4
Implementação e Resultados Experimentais

Com o objetivo de fazer a criação automática de visões materializadas, ou seja, prover uma solução *on-the-fly* para o problema de seleção de visões materializadas, implementamos as heurísticas e a arquitetura proposta no capítulo 3.

Como aplicações de suporte a decisões caracterizam-se por examinar grande quantidade de dados e executar consultas com alto grau de complexidade, as visões materializadas podem melhorar significativamente essas aplicações se forem bem selecionadas.

Para realizar testes com a finalidade de mensurar a eficácia de nossa abordagem, utilizamos o benchmark TPC-H (apresentado no apêndice B), de suporte a decisão. O TPC-H é formado por um conjunto de 22 consultas *ad-hoc* e possui uma estrutura padrão composta por oito tabelas. Para a realização dos testes foi utilizada uma base de dados de 1GB.

Os testes foram realizados utilizando-se duas máquinas, ou seja, a metabase local e a ferramenta dessa dissertação encontravam-se em uma máquina e servidor do banco de dados alvo, com a base de dados de 1GB, em outra máquina.

A configuração da máquina servidora do banco de dados alvo é Intel Core2 CPU E7400 2.80GHz com 3.23GB de memória RAM. E a configuração da máquina com a aplicação dessa dissertação é Intel Core2 CPU 6420 2.13GHz com 4GB de RAM.

O SGBD utilizado para teste foi o SQL Server 2008 e a arquitetura utilizada é implementada em Java. As heurísticas foram implementadas em Java no Componente de Manutenção de Visões Materializadas. O *driver* para captura de estatísticas (SADriver) e o *driver* para atualizar o SGBD (UDDriver) foram implementados com foco para gerenciar visões materializadas. A arquitetura é genérica e não intrusiva ao código do SGBD podendo ser utilizada com outros SGBDs.

A carga de trabalho submetida ao SGBD para teste foi obtida baseada no *toolkit Database Test 3* (DBT-3). Este *toolkit* fornece consultas baseado no benchmark TPC-H, com características de consultas dispendiosas, simulando um ambiente com características OLAP. Através do DBT3 criamos a base de dados segundo as diretrizes do TPC-H para o SQL Server. Com os scripts das consultas do DBT3 adaptados à sintaxe do SQL Server, criamos grupos de consultas a serem executados randomicamente.

Com a finalidade de comparar nossa seleção de visões materializadas, utilizamos a ferramenta *Database Engine Tuning Advisor* do SQL Server 2008 que seleciona um conjunto de visões materializadas sem requerer um entendimento da estrutura do banco de dados ou das características internas do Microsoft SQL Server. Entende-se por características internas as tabelas, quantidade de tuplas das tabelas, tamanho máximo da tupla, entre outras.

Para utilizar essa ferramenta é necessário passar um script representando a carga de trabalho, ou capturar uma carga através da mesma ferramenta e salvar, e depois passar a carga para a ferramenta a fim de ser analisada. A carga passada é uma carga fixa, ou seja, não varia o seu conteúdo enquanto a ferramenta analisa a carga. A estratégia utilizada nesta dissertação é diferente, pois a cada momento que parte da carga que está sendo executada é capturada, ela é analisada para se criar visões.

4.1.Instanciação da arquitetura não-intrusiva para criação de visões materializadas no SQL Server

Para adaptar e implementar a arquitetura proposta por [Monteiro08] para criação automática de visões materializadas foi necessário modificar e acrescentar funções nos *drivers* de captura da carga de trabalho, de obtenção de estatística e do executor.

A função do *driver* que captura a carga de trabalho é realizar a busca por consultas executadas na metabase do SGBD alvo, especificamente na visão do sistema *syscacheobjects do SQL Server*. Portanto, como o próprio nome dá a entender, esta visão armazena temporariamente objetos, neste caso, consultas executadas. Como a busca por novas consultas nessa visão é feita regulamente, o *driver* captura várias vezes uma consulta, que foi executada uma única vez. Para não capturar uma única consulta mais de uma vez, foi armazenado o identificador

da consulta e a quantidade de vezes que foi executada. Desta forma ele somente captura consultas com um identificador diferente, ou se tiver mesmo identificador, se a quantidade de vezes que foi executada é diferente. Constatado que é uma consulta nova, armazena-se na metabase local a seguinte tripla: a consulta em si, o plano de execução e sua estimativa de custo.

Cada componente tem o *driver* correspondente. Os *drivers* são responsáveis por acessar o banco de dados se necessário. Para se obter as estatísticas necessárias para seleção de visão materializada foi necessário criar funções para inserir dados relacionados à visão materializada na metabase local. Ao ser inicializado o componente de estatísticas, ele chama a função ***insertTablesSize*** do *driver* de obtenção de estatística, essa função foi acrescentada para armazenar na metabase local os tamanhos das tabelas do banco de dados alvo.

Para cada consulta nova capturada e inserida na metabase local, o componente CSO chama a função ***analyseNewQueries*** do *driver* de estatísticas para analisar essa consulta e inserir suas subconsultas e os respectivos custos estimados na metabase local. A função ***insertSubQuery*** insere na metabase local a subconsulta com seu respectivo custo estimado.

Ao selecionar o conjunto de visões materializadas hipotéticas, armazenamos as visões hipotéticas na metabase local, junto com seu custo estimado através da função ***insertView***. Através dessas funções capturamos as informações estatísticas necessárias para execução das heurísticas de seleção de visões hipotéticas e seleção final de visões materializadas.

Com as informações estatísticas já capturadas, foram criadas algumas funções para buscar as estatísticas necessárias para execução dos algoritmos. A seguir estão listadas essas funções:

- *getTableSize(t)*: função acrescentada para retornar o tamanho da tabela *t*
- *getRowNumberView(v)*: função acrescentada para retornar a quantidade de tuplas estimadas da visão *v*
- *getMaxTupla(v)*: função acrescentada para retornar o tamanho máximo de uma tupla da visão materializada candidata
- *getExcutionCostClauseView(v)*: função acrescentada para retornar o custo de execução da consulta que define a visão
- *getExecutionCostQuery(q)*: função acrescentada para retornar o custo estimado de execução de uma consulta ou subconsulta

- *getSubQueryList(q)*: função acrescentada para retornar uma lista de subconsultas da consulta q
- *getQueryList()*: função implementada para retornar uma lista de *consultas* não analisadas

No componente executor CE foi criada a função ***materializeView*** que recebe a visão a ser materializada como parâmetro para fazer as configurações necessárias para materializar a visão no banco de dados alvo. Após a materialização de uma visão, o componente CSO atualiza o custo de criação e de um *full scan* das visões hipotéticas através da função ***updateViews***.

As figuras exibidas a seguir mostram algumas classes utilizadas na implementação da arquitetura. A Figura 9 representa as classes dos drivers que os componentes utilizam. Os *drivers* nessa dissertação foram implementados para o *SQLServer*. O CSO utiliza o *driver* SADriver, o CWO utiliza o WADriver e o componente CE utiliza o UDDriver. A classe apresentada na Figura 10 representa o componente CWO. A Figura 11 mostra a classe CMVM, CSO e CE.

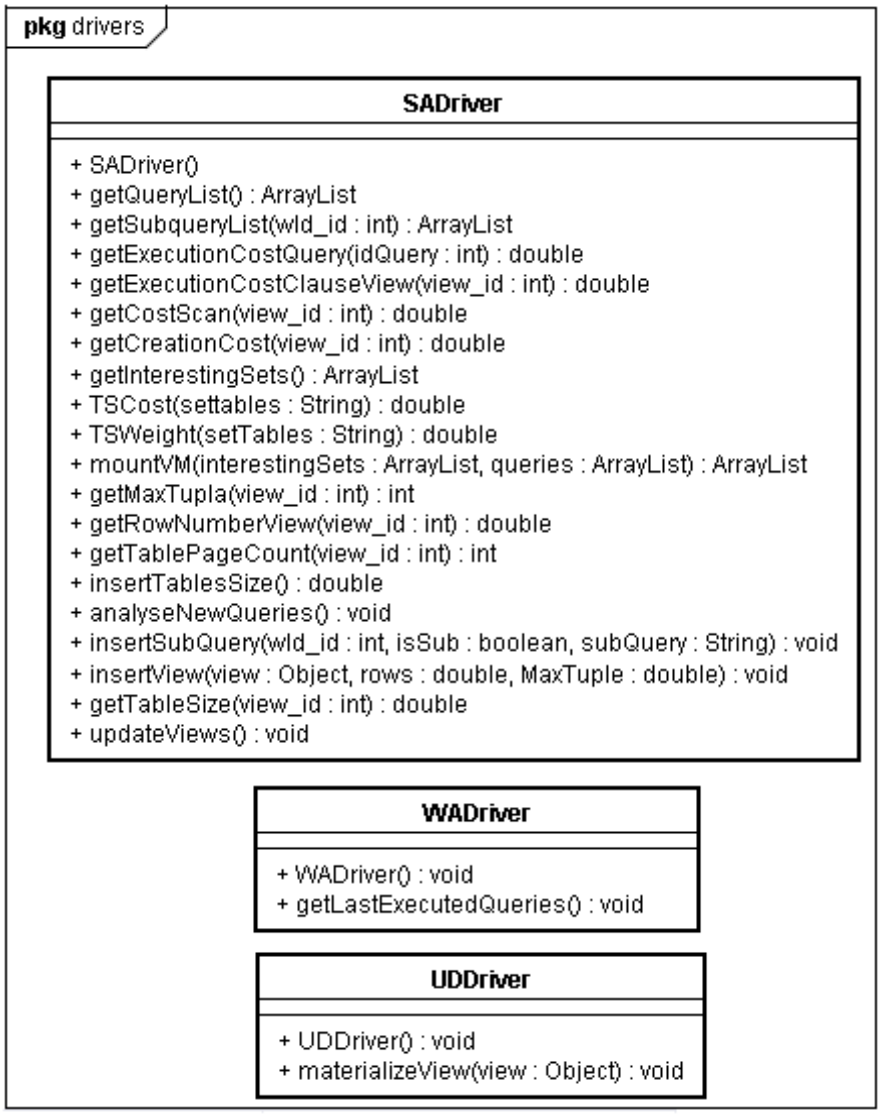

Figura 18 - Classes dos *drivers* da arquitetura

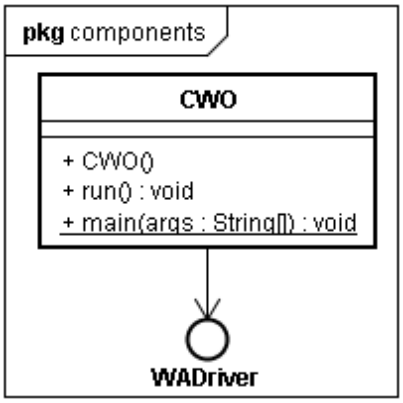

Figura 19 - Classe de CWO

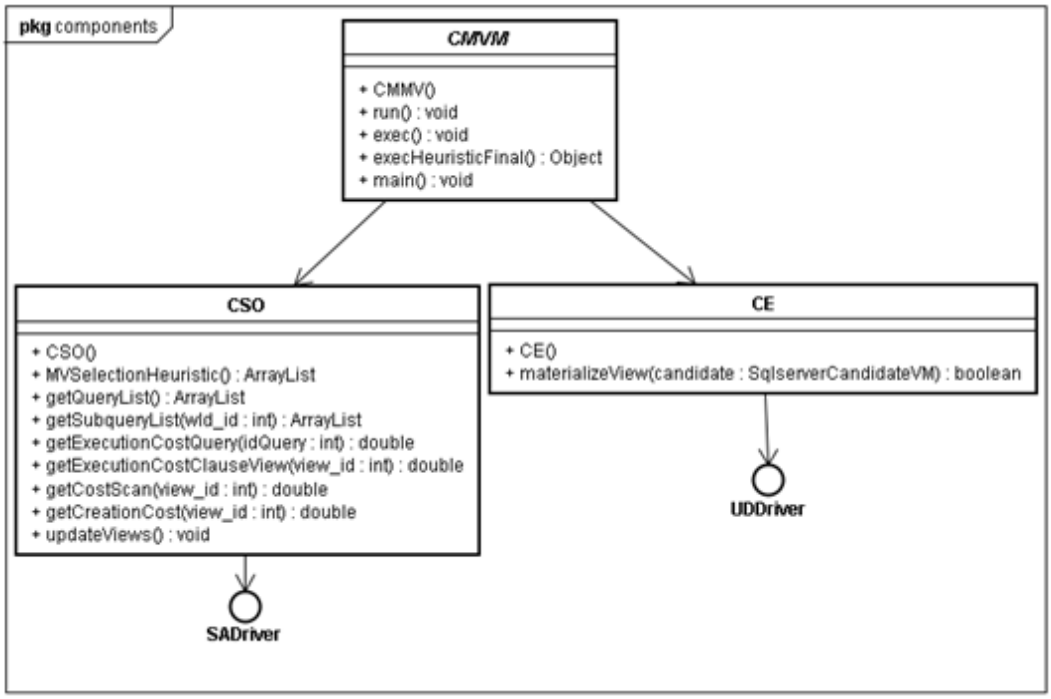

Figura 20 - Classes dos componentes CSO, CE e CMVM

4.2.Resultados Experimentais com SQL Server 2008

A princípio foram executados testes no *Database Engine Tuning Advisor*. Esta é uma ferramenta no SQL Server 2008 que capacita o usuário DBA a sintonizar o banco de dados para melhorar o processamento de consultas. Ela ajuda a selecionar um conjunto de índices, visões materializadas, e partições sem requerer um entendimento da estrutura do banco de dados ou das características internas do Microsoft SQL Server [MicrosoftTuning].

Essa ferramenta analisa uma carga de trabalho e uma implementação física do banco de dados. *Database Engine Tuning Advisor* usa scripts de transações para simular uma carga de trabalho como entrada para sugerir configurações para sintonia fina do banco de dados [MicrosoftTuning]. Ela processa a carga todo de uma vez para sugerir configurações, ao contrário da estratégia dessa dissertação que analisa a carga incrementalmente.

A fim de comparar as visões materializadas selecionadas na implementação do trabalho desta dissertação, analisamos as visões materializadas sugeridas por este trabalho e aquelas sugeridas pela ferramenta *Database Engine Tuning Advisor* a partir de uma carga de trabalho baseada no benchmark TPC-H. Foram executadas três cargas de trabalho no banco de dados alvo com os componentes automáticos capturando e analisando a carga executada. Essas mesmas duas

cargas foram passadas através de *scripts* SQL para o *Database Engine Tuning Advisor*.

As cargas de trabalhos usadas para realização dos testes encontram-se no apêndice C. As cargas são conjuntos contendo as consultas *ad-hoc* as quais são submetidas ao banco TPC-H apresentado no apêndice B. Como as cargas foram geradas aleatoriamente, utilizando todas as consultas fornecidas pelo benchmark TPC-H, era esperado que as visões hipotéticas criadas demorassem a ser materializadas. A geração aleatória é para simular um ambiente de produção onde as consultas são submetidas de forma variada, de forma que as consultas onerosas e as consultas leves sejam submetidas intercaladas.

4.2.1. Primeira carga de trabalho submetida

Para a primeira carga de trabalho, a ferramenta de *tuning* do SQL Server 2008 sugeriu as visões mv_1, mv_2 do apêndice D. O projeto implementado nesta dissertação materializou as visões mv_7 e mv_8 do apêndice D. Analisamos cada consulta da carga de trabalho na presença de todas as visões acima citadas.

Das visões sugeridas pela ferramenta de *tuning* do SQL Server e pela ferramenta dessa dissertação, o otimizador utilizou a visão mv_7, mv_8. A visão mv_1 é parecida com a visão mv_8, porém a mv_1 agrega o campo *l_extendedprice*. Esse campo foi agregado com o intuito de ser usado na consulta mais externa da consulta 17 do apêndice B na seção B.1. Porém este campo é apenas um dos campos necessários para a consulta mais externa. Desta forma, o otimizador optou por utilizar a mv_8, que contém apenas os campos necessários para a subconsulta da consulta 17. A visão mv_8 foi utilizada pelo otimizador e trouxe um benefício percentual acima de 50%.

A visão mv_2 não foi usada por nenhuma consulta. A visão mv_7 materializada pela ferramenta dessa dissertação foi utilizada pelo otimizador e trouxe um benefício percentual elevado para a consulta que a utiliza. A seguir segue uma tabela que mostra as consultas que utilizaram alguma visão, a visão utilizada, o custo de execução estimado da consulta sem a visão e o custo estimado com a visão materializada.

Consulta	VM utilizada	Custo S/ VM	Custo C/ VM
1	mv_7	110.6829	0.033706
17	mv_8	219.8217	104.0628

Tabela 2 - Visões Materializadas utilizadas para primeira carga

4.2.2.Segunda carga de trabalho submetida

Para a segunda carga de trabalho, a ferramenta de *tuning* do SQL Server 2008 sugeriu as visões mv_1, mv_2, mv_3 do apêndice D. E a ferramenta deste trabalho sugeriu a visão mv_7 do mesmo apêndice. Cada consulta da carga de trabalho foi analisada na presença de todas as visões acima citadas.

Das visões sugeridas pela ferramenta de *tuning* do SQL Server e pela ferramenta dessa dissertação, o otimizador utilizou a visão mv_1, mv_3 e mv_7. Como descrito em 4.2.1, na presença das visões mv_1 e mv_8, o otimizador utiliza a visão mv_8. Porém, para essa carga, a visão mv_8 não foi materializada, pois o seu benefício não atingiu o custo de criação da visão, valor escolhido nessa dissertação para se materializar a visão.

Seu benefício não foi atingido, pois a consulta 17, que dá benefício para essa visão, ocorreu com mais frequência nas últimas consultas submetidas pela carga. Como a carga era analisada para dar benefício de 2 em 2 minutos, a visão não foi materializada antes da execução da carga terminar.

A visão mv_3, sugerida pela ferramenta de *tuning* do SQL Server foi utilizada pelo otimizador. Ela também foi criada hipoteticamente pela ferramenta dessa dissertação, porém não atingiu o limite para ser materializada. A visão mv_7 materializada pela ferramenta dessa dissertação foi utilizada pelo otimizador e trouxe um benefício percentual elevado para a consulta que a utiliza. A seguir segue uma tabela que mostra as consultas que utilizaram alguma visão, a visão utilizada, o custo de execução estimado da consulta sem a visão e o custo estimado com a visão materializada.

Consulta	VM utilizada	Custo S/ VM	Custo C/ VM
1	mv_7	110.68290	0.03371
17	mv_1	219.82170	104.21170
18	mv_3	913.14300	795.62220

Tabela 3 - Visões Materializadas utilizadas para segunda carga

A visão mv_3 foi criada hipoteticamente, mas a ferramenta não decidiu materializá-la. A média de execuções estimadas da consulta 18 para materializar a visão mv_3 pela ferramenta dessa dissertação são 30 execuções. Pois o benefício hipotético é a metade do benefício real dela, e o custo de criação é 30 vezes maior do que o benefício hipotético. Nessa carga a consulta 18 foi executada 16 vezes. Se essa visão fosse materializada pela ferramenta dessa dissertação, ela traria um bom benefício à carga de trabalho como um todo. Desta forma, essa visão é um falso negativo.

4.2.3. Terceira carga de trabalho submetida

Para a terceira carga de trabalho, a ferramenta de *tuning* do SQL Server 2008 sugeriu as visões mv_1, mv_2, mv_3, mv_4, mv_5 e mv_6 do apêndice D. E a ferramenta deste trabalho sugeriu a visão mv_7 e mv_8 do mesmo apêndice. Cada consulta da carga de trabalho foi analisada na presença de todas as visões acima citadas.

Das visões sugeridas pela ferramenta de *tuning* do SQL Server e pela ferramenta dessa dissertação, o otimizador utilizou a visão mv_3, mv_5, mv_6, mv_7 e mv_8. As visões mv_5 e mv_6 sugeridas pela ferramenta de *tuning* do SQL Server, são sobre subconjunto de tabelas que a ferramenta dessa dissertação não considerou como subconjunto de tabelas interessantes nas primeiras avaliações. Dessa forma, quando elas foram consideradas interessantes, parte da carga já havia sido executada, e os benefícios deixaram de ser acumulados desde o princípio, assim elas não acumularam benefícios suficientes para serem materializadas.

As visões mv_7 e mv_8, materializadas pela ferramenta dessa dissertação, foram utilizadas pelo otimizador e trouxeram um benefício percentual elevado para a consulta que as utilizam. A seguir segue uma tabela que mostra as consultas que utilizaram alguma visão, a visão utilizada, o custo de execução estimado da consulta sem a visão e o custo estimado com a visão materializada.

Consulta	VM utilizada	Custo S/ VM	Custo C/ VM
1	mv_7	110.6829	0.03370602
16	mv_6	39.85137	39.72492
17	mv_8	219.8217	104.0628
18	mv_3	913.143	795.4163
22	mv_5	32.29761	9.984818

Tabela 4 - Visões Materializadas utilizadas para quarta carga

A visão mv_3, sugerida pela ferramenta de *tuning* do SQL Server, não foi materializada pela ferramenta dessa dissertação. Como foi explicado na seção 4.2.2, é necessário em média 30 execuções da consulta 18 para que a visão mv_3 seja materializada, e a consulta 18 foi submetida 16 vezes pela terceira carga. Se essa visão fosse materializada pela ferramenta dessa dissertação, ela traria um bom benefício à carga de trabalho como um todo. Desta forma, essa visão é um falso negativo.

A visão mv_6 não trouxe grande benefício para a consulta que a utiliza. Desta forma, não valia a pena materializá-la. A visão mv_5 trouxe um benefício percentual acima de 50%, porém a diferença do custo estimado da consulta com e sem a visão, é abaixo de 23 em unidade de custo estimado pelo otimizador. Dessa forma, não pode ser considerado um falso negativo, pois não traz um benefício elevado para a consulta que a utiliza.

4.3.Resultados de desempenhos

Com o intuito de verificar a eficiência das visões materializadas por este trabalho, submetemos algumas cargas de trabalho no banco de dados alvo. Uma mesma carga foi submetida em ambientes diferentes.

No primeiro ambiente, a carga é executada no banco de dados alvo, sem nenhuma visão materializada. No segundo ambiente, a carga é executada no banco de dados alvo com a ferramenta desta dissertação ligada. E no terceiro ambiente, a carga é executada com as visões materializadas pela ferramenta dessa dissertação, após ela ser desligada.

Cada vez que a carga era totalmente executada em uma situação, o servidor era parado para esvaziar resultados de consultas armazenados em *cache*, depois reiniciado para executar a carga na outra situação.

A cada dez consultas executadas da carga, o tempo era capturado para avaliarmos o tempo de execução a cada dez consultas. Cada carga foi executada dez vezes em uma mesma situação. Dessa forma, a seguir segue o resultado dos testes em gráficos com a média dos tempos capturados.

As mesmas cargas executadas na seção 4.2, foram executadas nesses três ambientes para avaliar a eficiência das visões materializadas por essa dissertação. As figuras 22, 23 e 24 representam os gráficos de tempo de execução das cargas 1, 2 e 3 respectivamente.

Nestes gráficos podem ser visto que o tempo de execução das cargas com a ferramenta dessa dissertação ligada e com ela desligada sem visões materializadas são parecidos. A ferramenta ligada não prejudica a carga de trabalho e ainda ajuda por materializar as visões úteis para a carga em execução. Isto mostra que a ferramenta pode ser usada sem prejudicar muito o tempo de resposta as consultas da carga de trabalho.

Também pode ser visto nestes gráficos, que as visões materializadas são usadas pelas consultas da carga de trabalho como pode ser visto pela linha que representa o terceiro ambiente. Ela representa o benefício em tempo que as visões materializadas trouxeram para a carga de trabalho.

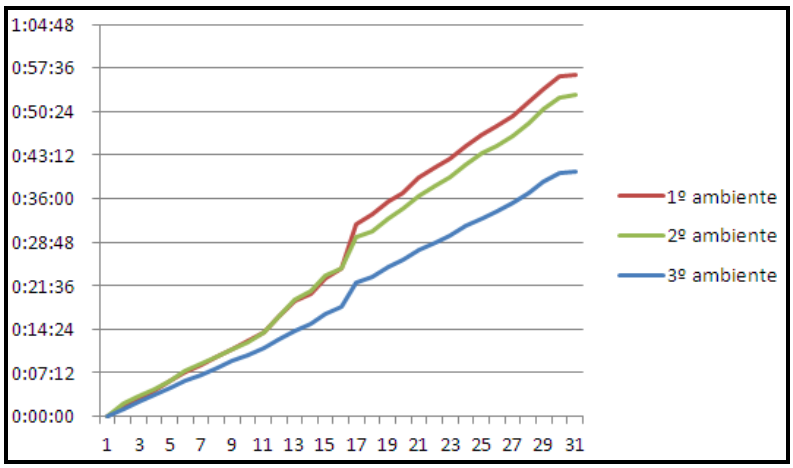

Figura 21 - Gráfico de tempo de execução Carga 1

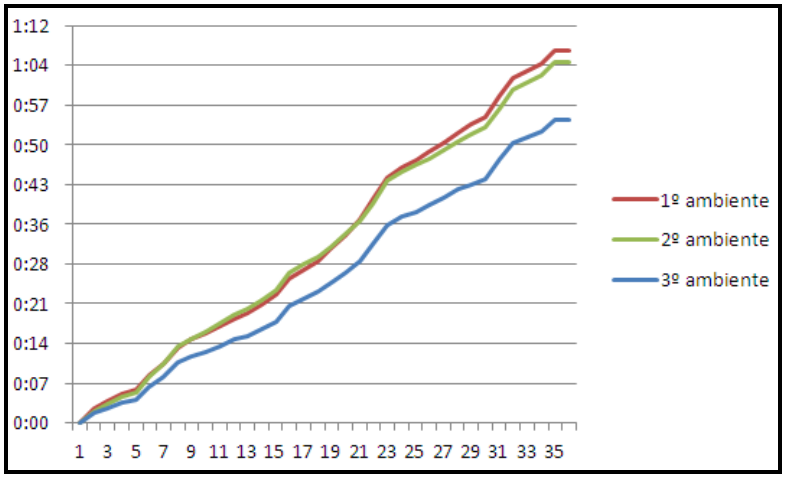

Figura 22 - Gráfico de tempo de execução Carga 2

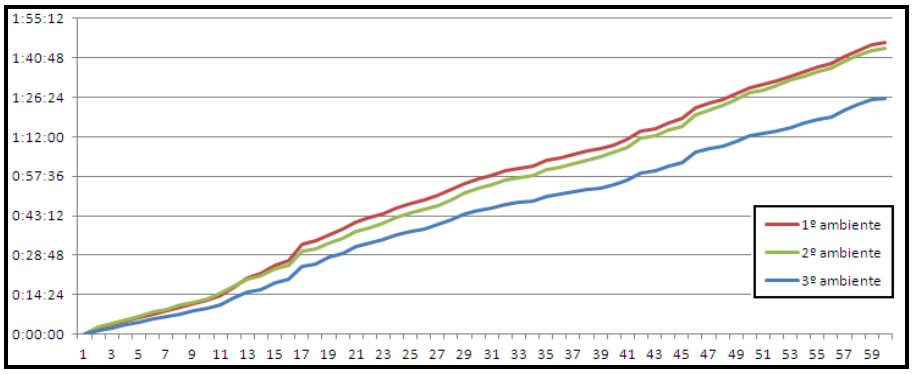

Figura 23 - Gráfico de tempo de execução Carga 3

4.4. Resumo do capítulo

Neste capítulo foram apresentados testes realizados e obtidos que demonstram a eficácia e a eficiência das heurísticas propostas. Os testes realizados com o objetivo de comparar as visões sugeridas neste trabalho com visões sugeridas por ferramentas privadas como *Database Engine Tuning Advisor* mostraram que a heurística de seleção adaptada de [Agrawal et al. 00] implementada neste trabalho trouxe alguns resultados semelhantes.

Os testes realizados com o objetivo de comprovar a eficiência de ter uma criação automática de visões materializadas mostraram que este trabalho materializou visões que melhoraram significantemente o tempo de resposta da carga de trabalho. Mesmo com a ferramenta ligada, o tempo de resposta as consultas não foi muito prejudicado. E com as visões materializadas pela ferramenta e com a ferramenta desligada, o tempo de resposta as consultas melhorou significantemente.