2
Conceitos Básicos

Esse capítulo descreve sucintamente os principais conceitos necessários para o entendimento e motivação da presente tese. O capítulo é dividido nos principais assuntos envolvidos nesta tese: sintonia fina de banco de dados e ontologia.

2.1.
Sintonia Fina de Banco de Dados

Antes de detalhar o conceito de sintonia fina de banco de dados é necessário entender como as consultas submetidas ao banco de dados são processadas de maneira geral.

2.1.1.
Processamento de Consultas

Para processar uma consulta submetida ao banco de dados, o SGBD realiza os seguintes passos principais (Elmasri, 2011) (Garcia-Molina et al, 2001):

1. O analisador sintático verifica a sintaxe do comando para determinar se o mesmo está de acordo com as regras de gramática da linguagem de consulta (ex.: SQL);
2. A consulta é validada, verificando se todos os atributos e tabelas são válidos e semanticamente significativos no esquema do banco de dados que está sendo consultado;
3. A consulta é transformada em um grafo de operadores, chamado de plano lógico;
4. O plano lógico é transformado em um plano físico, ou seja, o plano de execução. Esse plano de execução indica não só as operações desempenhadas como também a ordem em que elas devem ser executadas. Além disso, indica também o algoritmo usado para realizar

cada passo e os caminhos para obter os dados armazenados bem como a forma que os dados são passados de uma operação para outra.

5. O processador executa as operações para gerar o resultado da consulta no banco de dados, em tempo de execução.

No escopo da presente tese, é importante destacar o passo em que o plano lógico é transformado no plano de execução da consulta. Tal tarefa é realizada pelo otimizador.

Um otimizador de consulta de um SGBD relacional é responsável por avaliar, dentro de um espaço largo de alternativas disponíveis, o caminho menos custoso para executar o comando. A seleção desse caminho é particular de cada SGBD e o custo é calculado com base nas estatísticas nele armazenadas. As estatísticas envolvem número de tuplas, tamanho médio de cada tupla e outros valores, tais como o número de páginas físicas usadas pela tabela ou número de tuplas distintas na tabela. Os DBAs e usuários não possuem autonomia suficiente para interferir na escolha do caminho mais eficiente para executar uma consulta. Diante disso, o DBA pode apenas prover alternativas de acesso aos dados armazenados na base de dados, surgindo assim o conceito de sintonia fina de banco de dados.

2.1.2. Sintonia Fina

A atividade de sintonia fina de um banco de dados compreeende o ajuste de suas configurações, parâmetros e projeto físico, incluindo a seleção de estruturas de acesso, a duplicação de estruturas físicas, a determinação dos objetos a serem particionados e os respectivos tipos de particionamentos, sempre de acordo com a carga de trabalho, de forma que seja atingido um melhor desempenho. Entende-se por carga de trabalho, um conjunto de requisições, sejam de consultas ou manutenções de dados (comandos DML), submetido ao banco de dados.

Conforme descrito na seção anterior, o otimizador analisa os possíveis caminhos de acesso aos dados e tais caminhos podem ser realizados com o auxílio de estruturas de dados. Quando o otimizador considerar somente estruturas que existem fisicamente na base de dados para gerar o plano de execução,

chamaremos nesta tese, de plano de execução real. Por outro lado, quando o otimizador considerar estruturas hipotéticas, além das estruturas reais, chamaremos de plano de execução hipotético (Monteiro, 2008). Consideramos estruturas hipotéticas aquelas que só existem nos metadados do banco de dados, ou seja, não estão criadas fisicamente na base de dados. Os metadados são informações do esquema de banco de dados que descrevem a estrutura do banco de dados e suas restrições (Garcia-Molina et al, 2001). É relevante ressaltar que a busca por melhor desempenho, realizada pela sintonia fina, não inclui alterações na escolha do plano diretamente, ou seja, a atividade de sintonia fina pode influenciar decisivamente nas alternativas de planos, mas não determinar o plano que o otimizador deve seguir para responder uma determinada requisição, pois isso é particular do otimizador.

Os DBAs podem apenas criar novas estruturas de acesso ou alterar parâmetros de configuração, que possibilitem a geração de caminhos alternativos, deixando que o otimizador decida qual seria a melhor forma de executar a consulta, selecionando ou não um caminho alternativo. Entre as estruturas de acesso disponíveis, têm-se: índice, visão materializada e partições.

Além disso, os SGBDs oferecem diversos parâmetros de controle (ex.: quantidade de memória, nível de concorrência entre outros) que podem ser ajustados para alcançar um funcionamento mais eficiente (Bruno, 2011) (Elmasri, 2011) (Date, 2004) (Shasha et al, 2003) (Ramakrishnan et al, 2003) (Weikum et al, 1994). O ajuste desses parâmetros não é uma tarefa trivial, pois necessita de compreensão dos algoritmos envolvidos no SGBD e os impactos de cada ajuste realizado (Bruno, 2011) (Chaudhuri et al, 2007). Sem a compreensão dos impactos envolvidos nos ajustes de parâmetros, ou seja, um entendimento sobre as interações entre os recursos de sistema e a carga de trabalho submetida ao SGBD, os ajustes podem degradar o desempenho do Banco de Dados (BD) ao invés de melhorá-lo.

Em um cenário mencionado em (Shasha et al, 2003), tem-se um exemplo da necessidade do entendimento sobre os impactos entre as decisões de sintonia fina. Em termos de literatura, pode-se considerar como uma recomendação, que o usuário do banco de dados nunca utilize funções de agregação (ex.: AVG), quando o tempo de resposta de transação for crítico. A razão para tal orientação é o fato de que esse tipo de função precisa varrer uma quantidade substancial de

dados, podendo bloquear outras consultas. No entanto, se essa função for submetida a um conjunto pequeno de tuplas que tenha sido selecionado por um índice, essa situação desfavorável não mais deve ocorrer. Sendo assim, é importante que o administrador responsável pela base de dados (DBA) entenda as razões para as definições de cada recomendação, podendo enriquecer a sua justificativa de decisão.

Outro cenário que também ilustra a necessidade do entendimento sobre as consequências de determinadas decisões locais em relação ao desempenho global do sistema é citado em (Barrientos, 2004): uma regra comumente usada considera que se colocando todas as páginas de índices em memória, melhora-se o desempenho de buscas indexadas. Além disso, aumentando-se o tamanho do *buffer* em detrimento de uma redução no tamanho da memória RAM, mantêm-se por mais tempo os dados na memória, o que diminui os acessos e a contenção de disco. Como consequência, tais ações podem deixar pouca memória disponível para ordenação ou *hash* e, em função disso, fazer com que o otimizador opte por planos de execução menos eficientes para determinadas consultas, comprometendo o desempenho do sistema. Dessa forma, o DBA tem como justificar que optou por não aumentar o tamanho do *buffer*, em virtude das más escolhas de planos de execução pelo otimizador.

Sendo assim, é importante realizar a sintonia fina de banco de dados tendo consciência sobre as razões das recomendações existentes.

2.1.3. Sintonia Fina Automática de Banco de Dados

Dado que encontrar a configuração ótima do projeto físico de banco de dados para uma dada carga de trabalho de forma manual não é uma tarefa trivial (Bruno, 2011), faz-se necessário o desenvolvimento de ferramentas que automatizem as tarefas de sintonia fina. Além disso, as tarefas relacionadas à infra-estrutura (manutenções de sistemas operacionais, SGBDs entre outros) estão ficando cada vez mais transparentes para o usuário, o que também demanda por ferramentas que automatizem a busca por melhoria de desempenho. Tais ferramentas devem ser capazes de adaptar-se dinamicamente às modificações da carga de trabalho (Bruno, 2011) (Lifschitz et al, 2004).

Diante dessa necessidade de automação, surge o conceito da auto-sintonia de banco de dados, sintonia fina automática de banco de dados ou *self-tuning* de banco de dados. Auto-sintonia de banco de dados significa o ajuste automático de configurações que buscam melhorar o desempenho através da diminuição do tempo de resposta sobre as execuções de transações ou comandos submetidos ao banco de dados.

Podemos considerar que as ferramentas de auto-sintonia de banco de dados envolvem três componentes principais: espaço de busca, modelo de custo e estratégia de enumeração (Bruno, 2011).

- **Espaço de busca:** no contexto de sintonia fina usando a estrutura de acesso de índice, considera-se como espaço de busca, o conjunto de índices candidatos para um determinado banco de dados e uma dada carga de trabalho. Em uma visão mais ampla, podemos considerar como sendo o conjunto de estruturas de acesso candidatas para uma base de dados de acordo com uma determinada carga de trabalho.
- **Modelo de custo:** é elaborado fora do otimizador. Ele avalia qual seria o custo de execução dos comandos que fazem parte de uma determinada carga de trabalho sobre uma configuração simulada durante a tarefa de sintonia fina. Quando os custos de execução são estimados sobre configurações hipotéticas, ou seja, que consideram estruturas sem estarem realmente materializadas, chamamos de camada de otimização *what-if*.
- **Estratégia de enumeração:** considera as limitações do ambiente em que o SGBD se encontra para enumerar configurações candidatas. Por exemplo, a estratégia pode considerar um espaço em disco limitado disponível para a criação de índices. Com a limitação de espaço físico, a estratégia pode simplificar o espaço de busca, passando a considerar apenas índices simples.

Considerando os resultados complexos da área e o aumento da sofisticação das máquinas de consultas e cargas de trabalho, as técnicas recentes de sintonia fina usam heurísticas para simularem configurações que sejam relevantes no espaço de busca analisado.

Embora esse tipo de ferramenta de sintonia fina automática tenha a vantagem de simular de forma ágil uma série de configurações de ambientes, ela

também possui desvantagens. O fato desse tipo de ferramenta tomar decisões sozinhas, sem intervenção humana, e sem uma explicação convincente sobre suas formas de raciocínio, torna-a uma ferramenta de difícil aceitação no mercado. As heurísticas usadas por uma ferramenta de auto-sintonia precisam ter o seu raciocínio explicitado. Seja um administrador de banco de dados ou uma empresa que contrate um serviço de sintonia fina de banco de dados, eles precisam ter a certeza de que todas as alternativas existentes de melhoria de desempenho para o seu ambiente foram consideradas e analisadas. Além disso, deve-se obter o conhecimento do motivo pelo qual certa solução foi escolhida como a melhor opção possível. Existe uma necessidade de se prover uma ferramenta de sintonia de banco de dados auto-explicativa.

2.1.4. Estruturas de acesso

As principais estruturas de acesso que são usadas por DBAs para agilizar a execução de comandos DML são índices e visões materializadas. Além disso, a técnica de particionamento de dados também é frequentemente usada. É importante lembrar que as estruturas de acesso são consideradas redundantes e consomem um espaço físico em disco. Sendo assim, sempre deve haver algum benefício proporcionado por tais estruturas que justifiquem a sua existência na base de dados. Além disso, o DBA deve considerar sempre o espaço disponível para esse tipo de estrutura ou partição antes de começar a analisar as alternativas de sintonia fina.

2.1.4.1. Índice

Um índice é uma estrutura definida sobre um conjunto de colunas de uma tabela, que resulta em um caminho de acesso para obter tuplas que satisfaçam certos predicados (Bruno, 2011). Eles podem melhorar consideravelmente o desempenho de processamento de uma consulta.

Na sintonia fina, é comum simular configurações para verificar como seria o desempenho do banco de dados se certas estruturas de acesso existissem. Nesse contexto, apresenta-se o conceito de índice hipotético. Um índice é considerado

hipotético quando ele possui apenas informações de metadados, ou seja, não existe fisicamente na base de dados e também não pode ser usado efetivamente como estrutura para acessar o dado hipoteticamente indexado (Monteiro, 2008) (Bruno, 2011). Nesta tese, iremos referenciar os índices hipotéticos e, no caso dos índices que existem fisicamente na base de dados, chamaremos de índices reais.

Os índices podem ser considerados clusterizados ou não clusterizados (Bruno, 2011) (Elmasri, 2011) (Date, 2004) (Shasha et al, 2003) (Ramakrishnan et al, 2003).

Índices clusterizados são aqueles sobre atributos, cujas tuplas, que possuam determinados valores para uma chave de busca desse índice, encontram-se fisicamente o mais próximo possível (Garcia-Molina et al, 2001). Por exemplo, um índice clusterizado sobre um determinado atributo A1, que tenha como chave de busca o valor VA1, deve possuir as tuplas que referenciam o valor VA1 armazenadas fisicamente em sequência.

Já os índices não clusterizados não possuem essa ordenação física das tuplas de atributos indexados.

Além disso, o índice pode ser classificado em primário ou secundário (Bruno, 2011) (Elmasri, 2011) (Date, 2004) (Shasha et al, 2003) (Ramakrishnan et al, 2003).

Um índice é classificado como primário quando a sua estrutura de dados determina a localização dos registros indexados. O índice primário é um arquivo ordenado cujos registros são de tamanho fixo com dois campos. O primeiro campo é do mesmo tipo de dado do campo de chave primária (chave de ordenação) e o segundo campo é um ponteiro para um bloco de disco que possui o registro. Um índice *cluster* é dito não primário se o atributo indexado não for chave primária. Ao contrário do índice primário, a estrutura de dados de um índice secundário não determina a localização de registros em um arquivo de dados, ele oferece um meio secundário para acessar o registro para o qual já existe um acesso primário.

O índice secundário apresenta a localização dos registros que pode ter sido decidida por um índice primário sobre algum outro campo. O índice secundário pode ser criado em um campo que é uma chave candidata e tem um valor único em cada registro, ou em um campo não chave com valores duplicados.

Os índices também podem ser caracterizados como densos ou esparsos (Bruno, 2011) (Elmasri, 2011) (Date, 2004) (Shasha et al, 2003) (Ramakrishnan et al, 2003).

Um índice denso tem uma entrada de índice para cada valor de chave de busca, ou seja, para cada registro.

Já um índice esparso (ou não denso), por sua vez, tem entradas de índice para somente alguns dos valores de busca. Sendo assim, um índice primário é um índice esparso, pois inclui uma entrada para cada bloco de disco ao invés de cada registro.

No contexto de sintonia fina, os índices são usados para prover um caminho alternativo e muitas vezes, mais ágil, de acesso aos dados armazenados na base de dados. De uma forma geral, as pessoas consideram que os índices agilizam apenas o acesso aos dados em comandos DML do tipo consulta, sendo os índices considerados custosos para comandos DML do tipo atualização.

Os índices são benéficos para consultas porque podem substituir o custo de percorrer completamente uma tabela (*full scan*) por operações sobre o índice.

Existem duas operações principais que são desempenhadas sobre os índices: varredura do índice (*index scan*) e busca no índice (*index seek*).

A varredura em um índice não clusterizado localiza o nó folha mais a esquerda em uma árvore B+ e retorna os valores das colunas no índice. Esse tipo de varredura não precisa realmente ir até o índice clusterizado ou percorrer a tabela contendo uma sequencia de tuplas não ordenadas (caso de não possuir índices) para realizar a recuperação do valor. A varredura em índices não clusterizados é uma boa alternativa para o caso de consultas que requerem somente um subconjunto de colunas presentes em páginas indexadas (Bruno, 2011).

A outra operação, de busca no índice, provê um caminho eficiente para avaliar os predicados. Essa operação atravessa a árvore do nó raiz até a primeira tupla no nó folha que satisfaça todos os predicados (se existe tal tupla). Tal operação desempenha a varredura análoga ao *index scan*. Diferente de árvores de busca binária, as árvores B+ tem uma abrangência muito alta, que reduz o número de operações de entrada e saída (E/S) requerido para buscar um elemento na árvore (Bruno, 2011).

No caso de execuções de comandos de atualizações, os índices podem ser custosos. Esse é o principal fator contra a adoção de índices no caso de cargas de trabalho que possuam muitas atualizações (Bruno, 2011). Cada vez que uma tupla é modificada, inserida ou removida, todos os índices relevantes precisam ser atualizados (Bruno, 2011). Em inserções e remoções em uma tabela, todos os índices definidos sobre ela precisam ser atualizados, exceto no caso de índices esparsos, cujos ponteiros dos blocos não sejam impactados. No caso de modificações, somente os índices definidos sobre a coluna atualizada, que são modificados. Então, nem sempre um índice é considerado custoso para comandos de atualizações. Caso o índice esteja sobre uma coluna que não sofra atualização e tal coluna indexada seja usada como predicado de busca em um comando `UPDATE`, o índice pode contribuir na busca do predicado e na agilidade de execução do comando.

A seleção de índices de forma equivocada pode acarretar nos seguintes contratempos (Shasha et al, 2003):

- Índices criados, que precisam ser mantidos cada vez que ocorre uma atualização sobre a tabela indexada, mas que nunca são usados para agilizar quaisquer execução de comandos DML. Por exemplo, no cenário em que um índice é criado sobre uma tabela com poucos registros e o otimizador prefere realizar uma varredura da tabela do que usar o índice.
- Junções de múltiplas tabelas que demoram horas por causa da existência de índices errados.

Dessa forma, é um desafio para o DBA, no momento da tomada de decisão da sintonia fina, decidir quais índices são úteis. Um índice é considerado útil quando agiliza as consultas presentes na carga de trabalho, sem degradar o desempenho de outros comandos de forma significativa.

Existem boas práticas de sintonia fina, mencionadas em livros, em relação aos índices. São elas (Bruno, 2011) (Shasha et al, 2003):

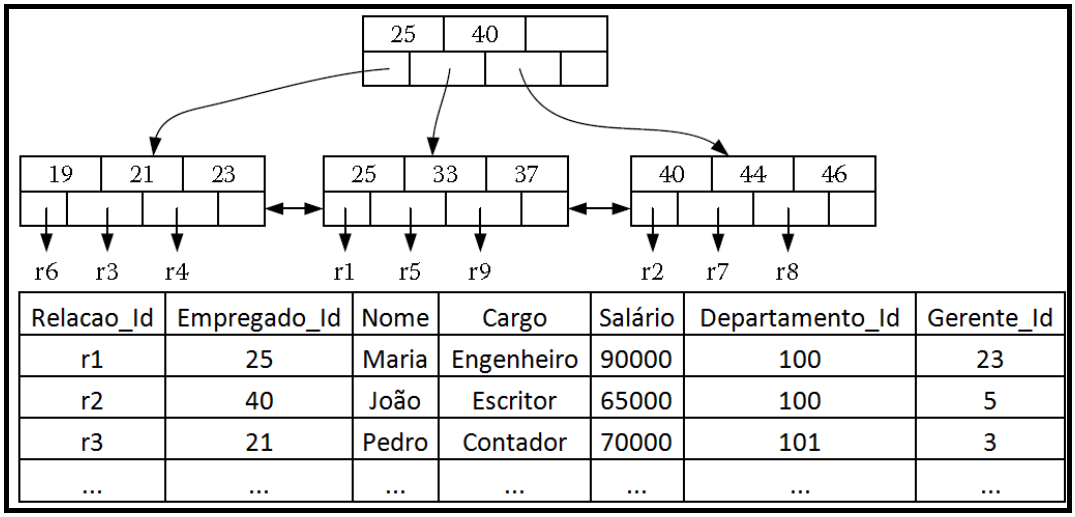
- **Índices de chaves primárias e mais chaves estrangeiras.** No caso de existirem muitas junções entre colunas que estejam presentes em chaves primárias e estrangeiras, incluir tais colunas em índices pode melhorar o desempenho de consultas complexas.

- **Colunas referenciadas frequentemente em cláusulas `where` são boas candidatas a serem indexadas.** Colunas referenciadas em predicados de igualdade ou diferença em cláusulas `where` podem ser eficientemente buscadas por índices apropriados.
- **Evitar índices redundantes.** Índices definidos sobre as mesmas colunas raramente são benéficos. Eles não concedem uma utilidade adicional sobre um índice único, mas ao mesmo tempo, usam espaço extra e precisam ser atualizados.
- **Usar índices com cautela sobre colunas atualizadas frequentemente.** Como já mencionado, quando as colunas são atualizadas no banco de dados, todos os índices definidos sobre tais colunas precisam também ser mantidos. Uma coluna que é atualizada frequentemente pode se transformar em um gargalo de desempenho ao invés de um índice útil.
- **Considerar índices de cobertura para consultas críticas.** Índices de cobertura são aqueles que possuem todas as colunas requeridas por uma consulta. Eles são bons para melhorar o desempenho de consultas críticas e muito frequentes. Por outro lado, esse tipo de índice pode ocupar bastante espaço e ser útil para poucas consultas.
- **Evitar índices em tabelas pequenas.** Índices em tabelas que ocupam uma ou duas páginas não apresentam benefícios de desempenho e ainda aumentam o número de estruturas a serem mantidas e administradas.

Estruturas de dados

A estrutura de dados mais usada pelos SGBDs comerciais para índices é a de árvore B+ (Figura 2-1). Esse tipo de estrutura é uma árvore, onde cada nó é associado a um intervalo de valores para as colunas indexadas (Bruno, 2011) (Knuth, 1973). Ao contrário da árvore B, todos os ponteiros de dados são armazenados no nível folha da árvore, que apontam para a tupla real na tabela indexada. Essa estrutura beneficia a maioria dos tipos de consultas, incluindo

consultas sobre faixas de valores (*range queries*) (Date, 2004) (Shasha et al, 2003).



Relacao_Id	Empregado_Id	Nome	Cargo	Salário	Departamento_Id	Gerente_Id
r1	25	Maria	Engenheiro	90000	100	23
r2	40	João	Escritor	65000	100	5
r3	21	Pedro	Contador	70000	101	3
...	...	...	...	...	...	...

Figura 2-1 Exemplo de estrutura de árvore B+ usada por um SGBD relacional (Bruno, 2011)

A estrutura de tabela *hash* (Figura 2-2) usa um método de armazenamento de pares do tipo chave-valor, baseado em uma função chamada de *hash* (Elmasri, 2011) (Shasha et al, 2003). Dada uma chave de busca, a função *hash* identifica onde essa chave deve ser armazenada e aloca-a juntamente com os ponteiros associados. Esses ponteiros podem ser para a página ou bloco que possui o registro referente à chave de busca. Índices que usam a estrutura de tabela *hash* são indicados para consultas que envolvam acesso a linhas específicas, pontuais (Date, 2004). Por outro lado, a estrutura de tabela *hash* não é boa para consultas que possuam predicados que considerem intervalos (*range*) e prefixos (Date, 2004) (Shasha et al, 2003).

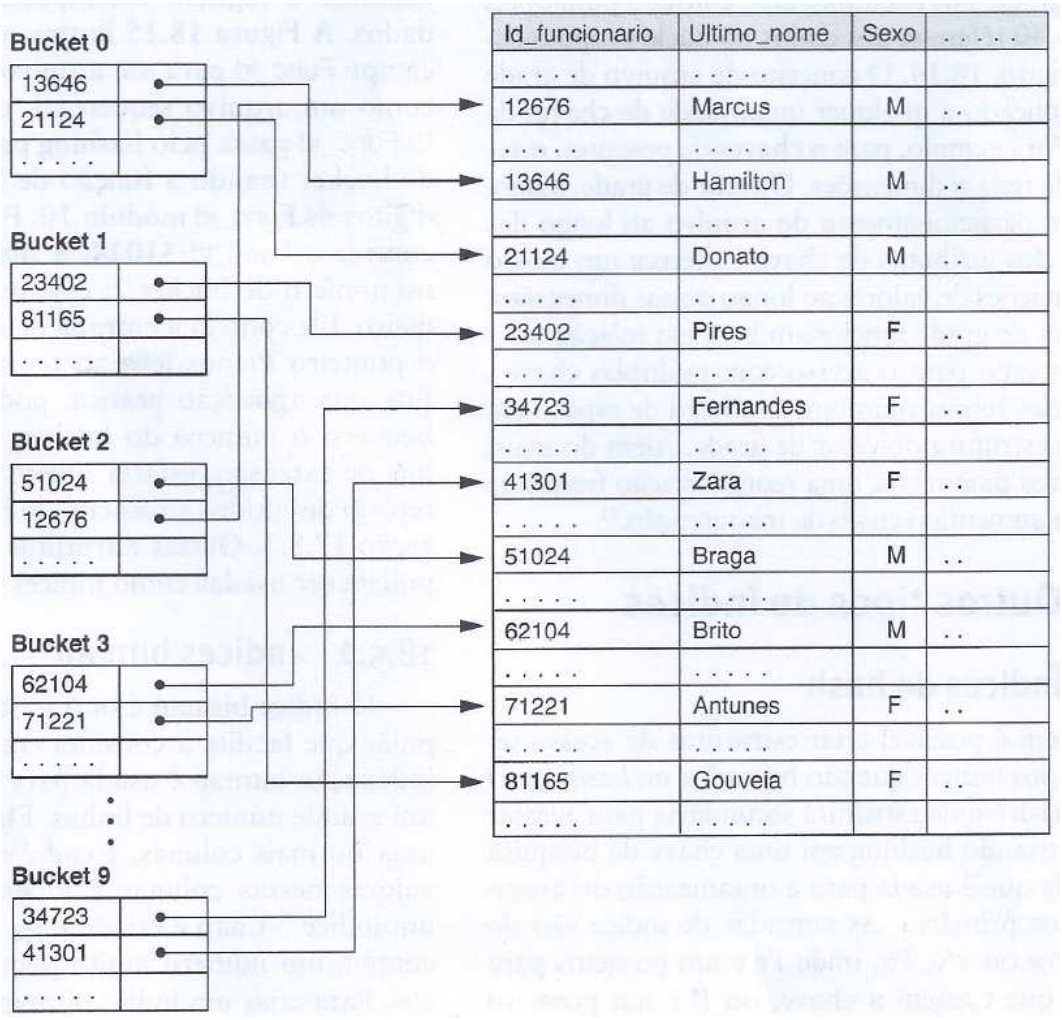
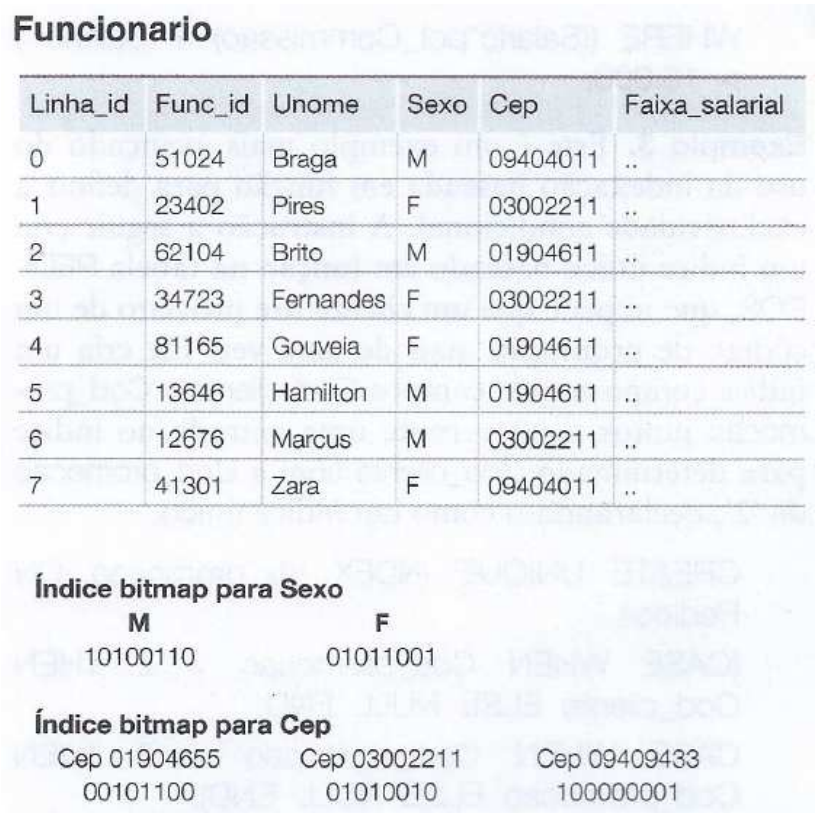


Figura 2-2 Exemplo de estrutura de tabela *hash* usada por um SGBD relacional (Elmasri et al, 2011)

A estrutura *bitmap* (Figura 2-3) é um vetor de *bits* por valor de atributo, onde o tamanho de cada *bitmap* é igual ao número de registros da tabela indexada (Elmasri, 2011) (Date, 2004) (Shasha et al, 2003). Cada entrada no vetor corresponde a um *bit*. Esse *bit* recebe valor 1 no caso do valor da chave corresponder ao valor da coluna e caso contrário, recebe 0. Por exemplo, se o atributo indexado for sexo (Figura 2-3), irão existir dois vetores *bitmaps*, cada um representando um possível valor (M – Masculino e F – Feminino). Se a tupla *t* tiver como valor do atributo, o sexo "M", o vetor correspondente ao sexo "M" recebe o valor 1 na posição *t*. A vantagem da estrutura de índice *bitmap* é que ela é mais compacta, e realizar a interseção entre vetores *bitmaps* é muito mais rápido do que realizar a interseção de coleções de identificadores de registros. O melhor caso do uso de índice *bitmap* é quando cada predicado é não seletivo mas todos os

predicados juntos são completamente seletivos (Shasha et al, 2003). Por exemplo, uma consulta que busque todas as pessoas que possuam cabelo marrom, use óculos, tenha entre 30 e 40 anos e olhos azuis. A execução dessa consulta seria uma completa interseção entre o *bitmap* de cabelo marrom, o bitmap de óculos, a união do *bitmap* de idade que representa as idades entre 30 e 40 e o *bitmap* de olhos azuis.



Funcionario

Linha_id	Func_id	Unome	Sexo	Cep	Faixa_salarial
0	51024	Braga	M	09404011	..
1	23402	Pires	F	03002211	..
2	62104	Brito	M	01904611	..
3	34723	Fernandes	F	03002211	..
4	81165	Gouveia	F	01904611	..
5	13646	Hamilton	M	01904611	..
6	12676	Marcus	M	03002211	..
7	41301	Zara	F	09404011	..

Figura 2-3 Exemplo de estrutura *bitmap* (Elmasri et al, 2011)

2.1.4.2. Visão materializada

Visão pode ser definida como uma única tabela derivada de outras tabelas (Elmasri et al, 2011). Essas outras tabelas podem ser tabelas da base ou outras visões. Uma visão normalmente não existe de forma física. Os seus dados devem ser obtidos em tempo de execução. No entanto, as visões que não existem de forma física não influenciam de forma significante no desempenho do SGBD (Shasha et al, 2003). Já a visão que possui seus dados fisicamente na base, podem agilizar consideravelmente a execução de uma consulta. Essas visões são chamadas de visões materializadas.

Visão materializada é o resultado de uma visão (consulta) que é armazenado e usado pelo otimizador de forma transparente (Bruno, 2011). Uma visão materializada deve ser mantida e persistida como uma tabela regular. Assim como uma tabela regular, ela também pode ser indexada.

A existência de visões materializadas faz com que o DBA tenha que identificar possíveis consultas na carga de trabalho que possam ser beneficiadas por elas. Uma vez identificadas, essas consultas precisam ser reescritas para passarem a referenciar a visão materializada correspondente.

As visões materializadas são consideradas boas para consultas que envolvam funções de agregação (Shasha et al, 2003). Isso ocorre porque os dados já ficam pré-calculados e armazenados na base de dados, agilizando a execução da consulta. Por outro lado, as visões materializadas podem consumir uma quantidade significativa de disco e ainda terem a necessidade de serem mantidas em comandos de atualização (Bruno, 2011). Além disso, existe um custo grande para manter as visões materializadas com os dados atualizados.

2.1.4.3. Particionamento de dados

No contexto de banco de dados, particionamento é uma técnica para reduzir a carga em certo componente do sistema, seja pela divisão da carga sobre mais recursos ou pela expansão da carga em função do tempo (Shasha et al, 2003). O particionamento ou fragmentação divide uma determinada tabela em um conjunto de partições ou fragmentos disjuntos, para fins de armazenamento físico (Date, 2004). Esse particionamento pode melhorar de modo significativo a facilidade de gerenciamento e de acesso da tabela em questão.

Para disponibilizar um melhor desempenho, a decisão mais importante é identificar a unidade apropriada de distribuição (Öszu et al, 2001). Normalmente, as aplicações acessam subconjuntos de dados, logo, uma tabela completa não é uma boa unidade de distribuição. Além disso, existem aplicações que estão alocadas em *sites* diferentes e têm visões definidas sobre uma dada tabela. Nesse caso, a tabela inteira pode ser a unidade de distribuição, podendo ou não ser replicada nos *sites* em que residem as aplicações. Por fim, existe a decomposição de uma tabela em partições/fragmentos, sendo cada um deles tratado como uma

unidade de distribuição. Isso permite que várias transações sejam executadas de forma concorrente. O particionamento ou fragmentação aumenta o nível de concorrência e, por consequência, a vazão do sistema (Öszu et al, 2001).

Todavia, se as aplicações que acessam a base de dados particionada têm requisitos conflitantes que impedem a decomposição da tabela em partições mutuamente exclusivas, elas podem sofrer a degradação do desempenho (Öszu et al, 2001). Por exemplo, quando uma aplicação precisar acessar dados de duas partições e depois realizar sua união ou junção, resultará em um custo elevado.

Tipos de Particionamento de dados

O particionamento vertical divide a tabela *t* em duas ou mais partições, cada uma contendo todas as linhas de *t* com somente um subconjunto de colunas (Bruno, 2011) (Elmasri et al, 2011) (Öszu et al, 2001). No caso de existirem muitas consultas que acessam poucas colunas em uma tabela, o particionamento vertical pode reduzir a quantidade de dados que precisa ser percorrida durante o processamento da consulta (Bruno, 2011). Porém, se existem consultas que precisam referenciar colunas presentes em diferentes partições, isso vai incluir um custo adicional de reconstrução da tabela, na forma de uma junção.

O particionamento horizontal divide a tabela *t* por tuplas entre múltiplas partições que compartilham o mesmo esquema de *t* (Bruno, 2011) (Elmasri et al, 2011) (Öszu et al, 2001). Essa divisão é especificada usando uma função de particionamento que mapeia uma dada linha da tabela para um número de partição. A maioria dos SGBDs usam dois tipos de função: *hash* e *range*. A função *hash* é definida por um conjunto de colunas *C* e um valor inteiro *K*. Para cada tupla da tabela original, a função mapeia os valores das colunas *C* em um número de partição pseudo-randômico entre 1 e *K*, através do uso da função de *hash* definida no sistema (Bruno, 2011). A função *range* é definida por um conjunto de colunas *C* e uma sequência ordenada de intervalos disjuntos *V* que cobrem o domínio de *C*. Uma tupla com valor $C = C_0$ mapeia para a partição associada com o intervalo em *V* que inclui C_0 (Bruno, 2011). O particionamento horizontal pode beneficiar consultas que possuam predicados de igualdade ou intervalo, pois eliminam a partição estática e processam somente as partições relevantes.

O particionamento híbrido ou misto é quando se mistura os dois tipos de particionamento (Elmasri et al, 2011) (Öszu et al, 2001). Mesmo não considerando esse tipo de particionamento um tipo primitivo de estratégia de particionamento, é evidente que muitos particionamentos reais podem ser híbridos (Öszu et al, 2001).

2.1.5. Classificação da abrangência da sintonia fina

Em (Lifschitz et al, 2004) é proposta uma classificação das linhas de pesquisa de acordo com o foco dado na abordagem do problema, ou seja, de acordo com a abrangência da solução. Embora seja proposto esse tipo de classificação para trabalhos de auto-sintonia, ele pode ser generalizado para a área de sintonia fina como um todo. A classificação pode ser: global ou local.

A sintonia fina realizada de forma global busca estudar como é possível tomar decisões que tragam benefícios de desempenho para o sistema como um todo. Nesse caso, a desvantagem é que esse tipo de sintonia exige muito mais conhecimento do DBA ou ferramenta sobre os relacionamentos de impactos que existem entre as diversas soluções disponíveis de sintonia fina. O DBA deve ser capaz, por exemplo, de identificar que uma solução pode não ser a melhor, se tomada isoladamente, mas que em um contexto mais amplo, pode gerar muito mais benefício para a carga de trabalho. No contexto de sintonia global, consegue-se identificar um problema que as ferramentas de auto sintonia trazem: a falta de flexibilidade na escolha da solução. A maioria dessas ferramentas apresenta um único cenário de solução. Talvez, o DBA possa ter experiência suficiente com o ambiente que está lidando, de forma a verificar que se a ferramenta indicasse uma segunda opção, esta fosse melhor do que a primeira solução indicada pela ferramenta. Como o DBA já conhece o domínio, ele saberia que, embora o ganho da primeira opção fosse maior para o momento, a escolha da segunda opção, em longo prazo e pensando globalmente, tornar-se-ia muito mais vantajosa.

Já a sintonia realizada de forma local, busca estudar problemas específicos de sintonia existentes nos SGBDs sem se preocupar com as consequências de forma geral. Entre as decisões de sintonia fina local, podemos citar a reescrita de consultas, que busca a melhoria do desempenho de uma consulta em particular e a

seleção de índices sem considerar a existência de outras estruturas de acesso. Além disso, a sintonia local também pode se concentrar na melhoria de determinados parâmetros, como aqueles ligados aos ajustes de áreas de memória.

A vantagem da sintonia local é que, focando em um problema específico, a possibilidade de se ter uma solução mais eficiente para esse problema é muito maior do que se preocupando com os impactos gerais que podem ocorrer. Como a literatura (Shasha et al, 2003) já menciona, o DBA deve pensar globalmente, mas corrigir localmente. E a desvantagem desse tipo de sintonia é, como já discutido, a possibilidade de se ter outras soluções locais, classificadas em segundo plano, mas que possam ser melhores globalmente. Tais soluções, não consideradas ótimas, são descartadas por ferramentas de auto-sintonia e não são nem apresentadas ao DBA para que o mesmo possa analisar a solução de forma global. Isso impossibilita o DBA de decidir por uma solução global mais eficiente.

Algumas decisões de sintonia fina local podem ser benéficas para certos tipos de comandos, mas acarretar em um custo muito alto para outros tipos. Por exemplo: um índice pode agilizar a execução de uma consulta, mas degradar o desempenho de um comando de atualização. Por isso, existe um grande interesse nas decisões tomadas para uma dada carga de trabalho (Bruno, 2011).

A sintonia fina local pode ser aplicada, principalmente, em relação a carga de trabalho, a aplicação ou ao requisito.

As principais ferramentas de sintonia fina são direcionadas à carga de trabalho (*workload-oriented*) que é submetida à base de dados. É essa a carga que vai conduzir as decisões da ferramenta. Através do conjunto de comandos DML, que fazem parte da carga de trabalho, que a ferramenta vai analisar o impacto entre as suas decisões. A ferramenta também pode avaliar o tipo de comando que domina a carga, ou seja, se existem mais comandos de consultas ou de atualizações. Com isso, a ferramenta consegue decidir melhor a sua opção de sintonia fina. Quanto mais a carga de trabalho refletir a realidade do SGBD, melhor será a decisão de sintonia fina, que é realizada com o apoio de alguma heurística.

As ferramentas orientadas à aplicação (*application-oriented*) buscam identificar pontos na aplicação que acessam a base de dados e que podem degradar o desempenho do SGBD. Algumas boas práticas que podem ser consideradas na análise da aplicação são (Shasha et al, 2003): evitar iterações do

usuário dentro de transações, acessar somente as linhas e colunas necessárias e minimizar o número de compilações de consultas. Além disso, a ferramenta pode propor reescrita de consultas direto na própria aplicação. A vantagem dessa orientação é que o DBA não precisa selecionar os principais comandos para uma carga de trabalho e submeter a uma ferramenta de auto-sintonia.

A ferramenta orientada ao requisito (*requirement-oriented*) foca nos requisitos do sistema e a partir deles, gera o modelo lógico, o transforma em modelo físico e com a ajuda de regras e heurísticas, infere possíveis consultas que possam derivar decisões de sintonia fina.

De uma forma geral, todas as orientações acabam sendo por carga de trabalho. O que difere uma orientação de outra é a etapa inicial de cada uma. A primeira exige que o DBA já possua um conhecimento prévio da carga submetida ao banco de dados; a segunda e a terceira isentam o DBA desse conhecimento e originam a carga de trabalho da aplicação e da especificação de requisitos, respectivamente.

2.2. Ontologia

Existem muitas definições de ontologia na literatura. Na área da filosofia, todas as definições mencionam basicamente a existência de objetos que possuem conceitos e que esses conceitos podem se relacionar dentro de um domínio específico (Corcho et al, 2003). Na área da computação, uma ontologia define um conjunto de primitivas representacionais, que modelam um domínio de conhecimento ou discurso. As primitivas são tipicamente classes (ou conjuntos), atributos (ou propriedades) e relacionamentos (ou relações entre membros de classes) (Gruber, 2009).

A ontologia se aplica à integração semântica, aumentando o nível de abstração, de maneira que as pessoas e os sistemas possam focar no contexto e no relacionamento. Através de uma ontologia, o sistema pode, de maneira automática, inferir algumas interpretações de acordo com os termos e relacionamentos definidos no domínio específico, facilitando o entendimento para a realização de uma integração semântica.

Segundo (NOY et al, 2001), as ontologias podem ser aplicadas com os seguintes objetivos:

- Compartilhar um entendimento comum da estrutura de informação entre pessoas ou agentes de *software*;
- Permitir o reuso de elementos do domínio de conhecimento;
- Tornar explícitas as premissas e suposições do domínio;
- Separar conhecimentos de domínio do conhecimento operacional;
- Analisar o conhecimento do domínio.

2.2.1. Classificação de ontologias

Foram propostas diferentes classificações de ontologias na literatura (Jasper et al, 1999) (Guarino, 1998) (Uschold et al, 1996). Um sistema de classificação que utiliza a generalidade da ontologia como o critério principal para a classificação foi proposto por Guarino (Guarino, 1998). Neste sistema, o autor identifica a seguinte classificação: ontologia de fundamentação, ontologia de domínio, ontologia de tarefa e ontologia de aplicação.

2.2.1.1. Ontologia de fundamentação

A ontologia de fundamentação, topo ou alto nível descreve conceitos gerais, que são independentes de um problema particular ou domínio específico (Herre et al, 2006). Em geral, esse tipo de ontologia pode ser reusado na elaboração de novas ontologias. Entre as principais ontologias de fundamentação usadas na literatura, temos:

- DOLCE (*Descriptive Ontology for Linguistic and Cognitive Engineering*) (Bottazzi et al, 2006) → desenvolvida como parte do projeto *WonderWeb Foundational Ontologies Library* (WFOL) e usada como um módulo de referência para bibliotecas.
- GFO (*General Formal Ontology*) (Herre et al, 2006) → desenvolvida pelo grupo de pesquisa Onto-Med (*Ontologies in Medicine*) na Universidade de Leipzig, integrando objetos e

processos. Criada inicialmente para áreas médicas, biológicas e biomédicas.

- UFO (*Unified Foundational Ontology*) (Guizzardi et al, 2008) → baseada em resultados da Lógica Filosófica, Filosofia da Linguagem, Psicologia Cognitiva e Linguística. Essa ontologia busca unificar outras ontologias de fundamentação, tais como a GFO e a DOLCE.
- UPPER CYC (Lenat et al, 1990) → ontologia proprietária e criada primeiramente com o objetivo de dar suporte a aplicações de inteligência artificial. Consiste em uma representação formal de fatos, regras e heurísticas de raciocínio sobre objetos e eventos da vida do dia a dia.
- SUMO (Suggested Upper Merged Ontology) (Niles et al, 2001) → ontologia criada com o objetivo de definir termos de propósito geral e usada por aplicações de busca, linguística e raciocínio. Essa ontologia é mapeada para todo léxico do WordNet e mantida pela IEEE (Institute of Electrical and Electronics Engineers).

2.2.1.2. Ontologia de domínio

Uma ontologia de domínio descreve as classes de conceitos e os relacionamentos entre esses conceitos que definem um domínio específico, ou seja, uma área de aplicação, podendo esse tipo de ontologia ser especificado através da especialização de conceitos presentes na ontologia de alto nível. Algumas ontologias de domínios propostas na literatura são:

- *Gene Ontology* (GO, 2013) → desenvolvida por membros do consórcio GO (*Gene Ontology*) e provê um vocabulário controlado de termos para descrever as características de genes e dados de anotações de genes, cruzando espécies e bases de dados.
- *Plant Ontology* (Cooper et al, 2013) → vocabulário controlado que descreve a anatomia da planta e a morfologia e estágios de desenvolvimento de todas as plantas. Ontologia iniciada em janeiro

de 2011 com membros fundadores das universidades: *Oregon State University*, *New York Botanical Garden* e *Cornell University.*

- *SQL Ontology* (Sosnovsky et al, 2008) → desenvolvida para dar suporte ao desenvolvimento de conteúdo educacional em SQL e facilitar a integração de múltiplos sistemas educacionais nesse domínio, enquanto garante uma representação objetiva de semânticas SQL.

2.2.1.3. Ontologia de tarefa

A ontologia de tarefa descreve os conceitos e relacionamentos entre esses conceitos usados para a realização de uma tarefa/atividade genérica, por exemplo, conceitos envolvidos na resolução de um determinado problema ou objetivos a serem atingidos por determinado domínio. Algumas tarefas podem ser o diagnóstico de um paciente, realização de vendas e audiências. Alguns exemplos de ontologias de tarefa são:

- Eureka (Everett et al, 2002) → uso de ontologia na tarefa de processamento de linguagem natural.
- E-OntoUML (Martins, 2009) → uso de ontologia de tarefa para capturar a decomposição de tarefa, com base no diagrama de atividades e a participação dos papéis de conhecimento nas mesmas.

2.2.1.4. Ontologia de aplicação

A ontologia de aplicação descreve conceitos que dependem tanto de um domínio em particular (especialização da ontologia de domínio) quanto de uma tarefa específica (especialização da ontologia de tarefa). Estas ontologias são mais específicas por serem utilizadas dentro das aplicações. Alguns exemplos de ontologias de aplicação são:

- (Everett et al, 2002) → esta ontologia de aplicação usa a ontologia do domínio de discurso do texto e a ontologia de tarefa Eureka.
- (Martins, 2009) → esta ontologia de aplicação usa a ontologia de domínio OntoUML, que é baseada no diagrama de classes, para a

modelagem dos papéis de conhecimento envolvidos e suas propriedades e relações. E também usa a ontologia de tarefa E-OntoUML, baseada no diagrama de atividades.

2.2.2. Metodologias para construção de ontologias

Existem diversas metodologias para suportar o processo de construção de ontologias. Algumas delas são: (Noy et al, 2001) (Fernandéz-Lopéz et al, 2002) (Grunninger et al, 2000) (Uschold et al, 1996).

A metodologia proposta por (Uschold et al, 1996) possui quatro etapas: identificação, construção, avaliação e documentação.

A etapa de identificação consiste em descrever o motivo do desenvolvimento da ontologia e com qual objetivo ela será usada. Durante a etapa de construção, os principais conceitos e relacionamentos são identificados e definidos através de textos e posteriormente, através de uma linguagem formal. Além disso, verifica-se a possibilidade de reuso de ontologias existentes. A próxima etapa, de avaliação, envolve o uso de critérios técnicos para validar a ontologia em relação ao mundo real. Por fim, na etapa de documentação, são elaboradas as descrições do processo realizado.

A metodologia proposta por (Grunninger et al, 2000) é chamada de TOVE (*Toronto Virtual Enterprise*). Essa metodologia inicia com a descrição de cenários motivacionais. Esses cenários envolvem problemas que não são cobertos adequadamente por ontologias existentes. Com base nesses cenários, são geradas diversas soluções possíveis. Em seguida, ocorre a elaboração de questões de competência que precisam ser representadas e respondidas pela ontologia a ser construída. Posteriormente, os termos da ontologia são especificados em uma linguagem formal. Da mesma forma, as questões de competência também são definidas em uma linguagem formal. Além disso, são definidos axiomas que irão definir a semântica e os relacionamentos entre os conceitos definidos na ontologia. Por fim, a ontologia é validada, quanto a sua completeza, através das questões de competência que foram descritas de maneira formal.

Diferente das demais metodologias, a *Methontology* (Fernandéz-Lopéz et al, 2002) fornece um apoio automatizado para a construção de ontologias. O

desenvolvimento da ontologia é de acordo com as atividades que precisam ser finalizadas. As atividades são classificadas em: atividades de gerenciamento de ontologia (elaboração de cronogramas, garantia de qualidade), atividades de desenvolvimento de ontologia (levantamento do ambiente, viabilidade, especificação, formalização, uso) e atividades de suporte (avaliação, documentação, gerência de configuração).

A *ontology 101* (Noy et al, 2001) consiste de um guia simples para construção de ontologias, que é direcionado para usuários não experientes. As etapas desse guia envolvem: (a) determinar o domínio e o escopo da ontologia, onde é definido o que a ontologia irá cobrir em um determinado domínio, para qual a finalidade que esta ontologia será utilizada, quais questões que a ontologia deverá responder, quem usará e manterá a ontologia. (b) considerar as ontologias existentes e reutilizá-las. (c) enumerar os termos importantes na ontologia. (d) definir as classes e a hierarquia entre as classes. Uma classe define um grupo de indivíduos que possuem propriedades comuns e compartilhadas. (e) definir as propriedades das classes, ou seja, determinar os relacionamentos entre os indivíduos e seus valores de dados. (f) definir as facetas das propriedades, ou seja, a cardinalidade, o tipo de valores, o domínio. (g) criar as instâncias individuais das classes na hierarquia.

A presente tese utiliza a *ontology* 101, por possuir um guia direcionado a usuários não experientes e também por ser a metodologia indicada pela ferramenta Protégé (Protégé, 2013). Tal ferramenta foi escolhida por ser gratuita e pelo fato de já existir um conhecimento prévio adquirido.

2.3. Considerações Finais

É importante realizar a sintonia fina de banco de dados tendo consciência sobre as razões das recomendações existentes. Um dos objetivos dessa tese não é buscar soluções ótimas de sintonia fina, mas sim ter uma base de raciocínio, em nível conceitual, para as decisões tomadas por um DBA ou uma ferramenta de auto-sintonia. Existe uma variedade de alternativas que o DBA ou uma ferramenta pode usar. Algumas dessas alternativas envolvem as estruturas de acesso. As principais estruturas são os índices, visões materializadas e partições de dados.

Uma vez que as alternativas sejam avaliadas, não basta apresentar somente a decisão final de uma configuração recomendada para o banco de dados. É importante demonstrar uma justificativa sobre tal decisão, apresentando as soluções que também foram pensadas e que por algum motivo foram descartadas.

Essa necessidade torna-se ainda mais evidente em ferramentas de auto-sintonia, onde as decisões são tomadas sozinhas, sem intervenção humana, e sem uma explicação convincente sobre suas formas de raciocínio, tornando-as com difícil aceitação no mercado.

Diante disso e pelo fato de explicitar o conhecimento do domínio, agregando maior qualidade de semântica, a ontologia de aplicação surge com grande potencial para ajudar na comprovação da eficácia do trabalho de sintonia fina do DBA. Com aplicação em diversas áreas, a ontologia tem se mostrado muito útil em cenários que exijam padronização de termos, organização de conceitos, definição de regras formais, realização de inferências e apoio no raciocínio humano. Por essas razões, a ontologia vem se consolidando na área de informática e demonstra ser uma forte candidata ao apoio na comprovação da eficácia do trabalho de sintonia fina de banco de dados.

Principalmente pelo fato de explicitar as premissas e suposições do domínio, a ontologia se relaciona com o objetivo da presente tese, que busca comprovar a eficácia do trabalho de sintonia fina do DBA. No escopo dessa tese, desenvolve-se uma ontologia de aplicação, onde se descreve a ontologia de domínio de sintonia fina de banco de dados com seus conceitos, propriedades, indivíduos e regras bem como a ontologia de tarefa de execução de heurísticas de sintonia fina.