# 7
# Resultados Experimentais

A partir da abordagem proposta neste trabalho implementamos, em linguagem Java, uma ferramenta não intrusiva para a manutenção automática de índices, a qual fornece suporte para os SGBDs PostgreSQL, Oracle 10g e SQL Server 2005. A ferramenta, no entanto, pode ser facilmente estendida para outros SGBDs, uma vez que todas as informações necessárias são extraídas da metabase do SGBD utilizado (Monteiro08a). Nossa ferramenta monitora periodicamente a carga de trabalho submetida ao SGBD, capturando esta carga (triplas <*cláusula SQL, plano de execução, custo*>) e armazenando-a em uma metabase local. Periodicamente, a ferramenta analisa as informações capturadas e de forma autonômica realiza os ajustes necessários na configuração dos índices.

A fim de mensurar a eficácia de nossa abordagem, utilizamos o *benchmark* TPC-H, o qual consiste em um *benchmark* voltado para aplicações de suporte a decisão. O TPC-H é formado por um conjunto de 23 consultas *ad-hoc* e possui uma estrutura padrão composta por oito tabelas[1]. Destas, seis são tabelas dimensionais (Region, Nation, Supplier, Part, Customer e Partsupp) e duas de fatos (Orders e Lineitem). Dentre as 23 consultas do TPC-H escolhemos a 6 mais significativas, ou seja, aquelas que apresentam tempos de resposta maiores, justificando a utilização das estruturas de índices (Morelli06a). Para a realização dos testes foi utilizada uma base de dados de 1G, onde a tabela Liniitem tem 6.000.000 de *tuplas*, por exemplo. O ambiente de execução utilizado foi uma estação Athlon 64 x2 de 2Ghz, com 2G ram e 250G de Hd, com SO Windows XP. O SGBD utilizado nos teste foi o PostgreSQL 8.3.

A carga de trabalho utilizada foi obtida através da utilização do *toolkit Database Test 3* (DBT-3), o qual é mantido pela organização *Open Source Development Labs* (OSDL). Este *toolkit* simula uma carga de trabalho baseado no *benchmark* TPC-H (TPC-H), que favorece a incidência de consultas onerosas, simulando um ambiente com características OLAP (*On Line Analytical Processing*). A utilização do *toolkit* DBT3 proporcionou a criação da base de dados segundo as diretrizes do TPC-H, com *scripts* das consultas adaptadas à sintaxe

[1] O Apêndice E explica em detalhes a constituição do *benchmark* TPC-H.

do PostgreSQL, bem como viabilizou a geração grupos de consultas a serem executadas.

Após a execução dos primeiros testes, foi constatada a competência do Agente AIM na criação dos índices necessários. Entretanto, a duração das consultas era extremamente alta, por conseguinte, o tempo necessário para uma única sessão de testes (executar cada uma das 22 consultas do *benchmark*, por exemplo) revelou-se bastante longo, o que dificultava sobremaneira a realização dos testes.

A fim de reduzir o tempo das sessões de teste, utilizamos um parâmetro do PostgreSQL denominado statement_timeout, o qual define um intervalo de tempo (medido em milisegundos) máximo para a execução de qualquer comando SQL. Assim, expressões SQL de longa duração teriam suas execuções interrompidas, após excederem o limite de tempo definido pelo parâmetro statement_timeout. Este artifício foi utilizado para reduzir a duração das sessões dos testes realizados inicialmente.



## 7.1 Trabalhos Relacionados com Avaliação TPC-H

Antes de elaborar a bateria de testes, buscou-se em trabalhos relacionados como se utiliza o benchmark TPC-H na prática e no contexto de sintonia automática de índices. Em (Morelli06a), o autor apresenta uma visão geral destes trabalhos e conclui que:

- Não há menção explícita aos testes Power ou *Throughput*, por exemplo. Simplesmente aproveita-se a base de dados e as consultas de algum *benchmark*.
- Nada indica que houve escolhas aleatórias para ordem de execuções das consultas, bem como para valores utilizados em filtros.

Desta forma, as observações realizadas em (Morelli06a) avalizam o mecanismo escolhido para testar a abordagem proposta nesta tese.

## 7.2 Análise da Seletividade das Consultas

A seletividade de uma consulta SQL refere-se ao número de *tuplas* recuperadas pela execução desta consulta. Quanto maior o número de *tuplas* retornadas, menor a seletividade da consulta. Os otimizadores de SGBDs levam este fato em conta ao avaliarem a utilizações de índices. Por exemplo, suponha uma tabela T possuindo um milhão de *tuplas*, um determinado campo denominado UF e um índice sobre este atributo. Considere também que 999.999

*tuplas* possuem UF = 'RJ'. Neste caso, o índice somente seria utilizado quando fosse solicitada a única *tupla* com valor diferente de 'RJ', pois caso contrário, o otimizador irá escolher realizar uma varredura na tabela T (*Seq Scan*).

Desta forma, a seletividade das consultas é um importante parâmetro a ser considerado pelo agente AIM, a fim de que não sejam criados índices que não venham a ser aproveitados. Observando-se o comportamento deste agente, quando submetido a uma carga de trabalho TPC-H, percebemos três fatos:

- Em consultas com baixa seletividade, não se criavam índices; nem mesmo hipotéticos;
- Em consultas com alta seletividade, índices eram criados nas primeiras execuções;
- Em consultas com seletividade média, índices hipotéticos eram criados e, após algumas execuções, aconteciam as transformações de hipotéticos para reais.

Portanto, o agente não intrusivo apresentou um comportamento similar ao apresentado pelo agente de benefícios (intrusivo) apresentado em (Morelli06a) e à atuação dos DBAs, onde consultas com seletividade média são executadas várias vezes para que se tenha certeza da necessidade real de se criar um determinado índice.

Uma diferença marcante entre as cargas de trabalho geradas a partir dos *benchmarks* TPC-C e TPC-H, reside no fator previsibilidade. Se por um lado os comandos TPC-C não sofrem variações em seus filtros (cláusula *where* dos comandos *select*, *delete* e *update*), nem na ordem de execução dos comandos, isto não ocorre com instruções TPC-H. Este fato faz com que a seletividade das consultas varie entre execuções, levando o otimizador a nem sempre utilizar índices para resolvê-las (Morelli06a). Assim, duas execuções de uma mesma consulta podem utilizar ou não um determinado índice, dependendo da seletividade gerada pelo *toolkit* em cada uma das duas execuções.

Desta forma, ao submeter as consultas do TPC-H ao agente AIM, percebeu-se que a quantidade de execuções necessárias para que um índice fosse criado, variava bastante, pois dependia da seletividade, a qual era gerada aleatoriamente. Explica-se este fato pela atuação do agente AIM, que atribui mais, ou menos, benefícios a um índice hipotético, dependendo da seletividade apresentada pela consulta, na qual houve participação deste índice.

Dada a grande variação entre as execuções de consultas TPC-H, resolveu-se escolher seis consultas mais representativas e ajustá-las, fixando a seletividade, a fim de que a atuação do agente AIM ficasse evidenciada e que a interpretação dos resultados dos testes fosse facilitada. Essa mesma estratégia

foi utilizada nos testes realizados em (Morelli06a). A relação das consultas escolhidas aparece no Apêndice E.

## 7.3 Resultados Experimentais com o PostgreSQL

A obtenção de resultados experimentais realizada (Sal04), caracterizou-se pela execução de cargas de trabalho durante três intervalos: 30, 60 e 90 minutos. Para avaliar a eficácia do agente AIM, utilizamos a mesma estratégia empregada em (Morelli06a), onde um conjunto de consultas representativas foi executado, independentemente do fator tempo.

Como medida da eficiência de nossa solução, utilizamos como referência o conjunto dos índices definidos para o *benchmark* TPC-H (Figura 7.1). Assim, quanto mais próximo deste grupo estiver a coleção de índices mantida pela abordagem proposta, maior será sua eficiência da solução proposta.



| Consulta | Tabela | Campos | Chave |
|---|---|---|---|
| 1 | Lineitem | L_shipdate | |
| 2 | Partsupp | Ps_partkey | Estrangeira |
| | Supplier | S_suppkey | Primária |
| 4 | Orders | O_orderdate | |
| | Lineitem | L_orderkey | Estrangeira |
| 10 | Orders | O_orderdate | |
| | Lineitem | L_orderkey | Estrangeira |
| | Customer | C_custkey | Primária |
| 17 | Lineitem | L_partkey | Estrangeira |
| 19 | Part | P_partkey | Primária |

Figura 7.1: Principais consultas e os índices sugeridos para o *benchmark* TPC-H.

Cada uma das seis consultas escolhidas foi executada em um ambiente desprovido de índices, ou seja, antes de cada execução de uma determinada consulta, eliminamos os índices que, por ventura, existissem. A Figura 7.2 apresenta os índices criados automaticamente pela nossa implementação e a quantidade de execuções do comando SQL necessárias para que nossa ferramenta decidisse pela criação do índice.

O índice a mais, criado por nossa abordagem (sobre a tabela Part, consulta 17) e não sugerido pelo *benchmark* TPC-H, proporcionou ganhos expressivos. O custo caiu 62,62% e a sua duração obteve um ganho da ordem de 61,45%. Este valor foi obtido executando-se a consula 17 primeiramente sem e depois com a presença do referido índice.

Desta forma, verificamos a eficácia da abordagem proposta em manter todos os índices definidos pelo *benchmark* TPC-H.

| Consulta | Tabela | Campos | Execuções |
|---|---|---|---|
| 1 | Lineitem | L_shipdate | 5 |
| 2 | Partsupp | Ps_partkey | 1 |
| | Supplier | S_suppkey | 1 |
| 4 | Orders | O_orderdate | 1 |
| | Lineitem | L_orderkey | 2 |
| 10 | Orders | O_orderdate | 2 |
| | Lineitem | L_orderkey | 2 |
| | Customer | C_custkey | 1 |
| 17 | Lineitem | L_partkey | 5 |
| | Part | P_brand, P_container, P_partkey | 1 |
| 19 | Part | P_partkey | 3 |

Figura 7.2: Resultados dos Testes Realizados.



### 7.3.1 Análise da Pré-Criação de Índices Envolvidos em Chaves Primárias e Estrangeiras

O *toolkit* DBT-3 sugere a presença de 23 índices, dos quais apenas cinco não estão envolvidos com chaves primárias ou estrangeiras. Logo, uma ação bastante simples, porém efetiva, para se chegar a configurações estáveis mais rapidamente consiste na pré-criação dos índices envolvidos com chaves primárias e estrangeiras. Neste caso, 78,2% dos índices indicados já estariam criados antes mesmo do agente AIM iniciar sua execução. Assim, a ferramenta apresentada em (Monteiro08a) permite a pré-criação destes índices. Através da metabase do SGBD utilizado, a ferramenta acessa o esquema de dados, extrai os campos envolvidos em chaves e executa a criação automática dos índices envolvidos em chaves primárias e estrangeiras, sem, no entanto, precisar coletar comandos que acusassem a necessidade óbvia de índices em tais campos.

### 7.3.2 Análise das Operações de *Seq Scan*

A principal estratégia utilizada para identificar os índices que poderiam melhorar o desemepenho de uma determina expressão SQL consiste em analisar o plano de execução, produzido pelo otimizador, e verificar a existência de varreduras (*Seq Scan*), uma vez que o custo dessa operação domina o custo total de um plano de execução. Vale ressaltar ainda que, praticamente, toda busca seqüencial poderia ser substituída por uma busca baseada em índice (*Index Scan*), caso um índice adequado exista. Neste sentido, decidimos analisar o relacionamento entre os índices sugeridos pelo *benchmark* TPC-H e as operações de busca seqüencial (varreduras ou *Seq Scan*). A Tabela 7.1 ilustra

| Consulta | Tabela | Campos | Envolvido em *Scan* | Cláusula SQL |
|---|---|---|---|---|
| 1 | Lineitem | L_shipdate | Seq Scan on Lineitem | Where |
| 2 | Partsupp | Ps_partkey | Seq Scan on Partsupp | Join |
| 2 | Supplier | S_suppkey | Seq Scan on Supplier | Join |
| 4 | Orders | O_orderdate | Seq Scan on Orders | Where |
| 4 | Lineitem | L_orderkey | Seq Scan on Lineitem | Join |
| 10 | Orders | O_orderdate | Seq Scan on Orders | Where |
| 10 | Lineitem | L_orderkey | Seq Scan on Lineitem | Join |
| 10 | Customer | C_custkey | Seq Scan on Customer | Join |
| 17 | Lineitem | L_partkey | Seq Scan on Lineitem | Join |
| 19 | Part | P_partkey | Seq Scan on Part | Join |

Tabela 7.1: Relacionamento entre os índices sugeridos pelo *benchmark* TPC-H e as operações de busca seqüencial (*Seq Scan*).



esse relacionamento. Observe que, para as seis consultas selecionadas, todos os índices sugeridos pelo *benchmark* estão relacionadas com operações de busca seqüencial.

### 7.3.3 Anásile da Ordem de Execução das Consultas

Em (Morelli06a), observou-se que as Consultas 10 e 4 compartilham dois índices, e destacou-se a necessidade de observar o que aconteceria caso índices previamente criados não fossem destruídos antes da execução das consultas. Neste trabalho, observou-se que:

- Executando a Consulta 10 após a 4, ou seja, aproveitando o fato do índice sobre o campo l_orderdate já ter sido criado, o índice sobre o campo c_custkey foi construído na primeira execução, porém não fez menção a l_orderkey.
- Executando a Consulta 4 após a 10, ou seja, aproveitando os três índices previamente criados, nenhum índice adicional foi sugerido, já que os índices necessários à Consulta 4 já tinham sido criados.

Assim, os autores concluíram que, a ordem da execução das consultas que possuem potenciais índices em comum influenciam o resultado final obtido, ou seja, o conjunto de índices que pertenceriam à configuração selecionada pelo agente. Este fato também foi observado em (Agra06), onde determina-se previamente a ordem de execução ótima para uma carga de trabalho, de forma a minimizar o custo de execução da carga de trabalho como um todo.

Uma vez verificada a capacidade do agente AIM em criar os mesmos índices propostos pelo *benchmark* TPC-H, achou-se interessante comparar o

conjunto de índices obtidos aos que seriam propostos por ferramentas de apoio disponibilizadas pelos principais SGBDs comercias, mais precisamente, Microsoft SQL Server 2005 e Oracle 10g. As conclusões obtidas aparecem nas próximas duas subseções.

### 7.3.4 Resultados Experimentais com SQL Server 2005

A segunda bateria de testes utilizou a ferramenta *Database Tuning Advisor* (DTA) e o SGBD Microsoft SQL Server 2005, na mesma estação descrita no início deste capítulo. As mesmas seis consultas escolhidas foram submetidas ao DTA. Criaram-se as mesmas tabelas, sem índices, os mesmos dados foram carregados e submeteu-se cada Consulta em separado. Não houve a necessidade de reiniciar o ambiente antes de cada execução, pois a ferramenta DTA apenas sugere a criação de índices, ficando a cargo do DBA aceitar, ou não, as recomendações. A Tabela 7.2 exibe, para cada consulta, os índices sugeridos pelo DTA.

| Consulta | Tabela | Campos |
|---|---|---|
| 1 | Lineitem | L_shipdate, L_returnflag, L_linestatus |
| 2 | Partsupp | Ps_partkey, Ps_suppkey, Ps_supplycost |
| 2 | Part | P_partkey |
| 4 | Orders | O_orderdate, O_orderkey, O_orderpriority |
| 4 | Lineitem | L_orderkey, L_commitDATE, L_receiptDATE |
| 10 | Orders | O_orderdate, O_custkey, O_orderkey |
| 10 | Lineitem | L_orderkey, L_returnflag |
| 10 | Customer | C_custkey, C_name, C_acctbal, C_phone, C_address, C_comment |
| 17 | Lineitem | L_partkey |
| 19 | Part | P_brand, P_container, P_size, P_partkey |
| 19 | Lineitem | L_shipinstruct, L_partkey, L_shipmode, L_quantity |

Tabela 7.2: Índices sugeridos pela ferramenta DTA do SQL Server 2005.

Deve-se ressaltar que não faria sentido executar cada consulta mais de uma vez, pois a análise não se baseia na atribuição gradativa de benefícios. Acontece uma análise do comando e das estatísticas presentes, para depois concluir as recomendações.

Apesar de constar um índice a mais (tabela Part, Consulta 19), um fato que chamou a atenção foi a não recomendação, na Consulta 17, do índice sobre a tabela Part (campos P_brand, P_container, P_partkey). De fato, uma vez criado, o Otimizador não somente aproveitou o índice, como também o registrou-se uma queda notável no custo da consulta, na orderm de 69,31%.

Vale ressaltar que não consideramos necessário executar a solução não intrusiva (agente AIM) sobre a base TPC-H construída no SQL Server, pois os índices sugeridos não seriam distintos daqueles criados durante sua execução com o PostgreSQL, uma vez que os que o calculo dos benefícios e a seletividade das consultas não seriam alterados.

### 7.3.5 Resultados Experimentais com Oracle 10g

A terceira bateria de testes utilizou o SGBD Oracle 10g. Assim como realizado com SQL Server, as mesmas seis consultas foram submetidas a uma ferramenta de apoio, denominada *SQL Adjust Advisor*, disponibilizada no produto *Tuning Pack* do *Enterprise Manager*. Criaram-se as mesmas tabelas, sem índices, os mesmos dados foram carregados e submeteu-se cada Consulta em separado. Também não houve a necessidade de reiniciar o ambiente antes de cada execução, pois também trabalha-se com recomendações. Como não se trabalha com atribuições gradativas de benefícios, também não ocorreram mais de uma execução por consulta. A Tabela 7.3 exibe, para cada consulta, os índices sugeridos pelo *SQL Adjust Advisor*.

| Consulta | Tabela | Campos |
|---|---|---|
| 1 | Lineitem | L_shipdate |
| 2 | | |
| 4 | Orders | O_orderdate |
| 4 | Lineitem | L_orderkey |
| 10 | Orders | O_orderdate |
| 10 | Lineitem | L_orderkey |
| 17 | Lineitem | L_partkey |
| 17 | Part | P_brand, P_container |
| 19 | Part | P_brand, P_container |
| 19 | Part | P_brand, P_container, P_size |
| 19 | Lineitem | L_partkey |

Tabela 7.3: Índices sugeridos pelo *SQL Adjust Advisor* do Oracle 10g.

Mesmo criando manualmente índices para Consulta 2, o otimizador do Oracle utilizou apenas um (tabela Partsupp, campo Ps_partkey). A duração teve ligeiro decréscimo (7%). Já ao re-submeter a Consulta 10, após a criação manual do índice sobre a tabela Customer, campo C_custkey, e 17, após criação

manual do índice sobre a tabela Part, campos P_brand, P_container, P_partkey, o otimizador preferiu ignorar os novos índices.

## 7.4 Resumo do Capítulo

Neste capítulo, apresentamos os testes realizados e os resultados obtidos, os quais demonstram a eficácia das idéias que norteiam a abordagem proposta nesta tese. Vale ressaltar, ainda, que, não foi possível comparar o desempenho da solução não-intrusiva, proposta nesta tese, com a abordagem intrusiva proposta em (Sal04, Morelli06a), uma vez que não conseguimos reproduzir o ambiente utilizado nestes trabalhos. Contudo, foi possível observar que a abordagem não-intrusiva sugeriu configurações de índices muito próximas daquelas obtidas pela abordagem intrusiva (Sal04, Morelli06a).

Uma comparação de desempenho com as abordagens anteriores automática (Sch06, Kai04, Luh07, Bruno07a), para o projeto físico automático de bancos de dados, também não foi possível, uma vez que a implementação destes protótipos são proprietárias (comerciais) ou não estão disponíveis.

O *benchmark* TPC-H, utilizado em nossos testes, simula um ambiente com características OLAP. Assim, o TPC-H é constituído apenas por consultas onerosas, não possuindo nenhuma operação de atualização (*update*, *insert* ou *delete*). Logo, não foi possível avaliar o acamponhamento do nível de fragmentação das estruturas de índices. Para isso, será necessário utilizar uma carga de trabalho que contenha uma grande quantidade de operações de atualização. Neste sentido, sugerimos utilizar o *benchmark* TPC-C, com algumas modificações, para acentuar a quantidade de operações de atualização.

A seguir, destacamos alguns testes que devem ser realizados no futuro:

- Comparar o desempenho das soluções intrusivas, apresentadas em (Sal04, Morelli06a), com a solução não-intrusiva, proposta nesta tese.
- Comparar a quantidade de índices candidatos gerados pelas abordagens intrusiva e não-intrusiva.
- Avaliar o acompanhamento do nível de fragmentação das estruturas de índices.