3
Ontologia de Sintonia Fina de Banco de Dados

Esse capítulo descreve a ontologia de aplicação proposta para representar a tarefa de executar heurísticas do domínio de sintonia fina de banco de dados. Não se pode representar totalmente o domínio na ontologia, pois conforme mostrado em (GUARINO, 1997), uma ontologia não será capaz de considerar em seus modelos e regras todas as possibilidades de um domínio específico. Como consequência, a ontologia fica dependente dos conceitos utilizados pela heurística em que a ontologia será aplicada até ser estendida para contemplar as demais regras e conceitos de novas heurísticas. O objetivo é que a ontologia proposta nessa tese seja estendida para cada necessidade de ferramenta automática ou cada DBA ao realizar a sintonia fina de banco de dados.

3.1.
Metodologia

Conforme mencionado no capítulo anterior, a metodologia de construção de ontologia usada nessa tese é a *ontology* 101 (Noy et al, 2001). Nessa subseção são descritos apenas os passos iniciais da metodologia. A partir do passo de detalhamento da ontologia, ou seja, da enumeração dos termos importantes, as etapas são descritas no anexo I da presente tese.

Para iniciar a construção da ontologia de aplicação proposta nesta tese, define-se o domínio e a tarefa específica em que a mesma será definida. O domínio usado por essa tese corresponde ao domínio de sintonia fina de banco de dados. A tarefa específica da ontologia compreende, inicialmente, as questões necessárias para justificar a tarefa de sintonia fina de índices. Como uma forma de limitar o espaço de busca por heurísticas que apoiem esse tipo de sintonia, decidiu-se usar as seguintes:

- Heurística de escolha de índices candidatos proposta por (Lohman, 2000), contemplando combinações de colunas referenciadas em cláusulas de consultas que fazem parte da carga de trabalho.

- Heurística de escolha de índices candidatos proposta por (Bruno, 2011), contemplando apenas índices de cobertura e suas combinações de colunas.
- Heurística de criação de índices, chamada de heurística de benefícios, usada em (Salles, 2004). (Salles, 2004) estudou a construção de arquiteturas de agentes de *software* que permitissem a completa automatização das atividades relacionadas à sintonia fina de índices em SGBDs relacionais.
- Heurística de benefícios estendida (criação e remoção de índices) e heurística de reconstrução automática de índices (Morelli, 2006). (Morelli, 2006) propõe índices de acordo com a carga de trabalho, que são criados, destruídos ou recriados de acordo com os comandos submetidos ao SGBD, considerando os malefícios causados pela fragmentação.

No capítulo 4, as heurísticas de (Lohman, 2000), (Bruno, 2011) e (Salles, 2004) são detalhadas um pouco mais.

A próxima etapa de construção da ontologia sinaliza que devemos considerar as ontologias existentes e reutilizá-las. O fato de reutilizar uma ontologia pode permitir a integração entre aplicações, visto que as mesmas utilizarão vocabulários controlados e similares. Durante as pesquisas realizadas, não foram encontradas ontologias do domínio de sintonia fina de banco de dados. A ontologia de SQL proposta por (Sosnovsky et al, 2008) é a mais próxima do domínio, mas não compreende conceitos importantes do escopo desta tese.

Conforme mencionado no início dessa subseção, as etapas seguintes são detalhadas no anexo I desta tese, incluindo: conceitos, hierarquia de classes, definições de propriedades e instâncias.

3.2. Ontologia de tarefa

A ontologia de tarefa é projetada para capturar a solução de um problema independente do domínio (Ikeda et al, 1998). A ontologia proposta nessa tese busca uma maneira genérica de executar heurísticas de sintonia fina de banco de

dados, independente da ontologia de domínio, ou seja, o seu metamodelo não precisará ser modificado, caso a ontologia do domínio seja alterado.

Na Figura 3-1, tem-se uma representação da ontologia proposta.

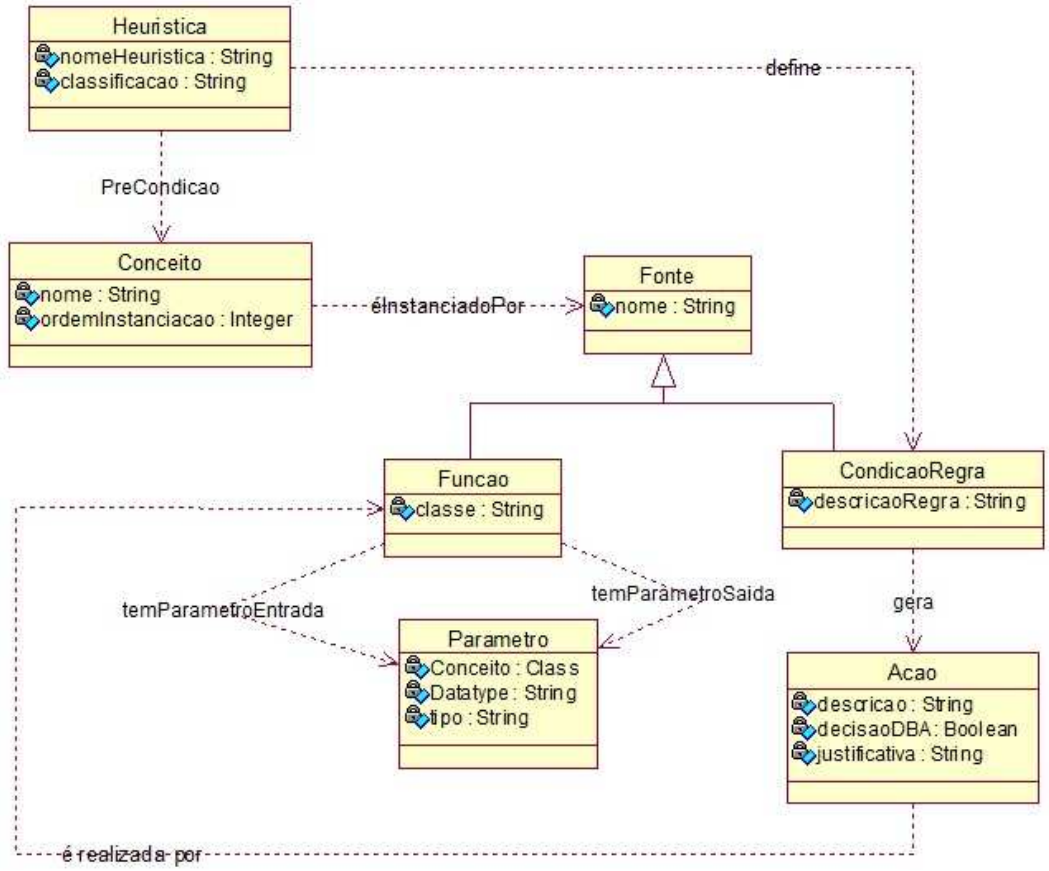

Figura 3-1 Ontologia de tarefa - Metamodelo do framework para execução de heurísticas de sintonia fina de banco de dados

Inicialmente, define-se o conceito de **heurística**, representada por seu nome, que engloba qualquer heurística proposta para realizar a sintonia fina de banco de dados. As heurísticas são classificadas de acordo com as possíveis ações de sintonia fina que podem ser tomadas. Dessa forma, consegue-se agrupar as heurísticas de acordo com os seus objetivos.

Cada heurística é aplicável quando determinados conceitos são instanciados. Dessa forma, a heurística possui como pré-condições, os **conceitos** definidos na ontologia de domínio de sintonia fina, descrita na seção 3.4. Esses conceitos são a base para que a heurística tome qualquer decisão em relação ao banco de dados. Cada conceito é representado por seu nome e ordem de instanciação. A ordem de instanciação é necessária porque existem conceitos dependentes de outros

conceitos. Por exemplo: não podemos executar regras responsáveis por induzir cláusulas (Figura 3-24) se não existir comando DML instanciado ainda.

Os conceitos podem ser instanciados ou obtidos através de duas **fontes**: **funções** ou **regras**. Por usar a ontologia como base para o *framework* e buscar explicitar qualquer raciocínio da heurística, opta-se pela preferência de usar regras para induzir os conceitos, ou seja, inferir novas instâncias de conceitos através de outros conceitos já instanciados.

As **regras** são definidas por seus nomes e descrições que correspondem à própria regra em si. As regras são formadas por pares de condiçãoRegra → ação, onde uma vez que a condição da regra seja satisfeita, uma ação é gerada. Nessa tese, as descrições das regras são definidas na linguagem SWRL (*Semantic Web Rule Language*), recomendada pela ferramenta Protégé (Protégé, 2013) usada para a definição da ontologia de aplicação. Essa linguagem inclui uma sintaxe abstrata, em alto nível, para especificar as regras combinadas em OWL DL (linguagem com completeza computacional, onde todas as conclusões são garantidas como sendo computáveis; e decidibilidade, onde todos os processamentos terminarão em um tempo finito) e OWL Lite (linguagem para classificação hierárquica e regras consideradas simples, tal como regra de cardinalidade) (Horrocks et al, 2004).

Em casos em que a maneira de escrever algo computacional seja complicada de descrever em regra, ou seja, sejam usadas muitas variáveis nas condições das regras, dificultando o seu entendimento, ou as representações de restrições sejam limitadas pela linguagem de regras usada, usam-se **funções**. Para que as funções possam ser invocadas e executadas, pensando em uma linguagem orientada a objetos, são necessárias as seguintes informações:

- Nome da função;
- Classe a qual a função pertence;
- Parâmetros de entrada;
- Parâmetros de saída.

Os **parâmetros**, sejam de entrada ou saída, precisam ser definidos também de acordo com a ontologia de domínio, ou seja, não pode ser algo desconhecido da sintonia fina de banco de dados. Para tal, o parâmetro deve possuir o *datatype*

ao qual está se referindo, a classe ao qual o *datatype* faz parte e o tipo desse *datatype*.

Após terem todos os conceitos, que fazem parte de sua pré-condição de execução, instanciados, a heurística já pode testar a sua condição para executar ações no banco de dados. Para isso, a heurística define **regras** que devem ser testadas e validadas pela máquina de inferência. Nesse caso, foi usada a máquina do próprio Protégé (JESS (Friedman-Hill, 2013)).

Uma vez que essas regras sejam verdadeiras, elas geram uma ou diversas **ações** de sintonia fina no banco de dados. Para cada ação, registramos:

- Descrição da ação;
- Decisão que o DBA tomou em relação à ação (aceita ou recusada);
- Justificativa, caso o DBA queira adicionar algum comentário útil à ação de sintonia fina e/ou a sua decisão.

Em princípio, as ações foram restringidas para serem realizadas por funções, pois exigem iterações com a base de dados. No entanto, caso seja necessário, a ontologia de tarefa pode ser generalizada e estendida para que a ação seja realizada por fonte, ou seja, contemplando também as regras.

Toda vez que uma ontologia de tarefa é instanciada significa que uma determinada heurística está sendo executada, utilizando instâncias de conceitos definidos na ontologia de domínio de sintonia fina de acordo com a carga de trabalho submetida ao banco de dados.

3.3. Instanciação da ontologia usada no *Framework*

Para ilustrar a instanciação da ontologia de aplicação (tarefa e domínio) no uso do *framework,* usamos a heurística de criação e remoção proposta por (Morelli et al, 2012) (Morelli, 2006).

Como essa heurística utiliza muitos conceitos do domínio, vamos ilustrar apenas com dois que possuam diferentes fontes de instanciação. São eles: **Tabela** e **Expressão de predicado**. Posteriormente, na seção seguinte, serão mostrados como esses conceitos são modelados na ontologia de domínio.

Para o conceito de **tabela**, tem-se a seguinte representação (Figura 3-2).

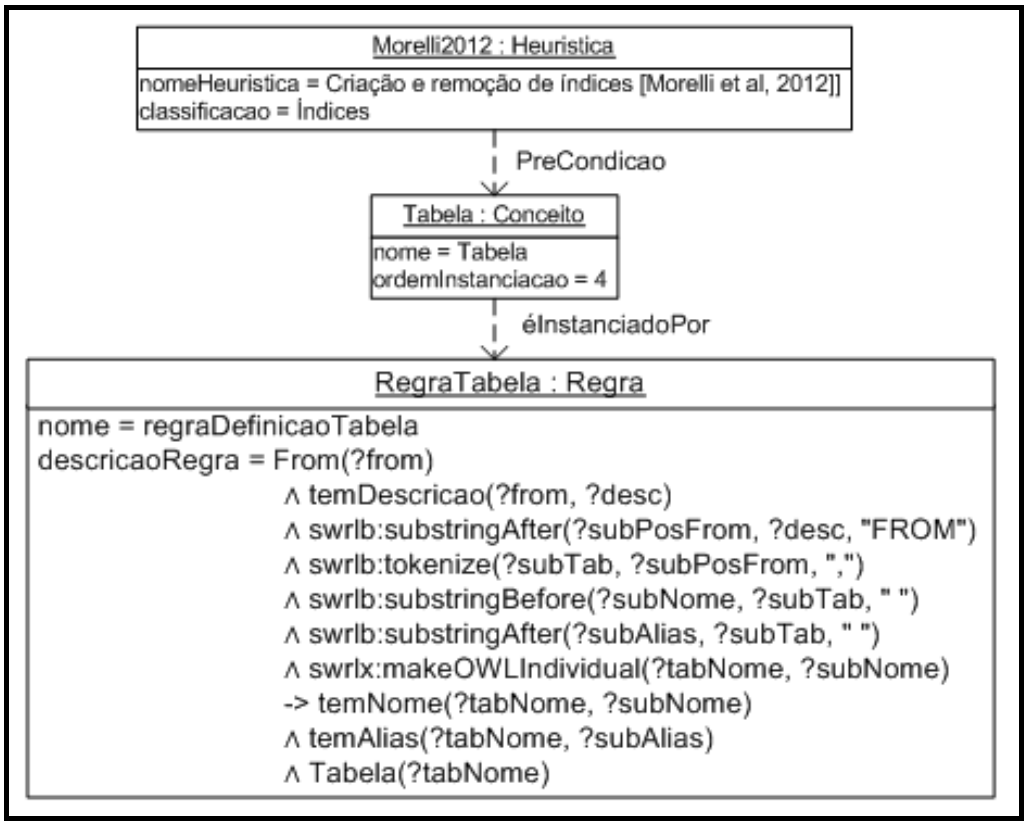

Figura 3-2 Instanciação da ontologia de tarefa usada no *framework* – Conceito obtido por regra

```
1.    From(?from)
2.    ∧ temDescricao(?from, ?desc)
3.    ∧ swrlb:substringAfter(?subPosFrom, ?desc, "FROM")
4.    ∧ swrlb:tokenize(?subTab, ?subPosFrom, ",")
5.    ∧ swrlb:substringBefore(?subNome, ?subTab, " ")
6.    ∧ swrlb:substringAfter(?subAlias, ?subTab, " ")
7.    ∧ swrlx:makeOWLIndividual(?tabNome, ?subNome)
8.    -> temNome(?tabNome, ?subNome)
9.    ∧ temAlias(?tabNome, ?subAlias)
10.   ∧ Tabela(?tabNome)
```

Figura 3-3 Regra SWRL para instanciar o conceito Tabela

Como uma forma de ilustrar o passo a passo seguido pela regra de inferência do conceito Tabela (Figura 3-3), criamos no Protégé o indivíduo apenas para simulação, dado que pela execução da regra, o mesmo apareceria na aba de indivíduos inferidos (*inferred*). A regra definida na Figura 3-3 trabalha da seguinte forma:

- ✓ **Linha 1:** Dado um indivíduo da classe `FROM` chamado *From_1*.

- ✓ **Linha 2:** Esse indivíduo possui a propriedade tem descrição (Figura 3-4).

```
1.    FROM lineitem l, orders o, order_line ol
```

Figura 3-4 Exemplo de instância da classe `FROM`, chamada de *From_1*

- ✓ **Linha 3:** A máquina responsável por processar a regra, extrai a *substring* posterior à palavra `FROM` (Figura 3-5).

```
1.    lineitem l, orders o, order_line ol
```

Figura 3-5 *Substring* posterior a palavra FROM

- ✓ **Linha 4:** A *substring* da Figura 3-5 é dividida em outras *substrings* que estejam separadas por vírgula (Figura 3-6).

```
1.    lineitem l
2.    orders o
3.    order_line ol
```

Figura 3-6 Resultado das *substrings* da Figura 3-5 dividida por vírgula

- ✓ **Linha 5:** Todas as *substrings* antes do espaço são recuperadas e atribuídas à variável `subNome` (Figura 3-7).

```
1.    subNome1: lineitem
2.    subNome2: orders
3.    subNome3: order_line
```

Figura 3-7 Valores atribuídos à variável subNome

- ✓ **Linha 6:** Todas as *substrings* após o espaço são recuperadas e atribuídas à variável `subAlias` (Figura 3-8).

```
1.    subAlias1: l
2.    subAlias2: o
3.    subAlias3: ol
```

Figura 3-8 Valores atribuídos à variável subAlias

- ✓ **Linha 7:** Um novo indivíduo para cada variável `subNome` é criado (Figura 3-9).

```
1.    Individuo1
2.    Individuo2
3.    Individuo3
```

Figura 3-9 Indivíduos criados com nomes aleatórios para cada variável subNome

- ✓ **Linha 8:** A propriedade `temNome` é atribuída aos novos indivíduos com o conteúdo da variável `subNome` (Figura 3-10 e Figura 3-11).

```
1.    Individuo1 - temNome: lineitem
2.    Individuo2 - temNome: orders
3.    Individuo3 - temNome: order_line
```

Figura 3-10 Atribuição da propriedade temNome para os novos indivíduos

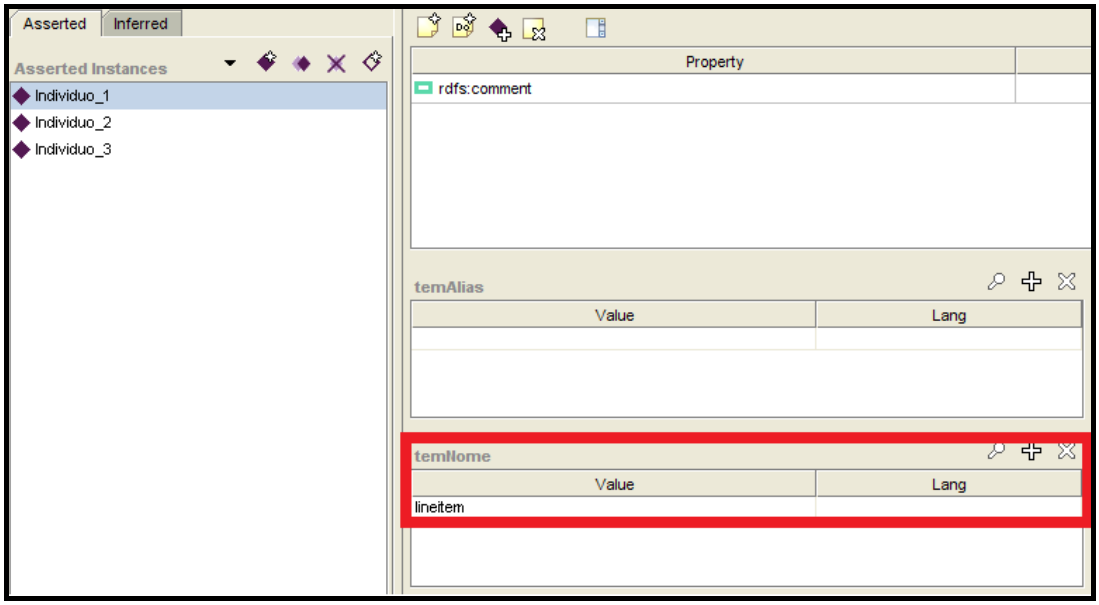

Figura 3-11 Exemplo de instância do nome da tabela

- ✓ **Linha 9:** A propriedade temAlias é atribuída aos novos indivíduos com o conteúdo da variável subAlias (Figura 3-12 e Figura 3-13).

```
1.    Individuo1 - temAlias: l
2.    Individuo2 - temAlias: o
3.    Individuo3 - temAlias: ol
```

Figura 3-12 Atribuição da propriedade temAlias para os novos indivíduos

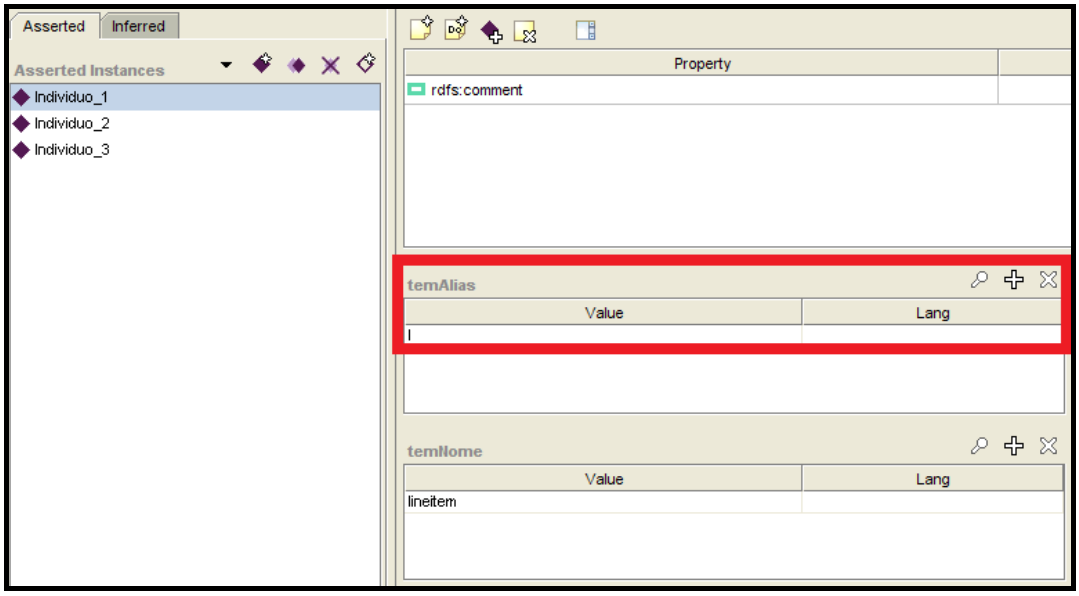

Figura 3-13 Exemplo de instância do alias da tabela

- ✓ **Linha 10:** Os novos indivíduos são classificados como sendo da classe Tabela (Figura 3-14).

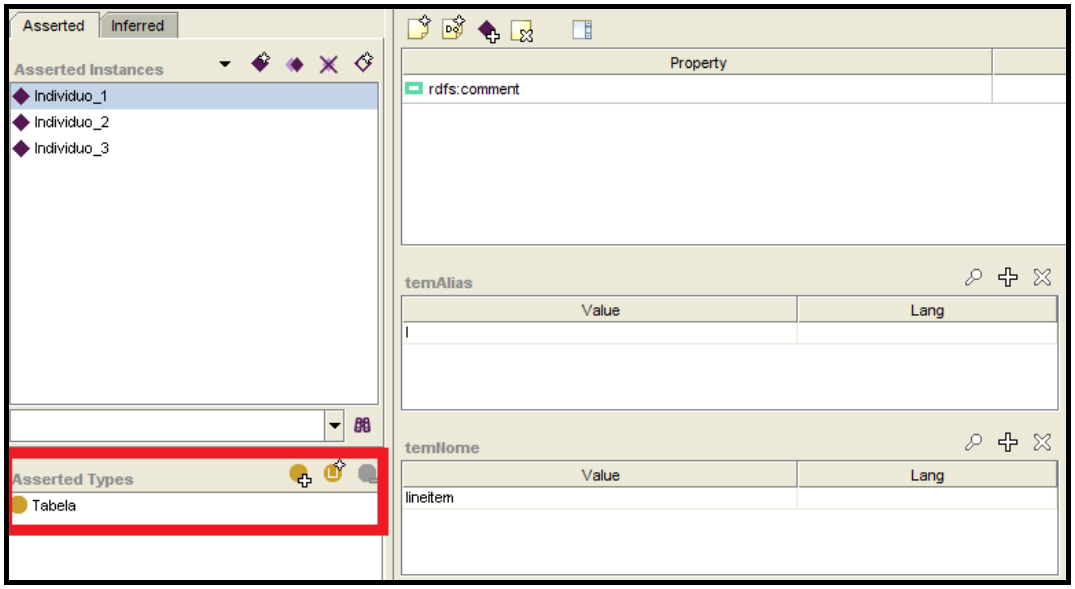

Figura 3-14 Exemplo de classificação da nova instância de tabela

Já para o conceito de **expressão de predicado**, que deve ser obtido através de função, apresentamos como seria a instância da ontologia de tarefa (Figura 3-15) e uma parte da função implementada em uma linguagem orientada a objeto (`java`). Esse é apenas um exemplo para demonstrar que a iteração entre a ontologia de tarefa e a implementação (código fonte da função) é viável.

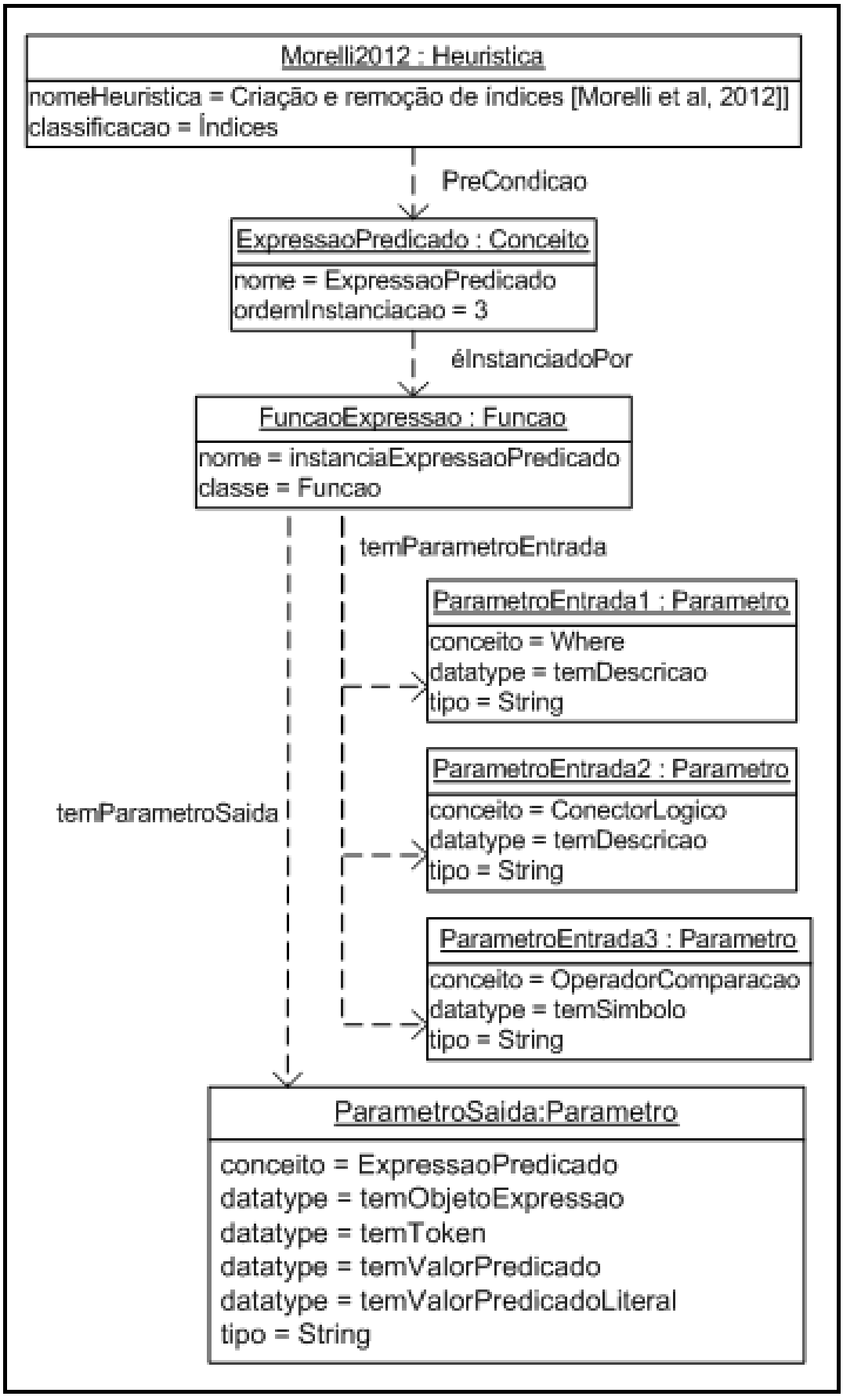

Figura 3-15 Instanciação da ontologia de tarefa usada no *framework* – Conceito obtido por função

Para obter o conceito de expressão de predicado, é necessário ter a cláusula `where` do comando DML, bem como os *tokens*, sejam eles conectores lógicos ou operadores de comparação, utilizados na cláusula. Com isso, a função chamada "`instanciaExpressaoPredicado`" consegue retornar os conceitos de expressão de predicado junto com suas propriedades. A função recebe um tipo de estrutura *map* para os parâmetros, recupera cada um dos parâmetros de entrada e utiliza expressões regulares para retornar o conceito solicitado.

A seguir, mostramos como a função recupera os parâmetros de entrada (Figura 3-16) e a expressão regular (Figura 3-17) usada para obter as expressões de predicado que usam conectivos lógicos.

```
// Primeiro parâmetro de entrada: Cláusula Where sem "WHERE"
inicial
String clausulaWhere[]
      = (String[]) mapParametrosEntrada.get("Where");
clausulaWhere
      = clausulaWhere[0].split("WHERE ");

// Segundo parâmetro de entrada: Lista de conectivos
String listaConectivos[]
      = (String[]) mapParametrosEntrada.get("Conectivo");

// Terceiro parâmetro de entrada: Lista de operadores
String listaOperadores[]
      = (String[]) mapParametrosEntrada.get("Operador");
```

Figura 3-16 Exemplo de recuperação de parâmetros de entrada de funções

```
//Cria string para ser colocada na expressao regex de acordo com o
número de conectivos
for(int contConectivo = 0;
      contConectivo < listaConectivos.length;
      contConectivo++)
{
      conectivos
            = conectivos.concat(listaConectivos[contConectivo] + "|");
}
// Divide a string, obtendo as expressões de predicado
resultado
      = clausulaWhere[1].split("(?<=[\\s+"+conectivos+"\\s+])\\s");
```

Figura 3-17 Exemplo de função usada para obter conceito da ontologia de domínio

Após todos os conceitos necessários à heurística estarem instanciados, a heurística passa a testar as suas regras e no caso dessas regras serem válidas, elas vão gerar ações responsáveis por alterar a estrutura do banco de dados em busca da solução de sintonia fina. No caso da heurística analisada, serão ações de criar ou remover índices reais e hipotéticos (Figura 3-18).

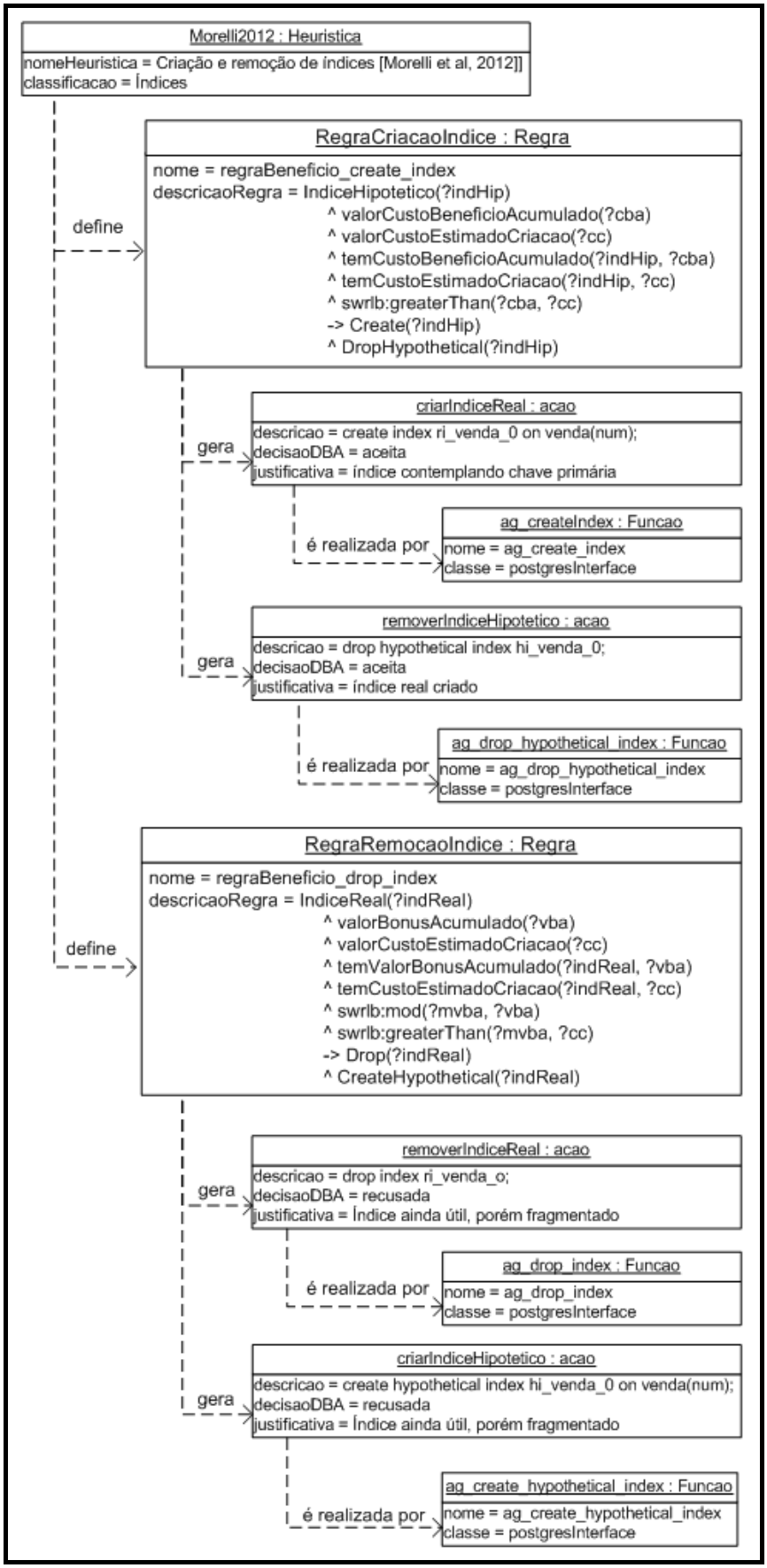

Figura 3-18 Instanciação da ontologia de tarefa usada no *framework* – Regras e ações definidas por heurística

3.4. Ontologia de domínio

Como mencionado anteriormente, a ontologia de tarefa precisa de instâncias de conceitos do domínio de sintonia fina. Para tal, precisa-se definir uma ontologia de domínio. A ontologia proposta nessa tese descreve alguns dos principais conceitos, propriedades e regras usadas para o domínio de sintonia fina de banco de dados. No anexo II, tem-se o modelo completo da ontologia de domínio. Para levantar os conceitos e as propriedades a serem usados na ontologia, tomam-se como base as heurísticas já mencionadas na subseção3.1. Para o desenvolvimento da ontologia, utilizou-se a ferramenta Protégé 3.5 (Protégé, 2013).

Sistema de Gerência de Banco de Dados (SGBD)

Iniciamos pelo conceito do próprio **Sistema de Gerência de Banco de Dados (SGBD)**. Como cada SGBD possui a sua própria maneira de implementar o otimizador e este faz uso de um determinado modelo de custos, foram definidas propriedades usadas para realizar cálculos de estimativas de valores. Estas estimativas são usadas para simular um valor próximo ao valor calculado pelo otimizador, de forma a simular o ambiente avaliado pelo otimizador e realizar a sintonia fina. As estimativas são obtidas através de fórmulas propostas pelas próprias heurísticas. Tais fórmulas podem ser definidas através de regras ou funções na ontologia de tarefa usada pelo *framework*. Embora sejam usadas fórmulas de heurísticas, tais fórmulas dependem de valores de parâmetros que variam de acordo com cada SGBD. Por esse motivo é que se opta por atribuir as propriedades ao conceito de SGBD. As propriedades definidas pelo SGBD são (Figura 3-19):

- Custo de entrada e saída (E/S). Cada SGBD possui um custo estimado de E/S;
- Custo de unidade central de processamento (CPU - *Central Processing Unit*) para processar uma tupla que esteja em memória;
- Coeficiente que relaciona, percentualmente, o custo de uma operação de E/S com o custo de CPU;

- Espaço em disco disponível para as atividades de sintonia fina (em *kilobytes*), tal como a criação de estruturas de acesso.
- Nome do SGBD.

Cada SGBD possui um valor padrão para as propriedades descritas acima (E/S, CPU e coeficiente). Por exemplo, segundo levantamento realizado em (Salles, 2004), para o SGBD PostgreSQL, cada E/S tem custo estimado igual a um e o coeficiente tem um valor padrão de 1%.

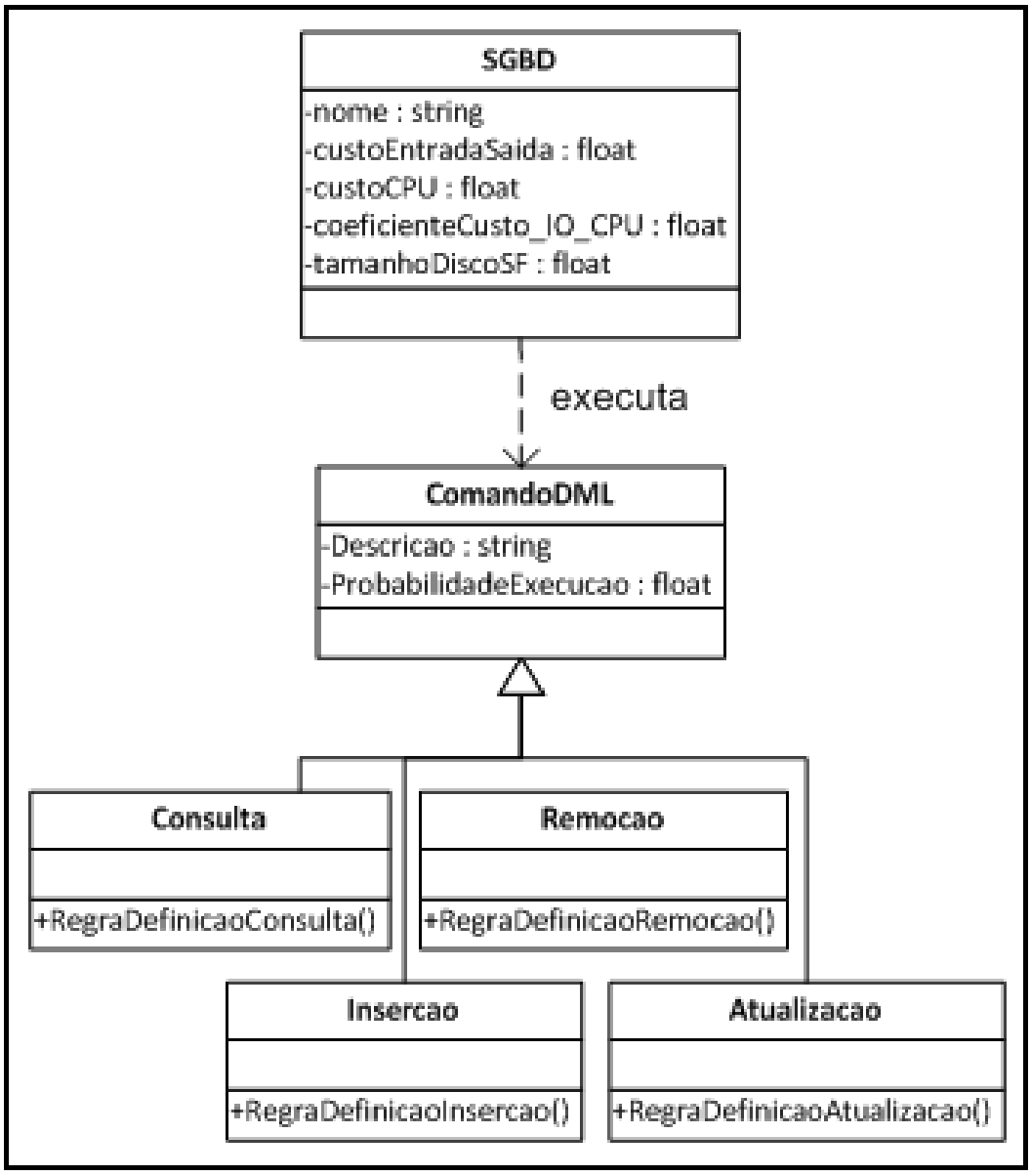

Figura 3-19 Ontologia de domínio – SGBD e Comando DML

Comandos DML

O SGBD recebe uma série de **comandos DML** (*Data Manipulation Language*) para serem executados. Com base nessa carga de trabalho, composta por esse tipo de comando, que a atividade de sintonia fina é realizada. O comando DML (Figura 3-19) possui:

- Descrição, que equivale exatamente ao texto do comando que é submetido à base de dados.
- Probabilidade de execução, que corresponde, em termos de porcentagem, a probabilidade de um comando DML ser executado na carga de trabalho. Esse valor pode ser usado para atribuir, por exemplo, pesos diferentes aos comandos, definindo o seu grau de importância para a carga de trabalho. Essa propriedade pode ter um valor padrão e quando o DBA possuir um conhecimento prévio da carga de trabalho, ele pode alterá-lo. Com isso, uma ferramenta que use essa informação, pode agregar o conhecimento do DBA a sua heurística. Outra alternativa seria usar ferramentas disponíveis na literatura que possam prever a frequência de consultas presentes na carga de trabalho.

Além disso, cada comando DML pode ser classificado em:

- **Consulta (*select*);**
- **Atualização (*update*)**;
- **Inserção (*insert*);**
- **Remoção (*delete*)**.

Essas classificações são consideradas disjuntas. Dessa forma, define-se na ontologia para cada uma delas, essa restrição. Na Figura 3-20, exemplifica-se a classificação de consulta com suas disjunções associadas, ou seja, quando um comando DML é classificado como sendo do tipo **consulta**, ele não pode ser classificado ao mesmo tempo como atualização, remoção ou inserção.

```
1.      <owl:Class rdf:ID="Consulta">
2.          <owl:disjointWith>
3.              <owl:Class rdf:ID="Atualizacao"/>
4.          </owl:disjointWith>
5.          <owl:disjointWith>
6.              <owl:Class rdf:ID="Remocao"/>
7.          </owl:disjointWith>
8.          <owl:disjointWith>
9.              <owl:Class rdf:ID="Insercao"/>
10.         </owl:disjointWith>
11.     </owl:Class>
```

Figura 3-20 Classificação sobre disjunções da classe "Consulta"

No contexto de SGBD relacional, os comandos DML são descritos usando a linguagem SQL (*Structured Query Language*). Por ser uma linguagem padronizada, consegue-se induzir, através de regras sobre a sintaxe do comando, o tipo do comando DML recebido pelo banco de dados de maneira automática. Como exemplo, na Figura 3-22, tem-se a regra definida para classificar uma instância como sendo do conceito **consulta**.

Imaginando o seguinte comando DML submetido ao banco de dados (Figura 3-21**Erro! Fonte de referência não encontrada.**):

```
1.      SELECT d.nome, count(*)
2.      FROM empregado e, departamento d
3.      WHERE
4.        e.matricula = d.matricula
5.        and e.salario > 10000
6.      GROUP BY d.nome
```

Figura 3-21 Exemplo de instância de comando DML

A regra (Figura 3-22) diz que:

- Linha 1 - Figura 3-22: se existe uma instância de comando DML e;
- Linha 2 - Figura 3-22: essa instância tem uma descrição (Figura 3-21);
- Linha 3 - Figura 3-22: essa descrição inicia pela palavra chave SELECT (linha 1 da Figura 3-21);

- Linha 4 - Figura 3-22: então, esse comando DML pode ser classificado com sendo do tipo **consulta**.

```
1.    ComandoDML(?cDML)
2.    ^ temDescricao(?cDML, ?d)
3.    ^ swrlb:startsWith(?d, "SELECT")
4.    → Consulta(?cDML)
```

Figura 3-22 Exemplo de regra de inferência para comandos de consulta

Cláusula

Considerando as boas práticas de sintonia fina, cada cláusula do comando DML, que faz parte da carga de trabalho, deve ser analisada. As cláusulas são analisadas, individualmente, para levantar as colunas usadas em cada uma delas e acrescentar ou não alguma importância. Portanto, descreve-se na ontologia o conceito de **cláusula** e seus principais tipos (Figura 3-23):

- **Select**;
- **From**;
- **Where;**
- **Group by;**
- **Having;**
- **Order by;**
- **Delete;**
- **Update;**
- **Set;**
- **Insert**.

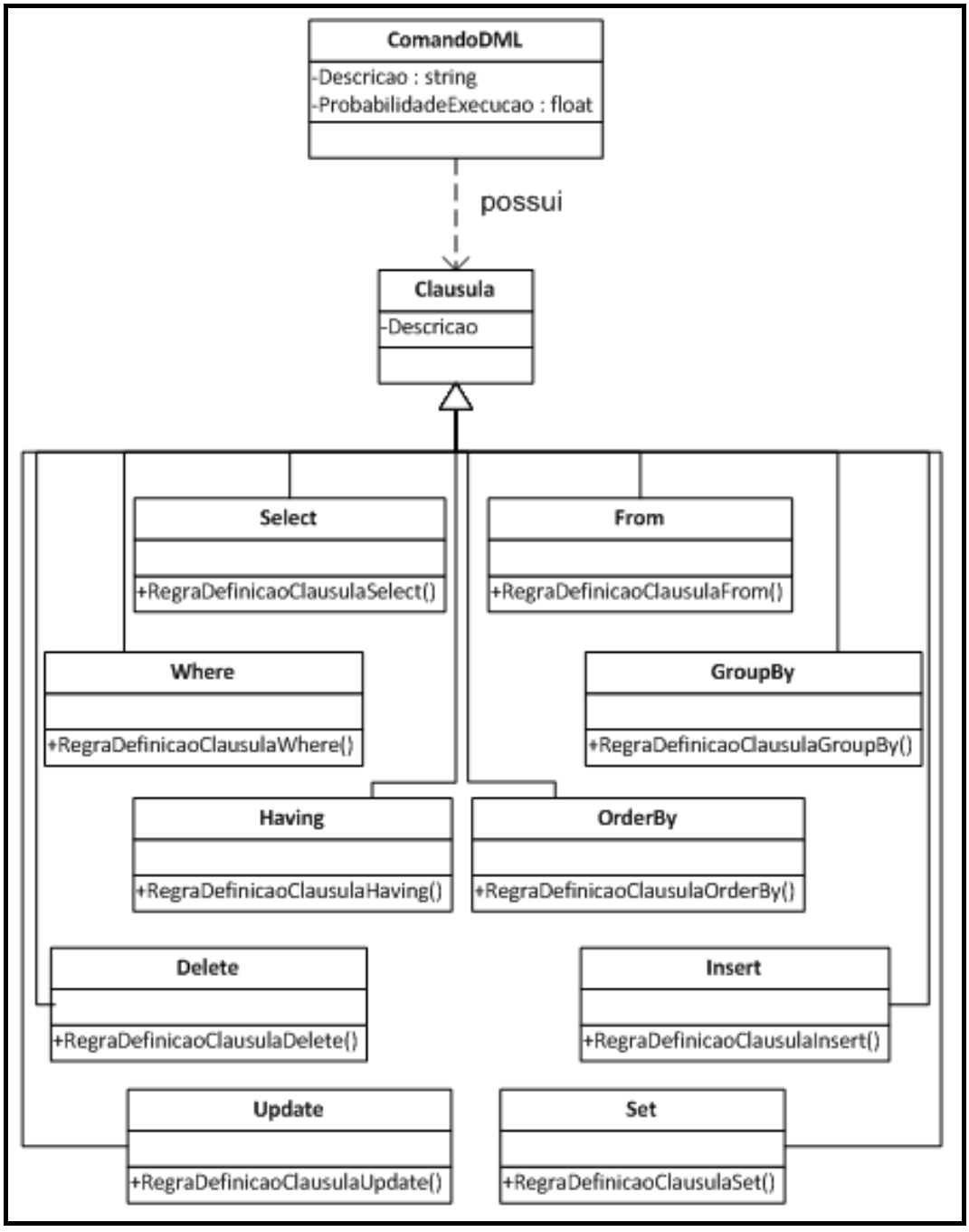

Figura 3-23 Ontologia de domínio – Comando DML e Cláusula

Assim como na classificação de comando DML, também se consegue criar regras de inferência para separar as cláusulas do comando, de acordo com a sintaxe da linguagem SQL. Por exemplo, para a cláusula **`WHERE`**, tem-se a regra descrita em SWRL na Figura 3-24.

Nessa regra (Figura 3-24), define-se que:

- Linha 1 - Figura 3-24: se existe uma instância de comando DML;

- Linha 2 - Figura 3-24: essa instância do comando DML possui uma descrição (Figura 3-21);
- Linha 3 e 4 – Figura 3-24: recupere a *substring* anterior a cláusula `GROUP BY` e posterior a cláusula `WHERE` (linhas 4 e 5 - Figura 3-21);
- Linha 5 - Figura 3-24: concatene essa *substring* com a palavra `WHERE` no início dela (linhas 3, 4 e 5 - Figura 3-21);
- Linha 6 - Figura 3-24: por fim, crie um novo indivíduo na ontologia com o conteúdo da *substring* obtida;
- Linha 7 - Figura 3-24: classifique o novo indivíduo como sendo do tipo de cláusula `WHERE`;
- Linha 8 - Figura 3-24: atribua a propriedade de descrição ao novo indivíduo com o conteúdo da *substring* obtida (linhas 3, 4 e 5 - Figura 3-21)

```
1.    ComandoDML(?com)
2.    ∧ temDescricao(?com, ?desc)
3.    ∧ swrlb:substringBefore(?subAntGroup, ?desc, "GROUP BY")
4.    ∧ swrlb:substringAfter(?subPosWhere, ?subAntGroup, "WHERE")
5.    ∧ swrlb:stringConcat(?claus, "WHERE", ?subPosWhere)
6.    ∧ swrlx:makeOWLThing(?where, ?claus)
7.    -> Where(?where)
8.    ∧ temDescricao(?where, ?claus)
```

Figura 3-24 Regra para extrair a descrição da cláusula WHERE a partir da descrição do comando DML

Expressão de predicado e *Token*

Uma vez tendo separado as cláusulas do comando DML, é importante analisar em quais **operadores de comparação** cada coluna está sendo usada. Além disso, para conseguir os conceitos detalhados e possibilitar maiores justificativas em relação aos operadores, cria-se o conceito de **expressão de predicado**. Na ontologia proposta nessa tese, define-se que apenas a cláusula `WHERE` tem expressão lógica do tipo expressão de predicado (Figura 3-25). Posteriormente, a ontologia pode ser estendida para que outras cláusulas (ex.:

`FROM`, `HAVING`) possam lidar com expressões de predicado, incluindo novos conceitos e definições na ontologia de domínio.

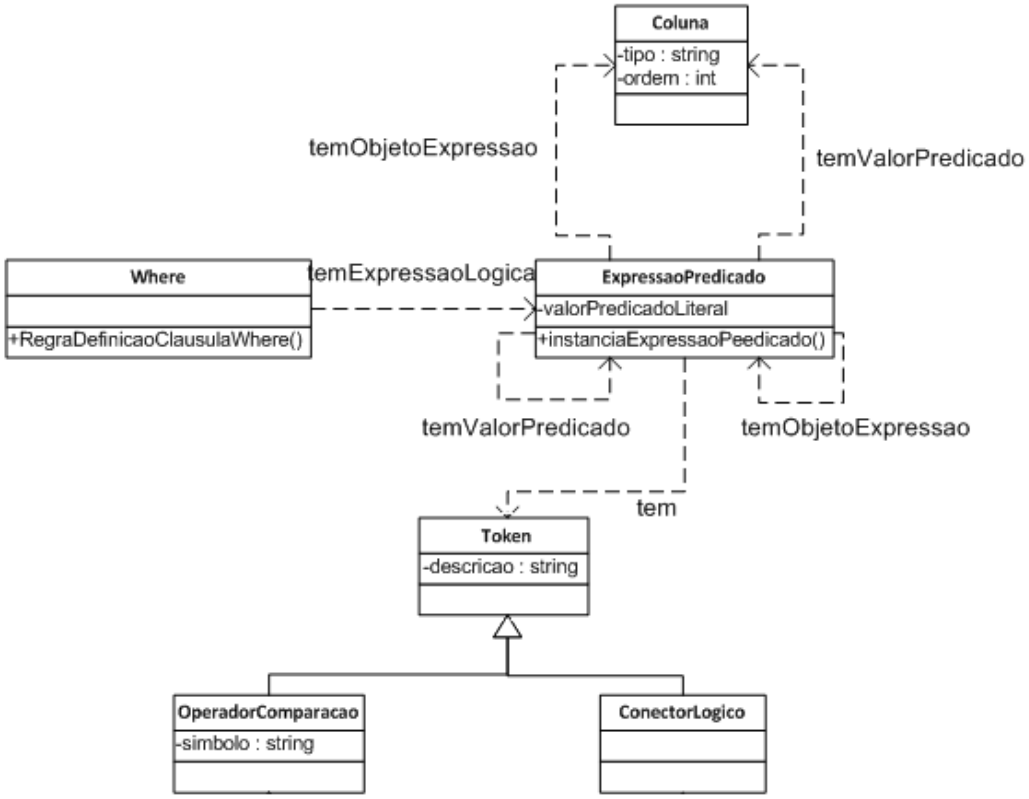

Figura 3-25 Ontologia de domínio – Cláusula Where, Expressao de Predicado, *Token* e Coluna

Cada expressão de predicado pode ser formada da seguinte maneira:

- objeto de expressão + *token* + valor de predicado

O objeto de expressão pode representar um dos seguintes conceitos (Figura 3-25):

- **coluna** ou;
- a própria **expressão de predicado**.

O *token* é usado nessa tese como uma forma de generalizar operadores e conectores. Ele pode ter suas instâncias já previstas na ontologia (vide anexo I).

Cada conector lógico e operador de comparação possui uma descrição, que corresponde ao texto que os representam no comando DML (ex.: between, and). Além disso, o operador de comparação também possui um símbolo (ex.: +, =),

que corresponde ao símbolo usado em comparações no comando DML (Figura 3-25).

O valor de predicado pode ser do tipo (Figura 3-25):

- **Coluna**;
- Literal (ex.: *string*, *integer*, *float*). Ele é expresso na ontologia como **valorPredicadoLiteral**) ou;
- **Expressão de predicado**.

Para ilustrar o uso dos conceitos envolvidos na expressão de predicado, instancia-se as classes envolvidas. Por exemplo, o comando DML apresentado na Figura 3-21**Erro! Fonte de referência não encontrada.**, onde existe uma instância da classe de cláusula **`where`** (linhas 3, 4 e 5).

Com o exemplo de instância da classe `where`, consegue-se obter as instâncias da classe de **expressão de predicado**. Na Figura 3-26 tem-se um exemplo das três instâncias de expressões de predicado que podem ser obtidas da instância do tipo `where` nas linhas 3, 4 e 5 da Figura 3-21.

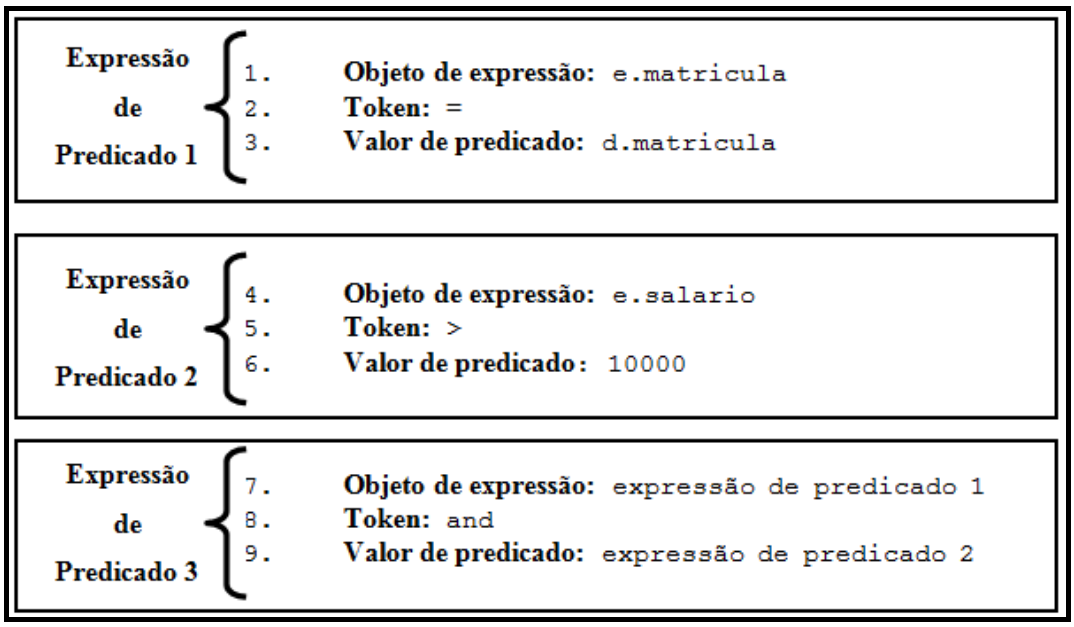

Figura 3-26 Exemplo de instâncias de expressões de predicado

Por ser um conceito complexo de ser obtido por regras, pois envolve o tratamento de *strings* que podem ser realizados de forma mais simples através de funções já existentes em bibliotecas de linguagens de programação, foi desenvolvida uma função de forma genérica que retorna as instâncias de cada um dos conceitos envolvidos na expressão de predicado.

Objeto do SGBD

As cláusulas do comando DML referenciam **objetos** existentes no SGBD. Esses objetos podem ser (Figura 3-27):

- **Tabelas**
- **Colunas**

Cada objeto do SGBD possui como propriedade o seu nome na base de dados (Figura 3-27). Para **coluna** também é definida a propriedade de tipo, que pode ser usada para justificar o tamanho estimado de uma tupla composta por n colunas, a propriedade ordem, que define a ordem da posição das colunas na tabela na base de dados e a propriedade restricaoDominio, que são valores restritos e pré-definidos para serem usados nas instâncias da coluna (Figura 3-27).

Uma determinada **coluna** pertence a uma **tabela** no SGBD. As informações de **tabela** que geralmente são usadas nas atividades de sintonia fina são (Figura 3-27): número de tuplas, número de páginas que ela ocupa em disco, chave primária e chave estrangeira presente na tabela.

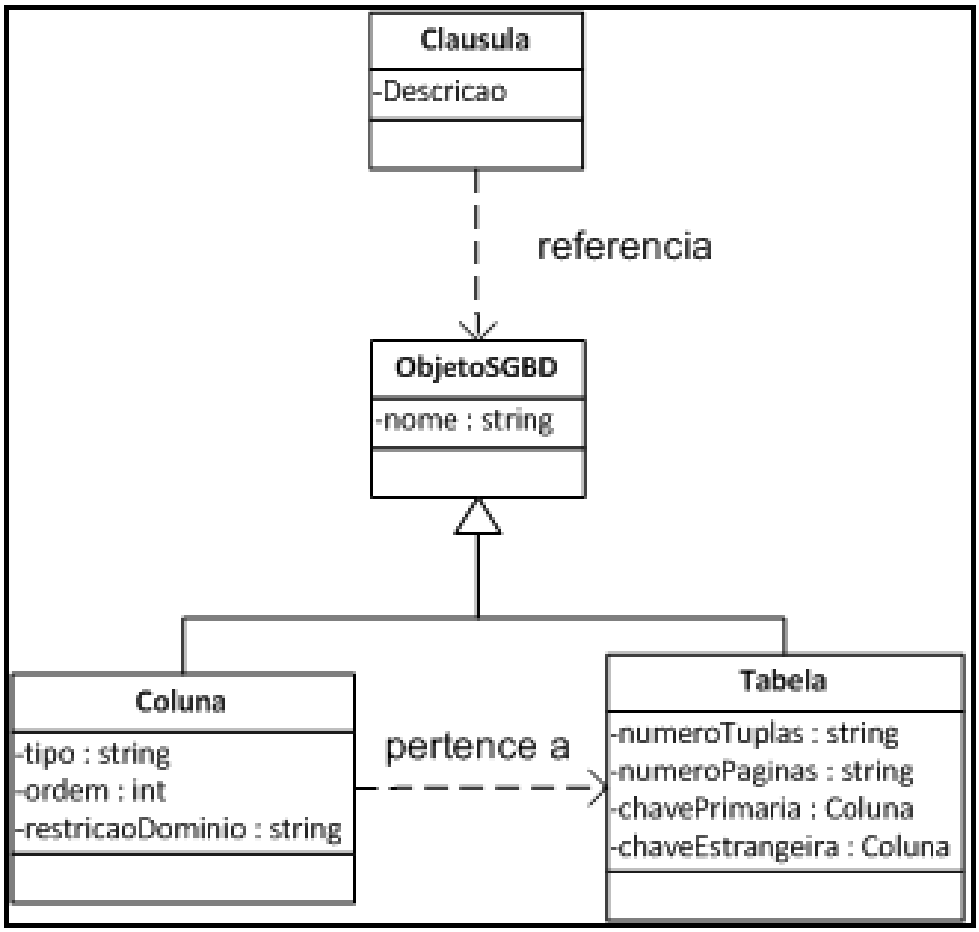

Figura 3-27 Ontologia de domínio – Cláusula e ObjetoSGBD

Plano de execução e Tipo de varredura

Ao receber um comando DML, o otimizador do SGBD deve gerar o seu **plano de execução** que possui a sequência de operações necessárias para a execução do comando. No escopo dessa tese, existem dois tipos de planos de execução disjuntos (Figura 3-28):

- Plano de execução hipotético;
- Plano de execução real.

O **plano de execução hipotético** (Monteiro, 2008) possui um custo estimado para execução do comando DML, pois ele simula valores com base em fórmulas e usa também, além de estruturas de acesso existentes na base, aquelas que existem somente na metabase do banco de dados (estruturas hipotéticas), ou seja, não existem fisicamente.

O **plano de execução real** é de fato aquele gerado pelo otimizador com o custo real que foi calculado para a execução do comando, considerando apenas estruturas de acesso que realmente existam fisicamente.

Ambos os planos possuem o número de tuplas estimado para atualização, que corresponde ao número estimado de tuplas a serem atualizadas de acordo com a execução de um determinado comando DML de inserção, remoção ou atualização e um valor de custo de benefício usado para verificar o quanto o custo diminuiria ou não ao usar determinada estrutura de acesso ainda não existente fisicamente. Esse valor de custo de benefício é obtido através da subtração do valor de custo total do plano de execução real e o valor de custo total do plano de execução hipotético (simulando o uso de alguma estrutura de acesso inexistente no banco de dados).

Conforme pode ser visto no capítulo 2, seção 2.1, o plano de execução gerado pelo otimizador executa diversas operações. Entre elas, está a operação de **varredura** sobre a tabela que o comando DML referencia. A varredura pode ser do tipo sequencial (**FullTableScan**) ou usando a estrutura de índice (**FullIndexScan**). Como as heurísticas selecionadas nessa tese, que originaram os conceitos da ontologia de aplicação, não precisam dos demais tipos de operações que podem ser usados por um plano de execução, estes não foram definidos. No entanto, a ontologia pode ser estendida para contemplar as demais operações, tais como: *nested-loop join, hash join, merge join* e *index seek* entre outras, em caso

de necessidade. Tal extensão pode ser realizada acrescentando os novos conceitos e suas definições à ontologia de domínio.

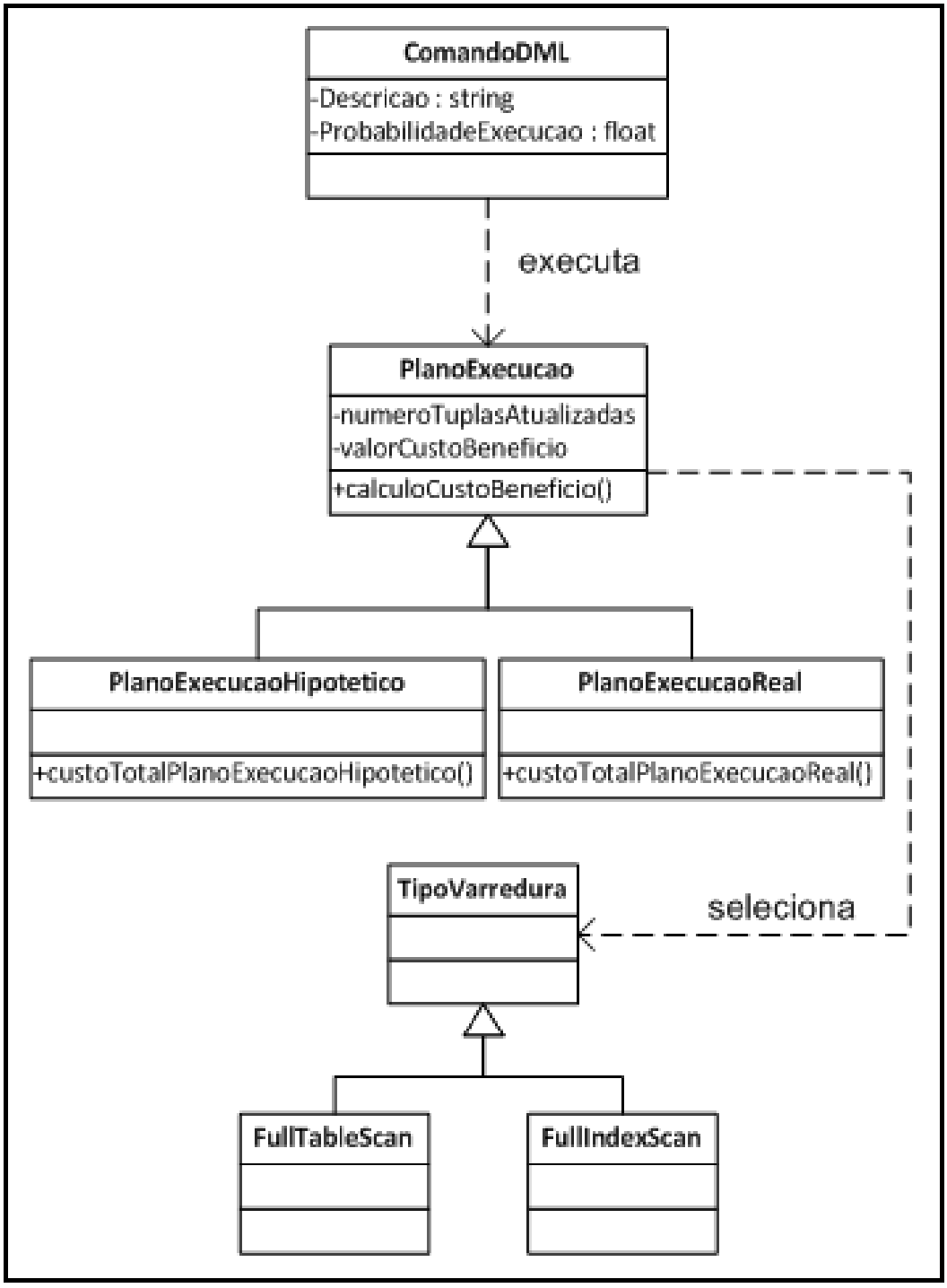

Figura 3-28 Ontologia de domínio – Plano de execução e Tipo de varredura

Estrutura de acesso e Estrutura de dados

Ao optar por selecionar um plano de execução que utilize uma **estrutura de acesso**, o otimizador pode agilizar a busca por informações, dependendo da necessidade do comando DML.

Nesta tese, focamos na estrutura de **índice**. No entanto, citamos na ontologia não só o índice, mas também outras estruturas, muito consideradas por ferramentas de sintonia fina, as **visões materializadas** e as **partições**, sem maiores detalhes (Figura 3-31).

Para a estrutura de **índice**, detalhamos as propriedades usadas pelas atividades de sintonia fina. São elas (Figura 3-31):

- Nome do índice;
- Valor estimado de custo de atualização de um índice. Tal valor é calculado por fórmulas propostas por heurísticas. Por exemplo, em (Salles, 2004), a fórmula sugerida é apresentada em **Figura 3-29**, onde:
  - C_A é o custo de atualização de um índice;
  - r é o número de tuplas atualizadas pelo comando;
  - R é o número de tuplas da tabela;
  - P é o número de páginas da tabela;
  - c é o coeficiente que relaciona, percentualmente, o custo de uma operação de E/S com o custo de CPU para processar uma tupla que esteja em memória.

$$C_A = 2 \left\lceil \frac{r}{R} \right\rceil P + cr$$

Figura 3-29 Exemplo de fórmula para calcular custo de atualização de um índice

- Valor estimado de custo de criação do índice. Tal valor também é calculado por fórmulas propostas por heurísticas. Por exemplo, em

(Salles, 2004), a fórmula sugerida é apresentada em **Figura 3-30**, onde:

- CC_I é o custo de criação de um índice;
- P é o número de páginas da tabela;
- c é o coeficiente que relaciona, percentualmente, o custo de uma operação de E/S com o custo de CPU para processar uma tupla que esteja em memória;
- R é o número de tuplas da tabela.

$$CC_I = 2P + cR \log R$$

Figura 3-30 Exemplo de fórmula para calcular custo de criação de um índice

- Número de blocos em disco que são ocupados pelo índice. Tal informação pode ser obtida dos metadados ou pode ser estimada através do tamanho ocupado pelos atributos indexados;
- Data/hora de atualização das propriedades do índice, para indicar se um determinado custo está crescendo ou decrescendo no decorrer da execução ou análise da carga de trabalho;
- Ordem que as colunas devem ser indexadas. Essa ordem pode ser usada para eliminar, por exemplo, o uso de um operador de ordenação, em casos de comandos do tipo consulta que ordene o resultado pela mesma ordem em que as colunas foram indexadas.

Todos os valores que fazem parte das propriedades do índice são calculados e/ou instanciados por funções de custo referenciadas na ontologia.

Os índices são implementados no banco de dados através de **estruturas de dados**, conforme descrito no capítulo 2. Citamos na ontologia, as estruturas mais utilizadas por SGBDs comerciais e ferramentas de sintonia fina, que são (Figura 3-31):

- **Árvore B+**;
- **Tabela *hash***;
- ***Bitmap***.

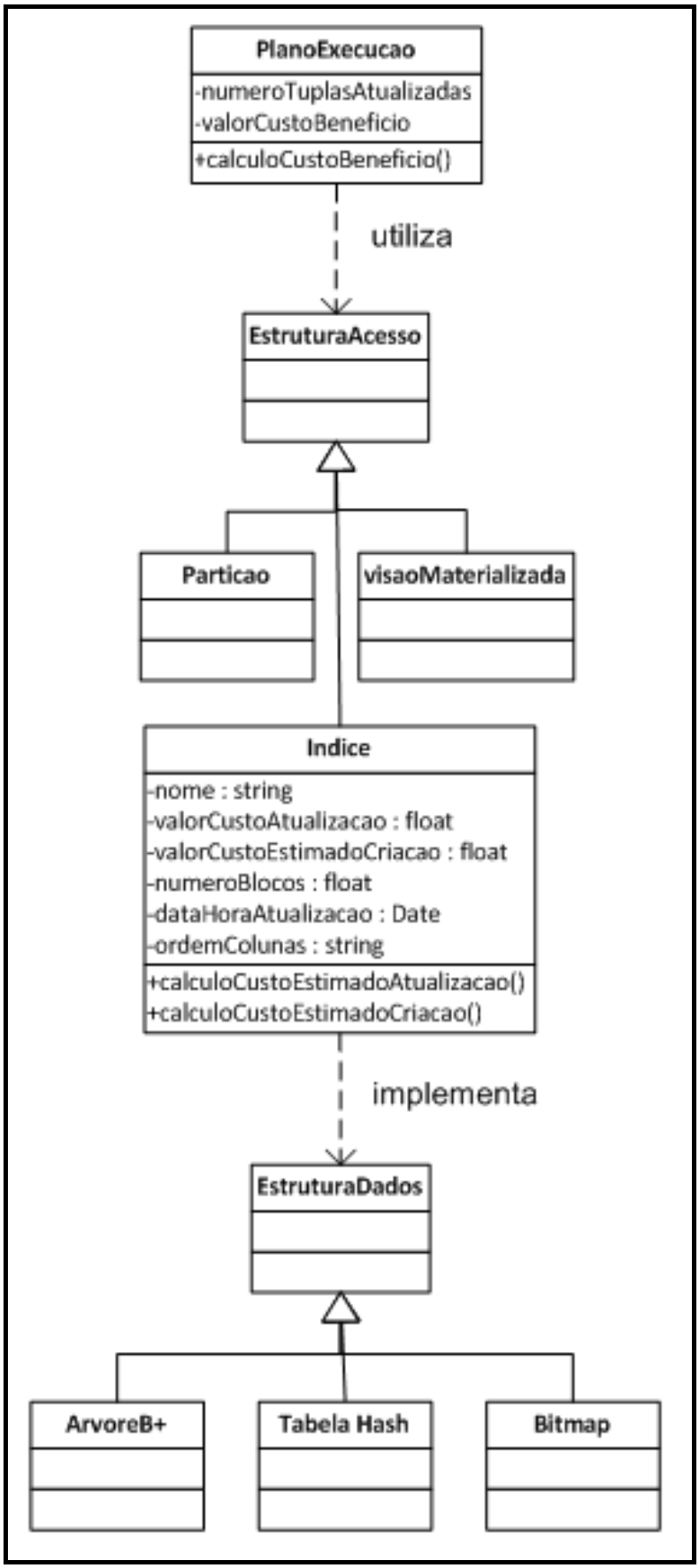

Figura 3-31 Ontologia de domínio – Estrutura de acesso e Estrutura de dados

Além disso, a estrutura de índice pode ser subdividida em **índice hipotético** e **índice real** (Figura 3-32). Esses conceitos foram definidos no capítulo 2, seção 2.1.2.1.

Utilizando regras sobre os objetos de expressão é possível definir os **índices hipotéticos** que podem ser úteis ou impactados por um determinado comando DML. Essas regras devem ser definidas em nível de ontologia de tarefa, pois faz parte das ações que devem ser tomadas por heurísticas de escolha de índices candidatos. A ontologia de tarefa é descrita na próxima seção.

Define-se que um comando DML origina um determinado índice hipotético quando os possíveis índices gerados a partir dos objetos de expressão analisados para o comando são benéficos, ou seja, são candidatos a diminuir o custo de execução do comando.

Já a propriedade em que um comando DML impacta um índice hipotético indica que o índice está sendo afetado negativamente pelo comando. Esse último caso ocorre muito em comandos que exijam atualizações no banco de dados (atualização, remoção e/ou inserção).

As informações registradas de **índice hipotético** são (Figura 3-32):

- Valor do custo de benefício acumulado para o índice durante toda a execução ou análise da carga de trabalho submetida ao banco. Tal valor é obtido através da soma ou acúmulo dos valores de benefícios originados de cada comando DML em que o índice é útil. O valor do benefício é derivado da subtração do custo do plano de execução do comando DML sem o índice e do custo do plano de execução do comando DML com a simulação da existência do índice.
- Número de vezes em que o índice foi utilizado por plano de execução gerado para um determinado comando DML.

A partir de um índice hipotético, pode ser gerado um **índice real**. O índice real pode ser criado no momento em que um índice hipotético for considerado realmente útil, com uma boa relação custo x benefício de acordo com determinado parâmetro.

Para realizar o acompanhamento desse tipo de índice, enquanto real, já que não se pode garantir que um índice criado será útil por toda a vida do banco de dados, são registradas algumas informações (Figura 3-32):

- Valor de bônus do índice, similar ao valor de custo benefício do índice hipotético. Esse valor pode ser atribuído positivamente ou negativamente ao índice, de acordo com a sua utilidade para um determinado comando DML;
- Valor de bônus acumulado do índice. Toda vez que o índice for considerado benéfico para um determinado comando, esse valor é incrementado positivamente com o valor de bônus. Caso contrário, se o índice for impactado por comandos de atualização, inserção ou remoção, esse valor é decrementado com o valor de bônus;
- Valor de razão inicial, onde se calcula a divisão entre o número de tuplas da tabela pelo número de blocos do índice;
- Valor do grau de fragmentação do índice;
- Número de varreduras em que o índice foi útil, ou seja, foi usado pelo plano de execução real;
- Valor do fator de preenchimento da página do índice.

Todos os valores citados como propriedades dos índices são obtidos através de estatísticas da base de dados ou de fórmulas descritas em funções para estimá-los. Inicialmente, apenas o valor de bônus do índice é dependente do comando DML, pois ele é derivado do custo de benefício, enquanto o índice existia como hipotético. Após a criação do índice como real, as propriedades citadas do índice não dependem mais do comando DML, pois nesse caso, já existe um valor padrão atribuído que passa a ser acumulado a cada utilização do índice.

As estruturas de acesso, sejam elas índices, visões materializadas ou partições, atuam sobre **colunas** existentes em tabelas na base de dados.

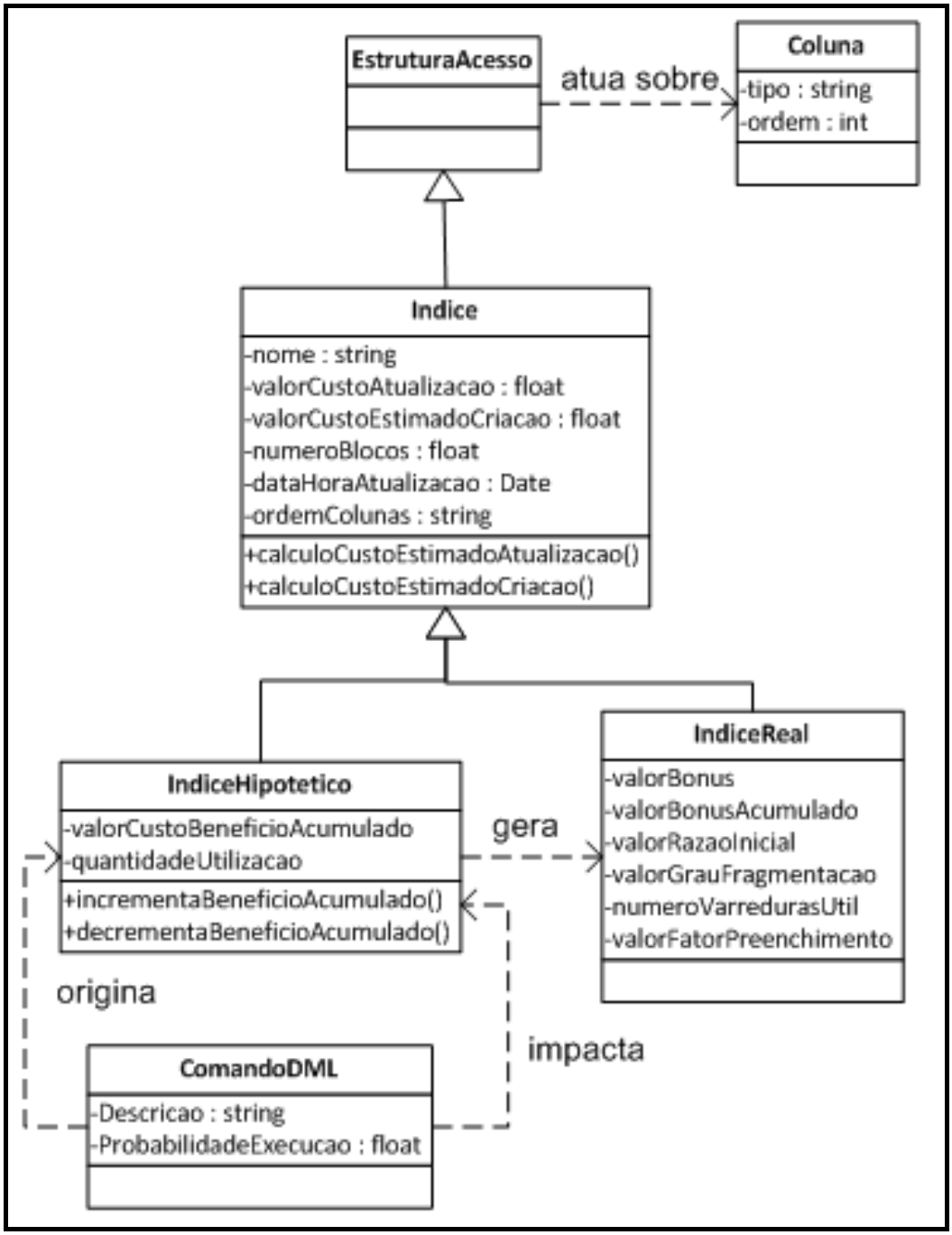

Figura 3-32 Ontologia de domínio – Comando DML, Índice e Coluna

Comando DDL

Por fim, define-se na ontologia de domínio os principais **comandos DDL** (*Data Definition Language*) que são usados para realizar ações de sintonia fina no banco de dados, criando ou removendo estruturas que podem prover maior agilidade no acesso aos dados registrados na base.

Os comandos DDL possuem as suas descrições de acordo com a sintaxe do SGBD (Figura 3-33). Os principais comandos DDLs são:

- Criação (**`create`**);
- Remoção (**`drop`**);
- Reconstrução (**`reindex`**).

Todos esses comandos podem ser aplicados sobre **tabelas** ou **estruturas de acesso** (Figura 3-33).

No caso do `reindex`, deve ser criada uma regra que o restrinja somente a estrutura de acesso denominada índice.

Além dos comandos oficiais do padrão SQL, define-se na ontologia, os comandos DDLs necessários à criação de estruturas hipotéticas. São eles: **`create hypothetical`** e **`drop hypothetical`**, responsáveis pela criação e remoção de estruturas hipotéticas, respectivamente.

Como visto anteriormente, as ações que são geradas no momento em que uma condição de regra é verdadeira acarretam em iterações com a base de dados. Dessa forma, são usados comandos DDL, onde existem funções definidas na própria ontologia de domínio, que são responsáveis por invocar suas ações no banco de dados.

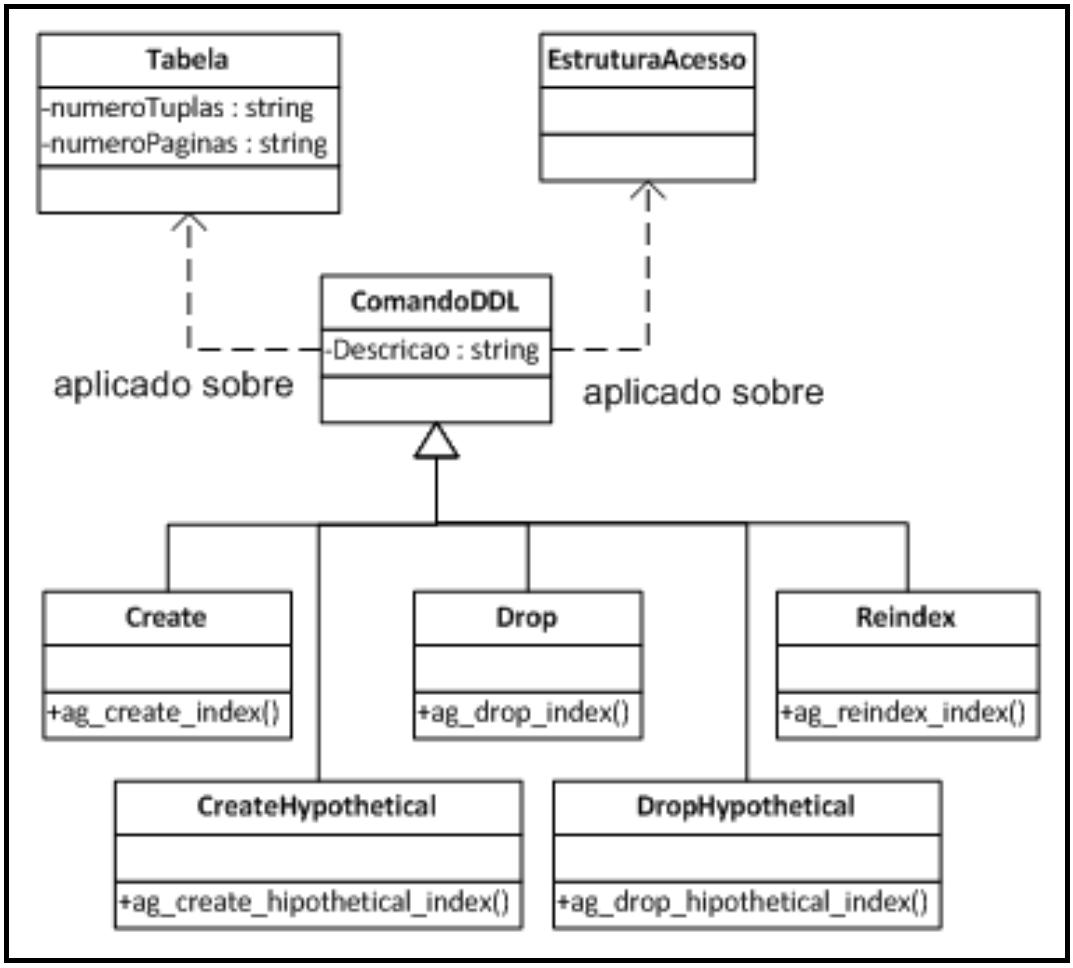

Figura 3-33 Ontologia de domínio – Comando DDL, Tabela e Estrutura de acesso

3.5. Considerações finais

A ontologia de aplicação descreve conceitos de uma tarefa aplicada em um determinado domínio. No escopo da presente tese, descrevem-se conceitos presentes no domínio das heurísticas de sintonia fina mencionadas no início desse capítulo, sobre a tarefa de executar tais heurísticas buscando um melhor desempenho da base de dados. Através do apoio da ontologia desenvolvida, conseguem-se informações semânticas sobre o domínio de forma a fornecer subsídios que possam ser usados como justificativas do DBA para demonstrar a eficácia do seu trabalho.

Nesse capítulo, apresenta-se também uma instanciação da ontologia usada no *framework* para uma heurística já conhecida na literatura.

A ontologia de domínio permite alterações em alto nível sobre os conceitos necessários para aplicar e ao mesmo tempo justificar a sintonia fina de banco de

dados. Por sua vez, a ontologia de tarefa possibilita um entendimento mais claro sobre quais regras que as heurísticas se baseiam para tomar suas decisões. Dessa forma, o DBA consegue adaptar qualquer conceito ou regra usada para a sintonia fina, de forma mais global e em alto nível. Com a ajuda da máquina de inferência, o DBA pode verificar qualquer inconsistência sobre suas definições.