4
Cenários de aplicação do *framework* e implementação

Neste capítulo são descritos cenários que demonstram como um DBA pode usar o *framework* como apoio para levantar argumentos que possam justificar e comprovar a eficácia do seu trabalho de sintonia fina. Além disso, é apresentado como o *framework* pode possibilitar uma maneira de combinar heurísticas de sintonia fina e compará-las. Através desse tipo de combinação pode-se gerar novas heurísticas com um conjunto de alternativas de sintonia fina cada vez mais amplo e gerar novas discussões sobre a eficácia ou não de heurísticas ou novas práticas para a área de sintonia fina.

4.1.
Usando o *framework* para levantar argumentos sobre decisão de sintonia fina

Com o uso do *framework*, o DBA consegue levantar argumentos para elaborar a sua justificativa em relação às ações de sintonia fina, através de consultas às instâncias da ontologia de aplicação. Como já mencionado no capítulo 3, a ontologia de aplicação foi desenvolvida com base em quatro heurísticas, ou seja, a ontologia contempla os conceitos necessários para que tais heurísticas possam ser executadas dentro do *framework*. No entanto, para ilustrar o uso do *framework* com as consultas que seriam úteis no momento de justificar a sintonia fina de um banco de dados, utilizam-se apenas as heurísticas usadas e/ou propostas no trabalho de (Costa et al, 2005) (Salles, 2004).

A heurística de escolha de índices candidatos (HEICL) usada por (Salles, 2004), foi proposta por (Lohman et al, 2000). Essa heurística trabalha com índices hipotéticos. Com isso, simula-se a possível existência de índices que podem ser úteis para uma determinada carga de trabalho e verifica-se se o índice seria usado pelo otimizador. Conforme a simulação vai ocorrendo de acordo com a carga de trabalho, uma outra heurística, denominada heurística de benefícios (HBM), vai acumulando benefícios aos índices hipotéticos que são escolhidos pelo

otimizador. Esse benefício equivale ao valor do custo de execução do comando sem a configuração hipotética, subtraído do custo de execução do comando com a configuração hipotética. No momento em que o benefício do índice hipotético atinge um determinado valor que ultrapasse o seu custo estimado de criação, o índice hipotético torna-se real, ou seja, passa a existir fisicamente na base de dados.

As heurísticas HEICL e HBM foram validadas através do uso do *benchmark* TPC-C no SGBD PostgreSQL. A validação, detalhada em (Salles, 2004), foi baseada em uma comparação entre a configuração sugerida pela ferramenta de auto-sintonia proposta por (Salles, 2004) e a própria configuração sugerida pelo *benchmark*. Foram realizados testes com a base de dados sem qualquer índice, com os índices sugeridos pela ferramenta de auto-sintonia e por fim, com os índices sugeridos pelo *benchmark*. Ao final dos testes, verificou-se que a ferramenta de auto-sintonia se aproximou do conjunto de índices sugerido pelo *benchmark*. A ferramenta não sugeriu apenas um índice, pois realmente esse não fazia sentido. Em uma análise pontual sobre a tabela que deveria ter sido indexada, verificou-se que o índice sugerido pelo *benchmark* - e não sugerido pela ferramenta - era apenas sobre uma tabela que possuía poucas tuplas. Como já mencionado nessa tese, é uma boa prática não definir índices sobre tabelas pequenas. Nos testes realizados, a tabela conseguiria no máximo quatro tuplas, o que justifica a sua não indexação.

Além disso, a ferramenta sugeriu um índice que não constava como sugestão pelo *benchmark*. Foi validado que, de fato, o índice era realmente útil, pois sem a criação desse índice, a solução de sintonia fina de índices proposta pela ferramenta de (Salles, 2004) para a carga de trabalho do TPC-C deixaria de ter um custo de execução menor em relação à solução proposta pelo *benchmark* e passaria a ter um custo a mais de 4,79% sobre o custo da solução do próprio *benchmark*. Dessa forma, conclui-se que embora o índice garantisse benefício para uma consulta com baixa frequência na carga de trabalho, a amplitude do benefício trazido justificava a criação do índice. Maiores informações sobre as heurísticas, seus testes e validações podem ser encontradas na dissertação de (Salles, 2004) ou no artigo (Costa et al, 2005).

A base de dados do TPC-C pode ser dimensionada para um número arbitrário de armazéns de itens de varejo. Para os cenários de justificativas

apresentados aqui, optamos pelo pior caso testado na dissertação de (Salles, 2004), onde são usados quatro armazéns. O objetivo dessa tese não é demonstrar eficácia de heurísticas, mas sim apresentar argumentos para que o DBA use para justificar a sua decisão final. Essa decisão pode ser com ou sem apoio de ferramentas. Então, para demonstrar nossos cenários de justificativas, selecionamos as duas heurísticas (HEICL e HBM) conhecidas do grupo de trabalho da PUC-Rio e um teste já realizado de 4 armazéns.

Como os argumentos para a justificativa são derivados de consultas realizadas pelo usuário do *framework* sobre a ontologia de aplicação, é um pré-requisito que o usuário tenha um mínimo de conhecimento sobre uma linguagem de regras para consultar ontologias descritas em OWL-DL. As consultas descritas nessa tese estão expressas em SQWRL (*Semantic Query-Enhanced Web Rule Language*). Optamos por SQWRL pela similaridade com a sintaxe das regras que foram descritas em SWRL e pela integração com a ferramenta Protégé.

TPC-C

O *benchmark* TPC-C simula uma carga de trabalho real, onde um fornecedor de varejo opera uma determinada quantidade de armazéns (***warehouse***) junto aos seus distritos (***district***) de vendas. Os armazéns possuem diversos estoques (***stock***) de produtos (***item***). Cada distrito de vendas possui diversos clientes (***customer***) associados. Cada cliente realiza pedidos de compra (***orders***) de produtos em estoque. Cada produto em estoque associado a um pedido de compra gera uma nova linha de pedido (**order_line**). Até que os pedidos de compra realizados pelos clientes sejam processados, eles são classificados como novos pedidos (***new_order***). Para fins de análises futuras, são mantidos históricos de compras dos clientes (***history***). Na Figura 4-1 tem-se um esquema da base de dados, composta por nove tabelas e suas respectivas quantidades de tuplas iniciais.

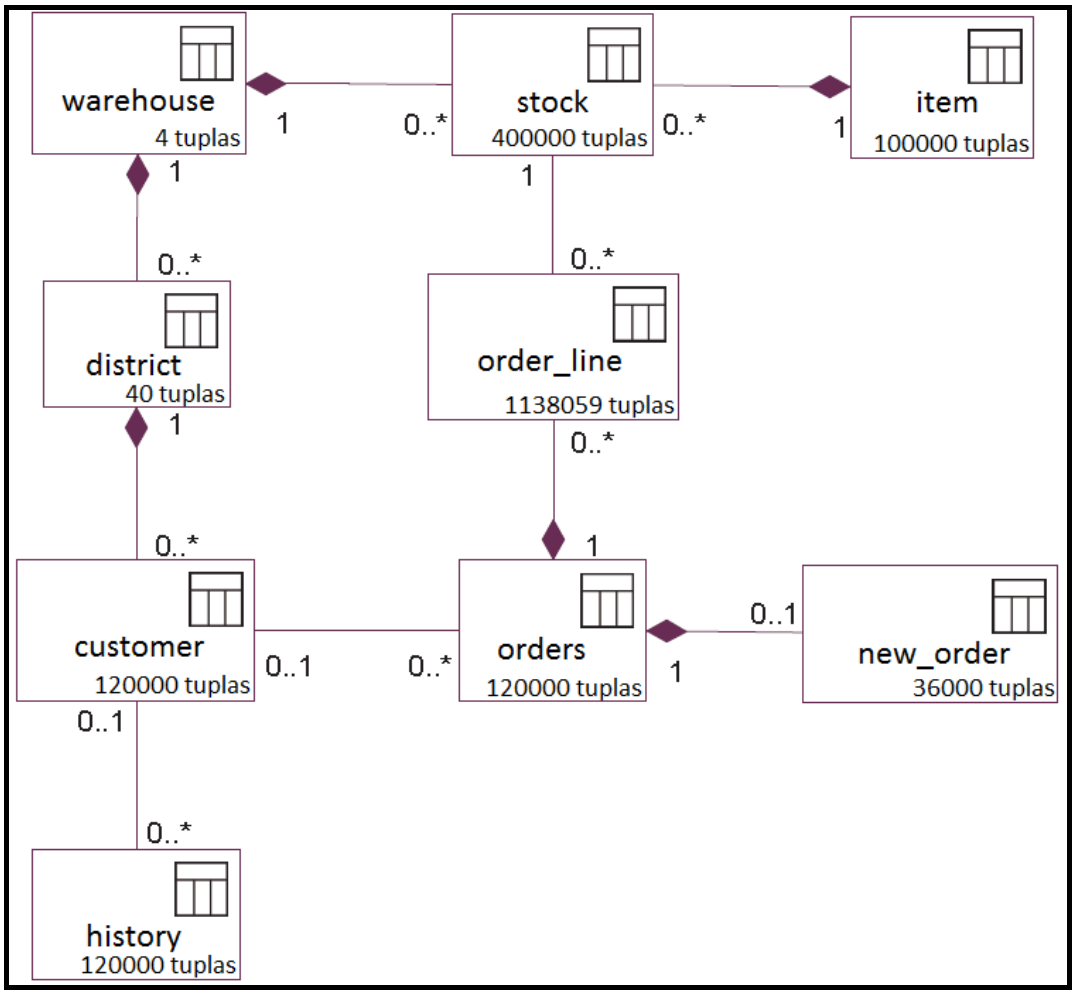

Figura 4-1 Esquema do TPC-C (TPC, 2013) (Salles, 2004)

A carga de trabalho possui diversas transações, incluindo consultas e comandos de atualizações, de um conjunto de operadores de terminais em um ambiente de entrada de pedidos. As transações que podem ser executas no SGBD são:

- Novos pedidos → atividades *on-line* de registro de pedidos de compra. Transação considerada de peso médio. Representa 45% das transações na carga de trabalho;
- Pagamentos → atualização do saldo de um cliente e reflexo do pagamento realizado nas informações estatísticas mantidas para o distrito e o armazém correspondentes. Transação considerada leve. Representa 43% das transações na carga de trabalho;
- Estado de pedido → verifica o estado do pedido mais recente de um cliente. Transação considerada de peso médio. Representa 4% das transações na carga de trabalho;

- Entrega → processamento de entrega de até dez pedidos de compra. Transação considerada leve. Representa 4% das transações na carga de trabalho;
- Nível de estoque → leitura que determina o número de produtos recentemente vendidos que têm um nível de estoque abaixo de um determinado limite. Transação considerada pesada. Representa 4% das transações na carga de trabalho.

4.1.1. Cenários com necessidade de justificar decisões de sintonia fina

No cenário utilizado para ilustrar a execução das heurísticas HEICL e HBM pelo *framework* sobre o *benchmark* TPC-C, imaginemos que o DBA não possui conhecimento sobre o raciocínio das mesmas. Tais heurísticas tiveram as suas regras refletidas no *framework* proposto nessa tese, adicionando semântica às decisões de sintonia fina.

As regras da heurística HEICL são:

- EQ + O
- EQ + O + RANGE
- EQ + O + RANGE + SARG
- EQ + O + RANGE + REF
- O + EQ
- O + EQ + RANGE
- O + EQ + RANGE + SARG
- O + EQ + RANGE + REF

Onde,

- EQ: colunas envolvidas em predicados de igualdade;
- O: colunas envolvidas em cláusulas `ORDER BY`, `GROUP BY` e predicados de junção;
- RANGE: colunas que aparecem em restrições de intervalos;
- SARG: colunas que aparecem em outros predicados indexáveis;
- REF: demais colunas referenciadas no comando SQL.

Tais regras são definidas em SWRL, em função dos conceitos descritos na ontologia de domínio e instanciadas nos conceitos de <condicaoRegra> → <acao> da ontologia de tarefa já apresentada e exemplificada no capítulo anterior. A partir dessas instâncias que o *framework* executa as heurísticas e gera as ações de sintonia fina no banco de dados.

Por exemplo: para a primeira regra – EQ + O, tem-se um exemplo da instância de **condição de regra** expressa em SWRL para verificar a condição sobre comandos DML que possuam colunas envolvidas em predicados de igualdade e não tenham cláusulas `ORDER BY`, `GROUP BY` ou predicados de junção (Figura 4-2).

```
1.    ComandoDML(?comando)
2.    ^ possui(?comando, ?clausula)
3.    ^ Where(?clausula)
4.    ^ temExpressaoLogica(?clausula, ?ep)
5.    ^ ExpressaoPredicado(?ep)
6.    ^ temOperador(?ep, ?op)
7.    ^ temSimbolo(?op, ?simb)
8.    ^ swrlb:matches(?simb, "=")
9.    ^ temObjetoExpressao(?ep, ?objExp)
10.   ^ Coluna(?objExp)
11.   ^ temNome(?objExp, ?nomeCol)
12.   ^ swrlx:makeOWLThing(?indHip, ?ep)
13.   ^ swrlb:stringConcat(?nomeHip, "hi_", ?nomeCol)
```

Figura 4-2 Exemplo de instância de condição de regra da heurística HEICL definida no *framework*

Uma vez que a condição de regra (Figura 4-2) seja satisfeita, ela vai gerar uma instância de **ação**, também definida na ontologia de tarefa, para a definição semântica de que será criado um índice hipotético (Linha 1 - Figura 4-3) sobre as colunas referenciadas nos predicados de igualdade (Linha 2 - Figura 4-3) e a informação do comando DML que originou o índice (Linha 4 - Figura 4-3). Além disso, o *framework* invoca a função responsável pela ação de executar a criação do índice hipotético no banco de dados.

```
1.    IndiceHipotetico(?indHip)
2.    ^ atuaSobre(?indHip, ?objExp)
3.    ^ temNome(?indHip, ?nomeHip)
4.    ^ origina(?comando, ?indHip)
```

Figura 4-3 Exemplo de instância de ação de regra da heurística HEICL definida no *framework*

O mesmo ocorre para as regras da heurística HBM, que são expressas em SWRL na Figura 4-39 e na Figura 4-41 como instâncias na ontologia de tarefa.

Inicialmente, todas as tabelas da base de dados encontram-se sem índices associados. O DBA inicia a execução do *framework*. A condição da regra da heurística HBM, responsável pela ação de sintonia fina que resulta em criação de índices reais é satisfeita e, como consequência, a primeira ação de sintonia fina é realizada no banco de dados, por uma função invocada pelo *framework*: criação de um determinado índice sobre a tabela de clientes (*customer*).

O DBA decide analisar as colunas da tabela indexada (Tabela 4-1).

O índice criado pela ação de sintonia fina chama-se **ri_customer_0_1_2**. O formato do nome de cada índice real é da seguinte forma:

ri + <nome da tabela indexada> + <ordem das colunas indexadas>

O prefixo “ri” é usado para diferenciar o nome de um índice real do nome de um índice hipotético. No caso de índice hipotético, usa-se o prefixo “hi”. A ordem das colunas é obtida pela metabase do SGBD. Essa ordem corresponde à posição das colunas no momento da descrição do comando de criação da tabela. A ordem das colunas é mostrada na Tabela 4-1.

Ordem	Nome	Definição
0	**c_id**	Número identificador do cliente
1	**c_d_id**	Número identificador do distrito (*district*) ao qual o cliente pertence
2	**c_w_id**	Número identificador do armazém (*warehouse*) ao qual o cliente pertence
3	c_first	Primeiro nome do cliente
4	c_middle	Nome do meio do cliente
5	c_last	Último nome do cliente
6	c_street_1	Endereço principal do cliente
7	c_street_2	Segundo endereço do cliente
8	c_city	Cidade do endereço principal do cliente
9	c_state	Estado do endereço principal do cliente
10	c_zip	Cep do endereço principal do cliente
11	c_phone	Telefone do cliente
12	c_since	Data em que o cliente começou a realizar compras na empresa
13	c_credit	Valor de crédito do cliente
14	c_credit_lim	Valor do limite de crédito do cliente
15	c_discount	Valor de desconto que o cliente tem direito
16	c_balance	Valor do saldo do cliente
17	c_ytd_payment	Período de pagamento do cliente
18	c_payment_cnt	Quantidade de pagamentos realizados pelo cliente
19	c_delivery_cnt	Quantidade de entregas realizadas para o cliente
20	c_data	Dados adicionais sobre o cliente

Tabela 4-1 Definição das colunas da tabela customer – TPC-C

Retornando a ação de sintonia fina que o DBA obteve da execução do *framework*, consegue-se verificar que o índice foi criado sobre as seguintes colunas: **c_id**, **c_d_id** e **c_w_id**.

Analisando as colunas da tabela e as colunas indexadas, o DBA não entende porque o índice foi criado e decide investigar o motivo de sua criação. Para realizar a sua investigação, o DBA segue alguns passos:

Passo 1: analisando a ontologia de domínio

O DBA utiliza a ontologia de domínio, buscando identificar como ele pode obter os comandos da carga de trabalho que derivaram o índice criado. Conforme apresentado na Figura 4-4, contendo um subconjunto da ontologia de domínio apenas para ilustração, o DBA verifica que existe um relacionamento que ele pode explorar entre ComandoDML, IndiceHipotetico e IndiceReal. Conforme explicado no capítulo anterior, é importante lembrar que todos os conceitos e relacionamentos definidos na ontologia de domínio foram derivados das informações necessárias para a execução das heurísticas selecionadas como base para a presente tese, mencionadas no capítulo anterior e detalhadas no início deste capítulo. Através do relacionamento (Figura 4-4), o DBA pode levantar os comandos DML que fizeram a ferramenta pensar nesse índice. O objetivo é obter todos os comandos desde quando o índice começou a ser considerado potencialmente útil e a obter benefícios enquanto era hipotético, até o momento em que o índice foi criado.

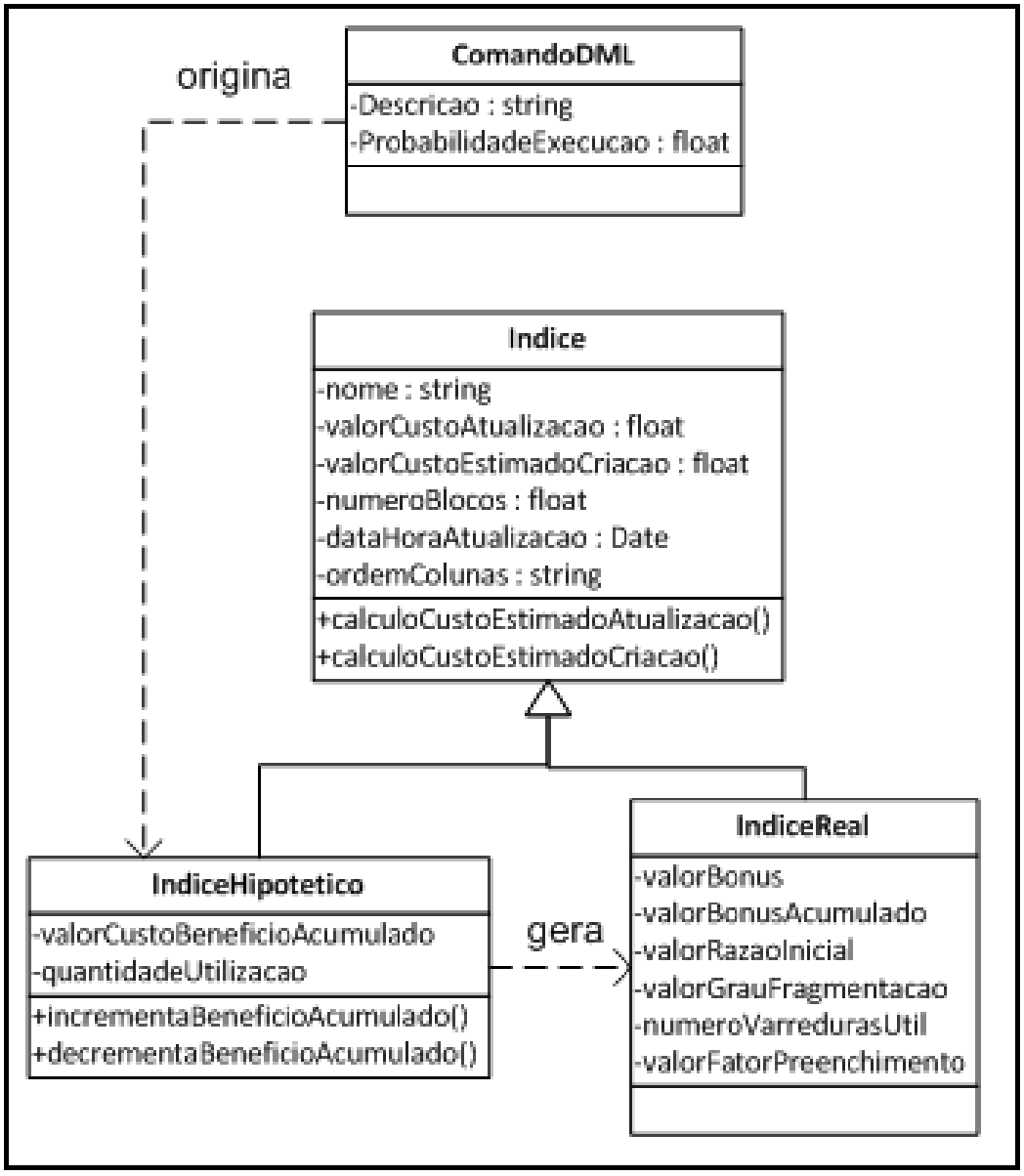

Figura 4-4 Subconjunto da ontologia – Consulta de comandos DML que originaram índice

Passo 2: obtendo comandos DML que originaram o índice

Para identificar os comandos DML, o DBA elabora uma possível consulta, apresentada na Figura 4-5. Nessa consulta (Figura 4-5), o DBA busca as descrições (linhas 7 e 8) de comandos DML (linha 1) que originaram (linha 3) o índice hipotético (linha 2) que gerou (linha 5) o índice real (linha 4), cujo nome é ri_customer_0_1_2 (linha 6).

```
1.    ComandoDML(?com)
2.    ^ IndiceHipotetico(?indHip)
3.    ^ origina(?com, ?indHip)
4.    ^ IndiceReal(?indReal)
5.    ^ gera(?indHip, ?indReal)
6.    ^ temNome(?indReal, "ri_customer_0_1_2")
7.    ^ temDescricao(?com, ?descCom)
8.    -> sqwrl:select(?descCom)
```

Figura 4-5 Consulta SQWRL para obter comandos DML que originaram índice real

Ao executar tal consulta (Figura 4-5) sobre a ontologia já instanciada com as informações do banco de dados durante a execução do *framework*, obtém-se o resultado (Figura 4-6) com os comandos DML que acarretaram em benefícios ao índice criado.

(a)
```
1.    UPDATE customer
2.    SET c_delivery_cnt = c_delivery_cnt + 1,
3.    c_balance = c_balance + 1
4.    WHERE c_id = 1 AND c_w_id = 1 AND c_d_id = 1;
```

(b)
```
1.    SELECT c_discount, c_last, c_credit
2.    FROM customer
3.    WHERE c_w_id = 1 AND c_d_id = 1 AND c_id = 1;
```

(c)
```
1.    SELECT c_first, c_middle, c_last, c_balance
2.    FROM customer
3.    WHERE c_w_id = 1 AND c_d_id = 1 AND c_id = 1;
```

Figura 4-6 Sequência de comandos que originaram ação de sintonia fina

Analisando os comandos DML obtidos (Figura 4-6), o DBA decide avaliar se faz sentido a indicação do índice para os comandos da sequência. Através da ontologia de tarefa, o DBA verifica que existem duas instâncias de heurísticas com regras ativas, ou seja, sendo executadas (HEICL e HBM). Uma heurística que gera ações de criação e remoção de índices hipotéticos (HEICL) e outra que gera ações de sintonia fina de criação e/ou remoção de índices reais (HBM). A

consulta em SQWRL, apresentada na Figura 4-7, sobre a ontologia de aplicação (ontologia de domínio e ontologia de tarefa), pode ser usada para buscar heurísticas com regras ativas, onde a variável <descricaoRegra> (Linha 7 - Figura 4-7) precisa ser substituída pelas descrições das regras retornadas do método `getEnabledImps`. Esse método consta na API (*Application Programming Interface*) do Protégé que é usada na implementação do *framework* e retorna uma coleção de regras que estão ativas na ontologia de aplicação.

```
1.    Heuristica(?heur)
2.    ∧ temNomeHeuristica(?heur, ?nomeHeur)
3.    ∧ temClassificacao(?heur, ?classHeur)
4.    ∧ CondicaoRegra(?condReg)
5.    ∧ define(?heur, ?condReg)
6.    ∧ temDescricaoRegra(?condReg, ?descReg)
7.    ∧ swrlb:matches(?descReg, <descricaoRegra>)
8.    →
9.    sqwrl:select(?nomeHeur, ?classHeur)
```

Figura 4-7 Consulta para obter heurísticas com regras ativas na ontologia de aplicação

Analisando as instâncias de conceitos que são pré-condições das heurísticas na ontologia de tarefa, o DBA nota que a heurística HEICL precisa gerar as instâncias do conceito de índices hipotéticos para que a outra heurística, HBM, possa executar as ações de sintonia fina. Dessa forma, o DBA decide analisar qual a condição de regra que foi satisfeita para a heurística HEICL criar o índice correspondente ao índice real.

O DBA verifica que possui uma relação entre índice hipotético e índice real através do modelo da ontologia de domínio (Figura 4-4). Além disso, de acordo com o modelo da ontologia de tarefa, para buscar a condição de regra que foi satisfeita e que gerou como consequência a ação de sintonia fina, o DBA precisa conhecer a descrição da ação (Figura 4-8).

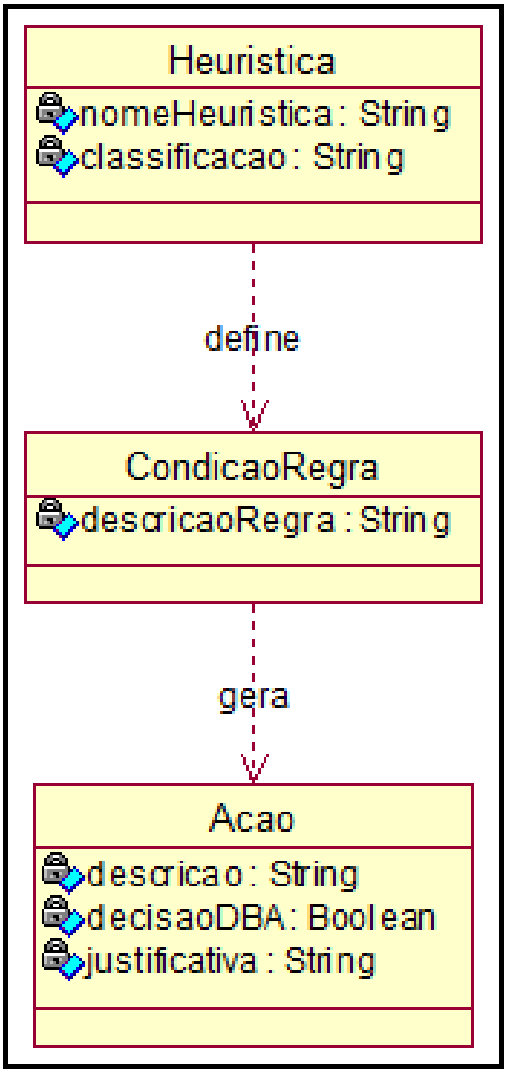

Figura 4-8 Subconjunto da ontologia de tarefa (metamodelo do *framework*) – Busca da origem da ação

Como explicado no capítulo 3, a descrição da ação é o comando exato da ação que deve ser executada na base de dados. Se a ação que o DBA deseja investigar é a criação de um índice hipotético, ele precisa, antes, obter o nome do índice hipotético.

Passo 3: obtendo o nome do índice enquanto hipotético

Para obter o nome do índice enquanto ele existia como hipotético, o DBA faz uso do relacionamento (Figura 4-4) entre o índice hipotético e o índice real. Como o DBA já sabe o nome do índice real (ri_customer_0_1_2), ele pode realizar a consulta apresentada na Figura 4-9.

```
1.    IndiceHipotetico(?indHip)
2.    ^ IndiceReal(?indReal)
3.    ^ gera(?indHip, ?indReal)
4.    ^ temNome(?indReal, "ri_customer_0_1_2")
5.    ^ temNome(?indHip, ?nomeHip)
6.    -> sqwrl:select(?nomeHip)
```

Figura 4-9 Consulta SQWRL para obter o nome do índice hipotético que gerou o índice real

Após a execução da consulta (Figura 4-9), o DBA descobre o nome do índice hipotético (hi_customer_0_1_2) que gerou o índice real criado na ação de sintonia fina.

Passo 4: elaborando o comando responsável pela ação de criação do índice hipotético

Agora, de posse do nome do índice hipotético, o DBA já pode elaborar a descrição da ação que a heurística HEICL resultou (Figura 4-10) após a validação de uma condição de regra.

```
1.    Create hypothetical index hi_customer_0_1_2
2.    on customer(c_id, c_d_id, c_w_id);
```

Figura 4-10 Descrição da ação gerada pela heurística HEICL

Passo 5: obtendo a condição de regra que gerou o índice enquanto hipotético

Conhecendo a descrição da ação e usando a ontologia de tarefa (Figura 4-8), o DBA já pode buscar a descrição da regra que originou o índice hipotético. Para tal, o DBA pode usar a consulta descrita na Figura 4-11.

```
1.    Acao(?acao)
2.    ^ CondicaoRegra(?regra)
3.    ^ temDescricaoRegra(?regra, ?descRegra)
4.    ^ gera(?regra, ?acao)
5.    ^ temDescricao(?acao,
6.        "Create hypothetical index hi_customer_0_1_2
7.        on customer(c_id, c_d_id, c_w_id);")
8.    -> sqwrl:select(?descRegra)
```

Figura 4-11 Consulta para obter a condição de regra que gerou determinado índice hipotético

Como resultado da consulta sobre as instâncias da ontologia de tarefa, o DBA consegue obter todas as condições de regras que foram verdadeiras para gerar a ação pesquisada (Figura 4-10). Por exemplo, para o comando DML (a) da Figura 4-6, a regra verdadeira foi a descrita na Figura 4-12.

```
1.    ComandoDML(?comando)
2.    ^possui(?comando, ?clausula)
3.    ^ Where(?clausula)
4.    ^ temExpressaoLogica(?clausula, ?ep)
5.    ^ ExpressaoPredicado(?ep)
6.    ^ temOperador(?ep, ?op)
7.    ^ temSimbolo(?op, ?simb)
8.    ^ swrlb:matches(?simb, "=")
9.    ^ temObjetoExpressao(?ep, ?objExp)
10.   ^ Coluna(?objExp)
11.   ^ temOrdem(?objExp, ?ordemCol)
12.   ^ swrlx:makeOWLThing(?indHip, ?ep)
13.   ^ swrlb:stringConcat(?nomeHip, "hi_", ?ordemCol)
```

Figura 4-12 Condição de regra que originou o comando (a) da Figura 4-6

Passo 6: interpretando a condição de regra

Utilizando a ontologia de domínio como apoio, o DBA começa a interpretar a condição da regra (Figura 4-12). Existe um comando DML (linha 1), que possui cláusula (linha 2). Essa cláusula é do tipo `where` (linha 3) e tem expressão lógica

(linha 4) do tipo expressão de predicado (linha 5). Tendo a expressão de predicado (linha 5), verifica se ela tem operador (linha 6), cujo símbolo (linha 7) é equivalente ao "=" (linha 8) e se tem objeto de expressão (linha 9) do tipo coluna (linha 10). Caso o objeto de expressão seja do tipo coluna (linha 10), recupera a sua ordem na tabela (linha 11) e cria uma nova instância (linha 12), concatenando a string "hi_" (prefixo usado para índice hipotético) com a ordem da coluna (linha 13).

Apenas como uma forma de ilustrar a validação da regra apresenta-se na Figura 4-13, na Figura 4-14 e na Figura 4-16 as instâncias da ontologia de domínio envolvidas na regra para o comando (a) da Figura 4-6.

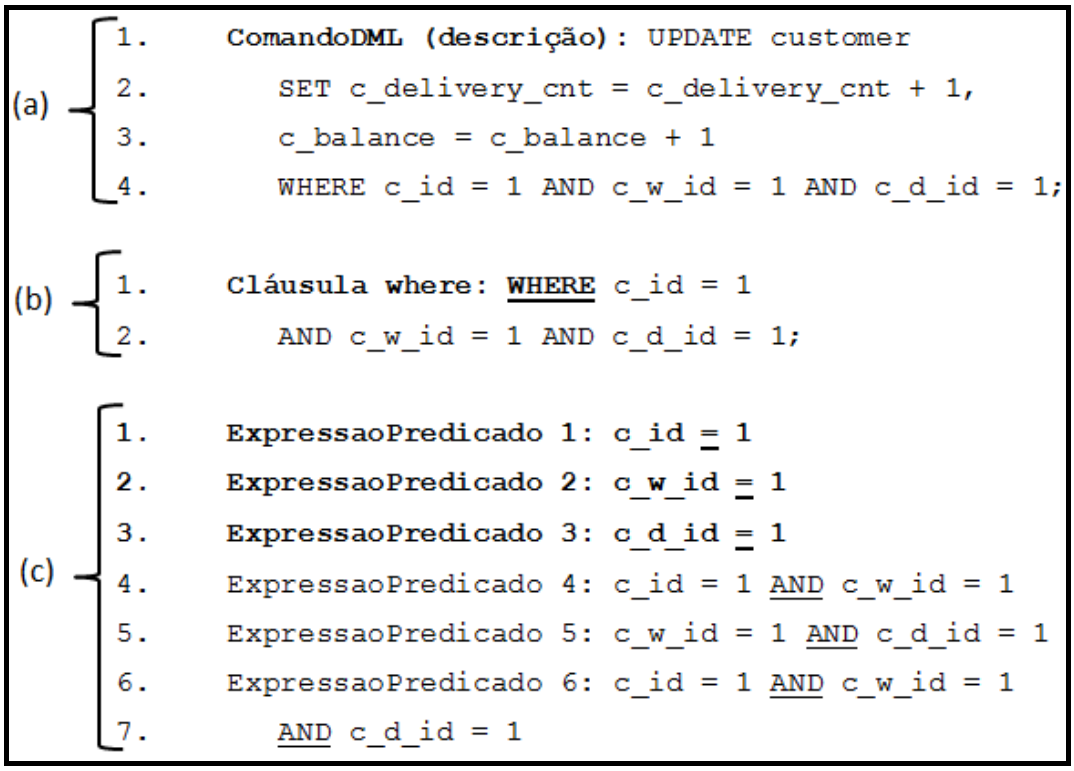

Figura 4-13 Instância de ComandoDML, `Where` e ExpressaoPredicado – Ontologia de domínio

A instância do comandoDML (Figura 4-13.a) corresponde ao comando que satisfez a regra que gerou a ação de criação do índice hipotético. Para obter tal instância, foi executada uma consulta contendo os itens da condição de regra (Linhas 1 à 11 - Figura 4-12, pois não é necessária a criação de um indivíduo como na regra), adicionando uma consequência (sqwrl: select (?comando). Conforme visto no capítulo anterior, a cláusula `where` (Figura 4-13.b) foi inferida, através da execução de regra para obter a sua instância, analisando o

comando DML (Figura 4-13.a). Executando a função, explicada no capítulo anterior, que deriva as instâncias de expressões de predicado, obtêm-se as expressões descritas em Figura 4-13.c em cima da instância de `where` (Figura 4-13.b). Das instâncias de expressões de predicado (Figura 4-13.c), somente as expressões de predicado 1 (linha c.1), 2 (linha c.2) e 3 (linha c.3) satisfazem os itens da regra que analisa o operador "=" e possuem o objeto de expressão do tipo coluna. Logo, sendo do tipo coluna, somente os objetos de expressão descritos na Figura 4-14 que são considerados.

```
1.    temObjetoExpressao 1: c_id
2.    temObjetoExpressao 2: c_w_id
3.    temObjetoExpressao 3: c_d_id
```

Figura 4-14 Objetos de expressão considerados na regra da Figura 4-12

Após a identificação dos objetos de expressão, é criada a instância do índice hipotético hi_customer_0_1_2 e seus relacionamentos (Figura 4-16). A criação destas instâncias na ontologia de domínio é realizada através da execução da ação da regra descrita na Figura 4-15, pela máquina de regras.

```
1.    -> IndiceHipotetico(?indHip)
2.    ^ atuaSobre(?indHip, ?objExp)
3.    ^ temNome(?indHip, ?nomeHip)
4.    ^ origina(?comando, ?indHip)
```

Figura 4-15 Ação derivada da condição de regra descrita na Figura 4-12

(a)
```
1.   atuaSobre: hi_customer_0_1_2 atuaSobre c_id
2.   atuaSobre: hi_customer_0_1_2 atuaSobre c_w_id
3.   atuaSobre: hi_customer_0_1_2 atuaSobre c_d_id
```

(b)
```
1.   Origina: UPDATE customer
2.         SET c_delivery_cnt = c_delivery_cnt + 1,
3.         c_balance = c_balance + 1
4.         WHERE c_id = 1 AND c_w_id = 1
5.         AND c_d_id = 1;
6.      origina hi_customer_0_1_2
```

Figura 4-16 Instâncias dos relacionamentos do índice hipotético

Assim, através do relacionamento presente na ontologia de tarefa, entre a regra e a ação (Figura 4-8), o DBA consegue identificar para cada índice, quais regras o derivaram. Como a regra está descrita de acordo com o modelo de domínio, o DBA consegue analisar se a regra, determinada pela heurística usada, faz mesmo sentido ou não. Por consequência, o DBA pode entender o motivo pelo qual o índice hipotético foi identificado como possivelmente útil, ou seja, pelo fato de suas colunas estarem presentes na cláusula `where` de um comando de atualização, por exemplo (Figura 4-13.b).

Continuando os questionamentos do DBA sobre as heurísticas executadas, ele analisa a carga de trabalho, manualmente ou através de consultas às instâncias da ontologia de domínio e identifica um comando DML que poderia ter originado o índice criado durante a sintonia fina (Figura 4-17), mas não o originou.

```
1.    SELECT c_id
2.    FROM customer
3.    WHERE c_w_id = 1 AND c_d_id = 1
4.       AND c_last = 'BARBARBAR'
5.    ORDER BY c_first ASC;
```

Figura 4-17 Possível comando DML que poderia ter originado o índice real ri_customer_0_1_2

Dado que o comando também referencia a tabela *customer* (linha 2 - Figura 4-17) e alguns dos atributos presentes no índice criado (linhas 1 e 3 - Figura 4-17), o DBA decide investigar quais foram os índices pensados pela ferramenta para àquele comando DML. Para obter a informação sobre quais índices foram pensados, o DBA poderia executar a consulta descrita na Figura 4-18.

```
1.    ComandoDML(?com)
2.    ^ temDescricao(?com, "SELECT c_id
3.       FROM customer
4.       WHERE c_w_id = 1 AND c_d_id = 1
5.         AND c_last = 'BARBARBAR'
6.       ORDER BY c_first ASC;")
7.    ^ IndiceHipotetico(?indHip)
8.    ^ origina(?com, ?indHip)
9.    ^ temNome(?indHip, ?nomeIndHip)
10.   ^ atuaSobre(?indHip, ?coluna)
11.   ^ temNome(?coluna, ?nomeCol)
12.   -> sqwrl:selectDistinct(?nomeIndHip, ?nomeCol)
```

Figura 4-18 Consulta SQWRL para obter índices hipotéticos pensados para um determinado comando DML

A consulta da Figura 4-18 indica que: se existe um comando DML (linha 1) com a descrição que o DBA deseja buscar (`linhas 2, 3, 4, 5 e 6`) e se existe um índice hipotético (linha 7) que tenha sido originado por tal comando (linha 8), retorne o nome dos índices (linhas 9 e 12) e os nomes (linhas 11 e 12) das colunas que ele indexa (linha 10). Dessa forma, a máquina de consultas retorna os nomes dos índices hipotéticos pensados para o comando DML (Tabela 4-2).

Nome do índice hipotético	Colunas indexadas
hi_customer_1_2_5_3	**c_d_id, c_w_id, c_last, c_first**
hi_customer_1_2_5_3_0	c_d_id, c_w_id, c_last, c_first, c_id
hi_customer_3_1_2_5	c_first, c_d_id, c_w_id, c_last
hi_customer_3_1_2_5_0	c_first, c_d_id, c_w_id, c_last, c_id
hi_customer_1	c_d_id
hi_customer_2	c_w_id
hi_customer_5	c_last
hi_customer_3	c_first

Tabela 4-2 Índices hipotéticos pensados para o comando DML da Figura 4-17

Analisando o resultado da consulta (Tabela 4-2), o DBA verifica que realmente o índice criado não consta na lista de índices pensados como útil para o comando da Figura 4-17. Nesse momento, o DBA decide investigar se o índice realmente não seria útil para o comando. O DBA cria os índices hipotéticos pensados pela ferramenta para que o otimizador avalie o custo de execução da consulta considerando a existência de tais índices juntamente com o índice já criado na sintonia fina. Ao gerar o plano de execução da consulta que gerou a dúvida, o DBA valida que de fato, o otimizador não seleciona o índice criado. O otimizador opta pelo índice hi_customer_1_2_5_3 como tendo menor custo para a execução do comando. Esse índice escolhido pelo otimizador é um dos índices pensados pela heurística HEICL, conforme resultado na Tabela 4-2. Dessa forma, o DBA entende o motivo pelo qual não faria sentido o comando originar o índice criado e possui argumentos para justificar o fato. Mesmo que o índice fosse criado, o otimizador não o selecionaria para executar o comando DML (Figura 4-17).

Para obter maior comprovação da eficácia da heurística, o DBA decide confirmar se a heurística considerou realmente o índice hi_customer_1_2_5_3, escolhido pelo otimizador, como sendo o benéfico para a consulta (Figura 4-17). A confirmação pode ser obtida pela consulta descrita na Figura 4-19.

```
1.    ComandoDML(?com)
2.    ^ temDescricao(?com, "SELECT c_id
3.       FROM customer
4.       WHERE c_w_id = 1 AND c_d_id = 1
5.         AND c_last = 'BARBARBAR'
6.       ORDER BY c_first ASC;")
7.    ^ PlanoExecucaoHipotetico(?plExecHip)
8.    ^ executa(?com, ?plExecHip)
9.    ^ IndiceHipotetico(?indHip)
10.   ^ utiliza(?com, ?indHip)
11.   ^ temNome(?indHip, ?nomeIndHip)
12.   ^ temValorCustoBeneficioAcumulado(?indHip, ?vBenef)
13.   -> sqwrl:select(?nomeIndHip,?vBenef)
```

Figura 4-19 Consulta SQWRL para verificar qual índice foi considerado benéfico para determinado comando DML

Como o DBA espera que a heurística tenha considerado o índice correto e tal índice ainda não foi criado, ele verifica a existência dos índices hipotéticos e seus custos de benefícios acumulados com o apoio da ontologia de domínio (Figura 4-4). A consulta usada para verificar o benefício acumulado pelo índice (Figura 4-19), cita que existe um comando DML (linha 1), cuja descrição é a consulta que gerou a dúvida sobre o índice (linhas 2, 3, 4, 5 e 6). O comando DML executa (linha 8) um plano de execução hipotético (linha 7) que utiliza (linha 10) um determinado índice hipotético (linha 9) que tem um nome (linha 11) e um valor de custo de benefício acumulado (linha 12). Com essas condições sendo verdadeiras, recupera-se o nome do índice considerado benéfico e seu valor de custo de benefício, como segue: hi_customer_1_2_5_3 – 16613.8. Com esse resultado, o DBA valida que a heurística realmente identificou o índice que o otimizador escolheu como tendo o menor custo para a execução do comando da Figura 4-17, ou seja, não faria mesmo sentido o comando ter originado o índice ri_customer_0_1_2 porque o otimizador não usaria.

Em resumo, verifica-se que com o uso do *framework*, o DBA consegue levantar argumentos para justificar a ação de criação de índice real no banco de

dados e o motivo pelo qual um determinado comando DML realmente não é beneficiado pelo índice criado pelas regras das heurísticas de sintonia fina usadas.

Em outro cenário, imaginemos que uma empresa de vendas possua um analista de sistemas sênior, com um mínimo de conhecimento sobre banco de dados. O analista contrata a consultoria de um DBA para realizar a sintonia fina da base de dados da empresa. O DBA analisa a carga de trabalho da empresa e sugere os índices presentes na Tabela 4-3 como sendo a configuração indicada para a base de dados da empresa.

Índice	Tabela indexada	Colunas indexadas
Índice 1	customer	c_id, c_d_id, c_w_id
Índice 2	customer	c_d_id, c_w_id, c_last, c_first
Índice 3	district	d_w_id
Índice 4	item	i_id
Índice 5	new_order	no_ o_id, no_d_id, no_w_id
Índice 6	new_order	no_w_id
Índice 7	order_line	ol_o_id, ol_d_id, ol_w_id
Índice 8	orders	o_id, o_d_id, o_w_id
Índice 9	orders	o_d_id, o_w_id, o_c_id, o_id
Índice 10	stock	s_i_id, s_w_id

Tabela 4-3 Configuração sugerida por um DBA contratado para realizar a sintonia fina do banco de dados de uma empresa de vendas

Como o analista possui um conhecimento sobre a base de dados e sabe que constam na carga de trabalho algumas consultas que referenciam a tabela de armazém (*warehouse*), ele questiona o DBA sobre o motivo de não existir nenhum índice sobre a tabela (*warehouse*).

Diante do questionamento, o DBA analisa a descrição das colunas da tabela *warehouse* (Tabela 4-4).

Ordem	Nome	Definição
0	w_id	Número identificador do armazém
1	w_name	Nome atribuído ao armazém
2	w_street_1	Endereço principal do armazém
3	w_street_2	Segundo endereço do armazém
4	w_city	Cidade do endereço principal do armazém
5	w_state	Estado do endereço principal do armazém
6	w_zip	Cep do endereço principal do armazém
7	w_tax	Valor da taxa cobrada pelo armazém
8	w_ytd	Período do armazém

Tabela 4-4 Descrição das colunas da tabela *warehouse* do TPC-C

Utilizando os dados da ontologia de domínio instanciada, o DBA resolve buscar argumentos que justifiquem essa ausência de índices sobre a tabela. Ele decide se certificar de que realmente existam consultas na carga de trabalho que possuam cláusulas que referenciem a tabela *warehouse*. Para apoiar o DBA nessa certificação, ele pode usar a consulta SQWRL apresentada na Figura 4-20.

```
1.    ComandoDML(?com)
2.    ^ temDescricao(?com, ?descCom)
3.    ^ Clausula(?claus)
4.    ^ possui(?com, ?claus)
5.    ^ Tabela(?tab)
6.    ^ referencia(?claus, ?tab)
7.    ^ temNome(?tab, "warehouse")
8.    -> sqwrl:selectDistinct(?descCom)
```

Figura 4-20 Consulta SQWRL para obter comandos DML que referenciam determinada tabela da base de dados

Um exemplo de comando DML obtido como resultado da consulta (Figura 4-20) é apresentado na Figura 4-21. O DBA analisa os resultados da consulta e conclui que todas são do mesmo tipo, ou seja, projetando a coluna `w_tax` e restringindo o resultado pela coluna `w_id`.

```
1.    SELECT w_tax
2.    FROM warehouse
3.    WHERE w_id = 1;
```

Figura 4-21 Exemplo de consulta da carga de trabalho que referencia a tabela *warehouse*

Com um exemplo de consulta (Figura 4-21), o DBA pode verificar quais os índices foram pensados para tal comando, buscando apresentar para o analista da empresa que ele pensou em índices para a tabela. Ao executar a consulta similar a que ele tinha executado quando estava questionando a ferramenta de auto-sintonia (Figura 4-18), o DBA obtém os nomes dos índices hipotéticos originados pelo comando da Figura 4-21. Os nomes dos índices hipotéticos e as suas respectivas colunas indexadas podem ser vistos na Tabela 4-5.

Nome do índice hipotético	Colunas indexadas
hi_warehouse_0	w_id
hi_ warehouse _0_7	w_id, w_tax

Tabela 4-5 Índices hipotéticos pensados pela heurística HEICL para a tabela *warehouse* do TPC-C

Após apresentar exemplos de índices que foram pensados para a tabela questionada para o analista da empresa, o DBA decide enriquecer os seus argumentos e analisar se algum desses índices proporcionou benefício para a execução do comando DML (Figura 4-21). Ele decide analisar o plano de execução que o otimizador gerou para o comando DML. No plano de execução consta o tipo de varredura que o otimizador decidiu realizar no momento de executar o comando. Através do tipo de varredura escolhida como menor custo pelo otimizador, o DBA consegue mais um argumento para justificar a decisão de não ter sugerido índice para a tabela. Analisando a ontologia de domínio (Figura 4-22), o DBA identifica o relacionamento (executa) entre o comando DML e o

plano de execução, bem como o relacionamento (seleciona) do plano de execução com o tipo de varredura.

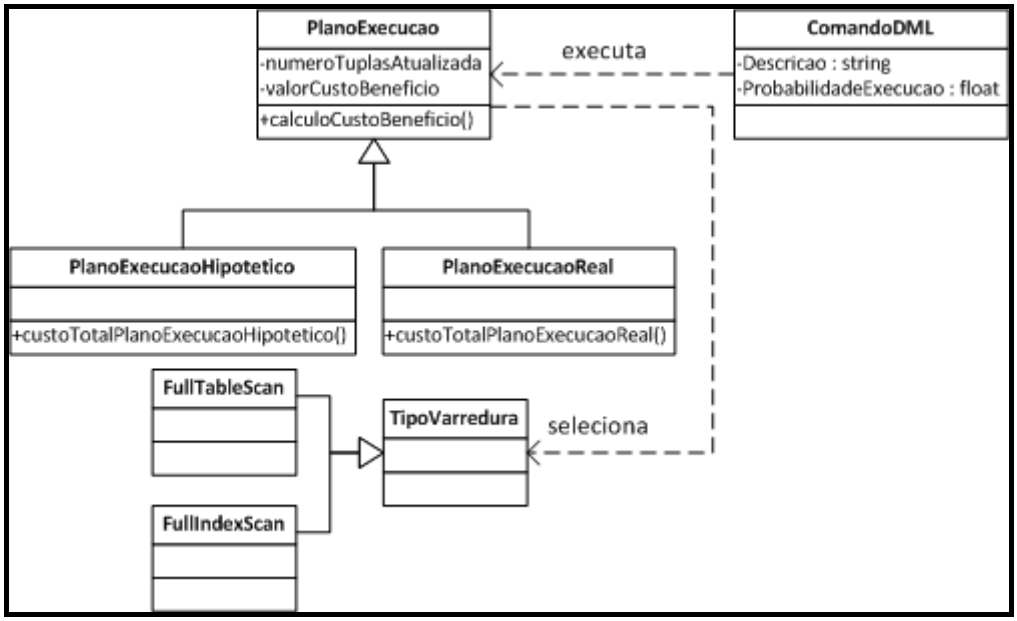

Figura 4-22 Subconjunto da ontologia de domínio – Relacionamento entre comando DML, Plano de execução e tipo de varredura

Com a existência dos relacionamentos (Figura 4-22), a consulta da Figura 4-23 pode apoiar o DBA na descoberta de qual tipo de varredura foi selecionada para o plano de execução hipotético gerado para o comando (Figura 4-21).

```
1.    ComandoDML(?com)
2.    ^ temDescricao(?com, "SELECT w_tax
3.       FROM warehouse
4.       WHERE w_id = 1;")
5.    ^ PlanoExecucaoHipotetico(?plExecHip)
6.    ^ executa(?com, ?plExecHip)
7.    ^ TipoVarredura(?tipoVarr)
8.    ^ seleciona(?plExecHip, ?tipoVarr)
9.    ^ abox:hasClass(?tipoVarr, ?classeTipoVarr)
10.   -> sqwrl:selectDistinct(?classeTipoVarr)
```

Figura 4-23 Consulta SQWRL para obter tipo de varredura usada em plano hipotético gerado para determinado comando DML.

Como resposta, o DBA obtém que o tipo de varredura optado pelo otimizador foi *FullTableScan*. A partir daí, o DBA justifica para o analista da empresa que mesmo tendo índice sugerido para a tabela, o otimizador prefere realizar uma varredura completa da mesma, do que utilizar um índice. Caso o DBA queira mais argumentos para a sua justificativa, ele também poderia consultar as informações da tabela. Uma das informações que seria útil para complementar a justificativa de não criação de índice seria a instância da propriedade numeroTuplas presente em tabela. Analisando o banco de dados, o DBA poderia verificar que a tabela *warehouse* possui apenas 4 tuplas, o que justifica a preferência do otimizador por uma varredura completa.

Além disso, o analista da empresa poderia realizar outro questionamento. Em geral, índices são considerados benéficos apenas para comandos DML do tipo consulta. Por acaso, existe algum comando DML de atualização que tenha se beneficiado, por exemplo, do índice 1 (Tabela 4-3) sobre a tabela *customer*?

Para explicitar um exemplo e justificar que nem sempre um índice é benéfico somente para consultas, o DBA busca (Figura 4-24) na ontologia instanciada as descrições do comando DML do tipo atualização que tenham originado algum índice hipotético que gerou o índice real apontado na questão do analista. Vale lembrar que o DBA já está familiarizado com o *framework* e sabe o nome real do índice (`ri_customer_0_1_2`).

```
1.    ComandoDML(?com)
2.    ^ temDescricao(?com, ?descCom)
3.    ^ Atualizacao(?com)
4.    ^ IndiceHipotetico(?indHip)
5.    ^ origina(?com, ?indHip)
6.    ^ IndiceReal (?indReal)
7.    ^ temNome(?indReal, "ri_customer_0_1_2")
8.    ^ gera(?indHip, ?indReal)
9.    -> sqwrl:selectDistinct(?descCom)
```

Figura 4-24 Consulta SQWRL para buscar comandos DML do tipo atualização que se beneficiariam de um determinado índice

Através da execução da consulta, o DBA apresenta para o analista da empresa um exemplo de comando de atualização (Figura 4-25) que obteve benefício ao utilizar o índice sugerido para criação sob a tabela *customer*. Isso se deve ao fato de que os atributos que fazem parte do índice são utilizados apenas para a realização de pesquisas no comando de atualização, ou seja, os seus dados não são atualizados. Sendo assim, o índice não sofre impactos físicos de manutenção.

```
1.    UPDATE customer
2.    SET c_delivery_cnt = c_delivery_cnt + 1,
3.       c_balance = c_balance + 1
4.    WHERE c_id = 1 AND c_w_id = 1 AND c_d_id = 1;
```

Figura 4-25 Exemplo de resultado da consulta da Figura 4-24

Nesse cenário, o *framework* foi usado para justificar uma tabela que não foi indexada e demonstrar que nem sempre os índices são considerados benéficos apenas para comandos de consulta, mas também para comandos de atualização.

As ferramentas de sintonia fina não disponibilizam todas as alternativas pensadas para a realização das ações de sintonia fina. Nesse contexto, poderiam existir alternativas de soluções não consideradas ótimas para uma sintonia local, mas que se fossem aplicadas, trariam benefícios maiores globalmente.

Imaginando a estrutura da base de dados do TPC-C, tem-se a tabela *stock*, responsável por armazenar os dados sobre os estoques de produtos (Tabela 4-6).

Ordem	Nome	Definição
0	s_i_id	Número identificador do estoque
1	s_w_id	Número identificador do armazém (*warehouse*) ao qual o cliente pertence
2	s_quantity	Quantidade em estoque
3	s_dist_01	Campo texto para anotações sobre o estoque
4	s_dist_02	Campo texto para anotações sobre o estoque
5	s_dist_03	Campo texto para anotações sobre o estoque
6	s_dist_04	Campo texto para anotações sobre o estoque
7	s_dist_05	Campo texto para anotações sobre o estoque
8	s_dist_06	Campo texto para anotações sobre o estoque
9	s_dist_07	Campo texto para anotações sobre o estoque
10	s_dist_08	Campo texto para anotações sobre o estoque
11	s_dist_09	Campo texto para anotações sobre o estoque
12	s_dist_10	Campo texto para anotações sobre o estoque
13	s_ytd	Período de estoque
14	s_order_cnt	Quantidades de pedidos de estoque
15	s_remote_cnt	Quantidade de pedidos remotos de estoque
16	s_data	Data do estoque

Tabela 4-6 Descrição das colunas da tabela *stock* do TPC-C

Analisando a carga de trabalho, tem-se que a consulta descrita na Figura 4-26 é frequente, alterando apenas o seu valor de busca.

```
1.    select s_i_id
2.    from stock
3.    where s_w_id = 8
4.     and s_quantity < 10;
```

Figura 4-26 Consulta frequente na carga de trabalho, definida sobre a tabela *stock*

O DBA opta por usar uma ferramenta de apoio para a realização da atividade de sintonia fina. Diante da consulta detectada na carga de trabalho como

sendo frequente (Figura 4-26), a ferramenta poderia decidir, imediatamente, por uma criação de um índice sobre as colunas presentes na cláusula `where`: `s_w_id` e `s_quantity`.

No entanto, após um determinado período de tempo, quando ocorrer atualização de estoque na loja, diversos comandos de `update` passam a ser submetidos ao SGBD em paralelo com essa consulta. Por exemplo, o comando apresentado na Figura 4-27.

```
1.    update stock
2.    set s_quantity = s_quantity + 50
3.    where s_i_id = 93054;
```

Figura 4-27 Comando de atualização sobre a tabela *stock*, presente na carga de trabalho

Se a ferramenta mostrasse as possíveis alternativas de solução de índices (Tabela 4-7) possivelmente benéficos para a consulta (Figura 4-26), o DBA, já com conhecimento prévio sobre a base de dados, saberia que a alternativa sugerida como ótima pela ferramenta, poderia não ser a melhor solução global para a carga de trabalho. Uma alternativa de solução para a consulta seria mesmo a escolha de indexar as colunas `s_w_id` e `s_quantity`, pois as mesmas trariam 50% de melhoria de desempenho. Já para o comando de atualização (Figura 4-27) seria a indexação da coluna `s_i_id`, pois a mesma está presente na cláusula `where` (linha 3).

O DBA analisa as duas possibilidades de solução:

- Indexar as colunas `s_w_id` e `s_quantity`
- Indexar a coluna `s_i_id`

O DBA conclui que indexar a coluna `s_w_id` não prejudica a execução de nenhum dos comandos mencionados na carga de trabalho e beneficia a execução da consulta em 40%. Logo, a decisão por indexar tal coluna já pode ser tomada. No entanto, a coluna `s_quantity`, embora beneficie a consulta, ela acaba sendo prejudicada pelo comando de atualização, visto que é uma coluna a ser mantida durante a execução do mesmo, acarretando em um custo grande para o SGBD

como um todo. Em relação à coluna `s_i_id`, ela não traz o mesmo benefício proporcionado por `s_quantity` para a consulta, mas essa seria a melhor decisão a ser tomada dado que traz melhoria tanto para a consulta (embora seja 30% ao invés de 40%) quanto para as execuções de atualizações.

Portanto, a melhor decisão do DBA seria indexar as colunas `s_w_id` e `s_i_id`, de forma a realizar a sintonia global, ou seja, visualizando o SGBD como um todo e os impactos de cada alternativa sobre ele. Por isso, o DBA precisa ter uma visão geral sobre os índices pensados para cada comando, tendo uma referência sobre os benefícios trazidos pelas possíveis ações de sintonia fina. Com isso, ele possui a flexibilidade de optar por outras soluções que não sejam consideradas ótimas pela ferramenta, mas que não foram consideradas ótimas por diferenças possivelmente insignificantes.

Ranking	Índice	Colunas indexadas	% Benefício
1	hi_stock_1_2	s_w_id, s_quantity	50%
2	hi_stock_0_1_2	s_i_id , s_w_id , s_quantity	45%
3	hi_stock_1	s_w_id	40%
4	hi_stock_2	s_quantity	40%
5	hi_stock_0	s_i_id	30%

Tabela 4-7 Possíveis soluções de índices para a consulta descrita na Figura 4-26

Por fim, podem existir cenários em que heurísticas diferentes possam sugerir recomendações diferentes para um mesmo banco de dados. Nesses casos, o DBA nem sempre consegue identificar o motivo desta diferença. Com a instanciação do *framework* sobre o SGBD Oracle, consegue-se demonstrar o cenário em que duas heurísticas são executadas e acabam recomendando índices diferentes para serem criados no banco de dados.

No banco de dados existe uma tabela chamada EMPREGADO, contendo quatro colunas: matrícula, nome, sexo e salário (Tabela 4-8).

Empregado	
Ordem de criação	**Coluna**
0	matricula
1	nome
2	sexo
3	salario

Tabela 4-8 Lista de colunas da tabela Empregado

A consulta presente na carga de trabalho do banco de dados e que foi analisada pelo *framework* e consequentemente, pelas heurísticas de sintonia fina é descrita na Figura 4-28.

```
1.    SELECT *
2.    FROM empregado
3.    WHERE salario in (1000, 1500, 2000, 2500, 3000, 3500,
      4000, 4500, 5000)
4.      and sexo = 'M';
```

Figura 4-28 Consulta analisada pelo *framework*

Em um primeiro momento, habilitam-se no *framework* apenas as regras (condições e ações) de uma determinada heurística de escolha de índices candidatos que vamos nomear de HEIC-A. Analisando o banco de dados, gera-se o plano de execução da consulta (Figura 4-29) e verifica-se que o otimizador utiliza dois índices hipotéticos (HI_EMPREGADO_3 - coluna salário e HI_EMPREGADO_2 – coluna sexo) que foram criados pela heurística HEIC-A.

```
SQL>select * from empregado
where salario in (1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000) and sexo='M';

1453 rows selected.

Elapsed: 00:00:02.03

Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=198 Card=754 Bytes=18850)
   1    0   TABLE ACCESS (BY INDEX ROWID) OF 'EMPREGADO' (Cost=198 Card=754 Bytes=18850)
   2    1     BITMAP CONVERSION (TO ROWIDS)
   3    2       BITMAP AND
   4    3         BITMAP OR
   5    4           BITMAP INDEX (SINGLE VALUE) OF 'HI_EMPREGADO_3'
   6    4           BITMAP INDEX (SINGLE VALUE) OF 'HI_EMPREGADO_3'
   7    4           BITMAP INDEX (SINGLE VALUE) OF 'HI_EMPREGADO_3'
   8    4           BITMAP INDEX (SINGLE VALUE) OF 'HI_EMPREGADO_3'
   9    4           BITMAP INDEX (SINGLE VALUE) OF 'HI_EMPREGADO_3'
  10    4           BITMAP INDEX (SINGLE VALUE) OF 'HI_EMPREGADO_3'
  11    4           BITMAP INDEX (SINGLE VALUE) OF 'HI_EMPREGADO_3'
  12    4           BITMAP INDEX (SINGLE VALUE) OF 'HI_EMPREGADO_3'
  13    4           BITMAP INDEX (SINGLE VALUE) OF 'HI_EMPREGADO_3'
  14    3         BITMAP INDEX (SINGLE VALUE) OF 'HI_EMPREGADO_2'
```

Figura 4-29 Plano de execução após ações de sintonia fina de HEIC-A

Posteriormente, sobre o mesmo estado inicial do banco de dados, ou seja, sem as ações de sintonia fina de HEIC-A, habilitam-se no *framework* apenas as regras referentes à outra heurística HEIC-B e executa-se o *framework* novamente.

Analisando o banco de dados após a execução do *framework*, obtém-se como resultado o plano de execução apresentado na Figura 4-30, onde o otimizador não utiliza índices para executar a consulta.

```
SQL> select * from empregado
where salario in (1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000) and sexo='M';

1453 rows selected.

Elapsed: 00:00:03.01

Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=601 Card=754 Bytes=18850)
   1    0   TABLE ACCESS (FULL) OF 'EMPREGADO'(Cost=601 Card=754 Bytes=18850)
```

Figura 4-30 Plano de execução após ações de sintonia fina de HEIC-B

Com isso, o DBA pode ficar na dúvida sobre qual recomendação deve seguir: a criação de um índice ou não. Diante da dúvida, o DBA inicia a sua investigação sobre ambas heurísticas.

Primeiramente, o DBA verifica quais as condições de regras que geraram os índices hipotéticos, usando a consulta descrita na Figura 4-31 ao final da execução do framework para cada uma das heurísticas.

```
1.      Acao(?acao)
2.      ^ CondicaoRegra(?regra)
3.      ^ temDescricaoRegra(?regra, ?descRegra)
4.      ^ gera(?regra, ?acao)
5.      ^ temDescricao(?acao, ?descAcao)
6.      ^ swrlb:startsWith(?descAcao, "create hypothetical index")
7.      → sqwrl:select(?descRegra)
```

Figura 4-31 Consulta para obter as condições de regras que geraram os índices hipotéticos

Como resultado, o DBA interpreta a condição da regra, identificando que ambas heurísticas consideram as colunas referenciadas na cláusula `WHERE`. Como a condição das regras é a mesma, o DBA não consegue entender o motivo que gerou as recomendações diferentes.

O DBA passa a verificar as informações sobre cada índice hipotético gerado por cada heurística. Através da consulta descrita na Figura 4-32, o DBA pode obter as informações dos índices na ontologia de domínio.

```
1.      Indice(?indice)
2.      ^ temNome(?indice, ?nomeIndice)
3.      ^ temValorEstimadoCriacao(?indice, ?criacaoInd)
4.      ^ EstruturaDados(?estr)
5.      ^ implementa(?indice, ?estr)
6.      ^ abox:hasClasse(?estr, ?classeEstr)
7.      → sqwrl:select(?nomeIndice, ?criacaoInd, ?classeEstr)
```

Figura 4-32 Consulta para obter as informações dos índices

Ao final da execução do framework com as regras da heurística HEIC-A, o resultado da consulta apresenta os dois índices usados pelo otimizador, indicando as suas respectivas informações, conforme Tabela 4-9.

Nome	Custo de criação	Tipo
HI_EMPREGADO_2	0.5625	Bitmap
HI_EMPREGADO_3	4	Bitmap

Tabela 4-9 Informações dos índices hipotéticos criados por HEIC-A

Já na Tabela 4-10, verifica-se que a outra heurística HEIC-B também criou os mesmos índices hipotéticos.

Nome	Custo de criação	Tipo
HI_EMPREGADO_2	13	ÁrvoreB+
HI_EMPREGADO_3	17	ÁrvoreB+

Tabela 4-10 Informações dos índices hipotéticos criados por HEIC-B

Analisando as duas tabelas com as informações dos índices, o DBA conclui que a heurística HEIC-B não considera o tipo de estrutura de índice *Bitmap*, que a outra heurística HEIC-A considera. Por esse motivo, os índices criados por HEIC-B acabam sendo mais custosos para o banco de dados, justificando o fato do otimizador não optar por utilizá-los.

Todos os cenários apresentados nesta seção demonstram, de alguma forma, como o *framework* pode ser usado para justificar as decisões inesperadas de sintonia fina que são tomadas através de execuções de heurísticas, usando a semântica atribuída ao domínio e tarefa de sintonia fina.

4.2. Combinação de regras

A partir dos cenários descritos anteriormente, pode-se observar que a maioria das heurísticas raramente considera a combinação de regras, o que acarreta, muitas vezes, em uma otimização local. Com o uso do *framework*, já se pode pensar em uma combinação de regras, que possivelmente nos leva a heurísticas mais globais, pois existem alternativas que podem ser complementares.

Com o auxílio da ontologia de tarefa, definida no capítulo 3, é possível definir um passo-a-passo para realizar estas combinações entre regras de heurísticas e comparar as ações propostas.

4.2.1. Passo-a-passo para realizar combinações de heurísticas

Todas as regras usadas por heurísticas de sintonia fina podem ser expressas na ontologia de tarefa, desde que usem conceitos definidos na ontologia de domínio.

Cada heurística é responsável por diversas ações. Primeiramente, devemos **classificar as heurísticas** de acordo com suas possíveis ações de sintonia fina. Essa classificação é realizada manualmente, mas pode ser automatizada de acordo com a sintaxe de comandos de ações que as heurística geram sobre o banco de dados. Uma vez classificadas, podemos combinar heurísticas que executam o mesmo tipo de ação (mesma classificação) ou heurísticas que executam ações diferentes (classificações diferentes) e que já foram ou não combinadas antes. Por exemplo, podemos combinar heurísticas de escolha de índices candidatos, tais como: HEICL e a heurística apresentada no capítulo 4 do livro de (Bruno, 2011) (HEICB). Nesse caso, as heurísticas resultariam no mesmo tipo de ação, que é a criação de índices hipotéticos que podem ser candidatos a criação física, para agilizar o desempenho da base de dados. Por outro lado, também podemos combinar heurísticas com ações diferentes, mas que são compatíveis entre si. Por exemplo, podemos combinar:

- Exemplo de combinação de heurística 1 (ECH1): HEICL com a heurística de criação de índices proposta por (HCIM) (Monteiro et al, 2012).
- Exemplo de combinação de heurística 2 (ECH2): Heurística de escolha de índices candidatos proposta por (HEICM) (Monteiro et al, 2012) e a HBM.

Ao final deste passo de classificação das heurísticas, obtêm-se grupos de heurísticas de acordo com as ações finais que cada grupo pode realizar sobre o banco de dados.

Posteriormente, podemos **analisar a compatibilidade entre os conceitos e ações das heurísticas** a serem combinadas. Para verificarmos a compatibilidade de heurísticas que estão classificadas em um mesmo grupo, basta analisar os

conceitos que são definidos como pré-condições das heurísticas. Se elas trabalham com o mesmo conjunto de conceitos, elas são possivelmente compatíveis. Caso contrário, se mesmo assim o usuário quiser combinar as heurísticas, ele pode verificar as instâncias dos conceitos que não estão sendo contemplados por uma das heurísticas e analisar a viabilidade e o impacto da inclusão dos mesmos.

No caso de heurísticas que estejam presentes em grupos diferentes, precisamos analisar quais conceitos são pré-condições para cada heurística e a ordem de execução dos tipos de ações de heurísticas. No caso dos exemplos ECH1 e ECH2, a ordem das ações é primeiramente sugerir índices candidatos e posteriormente realizar a manutenção de índices reais. Dessa forma, conseguimos ver que é uma ordem que faz sentido, pois uma heurística vai sugerir índices candidatos que vão ser usados por outra heurística que vai tomar ações de sintonia fina de índices. Sendo assim, em termos de ações, as heurísticas podem ser combinadas. Além disso, os conceitos que são pré-condições precisam ser instanciados no momento correto. Por exemplo, em ECH1, os conceitos que são pré-condições de HCIM precisam ser instanciados pelas regras definidas na ontologia de domínio, pela heurística anterior HEICL ou pela própria heurística em execução (HCIM). Caso os conceitos não estejam instanciados no momento correto, as heurísticas são consideradas incompatíveis.

Ao final do passo de análise de compatibilidade entre as heurísticas, obtêm-se grupos de heurísticas compatíveis.

Uma vez definidas as compatibilidades entre as heurísticas, **o *framework* pode ser executado com as regras das heurísticas compatíveis habilitadas**. Ao executar todas as regras definidas, a máquina de regras vai gerar todas as possíveis combinações de ações resultantes. Nesse momento, o DBA precisa **analisar o resultado e priorizar as ações**. Com isso, o DBA obtém um conjunto de ações de sintonia fina. Diante destas ações, o DBA **identifica quais delas são conflitantes**, são mutuamente exclusivas. A partir dessa análise, o DBA pode definir novas regras para refinar o processo de combinação, evitando cada vez mais ações que não façam sentido. Ao final do passo de identificação de conflitos, o DBA passa a ter um conjunto de ações mais refinado. Por fim, o **DBA toma a decisão** sobre quais ações resultantes da combinação de heurísticas ele quer executar na base de dados.

Para realizar a comparação entre as heurísticas, basta o DBA executar o processo de combinação para obter as ações sobre todas as regras e habilitar/desabilitar as regras de cada heurística separadamente e re-executar o *framework*. Dessa forma, o DBA consegue visualizar o resultado de ações sugeridas por cada heurística individualmente e o resultado combinado. Com os resultados de possíveis ações, o DBA consegue compará-los.

4.2.2. Exemplo de cenário de combinação de regras

Para ilustrar a combinação de regras de heurísticas diferentes e demonstrar que essa combinação pode ser útil, selecionamos as heurísticas HEICL e HEICB. Ambas heurísticas estão classificadas no mesmo grupo de ação: escolha de índices candidatos. Como a heurística HEICL foi implementada no SGBD PostgreSQL no trabalho de (Salles, 2004), optamos por restringir alguns conceitos usados pela heurística HEICB e não implementadas no SGBD PostgreSQL. Então, apenas para fins de testes, a heurística HEICB não usará os seguintes conceitos e restrições:

- colunas incluídas (colunas não chaves, que não são consideradas no cálculo do número de colunas chaves do índice e no tamanho do índice).
- índice *cluster* e índice não *cluster*. O PostgreSQL *clusteriza* a tabela baseada em um índice, mas quando a tabela é atualizada posteriormente, as atualizações não são *clusterizadas* automaticamente.

Conforme descrito anteriormente, para selecionar índices candidatos para comandos, a heurística HEICL combina regras, buscando colunas em cinco grupos:

- EQ: colunas que são referenciadas em predicados de igualdade;
- O: colunas que são referenciadas em cláusulas `ORDER BY`, `GROUP BY` e predicados de junção;
- RANGE: colunas que são referenciadas em restrições de intervalos;

- SARG: colunas que são referenciadas em outros predicados indexáveis, tal como `like`;
- REF: demais colunas referenciadas no comando.

Uma vez identificadas as colunas de cada grupo, a heurística HEICL usa regras para combinar as colunas dos grupos e gerar as ações de criação de índices hipotéticos da seguinte maneira:

- EQ + O
- EQ + O + RANGE
- EQ + O + RANGE + SARG
- EQ + O + RANGE + REF
- O + EQ + RANGE
- O + EQ + RANGE + SARG
- O + EQ+ RANGE + REF

A heurística HEICL forma índices hipotéticos com múltiplas colunas, em ordem, eliminando colunas duplicadas.

Já a heurística HEICB opta somente por índices de cobertura, ou seja, índices que possuam todas as colunas presentes no comando. Além disso, diferente da HEICL, a heurística HEICB realiza permutações sobre a ordem das colunas indexáveis. São consideradas colunas indexáveis aquelas presentes em EQ, O, RANGE e SARG.

Como as heurísticas estão classificadas no mesmo grupo de possíveis ações, basta que seus conceitos definidos como pré-condições sejam iguais. O DBA analisa os conceitos e valida que é o mesmo conjunto. Na Tabela 4-11 é apresentada a lista de conceitos definidos para ambas heurísticas.

Ordem de instanciação	Classe/Conceito
1	SGBD
2	ComandoDML
3	Clausula
4	Token
5	ObjetoSGBD
6	ExpressaoPredicado

Tabela 4-11 Conceitos usados como pré-condições para HEICL e HEICB

Sendo o mesmo conjunto de conceitos, as heurísticas são definidas como sendo possivelmente compatíveis. Com isso, o *framework* pode ser executado com todas as regras das heurísticas HEICL e HEICB habilitadas.

Executamos essas regras sobre a base de dados e carga de trabalho do TPC-C, no SGBD PostgreSQL. Analisando algumas etapas de sugestões de índices hipotéticos, podemos verificar que a heurística HCB, que atribui benefícios aos índices, adiciona benefícios tanto para índices resultantes de HEICL quanto para índices resultantes de HEICB.

Para visualizarmos melhor essa combinação, optamos por nomear os índices da seguinte forma:

hi + <m|b> + <nome da tabela indexada> + <ordem das colunas indexadas>

Onde,

- hi indica que é um índice hipotético;
- <l|b> é usado para referenciar se o índice foi originado de HEICL (l) ou de HEICB (b);
- <nome da tabela indexada> corresponde ao nome da tabela onde o índice será definido;
- <ordem das colunas indexadas> corresponde a ordem das colunas indexadas pelo índice.

Na Tabela 4-12, apresentamos alguns comandos da carga de trabalho com os índices escolhidos por HCB usando as heurísticas combinadas. Como pode ser visto, nas linhas 1 e 3, o índice que teve benefício atribuído é um índice que jamais seria escolhido pela heurística HEICB, pois o mesmo não é um índice de cobertura. No entanto, ele que foi considerado por HCB como sendo o índice a obter benefícios. E nas linhas 2, 4 e 5, a heurística HCB selecionou índices de HEICB, cuja ordem das colunas não seria definida por HEICL.

Dessa forma, conseguimos verificar para as heurísticas usadas, que através da combinação de regras, conseguimos ter novas discussões sobre a prática de sintonia fina, realizando combinações e comparando-as. Antes mesmo de executar as ações na base de dados e verificar as escolhas realizadas por HCB, conseguimos analisar e comparar as sugestões de ações de HEICL combinada com HEICB. A Tabela 4-13 apresenta alguns resultados. Nessa tabela são

destacados os índices escolhidos por HCB entre as alternativas de índices hipotéticos (em negrito) e os índices pensados em comum por ambas heurísticas HEICL e HEICB (em vermelho). Com isso, o DBA consegue analisar que as duas heurísticas combinadas geram um conjunto maior de alternativas de soluções, podendo ser uma análise mais completa e, talvez, mais eficaz.

Nº linha	**Comando DML**	**HEICL + HEICB**	
		Índice selecionado por HCB	**Colunas indexadas**
1	`SELECT no_o_id FROM new_order WHERE no_w_id = 1 AND no_d_id = 1;`	hi_l_new_order_1_2	no_d_id, no_w_id
2	`DELETE FROM new_order WHERE no_o_id = 1 AND no_w_id = 1 AND no_d_id = 1;`	hi_b_new_order_0_2_1	no_o_id, no_w_id, no_d_id
3	`SELECT o_c_id FROM orders WHERE o_id = 1 AND o_w_id = 1 AND o_d_id = 1;`	hi_l_orders_0_1_2	o_id, o_d_id, o_w_id
4	`UPDATE orders SET o_carrier_id = 1 WHERE o_id = 1 AND o_w_id = 1 AND o_d_id = 1;`	hi_b_orders_0_2_1	o_id, o_w_id, o_d_id
5	`UPDATE order_line SET ol_delivery_d = current_timestamp WHERE ol_o_id = 1 AND ol_w_id = 1 AND ol_d_id = 1;`	hi_b_order_line_0_2_1	ol_o_id, ol_w_id, ol_d_id

Tabela 4-12 Índices selecionados por HCB diante do conjunto de índices hipotéticos criados pela combinação de HEICL e HEICB

Nº linha	Comando DML	Índices sugeridos (HEICL + HEICB)

1	`SELECT no_o_id`
	`FROM new_order`
	`WHERE no_w_id = 1`
	`  AND no_d_id = 1;`
		hi_l_new_order_1_2
		hi_ l_new_order_1_2_0
		hi_ l_new_order_1
		hi_ l_new_order_2
		hi_b_new_order_1_2_0
		hi_b_new_order_2_1_0

2	`DELETE FROM`
	`new_order WHERE`
	`no_o_id = 1`
	`  AND no_w_id = 1`
	`  AND no_d_id = 1;`
		hi_ l_new_order_0_1_2
		hi_ l_new_order_0
		hi_ l_new_order_1
		hi_ l_new_order_2
		hi_b_new_order_0_1_2
		hi_b_new_order_0_2_1
		hi_b_new_order_1_0_2
		hi_b_new_order_1_2_0
		hi_b_new_order_2_0_1
		hi_b_new_order_2_1_0

3	`SELECT o_c_id`
	`FROM orders`
	`WHERE o_id = 1`
	`  AND o_w_id = 1`
	`  AND o_d_id = 1;`
		hi_ l_orders_0_1_2
		hi_ l_orders_0_1_2_3
		hi_ l_orders_0
		hi_ l_orders_1
		hi_ l_orders_2
		hi_b_orders_0_1_2_3
		hi_b_orders_0_2_1_3
		hi_b_orders_1_0_2_3
		hi_b_orders_1_2_0_3
		hi_b_orders_2_0_1_3
		hi_b_orders_2_1_0_3

4	`UPDATE orders`
	`SET o_carrier_id = 1`
	`WHERE o_id = 1`
	`  AND o_w_id = 1`
	`  AND o_d_id = 1;`
		hi_ l_orders_0_1_2
		hi_ l_orders_0
		hi_ l_orders_1
		hi_ l_orders_2
		hi_b_orders_0_1_2
		hi_b_orders_0_2_1
		hi_b_orders_1_0_2

		hi_b_orders_1_2_0 hi_b_orders_2_0_1 hi_b_orders_2_1_0
5	UPDATE order_line SET ol_delivery_d = current_timestamp WHERE ol_o_id = 1 AND ol_w_id = 1 AND ol_d_id = 1;	hi_ l_order_line_0_1_2 hi_ l_order_line_0 hi_ l_order_line_1 hi_ l_order_line_2 hi_b_order_line_0_1_2 **hi_b_order_line_0_2_1** hi_b_order_line_1_0_2 hi_b_order_line_1_2_0 hi_b_order_line_2_0_1 hi_b_order_line_2_1_0

Tabela 4-13 Índices hipotéticos criados pela combinação de HEICL e HEICB

Ao final da execução das heurísticas HEICL e HEICB, utilizamos a heurística HBM, explicada no início desse capítulo, para verificar se a combinação de regras realmente faz sentido. A heurística HBM identificou o índice hi_l_new_order_1_2, gerado pelas regras de HEICL, como sendo o melhor índice para diminuir o custo de execução da consulta descrita na Figura 4-33. Por outro lado, a mesma heurística HBM identificou o índice hi_b_new_order_0_2_1, gerado pelas regras de HEICB, como sendo o melhor índice para diminuir o custo de execução da consulta descrita na Figura 4-34.

```
1.    SELECT no_o_id
2.    FROM new_order
3.    WHERE no_w_id = 1 AND no_d_id = 1;
```

Figura 4-33 Consulta identificada por HBM como benéfica para índice hi_l_new_order_1_2

```
1.    DELETE FROM new_order
2.    WHERE no_o_id = 1
3.       AND no_w_id = 1
4.       AND no_d_id = 1;
```

Figura 4-34 Consulta identificada por HBM como benéfica para índice hi_b_new_order_0_2_1

Ambas as consultas estão presentes na carga de trabalho em que o *framework* foi executado. Como a heurística HBM escolheu como melhores índices para a carga de trabalho, àqueles gerados por regras de heurísticas diferentes, pode-se concluir que se as heurísticas fossem usadas individualmente talvez não conseguíssemos obter os melhores índices indicados por HBM.

4.3. Implementação do *Framework*

Nesta seção apresenta-se a arquitetura e modelagem parcial do *framework*, utilizando a notação UML. Além disto, descreve-se as restrições tecnológicas encontradas e que impediram a implementação do *framework* no prazo disponível até a defesa da presente tese.

4.3.1. Arquitetura do *Framework*

A arquitetura do framework está subdividida em 3 módulos, cuja interdependência está esquematizada na **Figura 4-35**.

O módulo **Administrador** realiza a interface com os usuários, de forma a permitir que eles realizem a manutenção (inclusão, alteração, remoção e consulta) da ontologia de aplicação, composta pela ontologia de domínio e ontologia de tarefa. Além disso, esse módulo permite a interação do usuário com as decisões sobre as ações de sintonia fina, sugeridas pelo **Agente Tunador**. Como a manutenção da ontologia pode ser realizada usando a ferramenta Protégé e a modelagem realizada no presente trabalho detalha a tarefa de auto-sintonia, ou seja, sem a intervenção de usuários para realizar as ações de sintonia fina sobre os sistemas de bancos de dados, o módulo Administrador será detalhado em trabalhos futuros.

O módulo **Agente Tunador** administra as instâncias da ontologia de aplicação e captura os comandos DML que são submetidos aos sistemas de bancos de dados. Além disso, ele realiza a interface com os sistemas de bancos de dados capturando informações para as instâncias de conceitos e propriedades da ontologia bem como realizando ações de sintonia fina derivadas de regras validadas de heurísticas (originadas da ontologia de tarefa) nos bancos de dados.

O módulo de **Drivers dos SGBDs** é responsável por realizar as conexões com os SGBDs para que o Agente Tunador possa receber os comandos DML, capturar as informações necessárias para instanciar a ontologia de aplicação e realizar as ações de sintonia fina.

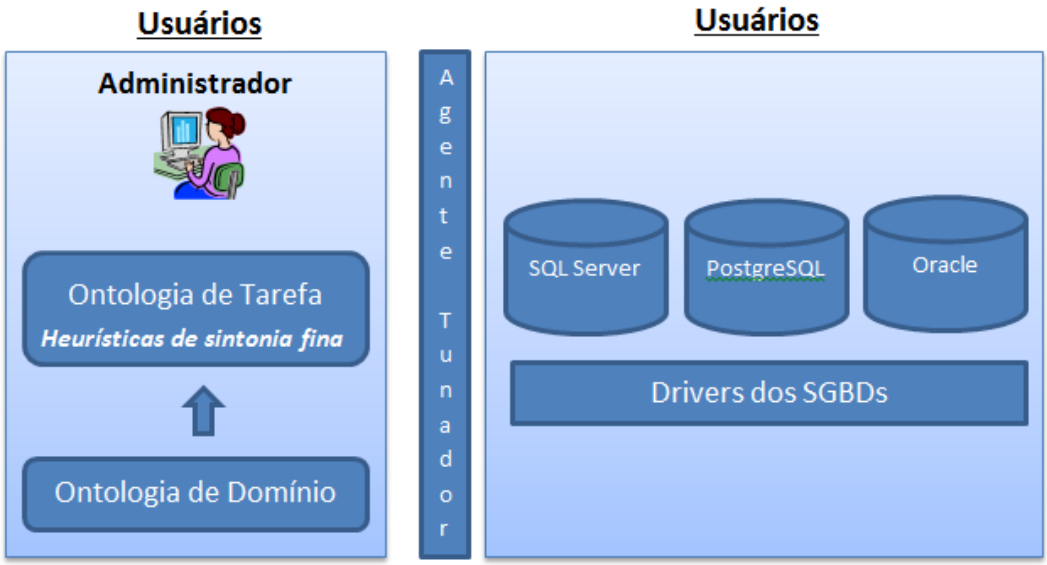

Figura 4-35 Arquitetura do *Framework*

Conforme ilustrado na **Figura 4-35**, os usuários do *framework* podem ser tanto o DBA humano como o DBA automático (auto-sintonia). O DBA humano faz uso do módulo Administrador para manter os conceitos e propriedades da ontologia de aplicação, realizar consultas em busca de argumentos para as ações de sintonia fina sugeridas por heurísticas ou para aceitar e recusar tais sugestões. O DBA automático é entendido como sendo os próprios bancos de dados, visto que estes fornecem as informações para que a ontologia seja instanciada bem como sofrem as ações de sintonia fina, que são sugeridas pelas heurísticas definidas na ontologia de tarefa.

Imaginando a execução de uma determinada heurística de forma automática (auto-sintonia), o sistema de banco de dados, após ter a sua conexão estabelecida com o *framework*, sinaliza que recebeu um comando DML e o agente tunador captura tal comando. A partir daí, o agente verifica e inicia a execução da heurística ativa na ontologia de tarefa. Para que a heurística possa ser executada, ela precisa ter todas as suas pré-condições satisfeitas de acordo com as instâncias

da ontologia de domínio. Uma vez que todas as pré-condições sejam validadas, a heurística é então executada. De acordo com as regras válidas, a ontologia de tarefa resulta em ações que são encaminhadas para o agente tunador. Em seguida, o agente providencia as ações de sintonia fina diretamente nos bancos de dados utilizando as conexões com os mesmos através do uso dos *drivers*. Com isso, os sistemas de bancos de dados são otimizados de acordo com as regras das heurísticas selecionadas como ativas na ontologia de tarefa. Caso o DBA queira maiores detalhes, argumentos e justificativas sobre as ações realizadas de sintonia fina, basta que o mesmo consulte as informações sobre as instâncias da ontologia de aplicação. É importante ressaltar que todo o comportamento do *framework* que resulta em uma ou mais ações de sintonia fina está sendo refletido na ontologia de aplicação, sempre agregando semântica ao processo de sintonia fina.

4.3.2. Modelagem do *Framework*

Para um melhor entendimento das funcionalidades pensadas para o framework, apresenta-se a modelagem parcial do mesmo fazendo uso da notação UML.

Na **Figura 4-36**, apresenta-se o diagrama de casos de uso. O primeiro caso de uso refere-se à tarefa de auto-sintonia, ou seja, executar heurísticas de sintonia fina de forma automática no sistema de banco de dados. Após a realização da auto-sintonia, o DBA possui todo o registro das atividades realizadas no banco de dados, dado que tais atividades são refletidas em instâncias da ontologia de aplicação. Através dessas instâncias, o DBA tem como consultar informações que ajudem em argumentos e justificativas para as ações de sintonia fina realizadas. A forma de consulta à ontologia já foi demonstrada anteriormente neste capítulo.

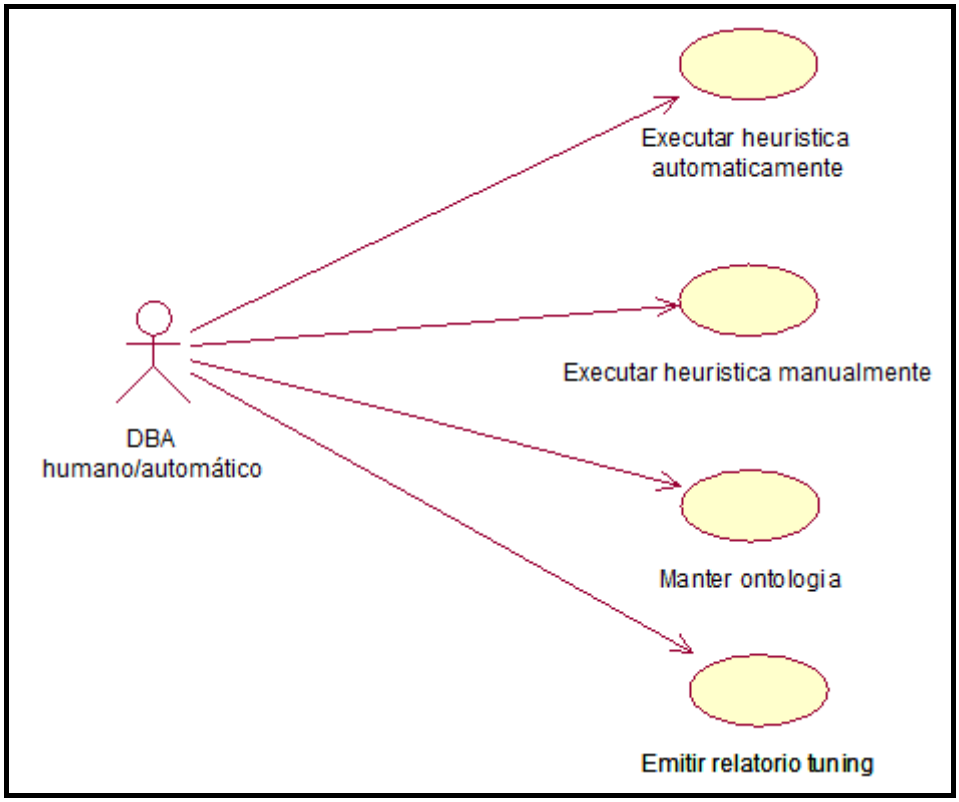

Figura 4-36 Diagrama de casos de uso do *framework*

Outro caso de uso apresentado no diagrama (**Figura 4-36**) refere-se à execução de heurísticas de sintonia fina de banco de dados de forma manual. Neste caso, o DBA pode selecionar as heurísticas de sintonia fina que ele deseja executar (na auto-sintonia, tal seleção pode ser realizada antecipadamente na manutenção da ontologia), concordar ou discordar com as ações sugeridas pelas heurísticas e descrever um complemento para a sua decisão sobre a ação, justificando, por exemplo, a sua discordância.

O caso de uso para manter ontologia permite a inserção, alteração, remoção e consulta de conceitos, propriedades e regras da ontologia de aplicação. Através desse caso de uso que o DBA pode consultar as instâncias da ontologia em busca de argumentos para as decisões de sintonia fina. É importante lembrar que existem diversas ferramentas na literatura que permitem a manutenção de ontologias. Nesta tese, usou-se a ferramenta Protégé.

Por fim, o diagrama apresenta o caso de uso de emitir relatório de tuning (sintonia fina). Tal caso de uso busca elaborar um relatório com as atividades de sintonia fina que foram realizadas no banco de dados, acrescentando justificativas

para tais ações e qualquer outra informação relevante que esteja presente na ontologia de aplicação.

A **Figura 4-37** apresenta o diagrama de classes inicial para o *framework*. É evidente que os módulos de administração e dos drivers envolverão outras classes. Contudo, tais módulos ainda não foram estudados e detalhados de forma a permitir tal especificação no diagrama de classes.

A classe SGBD é responsável por realizar, de maneira geral, a conexão do *framework* com o SGBD (módulo de **Drivers dos SGBDs**). Essa conexão é necessária para capturar informações do SGBD e também para realizar as ações de sintonia fina no banco de dados. A classe SGBD é considerada um ponto flexível (*hot-spot*) do *framework* porque ela pode ser estendida para os diversos SGBDs existentes no mercado, tais como: PostgreSQL, Oracle, SQL Server, MySQL, DB2 etc.

A classe de AgenteTunador (módulo do **Agente Tunador**) é como se fosse um controlador do *framework*. Essa classe é responsável por gerenciar o comportamento do *framework*, verificando e executando as heurísticas de sintonia fina que estão ativas, ou seja, aquelas que devem ser usadas para realizar a sintonia fina no banco de dados. A classe de AgenteTunador é considerada ponto fixo (*frozen-spot*), pois segue o metamodelo do *framework*, já definido pela ontologia de tarefa.

A classe de ontologia (módulo **Administrador**) também é considerada um ponto fixo, pois a sua implementação sempre será a mesma, independente do arquivo de ontologia que será indicado para o *framework*. Tal classe é responsável por indicar a localização da ontologia de sintonia fina para que o *framework* possa utilizá-la.

As demais classes fazem parte do metamodelo (ontologia de tarefa) do *framework*, já explicado no capítulo anterior. Todas as classes do metamodelo são consideradas pontos flexíveis, já que dependem das definições dos seus respectivos conceitos na ontologia de tarefa. Como a ontologia é extensível, suas classes também devem ser.

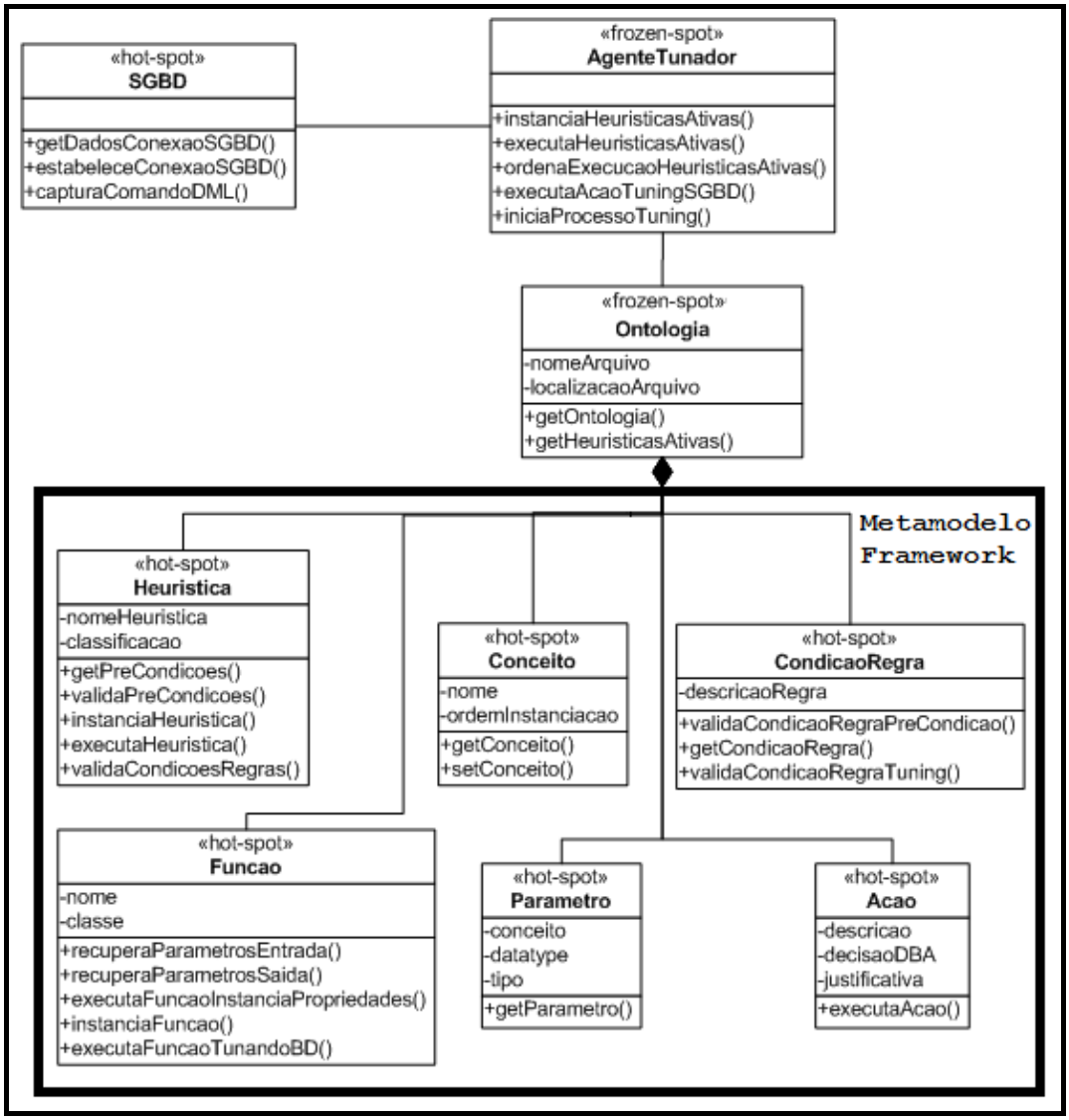

Figura 4-37 Diagrama de classes do *framework*

A **Figura 4-38** apresenta o diagrama de sequência para o caso de uso de executar heurística automaticamente.

Inicialmente, o agenteTunador recupera o arquivo que possui a ontologia de aplicação, que vai ser a responsável por manter a semântica de todo o processo de sintonia fina. Posteriormente, o agenteTunador estabelece a conexão com o banco de dados e inicia a captura por comandos DML da carga de trabalho.

O agenteTunador inicia o processo de sintonia fina, instanciando o conceito de comandoDML na ontologia e no próprio *framework*. Uma vez que a carga de trabalho está sendo executada no banco de dados, o framework já pode executar as heurísticas definidas como ativas na ontologia. Para isso, as heurísticas ativas são recuperadas, instanciadas, ordenadas de acordo com o conjunto de pré-

condições já definidas na ontologia, suas pré-condições são validadas e enfim, são executadas, gerando ações de sintonia fina no banco de dados.

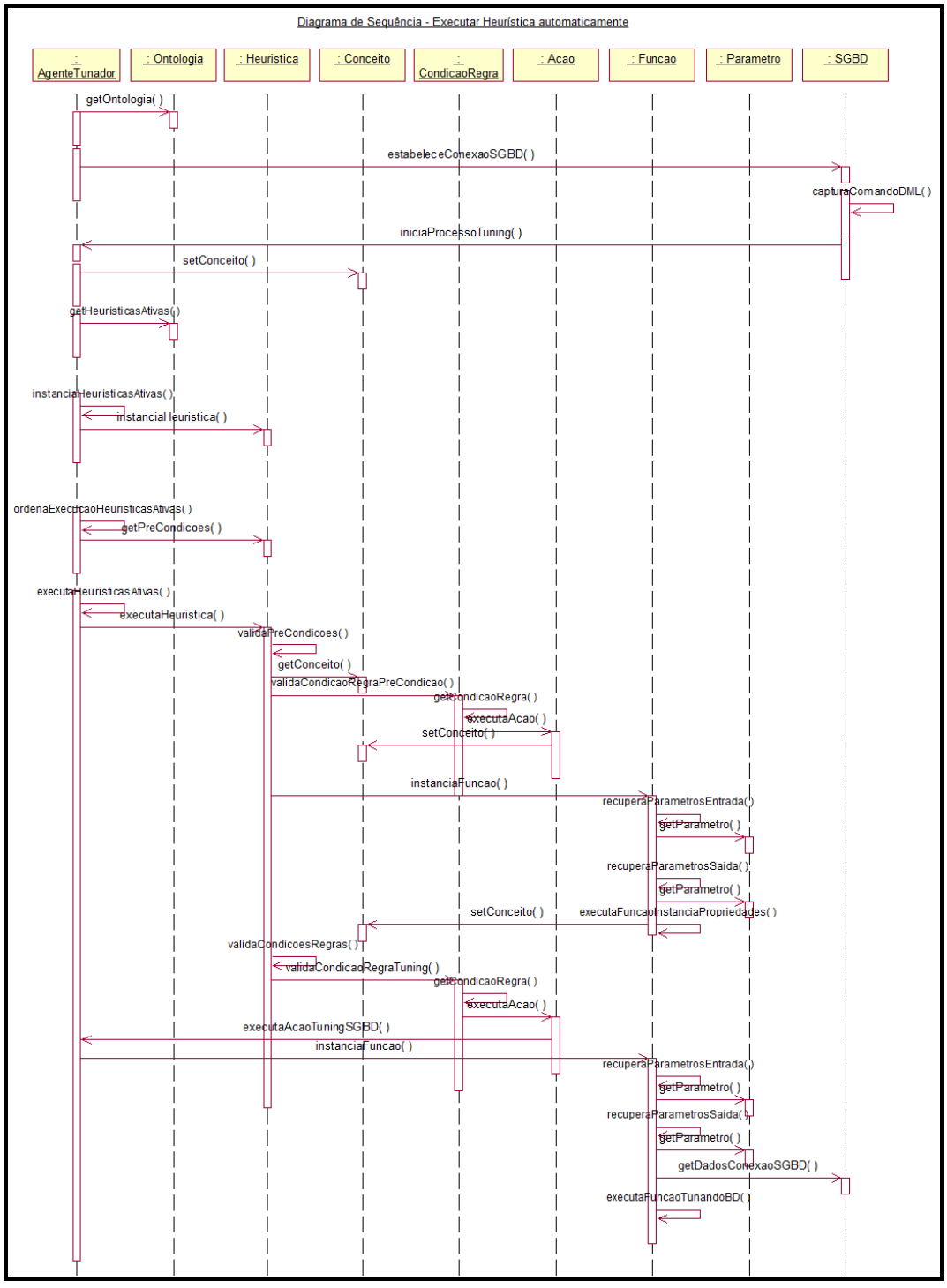

Figura 4-38 Diagrama de sequência – Executar heurística automaticamente

Durante a execução das heurísticas, os conceitos que são considerados suas pré-condições são instanciados através de ações derivadas de condições de regras verdadeiras ou de resultados de execuções de funções implementadas.

O resultado da execução da heurística, que é de fato a ação de sintonia fina, é obtido através da validação de determinadas condições de regra já definidas pela heurística na ontologia de tarefa. Esse resultado é transformado em comandos DDL, que são executados por funções, diretamente na base de dados.

Os diagramas de sequência dos demais casos de uso não foram elaborados. O caso de uso de execução manual das heurísticas deve possuir um diagrama de sequência similar ao apresentado, tendo que acrescentar os passos que contemplem as interações com o DBA (escolha de heurísticas, indicação de decisão sobre sugestão das heurísticas e descrição de justificativas). O caso de uso de manter ontologia deve possuir um diagrama de sequência que contenha interações com a API do Protégé, já que tal API possui funções prontas para manter os objetos presentes em uma ontologia. Por fim, o diagrama de emissão de relatório deve possuir um diagrama de sequência para interagir com os conceitos da ontologia e formatar o relatório da maneira que o DBA precisa.

4.3.3. Restrições tecnológicas

Optou-se por descrever a ontologia de aplicação na ferramenta Protégé (Protégé, 2013), pois a mesma possui as seguintes características:

- É uma ferramenta gratuita;
- Possibilita a descrição de ontologias em OWL-DL;
- Possui máquinas de inferências já acopladas.

Para definir as regras usadas no domínio de sintonia fina, escolheu-se a linguagem SWRL (O'Connor et al, 2005), que já consta na ferramenta usada para definir a ontologia (Protégé). Além disso, ela permite o uso de novas bibliotecas, até mesmo aquelas definidas pelos seus próprios usuários, o que torna a linguagem muito rica em recursos de representação.

No entanto, a linguagem SWRL não permite o uso do conector "ou" no Protégé (stackoverflow, 2013), o que dificulta, mas não impossibilita a definição e

o processamento de alguns conjuntos de regras. Por exemplo, ao definir regras que considerem as cláusulas de um comando DML para extrair os atributos potencialmente úteis de serem indexados em uma heurística de escolha de índices candidatos para a sintonia fina de um banco de dados.

Como os comandos DML possuem cláusulas que são opcionais, é necessário criar uma regra para cada opção de combinação das cláusulas. No caso de uma heurística de escolha de índices candidatos que decida selecionar as colunas que estejam presentes em cláusulas `where` e usem o operador de equivalência ("="), é necessário que sejam criadas múltiplas regras que combinem:

- Comandos DML que possuam
  - cláusula `SELECT` e
  - cláusula `FROM` e
  - cláusula `WHERE` e
  - operador "=";
- Comandos DML que possuam
  - cláusula `SELECT` e
  - cláusula `FROM` e
  - cláusula `WHERE` e
  - operador "=" e
  - cláusula `ORDER BY`;
- Comandos DML que possuam
  - cláusula `SELECT` e
  - cláusula `FROM` e
  - cláusula `WHERE` e
  - operador "=" e
  - cláusula `ORDER BY` e
  - cláusula `GROUP BY`;

- Comandos DML que possuam
  - cláusula `SELECT` e
  - cláusula `FROM` e
  - cláusula `WHERE` e

    - operador “=” e
    - cláusula `ORDER BY` e
    - cláusula `GROUP BY` e
    - cláusula `HAVING`;
- Comandos DML que possuam
    - cláusula `SELECT` e
    - cláusula `FROM` e
    - cláusula `WHERE` e
    - operador “=” e
    - cláusula `GROUP BY`;
- Comandos DML que possuam
    - cláusula `SELECT` e
    - cláusula `FROM` e
    - cláusula `WHERE` e
    - operador “=” e
    - cláusula `HAVING`;
- Comandos DML que possuam
    - cláusula `SELECT` e
    - cláusula `FROM` e
    - cláusula `WHERE` e
    - operador “=” e
    - cláusula `ORDER BY` e
    - cláusula `HAVING`;
- Comandos DML que possuam
    - cláusula `SELECT` e
    - cláusula `FROM` e
    - cláusula `WHERE` e
    - operador “=” e
    - cláusula `GROUP BY` e
    - cláusula `HAVING`.

Se tivesse a disponibilidade de usar o operador “ou” no Protégé para as regras SWRL, não seria necessário criar uma regra para cada tipo de combinação, ou seja, bastaria uma regra.

Diante das limitações da ferramenta, optou-se por pesquisar outras ferramentas que pudessem dar suporte às limitações encontradas no Protégé. Segundo as pesquisas realizadas (Sirin et al, 2007) (KAON2, 2013) (FaCT++, 2013) (Snoogle, 2013), não foram encontradas ferramentas gratuitas que pudessem dar suporte ao processador de regras com resultados esperados pela ontologia de tarefa (metamodelo do *framework*). Optou-se por ferramentas gratuitas para encorajar o uso por parte dos DBAs no momento de estender a ontologia com novas regras ou realizar alterações em regras existentes.

Diante da preocupação em facilitar o trabalho do DBA, trazendo o menor impacto possível ao seu trabalho no momento de adicionar semântica a sintonia fina, decidiu-se avaliar a transformação das regras já definidas em SWRL para DL (*Description Logic*). A linguagem DL disponibiliza o operador “ou” na definição de suas regras, por isso a decisão de estudá-la.

Utilizando o trabalho apresentado por (Karimi, 2008), as regras definidas nessa tese podem ser alteradas para DL. Como exemplo, citam-se duas regras já descritas nessa tese, no capítulo 3, em SWRL para as suas equivalentes em DL.

Por exemplo, tem-se a regra definida pela heurística HBM para gerar a ação de criação de um índice real e a remoção de um índice hipotético, se verdadeira. A Figura 4-39 mostra a regra descrita em SWRL e a Figura 4-40 apresenta a sua versão em DL.

```
1.    IndiceHipotetico(?indHip)
2.    ^ valorCustoBeneficioAcumulado(?cba)
3.    ^ valorCustoEstimadoCriacao(?cc)
4.    ^ temCustoBeneficioAcumulado(?indHip, ?cba)
5.    ^ temCustoEstimadoCriacao(?indHip, ?cc)
6.    ^ swrlb:greaterThan(?cba, ?cc)
7.    -> Create(?indHip)
8.    ^ DropHypothetical(?indHip)
```

Figura 4-39 Regra SWRL para criação de índice real da heurística HBM

```
1.    IndiceHipotetico
2.    ⊓
3.    ∃temCustoBeneficioAcumulado.
4.    ∃temCustoEstimadoCriacao.
5.    ∃temCustoBeneficioAcumulado.
6.    >.
7.    temCustoEstimadoCriacao
8.    ⊑ Create.DropHypothetical
```

Figura 4-40 Regra DL para criação de índice real da heurística HBM

O mesmo ocorre com a outra regra definida em HBM para gerar a ação de remoção de um índice real e de criação de um índice hipotético, se verdadeira. A Figura 4-41 mostra a regra descrita em SWRL e a Figura 4-42 apresenta a sua versão e DL.

```
1.    IndiceReal(?indReal)
2.    ^ valorBonusAcumulado(?vba)
3.    ^ valorCustoEstimadoCriacao(?cc)
4.    ^ temValorBonusAcumulado(?indReal, ?vba)
5.    ^ temCustoEstimadoCriacao(?indReal, ?cc)
6.    ^ swrlb:mod(?mvba, ?vba)
7.    ^ swrlb:greaterThan(?mvba, ?cc)
8.    -> Drop(?indReal)
9.    ^ CreateHypothetical(?indReal)
```

Figura 4-41 Regra SWRL para remoção de índice real da heurística HBM

```
1.    IndiceReal
2.    ⊓
3.    ∃temValorBonusAcumulado.
4.    ∃temCustoEstimadoCriacao.
5.    ∃|temCustoBeneficioAcumulado|.
6.    >.
7.    temCustoEstimadoCriacao
8.    ⊆ Drop.CreateHypothetical
```

Figura 4-42 Regra DL para remoção de índice real da heurística HBM

Por possuir o operador “ou”, seria interessante alterarmos o domínio para usar a linguagem DL. No entanto, o Protégé apresenta outra limitação que é a não permissão do tratamento das consequências da regra (ações posteriores ao “→”), bem como a não criação de novos indivíduos (`swrlx:makeOWLThing`) como é realizado na regra SWRL. O Protégé não permite a integração do mundo lógico (regra em DL) com o mundo computacional (consequências).

Diante de tais restrições tecnológicas, optou-se por não realizar o esforço de transformar as regras já definidas em SWRL para DL e avaliar, posteriormente, qual seria a melhor forma de implementar o *framework* e definir suas regras.

Uma vez que se decida utilizar as regras na linguagem DL, deve ser pensada uma forma de tratar as consequências das regras e seus *built-ins*. *Built-in* é um predicado que pode ter um ou mais argumentos e que se realiza uma operação com eles. No caso desta tese, as regras seriam submetidas a um raciocinador (ex.: pellet (Sirin et al, 2007)) e uma vez consideradas verdadeiras, gerariam um plano de ação para as consequências no código fonte do *framework*.

Dessa forma, a implementação do *framework* proposto nesta tese torna-se um trabalho futuro devido aos problemas tecnológicos encontrados.

4.4. Considerações finais

Buscando apoiar a comprovação da eficácia do trabalho de um DBA ou de uma ferramenta de auto-sintonia, adiciona-se semântica ao trabalho de sintonia fina, através da construção de um *framework* que utiliza como base uma ontologia do domínio de sintonia fina.

Com uma semântica associada, o DBA consegue entender melhor as decisões tomadas por uma ferramenta de auto-sintonia e até mesmo discutir ou questionar as regras usadas por ela. Com isso, a ferramenta de auto-sintonia acabará sendo auto-explicável, pois o DBA terá todas as informações explícitas na ontologia. Através do primeiro cenário apresentado nesse capítulo, descreve-se como um DBA experiente questionaria uma ação de sintonia fina realizada por heurísticas. Através dos seus questionamentos e das respostas providas pela instância das ontologias, o DBA pode concluir se uma ferramenta é de fato confiável ou não. E no caso de não ser, o DBA pode até estender a ontologia com novos conhecimentos ou alterar as regras na instância da ontologia de tarefa (*framework*) em alto nível. Esse é um diferencial para as ferramentas de apoio ao processo de sintonia fina disponíveis na literatura até o momento. Caso um DBA discorde de algo nas ferramentas atuais e estas possuam código fonte aberto, ele vai ter que entender do código para realizar as modificações e não saberá ao certo qual será o impacto geral de tal modificação.

Outro cenário apresentado nesse capítulo apresenta uma empresa que contrata um DBA para realizar a atividade de sintonia fina em sua base de dados. Como o analista da empresa responsável por acompanhar a atividade possui um conhecimento mínimo sobre banco de dados, ele acaba realizando diversos questionamentos ao DBA. Sendo assim, o DBA precisa possuir argumentos em mãos para justificar as suas decisões e informar que foram pensadas diversas soluções para a base de dados. Dessa forma, o analista da empresa pode valorizar ainda mais o trabalho de sintonia fina, validando que o trabalho realmente considerou diversas alternativas de soluções para a base de dados.

O último cenário que demonstra a necessidade de justificativa de decisões de sintonia fina apresenta recomendações diferentes de heurísticas que sugerem

índices para o banco de dados. Diante disso, o *framework* é usado para identificar o motivo pelo qual essas recomendações acabam sendo diferentes, conseguindo argumentos para justificar a decisão de sintonia fina de cada uma das heurísticas.

Além de apresentar cenários que requeiram argumentos para prover justificativas da atividade de sintonia fina, também apresentamos como é possível realizar combinações de regras de heurísticas. Como existem diversas heurísticas já validadas na literatura, é interessante combinar tais heurísticas, buscando cada vez mais opções de melhorias na atividade de sintonia fina. Como no exemplo apresentado nesse capítulo, onde uma determinada heurística combinada com outra similar, proporciona uma nova prática, tornando-a mais completa para a atividade de sintonia fina. Como as ferramentas que implementam essas heurísticas estão acopladas a outras heurísticas e não possuem uma semântica por trás que possibilite uma alteração em alto nível ou a identificação rápida de alguma regra usada, fica muito complexo combinar as heurísticas sem o auxílio do *framework* proposto nessa tese.

Por fim, apresentam-se argumentos para justificar a implementação parcial do *framework*. Por problemas de restrições tecnológicas e uma análise sobre o esforço estimado para a implementação e definição de todas as regras necessárias para a execução plena do *framework*, a implementação completa é designada como trabalho futuro.