

Sabemos que o problema da mochila é NP-Completo. Logo, não se conhece uma solução exata em tempo polinomial. Apesar da complexidade do algoritmo de programação dinâmica ser $O(nW)$, essa complexidade não é satisfatória. Suponha, por exemplo, que a escala de medida de pesos seja multiplica por 1000. Neste caso, o problema continua essencialmente o mesmo, mas o algoritmo consome 1000 vezes mais tempo. Assim, o algoritmo é denominado pseudo-polinomial, ou seja, na prática, se comporta como exponencial, pois $O(nW) = O(n2^{lg(W)})$ e $lg(W)$ é a medida correta do "tamanho" do número W. Portanto, a complexidade do algoritmo é influenciada pela escala de medidas utilizada para W. Desta forma, estratégias que reduzam o "tamanho" de W, ou seja, alterem a unidade com que W é medido, podem reduzir substancialmente a complexidade do algoritmo. Por exemplo, podemos utilizar como unidade de medida para W uma página de dados, ou o número de páginas da menor estrutura de acesso.

A programação dinâmica procura transformar recursão em iteração com o apoio de uma tabela. A idéia básica do algoritmo consiste em guardar em uma tabela t as soluções dos subproblemas do problema principal. A tabela t é definida da seguinte forma:

$t[i, Y]$ é o valor máximo da expressão $x[1..i] \times BA[1..i]$ sujeita à restrição $x[1..i] \times P[1..i] \leq Y$,

onde, $x[1..i]$ indica as estruturas que participam da solução (configuração) de um subproblema.

Assim, se $x[1] = 1$ a estrutura de acesso e_1 pertence à solução encontrada para o subproblema. Já se $x[1] = 0$ a estrutura e_1 não pertence à solução encontrada para o subproblema. A variável Y representa a restrição de espaço, logo pode receber valores entre 0 e W. O vetor $BA[1..i]$ armazena os benefícios acumulados das estruturas candidatas. Assim, $BA[1] = BA_{e_1}$. O vetor $P[1..i]$ armazena os tamanhos, números de páginas, das estruturas candidatas.

```
Para Y ← 0 ··· W faça
  t[0, Y] ← 0
  Para i ← 1 ··· n faça
    A ← t[i − 1, Y]
    Se P[i] > Y então
      B ← 0
    Se não
      B ← t[i − 1, Y − P[i]] + BA[i]
    Fim Se
    t[i, Y] ← max(A, B)
  Fim Para
Fim Para
Y ← W
Para i ← n ··· 1 faça
  Se t[i, Y] = t[i − 1, Y] então
    x[i] ← 0
  Se não
    x[i] ← 1
    Y ← Y − P[i]
  Fim Se
Fim Para
Retorne x
```

Figura 4.9: Algoritmo Baseado em Programação Dinâmica para Seleção de uma Configuração de Estruturas de Acesso.

4.4 Fase de Reação: Criação, Remoção e Reorganização das Estruturas de Acesso

Ao final da fase de predição, caso seja detectada a necessidade de mudanças na configuração de estruturas de acesso ($\bar{C} \neq C$, ou seja, o benefício total da configiração gerada $\bar{C}$ é superior ao benefício da configuração corrente C multiplicado pela constante k, $B_{\bar{C}} > B_C \times k$), a fase de reação é iniciada.

Neste caso, deve-se substituir a configuração atual C por uma uma nova configuração $\bar{C}$. Assim, se uma determinada estrutura $k \in C$ e $k \notin \bar{C}$, então k deve ser removido (inclusive do catálogo do SGBD). Todavia, a estrutura k continua existindo na metabase local (LM), porém, seu estado é alterado de real para hipotético e $BA_k = 0$. Logo, nada impede que, futuramente, esta estrutura seja novamente materializada. Note que o custo de excluir uma estrutura pode ser desprezado, uma vez que esta tarefa consiste basicamente em liberar espaço em disco, remover uma entrada do catálogo do SGBD, e atualizar o estado e do benefício acumulado desta estrutura na metabase local. Por outro lado, toda estrutura k tal que $k \in \bar{C}$ e $k \notin C$ deve ser materializada, ou seja, fisicamente criada.

A criação imediata da estrutura de acesso k pode, no entanto, degradar

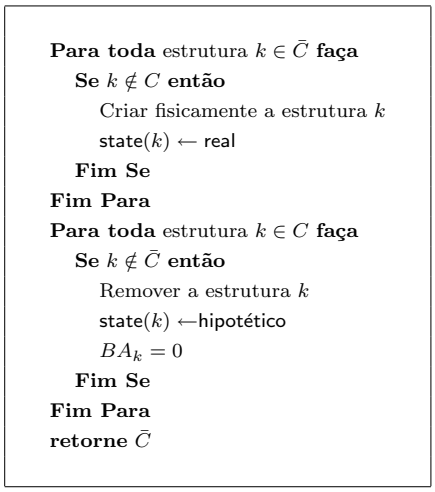

Figura 4.10: Algoritmo para Atualização da Configuração de Estruturas de Acesso.

o desempenho do sistema de banco de dados, caso este se encontre em nível de utilização elevado. As principais alternativas para este problema consistem em: 1) postergar a criação da estrutura para um momento no futuro, quando o nível de utilização do SBD esteja baixo e 2) integrar o processo de criação das estruturas de acesso ao processador de consultas, efetuando a construção destas estruturas durante a execução de uma operação de "*Table Scan*", por exemplo (como sugerido em (Luh07)). A primeira opção esbarra na dificuldade de prever, de forma automática, qual o próximo momento em que o SGBD estará com um nível de utilização baixo. Já a segunda opção requer alterações no processador de consultas, sendo assim uma solução intrusiva. Neste sentido, optamos por postergar a criação das estruturas de acesso para um momento no futuro, quando o nível de utilização do SBD esteja baixo. Para isso, propomos a utilização de um agente de *software*, denominado *Scheduler Agente (SA)*[10]. No entanto, a descrição detalhada desta solução está fora do escopo deste trabalho.

4.5 Qualidade das Configurações Obtidas

Nesta seção, discutiremos como avaliar e assegurar a qualidade das configurações de projeto físico selecionadas. Neste sentido, para mensurar a qualidade de uma configuração, propomos a utilização das seguintes métricas: relevância de uma estrutura de acesso, generalidade de uma estrutura de acesso e volatilidade de uma configuração de projeto físico.

[10]O agente *Scheduler Agente (SA)* é descrito no Capítulo 6.

4.5.1 Relevância e Generalidade

Na discussão realizada até aqui, existe uma suposição básica importante. Em princípio, consideramos todas as transações envolvidas igualmente importantes para os usuários do sistema. Isto não é sempre verdadeiro na prática. Neste caso, a criação de uma estrutura de acesso (índice, visão materializada, etc) que acelera determinada consulta pode ser obrigatória, mesmo que não traga um benefício para a vazão do sistema como um todo. Alguns exemplos de trabalhos na área de auto-sintonia que levam em conta restrições de objetivos de desempenho por classes de transações são (Brown94) e (Martin02).

A abordagem "não-intrusiva" facilita a utilização do conceito de relevância na solução do problema da manutenção automática do projeto físico de bancos de dados relacionais. Nesta abordagem, as consultas submetidas ao SGBD são capturadas e armazenadas em uma metabase local. Isto possibilita que o DBA, por meio de um *wizard* ou acessando diretamente a metabase, enriqueça as informações armazenadas com anotações que indiquem a relevância das consultas capturadas. Propomos que, para cada tarefa armazenada, o DBA forneça um valor entre 1 e 10, o qual representa a relevância da consulta (R_q). O valor 1 indica uma relevância mínima e o valor 10 uma relevância máxima (a consulta do presidente, por exemplo). Desta forma, pode-se continuar utilizando o algoritmo para a seleção de estruturas de acesso, como anteriormente discutido, sendo necessário apenas reescrever o cálculo do benefício acumulado:

$$BA_e = \sum_{k=1}^{NQ_e} B_{e,q_k} \times \frac{R_{q_k}}{10},$$

onde: B_{e,q_k} representa o benefício da estrutura e para uma consulta específica q_k, NQ_e é o número de comandos SQL onde a estrutura e foi ou poderia ter sido utilizada e R_{q_k} representa a relevância da consulta q_k.

Suponha a existência de duas estruturas de acesso e_1 e e_2 e dois comandos SQL q_1 e q_2. Considere que $R_{q_1} = 1$, $R_{q_2} = 10$, $B_{e_1,q_1} = 100$, $B_{e_1,q_2} = 0$, $B_{e_2,q_1} = 0$, $B_{e_2,q_2} = 60$. Observe que, se considerássemos somente o benefício absoluto, a estrutura e_1 seria considerada melhor do que a estrutura e_2; porém, a consulta q_2 é mais relevante do que a consulta q_1. Neste caso, considerando a relevância das consultas, o benefício acumulado da estrutura e_2 ($BA_{e_2} = 60$) é maior do que o benefício acumulado de e_1 ($BA_{e_1} = 10$). Logo, a estrutura e_2 seria mais indicada do que a estrutura e_1.

Existe ainda uma segunda suposição importante assumida até aqui.

Temos considerado que a carga de trabalho que será submetida ao SGBD no futuro (denominada carga de produção) será semelhante à carga capturada e utilizada para conduzir o processo de seleção de uma configuração de estruturas de acesso (denominada carga de treinamento), pois, caso contrário, as estruturas selecionadas podem não ser adequadas para a nova carga de trabalho. Na prática, no entanto, é possível que a carga de produção seja diferente da carga de treinamento (uma vez que esta foi coletada no passado) em diferentes aspectos: parâmetros, freqüência, ordem, sintaxe, etc. Desta forma, é essencial que uma solução automática para o problema da manutenção do projeto físico considere esta possibilidade.

Neste sentido, propomos uma métrica para representar o conceito de "generalidade", ou seja, a qualidade de uma estrutura para a carga de treinamento como um todo. A métrica proposta (G_e) consiste em um valor entre 0 e 1, onde 0 significa que a estrutura possui baixa generalidade e 1 aponta que a estrutura tem generalidade máxima. A métrica proposta é definida como:

$$G_e = \frac{NQe}{NQ_{Wt}},$$

onde NQe é o número de tarefas nas quais a estrutura e foi ou poderia ter sido utilizado e NQ_{Wt} é o número total de tarefas da carga de treinamento. Assim, quanto maior a generalidade de uma estrutura de acesso e, maior o número de tarefas onde a estrutura e foi ou poderia ter sido utilizada.

A fim de ilustrar a utilização deste conceito, suponha a existência de duas estruturas de acesso e_1 e e_2. Considere que $BA_{e_1} = 100$, $BA_{e_2} = 60$, $P_{e_1} = 10$, $P_{e_2} = 10$, $G_{e_1} = 0,5$ e $G_{e_2} = 1$. Observe que, neste caso, a estrutura e_2 foi ou poderia ter sido utilizada em todos os comandos SQL que compõem a carga de trabalho. Já a estrutura e_1 foi ou poderia ter sido utilizada em metade dos comandos SQL que compõem a carga de trabalho. Assim, temos que: $BR_{e_1} = 5$ e $BR_{e_2} = 10$. Vale ressaltar que, se observássemos apenas o benefício acumulado, a estrutura e_1 seria considerada mais indicada do que a estrutura e_2. Contudo, a estrutura e_2 é mais genérica do que a estrutura e_1 e apresenta um benefício relativo maior.

Observe que, para utilizar o conceito de "generalidade", necessitamos apenas reescrever o benefício relativo, o qual pode ser reescrito da seguinte forma:

$$BR_e = \frac{BA_e \times G_e}{P_e}.$$

4.5.2 Volatilidade das Configurações

A volatilidade de uma configuração de estruturas de acesso também é uma questão relevante. Configurações instáveis conduzem a freqüentes criações e remoções das mesmas estruturas. Neste caso, uma determinada estrutura e pode ser eliminada em uma fase de predição k para logo em seguida ser criada novamente na fase de predição $k+1$. Logicamente, este fato gera sobrecarga, podendo, até mesmo, levar à perda de desempenho (Luh07).

Dois aspectos podem influenciar a geração de configurações instáveis: a escolha do momento de se iniciar a fase de predição e a desatualização das estatísticas. A escolha do momento de iniciar a fase de predição é uma questão importante. Executar a fase de predição sempre que uma consulta for submetida ao SGBD (como proposto em (Kai04, Sal04, Sal05, Cos05, Morelli06a, Lif06)) pode ocasionar uma sobrecarga desnecessária e resultar em configurações instáveis. Já as estatísticas coletadas remotamente (ou seja, benefícios calculados e acumulados há bastante tempo) podem não mais representar de forma satisfatória o estado atual do sistema. A utilização de estatísticas desatualizadas no processo de predição de índices também pode resultar em configurações instáveis e levar à perda de desempenho.

Para solucionar estes problemas, foi proposto em (Kai04, Luh07) o conceito de época, o qual delimita a duração de um período de observação e a validade das estatísticas coletadas (benefícios). A duração de uma época pode ser definida em termos de tempo, número de consultas, valor do benefício de uma estrutura, ou, ainda, como utilizado na implementação descrita em (Luh07), do número máximo de recomendações (ou seja, quantidade de benefícios atribuídos) para uma mesma estrutura de acesso. Além disso, esta abordagem sugere dar um peso maior para os benefícios atribuídos pelas recomendações mais recentes. Assim, os benefícios atribuídos mais recentemente terão peso maior do que os benefícios calculados mais remotamente. Esta característica pode ser implementada da seguinte forma: para cada recomendação, ou seja, para cada benefício calculado, atribui-se um marcador de tempo (*timestamp*) monotonicamente crescente, $ts_1, ts_2, ..., ts_k$. Assim, uma recomendação com *timestamp* ts_2 é posterior a uma recomendação com *timestamp* ts_1 se $ts_1 < ts_2$. Seja ts_E *timestamp* referente ao encerramento de uma época, o benefício de uma determinada estrutura e pode ser calculada da seguinte forma:

$$BA(e) = \sum_{j=1}^{k} \frac{B(e, ts_j)}{ts_E - ts_j}.$$

A idéia básica da utilização do conceito de época consiste em atribuir peso maior aos benefícios calculados mais recentemente e peso menor aos benefícios calculados mais remotamente. Desta forma, busca-se privilegiar as estruturas que tenham sido recentemente utilizadas. A fim de ilustrar a utilização do conceito de época, considere a existência de duas estruturas de acesso e_1 e e_2. Suponha que a estrutura e_1 recebeu um benefício de valor 200 no *timestamp* ts_0 e que a estrutura e_2 recebeu um benefício de valor 100 no *timestamp* ts_{10}. Observe que o benefício acumulado absoluto da estrutura e_1 (200) é maior do que o benefício acumulado absoluto da estrutura e_2 (100). Considerando apenas o benefício acumulado absoluto a estrutura e_1 seria mais indicada do que a estrutura e_2. Contudo, assuma que a época corrente seja encerrada no *timestamp* ts_{10}. Neste caso, considerando a utilização do conceito de época, o benefício acumulado da estrutura e_1 ($BA_{e_1} = 20$) será maior do que benefício acumulado da estrutura e_2 ($BA_{e_2} = 100$). Logo, a estrutura e_2 seria mais indicada do que a estrutura e_1, uma vez que a estrutura e_2 não foi recentemente utilizada. Esta estratégia busca diminuir o benefício das estruturas não utilizadas recentemente.

Em (Luh07), propõem-se que o benefício e o *timestamp* de cada recomendação sejam armazenados em campos de tamanho fixo. Uma época termina quando o valor do *timestamp* gerado para uma determinada recomendação supera o tamanho utilizado para este campo.

Observamos que o conceito de época pode levar a uma baixa sobrecarga e a configurações mais estáveis (menos voláteis). Contudo, propomos algumas modificações:

A primeira alteração diz respeito à escolha do momento de se iniciar a fase de predição, ou seja, a duração de uma época. Diferentemente do que foi proposto em (Luh07) (a utilização de um campo de tamanho fixo) e da abordagem utilizada em (Kai04, Sal04, Sal05, Cos05, Morelli06a, Lif06) (captura de uma consulta submetida ao SGBD), propomos que a fase de predição seja iniciada sempre que o benefício acumulado de uma estrutura hipotética e qualquer ultrapassar o seu custo estimado de criação ($BA_e > EC_{C_e}$), indicando que a existência do índice contribuiria para diminuir o custo de processamento da carga de trabalho submetida ao SGBD, ou quando o benefício acumulado de uma estrutura real e' alcança um valor negativo que

supera, em módulo, o seu custo de criação ($BA_{e'} < 0$ e $|BA_{e'}| > EC_{C_e}$), indicando que a existência de e' não está contribuindo para diminuir o custo de processamento da carga de trabalho.

A segunda modificação consiste na utilização de um benefício inicial (B_{0_e}), ou seja, um benefício referente ao *timestamp* ts_0. Desta forma, cada estrutura e deve possuir um benefício inicial (B_{0_e}), o qual será calculado da seguinte forma:

- Inicialmente, $B_{0_e} = 0$, para toda estrutura e.
- Caso uma determinada estrutura e seja materializada, então $B_{0_e} = EC_{C_e}$, ou seja, o benefício inicial passa a ser igual ao custo de criação.
- Caso uma determinada estrutura e seja eliminada (deixa de ser real e volta ser hipotética) então $B_{0_e} = -EC_{C_e}$, ou seja, o benefício inicial passa a ser igual ao custo de criação com sinal negativo.
- Para as estruturas que nunca foram materializadas (ou seja, só existiram como hipotéticas), o benefício inicial é atualizado a cada conclusão de uma época. Esta atualização é feita da seguinte maneira:

$$B_{0_e} = -EC_{C_e} + EC_{C_e} \times \frac{N_{E_e}}{N_E},$$

onde N_E é o número total de épocas e N_{E_e} é o número de épocas nas quais a estrutura e foi utilizada, ou seja, obteve algum benefício.

Para ilustrar as idéias discutidas, considere que uma determinada estrutura de acesso e foi materializada. Neste momento, o benefício inicial de e corresponde ao custo de criação de e ($B_{0_e} = EC_{C_e}$). Logo, o benefício acumulado de e é incrementado de EC_{C_e}. Portanto, para que a estrutura e venha a ser removida, ela precisa receber benefícios negativos ("malefícios") que superem o seu benefício acumulado. Desta forma, busca-se aumentar o grau de dificuldade para remover uma estrutura, e, conseqüentemente, aumentar a estabilidade da configuração. Considere agora que uma determinada estrutura e foi removida (ou seja, deixou de ser uma estrutura real e voltou a ser uma estrutura hipotética). Neste momento, o benefício inicial de e corresponde em módulo ao custo de criação de e, só que com sinal negativo, ($B_{0_e} = -EC_{C_e}$). Logo, o benefício acumulado de e é decrementado de EC_{C_e}. Portanto, para que a estrutura e venha a ser criada (fisicamente) novamente, ela precisa receber benefícios

que superem o seu benefício acumulado. Desta forma, busca-se aumentar o grau de dificuldade para criar uma estrutura que já tenha sido removida, e, conseqüentemente, aumentar a estabilidade da configuração. Por fim, considere que uma estrutura hipotética e nunca foi fisicamente criada e nem removida. Assuma também que esta estrutura foi utilizada hipoteticamente (ou seja, utilizada em planos hipotéticos), durante todas as épocas. Desta forma, temos que $N_{E_e} = N_E$ e daí $B_{0_e} = 0$. Agora, assuma que a estrutura e tenha sido hipoteticamente utilizada em apenas metade das épocas. Neste caso, $N_{E_e} = \frac{N_E}{2}$. Logo, $B_{0_e} = -\frac{EC_{C_e}}{2}$. Assim, podemos concluir que o benefício inicial de uma estrutura hipotética que nunca foi materializada varia entre 0 (caso a estrutura e tenha sido hipoteticamente utilizada em todas as épocas) e $-EC_{C_e}$ (caso a estrutura e não tenha sido hipoteticamente utilizada em nenhuma época). Assim, aumenta-se o grau de dificuldade para a materialização de estruturas hipotéticas pouco utilizadas. Conseqüentemente, aumenta-se a estabilidade das configurações selecionadas.

O benefício inicial é utilizado para corrigir a distorção de equiparar estruturas hipotéticas, que nunca causaram malefícios, com outras que já tenham prejudicado comandos durante a fase como estruturas materializadas (reais) (Morelli06a). Procuramos com essa decisão obter configurações mais estáveis, uma vez que aumentamos o grau de exigência para que uma estrutura real seja removida, bem como para que uma estrutura hipotética, que já tenha sido anteriormente materializada, seja fisicamente criada. Além disso, privilegiam-se estruturas que sejam regularmente utilizadas. Assim, o benefício acumulado passa a ser calculado da seguinte forma:

$$BA(e) = B_{0_e} + \sum_{j=1}^{k} \frac{B(e, ts_j)}{ts_E - ts_j}.$$

4.6 Discussão

A seguir, discutiremos algumas importantes questões relacionadas ao funcionamento da abordagem proposta.

– Considere que o mecanismo proposto decidiu, de forma autônoma, ajustar o projeto físico, materializando uma determinada estrutura e (como um determinado índice, por exemplo). Assuma também que a estrutura

e, durante seu ciclo de vida como hipotética, recebeu um benefício por potencialmente diminuir o custo de execução de uma determinada consulta q. Neste caso, o plano hipotético gerado para a consulta q, que utilizava a estrutura e, apresentou custo de execução menor do que o plano de execução originalmente produzido, pelo otimizador de consultas do SGBD utilizado, para a consulta q. Logo, é razoável esperar que, caso a consulta q seja novamente executada, a estrutura e seja utilizada no plano de execução real gerado pelo otimizador de consultas para a consulta q. Esta é, no entanto, uma decisão do otimizador de consultas. O mecanismo proposto não assegura que a estrutura e seja obrigatoriamente utilizada no plano de execução gerado pelo otimizador para a consulta q. Desta forma, uma possível solução seria punir o índice i subtraindo do seu benefício acumulado o valor do benefício estimado de i para a cláusula SQL q. Para isso, deve-se armazenar na metabase local (LM), para cada índice i, a lista das cláusulas SQL nas quais o índice i foi ou poderia ter sido utilizado, juntamente com o benefício de i para cada uma destas cláusulas SQL. Vale ressaltar que este problema, em geral, não ocorre nas soluções intrusivas. Outra possibilidade para solucionar este problema consiste em dotar o otimizador hipotético de algum mecanismo de aprendizado.

– Poderíamos redefinir o problema da sintonia automática do projeto físico utilizando uma função de maximização multiobjetivo que levasse em consideração não somente a maximização do benefício de uma estrutura, mas também sua generalidade e relevância.

– Outra discussão importante consiste na premissa de que todos os SGBDs possuam em suas metabases os dados utilizados pela abordagem proposta: cláusulas SQL executadas, planos de consulta e estimativas de custo. Em (Monteiro07a), podemos observar que, apesar dos principais SGBDs comerciais (Oracle, SQL Server e PostgreSQL) apresentarem metabases específicas, estes disponibilizam, de alguma forma, os dados utilizados como entrada pela abordagem proposta.

– Uma das principais características da abordagem proposta consiste na sua capacidade de funcionar com qualquer SGBD. Poderíamos questionar, no entanto, como esta abordagem poderia ser genérica, uma vez que as tarefas (*triplas*) são específicas para cada SGBD (uma vez que as metabases são proprietárias e particulares). Neste sentido, vale destacar que a arquitetura apresentada no Capítulo 6 utiliza *drivers* que tornam transparente o acesso às metabases dos SGBDs, fornecendo para

o restante da arquitetura uma visão da carga de trabalho genérica (independente de SGBD).

– Os planos hipotéticos são produzidos a partir da manipulação de planos de execução reais, anteriormente executados, que foram capturados e armazenados na metabase local. Contudo, cada SGBD tem seu próprio formato para representar um plano de execução, além de operadores particulares. Desta forma, poderia ser imprescindível para o funcionamento da abordagem proposta uma transformação dos planos capturados (nas metabases dos diferentes SGBDs) para uma representação canônica (independente de SGBD). Contudo, mediante a simplicidade das heurísticas concebidas, esta representação canônica não foi necessária. Mas, provavelmente, para utilizar heurísticas mais elaboradas, serão necessários a concepção e utilização de planos de execução canônicos. Neste sentido, consideramos que definição (e utilização) de planos de execução canônicos está fora do escopo deste trabalho.
– Para reduzir ainda mais a sobrecarga da execução do otimizador hipotético, podemos evitar a execução das heurísticas quando as tabelas forem pequenas ou a seletividade da consulta for baixa, casos nos quais a existência ou não de uma estrutura de acesso (como um índice, por exemplo) não afeta de forma significativa o desempenho das consultas que utilizam estas tabelas.
– Espera-se que as soluções intrusivas sejam mais precisas do que a abordagem não-intrusiva e, conseqüentemente, cheguem mais rapidamente a uma configuração adequada para a carga de trabalho submetida ao SGBD.

4.7 Resumo do Capítulo

Neste capítulo, apresentamos uma abordagem não-intrusiva para a manutenção automática do projeto físico de bancos de dados relacionais. Esta abordagem é completamente desacoplada do código do SGBD, podendo ser utilizada com qualquer SGBD e executada sem intervenção humana. A manutenção do projeto físico é realizada mediante um conjunto de heurísticas baseadas em regras. Um conjunto de algoritmos (gulosos e baseados em programação dinâmica) para seleção e atualização de uma configuração de estruturas de acesso também foi apresentado neste capítulo. Além disso, definiu-se os conceitos de relevância e generalidade de uma estrutura de acesso, bem como o conceito de estabilidade de uma configuração, e discutiu-se como aplicar estes conceitos no processo de manutenção do projeto físico de bancos de dados.

O próximo capítulo discutirá como instanciar a abordagem proposta para solucionar problemas concretos referentes ao projeto físico de bancos de dados relacionais. Dois estudos de caso serão apresentados: uma instanciação da abordagem proposta para realizar a manutenção automática de estruturas de índices e a instanciação da abordagem proposta para efetuar a duplicação automática de estruturas físicas.