2
Conceitos Básicos

Antes de descrevermos a abordagem para sintonia automática do projeto físico de bancos de dados proposta neste trabalho, vale ressaltar alguns conceitos relativos ao projeto físico, às estruturas que compõem um projeto físico, à sintonia e auto-sintonia em sistemas de banco de dados relacionais. Além disso discutimos a importância do uso de *benchmarks* na realização de testes de desempenho.

2.1
Projeto Físico de Bancos de Dados

O projeto físico de um banco de dados é uma atividade na qual o objetivo não é apenas obter uma estrutura apropriada para o armazenamento dos dados, mas fazê-lo de maneira que se assegure o acesso aos dados com um desempenho adequado. Assim, o projeto físico inclui a seleção das estruturas de índices, as visões materializadas a serem utilizadas, as tabelas a serem particionadas e os tipos de particionamento mais adequados, a duplicação de estruturas físicas e até mesmo a desnormalização de tabelas, como forma de acelerar o desempenho das consultas. Desta maneira, para um determinado esquema conceitual, há diversas alternativas de projeto físico. Projetos físicos distintos irão apresentar desempenhos diferentes. Contudo, não é viável tomar decisões de projeto físico e realizar análises de desempenho significativas até que se conheçam as consultas, as transações e as aplicações que se espera que sejam executadas no banco de dados. Deve-se analisar essas aplicações, suas freqüências esperadas, as restrições de tempo para suas execuções e a freqüência esperada das operações de atualização (Ramakrishnan02, Navathe05).

A seguir destacamos alguns fatores que influenciam o projeto físico de banco de dados (Sal04, Ramakrishnan02, Navathe05).

1. **Periodicidade e perfis da carga de trabalho:** Antes de empreender uma modificação no projeto físico do banco de dados, deve-se ter uma expectativa razoável acerca do uso pretendido para o banco de dados, ou seja, deve-se conhecer as características da carga de trabalho que será submetida ao banco de dados. Por exemplo, deve-se saber se a carga

de trabalho terá características OLAP (*On-Line Analytical Processing*) ou OLTP (*On-Line Transaction Processing*). Além disso, algumas cargas de trabalho tendem a possuir variações sazonais em seus comandos. Isto pode trazer a necessidade de criar diferentes índices para atender aos distintos perfis e sazonalidades da carga de trabalho.

2. **Freqüência e tipos de comandos executados:** Além da identificação das características da carga de trabalho esperada, deve-se obter uma aproximação da freqüência com que as consultas e atualizações que compõem esta carga de trabalho são executadas. Assim, há de se priorizar a melhoria de performance das cláusulas mais freqüentemente executadas. O processamento de cargas de trabalho que misturam atualizações e consultas, como é comum em sistemas de processamento *on-line* de transações (OLTP), dificulta as decisões sobre o projeto físico (quais índices devem ser criados, por exemplo). Para melhor uso dos recursos computacionais disponíveis, é necessário considerar se o custo de manutenção de cada índice será realmente compensado pelos benefícios que este traz em consultas.

3. **Restrição de tempo das consultas e transações:** Algumas consultas e transações podem possuir restrições rigorosas de desempenho. Por exemplo, uma transação pode ter a restrição de que deva terminar dentro de 5 segundos em 95% das vezes em que é executada e nunca deve demorar mais de 20 segundos. Tais restrições de desempenho determinam prioridades adicionais na definição das estruturas de acesso (índices e visões materializadas, por exemplo).

4. **Escolha de quais tabelas e colunas devem ser indexadas:** Antes mesmo de conseguir determinar se criar um índice é benéfico, é necessário saber quais índices podem ser criados. O número de índices possíveis sobre uma base de dados tende a crescer rapidamente, à medida que mais colunas e tabelas estão presentes.

5. **Determinação do momento para criação ou remoção dos índices:** Como a criação ou remoção de índices pode implicar em bloqueios, devemos ser cautelosos para que transações importantes não sejam prejudicadas por estas atividades.

6. **Obtenção da carga de trabalho:** Apesar de a carga de trabalho ser um insumo básico para se obter um projeto físico adequado, raramente é uma tarefa simples obtê-la.

2.2 Indexação de Dados

O método básico que os SGBDs (Sistemas Gerenciadores de Banco de Dados) utilizam para localizar registros (*tuplas*) consiste em varrer sistematicamente as tabelas em busca dos registros que satisfazem uma determinada chave (condição de seleção). Se uma consulta envolve um número muito grande de registros, o processo de busca pode apresentar desempenho indesejável. Os índices têm como principal característica acelerar o processamento de consultas submetidas ao banco de dados (Sal04).

As estruturas de índices fornecem um caminho eficiente para localizar e acessar os dados de maneira mais rápida, além de reduzirem a sobrecarga de busca a um pequeno conjunto de *tuplas*. Uma chave de busca de um índice é uma seqüência de atributos. O índice define um mapeamento entre os valores desta seqüência de atributos e seus registros correspondentes. Em sistemas relacionais, a estrutura mais comumente utilizada para representar índice são as árvores B^+. Uma árvore B^+ é uma árvore balanceada, cujas folhas contêm uma seqüência de pares chave-ponteiro, onde as chaves são ordenadas pelo seu valor. Podemos ver um exemplo deste tipo de estrutura na Figura 2.1.

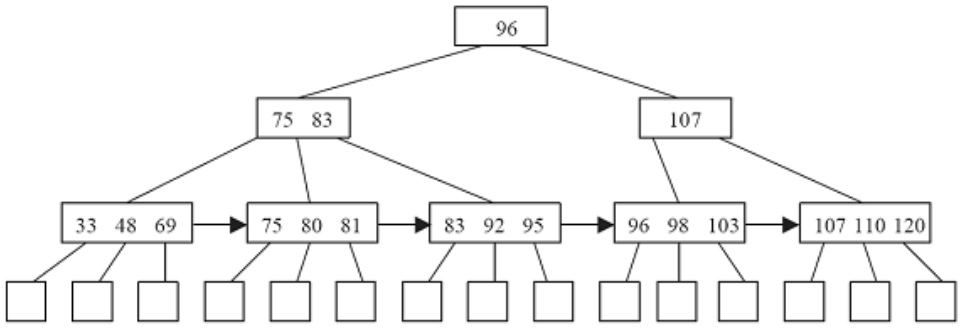

Figura 2.1: Exemplo de uma Árvore B+.

Basicamente, uma busca por meio de uma árvore começa pela raiz, chegando-se às folhas diretamente por comparação de chaves. Uma característica que torna as árvores B^+ interessantes para o uso em sistemas de bancos de dados é o armazenamento de diversas chaves por nó, criando uma árvore de grau elevado. Isto implica que a altura da árvore tende a ser pequena mesmo para um conjunto considerável de chaves. Como a obtenção de cada nó é feita por um acesso a disco, podemos consultar informações com uma determinada chave após alguns poucos acessos (Sal04).

Um ponto interessante a observar é que, em uma árvore B^+, as folhas são ordenadas pela chave do índice e ligadas por ponteiros. Isto permite que

consultas que acessam um intervalo de valores da chave sejam processadas de forma também eficiente. O acesso às *tuplas* da relação é feito seguindo os ponteiros associados a cada chave no nível folha. Os ponteiros das chaves no nível folha da árvore podem apontar tanto para os registros da tabela subjacente quanto para os blocos em que estes registros estão armazenados. Chamamos o índice de denso no primeiro caso e de esparso no segundo (Sal04).

Podemos ainda classificar os índices em primários e secundários. Um índice primário é aquele no qual a ordem dos registros na tabela subjacente é idêntica à ordem das chaves contidas nas folhas do índice. Já um índice secundário é aquele em que não há relação entre a ordem das chaves nas folhas do índice e dos registros armazenados na tabela. Consultas por intervalos que acessam informações da tabela são mais eficientes quando realizadas com um índice primário (Sal04).

Um índice primário pode ser denso ou esparso; já um índice secundário somente pode ser denso. Isto ocorre porque os índices esparsos não mantêm um ponteiro para cada registro da tabela e sim para cada página. A localização dos registros dentro das páginas da tabela é feita levando em consideração a ordem em que estes estão dispostos, que deve ser a mesma ordem das chaves do índice (Sal04).

Para exemplificar os benefícios trazidos pela utilização dos índices, consideremos a carga de trabalho ilustrada na Figura 2.2, a qual envolve dois comandos SQL definidos sobre uma tabela denominada Venda.

```
(1)
  insert into Venda (prodNum, data, qtd, valor)
  values (4, current_timestamp, 20, 348)

(2)
  select prodNum, data, sum(valor) as total
  from venda
  where valor > 1500000 and
    data between '20040101' and '20040131'
  group by prodNum, data
```

Figura 2.2: Exemplo de Carga de Trabalho.

Suponha, inicialmente, que os comandos são submetidos ao SGBD de forma concorrente e que temos uma freqüência de comandos do tipo (2) muito maior do que a de comandos do tipo (1). Neste tipo de cenário, um índice sobre as colunas de valor e data da tabela Venda pode trazer benefícios de desempenho no processamento das transações em que estes comandos estão inseridos. Como os comandos do tipo (1) são menos freqüentes, espera-se um

pequeno custo decorrente da atualização do índice sobre a tabela de vendas. Este custo de atualização pode ser amplamente compensado pelas vantagens de desempenho trazidas nas consultas, uma vez que o índice pode trazer grandes benefícios ao processarmos a restrição sobre valor e data (Sal04).

Por outro lado, se tivermos a situação inversa, com uma freqüência de comandos do tipo (1) muito superior à freqüência de comandos do tipo (2), a criação de índices sobre a tabela de vendas pode não ser recomendável. Neste cenário, podemos ter um custo significativo associado à atualização dos índices presentes sobre a tabela. A atualização dos índices decorre de comandos que realizam modificações sobre os dados da tabela, como inserções, remoções e atualizações que envolvem as colunas indexadas. Em cenários com intensa execução de operações de atualização, os benefícios trazidos pelos índices nas consultas podem não compensar os custos de manutenção envolvidos (Sal04).

Podemos ter ainda uma situação em que as transações envolvendo os comandos dos tipos (1) e (2) estão particionadas no tempo. Os comandos do tipo (1) podem ocorrer em uma carga noturna, enquanto as consultas do tipo (2) ocorrem durante o dia. Neste cenário, pode ser interessante remover os índices da tabela Venda antes da carga noturna e voltar a criá-los antes do período diurno de consultas (Sal04).

Como os índices tratados neste trabalho obedecem a organizações em árvore, um grande número de inserções, atualizações ou eliminações de chaves pode provocar a divisão de páginas (*page splits*). Este fato pode, dependendo da política de alocação de páginas utilizada pelo SGBD, causar fragmentação na estrutura de índice. Um índice fragmentado pode degradar o desempenho das operações de busca, principalmente operações de varredura (*scans*) (Morelli06b).

Por exemplo, suponha a existência de uma consulta sobre uma tabela que seja alvo de muitas inserções e atualizações (Figura 2.3):

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

Figura 2.3: Consulta que Realiza Cinco Varreduras Sobre a Tabela Venda.

A existência de um índice definido sobre o atributo num pode acelerar a execução da consulta ilustrada na Figura 2.3. O benefício trazido por

este índice, entretanto, depende do seu grau de fragmentação. A Figura 2.4 apresenta um índice que não passou por nenhum *page split*, enquanto que a Figura 2.5 exibe o mesmo índice após muitas atualizações (Morelli06a, Morelli06b).

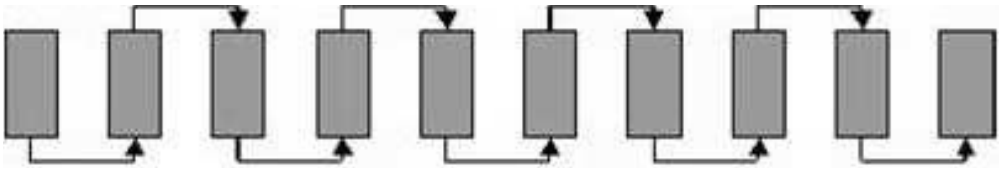

Figura 2.4: Índice Não Fragmentado

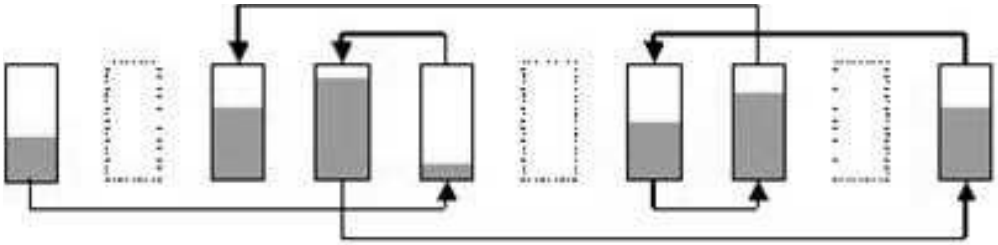

Figura 2.5: Índice Fragmentado

As setas apresentadas nas Figuras 2.4 e 2.5 revelam a ordem na qual o índice será percorrido. Deve-se perceber a ausência de "idas e voltas" em um índice apresentando grau de fragmentação zero. Já um índice fragmentado prejudica as operações de varredura, já que o artefato físico responsável pela leitura pode ser obrigado a realizar muitos deslocamentos físicos. Logicamente, uma varredura em índice sem fragmentação apresenta desempenho melhor que uma varredura executada sobre um índice fragmentado (Morelli06a).

Uma solução imediata para este problema seria a completa reconstrução de todos os índices de forma periódica e, de preferência, em instantes de pouca atividade. Esta estratégia, no entanto, traz dois graves inconvenientes, os quais também podem comprometer o desempenho do SGBD:

1. **Quantidade de Índices:** Os grandes bancos de dados possuem dezenas de milhares de índices. Reconstruir todos os índices existentes de uma vez, por mais "quieto" que esteja o Sistema, causa um esforço computacional considerável.

2. **Ausência de Atualizações:** Há índices, baseados em tabelas que não passam por atualizações. Reconstruí-los não traz benefício algum.

Uma abordagem mais sofisticada para solucionar este problema consiste em selecionar somente um subconjunto das estruturas de índices para reconstruir. Neste caso, busca-se selecionar as estruturas de índices que apresentem

um grau de fragmentação elevado, a ponto de comprometer o desempenho de uma varredura.

2.3 Duplicação de Estruturas Físicas

A duplicação de estruturas físicas, como tabelas, por exemplo, a fim de permitir duas ou mais ordenações físicas distintas, ainda é uma alternativa de projeto físico pouco explorada na literatura. Atualmente os SGBDs permitem que uma determinada tabela seja fisicamente ordenada (classificada) por um único critério. Esse critério determina a ordem com que os registros da tabela são fisicamente armazenados em disco. Em geral, o critério de ordenação utilizado coincide com a chave do índice primário definido sobre a tabela.

Desta forma, dada uma determinada tabela, por exemplo Venda, não é possível ordená-la por dois critérios diferentes, data da venda e valor da venda, por exemplo. Contudo, a ordenação física dos registros que compõem uma tabela influencia fortemente o desempenho de determinadas consultas.

Considere inicialmente a seguinte consulta:

```
SELECT data, count(*) as qtd
FROM Venda
GROUP BY data
```

Esta consulta apresentaria um tempo de execução menor se a tabela Venda estivesse fisicamente ordenada pelo atributo data, em comparação com a mesma tabela ordenada pelo atributo valor.

Considere agora uma segunda consulta:

```
SELECT valor, count(*) as qtd
FROM Venda
GROUP BY valor
```

Esta segunda consulta apresentaria um tempo de execução menor se a tabela Venda estivesse fisicamente ordenada pelo atributo valor, em comparação com a mesma tabela ordenada pelo atributo data.

Agora, suponha que estas duas consultas são igualmente importantes. Neste caso, o DBA necessita escolher que consulta deve priorizar, uma vez que a tabela Venda somente pode ser ordenada fisicamente por apenas um critério,

ou seja, a tabela Venda será fisicamente ordenada pelo atributo data ou pelo atributo valor.

Uma possível solução para este problema consiste na duplicação de estruturas físicas. Neste caso, poderíamos duplicar fisicamente a tabela Venda, uma "cópia" seria fisicamente ordenada pelo atributo data e outra pelo atributo valor. Esta estratégia tem por finalidade melhorar simultaneamente o desempenho das duas consultas discutidas. A duplicação de estruturas físicas é denominada de "*clusterização* alternativa de dados" e as "cópias" destas estruturas são chamadas "*clusters* alternativos de dados". Está claro que a duplicação física fará aumentar o volume dos dados, e, conseqüentemente, a demanda por espaço de armazenamento. No entanto, o custo dos dispositivos de armazenamento tem caído vertiginosamente, e, atualmente, não mais constitui fator determinante para a adoção dessa tecnologia. Além disso, a duplicação destas estruturas aumenta a disponibilidade dos dados.

2.4 Visões Materializadas

Uma visão é uma relação derivada, definida em termos de relações base, que é computada todas as vezes em que uma referência a ela é realizada. Uma visão é dita materializada quando esta é fisicamente armazenada na base de dados em vez de ser computada a partir das relações base em resposta a consultas, ou seja, uma visão materializada é uma consulta cujo resultado já está computado e armazenado na base de dados (Quass96).

Formalmente, (Gupta95) define uma visão como uma função que parte de um conjunto de tabelas base para uma tabela derivada, sendo que esta função é recalculada todas as vezes em que é referenciada. Uma visão pode ser materializada, por meio do armazenamento das *tuplas* da visão na base de dados. Com esta materialização, é possível criar índices que tornarão o acesso a estas visões materializadas muito mais rápido do que o recálculo, que é executado todas as vezes em que as visões são referenciadas.

Assim, uma visão materializada pode ser vista como um *cache* (área de armazenamento temporário), ou como uma cópia dos dados que pode ser acessada rapidamente. Por se comportar como um *cache*, uma visão materializada oferece acesso rápido aos dados, sendo que esta velocidade pode ser crítica em aplicações nas quais a quantidade de consultas é alta e a complexidade das visões elevada. Neste caso, o recálculo da visão a cada acesso se torna impraticável.

A desvantagem das visões materializadas reside no fato das alterações realizadas nos dados das tabelas base, a partir dos quais uma visão materi-

alizada é definida, tornarem a visão desatualizada. Para que a visão possa estar novamente sincronizada com os dados das tabelas base, será necessário recriar a visão a partir dos dados da origem, ou então atualizá-la de forma incremental (Quass96). Vários trabalhos descrevem algoritmos que permitem a manutenção incremental das visões materializadas, com diferentes domínios de aplicabilidade, como (Ceri91, Harrison92, Gupta93). Outro aspecto importante é que estas atualizações podem ser executadas imediatamente, tão logo a alteração é recebida, ou podem ser proteladas, de tal forma que todas as alterações ocorridas nos dados base serão reunidas e aplicadas em processos *batch*, que muitas vezes podem ser demorados.

Além disso, em decorrência das restrições de espaço e do custo de manutenção, a materialização de todas as visões possíveis torna-se inviável. Assim, surge o primeiro aspecto a ser analisado, que se refere à escolha das visões que serão materializadas. Esta escolha se baseia na determinação de um conjunto de visões, de uso compartilhado, de modo a combinar bom desempenho com baixo custo de manutenção. O objetivo é selecionar um conjunto apropriado de visões que minimize o tempo total de reposta da carga de trabalho (*workload*) submetida ao SGBD e o custo de manutenção das visões selecionadas, dado um certo limite de recursos, como, por exemplo, tempo de materialização, espaço para armazenamento, etc. Contudo, este também é um problema NP-Difícil (Agra00). Logo, a busca exaustiva torna-se uma solução inviável. Vários trabalhos tratam deste tema, como, por exemplo, (Agra00) e (Monteiro06b).

As abordagens existentes para resolver os problemas de seleção de visões podem ser agrupadas em quatro categorias, de acordo com (Zhang01): algoritmos determinísticos (Valluri02), algoritmos randômicos (Kalnis02), algoritmos evolucionários e algoritmos híbridos (Zhang01) que combinam características dos algoritmos anteriores. Normalmente, as soluções exploram exaustivamente o espaço de busca para enumerar e avaliar as soluções encontradas ou aplicam funções heurísticas, restringindo o espaço de busca para obter reduções nos custos de consultas e manutenção. Em (Bouzeghoub00) é apresentada uma solução que adota um conjunto de fatores de qualidade para o projeto físico de um *data warehouse*, consistente com a idéia de seleção de fontes de dados com base em múltiplos critérios de qualidade para materialização ou fornecer respostas a consultas de usuários.

2.5 Particionamento de Tabelas

O particionamento de dados é um método que consiste em dividir fisicamente as grandes tabelas em diversos segmentos menores de dados, tornando o acesso aos dados mais rápido e seu gerenciamento mais fácil (Papa06). Assim, o particionamento de tabelas e índices alivia o gerenciamento dos bancos de dados de grande porte, facilitando o gerenciamento em partes menores e mais manejáveis.

No modelo relacional, o particionamento de dados consiste na divisão de uma relação em segmentos (fragmentos) menores através de um processo de seleção de *tuplas* (particionamento horizontal), projeção de colunas (particionamento vertical) ou uma combinação de ambas (particionamento híbrido ou misto). Na fragmentação horizontal, cada partição contém um subconjunto das *tuplas* da relação completa (original); cada *tupla* da relação base precisa ser armazenada em pelo menos uma partição e a relação completa pode ser obtida fazendo a união das partições (Ramakrishnan02, Navathe05).

Já no particionamento vertical, as relações são decompostas em conjuntos de atributos mantidos em partições diferentes; cada partição é uma projeção da relação original e a relação completa pode ser obtida fazendo a junção de todas as partições. O particionamento híbrido ou misto consiste em uma combinação do particionamento horizontal e vertical (Ramakrishnan02, Navathe05).

Uma tabela particionada pode ter n partições, que funcionam como tabelas normais (não particionadas), sendo que cada partição pode ter sua própria configuração de *tablespace* e índice. A única diferença que podemos considerar entre uma tabela normal e uma tabela particionada é que a tabela particionada será sempre dependente estruturalmente da tabela básica, ou seja, a tabela que originou as partições, e a tabela normal funciona de forma independente de qualquer outra tabela (Ramakrishnan02, Navathe05).

Como exemplo, considere o comando SQL a seguir, o qual cria uma tabela denominada TABPART com três partições horizontais: SEMESTRE1, SEMESTRE2 e SEMESTRE3. Observe que a divisão das *tuplas* foi realizada utilizando-se o atributo col3.

Para recuperar os dados da partição SEMESTRE1, podemos utilizar:

```
SELECT * FROM TABPART
WHERE col3 < to_date(‘‘01-JUL-2006’’,‘‘DD-MON-YYYY’’);
```

ou

```
SELECT * FROM TABPART PARTITION (SEMESTRE1);
```

```
CREATE TABLE TABPART
(
col1 number(5) not null,
col2 varchar2(50) not null,
col3 date not null
)
tablespace TBS_1
partition by range (col3)
(partition SEMESTRE1 values less than (to_date("01-JUL-2006","DD-MON-YYYY")) tablespace TBS_1,
partition SEMESTRE2 values less than (to_date("01-JAN-2007","DD-MON-YYYY")) tablespace TBS_1,
partition SEMESTRE3 values less than (MAXVALUE) tablespace TBS_1);
```

Figura 2.6: Comando SQL para Criação de Tabela com Particionamento Horizontal.

Agora considere a seguinte consulta:

```
SELECT * FROM TABPART
WHERE col3 < to_date(''01-JAN-2008'',''DD-MON-YYYY'');
```

Observe que, neste caso, as três partições foram utilizadas para se recuperar os valores desejados.

Vale ressaltar também que, para recuperar todas as *tuplas* da relação TABPART, como mostra a consulta a seguir, será efetuada a união entre as partições.

```
SELECT * FROM TABPART;
```

2.6 Sintonia de Banco de Dados

O conceito de sintonia de bancos de dados refere-se à técnica de ajustar os parâmetros, configurações e estruturas de um sistema de bancos de dados às necessidades de uma determinada carga de trabalho (*workload*). Por carga de trabalho, entende-se um conjunto de tarefas (consultas ou atualizações) realizadas por unidade de tempo. Alguns autores, como (Ramalho02), defendem que, para a realização de sintonia, o DBA deve também considerar fatores externos ao SGBD, tais como o *hardware* e o *software*, antes de fazer alterações nas configurações do SGBD (Monteiro06a).

A tarefa de sintonia busca obter um melhor desempenho dos bancos de dados, fazendo com que as operações das aplicações sejam executadas em um menor tempo (tempo de resposta). Os SGBDs atuais possuem certo grau de

complexidade em razão dos milhares de parâmetros e configurações que podem ser ajustados. Tais ajustes requerem grande compreensão dos algoritmos utilizados pelo sistema, bem como nas inter-relações entre os ajustes. Na maioria dos SGBDs atuais, a sintonia é realizada de forma manual (Lif04a).

Desta maneira, o nível de conhecimento dos profissionais especializados na realização de sintonia de banco de dados é cada vez maior, sendo necessária uma quantidade crescente desses profissionais para suprir as necessidades atuais. E, à medida que os bancos de dados vão se tornando mais complexos, a tarefa de sintonia fica cada vez mais difícil de ser realizada pelo DBA e impraticável ao longo do tempo.

Todos esses fatores fizeram com que o meio acadêmico adotasse como objeto de pesquisa a realização de sintonia de banco de dados de forma automática, ou seja, de auto-sintonia. Este conceito visa diminuir a complexidade de administração dos SGBDs, bem como as tarefas manuais que o profissional DBA precisa realizar (Sal04).

2.7 Sintonia Automática de Banco de Dados

A relevância do estudo de auto-sintonia para a área de banco de dados aumentou significativamente nos últimos anos. Espera-se que, no futuro, os sistemas demandem uma quantidade menor de profissionais especializados para a manutenção de seu desempenho e de sua operação. Num cenário ideal, os sistemas conseguiriam alcançar um desempenho adequado sem qualquer necessidade de ajuste manual. Mais do que isto, à medida que as cargas de trabalho submetidas ao sistema mudassem, este iria procurar se adaptar dinamicamente para continuar apresentando um desempenho aceitável (Lif04a).

A auto-sintonia, bem como a atividade de sintonia de bancos de dados, deve respeitar alguns limites quanto ao tipo de ações que podem ser empreendidas para aumentar o desempenho do sistema. Assim, expandir a configuração de *hardware*, por meio da adição de processadores, memória e discos, ou mudar a versão do *software* do SGBD não constituem ações que estariam no escopo de um componente de auto-sintonia do sistema. Este tipo de componente deve criar controles sobre a forma como o sistema se utiliza do *hardware* disponível para acessar os dados (Lif04a).

Ações válidas incluem, portanto, a criação de estruturas de apoio para acelerar a busca de informações, a coleta de estatísticas sobre os dados armazenados, a alocação dos dados em disco e em memória, de forma a acelerar o seu acesso, e o controle de quantas transações serão atendidas pelo sistema e com qual tempo de resposta. Nos sistemas atuais, muitas destas decisões

exigem a intervenção de DBAs especializados. O sistema não é capaz de analisar automaticamente qual é a melhor forma de utilizar o *hardware* e o *software* em que foi instalado para processar a carga de trabalho que lhe é submetida (Lif04a).

Uma das grandes dificuldades da auto-sintonia de bancos de dados reside no fato de que a configuração de melhor desempenho para o sistema depende fundamentalmente do ambiente em que está inserido. A definição do ambiente varia de acordo com o foco de estudo. Para o sistema como um todo, o ambiente refere-se à combinação de *hardware*, carga de trabalho e outros programas executados no mesmo *hardware* (por exemplo, o sistema operacional). Já para um módulo específico do sistema, o ambiente pode ser definido pelos demais componentes do SGBD em contato com esse módulo e pelas requisições a que este módulo precisa atender (carga de trabalho) (Lif04a).

2.8 Sintonia Automática do Projeto Físico

Após o banco de dados ser implementado e entrar em operação, o uso real das aplicações, transações, consultas e visões revela fatores e áreas de problemas que podem não ter sido consideradas durante o projeto físico inicial. As considerações que direcionaram o projeto físico podem ser revisitadas por meio da coleta de estatísticas reais sobre os padrões de uso. O monitoramento dos recursos disponíveis pode revelar gargalos de desempenho. Os volumes das atividades e dos dados podem ser melhores estimados. Portanto, é necessário monitorar e revisar o projeto físico do banco de dados constantemente, atividade conhecida como sintonia do projeto físico de bancos de dados (Navathe05).

As mesmas orientações utilizadas no projeto físico (Seção 2.1) também podem ser utilizadas para guiar a sintonia do projeto físico. Contudo, o projeto físico é realizado uma única vez. Já a sintonia do projeto físico consiste em um ajuste continuado do projeto (Navathe05). Desta forma, as atividades que compõem a sintonia automática do projeto físico constituem um subconjunto das atividade de sintonia automática de banco de dados. Neste contexto, os objetivos da sintonia do projeto físico são:

- Diminuir o tempo de resposta das consultas/transações.
- Melhorar o desempenho geral das transações.

Assim, podemos entender a sintonia automática do projeto físico como um processo automático e contínuo de revisão do projeto físico. Este processo tem por objetivo manter um conjunto de estruturas de acesso (ou seja,

uma configuração de projeto físico) sempre adequado, a fim de maximizar o desempenho da carga de trabalho submetida ao SGBD. Estas estruturas de acesso podem incluir índices, visões materializadas, partições de grandes tabelas, etc. No entanto, o espaço disponível para a materialização destas estruturas é limitado. Desta forma, pode-se vislumbrar duas abordagens para a solução do problema de sintonia automática do projeto físico: abordagem conjunta e abordagem particular.

A abordagem conjunta busca selecionar estruturas de acesso de diferentes tipos (índices, visões materializadas, etc) de maneira integrada. Nesta abordagem, o problema em questão pode ser descrito como uma simplificação de um caso especial do problema da mochila (Graefe00, Loh00, Marques00). Mais precisamente, o problema da mochila compartimentada.

O "problema da mochila compartimentada" é uma variação do clássico "problema da mochila" e pode ser enunciado considerando-se a seguinte situação hipotética: um alpinista deve carregar sua mochila com possíveis itens de seu interesse. A cada item atribuem-se o seu peso e um valor de utilidade (até aqui, o problema coincide com o clássico Problema da Mochila). Entretanto, os itens são de agrupamentos distintos (alimentos, medicamentos, utensílios, etc.) e devem estar em compartimentos separados na mochila. Os compartimentos da mochila são flexíveis e têm capacidades limitadas. A inclusão de um compartimento tem um custo fixo que depende do agrupamento com que foi preenchido, além de introduzir uma perda da capacidade da mochila. O problema consiste em determinar as capacidades adequadas de cada compartimento e como esses devem ser carregados, maximizando o valor de utilidade total, descontado o custo de incluir compartimentos (Marques00).

Observe que, no problema da sintonia automática do projeto físico a capacidade da mochila seria a capacidade de armazenamento disponível para as estruturas de acesso que compõem o projeto físico; os itens são de agrupamentos distintos (índices, visões materializadas, partições de tabelas, etc); cada item tem um peso (tamanho do índice ou da visão materializada, por exemplo) e um valor de utilidade (o benefício da estrutura de acesso para a carga de trabalho).

Já a abordagem particular utiliza uma estratégia "dividir para conquistar", onde o problema maior, sintonia do projeto físico, passa a ser visto e enfrentado a partir de problemas menores: sintonia de índices, sintonia de visões materializadas, etc. Neste caso, solucionando-se cada sub-problema isoladamente, chegaríamos a uma solução para o problema global (sintonia do projeto físico como um todo). A solução para o problema de sintonia automática de um único tipo de estrutura em particular, índices, por exemplo, assemelha-se

ao problema da mochila booleana. Neste sentido, uma estratégia possível seria solucionar o problema da mochila booleana várias vezes (uma para cada tipo de estrutura em que estamos interessados), em vez de utilizar uma abordagem conjunta, baseada no problema da mochila compartimentada.

Antes de apresentar a abordagem proposta nesta tese para a manutenção automática do projeto físico de bancos de dados, algumas definições necessitam ser apresentadas.

Primeiramente, a seleção de uma configuração de projeto físico deve basear-se na carga de trabalho submetida ao sistema de banco de dados. Esta é uma diretriz que tem por objetivo assegurar que a configuração escolhida será efetivamente utilizada no futuro. Assim, um pressuposto básico é que a carga de trabalho utilizada no processo de seleção da configuração de projeto físico tenha as mesmas características da carga de trabalho que se espera no futuro. A seguir, definimos a expressão "carga de trabalho".

Definição 1 (Tarefa) *Uma tarefa é definida como uma tripla <expressão SQL, plano de execução, estimativa de custo>.*

Definição 2 (Carga de Trabalho (W)) *Uma carga de trabalho consiste em conjunto de tarefas.*

Agora, definiremos uma estrutura de acesso.

Definição 3 (Estrutura de Acesso) *Uma estrutura de acesso consiste em um mecanismo utilizado para acelerar o acesso aos dados.*

Exemplos de estruturas de acesso são: índices, visões materializadas, partições de grandes tabelas, réplicas de dados, estruturas físicas duplicadas, etc.

Neste trabalho, consideramos que uma estrutura de acesso pode estar em dois estados distintos: real e hipotético.

Definição 4 (Estrutura Real ou Materializada) *Uma estrutura real (ou materializada) é aquela que existe fisicamente, ou seja, ocupa espaço em disco e pode, efetivamente, ser utilizada no acesso aos dados (um índice real ou uma visão materializada real, por exemplo).*

Definição 5 (Estrutura Hipotética ou Virtual) *Uma estrutura hipotética (ou virtual) é aquela que existe apenas na metabase do SGBD, ou seja, não existe fisicamente. Logo, uma estrutura hipotética não ocupa espaço em disco e nem pode ser utilizada no acesso aos dados (um índice hipotético ou uma visão materializada hipotética, por exemplo).*

Definição 6 (Benefício) *Seja B_{e_j,q_k} o benefício proporcionado pela estrutura e_j para o processamento de uma tarefa q_k. Este benefício pode ser formalmente definido como:*

$$B_{e_j,q_k} = max\{0, cost(q_k) - cost(q_k, e_j)\},$$

onde, $cost(q_k)$ representa o custo de execução da tarefa q_k sem a utilização da estrutura e_j e $cost(q_k, e_j)$ representa o custo de execução da tarefa q_k utilizando-se a estrutura de acesso e_j.

Observe que se $cost(q_k) - cost(q_k, e_j) > 0$, a utilização da estrutura e_j contribui para aumentar o desempenho da tarefa q_k, logo: $B_{e_j,q_k} > 0$. Caso contrário, a utilização da estrutura e_j não proporciona nenhuma melhoria no desempenho da tarefa q_k, podendo inclusive, se fosse utilizada, reduzir este desempenho ($cost(q_k) - cost(q_k, e_j) < 0$). Contudo, neste caso, a estrutura e_j não seria utilizada pelo otimizador do SGBD, e, por conseguinte, $B_{e_j,q_k} = 0$.

Definição 7 (Benefício Acumulado) *O benefício acumulado de uma determinada estrutura de acesso $\mathsf{e_j}$, representado por (BA_{e_j}), corresponde à soma dos seus benefícios para cada uma das tarefas anteriormente executadas. Logo:*

$$BA_{e_j} = \sum_{k=1}^{NQ_{Wt}} B_{e_j,q_k},$$

onde, NQ_{Wt} é número total de tarefas da carga de trabalho.

Definição 8 (Benefício Relativo) *O benefício relativo de uma determinada estrutura de acesso $\mathsf{e_j}$, representado por (BR_{e_j}), corresponde ao seu benefício acumulado dividido pelo seu tamanho, ou seja pelo número de páginas em disco utilizadas para armazenar $\mathsf{e_j}$, denominado P_{e_j}. Logo:*

$$BR_{e_j} = \frac{BA_{e_j}}{P_{e_j}}$$

O benefício acumulado de uma determinada estrutura e corresponde à quantidade de páginas que deixariam de ser transferidas do disco para a memória principal caso a estrutura e existisse fisicamente durante o processamento da carga de trabalho submetida ao SGBD. Diferentes estruturas, entretanto, podem apresentar o mesmo benefício, mas ocupar tamanhos distintos

em disco. O benefício relativo é utilizado na tentativa de se obter uma medida de comparação mais justa, levando em consideração o espaço ocupado em disco pelas estruturas de acesso, uma vez que o espaço disponível para a materialização das estruturas selecionadas é limitado. Considere, por exemplo, duas estruturas e_1 e e_2. Assuma que $BA_{e_1} = 100$, $P_{e_1} = 10$, $BA_{e_2} = 60$, $P_{e_2} = 3$. Observando apenas o benefício acumulado, poderíamos ser induzidos a concluir que, para reduzir o tempo de processamento da carga de trabalho como um todo, a criação da estrutura e_1 seria uma opção melhor do que a criação da estrutura e_2. Porém, a estrutura e_1 ocupa um espaço em disco maior que a estrutura e_2 e o espaço disponível para a materialização das estruturas de acesso é limitado. Agora, se observarmos o benefício relativo, veremos que o benefício relativo de e_2 ($BR_{e_2} = 20$) é maior que o benefício relativo de e_1 ($BR_{e_1} = 10$). Desta forma, considerando que o espaço disponível é limitado, a estrutura e_2 se mostra mais indicada do que a estrutura e_1.

Definição 9 (Plano de Execução) *Um Plano de Execução ou Plano de Consulta consiste em uma árvore formada por nós e arestas na qual:*

- *existe um nó especial,* r*, chamado raiz do plano;*
- *os demais nós constituem um conjunto vazio ou são particionados em conjuntos disjuntos não vazios, os sub-planos (ou sub-árvores) de* r*. As raízes desses sub-planos são ligadas diretamente a* r*. Esses nós raízes das sub-árvores são ditos filhos do nó pai,* r*;*
- *cada sub-plano, por sua vez, também é um plano;*
- *uma aresta consiste em par de nós (*u, v*) onde* u *é pai de* v*, representando assim uma relação hierárquica entre os nós;*
- *nós que possuem filhos são chamados de nós internos e nós que não têm filhos são chamados de folhas;*
- *as folhas representam as relações de entrada de uma consulta;*
- *os nós internos representam as operações da álgebra relacional, anotadas com instruções especificando como avaliar cada operação. As anotações podem indicar o algoritmo a ser usado para uma operação específica, o método de acesso, ou um índice a ser utilizado.*
- *as estruturas utilizadas nas anotações são sempre reais (fisicamente existentes).*

A seguir, definimos formalmente a equivalência entre dois planos de execução distintos:

Definição 10 (Planos de Execução Equivalentes) *Dois planos de consulta p e $p^{'}$ são considerados equivalentes (ou plano-equivalentes, $p \cong p^{'}$) se:*

- $p = p^{'}$

  ou

- $p \neq p^{'}$ *e as seguintes condições são satisfeitas:*

  1. p *e* $p^{'}$ *produzem a mesma relação de resultado;*
  2. *possuem topologias distintas (ou seja,* $\exists$ *um nó* n *tal que:* $n \in p$ *e* $n \notin p^{'}$*, ou* $n \in p^{'}$ *e* $n \notin p$*).*

Definição 11 (Plano de Execução Hipotético) *Um Plano de Execução Hipotético (ou Plano de Consulta Hipotético) consiste basicamente em plano de execução (Definição 9), com as seguintes diferenças:*

- *as estruturas utilizadas nas anotações podem ser hipotéticas (virtuais), como índices hipotéticos, por exemplo;*
- *foi gerado a partir da manipulação de um plano de execução real (ou seja, plano de execução gerado através do processamento de uma consulta* q *submetida ao SGBD).*

Definição 12 (Otimização Hipotética) *Otimização hipotética consiste na manipulação de um plano de execução real (ou seja, plano de execução gerado através do processamento de uma consulta* q *submetida ao SGBD), através da utilização de estruturas hipotéticas (índices hipotéticos, por exemplo), buscando gerar um plano de execução hipotético com custo de execução menor que o plano de execução original (real).*

Definição 13 (Conjunto das Estruturas Reais (ER)) *ER representa o conjunto de todas as estruturas reais existentes no SGBD em um determinado instante de tempo.*

Definição 14 (Conjunto das Estruturas Hipotéticas (EH)) *EH representa o conjunto de todas as estruturas hipotéticas existentes na metabase do SGBD em um determinado instante de tempo.*

Definição 15 (Conjunto das Estruturas de Acesso (E)) *E representa o conjunto de todas as estruturas, reais e hipotéticas, existentes no SGBD em um determinado instante de tempo. Logo, $E = ER \cup EH$.*

Proposição 1 *Seja ER o conjunto das estruturas reais e EH o conjunto das estruturas hipotéticas, em um determinado instante de tempo t. Então, $ER \cap EH = \phi$.*

O resultado de uma solução para o problema da sintonia automática do projeto físico é uma configuração de estruturas de acesso, ou configuração de projeto físico.

Definição 16 (Configuração de Estruturas de Acesso (C)) *Uma configuração C consiste um determinado conjunto de estruturas de acesso, reais ou hipotéticas, em um determinado instante de tempo, tal que:*

se uma determinada estrutura $e \in C$, então: $e \in ER$ ou $e \in EH$. Logo, $C \subseteq E$.

Definição 17 (Benefício Total de uma Configuração de Estruturas de Acesso (C)) *O benefício total de uma configuração C, representado por B_C, consiste na soma dos benefícios acumulados das estruturas de acesso que compõem a configuração C. Assim:*

$$B_C = \sum_{j=1}^{m} BA_{e_j},$$

onde m representa o número de estruturas de acesso pertencentes à configuração C.

Agora, pode-se definir formalmente o problema da seleção de uma configuração de estruturas de acesso.

Definição 18 (Problema da Seleção de uma Configuração de Estruturas de Acesso) *Dado um conjunto de consultas $q_1, \cdots, q_m$ e um conjunto de estruturas de acesso $e_1, \cdots, e_n$, onde cada estrutura e_j está associada a custo de gerenciamento, $mcost(e_j)$, um tamanho (quantidade de páginas de disco ocupadas pela estrutura e_j), P_{e_j}, e um benefício acumulado, BA_{e_j},o problema da seleção de uma configuração de estruturas de acesso consiste em encontrar uma configuração $C \subseteq e_1, \cdots, e_n$ de estruturas de acesso que otimize o tempo de execução total da carga de trabalho submetida ao SGBD através da maximização do benefício total, sujeito à uma restrição de espaço em disco S, tal que:*

$$\sum_{e_j \in C} P_{e_j} \leq S.$$

Além disso, existe uma restrição de acesso L_k para cada tipo diferente de estrutura de acesso (por exemplo, índices, visões materializadas, etc). Assim, seja m a quantidade de tipos distintos de estruturas de acesso e $M = 1, \cdots, m$ o conjunto destes tipos de estruturas. Assuma que existe um agrupamento (compartimento), denominado A_k, para cada tipo de estrutura de acesso, onde $A_k \subseteq C$. Neste caso, teremos m agrupamentos distintos ($A_1, \cdots, A_m$). Seja L_k a capacidade de armazenamento do agrupamento A_k, temos que:

$$\sum_{e_j \in A_k} P_{e_j} \leq L_k$$

e

$$\sum_{k=1}^{m} L_k \leq S.$$

Definição 19 (Problema da Sintonia Automática do Projeto Físico) *O conceito de sintonia automática do projeto físico refere-se a solucionar continuamente (ou periodicamente) o problema da seleção de uma configuração de estruturas de acesso.*

2.9 Benchmark

A avaliação de desempenho está presente em todos os momentos do ciclo de vida de um sistema computacional. Na hora de projetar, produzir ou implantar um sistema, o objetivo final é sempre o mesmo: escolher, dentre diversas opções, aquela que proporcione o melhor desempenho, com o menor custo possível. Não existe, porém, um meio universal com o qual possamos avaliar o desempenho dos diversos sistemas computacionais. Em razão da diversidade de aplicações, é necessário definir técnicas e métricas para cada caso.

Neste sentido, o termo *Benchmarking* representa um processo sistemático e contínuo de avaliação dos produtos, serviços e processos de trabalho de organizações que são reconhecidas como representantes das melhores práticas, com a finalidade de introduzir melhorias na organização; as cargas usadas nesse processo são chamadas de *benchmark*.

Benchmarks são projetados para testar características particulares dos sistemas, como, por exemplo, o poder de processamento ou o desempenho gráfico, ou ainda a capacidade do subsistema de E/S. Os *benchmarks* costumam ser classificados em sintéticos e de aplicação. *Benchmarks* sintéticos são programas que executam um conjunto reduzido de instruções, testando componentes muito específicos de um sistema. Já os *benchmarks* de aplicação simulam um ambiente real, dando uma melhor noção do desempenho do sistema como um todo. O trabalho realizado em (Vieira05) considera algumas características que os *benchmarks* devem possuir.

1. **Representatividade:** os *benchmarks* devem representar sistemas reais, ou seja, não devem ser aplicados em sistemas simulados.

2. **Portabilidade:** os *benchmarks* devem ser portáveis para diferentes plataformas, proporcionando, assim, comparativos de desempenhos entre diversos distribuidores de sistemas.

3. **Repetibilidade:** quando um *benchmark* é aplicado no mesmo ambiente, mais de uma vez, ele deve produzir resultados semelhantes.

4. **Escalabilidade:** essa característica prevê que os *benchmarks* realizem avaliações em ambientes com diferentes capacidades.

5. **Não Intrusividade:** na necessidade de avaliar outro ambiente deve-se realizar o mínimo ou nenhuma alteração nesse novo ambiente.

6. **Simplicidade:** os *benchmarks* devem ser de fácil implementação e utilização pelos usuários que farão as avaliações.

O nascimento do *benchmarking* se deu no começo da década de 1980. Naquela época, os processadores eram os componentes mais caros dos sistemas, de maneira que se costumava denotar o desempenho do sistema como o desempenho do próprio processador. Foram desenvolvidos, então, os primeiros programas sintéticos, com o intuito de testar a capacidade de processamento dos sistemas. Exemplos da primeira geração de *benchmarks* incluem o *Dhrystone* (desempenho aritmético com inteiros), *Whetstone* (desempenho aritmético de ponto flutuante) e *Linpack* (operações com sistemas de equações lineares).

Com o passar dos anos, no entanto, ficou claro que não era suficiente medir o desempenho somente do processador. Além do mais, era comum os fabricantes realizarem otimizações nos compiladores específicos para os conjuntos de instruções desses *benchmarks*, o que gerava resultados distantes dos obtidos em aplicações do mundo real. Essa falta de credibilidade ocasionou o

surgimento das primeiras organizações de *benchmark*, entidades neutras, sem fins lucrativos, que normalmente têm como membros os grandes fabricantes de *hardware* e *software*. Dentre elas, as mais conhecidas são a SPEC (*Standard Performance Evaluation Corporation*), e a TPC (*Transaction Processing Performance Council*), criado em 1988. A SPEC é responsável por diversos *benchmarks*, voltados para diversas áreas, dentre elas desempenho de CPU, de servidor *Web*, de servidor de banco de dados, de aplicações cientificas, etc. Já a TPC mantém *benchmarks* voltados ao desempenho de sistemas de bancos de dados. Essas organizações ajudaram a recuperar a credibilidade dos *benchmarks*, visto que a publicação de resultados é sujeita a regras rígidas, passando, inclusive, por procedimentos de auditoria.

O TPC tem por objetivo estabelecer critérios de performance de processamento de transações e de bancos de dados por meio de *benchmarks*, ou testes para estabelecer padrões de referência (tais como o TPC-C, o TPC-W e o TPC-H), e divulgar os dados reais dessa performance com base nos testes realizados. Os *benchmarks* TPC são submetidos a exigências extremamente rigorosas, principalmente nos quesitos confiabilidade e durabilidade e são sempre aplicados perante uma auditoria independente. Os membros desse Conselho incluem as principais empresas de bancos de dados e fornecedores de sistemas de *hardware* do mercado - HP, IBM, Oracle, Microsoft, Unisys, Sun, Intel, AMD, Dell, Fujitsu, NEC, Teradata, Novell, Sybase, Bull, Netezza. As empresas participam dos *benchmarks* TPC para demonstrar, objetivamente, a performance de seus sistemas em um ambiente controlado, e para aplicar as tecnologias utilizadas durante o processo de testes à criação de produtos de *hardware* e de *software* ainda mais robustos e escaláveis. Porém, o TPC não disponibiliza um "*toolkit*" (*kit* de ferramentas) que possibilite ou facilite os testes de performance.

Já a organização OSDL (*Open Source Development Labs*) foi fundada em 2000 e mantém um conjunto de testes de desempenho para avaliar o comportamento de soluções de *software* livre, em especial, o sistema operacional Linux. Na verdade, ela fornece o "estado da arte" em computação e ambientes de teste para acelerar o crescimento e adoção do S.O. Linux nas empresas. Existem testes voltados especificamente para sistemas de bancos de dados e inspirados nos *benchmarks* do TPC, que são os *toolkits* DBT-1 (*Database Test 1*), DBT-2 (*Database Test 2*), DBT-3 (*Database Test 3*) e DBT-4 (*Database Test 4*). A OSDL recebe investimentos de grandes empresas como Fujitsu, HP, IBM e Intel. Todavia, estes *toolkits* somente podem ser utilizados com os SGBDs PostgreSQL e MySQL, sob o sistema operacional Linux.

Nesta tese, utilizamos os *benchmarks* TPC-C (Apêndice D) e TPC-H

(Apêndice E). O *benchmark* TPC-C é uma carga de trabalho composta por um conjunto de transações de leitura e escrita, que simula as atividades encontradas em um ambiente OLTP (*On-Line Transaction Processing*) complexo. O propósito do TPC-C é prover dados relevantes de desempenho que auxiliem os usuários a comparar de maneira objetiva dois ou mais sistemas quanto a sua capacidade de processar transações. Já o TPC-H é um *benchmark* de suporte a decisões que consiste em um conjunto de consultas *ad-hoc* voltadas para ambientes de *Data Warehouse* (OLAP - *On-Line Analytical Processing*).

2.10 Resumo do Capítulo

Este capítulo apresentou os conceitos básicos sobre indexação de dados, visões materializadas, particionamento de tabelas, projeto físico de bancos de dados, sintonia e auto-sintonia de banco de dados (definições, arquiteturas e características) e *benchmarks*. Estas definições serão utilizadas no restante deste trabalho, principalmente na discussão acerca das principais contribuições desta tese, como a arquitetura e a abordagem não-intrusiva para manutenção automática do projeto físico.

No próximo capítulo, serão examinadas as principais abordagens encontradas na literatura para o projeto físico automático de bancos de dados. Uma análise comparativa detalhada destas abordagens também será apresentada. Além disso, uma nova classificação para as pesquisas em auto-sintonia de bancos de dados será discutida.