# 5
# Estudo de Caso: Instanciações da Abordagem de Sintonia Automática

Neste capítulo, discutiremos como a abordagem apresentada no capítulo anterior pode ser instanciada com a finalidade de possibilitar a manutenção automática e não-intrusiva de diferentes estruturas que compõem o projeto físico de bancos de dados relacionais. Particularmente, descreveremos como instanciar a abordagem proposta para realizar a manutenção automática de estruturas de índices e a manutenção automática de *clusters* alternativos.

A seguir, mostramos como este capítulo está organizado. A Seção 5.1 ilustra como a abordagem não-intrusiva foi instanciada para solucionar o problema da manutenção automática de índices. Já a Seção 5.2 discute como a abordagem não-intrusiva foi instanciada para solucionar o problema da manutenção automática de *clusters* alternativos.

## 5.1
## Estudo de Caso 1: Manutenção Automática e Não-Intrusiva das Estruturas de Índices

Esta seção apresenta a instanciação da abordagem não-intrusiva para a manutenção automática de índices.

### 5.1.1
### Preliminares

A manutenção automática de índices consiste em prover uma solução dinâmica para o já bastante estudado problema da seleção de índices (*Index Selection Problem (ISP)*) (Luh07). Na sua abordagem estática, o ISP pode ser descrito como uma simplificação de um caso especial do problema da mochila (Graefe00, Loh00), mais precisamente, o problema da mochila booleana.

Antes de definirmos formalmente o problema da sintonia automática de índices, algumas definições necessitam ser apresentadas.

Primeiramente, iremos definir uma estrutura de índice.

**Definição 20 (Chave de Busca de um Índice)** *Um índice $i_k$ definido sobre a relação $r$ possui como chave de busca (*$\mathsf{chave\_busca_{i_k}}$*) um subconjunto dos atributos de $r$, tal que:* $\mathsf{chave\_busca_{i_k}} \subseteq \{A_1, A_2, \cdots, A_n\}$.

A partir desta definição, pode-se classificar os índices em: índices simples e índices compostos. Um índice é simples quando sua chave de busca possui um único atributo e composto quando sua chave de busca possui dois ou mais atributos. A seguir, dados dois índices $i_k$ e $i_j$, definidos sobre uma mesma relação $r$, define-se quando um índice $i_k$ é um prefixo de um outro índice $i_j$.

**Definição 21 (Prefixo de um Índice)** *Sejam dois índices $i_k$ e $i_j$, definidos sobre uma mesma relação $r$, com chaves de busca* $\mathsf{chave\_busca_{i_k}}$ *e* $\mathsf{chave\_busca_{i_j}}$*, respectivamente. O índice $i_k$ é um prefixo de $i_j$, se:*

*(i)* $\mathsf{chave\_busca_{i_k}} \subset \mathsf{chave\_busca_{i_j}}$

*(ii)* $\forall p, q \in \mathsf{chave\_busca_{i_k}}, p <_{\mathsf{chave\_busca_{i_k}}} q \Leftrightarrow p <_{\mathsf{chave\_busca_{i_j}}} q$

*(iii)* $\forall q \in \mathsf{chave\_busca_{i_k}}, \forall p \in \mathsf{chave\_busca_{i_j}}, p <_{\mathsf{chave\_busca_{i_j}}} q \Rightarrow p \in \mathsf{chave\_busca_{i_k}}$

Neste trabalho, consideramos que uma estrutura de índice pode estar estar em dois estados distintos: real e hipotético. Além disso, utilizamos $IR$, $IH$ e $I$ para representar, respectivamente, o conjunto dos índices reais, o conjunto dos índices hipotéticos e o conjunto total de índices[1].

### 5.1.2 Fase de Observação: Monitoramento da Carga de Trabalho e Atualização do Benefício dos Índices

Nesta fase, mediante consultas à metabase do SGBD utilizado, captura-se, periodicamente[2], a carga de trabalho (conjunto de *triplas* do tipo <cláusula SQL, plano de execução, estimativas de custo>) submetida ao sistema de banco de dados (através do agente *AWO*, descrito no Capítulo 6). Desta forma, conseguimos obter as cláusulas SQL que foram ou estão sendo executadas, juntamente com seus respectivos planos de execução e estimativas de custo. Estas informações são armazenadas na metabase local (*LM*, Capítulo 6).

Em seguida, cada *tripla <cláusula SQL, plano de execução, custo>* é analisada com a finalidade de identificar os índices mais adequados e seus respectivos benefícios para carga de trabalho como um todo (tarefa realizada

[1]Observe que os conjuntos $IR$, $IH$ e $I$ são especializações dos conjuntos $ER$, $EH$ e $E$, definidos na Seção 2.8

[2]Essa periodicidade pode ser parametrizada pelo DBA.

pelo o agente *AIS*, Capítulo 6). As informações armazenadas, para cada *tripla*, permitem gerar planos hipotéticos, equivalentes ao plano de execução original, que potencialmente apresentem custos de execução menores do que o custo do plano original. As estimativas de custos dos planos hipotéticos são obtidas utilizando-se o modelo de custos externo discutido no Apêndice A. Os índices hipotéticos que compõem um plano de execução hipotético são considerados os mais indicados para a a cláusula SQL analisada, uma vez que sua existência faria com que o otimizador de consultas do SGBD, potencialmente, gerasse um plano de execução de custo mais baixo do que o plano de execução original. O benefício de um índice hipotético é calculado a partir da diferença entre o custo do plano hipotético e o custo do plano original. Vale destacar que esta análise é realizada sem que seja necessário fazer chamadas adicionais ao otimizador de consultas, podendo até mesmo ser executada em máquina distinta daquela utilizada para hospedar o SGBD.

**- Heurística Integrada para Seleção e Acompanhamento de Índices**

A seguir, descrevemos uma heurística integrada para a seleção e acompanhamento de índices (hipotéticos e reais). Na abordagem proposta, o agente *AIM* (*AIM - Agent for Index Maintenance*), definido no Capítulo 6, verifica periodicamente[3] se novas tarefas foram capturadas e adicionadas à metabase local (LM). Em caso afirmativo, a heurística HISAI, apresentada nas Figuras 5.1 e 5.2, é executada [4].

Como afirmado no capítulo anterior, a heurística HISAI baseia-se no conceito de "benefício" (Definição 6). A idéia básica consiste em acompanhar os índices, atribuindo-lhes benefícios à medida que estes contribuam, real ou potencialmente, de forma positiva, nos comandos executados pelo SGBD, ou seja, contribuam para reduzir o custo de execução das cláusulas SQL executadas. Assim, o benefício de um índice (que é um valor numérico) é incrementado quando detectado que a existência deste índice reduziu (caso índice real) ou poderia reduzir (caso índice hipotético) o custo de execução de comando SQL anteriormente executado.

Contudo, na abordagem proposta nesta tese, cada expressão SQL é otimizada uma única vez, diferentemente das soluções propostas em (Sal04, Morelli06a, Luh07), nas quais cada consulta submetida ao SGBD é otimizada duas (Luh07) e até três vezes (Sal04, Morelli06a). Outra diferença fundamental é que as propostas apresentadas em (Sal04, Morelli06a, Luh07) utilizam

[3]Essa periodicidade também pode ser parametrizada pelo DBA.
[4]A descrição detalhada desta heurística encontra-se no Apêndice B.

**Para** cada tripla <Consulta SQL, Plano de Execução, Custo> capturada **faça**
**Passo 1:** Inicializar a lista de índices candidatos para a tripla corrente ($L_{candidatos} = \phi$)
**Passo 2:** Percorrer o plano de execução (árvore de consulta) procurando por uma operação "*Seq Scan*"
**Passo 3:** Caso uma operação "*Seq Scan*" seja encontrada **faça**
**Caso 1:** Se a operação de "*Seq Scan*" possui filtro (condição de seleção)
Para cada atributo presente no "filtro" atualize a lista de índices candidatos ($L_{candidatos}$)
**Caso 2:** Se a operação de "*Seq Scan*" não possui filtro
Atualize a lista de índices candidatos a partir dos atributos presentes na cláusula SELECT que pertencem à tabela
**Passo 4:** Para cada índice candidato $i \in L_{candidatos}$ **faça**
Estimar o custo de utilização de um *Index Scan*, caso existisse um índice primário ("*clustering*") $i$ ($EC_{S_{IP}}$)
**Se** $EC_{S_{IP}} < EC_{FS}$ **então faça:**
$BA_{i\_primario} \leftarrow BA_{i\_primario} + (EC_{SS} - EC_{S_{IP}})$
Estimar o custo de utilização de um *Index Scan*, caso existisse um índice secundário ("*not clustering*") $i$ ($EC_{S_{IS}}$)
**Se** $EC_{S_{IS}} < EC_{FS}$ **então faça:**
$BA_{i\_secundario} \leftarrow BA_{i\_secundario} + (EC_{SS} - EC_{S_{IP}})$

Figura 5.1: Heurística Integrada: Avaliação de Consultas (Parte 1).



duas heurísticas distintas: uma para seleção e outra para o acompanhamento dos índices candidatos. Já na abordagem proposta neste trabalho, realizamos estas duas tarefas em um passo único, mediante uma heurística integrada. Além disso, a heurística HISAI produz uma quantidade de índices hipotéticos substancialmente menor do que a heurística apresentada em (Loh00) e implementada em (Sal04, Sal05, Cos05, Morelli06a, Lif06, Luh07).

Observe ainda que, ao se atualizar o benefício acumulado de um determinado índice $i_k$ cuja chave de busca é formada pela seqüência de atributos $(a_1, a_2, ...a_n)$, deve-se automaticamente atualizar também o benefício acumulado de todo índice $i_j$ cuja chave de busca tem por prefixo os atributos da chave de busca do índice $i_k$, $(a_1, a_2, ...a_n, b_1, b_2, b_m)$. Para ilustrar este fato, considere um índice $i_k$ cuja chave de busca é formada pelo atributo $A$ e um índice $i_j$ cuja chave de busca é formada pelos atributos $A, B$. Note que, nos contextos onde o índice $i_k$ foi utilizado, poderia-se utilizar em seu lugar o índice $i_j$.

Assim, seja $I$ o conjunto dos índices e dados dois índices $i_k$ e $i_j$, tais que $i_k \in I$ e $i_j \in I$, dizemos que $i_k$ é um prefixo de $i_j$ se $\mathsf{chave\_busca}_{\mathsf{i_k}}$ é um prefixo[5] de $\mathsf{chave\_busca}_{\mathsf{i_j}}$. Seja, $B_{i_k,q}$ o benefício do índice $i_k$ para uma determinada consulta $q$, temos que:

$\forall i_k \in I$ : Se $i_k$ é prefixo de $i_j \Rightarrow B_{i_j,q} = B_{i_k,q}$.

As operações de atualização ("update") também necessitam ser analisadas. Uma operação de atualização sobre uma determinada tabela $t$ pode implicar na necessidade de se atualizar as estruturas de índices definidas sobre $t$. Neste caso, o custo de atualização da estrutura de índice pode ser conside-

[5]A definição de prefixo de um índice foi apresentada na Seção 5.2.1.

**Para** cada tripla <Consulta SQL, Plano de Execução, Custo> capturada **faça**
**Passo 1:** Inicializar a lista de índices reais para a tripla corrente ($L_{reais} = \phi$)
**Passo 2:** Percorrer o plano de execução (árvore de consulta) procurando por uma operação "*Index Scan*"
**Passo 3:** Caso uma operação "*Index Scan*" seja encontrada **faça**
**Caso 1:** Se a operação de "*Index Scan*" possui filtro (condição de seleção)
Para cada atributo presente no "filtro" atualize a lista de índices reais ($L_{reais}$)
**Caso 2:** Se a operação de "*Index Scan*" não possui filtro
Atualize a lista de índices reais a partir dos atributos presentes na cláusula SELECT que pertencem à tabela
**Passo 4:** Para cada índice real $i \in L_{reais}$ **faça**
Estimar o custo de utilização de um *Seq Scan*, caso o índice real $i$ não existisse ($EC_{FS}$)
**Caso 1:** Se $i$ é um índice primário ("*clustering*")
Estimar o custo do *Index Scan* usando $i$ ($EC_{S_{IP}}$)
**Se** $EC_{S_{IP}} < EC_{FS}$ **então faça:**
$BA_{i\_primario} \leftarrow BA_{i\_primario} + (EC_{SS} - EC_{S_{IP}})$
**Caso 2:** Se $i$ é um índice secundário ("*not clustering*")
Estimar o custo do *Index Scan* usando $i$ ($EC_{S_{IS}}$)
**Se** $EC_{S_{IS}} < EC_{FS}$ **então faça:**
$BA_{i\_secundario} \leftarrow BA_{i\_secundario} + (EC_{SS} - EC_{S_{IP}})$

Figura 5.2: Heurística Integrada: Avaliação de Consultas (Parte 2).

rado como um "malefício" (ou seja, benefício negativo). Seja $q_u$ uma operação de atualização sobre a tabela $t$, $A$ o conjunto dos atributos de $t$ que ocorrem na cláusula "*set*" de $q_u$, $N_{q_u}$ o número de linhas afetadas por $q_u$ e $I$ o conjunto dos índices hipotéticos e reais (Luh07). Temos que:

$$\forall i_k \in I : a_j \in A \text{ e } a_j \in i_k \Rightarrow BA_{i_k} = BA_{i_k} - HT_{i_k} \times N_{q_u} \times F,$$

onde $F$ é um fator derivado empiricamente para representar o custo de atualização de uma entrada na estrutura de índices.

A idéia utilizada para se quantificar o "malefício" (ou benefício negativo) de uma operação de atualização para um determinado índice $i$ consiste no somatório do custo de atualização de cada entrada uma das entradas do índice que serão atualizadas (individualmente). O custo de atualização de uma única entrada consiste no custo de percorrer a estrutura de índice para encontrar a entrada a ser atualizada mais o fator $F$.

A seguir, discute-se uma estratégia para avaliação das operações de atualização.

### Estratégia para Avaliação de Atualizações

Um comando SQL de atualização (update, insert, delete) sobre uma determinada tabela $t$ pode implicar na necessidade de se atualizar as estruturas de índices definidas sobre $t$. Neste caso, o custo de atualização da estrutura de índice é considerado como um "malefício" (ou seja, um benefício negativo).

Em (Luh07), os autores argumentam que a estimativa do custo de uma

operação de atualização $q_u$ deve considerar o número de linhas afetadas por $q_u$ ($N_{q_u}$), a altura da árvore ($HT_{i_k}$) e um fator $F$, conforme discutimos anteriormente. Vale ressaltar, no entanto, que a presença de um índice também pode acelerar as operações de atualização, apesar da necessidade de reorganização da estrutura do índice. É interessante observar que comandos de atualização devem inicialmente trazer os dados para a memória, exatamente como consultas, para, somente então, processar modificações sobre os dados. Neste caso, não se mostra adequado somente apenar os índices afetados pelas operações de atualização. Pelo contrário, devemos calcular o benefício que estes índices trazem para as operações de atualização.

Em (Sal04), os autores propõem que, se o comando submetido ao SGBD for uma atualização, deve-se inicialmente aplicar a heurística de "Avaliação de Consultas" (de forma idêntica à realizada para consultas), a fim de calcular o benefício trazido pelos índices para a operação de atualização. Normalmente, os otimizadores de SGBDs relacionais oferecem estimativas de custos para recuperar os dados necessários à atualização, mas não para os custos envolvidos na modificação propriamente dita dos dados (o que envolve também o custo de reorganização das estruturas de índices afetadas pela atualização). Após a aplicação do mesmo procedimento para avaliação de consultas (Figuras 5.1 e 5.2), contabilizam-se os custos de manutenção dos índices. Estes custos de manutenção serão debitados do benefício acumulado tanto de índices candidatos quanto de índices reais. Em seguida, verifica-se se manter os índices na base continua sendo vantajoso.

Neste trabalho, propomos uma estratégia de avaliação de atualizações semelhante à abordagem proposta em (Sal04). A figura 5.3 mostra o algoritmo utilizado para avaliação das operações de atualização.

**Para** cada tripla <Cláusula SQL de Atualização, Plano de Execução, Custo> capturada **faça:**
  Executar a heurística de "Avaliação de Consultas"
  **Para** cada índice $i$ afetado dela atualização **faça:**
    $BA_i \leftarrow BA_i - EC_{A_i}$
    **Se** ($i$ é real) e ($BA_i < 0$) e ($|BA_i| > EC_{C_i}$) **então**
      $BA_i \leftarrow -EC_{C_i}$
      **Retornar**
    **Fim Se**
  **Fim Para**
**Fim Para**

Figura 5.3: Heurística Integrada: Avaliação de Atualizações.

### Reconstrução Automática de Índices

Em (Morelli06a), os autores apresentam uma heurística para reconstrução automática de índices, a qual analisa situações de eliminação iminente e, caso julgue interessante, dispara a reconstrução do índice. Assim, esta heurística estabelece regras que permitem decidir se um índice deve ser reconstruído ou eliminado.

Algumas observações importantes foram discutidas em (Morelli06a):

1. **Tamanho das Tabela:** Tabelas com poucos blocos não devem ter seus índices cogitados à reconstrução. Por exemplo, imagine a tabela $t_1$, com 1.000 *tuplas* e um índice com apenas dois blocos plenamente ocupados (razão *tuplas*/blocos = 500). Quando ocorrer a inserção da milésima primeira linha, o índice ganhará um terceiro bloco, porém sua razão cairá para 333 (um decréscimo de 33,4%). Apesar de apresentar considerável queda em sua razão tuplas/blocos, a reconstrução do índice jamais proporcionaria menos de três blocos.

2. **Quantidade de Varreduras:** Consultas envolvendo operações em que várias páginas precisem ser lidas de forma contígua (varreduras) devem ter seu desempenho degradado, caso utilizem índices fragmentados. Neste caso, a reconstrução destes índices impede a degradação do desempenho das consultas. Logicamente, índices utilizados em varreduras devem ser candidatos à reconstrução.

3. **Quantidade de Atualizações:** Índices que passam por grande quantidade de atualizações (operações de update, insert e delete) tendem a apresentar elevada incidência de *page splits* (divisão de páginas) e, conseqüentemente, apresentar fragmentação.

4. **Consultas Pontuais:** Índices fragmentados não degradam o desempenho de consultas pontuais, ou seja, que não realizem varreduras, como por exemplo, consultas de igualdade (*exact match*), já que a duração de acessos contingenciais tende a ser a mesma.

5. **Fator de Preenchimento:** Uma vez constatada a fragmentação de um índice, causada por sucessivas ocorrências de *page splits*, caso decida-se recriá-lo, recomenda-se fazê-lo deixando uma margem para futuras atualizações. Normalmente SGBDs possuem mecanismos que permitem dosar quantos *bytes* podem ser gravados por bloco. No Oracle, por exemplo, a cláusula pctfree do comando create index estabelece um limite, que representa o percentual de espaço livre a ser deixado por bloco.

Indica uma folga capaz de receber valores alterados de chaves sem haver necessidade de alocação de novos blocos ou migração de chaves para outros blocos. Já no SQL Server, a cláusula denomina-se fillfactor e determina o percentual (0-100) de preenchimento das páginas do nível folha. Quanto menor o valor especificado, maior a folga por página. O PostgreSQL possui este conceito, porém não permite configurá-lo. O valor deste parâmetro está fixado, no próprio código fonte (*hard coded*), em 90% para páginas nível folha e 70% para as demais.

Vale destacar, no entanto, que tabelas muito atualizadas devem possuir índices com *fillfactor* pequeno, a fim de evitar *page splits* e, por conseguinte, a fragmentação do índice. Por outro lado, deve-se ressaltar que, ao reduzir o *fillfactor*, diminui-se a quantidade de informações por bloco, levando à necessidade de maior alocação de páginas. Essa maior quantidade de páginas diminui o desempenho das operações de varredura. Por outro lado, se utilizarmos um *fillfactor* grande, teremos um decréscimo na quantidade de páginas, o que fará com que as varreduras aconteçam em menos tempo (já que há menos páginas a percorrer). Entretanto, futuras atualizações causarão *page splits*, o que prejudicará o desempenho destas mesmas consultas no futuro. Na prática, para tabelas muito atualizadas, diminuir o *fillfactor* traz melhores resultados de *performance*. Agora, vale destacar que, se uma determinada tabela $t$ é muito atualizada, mas não é utilizada em consultas de varredura (faixa de valores), a fragmentações dos índices definidos sobre esta tabela não constitui um grande problema, uma vez que os índices fragmentados não degradam o desempenho de consultas circunstanciais. Por outro lado, tabelas poucos atualizadas tendem a apresentar pequena quantidade de *page splits*, o que resulta em índices pouco fragmentados.

A Heurística de Reconstrução Automática de Índices, apresentada em (Morelli06a), determina um fator de preenchimento de páginas com base no histórico de operações de varreduras nas quais participou de forma positiva o índice em vias de reconstrução. Nesta abordagem, quanto mais varreduras um índice fragmentado tiver tido, menor será o seu *fillfactor*, objetivando reduzir a incidência de *page splits*. Como este trabalho utilizou o PostgreSQL para realizar as implementações, houve a necessidade de estender os comandos create index, reindex index e reindex table para que aceitassem uma nova cláusula, denominada *fillfactor*, o qual pode receber valores entre 1 e 9. O menor valor significa que apenas um décimo de cada página será ocupado, enquanto o maior sinaliza 90% de ocupação.

Este fator de preenchimento (*fillfactor*) é calculado segundo a fórmula:

$$F = 10 - [(V + 10)div10].$$

O fator $F$ representa o *fillfactor* de um índice a ser reconstruído, com base na quantidade de varreduras ($V$). Já a variável $V$ corresponde ao número de varreduras nas quais o índice participou desde sua criação. Assim, quanto mais varreduras um índice fragmentado tiver tido, menor será o seu *fillfactor* objetivando reduzir a incidência de *page splits*. Entretanto, deve-se ressaltar que, ao reduzir o *fillfactor*, diminui-se a quantidade de informações por bloco, levando à necessidade de maior alocação de páginas (Morelli06a).

A heurística de reconstrução automática de índices, no entanto, poderia decidir pela reconstrução antes da eliminação iminente. Neste caso, não seria necessário manter um índice fragmentado até o momento em que se decida eliminá-lo ou recriá-lo. Neste sentido, propomos algumas modificações na heurística de reconstrução a fim de acompanhar com maior precisão a incidência de *page splits* e, conseqüentemente, o nível de fragmentação das estruturas de índices.

Primeiramente, introduzimos um parâmetro denominado "*fraglimit*", a ser definido pelo DBA. Este parâmetro indica o nível máximo de fragmentação permitido para um índice, podendo receber valores entre 1% e 100%. Assim, se "*fraglimit*" foi definido como 60% e o nível de fragmentação ultrapassar este valor, o índice deve ser reconstruído. Para obter o valor atual do nível de fragmentação de um determinado índice, consultamos a metabase do SGBD utilizado. No SQL Server 2005, por exemplo, podemos utilizar a instrução DBCC SHOWCONTIG ou a visão sys.dm_db_index_physical_stats. Para SGBDs que não disponibilizem essa informação na sua metabase, podemos utilizar o conceito de razão de fragmentação, como proposto em (Morelli06a). Em nossa abordagem, a heurística de reconstrução automática de índices é executada sempre que um índice é afetado por uma operação de atualização. Logo, esta heurística foi definida em conjunto com a estratégia de avaliação de atualizações. A figura 5.4 mostra o algoritmo utilizado para a avaliação conjunta das operações de atualização e da necessidade de reconstrução de índices.

Ao reconstruir uma estrutura de índice, deve-se estipular o fator de preenchimento. Para isso, podemos utilizar a estratégia proposta em (Morelli06a).

$L_{ind\_frag} \leftarrow \phi$ // Lista de índices fragmentados
**Para** cada tripla <Cláusula SQL de Atualização, Plano de Execução, Custo> capturada **faça:**
  Executar a heurística de "Avaliação de Consultas"
  **Para** cada índice $i$ afetado dela atualização **faça:**
    $BA_i \leftarrow BA_i - EC_{A_i}$
    $NF_i \leftarrow$ Nível de fragmentação do índice $i$ (Recuperado da metabase do SGBD
    $TU_i \leftarrow$ Total de vezes que o índice $i$ foi utilizado
    **Se** ($i$ é real) e ($BA_i < 0$) e ($|BA_i| > EC_{C_i}$) **então**
      $BA_i \leftarrow -EC_{C_i}$
      **Retornar**
    **Fim Se**
    **Se** ($i$ é real) e ($NF_i * \frac{V}{TU_i} > fraglimit$) **então**
      $L_{ind\_frag} \leftarrow L_{ind\_frag} \cup i$
    **Fim Se**
  **Fim Para**
**Fim Para**

Figura 5.4: Heurística Integrada: Avaliação de Atualizações.

### 5.1.3 Fase de Predição: Seleção *On-Line* de Índices

Esta fase é iniciada sempre que o benefício acumulado de um índice hipotético $k$ qualquer ultrapassar o seu custo estimado de criação ($BA_k > EC_{C_k}$, onde $EC_{C_K} = 2 \times P_R + c \times N_R \log N_R$[6], $P_R$ é o número de páginas da relação $R$, relação sobre a qual o índice $k$ é definido, $N_R$ é o número de *tuplas* da relação $R$ e $c$ é uma constante (Sal05)), indicando que a existência do índice iria contribuir para diminuir o custo de processamento da carga de trabalho submetida ao SGBD, ou quando o benefício acumulado de um índice real $k'$ alcança um valor negativo que supera, em módulo, o seu custo de criação ($BA_{k'} < 0$ e $|BA_{k'}| > EC_{C_k}$), indicando que a existência de $k'$ não está contribuindo para diminuir o custo de processamento da carga de trabalho.

O problema a ser resolvido nesta fase consiste basicamente em buscar obter uma configuração de índices ótima, levando em consideração as restrições de espaço físico. Para isso, utilizaremos as informações coletadas durante a fase de observação, as quais foram armazenadas na metabase local (definida no capítulo 6).

Seja $I$ o conjunto dos índices (incluindo índices hipotéticos e reais). Para cada índice $i \in I$, iremos utilizar: $BA_i$, $P_i$, $state(i)$ (função que indica se o índice é hipotético ou real) e $type(i)$ (função que indica se o índice é primário (P) ou secundário (S)). Além disso, definimos *storage_constraint* como o

[6]O custo estimado de criação de uma estrutura de índices é definido no Apêndice A.

```
I_s[1...n] ← ordenar(I) através do benefício relativo
C̄ ← φ
espaço_disponível ← storage_constraint
benefício_total ← 0
Para todo k ← 1 ··· n faça
  Se (type(I_s[k]) = P) e (C̄ ⊃ I_s[j]) e (type(I_s[j]) = P) e (I_s[j] e I_s[k] definidos sobre a mesma tabela t) e k ≠ j então
    continue
  Se não
    Se espaço_disponível - P_{I_s[k]} > 0 então
      C̄ ← C̄ ⋃ I_s[k]
      espaço_disponível ← espaço_disponível - P_{I_s[k]}
      benefício_total ← benefício_total + BA_{I_s[k]}
    Fim Se
  Fim Se
Fim Para
Se benefício_total < threshold então
  C̄ ← C
Fim Se
retorne C̄
```

Figura 5.5: Algoritmo Guloso para Seleção de Índices.

espaço disponível para a materialização das estruturas de índices. A solução adotada utiliza um algoritmo guloso adaptado de (Luh07), o qual estende as funcionalidades originais a fim de considerar a seleção de índices primários e secundários (Figura 5.5). Neste sentido, todos os índices (reais e hipotéticos) são ordenados de acordo com o seu benefício relativo (Definição 8).

A lista contendo todos os índices ordenados pelo benefício relativo é utilizada como entrada pelo algoritmo apresentado na figura 5.5. Este algoritmo produz como saída uma nova configuração de índices $\bar{C}$, ou seja, um novo conjunto de índices 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 os mesmos índices sejam freqüentemente adicionados e removidos, uma nova configuração de índices $\bar{C}$ 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 (tipicamente $> 1.0$, como sugerido em (Luh07)).

Logicamente, esta abordagem não assegura uma solução ótima. Porém, os resultados dos testes de desempenho mostram que o resultado é suficientemente preciso, especialmente se considerarmos que as variáveis de entrada são estimadas e que o resultado é utilizado como um prognóstico do uso futuro das estruturas de índices, conforme veremos no Capítulo 7.

**Pré-Criação de Índices Envolvidos em Chaves Primárias e Estrangeiras**

Nos principais bancos de dados atuais, ao definirmos um atributo (ou conjunto de atributos) como chave primária, o SGBD cria automaticamente um índice primário definido sobre os atributos que compõem a chave primária. Os SGBDs adotam essa política como comportamento padrão por acreditar que o índice trará benefícios (ou seja, diminuirá o tempo de processamento) para operações de junções e manutenção das restrições de integridade referencial. Além disso, pelos mesmos motivos, recomenda-se a criação de índices secundários para as chaves estrangeiras.

Neste sentido, propomos a pré-criação automática dos índices envolvidos em chaves primárias e estrangeiras como forma de melhorar a qualidade do processo de seleção *on-the-fly* de índices. Vale ressaltar que, a partir de consultas à metabase do SGBD, é possível capturar, de forma automática, o esquema do banco de dados e, conseqüentemente, descobrir os atributos que compõem as chaves primárias e estrangeiras.

Outra possibilidade é atribuir um peso para os índices envolvidos em chaves primárias e secundárias e deixar que eles concorram com os demais índices para serem inseridos na mochila [7]. Este peso pode ser definido como um valor numérico no intervalo $[1, 2]$. Por exemplo, poderíamos atribuir peso 2 aos índices envolvidos em chaves primárias, 1.5 aos índices envolvidos em chaves secundárias e 1 aos demais índices. Neste caso, propomos uma pequena alteração no cálculo do benefício relativo:

$$BR_i = \frac{BA_i \times Peso_i}{P_i}.$$

Para ilustrar a utilização da idéia de "peso", considere a existência de dois índices $i_1$ e $i_2$. Assuma que o peso de um índice envolvido em uma chave primária foi definido como 2 e que o peso de um índice envolvido em uma chave estrangeira foi definido como 1,5. Assuma que o índice $i_1$ é um índice envolvido em uma chave primária e que $i_2$ é um índice envolvido em uma chave estrangeira. Logo, temos que: $Peso_{i_1} = 2$ e $Peso_{i_2} = 1,5$. Assuma também que: $BA_{i_1} = 80$, $BA_{i_2} = 100$, $P_{i_1} = 10$ e $P_{i_2} = 10$. Observe que se considerássemos somente o benefício acumulado, o índice $i_2$ seria mais indicado do que o índice $i_1$. Contudo, o índice $i_1$ está relacionado a uma chave primária, podendo

[7] O problema de seleção de índices é semelhante ao problema da mochila, conforme discutido na Seção 5.2.1.

$\bar{C} \leftarrow \phi$
espaço_disponível ← storage_constraint
benefício_total ← 0
pk ← conjunto das chaves primárias
fk ← conjunto das chaves secundárias (estrangeiras)
Seja $(I)$ o conjunto dos índices tais que: Se $k \in (I)$ então $k \notin pk$ e $k \notin fk$
$I_s[1...n] \leftarrow ordenar(I)$ através do benefício relativo
**Para todo** $k \in pk$ **faça**
  **Se** espaço_disponível - $P_{I_s[k]} > 0$ **então**
    $\bar{C} \leftarrow \bar{C} \bigcup I_s[k]$
    espaço_disponível ← espaço_disponível - $P_{I_s[k]}$
    benefício_total ← benefício_total + $BA_{I_s[k]}$
  **Fim Se**
**Fim Para**
**Para todo** $k \in fk$ **faça**
  **Se** espaço_disponível - $P_{I_s[k]} > 0$ **então**
    $\bar{C} \leftarrow \bar{C} \bigcup I_s[k]$
    espaço_disponível ← espaço_disponível - $P_{I_s[k]}$
    benefício_total ← benefício_total + $BA_{I_s[k]}$
  **Fim Se**
**Fim Para**
**Para todo** $k \leftarrow 1 \cdots n$ **faça**
  **Se** $(type(I_s[k]) = P)$ e $(\bar{C} \supset I_s[j])$ e $(type(I_s[j]) = P)$ e ($I_s[j]$ e $I_s[k]$ definidos sobre a mesma tabela $t$) e $k \neq j$ **então**
    **retorne**
  **Se não**
    **Se** espaço_disponível - $P_{I_s[k]} > 0$ **então**
      $\bar{C} \leftarrow \bar{C} \bigcup I_s[k]$
      espaço_disponível ← espaço_disponível - $P_{I[k]}$
      benefício_total ← benefício_total + $BA_{I_s[k]}$
    **Fim Se**
  **Fim Se**
**Fim Para**
**Se** benefício_total < threshold **então**
  $\bar{C} \leftarrow C$
**Fim Se**
**retorne** $\bar{C}$

Figura 5.6: Algoritmo Guloso para Seleção de Índices: Considerando a Pré-Criação de Índices Envolvidos em Chaves Primárias e Estrangeiras.

apresentar uma utilidade maior ao longo do tempo. Porém, se observarmos o benefício relativo, calculado a partir da idéia de "peso", veremos que o benefício relativo de $i_1$ ($BR_{i_1} = 16$) é maior do que o benefício relativo de $i_2$ ($BR_{i_2} = 15$). Assim, utilizando-se o benefício relativo, temos que o índice $i_2$ é mais indicado do que o índice $i_1$.

Vale ressaltar que, neste caso, podemos utilizar o mesmo algoritmo descrito na figura 5.5, alterando apenas o cálculo de $BR_i$.

### 5.1.4 Fase de Reação: Criação, Remoção e Reorganização das Estruturas de Índices

Ao final da fase de predição, caso seja detectada a necessidade de mudanças na configuração de índices ($\bar{C} \neq \phi$) ou a existência de índices fragmentados ($L_{frag} \neq \phi$), a fase de reação é iniciada.

No primeiro caso, deve-se substituir a configuração de índices atual $C$ por uma uma nova configuração $\bar{C}$. Assim, se um determinado índice $k \in C$ e $k \notin \bar{C}$, então $k$ deve ser removido (inclusive do catálogo de índices do SGBD). Todavia, o índice $k$ continua existindo na metabase local, porém, seu estado é alterado de real para hipotético e $BA_k = 0$. Logo, nada impede que, futuramente, este índice seja novamente materializado. Note que o custo de excluir um índice pode ser desprezado, uma vez que esta tarefa consiste basicamente em liberar espaço em disco, remover uma entrada do catálogo de índices do SGBD e atualizar o estado e o benefício acumulado deste índice na metabase local. Por outro lado, todo índice $k$ tal que $k \in \bar{C}$ e $k \notin C$ deve ser materializado, ou seja, fisicamente criado. Já no segundo caso, deve-se executar a reconstrução de cada índice $i \in L_{frag}$. A Figura 5.7 mostra o algoritmo concebido para atualizar a configuração de índices.

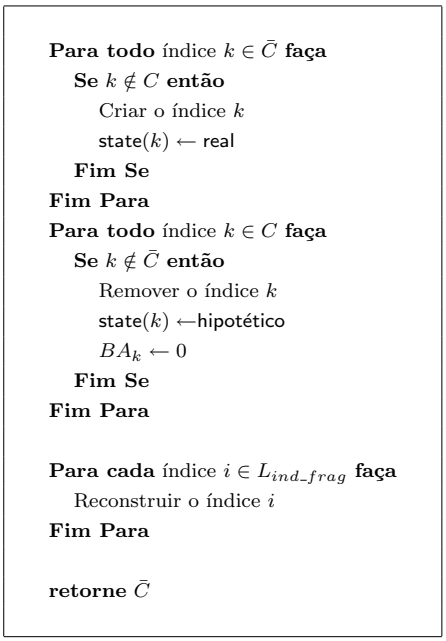


Figura 5.7: Algoritmo para Atualização da Configuração de Índices.

### 5.1.5 Considerando a Qualidade da Configuração de Índices

#### Relevância e Generalidade

Como discutido no capítulo anterior, em princípio, consideramos todas as transações envolvidas igualmente importantes para os usuários do sistema. Essa premissa, no entanto, nem sempre é verdadeira. Desta forma, propomos no capítulo anterior a utilização do conceito de "Relevância". A seguir, utilizamos o conceito de relevância na redefinição do benefício relativo de uma estrutura de índice. Assim, o benefício relativo passa a ser calculado da seguinte forma:

$$BR_i = \frac{BA_i \times \frac{Ri}{10}}{P_i}.$$

Ainda no capítulo anterior, propomos uma métrica, denominada "generalidade", para representar a qualidade de um índice para carga de treinamento como um todo. A métrica proposta ($G_i$) consiste em um valor entre 0 e 1, onde, 0 significa que o índice possui baixa generalidade e 1 indica que o índice tem generalidade máxima. A métrica proposta ("generalidade", $G_i$) é definida como:

$$G_i = \frac{NQi}{NQ_{Wt}},$$

onde $NQi$ é o número de cláusulas SQL nas quais o índice $i$ foi ou poderia ter sido utilizado e $NQ_{Wt}$ é o número total de cláusulas SQL da carga de treinamento.

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

$$BR_i = \frac{BA_i \times G_i}{P_i}.$$



**Considerando a Estabilidade das Configurações**

No capítulo anterior, argumentamos que a estabilidade de uma configuração de índices é uma questão relevante. Configurações instáveis conduzem a freqüentes criações e remoções dos mesmos índices (Luh07).

Para solucionar estes problemas, a abordagem não-intrusiva propõe a utilização do conceito de época, o qual foi inicialmente proposto em (Kai04, Luh07), com algumas alterações. Dentre as principais alterações discutidas no capítulo anterior, temos:

– A redefinição do cálculo do benefício acumulado de uma estrutura de índice utilizando o conceito de "época", como mostrado a seguir:

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

– A utilização de um benefício inicial:

$$B_{0_i} = -EC_{C_i} + EC_{C_i} \times \frac{N_{E_i}}{N_E}.$$

– A utilização conjunta do conceito de "época" e do benefício inicial:

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

### 5.1.6 Atribuição Proporcional de Benefícios

Em (Sal04), propõe-se que todo índice participante de um comando receba o mesmo benefício de todos os índices presentes no comando. Esta abordagem pode criar distorções, já que um índice, que pouco tenha contribuído para baixar o custo de um comando, tenha sido selecionado em um comando com um índice com significativa contribuição, terá um benefício acumulado falacioso.

Por exemplo, imagine que o custo de uma determinada consulta seja 2.000, mas, graças à utilização de dois índices, caia para 500. Suponha também que, dos 1.500 obtidos de benefício, 1.490 sejam devidos apenas ao primeiro índice. Ora, segundo (Sal04), os dois índices receberiam o benefício de 1.500, o que criaria uma flagrante distorção, já que o índice, que contribuiu apenas com 10, ganharia um benefício desproporcional à sua participação (Morelli06a).

Uma alternativa mais justa consistiria em atribuir a cada índice uma parcela de ganhos proporcional à sua contribuição. Em vez de cada índice participante de um comando receber um benefício fixo, tal qual o ganho total do comando, como proposto em (Sal04), seria concedido um valor variável que dependeria da real contribuição de cada índice ao comando no qual este índice estaria participando. Em (Cha04), os autores apresentam um algoritmo que associa a cada índice ganhos proporcionais a sua contribuição para reduzir o custo global da carga. Já (Morelli06a) adota uma estratégia intermediária entre (Sal04) e (Cha04).

Em (Morelli06a), decidiu-se realizar o acompanhamento de índices criados: durante a fase como hipotético, enquanto o índice vai acumulando benefícios, também se registra a quantidade de vezes em que ele foi útil. No momento em que deixa de ser hipotético para ser real, o índice ganha um "bônus" resultante da divisão entre o benefício acumulado, ou seja, igual ou um pouco superior ao custo de criação, pela quantidade de vezes em que foi utilizado. Durante sua fase como índice criado, toda vez que seja utilizado, seu benefício acumulado receberá um valor idêntico ao bônus. Logicamente, esta estratégia consiste apenas em uma aproximação e apresenta uma precisão menor do que a abordagem utilizada em (Cha04).

Vale ressaltar que a heurística integrada de seleção e acompanhamento de índices proposta neste capítulo atribui a cada índice uma parcela de ganhos proporcional à sua contribuição. Para constatar este fato, basta observar que o benefício atribuído a um determinado índice $i$ é calculado de acordo com o ganho obtido por sua utilização em uma determinada operação do plano hipotético e não da consulta como um todo.

### 5.1.7 Discussão

Nesta seção, apresentamos como instanciar a abordagem de sintonia automática proposta nesta tese, a fim de realizar a manutenção automática de índices. Esta instanciação apresenta as seguintes inovações: uma heurística integrada para a seleção e acompanhamento de índices (Heurística *HISAI*), um conjunto de algoritmos para seleção e atualização de uma configuração de índices, a consideração tanto de índices primários quanto secundários, uma avaliação da pré-criação de índices envolvidos em chaves primárias e estrangeiras, o acompanhamento do nível de fragmentação das estruturas de índice, a utilização dos conceitos de relevância e generalidade de índices, bem como do conceito de estabilidade de uma configuração de índices.

Além disso, a instanciação elaborada produz uma quantidade baixa de índices hipotéticos, diminuindo a sobrecarga despendida com a manutenção destas estruturas na metabase local; cada expressão SQL capturada é otimizada uma única vez; a atribuição de benefícios é realizada considerando a contribuição específica de cada índice, de forma proporcional; e o processo de seleção de uma configuração de índices leva em consideração as restrições quanto ao espaço físico disponível para a materialização dessas estruturas.

Na Seção 3.3.7, foi apresentado um panorama do estado da arte dos trabalhos relacionados à manutenção automática de índices, destacando-se as características de cada uma das soluções encontradas na literatura. A tabela 3.1 apresentada naquele Capítulo é repetida aqui, acrescentando-se as informações relativas à abordagem desenvolvida nesta tese. As características utilizadas nesta análise comparativas foram descritas na Seção 3.3.7.

| Carac. | Sal04 | Morelli06 | Sattler03 | Sch06 | Luh07 | Bruno07 | Tese |
|---|---|---|---|---|---|---|---|
| C1 | Índice Sec., $B^+$ | Índice Sec., $B^+$ | Índice Sec., $B^+$ | Índice Sec., $B^+$ | Índice Sec., $B^+$ | Índice Sec., $B^+$ | Índices Prim. e Sec., $B^+$ |
| C2 | Criação, Remoção | Criação, Remoção, Reorganização | Criação | Criação, Remoção | Criação, Remoção | Criação, Remoção | Criação, Remoção, Reorganização |
| C3 | Intrusivo | Intrusivo | Intrusivo | Intrusivo | Intrusivo | Intrusivo | Não-Intrusivo |
| C4 | Postgre | Postgre | DB2 | Postgre | Postgre | SQL Server | Qualquer |
| C5 | 2 | 2 | 2 | 2 | 2 | 1 | 1 |
| C6 | Alta | Alta | Alta | Alta | Alta | Baixa | Baixa |
| C7 | Interno | Interno | Interno | Interno | Interno | Interno | Externo |
| C8 | Não | Não | Sim | Sim | Sim | Sim | Sim |
| C9 | Não | Parcial | Não | Sim | Sim | Sim | Sim |
| C10 | 3 | 3 | 2 | 2 | 2 | 1 | 1 |
| C11 | 4 | 6 | 0 | * | * | * | 0 |
| C12 | TPC-C | TPC-H | TPC-H | * | TPC-H | TPC-H | TPC-H |
| C13 | Não | Não | Sim | Sim | Sim | Sim | Sim |
| C14 | Não | Não | Não | Não | Não | Não | Sim |
| C15 | Não | Não | Não | Não | Não | Não | Sim |
| C16 | Não | Não | Não | Não | Não | Não | Sim |

Tabela 5.1: Análise Comparativa entre os Trabalhos Relacionados e a Abordagem Proposta.



## 5.2 Estudo de Caso 2: Manutenção Automática e Não-Intrusiva de *Clusters* Alternativos de Dados

Esta seção apresenta a instanciação da abordagem não-intrusiva para a manutenção automática de *clusters* alternativos de dados. Denominamos *clusters* alternativos de dados as estruturas físicas duplicadas, a fim de permitir a ordenação física dos dados armazenados por diferentes critérios. Desta forma, como definido na Seção 2.3, o processo de duplicação de estruturas físicas é denominado de "*clusterização* alternativa de dados" e as "cópias" destas estruturas são chamadas "*clusters* alternativos de dados".

### 5.2.1 Preliminares

A duplicação de estruturas físicas, como tabelas, por exemplo, a fim de permitir duas ou mais ordenações físicas distintas, ainda é uma alternativa de projeto físico pouco explorada na literatura. Atualmente os SGBDs permitem

que uma determinada tabela seja fisicamente ordenada (classificada) por um único critério. Este critério determina a ordem com que os registros da tabela são fisicamente armazenados em disco. Em geral, o critério de ordenação utilizado coincide com a chave do índice primário[8] definido sobre a tabela.

Desta forma, como discutido na Seção 2.3, não é possível ordenar fisicamente uma mesma estrutura, uma tabela, por exemplo, por dois ou mais critérios diferentes. Contudo, a ordenação física dos registros que compõem uma tabela influencia de forma significativa o desempenho de determinadas consultas. Por exemplo, a existência de "cópias" de uma mesma tabela, fisicamente ordenadas por critérios distintos, permitem melhorar simultaneamente o desempenho de consultas agregadas por estes diferentes critérios. Uma possível solução para este problema consiste na duplicação de estruturas físicas com base na utilização de visões materializadas.

A idéia básica consiste primeiramente em descobrir quais os critérios de ordenação física mais relevantes para uma determinada estrutura física, uma tabela, por exemplo. O critério considerado mais relevante, ou seja, a ordenação física que proporcione o maior benefício para a carga de trabalho como um todo, será utilizado para ordenar fisicamente a tabela. Em seguida, para cada um dos $k$ critérios de ordenação considerados relevantes, ou seja, que proporcionem um benefício considerável para a carga de trabalho, é criada uma visão materializada fisicamente ordenada por este critério. Vale ressaltar que neste momento duas estratégias são possíveis: a) criar uma cópia completa da tabela (neste caso, a visão materializada conterá todas as *tuplas* da tabela base, só que fisicamente ordenada de uma maneira distinta) e b) criar uma cópia parcial da tabela (neste caso, a visão materializada conterá um subconjunto das *tuplas* da tabela base, e será fisicamente ordenada de maneira distinta). Para a implementação desta segunda estratégia, uma solução possível seria manter uma lista das varreduras (*Table Scan*, por exemplo) executas sobre a tabela, juntamente com os predicados (filtros) utilizados. Assim, o predicado usado para definir a visão materializada seria uma disjunção dos predicados empregados nas varreduras executadas sobre a tabela.

O primeiro desafio consiste em descobrir quais os critérios de ordenação física mais relevantes. Para isso, utilizamos a heurística de benefícios definida no estudo de caso anterior. Por meio desta heurística, podemos obter a lista dos índices primários candidatos e seus respectivos benefícios relativos, ordenada pelo benefício relativo. Vale ressaltar que cada índice primário implica em um critério de ordenação física específico. Logo, a lista ordenada de índices primários candidatos corresponde à lista dos critérios de ordenação física mais

[8] O conceito de índice primário é definido na Seção 2.2.

relevantes (já ordenada). Assim, cada índice primário candidato representa um *cluster* alternativo de dados. O índice primário candidato de maior benefício relativo é utilizado para determinar a ordem física dos registros da tabela (denominado *cluster* principal). Em seguida, selecionam-se os $k$ índices primários candidatos de maior benefício relativo, cujo benefício ultrapasse um determinado *threshold*. Para cada um dos índices primários candidatos selecionados cria-se uma visão materializada ordenada pelos atributos que compõem a chave do índice. Essas visões materializadas são denominadas *clusters* alternativos de dados.

Desta forma, a manutenção automática de *clusters* alternativos de dados consiste em prover uma solução que continuamente mantenha (crie, remova e reorganize) um conjunto de *clusters* alternativos de dados adequado, ou seja, que assegure um desempenho aceitável para a carga de trabalho submetida ao SGBD. Contudo, apesar de sua grande importância, este é um problema ainda pouco estudado na literatura.



### 5.2.2 Fase de Observação: Monitoramento da Carga de Trabalho e Atualização do Benefício dos *Clusters* Alternativos de Dados

Esta fase é idêntica à fase de observação do estudo de caso anterior. Nela, mediante consultas à metabase do SGBD utilizado, capturam-se, periodicamente[9], as cláusulas SQL que foram ou estão sendo executadas, juntamente com seus respectivos planos de execução e estimativas de custo. Estas informações são armazenadas na metabase local (*LM*, Capítulo 6).

Em seguida, a carga de trabalho capturada é analisada com a finalidade de se identificar os índices mais adequados e seus respectivos benefícios para carga de trabalho como um todo. Para isso, utiliza-se a heurística integrada para seleção e acompanhamento de índices ($HISAI$), definida no estudo de caso anterior e apresentada nas Figuras 5.1, 5.2 e 5.3.

#### Estratégia para Avaliação de Atualizações

Como discutido na Seção 5.2, um comando SQL de atualização (update, insert, delete) sobre uma determinada tabela $t$ pode implicar na necessidade de se atualizar as visões materializadas que utilizam $t$ como tabela base. Neste caso, o custo de atualização da visão materializada (ou *cluster* alternativo de dados) é considerado como um "malefício" (ou seja, um benefício negativo). A estratégia utilizada para calcular e atualizar o benefícios dos *clusters*

[9]Essa periodicidade pode ser parametrizada pelo DBA.

alternativos que precisaram ser alterados em virtude de um comando SQL de atualização é a mesma definida no estudo de caso anterior (Figura 5.3).

### 5.2.3 Fase de Predição: Seleção *On-The-Fly* de *Clusters* Alternativos de Dados

Esta fase é iniciada sempre que o benefício acumulado de um *cluster* hipotético $k$ qualquer ultrapassar o seu custo estimado de criação ($BA_k > EC_{C_k}$, onde $EC_{C_K} = 2P$ e $P$ é o número de páginas da relação $R$), indicando que a existência do *cluster* contribuiria para diminuir o custo de processamento da carga de trabalho submetida ao SGBD, ou quando o benefício acumulado de um *cluster* real $k^{'}$ alcança um valor negativo que supera, em módulo, o seu custo de criação ($BA_{k'} < 0$ e $|BA_{k'}| > EC_{C_k}$), o que indica que a existência de $k^{'}$ não está contribuindo para diminuir o custo de processamento da carga de trabalho.

O problema a ser resolvido nesta fase consiste basicamente em buscar obter uma configuração de *clusters* alternativos ótima, levando em consideração as restrições de espaço físico. Para isso, utilizaremos as informações coletadas durante a fase de observação, as quais foram são armazenadas na metabase local (definida no Capítulo 6).

Seja $IP$ o conjunto dos índices primários candidatos (incluindo índices primários hipotéticos e reais) e $CA$ o conjunto dos *clusters* alternativos (incluindo *clusters* hipotéticos e reais). Para cada índice $i \in IP$ crie um *cluster* alternativo hipotético $c$ cujo critério de ordenação coincida com a chave de $i$. Além disso, faça $BA_c \leftarrow BA_i$ e estime $P_c$. Se $c \notin CA$ faça $CA \leftarrow CA \cup \{c\}$. Assuma que a variável *storage_constraint* representa o espaço disponível para a materialização das estruturas dos *clusters* alternativos.

A solução adotada utiliza um algoritmo guloso adaptado de (Luh07) (Figura 5.8). Neste sentido, todos os *clusters* (reais e hipotéticos) são ordenados de acordo com o seu benefício relativo, conforme definido a seguir:

$$BR_c = \frac{BA_c}{P_c}.$$

Em seguida, o algoritmo apresentado na figura 5.8 é executado e produz como saída uma nova configuração de *clusters* $\bar{C}$, ou seja, um novo conjunto de *clusters* alternativos (construídos a partir de visões materializadas) 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 os mesmos *clusters* sejam freqüentemente adicionados e removidos, uma nova

```
IP_s[1...n] ← ordenar(IP) através do benefício relativo
CA_s[1...n] ← ordenar(CA) através do benefício relativo
C̄ ← φ
espaço_disponível ← storage_constraint
benefício_total ← 0
Para todo k ← 1 ··· n faça
  Se (C̄ ⊃ I_s[j]) e (I_s[j] e I_s[k] definidos sobre a mesma tabela t) e k ≠ j então
    Se espaço_disponível - P_{CA_s[k]} > 0 então
      C̄ ← C̄ ⋃ CA_s[k]
      espaço_disponível ← espaço_disponível - P_{CA_s[k]}
      benefício_total ← benefício_total + BA_{CA_s[k]}
    Fim Se
  Se não
    Se espaço_disponível - P_{IP_s[k]} > 0 então
      C̄ ← C̄ ⋃ IP_s[k]
      espaço_disponível ← espaço_disponível - P_{IP_s[k]}
      benefício_total ← benefício_total + BA_{IP_s[k]}
    Fim Se
  Fim Se
Fim Para
Se benefício_total < threshold então
  C̄ ← C
Fim Se
retorne C̄
```

Figura 5.8: Algoritmo Guloso para Seleção de *Clusters* Alternativos de Dados.

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 (tipicamente $> 1.0$, como sugerido em (Luh07)). Logicamente, esta abordagem não assegura uma solução ótima.

### 5.2.4 Fase de Reação: Criação e Remoção de *Clusters* Alternativos de Dados

Ao final da fase de predição, caso seja detectada a necessidade de mudanças na configuração de índices ($\bar{C} \neq \phi$), a fase de reação é iniciada. Neste caso, deve-se substituir a configuração de *clusters* alternativos de dados atual $C$ por uma uma nova configuração $\bar{C}$. Assim, se um determinado *cluster* alternativo $k \in C$ e $k \notin \bar{C}$, então $k$ deve ser removido. Todavia, o *cluster* $k$ continua existindo na metabase local, porém, seu estado é alterado de real para hipotético e $BA_k = 0$. Logo, nada impede que, futuramente, este *cluster* seja novamente materializado. Note que o custo de excluir um *cluster* pode ser desprezado, uma vez que esta tarefa consiste basicamente em liberar espaço em disco e remover uma entrada do catálogo do SGBD, além da atualização do estado e do benefício acumulado deste *cluster* na metabase local. Por outro lado, todo *cluster* alternativo $k$ tal que $k \in \bar{C}$ e $k \notin C$ deve ser materializado,

```
Para todo cluster k ∈ C̄ faça
  Se k ∉ C então
    Criar o cluster k
    state(k) ← real
  Fim Se
Fim Para
Para todo cluster k ∈ C faça
  Se k ∉ C̄ então
    Remover o cluster k
    state(k) ←hipotético
    BA_k ← 0
  Fim Se
Fim Para
retorne C̄
```

Figura 5.9: Algoritmo para Atualização da Configuração de *Clusters* Alternativos de Dados.

ou seja, fisicamente criado. A Figura 5.9 mostra o algoritmo concebido para atualizar a configuração dos *clusters* alternativos.

## 5.3 Resumo do Capítulo

Neste capítulo, apresentamos dois estudos de caso, os quais ilustram como instanciar a abordagem não-intrusiva proposta para solucionar dois importantes problemas relacionados ao projeto físico de banco de dados: a manutenção automática das estruturas de índices e a manutenção automática de *clusters* alternativos. No primeiro estudo de caso, a instanciação elaborada apresentou as seguintes inovações: uma heurística integrada para a seleção e acompanhamento de índices (Heurística *HISAI*), um conjunto de algoritmos para seleção e atualização de uma configuração de índices, a consideração tanto de índices primários quanto secundários, uma avaliação da pré-criação de índices envolvidos em chaves primárias e estrangeiras, o acompanhamento do nível de fragmentação das estruturas de índice, a utilização dos conceitos de relevância e generalidade de índices, bem como do conceito de estabilidade de uma configuração de índices. Já o segundo estudo de caso utilizou visões materializadas para construir *clusters* alternativos de dados e permitir diferentes ordenações físicas para uma mesma tabela (ou fonte de dados). Contudo, a abordagem foi aplicada individualmente a cada um desses dois tipos de estruturas de acesso: índices e *clusters* alternativos de dados. Vale destacar, porém, que seria mais indicada uma solução conjunta, a qual poderia ser realizada mediante a negociação entre os agentes de *software* responsáveis pela manutenção de cada tipo de estrutura de acesso individualmente, por intermédio da uti-

lização de uma função multiobjetivo ou ainda por algum algoritmo aplicado ao clássico problema da mochila compartimentada.

No próximo capítulo, será apresentada uma arquitetura baseada em agentes de *software*, concebida e utilizada para fornecer suporte para a instanciação da abordagem não-intrusiva proposta nesta tese.