4
Reconstrução Automática de Índices

A Heurística de Benefícios possui dois pilares: enumeração de índices candidatos e acompanhamento de índices, sejam hipotéticos ou não, mantendo-lhes uma determinada carga de benefícios (positiva ou negativa), calculada a partir de participações em comandos.

Uma vez detectado um benefício acumulado negativo, cujo módulo supere o custo de criação do índice, dispara-se a eliminação do mesmo. Ocorre, entretanto, que, muitas vezes, um índice poderia continuar sendo útil, uma vez reconstruído. A Heurística de Reconstrução Automática de Índices analisa casos de eliminação iminente e, caso julgue interessante, dispara a reconstrução do índice.

O restante do capítulo está organizado da seguinte maneira: na próxima seção descrevem-se as bases de sustentação da heurística. Nas seções 4.2 e 4.3 discutem-se dois elementos que sustentam a concretização da heurística deste capítulo: o mecanismo que permite analisar níveis de fragmentação de um índice e a relação entre fator de preenchimento de páginas folha e estes níveis de fragmentação. Em 4.4 discute-se o funcionamento da heurística, em alto nível. Já em 4.5 focam-se os detalhes de implementação com diagramas de classe e seqüência. A seção 4.6 detém-se nas interações entre os componentes de software responsáveis pelas implementações das heurísticas de Benefícios e Reconstrução Automática de Índices. Finalmente, a penúltima seção apresenta resultados experimentais. Conclui-se o capítulo em 4.7 tecendo comentários gerais sobre os assuntos discutidos.

4.1 Bases da Heurística de Reconstrução Automática de Índices

A Heurística de Reconstrução Automática de Índices estabelece regras que permitem decidir se um índice deve ser reconstruído ou eliminado. Vale ressaltar que, por enquanto, apenas tratam-se índices possuindo organização em árvore B+ (Btree) [12]. Como índices obedecendo a esta organização prevalecem nos

PUC-Rio - Certificação Digital Nº 0410827/CA

principais SGBDs, julgou-se inoportuna a extensão da heurística para que sejam contemplados índices construídos com organizações alternativas.

À medida que acontecem alterações nos dados de uma tabela, causadas pelos comandos *insert*, *delete* ou *update*, os índices associados à tabela sofrem os malefícios da fragmentação [28]. A Figura 4.1 apresenta o estado do nível folha de um índice recém criado e, portanto, não fragmentado.

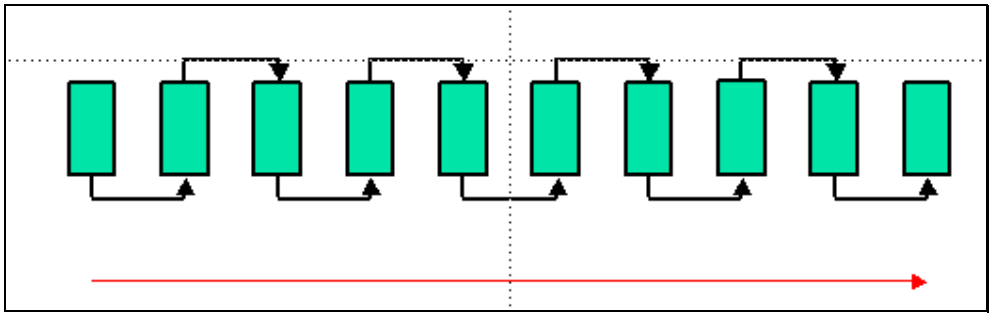

Figura 4.1: nível folha de um índice recém criado.

A Figura 4.1 revela um conjunto de páginas, cuja leitura ocorre no sentido da seta longa. A ordem de alocação aconteceu seguindo-se as setas pequenas. Deve-se observar que não existe espaço livre dentro de cada página, portanto, caso exista necessidade de inserção de uma chave em qualquer ponto que não seja após a última página, ocorrerá uma cisão em uma página interna (*page split*)

Normalmente, o nível de compactação apresentado na Figura 4.1 também estende-se ao restante da árvore (raiz e nível não-folha). Recomenda-se esta alta densidade a índices cujas tabelas onde foram criados não sofram atualizações. A Figura 4.2 revela as conseqüências de uma alteração (*insert* ou *update*) no mesmo índice apresentado na Figura 4.1 após a ocorrência de um *page-split*.

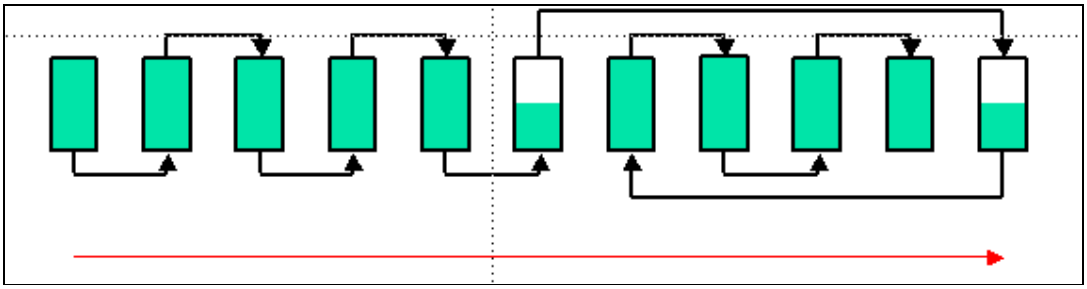

Figura 4.2: nível folha de um índice que sofreu um *page-split*.

PUC-Rio - Certificação Digital Nº 0410827/CA

Operações que realizem varreduras, isto é, percorrem o nível folha completamente, têm seu desempenho degradado à medida que aumenta a incidência de *page splits*, pois as páginas necessárias estão espalhadas. Imediatamente após a criação, um índice não apresenta grau de fragmentação, porém, à medida que acontecem *page splits*, este grau cresce. A Figura 4.3 apresenta um índice muito fragmentado.

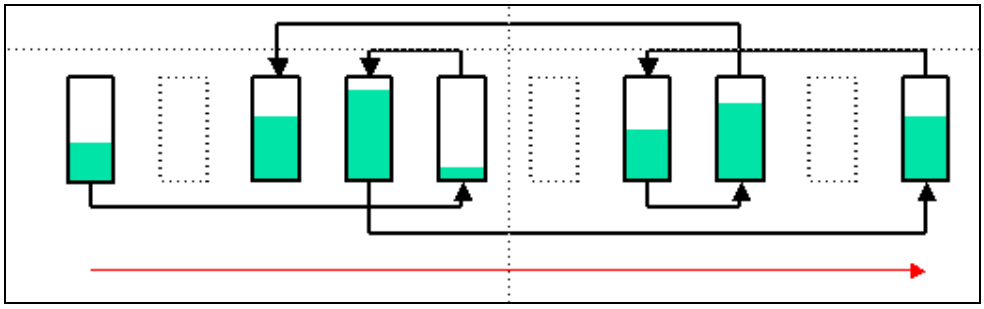

Figura 4.3: nível folha de um índice com alto grau de fragmentação.

A decisão quanto à recriação de um determinado índice deve analisar três fatores: grau de fragmentação, tamanho e varreduras.

Grau de Fragmentação

A medida do grau de fragmentação de um índice poderia ser obtida comparando-se a razão:

```
R = tuplas de uma tabela / quantidade de blocos do índice
```

em dois momentos: logo após a criação, quando não existe fragmentação, e quando deseja-se verificar se vale a pena aplicar uma reconstrução, ou simplesmente seu descarte.

Propomos a seguinte fórmula para cálculo do grau de fragmentação de um índice:

$$G_{rF} = 100 - [(R_a / R_i) * 100]$$

Figura 4.4: fórmula que obtém o grau de fragmentação de um índice

PUC-Rio - Certificação Digital Nº 0410827/CA

Onde R_a representaria a razão atual e R_i a razão inicial, gerada logo após a criação do índice.

Por exemplo, identificou-se um determinado índice, **ir1**, criado sobre a tabela **Venda** (descrita na Figura 3.1) com 879 blocos comportando 400.000 tuplas (R_i = 455,06). Após várias operações, a quantidade de blocos aumentou para 2.442 em 450.000 tuplas (R_a = 184,27). Desta forma, calculou-se $\mathbf{G_{rF}}$ como 59,50%

Tamanho

Tabelas com poucos blocos não devem ter seus índices cogitados à reconstrução. Por exemplo, imagine a tabela **T1**, 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 uma considerável queda em sua razão tuplas/blocos, a reconstrução do índice jamais proporcionaria menos de três blocos.

Varreduras

Consultas envolvendo operações onde várias páginas precisem ser lidas de forma contígua devem ter seu desempenho degradado, caso utilizem índices fragmentados. Por exemplo, dada a tabela denominada **Venda**, cuja estrutura foi mostrada na Figura 3.1, se as páginas da tabela contendo os intervalos requeridos estiverem fisicamente distantes entre si, o tempo de resposta da consulta descrita na Figura 4.5 não alcançará aquele obtido, caso o índice sobre a coluna **num** não esteja fragmentado.

```
select sum(valor) from venda
where num between 8 and 64008
or num between 12800000 and 12864000
or num between 28800000 and 28864000
or num between 44800000 and 44864000
or num between 60800000 and 60864000;
```

Figura 4.5: consulta efetuando várias varreduras sobre a tabela **Venda**.

PUC-Rio - Certificação Digital Nº 0410827/CA

O fato de um índice ter sido útil em uma consulta onde houve varreduras já faz com que seja um candidato à reconstrução.

Deve-se ressaltar que índices fragmentados não degradam o desempenho de consultas que não realizem varreduras, já que a duração de acessos pontuais tende a ser a mesma.

4.2 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.

Em Oracle, 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. Por exemplo, atribuindo-se o valor 20 à cláusula **pctfree**, significa que, enquanto um bloco não atingir 80% de ocupação, continuará a receber novas chaves. Uma vez atingido este limite, os 20% restantes somente poderão ser utilizados como conseqüência de atualizações de chaves que já estejam presentes. Tabelas muito atualizadas devem possuir índices com **pctfree** grande.

Em SQL Server, a cláusula denomina-se **fillfactor** e determina o percentual de preenchimento das páginas do nível folha. Quanto menor o valor especificado, maior a folga por página. Tabelas muito atualizadas devem possuir índices com **fillfactor** pequeno.

PostgreSQL possui o conceito, porém resulta impossível ao DBA configurar um fator de preenchimento próprio. O valor está especificado no código fonte em 90% para páginas nível folha e 70% para as demais.

A Heurística de Reconstrução Automática de Índices 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. Como utilizou-se 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, **fillfactor**, valendo entre 1 e 9. O menor valor

PUC-Rio - Certificação Digital Nº 0410827/CA

significa que apenas um décimo de cada página será ocupada, enquanto que o maior sinaliza 90% de ocupação.

4.3 GETSIZE

Um índice cuja razão blocos_índice/tuplas_tabela (**R**) apresente valores distantes daqueles medidos imediatamente após sua criação, pode ser considerado como fragmentado. Por exemplo, estando a tabela **Venda** carregada com 400.000 tuplas, e existindo um índice denominado **ix_num** sobre o atributo **num,** pode-se conhecer as quantidades de tuplas e blocos, recorrendo-se à metabase.

A Figura 4.6 mostra um comando de consulta à metabase, e seu resultado, disparado no SGBD PostgreSQL, que revela quantidades de páginas e tuplas da tabela **Venda**.

```
select relname, reltuples, relpages from pg_class
where relname in ('ix_num','venda') ;


 relname | reltuples | relpages
---------+-----------+----------
 ix_num  |    400000 |      879
 venda   |    400000 |     3147
```

Figura 4.6: investigação do tamanho de uma tabela em PostgreSQL

Analisando-se o resultado da consulta, chega-se ao valor de $\mathbf{R_i}$: 455,06, obtido pela razão: 400.000/879. Assim como em qualquer SGBD, a confiabilidade dos valores da metabase depende da atualização periódica das estatísticas. Isto fica evidente após uma nova consulta à metabase, uma vez executado o comando da Figura 4.7.

```
update venda set num = num + 800000;
```

Figura 4.7: comando fragmentando índice sobre o campo **num**.

Nova consulta à metabase mostraria as mesmas 879 páginas iniciais, entretanto, uma vez atualizadas as estatísticas, o resultado muda para a saída ilustrada na Figura 4.8

PUC-Rio - Certificação Digital Nº 0410827/CA

```
relname | reltuples | relpages
---------+-----------+----------
 ix_num  |    400000 |     2097
 venda   |    400000 |     3147
```

Figura 4.8: novos valores obtidos em consulta à metabase, após fragmentar o índice.

Observa-se que tuplas (400.000) e páginas (3.147) da tabela estão corretas (idênticas à primeira consulta), porém o mesmo não se poderia afirmar das páginas do índice. Infelizmente, o *update* provocou grande crescimento no índice, dada uma característica do PostgreSQL de não alocar páginas, possuindo chaves com valores distantes do original, fisicamente próximas das originais. Por exemplo, a tupla cujo conteúdo do atributo **num** era de 1 passou a ser 800.001. Esta mudança ocasionou a alocação de uma nova página em local físico afastado daquela página que possuía a chave 1.

As decisões quanto às utilizações de índices realizadas pelo otimizador de qualquer SGBD são baseadas nas informações que constam na metabase. Caso esta não seja atualizada periodicamente, corre-se o risco de acontecerem otimizações errôneas. Para que não exista o risco de obterem-se dados incorretos, a implementação da Heurística de Reconstrução Automática de Índices precisou comprovar as quantidades reais de tuplas e páginas de tabelas e índices, por meios próprios.

Uma forma alternativa de comprovar a quantidade real de páginas e tuplas de uma tabela no PostgreSQL consiste em utilizar uma ferramenta opcional denominada **pgstattable**. Trata-se de uma função que checa todas as páginas da tabela, informando quantidades reais. A Figura 4.9 mostra um exemplo de chamada à função e seu resultado.

```
select tuple_count, table_number_blocks from pgstattable
('venda');


tuple_count  | table_number_blocks
-------------+--------------------+---
      400000 |               3147
```

Figura 4.9: exemplo de utilização da função **pgstattable**.

Como não se informam dados de índices, foi necessário, com base em **pgstattable**, desenvolver uma nova ferramenta, **pgstatindex**, que também checa

PUC-Rio - Certificação Digital Nº 0410827/CA

todas as páginas de um dado índice. A Figura 4.10 mostra um exemplo de chamada à função e seu resultado.

```
select tuple_count, table_number_blocks, index_len,
index_number_blocks, index_tuple_count
from pgstatindex ('venda', 'ix_num');

tuple_count|table_number_blocks|index_len|index_number_blocks|index_tuple
_count
-----------+-------------------+---------+-------------------+-----------
------
400000       |       3147          |17178624 |        2097          |
800000
```

Figura 4.10: exemplo de utilização da função **pgstatindex**.

A nova função, **pgstatindex,** serviu de inspiração para o desenvolvimento de um novo comando para o PostgreSQL, **getsize**. Dado um nome de índice, o comando checa todas as tuplas da tabela correspondente e conta as páginas do índice. Estes dois dados permitirão calcular as duas razões $\mathbf{R_i}$ (logo após sua criação) e $\mathbf{R_a}$ (uma vez detectada a possibilidade de eliminá-lo). Com as informações retornadas por **getsize** pode-se derivar dois dos três fatores necessários à decisão de reconstruir, ou não, um índice. Bastaria apenas o histórico de varreduras, que será informado pelo Agente de Benefícios, como detalhado na próxima seção.

Uma aplicação imediata da função **pgstatindex**, consistiu em verificar a eficácia da nova cláusula **fillfactor**, presente nos comandos **create index, reindex index** e **reindex table**.

A Figura 4.11 mostra a criação de um índice com fator de preenchimento 50% sobre a tabela **Venda**, carregada com 400.000 tuplas. Também aparece a chamada à função **pgstatindex** e seu resultado.

```
create index ix_num on venda (num) fillfactor 5;

select tuple_count, table_number_blocks, index_len, index_number_blocks,
index_tuple_count from pgstatindex ('venda', 'ix_num');

tuple_count  |table_number_blocks|  index_len  |  index_number_blocks  |
index_tuple_count
-------------+--------------------+-----------+---------------------+---
------------      400000 |            3147 |   13017088 |           1589 |
400000
```

Figura 4.11: criação de um índice parcialmente preenchido

PUC-Rio - Certificação Digital Nº 0410827/CA

Deve-se comparar a diferença entre as quantidades de páginas de índices criados com fatores de preenchimentos distintos. A Figura 4.11 mostra um índice criado com fator de preenchimento pequeno (5), portanto a quantidade de páginas (1.589) resulta bastante maior do que o mesmo índice recriado com fator de preenchimento 9 (879), como apresentado na Figura 4.12.

```
reindex index ix_num fillfactor 9;

select tuple_count, table_number_blocks, index_len, index_number_blocks,
index_tuple_count from pgstatindex ('venda', 'ix_num');

tuple_count|table_number_blocks   |   index_len   |   index_number_blocks   |
index_tuple_count
-------------+--------------------+-----------+---------------------+---
------------      400000 |              3147   |            7200768      |
879 |            400000
```

Figura 4.12: recriação de um índice utilizando fator de preenchimento maior

Observar que decréscimo na quantidade de páginas fará com que varreduras aconteçam em menos tempo, já que há menos páginas por percorrer, entretanto, futuras atualizações causarão *page splits*, o que prejudicará o desempenho destas mesmas consultas no futuro.

4.4 Heurística de Reconstrução Automática de Índices

As ferramentas de apoio a DBAs disponíveis atualmente no mercado, dentre as funcionalidades voltadas à gerência de índices, limitam-se à seleção, carecendo de recomendações quanto à reconstrução de índices fragmentados que tenham contribuído de forma positiva em comandos anteriores. O Apêndice B desta dissertação apresenta um conjunto representativo de ferramentas comerciais de apoio ao DBA, mas que não recriam índices.

Desta forma, foi necessário desenvolver uma heurística própria que fosse capaz de reconstruir índices fragmentados. Assim como em [32], a heurística será executada simultaneamente com outros serviços providos pelo SGBD.

Para apresentar a heurística, devem-se definir cinco fatores:

- T_t: tamanho em tuplas de uma determinada tabela.
- T_i: tamanho em blocos de um determinado índice.
- R: razão entre T_t e T_i.

PUC-Rio - Certificação Digital Nº 0410827/CA

- G_{rF}: grau de fragmentação de um determinado índice, obtido através da fórmula descrita na Figura 4.4.
- V: número de varreduras nas quais o índice participou desde sua criação.

A heurística é composta pelas funcionalidades:

i. Novo Índice: deve-se armazenar o nome de um índice recém criado;

ii. Solicitação de Tamanho: solicita-se ao SGBD T_i e T_t referentes a um determinado índice;

iii. Cálculo de Razão: obtém-se R com base em T_i e T_t.

iv. Solicitação de Varreduras: pede-se V de um dado índice;

v. Cálculo de Grau de Fragmentação: obtém-se G_{rF} com base em R_i e R_a;

vi. Reconstrução: caso G_{rF}, V e T_i atendam a determinados limites (*thresholds*), efetua-se a reconstrução.

A solicitação de tamanho (segunda funcionalidade), bem como o cálculo da razão (terceira) ocorrem, para cada índice armazenado, em dois momentos: logo após a criação (R_i) e previamente à decisão de reconstruir ou não (R_a). A solicitação tem por conseqüência a execução do comando **getsize** pelo SGBD.

Antes da reconstrução, deve-se calcular o novo fator de preenchimento segundo a fórmula apresentada na Figura 4.13. 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.

F = 10 - [(V + 10) div 10]

Figura 4.13: fórmula para obtenção do fator de preenchimento

PUC-Rio - Certificação Digital Nº 0410827/CA

Por exemplo, supondo que um determinado índice tenha sofrido 34 varreduras, seu novo fator de preenchimento seria:

F = 10 – [(34 + 10) div 10]

F = 10 – (44 div 10)

F = 6

Portanto, o índice terá suas páginas em nível folha preenchidas até 60%

4.5 Estrutura Funcional

O componente de software responsável pela implementação das idéias propostas pela Heurística de Reconstrução Automática de Índices denomina-se Agente Desfragmentador. Sua construção seguiu a mesma linha de estudos utilizada em [32] e [26], onde foram adaptadas propostas presentes em [KPP+99] (Figura 2.3).

O Agente Desfragmentador interage com o Agente de Benefícios, cuja implementação estendeu os trabalhos presentes em [32]. Enquanto o Agente de Benefícios cria e destrói índices, o Agente Desfragmentador preocupa-se com reconstruções. As interações entre os agentes aparecem na próxima seção.

Nesta seção apresenta-se uma descrição detalhada dos papéis desempenhados em cada camada do Agente Desfragmentador, utilizando diagramas de Classes, segundo a notação UML [41].

A Figura 4.14 mostra as classes envolvidas na configuração de camadas do Agente Desfragmentador.

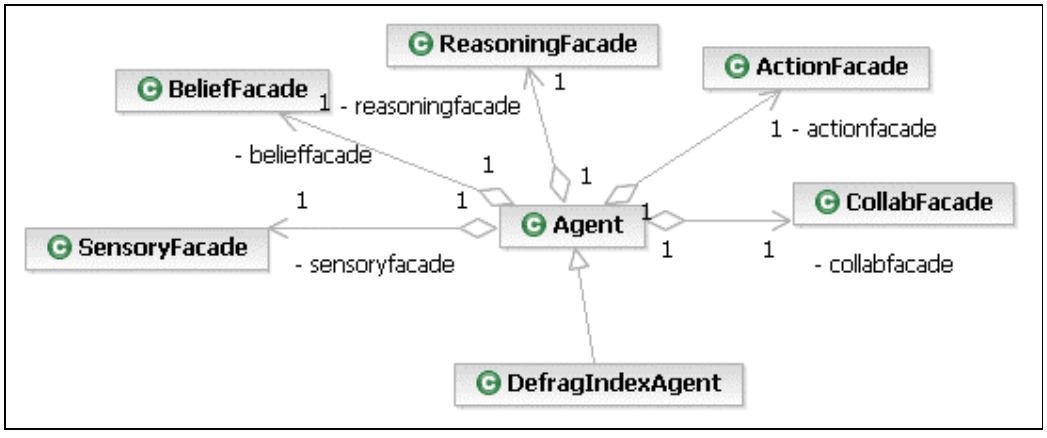

Figura 4.14: diagrama de classes envolvidas na configuração das camadas

PUC-Rio - Certificação Digital Nº 0410827/CA

Ao iniciar o PostgreSQL, cria-se uma instância da classe *DefragIndexAgent*, representando o Agente Desfragmentador. A implementação da ferramenta de criação, eliminação (e agora, recriação) de índices prevê sua ativação via parâmetros de inicialização do PostgreSQL.

Ao estender a classe abstrata *Agent*, *DefragIndexAgent* herda atributos e métodos das classes associadas às camadas do agente (*SensoryFacade, BeliefFacade, ReasoningFacade, ActionFacade* e *CollabFacade*). A seguir, apresentam-se os detalhes de cada camada.

Camada Sensor

Responsável por checar permanentemente a chegada de uma estrutura de dados com nome de um índice e as quantidades reais de páginas dele e tuplas da tabela à qual pertence. Esta camada captura a resposta do PostgreSQL à execução do comando **getsize** (disparado na Camada *Ação*). Uma vez acusada a recepção, encaminham-se os dados (nome índice, páginas do índice, tuplas da tabela) à camada *Crença*.

A Figura 4.15 mostra as classes presentes na camada Sensor. As classes *BeliefFacade* e *ABelief* pertencem à camada Crença e são mostradas para melhorar a compreensão. A classe *IndexSizeInformation* também não pertence à camada Sensor, porém deve ser mostrada pois encapsula as informações associadas à resposta do comando **getsize**.

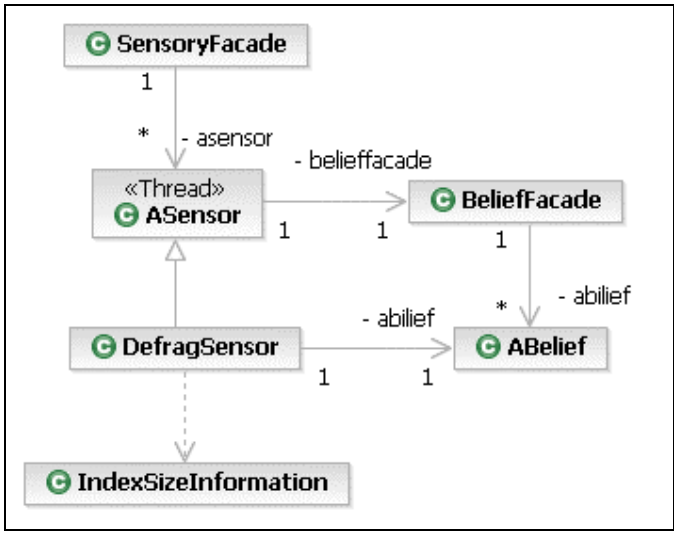

Figura 4.15: diagrama de classes na camada *Sensor*

PUC-Rio - Certificação Digital Nº 0410827/CA

Camada Crença

Armazena os conhecimentos obtidos pelo agente. Na Figura 4.16 mostram-se as classes presentes na camada *Crença*. A classe *ReasoningFacade* pertence à camada Raciocínio e aparece apenas para melhorar a compreensão. As classes *Subject* e *Observer* têm o propósito de viabilizar a notificação entre camadas.

A classe *DefragBelief* armazena o último objeto *IndexSizeInformation* proveniente da camada *Sensor* e o encaminha à camada *Raciocínio.*

A classe *DefragPreviousBelief* registra dados históricos que permitirão a verificação da viabilidade da reconstrução de um índice. Há um atributo-vetor, **_indexSet**, onde armazenam-se informações referentes aos índices registrados: nomes do índice e da tabela à qual pertence; atributos numéricos representando as quantidades inicial e final de blocos do índice e tuplas da tabela; e um indicador que revela se o índice pode ser eliminado ou não.

Além desse vetor, ainda guardam-se atributos numéricos: **_indexCount**, **_lastIndexReceived** (posição do último índice motivo de notificação por parte do Agente de Benefícios), **_lastOperation** (0: eliminação; 1: primeira utilização) e **_scans** (número de varreduras sofridas pelo índice notificado).

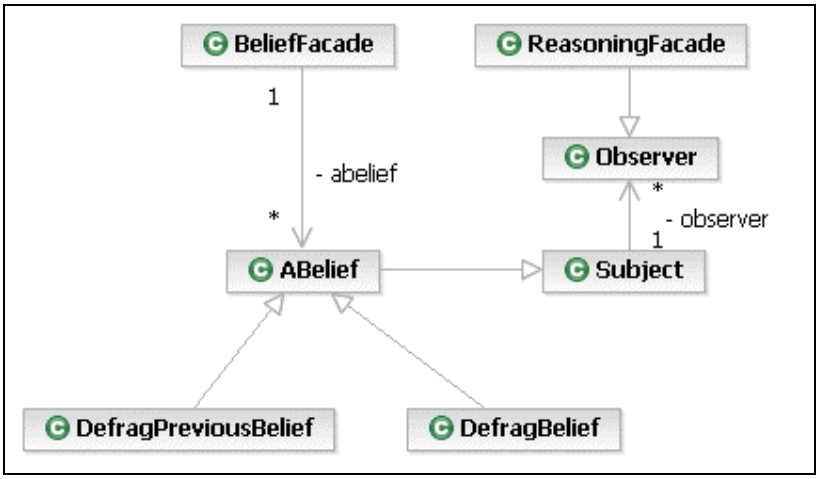

Figura 4.16: diagrama de classes na camada *Crença*

PUC-Rio - Certificação Digital Nº 0410827/CA

Camada Raciocínio

Local onde concentra-se a inteligência do agente, já que responsabiliza-se pela análise dos dados da camada *Crença* e tomada de decisões que podem transformar-se em instruções à camada *Ação*. Seu componente único, responsável pela decisão de solicitar a reconstrução de um índice ou permitir sua destruição, recebe notificações da camada *Crença*, quando o sensoriamento captura o resultado de um **getsize**, ou da camada *Colaboração*, quando houver notificações do Agente de Benefícios.

A Figura 4.17 mostra as classes presentes na camada *Raciocínio*. A classe *Aplan* representa o plano que pode ser aplicado à camada *Ação.* A classe *DefragPlan* implementa os detalhes da Heurística de Reconstrução Automática de Índices, descrita na seção anterior. Nela tomam-se decisões, tais como solicitação de tamanho de um determinado índice (tuplas de tabela e páginas de índice) ou requerimento para que um determinado índice seja reconstruído.

ActionFacade e *ActionPlan* pertencem à camada *Ação* e são mostradas para melhorar a compreensão.

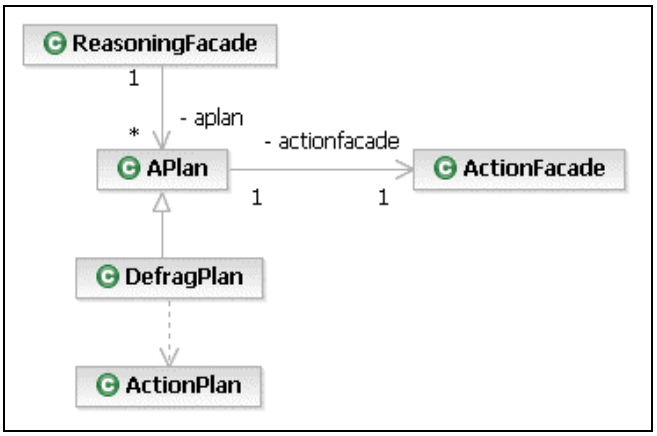

Figura 4.17: diagrama de classes na camada *Raciocínio*

Camada Ação

Realiza dois procedimentos possíveis, a partir de requisições da camada *Raciocínio*:

- Obtenção de quantidades reais de blocos de um índice e tuplas da tabela à qual pertence;

PUC-Rio - Certificação Digital Nº 0410827/CA

- Reconstrução de um índice levando em conta um fator de preenchimento de blocos decidido pela camada Raciocínio, com base na quantidade de varreduras que o índice sofreu até então.

Na verdade, as ações concretizam-se nos domínios do SGBD. O Agente simplesmente requisita que estas aconteçam.

A Figura 4.18 mostra as classes presentes na camada *Ação*.

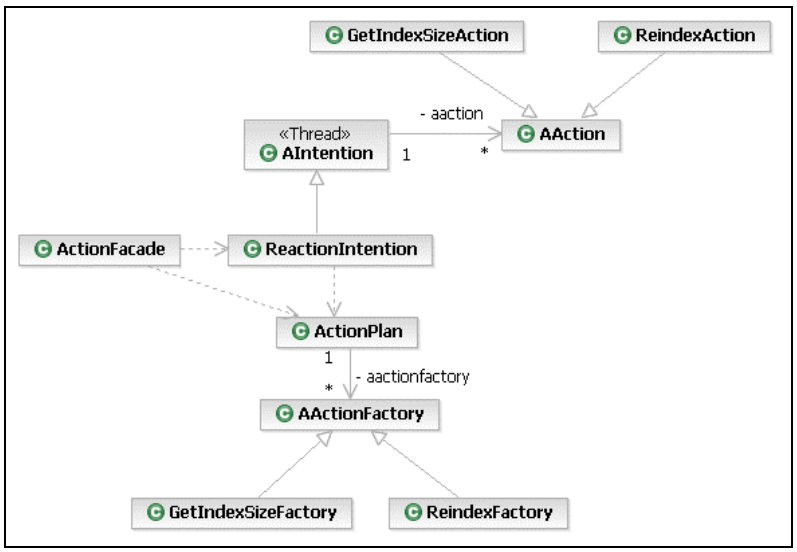

Figura 4.18: diagrama de classes na camada *Ação*

Camada Colaboração

Viabiliza a comunicação entre agentes materializada a partir de **serviços** (veja Tabela 4.1). O Agente Desfragmentador recebe dois estímulos, via esta camada, provenientes do Agente de Benefícios:

- Nome do índice recém criado;
- Nome do índice candidato à reconstrução e quantas varreduras ele sofreu.

O Agente Desfragmentador gera dois estímulos para o Agente de Benefícios:

- Proibição de eliminação;
- Liberação de eliminação.

A Figura 4.19 mostra as classes presentes na camada *Colaboração*. Para que dois agentes possam travar uma comunicação, deve existir um objeto à parte, caracterizado como *singleton* (somente há uma instância do objeto), nomeado em [26] como **PostOffice**. Na verdade, trata-se de uma classe acompanhada de outras

PUC-Rio - Certificação Digital Nº 0410827/CA

duas, **Connector**, responsável pela emissão de mensagens e **Acceptor**, encarregada pela recepção. A mensagem em si é transportada por outra classe, **Message**, onde registram-se emissor, receptor, tipo (informativa, requisição, resposta, etc.), conteúdo propriamente dito e forma de sincronismo (*broadcasting* ou ponto a ponto).

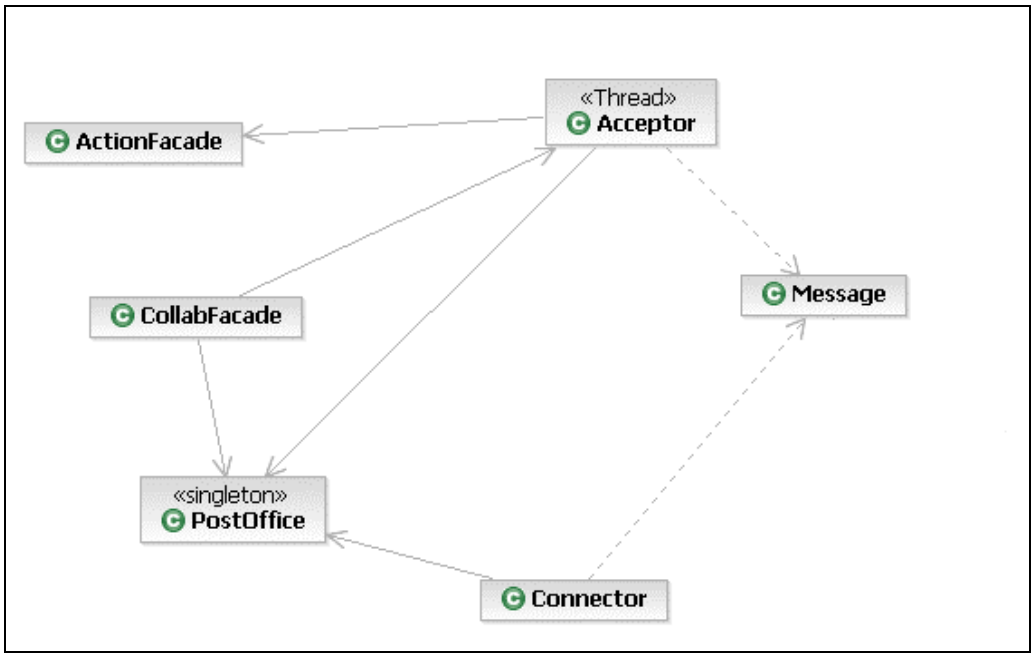

Figura 4.19: diagrama de classes na camada *Colaboração*

O conteúdo da mensagem estrutura-se em **serviço** e **complemento**. A primeira parte possui menção a alguma funcionalidade dos agentes. A Tabela 4.1 relaciona os serviços implementados para realizar a comunicação entre os agentes

Serviço	Descrição
First_Usage	Agente de Benefícios (AB) informa ao Agente Desfragmentador (AD) que um índice criado foi utilizado por primeira vez.
Drop	AB notifica AD sobre a intenção de eliminar um índice.
Allow	AD notifica AB que um determinado índice pode ser destruído.
Forbid	AD proíbe AB que um determinado índice seja destruído.
Rebuild	AD notifica AB que um determinado índice foi reconstruído.

Tabela 4.1: relação de serviços caracterizando a cooperação entre agentes.

A parte **complemento** de uma mensagem traz detalhes necessários à concretização do serviço. Por exemplo, em **first_usage** deve-se informar nomes de índice a ser criado e tabela à qual pertence.

PUC-Rio - Certificação Digital Nº 0410827/CA

As ocorrências de **first_usage** ou **drop** farão com que sejam disparadas a coleta de informações que, uma vez concluída, permite ao Agente Desfragmentador conhecer os níveis de fragmentação de um determinado índice. Eventualmente, este pode ser reconstruído.

A comunicação entre os agentes aparece ilustrada na Figura 4.20

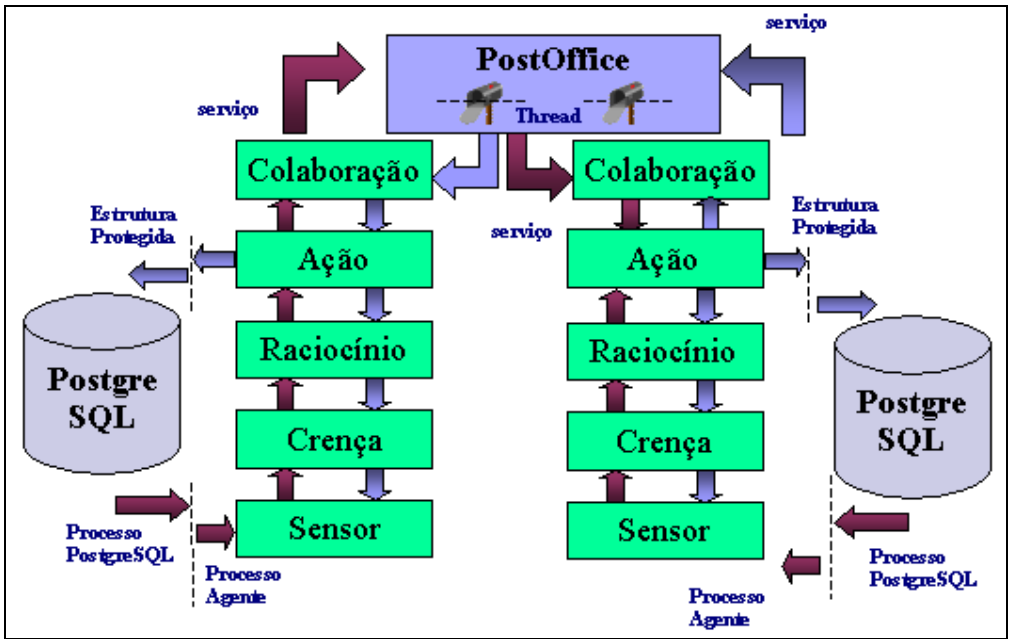

Figura 4.20: interação entre agentes via camada *Colaboração*

4.6 Interações entre Agentes

Nesta seção apresentam-se dois diagramas de Seqüências segundo a notação UML [41]. Ambos tratam apenas do serviço **first_usage**, listado na Tabela 4.1, onde o Agente de Benefícios informa ao Agente Desfragmentador que um índice foi criado. O primeiro diagrama (Figura 4.21) revela as interações desde a geração da notificação por parte do Agente de Benefícios; o segundo diagrama foca a recepção da mensagem e a execução do comando **getsize**.

Quando o Agente de Benefícios precisa informar ao Agente Desfragmentador que um índice deve ser registrado, toma-se a decisão na camada *Raciocínio*, classe *DifferenceHeuristicEvaluatorPlan*, que implementa a Heurística de Benefícios. Gerada a notificação (método **newIntention**), invocam-se os métodos **collaborate, send** e **put**, presentes na camada *Colaboração*.

PUC-Rio - Certificação Digital Nº 0410827/CA

Finalmente, registra-se a mensagem no objeto **PostOffice**, a partir do qual partirá para o agente destino.

A Figura 4.22 mostra a segunda parte da interação entre os agentes, revelando, desde a chegada da mensagem ao Agente Desfragmentador, até a execução do comando **getsize**. A mensagem chega via camada *Colaboração* (métodos **enqueue** e **get**); Repassada à camada *Ação*, cria-se uma notificação à camada *Raciocínio* (método **notify**). A camada *Crença* tem seus dados atualizados pelo método **update_from_reasoning** (classe *DefragPreviousBelief*). Este fato cria um aviso à camada *Raciocínio*, que deve avaliar se alguma ação deve ser disparada. Como trata-se de um novo índice (serviço **first_usage**), é necessário enviar um pedido à camada Ação para que o comando **getsize** seja disparado. A função **ag_GetIndexSize** faz parte da interface entre o SGBD e o agente.

A dinâmica revelada nos dois diagramas resulta idêntica para o serviço **drop**, já também resulta na execução de **getsize**. Tanto **allow** quanto **forbid** partem do Agente Desfragmentador; o primeiro tem por conseqüência a eliminação de um índice criado, enquanto que o segundo impede a remoção para que seja possível avaliar a possibilidade de reconstrução. Finalmente, **rebuild** informa o nome do índice que acabou de ser reconstruído.

PUC-Rio - Certificação Digital Nº 0410827/CA

Figura 4.21: Interação causada pelo serviço **first_usage**: primeira parte

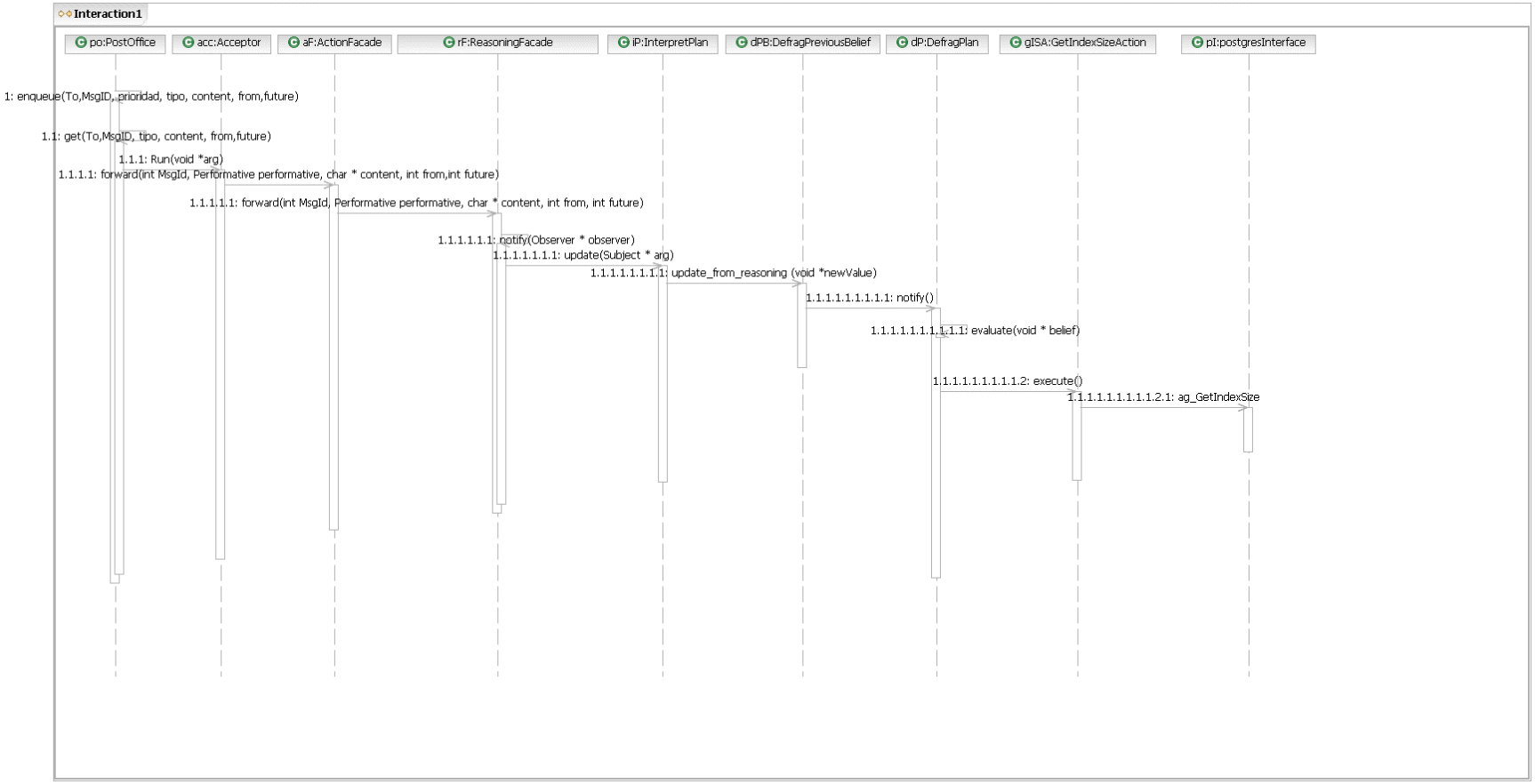

Figura 4.22: Interação causada pelo serviço **first_usage**: segunda parte

4.7 Implementações e Resultados Obtidos

Como não foi possível determinar valores ótimos para os fatores G_{rF}, V e T_i, preferiu-se deixá-los como parâmetros de configuração para serem ajustados livremente pelo DBA.

No caso específico do PostgreSQL, deve-se alterar o arquivo de inicialização denominado **postgresql.conf**. Nele aparecem as configurações necessárias ao funcionamento do PostgreSQL, tais como área em memória destinada para dados, nível de depuração do ambiente, padrões de comunicação entre clientes e servidor, etc. A Tabela 4.2 lista os novos parâmetros e sugere valores padrão.

Parâmetro	**Valor**	**Observações**
enable_agents	true	Dispara Agente de Benefícios.
enable_drop_indices	true	Permite destruição de índices reais pelo Agente de Benefícios.
enable_collaboration	true	Permite a comunicação entre agentes.
enable_alter_indices	true	Dispara Agente Desfragmentador.
default_fillfactor	9	Número inteiro representando o percentual de preenchimento de páginas de índices no nível folha. Varia de 1 a 9; 1 = 10%; 2 = 20%; ...; 9 = 90%.
dia_size[1]	800	Tamanho em blocos de 8KB, aquém do qual índices não devem ser considerados para efeitos de reconstrução.
dia_scans	2	Número mínimo de varreduras que um índice deve sofrer para ser passível de reconstrução.
dia_ratio	50	Grau de fragmentação (G_{rF}) mínimo para considerar reconstruções.

Tabela 4.2: valores de parâmetros relevantes de inicialização do PostgreSQL

Os parâmetros cujos nomes começam por **enable_** devem ser utilizados em forma crescente, como listado na Tabela 4.2, isto é, se **enable_drop_indices** tiver valor **false**, os que seguem (**enable_collaboration** e **enable_alter_indices**) também devem receber **false**. Esta característica foi introduzida para facilitar a depuração do funcionamento dos agentes.

PUC-Rio - Certificação Digital Nº 0410827/CA

O parâmetro **default_fillfactor** pode ser alterado a qualquer momento. Por exemplo, após emitir o comando:

```
Set default_fillfactor = 7;
```

Todo índice criado (**create index**) ou recriado (**reindex index**) que não utilize explicitamente a cláusula **fillfactor**, assumirá 70% de preenchimento de páginas.

Estrutura dos Testes

A bateria de testes utilizou uma base constituída por apenas uma tabela, **Venda** (descrita na Figura 2.3), com 400.000 tuplas. Procura-se observar o comportamento dos dois agentes, bem como a criação, destruição e recriação de índices.

Durante os testes aproveitou-se o comando **evaluate**, apresentado na seção 2.2, cuja forma geral aparece a seguir:

```
evaluate <complemento>
```

O complemento, que pode ser qualquer *select, update, delete* ou *insert*, é repassado ao Agente de Benefícios; assim, por conta da utilização do comando *evaluate*, ganha-se em agilidade, já que não se perde tempo com longas execuções.

A Tabela 4.3 revela uma sequência de comandos onde verificam-se todas as interações possíveis entre os agentes.

Instante	Comando	Status do Sistema
1	Concluída oitava execução do comando: `evaluate select prodnum, valor, data, qtd from venda where num = 10000;`	Criado índice **ri_venda_0** com benefícios acumulados de 65.151,9 e Bônus igual a 8.143,99. Durante as sete execuções prévias, existiu um índice hipotético denominado **hi_venda_0**.

Segue...

[1] O prefixo **dia** representa um acrônimo para *Defrag Index Agent*

PUC-Rio - Certificação Digital Nº 0410827/CA

Instante	Comando	Status do Sistema
2	Mais uma execução do comando acima.	Agente de Benefícios (AB) notifica Agente Desfragmentador (AD) via serviço **first_usage**; AD dispara **getsize**; B_i = 879 T_t = 400.000
3	Concluída décima segunda execução do comando: `evaluate select prodnum, valor, data, qtd from venda where num between 10000 and 10020;`	Como trata-se de uma consulta com varreduras, AB atualiza o atributo **scans** de **ri_venda_0** para 12.
4	Executado: `update venda set num = num + 800000;`	Como trata-se de uma atualização, AB decrementa os benefícios acumulados de **ri_venda_0**. Valor corrente: 160.730
5	Concluída vigésima primeira execução do comando: `evaluate update venda set num = num + 800000;`	Benefícios acumulados de **ri_venda_0**: -55.444,30. Custo de Eliminação: -65.151,90
6	Mais uma execução do comando acima. Mesmo que ocorra um comando que afete negativamente o índice, o Agente de Benefícios somente o destruirá uma vez recebida a liberação	AB notifica AD via serviço **drop**; AB informa Varreduras a AD (12). AD dispara **getsize**; AD notifica AB via serviço **forbid**; B_a = 2.097 T_a = 400.000. R_i = 455,06 R_a = 190,74; G_{rF} = 58,08 (Tabela 4.1)
7	Reconstruído índice **ri_venda_0** com **fillfactor** 8 (Figura 4.13)	AD notifica AB via serviço **rebuild**
8	Executado comando: `Select table_number_blocks, index_len, index_number_blocks from pgstatindex ('venda', 'ri_venda_0');`	Informada a nova quantidade de páginas de **ri_venda_0** (991), um pouco superior às 879 iniciais, devido ao **fillfactor** menor (8 ao invés de 9).
9	Executado comando: `evaluate select prodnum, valor, data, qtd from venda where num between 10000 and 10020;`	Benefícios acumulados de **ri_venda_0**: 72.295,90

PUC-Rio - Certificação Digital Nº 0410827/CA

Instante	Comando	Status do Sistema
10	Concluída nona execução do comando: `evaluate update venda set num = num + 800000;`	Benefícios acumulados de **ri_venda_0**: -92.524,30 Custo de Eliminação: -65.151,90. AB notifica AD via serviço **drop**; AB informa Varreduras a AD (12). AD dispara **getsize**; AD notifica AB via serviço **forbid**; $B_i = 991$ $T_i = 400.000$. $R_i = 455,06$ $R_a = 403,63$; $G_{rF} = 11,03$
11	Índice **ri_venda_0** destruído;	Como G_{rF} foi inferior ao parâmetro **dia_ratio**(50), AD notifica AB via serviço **allow**. AB cria Índice hipotético **hi_venda_0** com benefícios acumulados iguais a 92.524,10

Tabela 4.3: resumo da bateria de testes

A realização da bateria de testes resumida na Tabela 4.3 permitiu afirmar com segurança que a implementação da heurística obteve sucesso. Entretanto, ao elaborar situações mais complexas, tais como cargas de trabalho derivadas do *benchmark* TPC-C, não foi possível obter resultados satisfatórios. Para comprovar a eficácia da Heurística de Reconstrução Automática de Índices, deveria ser criada uma segunda bateria de testes, talvez uma extensão dos testes apresentados na seção 3.5, onde seriam apresentados resultados prévios e posteriores às criações, eliminações e recriações de índices. Infelizmente, ainda não foi possível chegar a tais resultados, devido às dificuldades inerentes às atividades de configuração do ambiente propício à realização dos testes.

4.8 Comentários Finais

Neste capítulo foi apresentada a Heurística de Reconstrução Automática de Índices, responsável por decidir se um índice deve ser eliminado ou reconstruído.

Após apresentar o problema que leva à necessidade de reconstruir um índice: a degradação do desempenho de consultas que utilizam índices que sofreram muitos *page splits*, discutem-se os fatores que devem ser levados em

PUC-Rio - Certificação Digital Nº 0410827/CA

conta, antes de reconstruir um índice: fragmentação, tamanho de tabelas e participações do índice em operações de varreduras

Como foi utilizado o SGBD de código aberto PostgreSQL, foi preciso realizar extensões: cláusula **fillfactor**, que permite determinar o fator de preenchimento de páginas do nível folha de índices; e o comando **getsize**, cuja função consiste em obter quantidades de páginas e tuplas que não dependam do fato de estar a metabase atualizada.

Uma vez estabelecidas as bases da heurística, pôde-se descrever suas funcionalidades e detalhar seu projeto. Finalmente, foram apresentados amostras colhidas de testes, onde apresenta-se o funcionamento da heurística.

No próximo derradeiro capítulo, apresentam-se as conclusões desta dissertação e sugerem-se alguns desdobramentos deste trabalho.

PUC-Rio - Certificação Digital Nº 0410827/CA