Universidade de Évora
Mestrado em Engenharia Informática

Estudo da performance de leitura/pesquisa em Sistemas de Gestão de Base de Dados *row-oriented* e *column-oriented* em diferentes ambientes de execução

Aluno: Rui Miguel Dias Anastácio
Orientador: Luis Arriaga da Cunha

Évora, Janeiro de 2011

Universidade de Évora
Mestrado em Engenharia Informática

Estudo da performance de leitura/pesquisa em Sistemas de Gestão de Base de Dados *row-oriented* e *column-oriented* em diferentes ambientes de execução

Aluno: Rui Miguel Dias Anastácio
Orientador: Luis Arriaga da Cunha

Resumo

Tem-se assistido, nos últimos anos, a um aumento considerável no recurso a *data warehouses* que manipulam enormes quantidades de dados mantendo elevada capacidade para responder a análises e pesquisas *ad-hoc*.

Muitas das soluções actuais para *data warehouses* baseiam-se em Sistemas de Gestão de Bases de Dados Relacionais com arquitecturas clássicas, designadas por *row-oriented*. Recentemente surgiram sistemas adoptando o paradigma *column-oriented* que, em certos casos, apresentam um desempenho com ordens de grandeza superiores.

Neste trabalho estuda-se sistematicamente a performance nas operações em *data warehouses* suportadas por sistemas *row-oriented* (PostgreSQL e MySQL) e *data warehouses* suportadas por sistemas *column-oriented* (MonetDB e ICE). Utilizando o padrão *Star Schema Benchmark* são efectuadas medições em diferentes ambientes de execução, fazendo variar parâmetros tais como a escala e o CPU.

As diferentes medições são armazenadas num *data warehouse* sendo este explorado por técnicas convencionais e técnicas de *Data Mining*. Como resultado obteve-se uma caracterização das situações em que o recurso a uma arquitectura *column-oriented* é mais, ou menos, favorável, face a uma arquitectura *row-oriented*, no contexto dos *data warehouses*.

As diferentes medições são armazenadas num *data warehouse* sendo este explorado por técnicas convencionais e técnicas de *Data Mining*. Como resultado da análise obteve-se uma caracterização das situações nas quais uma arquitectura é mais favorável face à outra, no contexto dos *data warehouses*.

Resume

Read/Search Performance Study in row-oriented and column-oriented Database Management Systems in different execution environments

There has been, in recent years, a considerable increase in the use of data warehouses that handle massive amounts of data while maintaining high capacity to respond to analysis and ad-hoc queries.

Many of the current solutions for data warehouses are still based on classical database management systems architecture, designated by row-oriented. Recently the column-oriented paradigm has been exploited and implemented in several database systems, revealing in some cases a performance orders of magnitude higher.

In this work, a systematic query performance study is conducted on data warehouses supported by row-oriented database systems (PostgreSQL and MySQL) and data warehouses supported by column-oriented database systems (MonetDB and ICE). Using the Star Schema Benchmark measures are performed under different execution environments, varying parameters like scale and CPU.

The different measurements are stored in a data warehouse which is then explored by conventional techniques and Data Mining techniques. As a result of the analysis we obtained a characterization of the situations in which an architecture is more favorable compared to the other, in the context of data warehousing.

À minha mulher Manuela e filhas Madalena e Lara

Agradecimentos

Gostaria de agradecer ao Professor Luís Arriaga da Cunha, orientador desta dissertação de mestrado, por toda a ajuda prestada e todo o apoio concedido, sempre disponível para qualquer esclarecimento. A sua orientação foi sem dúvida determinante para um trabalho motivante e de sucesso.

Gostaria de agradecer à Professora Teresa Gonçalves pelo apoio dado na análise dos dados, em particular nas técnicas de *Data Mining* e *Inteligência Artificial.*

Gostaria ainda de agradecer os docentes do Departamento de Informática, com os quais tive o prazer de aprender e partilhar conhecimentos durante a parte lectiva deste mestrado.

Por fim gostaria de agradecer à minha mulher, Manuela, pelo seu apoio e compreensão no momentos mais críticos, permitindo que grande parte do meu tempo fosse canalizado para esta dissertação.

Conteúdo

Lista de Tabelas

Lista de Figuras

Acrónimos

ATA Advanced Technology Attachment

BD Base de Dados

BI Business Intelligence

DDL Data Definition Language

DM Data Mining

DW Data Warehouse

ICE Infobritgh Comunity Edition

IDC International Data Corporation

IO Input/Output

MPP Massive Parallel Processing

MV Materialized View

RAM Random Access Memory

ROS Read Optimized Store

SATA Serial Advanced Technology Attachment

SF Scale Factor

SGBD Sistema de Gestão de Bases de Dados

SGBDR Sistema de Gestão de Bases de Dados Relacional

SQL Structured Query Language

SSB Star Schema Benchmark

TPC Transaction Processing Performance Council

WOS Write Optimized Store

Capítulo 1

Introdução

O modelo de Bases de Dados mais utilizado na actualidade, e provavelmente nos próximos anos, é o modelo Relacional. Neste modelo a informação é modelada em tabelas, representando tipos de entidades e associações entre elas. Cada entidade é assim representada em tabelas bidimensionais e cada associação dará origem a novas colunas em tabelas. Estruturas de dados simples, um reduzido número de conceitos e uma independência lógica e física dos dados, são algumas das características fundamentais que levaram ao sucesso deste modelo e à sua larga aceitação por parte da comunidade informática.

Uma das principais tarefas do Sistema de Gestão de Bases de Dados Relacionais (SGBDR), software que implementa o modelo Relacional de Bases de Dados, é o armazenamento físico das tabelas bidimensionais resultantes do processo de modelação. Acontece que os dispositivos de armazenamento utilizados actualmente são dispositivos com endereçamento unidimensional, ou seja, a posição de cada dado é determinada por um único índice numérico (estrutura de vector). Há assim que mapear uma tabela bidimensional (linha/coluna) num sistema unidimensional (posição). Este problema é geralmente resolvido de uma das seguintes formas: armazenamento linha a linha ou armazenamento coluna a coluna, conforme representado na figura 1.1.

Tradicionalmente os sistemas adoptaram o armazenamento linha a linha, uma vez que fornecem uma boa performance para grande parte dos sistemas de bases de dados, particularmente sistemas transaccionais. Este sistemas caracterizam-se por um tratamento registo a registo (linha a linha), em que o número de registos a processar é reduzido, e estão optimizados para cargas de trabalho (*workloads*) de escrita intensa (*write-intensive*).

Recentemente sugiram aplicações com características diferentes dos sistemas transaccionais, nomeadamente as aplicações analíticas. Estas são utilizadas para explorar os dados de forma a compreendê-los e a retirar conhecimento útil para a gestão do negócio. São suportadas por bases de dados com características próprias, designadas por *data warehouses*, que armazenam informação de vários anos dos diversos processos

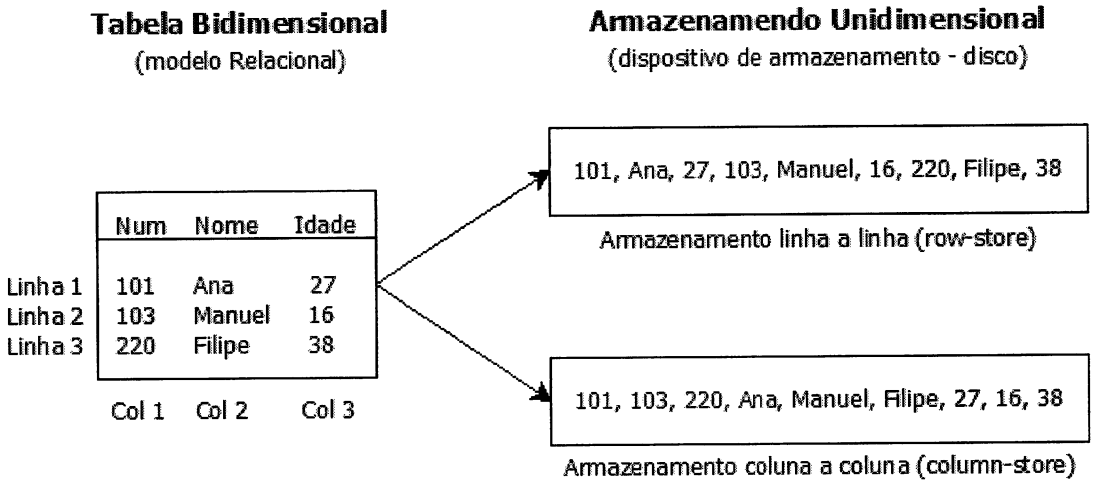


Figura 1.1: Mapeamento de uma tabela bidimensional num dispositivo de armazenamento unidimensional

de negócio, de forma integrada, proporcionando uma visão única do "negócio" da empresa ou instituição.

As consultas realizadas sobre um *data warehouse*, diferentes das consultas realizadas sobre um sistema transaccional [1], são tendencialmente:

- **Menos previsíveis**. Enquanto que os sistemas transaccionais são essencialmente utilizados para automatizar tarefas, originando consultas pré-programadas (mudando apenas os valores das variáveis em tempo de execução), as consultas em sistemas analíticos têm uma natureza muitas vezes exploratória, na qual o analista inicia o seu trabalho com consultas *ad hoc* seguindo a exploração os dados de forma iterativa, baseada nos resultados que vão sendo obtidos;

- **Mais demoradas**. Uma consulta transaccional é geralmente de curta duração, envolvendo poucos dados, e.g. actualizar o stock, adicionar um produto, etc. Pelo contrário, uma consulta ao *data warehouse* necessita geralmente de mais dados para poder responder a questões analíticas, tais como: cálculo de valores agregados, resumos, descoberta de tendências, correlação de valores, entre outros;

- **Mais orientadas à leitura do que à escrita**. A análise implica essencialmente a leitura dos dados existentes, sendo a sua escrita (e eventual actualização) feita em *batch*, em períodos de baixa actividade do sistema. Dessa forma um sistema analítico é um sistema optimizado para um *workload* de leitura intensa (*read-intensive*) ao contrário dos sistemas transaccionais optimizados para um *workload write-intensive*.

- **Foco no atributo em vez da entidade**. As consultas geralmente não incidem sobre uma entidade específica, olhando para todos os atributos, mas sim sobre

um conjunto grande de entidades, agregando e sumariando atributos específicos. Por exemplo, consultas do tipo "qual a média do preço dos produtos?" são mais comuns do que "qual o preço do produto X?". Estas consultas geralmente focam poucos atributos de cada vez.

Tendo em conta estas características especificas tem havido recentemente um crescente interesse nos sistemas de armazenamento coluna a coluna (*column-oriented*), tendo surgido vários projectos de novos SGBD's baseados nesta arquitectura, e diversos estudos comparativos, nos quais se tenta perceber quais a suas vantagens.

1.1 Objectivos e contributos da dissertação

O objectivo deste trabalho de dissertação é o estudo e a caracterização das situações em que o recurso a sistemas de gestão de bases de dados de arquitectura *column-oriented* é mais (ou menos) favorável à utilização de sistemas de arquitectura *row-oriented*, quando utilizados com um *workload* típico de um *data warehouse*.

Para tal foi estudado o comportamento, face a variáveis tais como a memória RAM disponível, a dimensão da Base de Dados ou a velocidade do leitura do disco, de sistemas orientados à coluna e de sistemas orientados à linha, num *data warehouse* analítico de referência, o *Star Schema Benchmark* (SSB).

Foram criadas várias bases de dados SSB, de diferentes escalas, em 4 SGBD's distintos: dois de arquitectura *row-oriented*, PostgreSQL e MySQL, e dois de arquitectura *column-oriented*, Infobritgh Comunity Edition (ICE) e MonetDB. Depois foi medida a performance de execução das 13 consultas do *benchmark* em diferentes ambientes de execução, fazendo variar parâmetros como a memória RAM disponível, a velocidade do disco, entre outros. Estas medições foram armazenadas num *data warehouse* sobre o qual foi feita a análise dos resultados. Recorreu-se a técnicas de *Data Mining* para descobrir conhecimento neste *data warehouse*.

Pretende-se assim contribuir para o estudo da aplicabilidade dos sistemas *column-oriented* a bases de dados analíticas, face à utilização do tradicionais sistemas *row-oriented*, evoluindo no sentido de melhor compreender as variáveis que influenciam a sua performance.

1.2 Estrutura da dissertação

A dissertação é constituída por 6 capítulos organizados da seguinte forma:

O presente capítulo 1 introduz o problema do mapeamento de uma tabela bidimensional num dispositivo unidimensional, suas formas de resolução, a arquitectura de armazenamento linha a linha e a arquitectura de armazenamento coluna a coluna.

Refere o crescente interesse pelos sistemas analíticos e caracteriza as consultas típicas de um *data warehouse*. Termina apresentando os objectivos e os contributos da dissertação.

O capítulo 2 explora as bases teóricas dos SGBD's orientados à coluna. Enquadra esta arquitectura de armazenamento como outra das técnicas utilizadas em sistemas optimizados para leitura, utilizados nas aplicações analíticas. Refere as principais optimizações no motor de pesquisas, tais como a materialização tardia e a compressão de dados, e a forma como os dados das tabelas são organizados no disco. Continua com uma apresentação dos principais SGBD's, comerciais e *open-source*, que empregam esta arquitectura, e termina com um breve resumo de vários estudos comparativos destas duas arquitecturas.

No capítulo 3 é apresentado o *Star Schema Benchmark*. Este é o *benchmark* utilizado para medir a performance dos sistemas de base de dados considerados neste trabalho. É descrito o modelo de dados, as suas consultas, os objectivos, e o carregamento de dados.

No capítulo 4 é descrita a metodologia utilizada para a realização dos testes, a bateria de testes e o *data warehouse* onde serão armazenados os resultados obtidos.

No capítulo 5 são apresentados os resultados da análise efectuada sobre o *data warehouse* descrito no capítulo 4. Serão utilizadas técnicas de *data mining* e de exploração de dados como ferramentas de análise.

O capítulo 6 conclui e apresenta ideias de trabalho futuro.

Capítulo 2

Sistemas *Column-Oriented*

A ideia da gravação coluna a coluna não é nova, tendo sido primeiramente apresentada em 1985 como um modelo de armazenamento decomposto (*Decomposition Storage Model* [2]) no qual uma tabela seria dividida em colunas, correspondendo a cada um dos atributos, aos quais se adicionava uma chave artificial (posição) necessária para a reconstrução do registo. No entanto, e dada a natureza transaccional das aplicações da altura, o modelo de armazenamento linha a linha foi sendo adoptado, mostrando desde início bons resultados.

Com a evolução e maturação dos sistemas de informação foi crescendo a procura de aplicações de apoio à decisão (*Decision Support Systems - DSS*), aplicações que ajudam o decisor a olhar para o negócio de uma forma integrada, única, global, e que lhe permitem efectuar análise de dados, e.g. cálculo de indicadores, previsão de tendências, descoberta de padrões, entre outros. Actualmente esta área é designada por *Business Intelligence* (BI) que, segundo Howson [3], "*é um conjunto de tecnologias e de processos que permitem às pessoas em qualquer nível da organização aceder e analisar dados*". É uma área em franco crescimento, sendo que a IDC[1] estima um crescimento do mercado de software do *Business Intelligence* na ordem dos 6.9% até 2014 [4].

Um sistema de BI é geralmente suportado por uma base de dados própria para o tipo de carga de trabalho que lhe é atribuída, nomeadamente leitura-intensiva e escrita em *batch*. Estas bases de dados, designadas por *data warehouses* (conforme referido anteriormente) são, segundo Inmon "*uma colecção de dados orientada a uma determinada área de interesse, integrada, não volátil e variam com o tempo (time-variant) utilizada no suporte às decisões da gestão*"[5].

São orientadas a uma determinada área no sentido em que se referem a um processo de negócio específico, e não a uma área operacional, como acontece nas aplicações não analíticas; por exemplo para uma indústria os processos de negócio serão o produto, a ordem, o vendedor, a factura ou a matéria prima.

[1]International Data Corporation, `http://www.idc.com`

Os dados do DW são recolhidos de várias fontes, sejam bases de dados aplicacionais da organização, ficheiros, redes de comunicação ou fontes de dados externas à organização; são convertidos, reformatados, normalizados e integrados no DW de forma a criar uma imagem única da organização. Este processo de integração é fundamental e permite criar a *verdade única* do negócio, algo que muitas vezes não é fácil de atingir dada a proliferação de bases de dados numa organização nas quais existem por vezes diferentes vistas da mesma realidade.

Não volátil uma vez que os dados, após serem escritos na base de dados, não são eliminados e raramente serão actualizados. Estes dados são consultados e mantêm-se na base de dados por largos períodos de tempo, geralmente vários anos, durante os quais são essencialmente lidos por forma a serem analisados. Constituem o histórico da organização. São um conjunto de fotografias (*snapshots*) tiradas à organização ao longo do tempo.

Todos os dados do *data warehouse* estão associados a um momento especifico no tempo, correspondente à imagem da qual fazem parte. Desta forma podemos realizar uma análise temporal dos dados dando-nos a evolução histórica da organização. Geralmente os registos são marcados com uma data ou são englobados numa transacção esta sim marcada com uma data.

As primeiras DW's foram implementadas nos sistemas de bases de dados transaccionais existentes, de arquitectura *row-oriented*, mas com o aumento de complexidade e volume de dados começou a surgir a necessidade de uma sistema específico para o *workload* em causa, capaz de dar resposta a consultas, maioritariamente *ad hoc*, cada vez mais pesadas. Os fabricantes começam a explorar diferentes tecnologias e arquitecturas de software entre elas a forma de armazenamento dos dados coluna por coluna. Começam a surgir cada vez mais projectos de SGBD's *column-oriented*, entre eles o *C-Store* [6], liderado por Mike Stonebraker e desenvolvido por um grupo de Universidades, que recentemente deu origem ao sistema comercial *Vertica*, neste momento um dos principais *players* de mercado.

A secção 2.5 refere os principais SGBD's analíticos, comerciais e *open-source*, que utilizam uma arquitectura *column-oriented*, e que actualmente se encontram no mercado.

À primeira vista a principal vantagem de um sistema *column-oriented* é o facto de ser possível ler apenas as colunas que são relevantes para uma determinada consulta, em vez de ler todas as colunas do tuplo, e a principal desvantagem o facto das actualizações requerem escritas em localizações distintas do disco, obrigando a mais operações de *seek* (posicionamento das cabeças do disco magnético).

Conforme referido anteriormente o *workload* nestes sistemas é essencialmente de leitura intensa, sendo as actualizações realizadas periodicamente (geralmente em períodos de baixa utilização do sistema, e.g. à noite), não havendo a necessidade de actualizações em tempo real. Como tal são sistemas optimizados para leitura (*read-optimized*).

O armazenamento dos dados coluna a coluna é uma das técnicas utilizadas num sistema *read-optimized*, com tal é relevante uma breve apresentação da arquitectura básica destes sistemas.

2.1 Componentes de um sistema *read-optimized*

Genericamente um sistema *read-optimized* (ver figura 2.1) é composto por uma área de armazenamento (*DB storage*) e pelo motor de consultas (*query engine*).

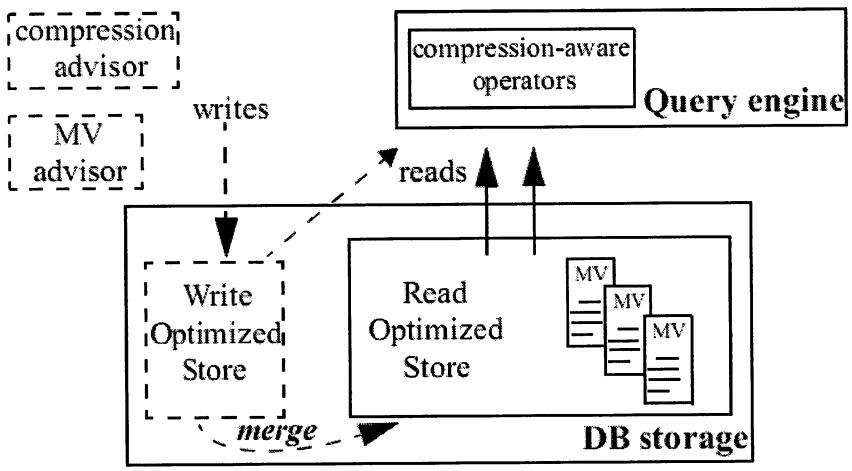


Figura 2.1: Componentes básicos de um sistema *read-optimized*
Retirado de [7]

A área de armazenamento é dividida em duas componentes, o armazenamento optimizado de escrita (*Write Optimized Store - WOS*) que regista todas as alterações aos dados, e o armazenamento optimizado de leitura (*Read Optimized Store - ROS*), principal fonte de dados, desenhada exclusivamente para leitura. Regularmente é despoletado um processo de transferência dos dados da WOS para a ROS (na figura o processo de *merge*), no qual as alterações existentes na WOS são fundidas na ROW.

A ROS é responsável pelo armazenamento dos dados de forma a maximizar a sua performance de leitura. Geralmente são usadas técnicas de compressão de dados, armazenamento coluna a coluna, *densed packed*, criação de vistas materializadas (*Materialized View* - MV) entre as principais.

No motor de consultas são executadas leituras sobre o ROS e WOS, e podem existir operadores próprios para trabalhar com dados comprimidos e dados em blocos (vectores).

Os módulos *Compression Advisor* e *MV Advisor* analisam o *workload* e determinam a melhor estratégia de compressão (tendo em conta o tipo de dados, a sua frequência,

distribuição, etc.) e melhor estratégia de criação das diferentes projecções utilizadas, traduzidas em vistas materializadas, e armazenadas segundo uma arquitectura *column-oriented*.

Tem havido também diversos desenvolvimentos no hardware utilizado em sistemas optimizados para leitura, como é o caso da memória flash em estado sólido, cada vez mais utilizado como meio de armazenamento nos principais sistemas de bases de dados comerciais.

Vejamos agora mais em pormenor a forma como os dados são armazenados em disco.

2.2 Armazenamento em disco

A figura 2.2 apresenta a estrutura genérica das páginas em disco no armazenamento de uma tabela nas arquitecturas *row-oriented* e *column-oriented*. Na primeira uma tabela é armazenada num único ficheiro. Esse ficheiro é composto por uma ou mais páginas (no exemplo com 4Kb cada), sendo que cada uma contém um determinado número de linhas (registos completos) com os dados de cada registo guardados sequencialmente, A1, B1, ..., Z1. A tabela (ficheiro) ocupa as páginas P0, P1, ..., necessárias para guardar todos os dados.

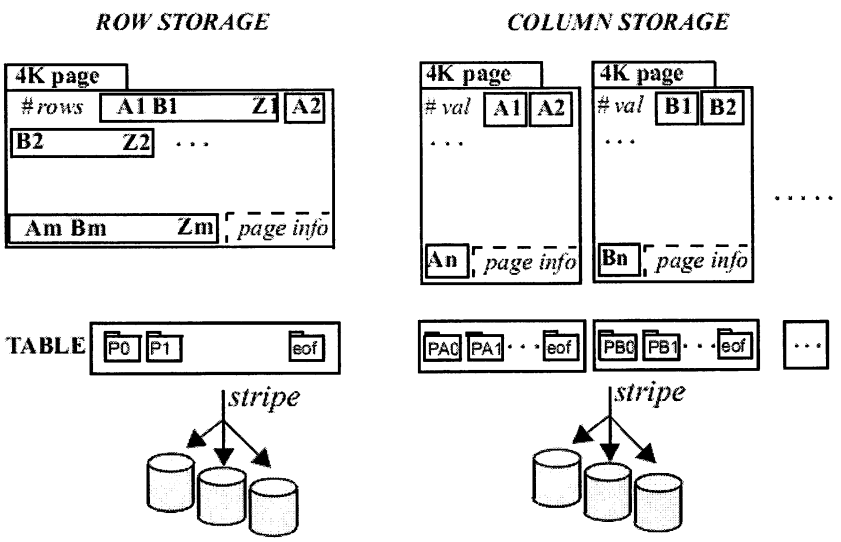


Figura 2.2: Estrutura da página e *layout* do armazenamento para uma tabela
Retirado de [7]

Num sistema *column-oriented* uma tabela é guardada em tantos ficheiros quantos atributos existem (no exemplo, um ficheiro para o atributo A, outro ficheiro para o atributo B, e assim sucessivamente). Cada ficheiro é fisicamente armazenado numa ou

mais páginas do disco (para o atributo A, as páginas PA0, PA1,...). A página contém a indicação do número de valores do atributo e dos seus valores.

2.3 Motor de Consultas

O motor de consultas é responsável pela elaboração do plano de execução da consulta, pedindo os dados de que necessita ao *storage*. Aqui é que se encontra a maior parte da "inteligência" do sistema.

O facto dos dados estarem organizados coluna a coluna permite toda uma nova forma de trabalhar por parte do motor de consultas, tal como representado na figura 2.3.

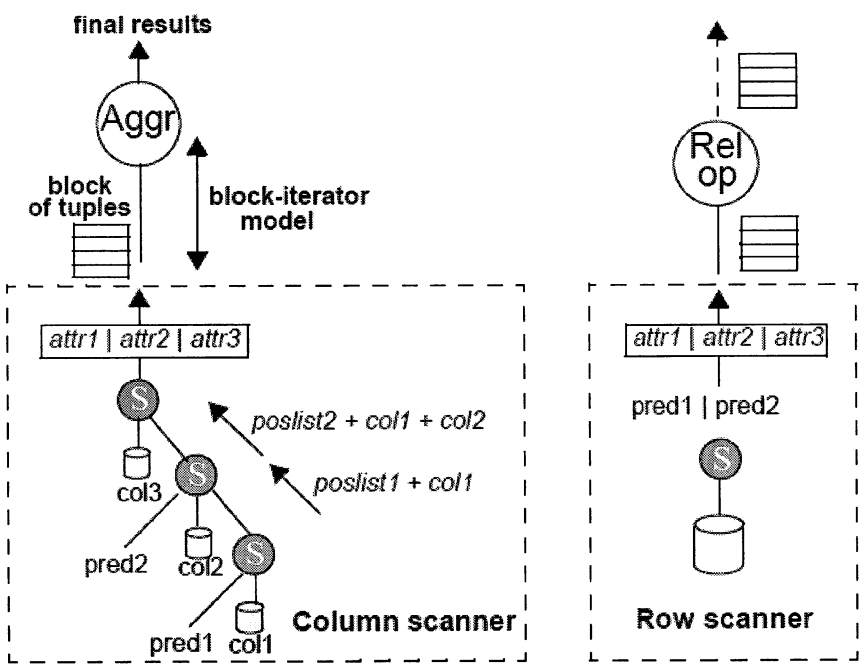


Figura 2.3: Motor de consultas
Retirado de [7]

A figura mostra a forma como o *scanner* (componente responsável pela leitura dos dados, aplicação dos predicados, criação da projecção e disponibilização dos tuplos aos operadores de topo) pode ser muito diferente nas duas arquitecturas. Numa arquitectura *row-oriented* o scanner lê o ficheiro onde residem os dados da tabela, e vai formando os tuplos um a um, e aplicando os predicados às colunas de forma a determinar quais os tuplos a passar para os operadores relacionais. A esta operação de formar os tuplos designa-se por materialização de tuplos (*tuple materialization*).

O *column scanner* é constituído por uma série de nós que trabalham em modo *pipeline*, um para cada coluna analisada, sendo passado de nó para nó uma lista de pares

(posição, valor) correspondente aos tuplos que foram seleccionados pelo predicado. Um nó que receba essa lista só necessita de verificar o seu predicado para os elementos da lista recebida. O resultado será passado numa nova lista.

No exemplo da figura, o *column scanner* começa por ler a coluna *col1* e aplicar o predicado *pred1* (nó mais fundo da árvore, geralmente aquele cujo predicado é o mais selectivo de forma a minimizar a lista de pares). Dessa aplicação resulta a lista de posições *poslist1* e os valores correspondentes *col1*. Na próxima operação apenas o valores da lista *poslist1* são testados relativamente ao pred2, resultando uma nova lista *poslist2*, juntamente com os valores de *col1* e *col2*. O processo continua até formar o tuplo completo.

A figura mostra também outra particularidade dos sistemas *column-oriented*, que é a possibilidade de realizar operações sobre blocos de dados, do qual falaremos mais adiante.

2.4 Optimizações específicas

A organização dos dados em colunas tem proporcionado o desenvolvimento de optimizações específicas capazes de melhorar a eficiência dos sistemas. Abadi et al. [8] refere as técnicas mais importantes:

- **Materialização tardia (*Late Materialization*)** - A materialização do tuplo, ou seja o processo de juntar as várias colunas (guardadas em locais físicos distintos) de forma a formar o tuplo completo para fornecer ao cliente, é um processo que pode ser realizado logo de inicio quando o scanner varre os ficheiros de dados, ou ser deixado para mais tarde, e se possível ser evitado caso o tuplo não faça parte do resultado final. Sistemas recentes tais como o X100 ou o C-Store optam por manter os dados em colunas até mais tarde no plano de execução da consulta, utilizando operadores capazes de processar esses dados em coluna. A ideia é reduzir o tempo gasto a construir os tuplos (*tuple-overhead*), já que essa é uma operação pesada, e que apenas é necessária caso o tuplo seja efectivamente seleccionado;

- **Iteração de blocos (*Block Iteration*)** - Os valores de uma coluna são passados em blocos de um operador para o próximo. Quando a coluna tem um tamanho fixo esses blocos de dados são iterados como um vector, o que não só minimiza o *tuple-overhead*, como também permite a paralelização. Os blocos podem também conter dados comprimidos reduzindo dessa forma o tempo de CPU e aumentando ainda mais a performance;

- **Técnicas de compressão específicas** - Quando os dados são guardados em coluna são mais comprimíveis, uma vez que o domínio de valores é mais reduzido. Por exemplo, se considerarmos a coluna cidade numa tabela com um milhão de

clientes portugueses, teremos certamente muitos registos com a mesma cidade. Se a coluna for ordenada podemos por exemplo aplicar o algoritmo RLE (*Run Lenght Encoding*) reduzindo drasticamente os dados, neste caso, a umas poucas dezenas de valores;

- ***Joins* Invisíveis (*Invisible Joins*)** - Optimização proposta em [7], com a finalidade de reduzir os joins necessários entre tabelas de factos e dimensões num esquema em estrela. É uma técnica de materialização tardia de joins [9] que funciona reescrevendo os *joins* como predicados sobre as chaves estrangeiras nas tabelas de factos. Segundo os autores esta técnica melhora a performance em cerca de 50%.

Muitos do actuais sistemas *column-oriented* já implementam este tipo de optimizações.

2.5 Implementações *Column-Oriented*

Segue-se uma referência aos principais sistemas *column-oriented* existentes na actualidade:

SISTEMAS COMERCIAIS

- **Vertica** [10] - Este sistema surgiu da comercialização do C-Store (ver em sistemas open-source), e tem verificado um grande crescimento. Além da arquitectura *column-oriented*, que permite consultas 50x-1000x mais rápidas através da eliminação de IO de disco (segundo a informação apresentada no site), apresenta uma arquitectura MPP (*Massive Paralell Processing*), utiliza compressão agressiva podendo reduzir os custos de armazenamento até 90%, fornece alta disponibilidade automática, entre as principais características;

- **VectorWise** [11] - Utiliza uma nova tecnologia no processamento das consultas que permite atingir todo o potencial dos modernos micro-processadores e das arquitecturas de memória, denominada *Vectorização*. Este motor, designado por X100 [12], combina um processador de consultas de altíssima performance e uma arquitectura de storage que vai muito além das outras implementações *column-oriented*. Resultou do *spin out* do projecto *open-source* MonetDB;

- **ParAccel** [13] - Segundo o site, e passo a citar "*The Fastest, Simplest Data Warehousing on the Planet. Period.*". Além da arquitectura de coluna disponibiliza alta performance nos carregamentos e *updates*, MPP em *shared-nothing*, compressão adaptativa, compilação de *queries* analíticas e uma aproximação *schema-neutral* que garante a melhor performance sobre dados normalizados ou de-normalizados, tais como os esquemas em estrela;

- **Sybase IQ** [14] - Um produto da Sybase, que anuncia ser 10x-100x mais rápido que as bases de dados transaccionais convencionais, suportar consultas analíticas pesadas, um número elevado de utilizadores em simultâneo, uma redução de até 70% do volume de armazenamento graças à compressão utilizada, e com custos de gestão reduzidos. Este SGBD foi um dos pioneiros a nível comercial na utilização de uma arquitectura *column-oriented*;

- **Infobright Enterprise Edition (IEE)** [15] - Desenhado de raiz para responder a consultas *ad-hoc*, complexas, sobre grandes volumes de dados, eliminando a necessidade de construir modelos de dados físicos específicos para a obtenção de melhor performance. As suas componentes *Knowledge Grid* e *Integrated Optimizer* garantem uma gestão eficiente dos dados guardados em formato comprimido. A Infobright também disponibiliza uma versão open-source desta tecnologia, a *Infobright Community Edition (ICE)*.

- **Oracle Exadata Hybrid Columnar Compression** [16] - A mais recente "arma"da Oracle, a Exadata, contém agora um módulo especifico para o armazenamento em coluna. Na realidade esta tecnologia utiliza um armazenamento híbrido (linha e coluna) retirando o melhor dos dois mundos (segundo a Oracle).

SISTEMAS OPEN-SOURCE

- **C-Store** [17] - Este foi um dos projectos pioneiros que motivou o recente interesse pela arquitectura *column-oriented* e resultou da colaboração das universidades do MIT, Yale, Brandeis University, Brown University, e UMass Boston. É um SGBD relacional optimizado para leitura que guarda os dados coluna a coluna, que guarda conjuntos de projecções com sobreposição, utiliza uma codificação e empacotamento dos dados muito particular não só em disco mas também em memória durante o processamento das consultas, e uma implementação nada tradicional das transacções que garante alta disponibilidade e isolamento para transacções de leitura, são algumas das suas principais características. O projecto encerrou em Outubro de 2006 com o lançamento da versão 0.2, e continuou numa perspectiva comercial de nome *Vertica*, conforme já referido.

- **MonetDB** [18] - Desenvolvido por uma equipa de investigadores do *Centrum Wiskunde & Informatica (CWI)* é um SGBD para aplicações de alta performance em *Data Mining, GIS, XML Query, text and multimedia retrieval*. Apresenta inovações nas várias camadas do SGBD tais como um modelo de armazenamento baseado em fragmentação vertical, um *query-engine* optimizado para os modernos CPU's, índices automáticos e auto-optimizados, optimização de queries em tempo de execução e uma arquitectura de software modular.

- **Ingres VectorWise** [19] - Colaboração entre a Ingres e a VectorWise com o objectivo de integrar o motor VectorWise no SGBD open-source Ingres já existente, garantindo alta performance no modelo de negócio aberto do Ingres. Algumas

das principais características incluem: processamento baseado em vectores (vectorizado), utilização de cache *on-chip* para processar dados, armazenamento em coluna *updatable*, compressão automática optimizada, compressão baseada em vectores e indexação de armazenamento automática.

- **Infobright Community Edition (ICE)** [20] - Versão comunitária da Infobright. Combina a arquitectura *column-oriented* com a arquitectura *Knowledge Grid* para proporcionar um SGBD auto-gerido e auto-optimizado para sistemas analíticos. Criado com base no MySQL permite carregar e questionar grandes quantidades de dados facilmente e com menor esforço.

- **LucidDB** [21] - Segundo o site é o primeiro e único SGBDR construído especificamente para *data warehousing* e *business intelligence*. Utiliza tecnologias como o armazenamento coluna a coluna, índices *bit-map*, agregação *hash join*, e *page-level multiversioning*.

Neste estudo foram utilizados os sistemas MonetDB e Infobright Community Edition (ICE).

2.6 Simulação em sistemas *Row-Oriented*

Alguns investigadores estudaram a possibilidade de implementar um sistema *column-oriented* sobre os sistemas tradicionais *row-oriented*, na tentativa de perceber a necessidade de mudar de tecnologia, algo que oferece sempre alguma resistência. Existem algumas técnicas que permitem essa simulação, no entanto, conforme concluíram Abadi et al. [8], o resultado é geralmente pobre. Das principais técnicas destacam-se:

- **Particionamento Vertical (*Vertical Partitioning*)** - Vertical no sentido de coluna, ou seja, uma tabela com N atributos é particionado em N tabelas com dois campos, o atributo original e uma chave artificial que representa a posição desse valor e que será igual para todos os valores do mesmo tuplo nas várias tabelas a criar, e serve para o processo de reconstrução do tuplo original. Desta forma apenas as tabelas correspondentes aos atributos necessários para resolução da consulta serão lidas.

- **Planos à base de índices (*Index-only Plans*)** - A existência de índices para todas as colunas utilizadas numa consulta torna possível a resolução da mesmo consultando apenas esses índices, ou seja, o plano de consulta é realizado com base apenas nos índices já que estes contêm toda a informação necessária;

- **Vistas Materializadas (*Materialized Views*)** - Esta técnica supõe a criação de uma vista materializada para cada consulta necessária, de forma a prover apenas os dados necessários. Apresenta bom desempenho no entanto leva à necessidade de bastante espaço de armazenamento.

2.7 Outros estudos comparativos

Terminamos este capítulo com uma breve apresentação de estudos anteriores elaborados com o mesmo objectivo deste trabalho, ou seja, o de comparar a performance de sistemas de arquitectura *row-oriented* e de sistemas de arquitectura *column-oriented* para cargas de trabalho típicas de um *data warehouse*.

Halverson et al. [22] apresentam duas optimizações de armazenamento, *super-tuples* e *column-abstraction*, para sistemas *row-oriented*, e comparam essa implementação com o C-Store numa plataforma comum. Segundo os autores deste trabalho, a comparação do C-Store com os sistemas comerciais *row-oriented* (optimizados para um mix de leitura e escrita) esconde os benefícios relativos ao sistemas *row-oriented* e *column-oriented*. Verificaram vantagens tremendas do sistema *column-oriented* sobre o sistema *row-oriented* para cargas de trabalho que acedem a uma pequena fracção das colunas da tabela. Concluíram que a optimização *super-tuples* permite um ganho significativo de performance e que o *column-abstraction* pode ser utilizado de forma efectiva para reduzir as necessidades de armazenamento.

Harizopoulos et al.[7] compararam a performance de sistemas de leitura intensa de arquitectura *column-oriented* e *row-oriented*. Concluíram que os sistemas *column-oriented*, com um *prefetching* apropriado, conseguem quase sempre fazer uma melhor utilização da largura de banda do disco, do que os sistemas *row-oriented*, mas que num número limitado de situações a performance do CPU não é tão boa. Concluí ainda que determinadas vantagens tais como a possibilidade de operar directamente em dados comprimidos e o processamento vectorizado, torna os sistemas de base de dados *column-oriented* sistemas mais atractivos para os futuros sistemas analíticos.

Abadi et al. [8] compararam a performance do C-Store com diversas variantes de um sistema comercial, usando o *Star Schema Benchmark (SSB)*. Concluíram que a tentativa de simular um sistema *column-oriented* num sistema *row-oriented* (ver secção anterior) não resulta numa boa performance. Que os sistemas *column-oriented* processam os dados de forma mais eficiente, recorrendo a tecnologias tais como a materialização tardia, que melhora a performance num factor de 3, e a compressão, que melhora num factor de 2, em média.

Capítulo 3

Star Schema Benchmark

O *Star Schema Benchmark* (SSB) [23] foi desenhado para medir a performance de sistemas de gestão de bases de dados no suporte a aplicações de *data warehousing* clássicas, e o seu desenvolvimento foi baseado no TPC-H *benchmark* [24], com algumas alterações significativas.

Ao contrário do TPC-H, que apresenta um modelo de dados normalizado do tipo *snowflake*, o SSB apresenta um modelo de dados desnormalizado em estrela (*star schema*), que segundo Kimbal [25, 55-57] é melhor do que o *snowflake* em termos de usabilidade, e de performance.

O SSB é constituído por menos consultas que o TPC-H e tem menos restrições sobre as formas de *tunning* permitidas, não impondo restrições, por exemplo, na compressão das colunas, dando assim mais liberdade na implementação de novas tecnologias.

3.1 Modelo de Dados

O modelo de dados, representado na figura 3.1, é constituído por uma única tabela de factos, a tabela `LINEORDER`, que contém a informação sobre as encomendas. A chave da tabela é uma chave composta formada pelos atributos `ORDERKEY` e `LINENUMBER`. A tabela contém as chaves estrangeiras `CUSTKEY`, `PARTKEY`, `SUPKEY`, `ORDERDATE` e `COMMITDATE`, que fazem referência às respectivas tabelas de dimensão, nomeadamente `CUSTOMER`, `PART`, `SUPPLIER` e `DATE`. Outros atributos descrevem a prioridade, quantidade, preço, desconto, etc. É um modelo em estrela, próprio de um *data mart* desnormalizado.

O código SQL utilizado para gerar fisicamente este modelo nas respectivas bases de dados encontra-se listado nos anexos A (página 46) e B (página 49).

A cardinalidade de cada tabela é definida em função do *Scale Factor* (SF), ou factor de escala, que aparece na figura por de baixo da respectiva tabela. Este factor permite

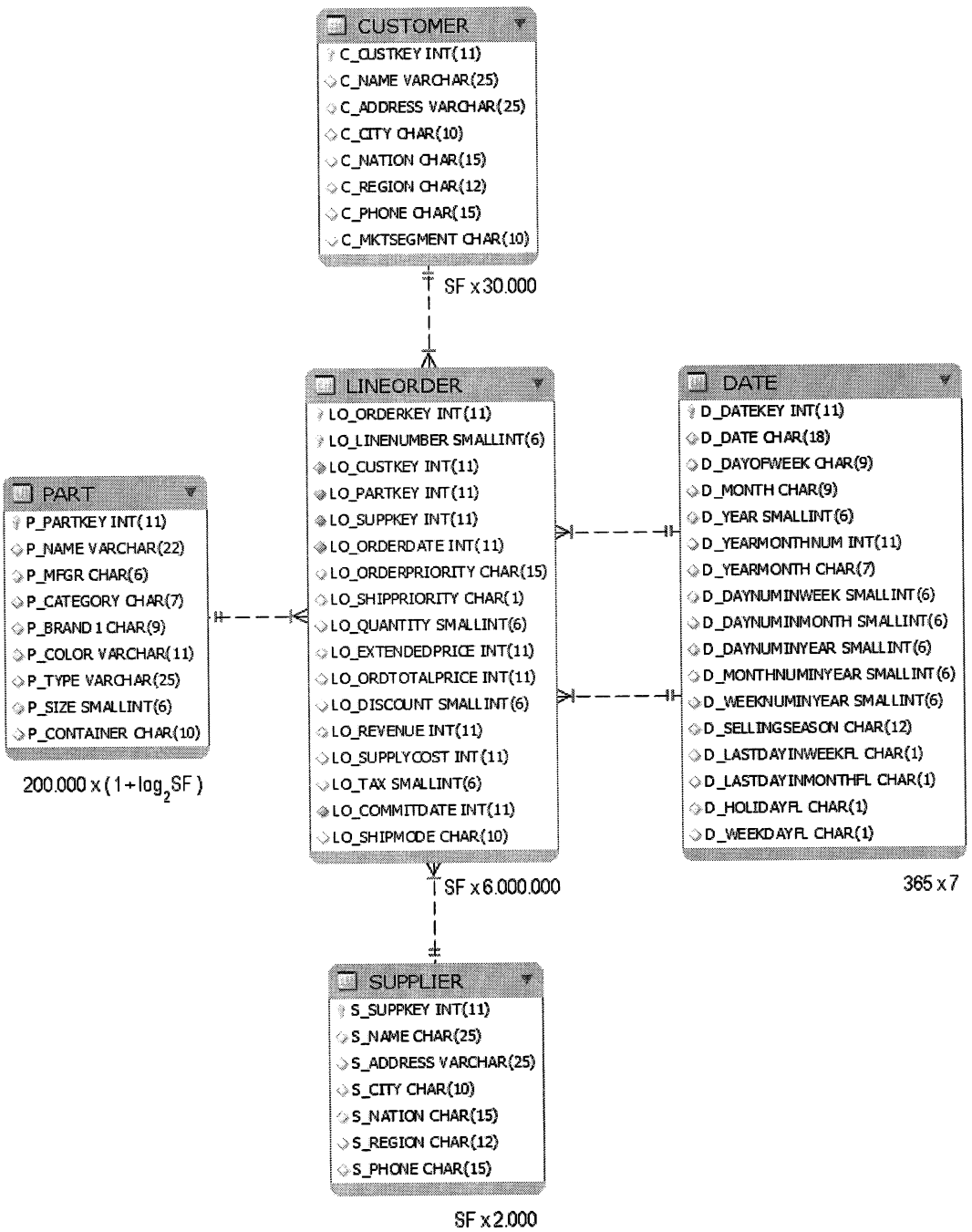


Figura 3.1: Modelo de Dados do SSB

escalar o tamanho do teste. Neste trabalho foram considerados os factores de escala 1, 5 e 10, correspondendo a uma cardinalidade na tabela `LINEORDER` de 6 milhões para o factor 1, 30 milhões para o factor 5, e de 60 milhões para o factor 10.

A tabela 3.1 apresenta a cardinalidade de todas as tabelas nos três factores de escala considerados.

SF	LINEORDER	CUSTOMER	PART	SUPPLIER	DATE
1	6.000.000	30.000	200.000	2.000	2.556
5	30.000.000	150.000	600.000	10.000	2.556
10	60.000.000	300.000	800.000	20.000	2.556

Tabela 3.1: Factores de escala e cardinalidade respectiva

3.2 Consultas

O SSB consiste em treze consultas, listadas no anexo C (página 51), divididas por quatro categorias, também designadas por voos (*query flights*):

Flight 1: Contém três consultas (Q1.1, Q1.2 e Q1.3), com restrição apenas numa dimensão, a data. Têm como objectivo medir o ganho do retorno (produto entre o `EXTENDEDPRICE` e o `DISCOUNT`) que pode ser obtido com diferentes valores de desconto, para diferentes quantidades, num determinado ano. Os resultados das consultas não são agrupados nem ordenados. As selectividades (percentagem de tuplos seleccionados pela consulta) na tabela LINEORDER para cada consulta são, respectivamente, $1.9 \times 10^{-2}, 6.5 \times 10^{-4}$ e 7.5×10^{-5}. Por exemplo, para o factor de escala 1, a consulta 1.1 irá devolver $6.000.000 \times (1.9 \times 10^{-2}) \approx 114$ mil tuplos.

Flight 2: Contém três consultas (Q2.1, Q2.2 e Q2.3) com restrição em duas dimensões. Calculam o retorno para uma determinada classe de produtos, numa região particular. Os resultados são agrupados por classe de produto e ano. As selectividades na tabela LINEORDER são, respectivamente, $8.0 \times 10^{-3}, 1.6 \times 10^{-3}$ e 2.0×10^{-4}.

Flight 3: Contém quatro consultas (Q3.1, Q3.2, Q3.3 e Q3.4) com restrição em três dimensões. Calculam o retorno numa determinada região num determinado período de tempo. Os resultados são agrupados por nação, nação do fornecedor e ano. As selectividades na tabela LINEORDER são, respectivamente, $3.4 \times 10^{-2}, 1.4 \times 10^{-3}, 5.5 \times 10^{-5}$ e 7.6×10^{-7}.

Flight 4: Contém três consultas (Q4.1, Q4.2 e Q4.3) com restrição em três dimensões. Calculam os ganhos (`REVENUE - SUPPYCOST`) agrupados por ano, nação e categoria (consulta 1) e agrupados por região e categoria (consultas 2 e 3). As

selectividades na tabela LINEORDER são, respectivamente, 1.6×10^{-2}, 4.5×10^{-3} e 9.1×10^{-5}.

As consultas foram escritas de forma a minimizar o efeito de cache, ou seja, a redução do número de acessos ao disco que são necessários numa consulta B, dado que anteriormente foi executada uma consulta A, devido à sobreposição dos acessos aos dados.

As consultas têm como objectivos garantir as funcionalidades de *cobertura funcional*, garantindo a realização das funções requeridas pelos utilizadores, e de *cobertura de selectividade*, disponibilizando consultas que devolvem desde um pequeno número de tuplos até um número elevado de tuplos.

3.3 Gerador de Dados

A especificação SSB disponibiliza um gerador de dados, o `dbgen`, que nos permite popular as tabelas do modelo em qualquer factor de escala. O código fonte do gerador, escrito na linguagem C, pode ser obtido no endereço `http://www.cs.umb.edu/~poneil/publist.html`.

Após correr o `makefile` é gerado o executável `dbgen`. Este aceita como parâmetros o factor de escala e a tabela (ou tabelas) a gerar, e cria os ficheiros de texto com os respectivos dados, separados pelo caractere | (*pipe*). Esta versão contém um pequeno *bug* que origina um *pipe* extra no final da linha. Antes de poder carregar os dados com o auxílio do comando `COPY FROM` torna-se necessário eliminar este caractere extra.

Capítulo 4

Metodologia e Testes

Conforme referido no capítulo introdutório, o objectivo deste estudo é a caracterização das situações em que o recurso a sistemas de gestão de bases de dados de arquitectura *column-oriented* é mais (ou menos) favorável comparativamente à utilização de sistemas de arquitectura *row-oriented*, quando utilizados com um *workload* típico de um *data warehouse*. Para tal serão efectuados testes, utilizando o *Star Schema Benchmark* (ver capítulo anterior) sobre quatro sistemas de gestão de bases de dados, dois de arquitectura *column-oriented* e dois de arquitectura *row-oriented*, sobre diferentes ambientes de execução, fazendo variar parâmetros tais como a RAM disponível ou o CPU, e considerando diferentes factores de escala.

Com a arquitectura *column-oriented* foram utilizados os sistemas *MonetDB* e *Infobright Comunity Edition (ICE)*, pelo facto de apresentarem grande potencial.

Com a arquitectura *row-oriented* consideraram-se os sistemas *MySQL (com o motor de armazenamento MyISAM)* e *PostgreSQL*, pelo facto de serem sistemas sólidos, largamente utilizados e reconhecidos pela comunidade.

Em cada um destes quatro sistemas foram criadas três bases de dados SSB, nos factores de escala 1, 5 e 10 (conforme referido em 3.1). Foram assim criadas 12 bases de dados distintas.

Sobre cada uma destas 12 bases de dados foram realizados 11 testes diferentes fazendo variar os parâmetros CPU, Disco, RAM disponível e Velocidade da RAM, o que totaliza 132 testes.

De forma a minimizar o efeito de possíveis flutuações na velocidade de processamento das consultas, cada teste foi repetido 3 vezes, tendo sido considerado como tempo de execução a média aritmética das três repetições. Assim, e dado que um teste é composto por 13 consultas, foi realizado um total de 5.148 consultas às bases de dados (132 testes $\times$ 13 consultas $\times$ 3 repetições).

Estes dados foram registados num *data warehouse* em que a tabela de factos regista uma medição, ou seja, o tempo de execução de uma consulta. As tabelas de dimensão

descrevem a própria consulta e o ambiente no qual foi ela executada (CPU, Disco, RAM, etc.).

Todos os testes foram executados numa estação de trabalho Linux 64 Bit. Nos anexos D (página 55) e E (página 56) encontra-se a especificação do hardware e software utilizado.

As secções seguintes apresentam a metodologia utilizada neste trabalho, a bateria de testes e por fim a descrição do *data warehouse* de resultados, que servirá de base para a análise apresentada no capítulo 5.

4.1 Metodologia

A metodologia seguida neste trabalho encontra-se representada na figura 4.1:

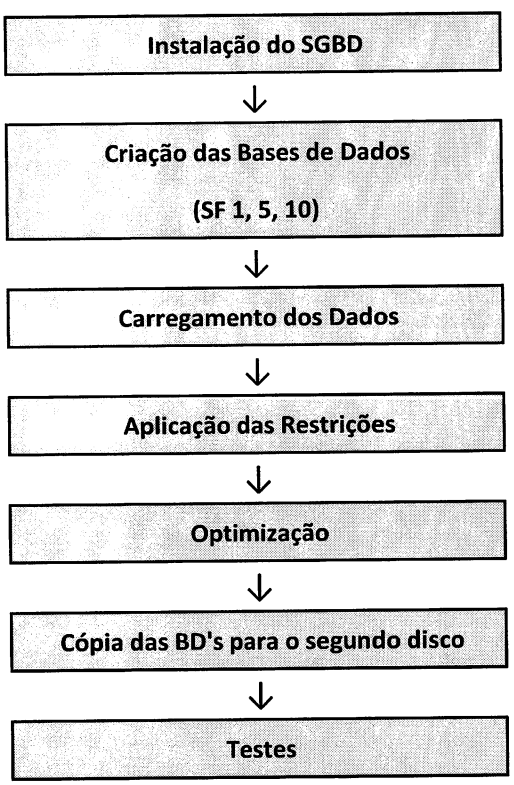


Figura 4.1: Metodologia

Passa-se a descrever cada uma das fases consideradas.

4.1.1 Instalação dos SGBD's

Após a instalação do Linux foram instalados os quatro SGBD's, nomeadamente o PostgreSQL, MySQL, MonetDB e ICE, processo que correu sem problemas.

4.1.2 Criação das Bases de Dados

Em cada um dos SGBD's foram criadas fisicamente as três bases de dados nomeadas SSB1, SSB5 e SSB10, correspondentes ao SSB com factores de escala 1, 5 e 10, respectivamente.

De seguida foi criado o modelo de dados sem restrições de chaves primárias ou estrangeiras de forma a minimizar o tempo de carregamento dos dados. No anexo A (página 46) apresenta-se o *Data Definition Language* (DDL) utilizado para criar o modelo de dados SSB, comum aos vários SGBD's.

4.1.3 Carregamento dos dados

No carregamento dos dados de teste gerados pelo gerador `dbgen` (ver secção 3.3) foi utilizado o comando `COPY`, da linguagem SQL. Em baixo apresenta-se o exemplo de carregamento da tabela `customer` no caso do PostgreSQL:

```
COPY CUSTOMER FROM 'customer.tbl' WITH DELIMITER AS '|';
```

4.1.4 Aplicação das restrições

Dado o elevado volume de dados é preferível efectuar o carregamento dos dados sem ter os índices criados, evitando assim a sua permanente actualização durante a carga. Por esta razão a criação das chaves primárias e chaves estrangeiras foi adiada para esta fase.

Como exemplo o anexo B (página 49) apresenta o DDL utilizado para criar as restrições no PostgreSQL.

4.1.5 Optimização

Antes de executar as consultas procedeu-se à optimização das bases de dados, evocando os respectivos optimizadores de forma a actualizar as estatísticas e assim garantir um bom plano de execução, nomeadamente:

PostgresSQL Comando `VACUUM ANALYZE`;

MySQL e ICE Comando `ANALYZE TABLE` sobre cada uma das tabelas do modelo;

MonetDB Não necessita. Este sistema tem um optimizador automático que liberta o utilizador desta função.

4.1.6 Cópia para segundo disco

Um dos casos de teste é a execução das consultas num disco mais lento. Esse disco foi montado no *file system* do Linux e foi feita uma cópia dos *datafiles* (ficheiros da base de dados) para esse disco.

Foram assim preparadas as 12 bases de dados (na realidade 24 considerando os dois discos) seguindo-se a fase dos testes.

4.1.7 Testes

Um teste consiste na execução das treze consultas do *Star Schema Benchmark*, por ordem, numa base de dados. Cada teste foi realizado num ambiente de execução distinto, fazendo variar os parâmetros do ambiente.

Na realização de cada teste foi seguida a seguinte metodologia:

1. Instalar o Hardware associado ao teste, nomeadamente o CPU e a RAM;
2. Configurar a BIOS do computador relativamente à velocidade da RAM (o CPU e a RAM são automaticamente detectados);
3. Montar no *file system* do sistema operativo o segundo disco, caso necessário;
4. Configurar a base de dados, via *scripts* de inicialização, para utilizar apenas uma determinada quantidade de RAM disponível;
5. Configurar a base de dados para utilizar os *data files* correctos, tendo em conta a base de dados a utilizar;
6. Configurar a ligação à BD e lançar o teste:
   (a) Fazer o *Startup* das bases de dados SSB1, SSB5 e SSB10;
   (b) Esperar 1 minuto (deixar o sistema estabilizar);
   (c) Executar as consultas e registar os respectivos tempos de execução;
   (d) Fazer o *Shutdown* das bases de dados SSB1, SSB5 e SSB10.
   (e) Repetir os quatro passos anteriores mais duas vezes, de forma o obter 3 medições.

Teste #	CPU	Disco	RAM (Gb)	RAM (MhZ)
1	Dual Core	SATA II	3	800
2	Dual Core	SATA II	7	800
3	Dual Core	SATA II	1	800
4	Dual Core	SATA II	3	677
5	Dual Core	SATA II	3	1066
6	Dual Core	ATA	3	800
7	Celeron	SATA II	3	800
8	Celeron	SATA II	7	800
9	Celeron	SATA II	1	800
10	Celeron	SATA II	3	677
11	Celeron	ATA	3	800

Tabela 4.1: Bateria de Testes

4.2 Bateria de Testes

A tabela 4.1 apresenta a bateria de testes considerada. Inicialmente considerou-se a realização de todas as combinações possíveis dos vários parâmetros, no entanto, e por questões de tempo, decidiu-se restringir a um conjunto menor, desde que permitisse a comparação de resultados. Assim, na análise de um determinado parâmetro, por exemplo a velocidade da RAM, serão considerados apenas parte dos testes realizados, neste caso os testes #1, #4 e #5, ou seja, aqueles que variam apenas o parâmetro em estudo.

4.3 *Data Warehouse* de resultados

Conforme já foi referido, todos os resultados foram registados num DW formado por um modelo dimensional em estrela, conforme figura 4.2. Neste modelo a tabela de factos armazena o tempo de execução de cada consulta sendo as dimensões utilizadas para descrever o ambiente no qual ela foi executada.

Passamos a descrever cada uma das tabelas do modelo assim como os valores considerados neste estudo, omitindo as chaves primárias uma vez que são artificiais e irrelevantes para a compreensão do modelo.

4.3.1 Dimensão Processador

Na tabela 4.2 representa-se a dimensão Processador, descrita pelas seguintes propriedades:

Nome do Processador - Referência do processador, indicando marca e modelo;

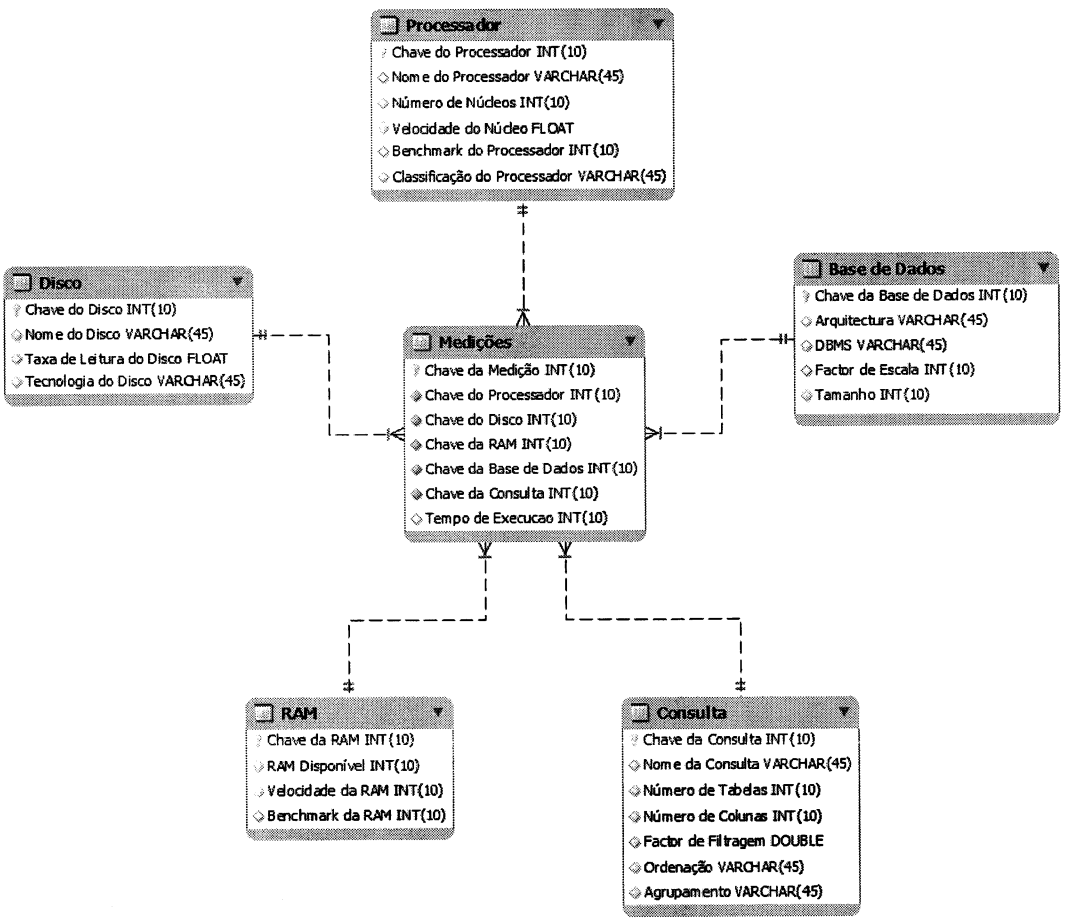


Figura 4.2: Modelo dimensional do *data warewouse*

Número de Núcleos - Um ou mais no caso dos modernos processadores multi-nucleo;

Velocidade do Núcleo (Gh) - Velocidade de relógio da RAM;

Benchmark do Processador - Medida de performance do processador segundo [26];

Classificação do Processador - Classe (ou gama) *low-end/high-end* segundo [26].

Foram considerados dois processadores: o *Intel Pentium Dual Core* (mais recente, de dois núcleos, e da gama alta) e o *Intel Celeron* (mais antigo, com um núcleo, e da gama baixa).

Nome do Processador	Número de Núcleos	Velocidade do Núcleo (Gh)	Benchmark do Processador	Classificação do Processador
Intel Pentium Dual Core	2	2.8	1913	High-End
Intel Celeron	1	1.8	210	Low-End

Tabela 4.2: Dimensão Processador

4.3.2 Dimensão Disco

Na tabela 4.3 representa-se a dimensão Disco, descrita pelas seguintes propriedades:

Nome do Disco - Referência do disco, indicando marca e modelo;

Taxa de Leitura do Disco - Medida da performance de leitura do disco obtida com o utilitário `hdparm -t` do sistema operativo;

Tecnologia do Disco - Tecnologia do BUS;

Foram considerados dois discos: o *Western Digital 1Tb* (mais recente, SATA II, e com uma taxa de 98.20 Mb/segundo) e o *Maxtor 80Gb* (mais antigo, ATA, e com uma taxa de leitura de 55.48 Mb/segundo).

Nome do Disco	Taxa de Leitura do Disco (Mb/seg)	Tecnologia do Disco
Western Digital 1Tb Caviar Black	98.20	SATA II
Maxtor ATA 80Gb	55.48	ATA

Tabela 4.3: Dimensão Disco

4.3.3 Dimensão RAM

A tabela 4.4 apresenta a dimensão RAM, descrita pelas seguintes propriedades:

RAM Disponível (Mb) - Memória RAM efectivamente disponível para o SGBD no momento da execução do teste;

Velocidade da RAM (Mhz) - Velocidade de relógio;

Benchmark da RAM - Medida da performance segundo [27].

Foram realizados testes com três valores para a RAM disponível, nomeadamente 1Mb, 3Mb e 7Mb. Com 3Mb foram ainda realizados mais dois testes, um com uma memória mais lenta a 677Mhz, e outro com uma memória mais rápida a 1066Mhz.

4.3.4 Dimensão Base de Dados

A tabela 4.5 apresenta a dimensão Base de Dados, descrita pelas seguintes propriedades:

Arquitectura - Arquitectura *row-oriented* ou *column-oriented*;

SGBD - Sistema de Gestão de Base de Dados;

Factor de Escala - Dimensão da base de dados;

Tamanho (Kb) - Espaço em disco ocupado pela base de dados.

RAM Disponível (Mb)	Velocidade da RAM (Mhz)	Benchmark da RAM
1024	800	3877
3072	800	3877
7168	800	3877
3072	677	3584
3072	1066	4315

Tabela 4.4: Dimensão RAM

Arquitectura	DBMS	Factor de Escala	Tamanho (Kb)
column-oriented	MonetDB	1	507.176
column-oriented	MonetDB	5	2.509.584
column-oriented	MonetDB	10	4.763.284
column-oriented	ICE	1	102.244
column-oriented	ICE	5	517.816
column-oriented	ICE	10	1.046.232
row-oriented	PostgreSQL	1	836.834
row-oriented	PostgreSQL	5	4.106.720
row-oriented	PostgreSQL	10	8.153.312
row-oriented	MySQL	1	833.360
row-oriented	MySQL	5	4.127.604
row-oriented	MySQL	10	8.216.212

Tabela 4.5: Dimensão Base de Dados

Conforme referido anteriormente foram realizados testes com quatro sistemas de bases de dados, nas duas arquitecturas em estudo, em três escalas diferentes.

4.3.5 Dimensão Consulta

Esta dimensão descreve as treze consultas que compõem o *Star Schema Benchmark*. Os seus atributos são:

Nome da Consulta - Identificação da consulta;

Número de Tabelas - Número de tabelas acedidas pela consulta;

Número de Colunas - Número de colunas acedidas pela consulta;

Factor de Filtragem - Selectividade da consulta;

Ordenação - Indica se os resultados da consulta são ordenados ou não (se foi utilizada a claúsula `ORDER BY` no `Select`);

Agrupamento - Indica se os resultados da consulta são agrupados ou não (se foi utilizada a claúsula `GROUP BY` no `Select`);.

Nome da Consulta	Numero de Tabelas	Numero de Colunas	Factor de Filtragem	Ordenação	Agrupamento
Q1.1	2	6	0.0194805	Sem Ordenação	Sem Agrupamento
Q1.2	2	6	0.00064935	Sem Ordenação	Sem Agrupamento
Q1.3	2	6	0.000075	Sem Ordenação	Sem Agrupamento
Q2.1	4	11	0.008	Com Ordenação	Com Agrupamento
Q2.2	4	10	0.0016	Com Ordenação	Com Agrupamento
Q2.3	4	10	0.0002	Com Ordenação	Com Agrupamento
Q3.1	4	12	0.034285714	Com Ordenação	Com Agrupamento
Q3.2	4	12	0.001371429	Com Ordenação	Com Agrupamento
Q3.3	4	10	0.0000548571	Com Ordenação	Com Agrupamento
Q3.4	4	11	0.000000761905	Com Ordenação	Com Agrupamento
Q4.1	5	15	0.016	Com Ordenação	Com Agrupamento
Q4.2	5	16	0.004571429	Com Ordenação	Com Agrupamento
Q4.3	5	16	0.0000914286	Com Ordenação	Com Agrupamento

Tabela 4.6: Dimensão Consulta

4.3.6 Tabela de Factos Medições

A tabela de factos regista o tempo de execução de cada consulta, devidamente enquadrada pelas chaves estrangeiras para as respectivas dimensões.

Após a realização dos testes e do carregamento deste *data warehouse* procedeu-se então à análise dos dados, cujos resultados são descritos no próximo capitulo.

Capítulo 5

Análise de resultados

O processo de análise envolveu a exploração dos dados do *data warehouse*, usando as funcionalidades da folha de cálculo Excel, nomeadamente as tabelas e os gráficos *Pivot*, e técnicas de *Data Mining*.

Tendo em conta que existem múltiplas dimensões, cada uma com várias variáveis de análise, a exploração centrar-se-á sobre as mais óbvias e de senso-comum, visto ser um trabalho manual e exaustivo, no qual o utilizador interage com o *data warehouse* procurando respostas para questões previamente formuladas.

As técnicas de *Data Mining* permitem uma procura automática de conhecimento, não havendo a necessidade de pré-conceber as questões. Algoritmos de indução de regras e árvores de decisão, aqui utilizados, permitem a descoberta automática de relações, sem a necessidade de interacção com o utilizador.

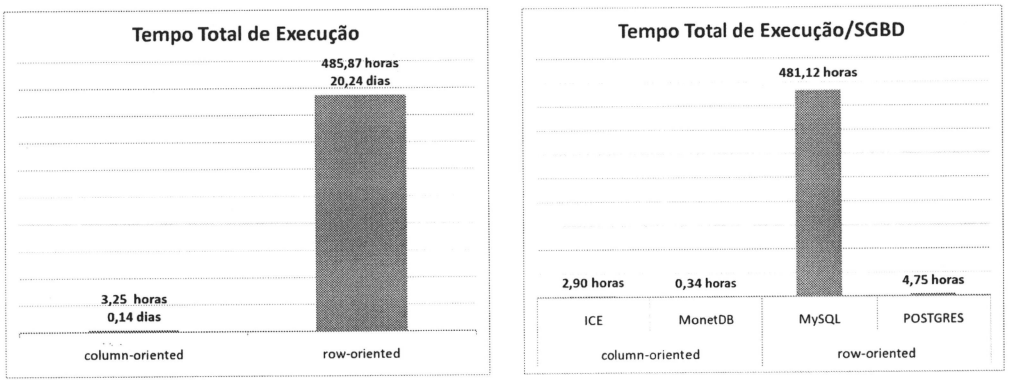
Considerando o objectivo deste trabalho em determinar a influência das variáveis na performance dos sistemas das duas arquitecturas consideradas, esta análise estuda por arquitectura os resultados de cada variável. Também se apresentam alguns resultados por arquitectura/SGBD, dando assim uma carácter mais prático a este estudo.

5.1 Tempo total de execução

Somando o tempo total necessário para executar as consultas nos vários ambientes de execução e agrupando os resultados por arquitectura e SGBD obtemos os resultados apresentados na figura 5.1.

Pela figura 5.1 a) podemos verificar que a diferença é muito significativa, sendo o tempo total na arquitectura *row-oriented* (485,87 horas) cerca de 150 vezes o tempo total na arquitectura *column-oriented* (3,25 horas).

Fazendo o chamado *drill down*, ou seja, uma análise mais detalhada, e considerando



a) b)

Figura 5.1: Tempo total de execução

agora a variável SGBD, verifica-se pela figura 5.1 b) que esta diferença é provocada essencialmente devido ao desempenho do MySQL.

Comparando os melhores SGBD's dentro de cada arquitectura verifica-se que o MonetDB é cerca de 14 vezes mais rápido que o PostgreSQL. Fazendo a mesma análise para os piores constata-se que o ICE é cerca de 166 vezes mais rápido que o MySQL. Em ambos os casos há um performance claramente superior na arquitectura *column-oriented.*

5.2 Processador

A figura 5.2 apresenta o tempo total de processamento agrupado por arquitectura / processador (foram utilizados todos os casos de teste com a excepção do número 5, uma vez que o processador Celerom utilizado não suporta memória de 1066Mhz).

A análise da figura sugere-nos que a *column-oriented* é mais sensível à performance do processador uma vez que se verificou uma redução de 41% no tempo total de execução na utilização do processador da gama alta. No caso da arquitectura *row-oriented* essa redução foi apenas de 10%. A arquitectura *column-oriented* faz uma utilização mais intensiva do processador, logo tem mais a ganhar com a performance deste.

Analisando por SGDB (figura 5.3) confirmam-se os resultados, verificando-se uma melhoria de performance de 42% no ICE e de 33% no MonetDB, e de apenas 10% no MySQL e de 11% no PostgreSQL.

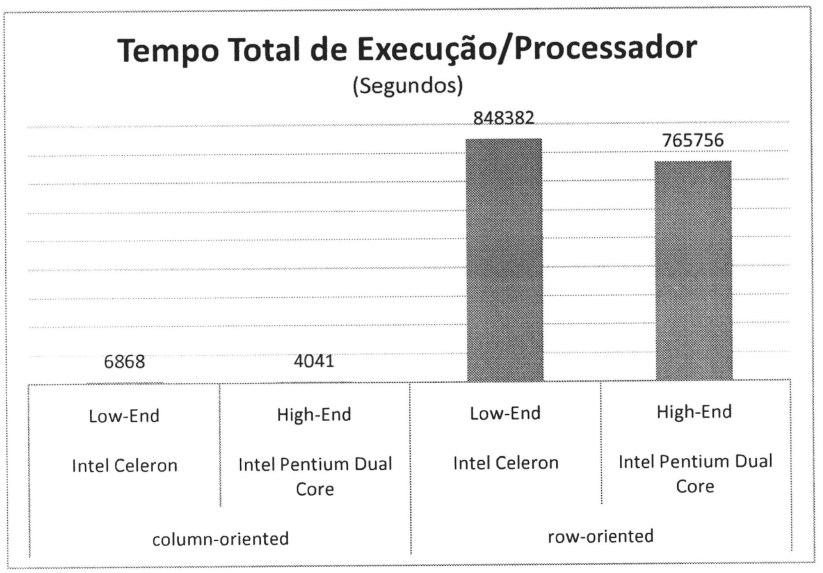


Figura 5.2: Tempo total por processador

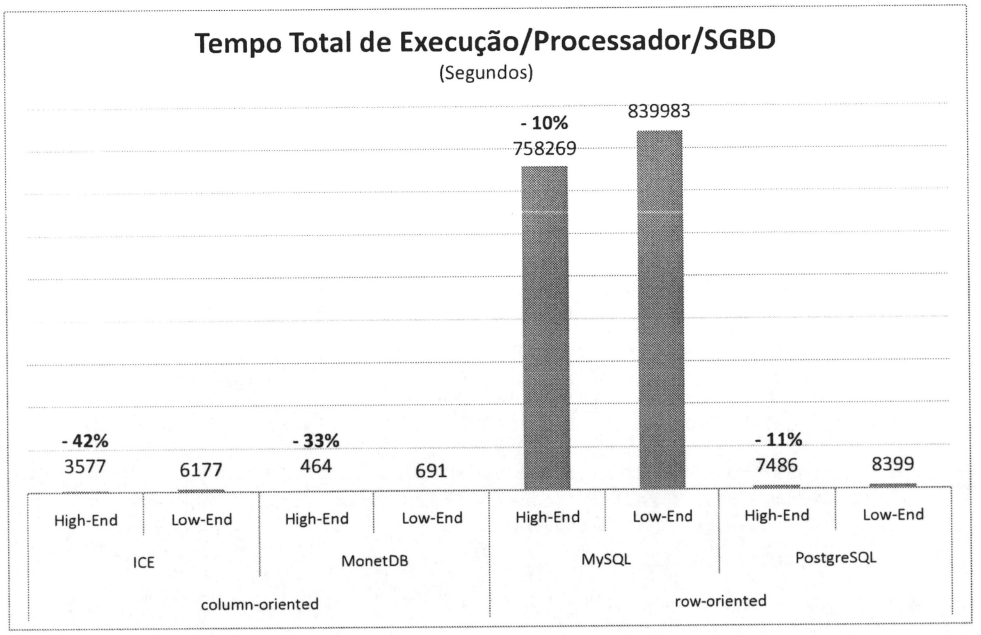


Figura 5.3: Tempo total por processador/SGBD

5.3 Disco

Na análise desta variável foram considerados os testes #1, #6, #7 e #11 que se referem a testes realizados nos dois discos. Agrupando o tempo total de execução por disco (figura 5.4) verifica-se que no caso da arquitectura *column-oriented* há um decréscimo de 2,4% no tempo total de execução pelo facto de utilizar-mos o disco SATA II (cerca de 1,77 vezes mais rápido do que o disco ATA). Esse descéscimo é bastante mais acentuado na arquitectura *row-oriented* com um valor de 16%. Isto sugere que estes sitemas são mais dependentes da performance do disco, tendo mais a ganhar com a utilização de um disco mais rápido.

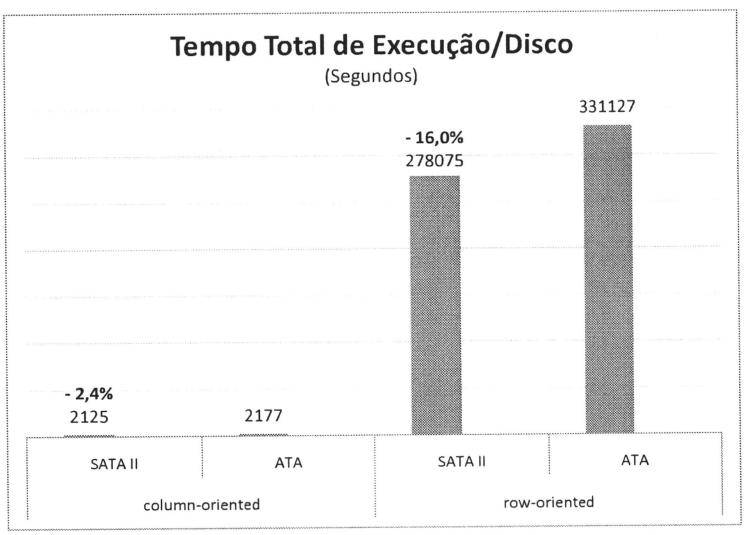


Figura 5.4: Tempos de execução/Disco

5.4 RAM disponível

Relativamente à RAM disponível para o SGBD no momento de execução do teste as tabelas 5.1 e 5.2 agrupam estes dados nas arquitecturas *column-oriented* e *row-oriented*, respectivamente. Nesta análise foram considerados apenas os testes #1, #2, #3, #7, #8 e #9.

RAM (Mb)	Tempo total execução	% do total	Ganho face a 1024Mb
1024	2383	0,24%	-
3072	2125	0,21%	1,12
7168	2060	0,20%	1,16

Tabela 5.1: Tempo de execução na arquitectura *column-oriented* por RAM disponível

RAM (Mb)	Tempo total execução (seg)	% do total	Ganho face a 1024Mb
1024	716623	71,09%	-
3072	278074	27,57%	2,58
7168	6810	0,68%	105,23

Tabela 5.2: Tempo de execução na arquitectura *row-oriented* por RAM disponível

Pela análise da tabela 5.1 podemos deduzir que os sistemas *column-oriented* são pouco sensíveis à quantidade de RAM disponível, e.g., o aumento da RAM de 1024Mb para 7168Mb proporcionou apenas um ganho de 1,16 vezes.

Quanto à arquitectura *row-oriented* (tabela 5.2) verifica-se um ganho muito significativo quando disponibilizamos mais memória ao SGBD. Um ganho de 2,58 vezes ao passar de 1024Mb para 3072Mb, e um ganho de 105,23 vezes ao passar para 7168Mb. A manter-se esta tendência um sistema *row-oriented* poderá atingir níveis de performance equivalentes a um sistema *column-oriented* através do aumento da RAM.

5.5 Velocidade da RAM

A análise da velocidade da RAM resume-se nas tabelas 5.3 e 5.4. Em ambas as arquitecturas verifica-se que a influência da velocidade da RAM é pouco relevante na performance dos SGBD's. O ganho na utilização de uma memória rápida a 1066Mhz, em alterativa à utilização de uma memória mais lenta a 677Mhz, é de apenas 1,023 vezes na arquitectura *column-oriented* de de apenas 1,010 vezes na arquitectura *row-oriented*.

RAM (Mhz)	Tempo total execução (seg)	Ganho face a 677Mhz
677	792	-
800	784	1,010
1066	774	1,023

Tabela 5.3: Velocidade da RAM na arquitectura *column-oriented*

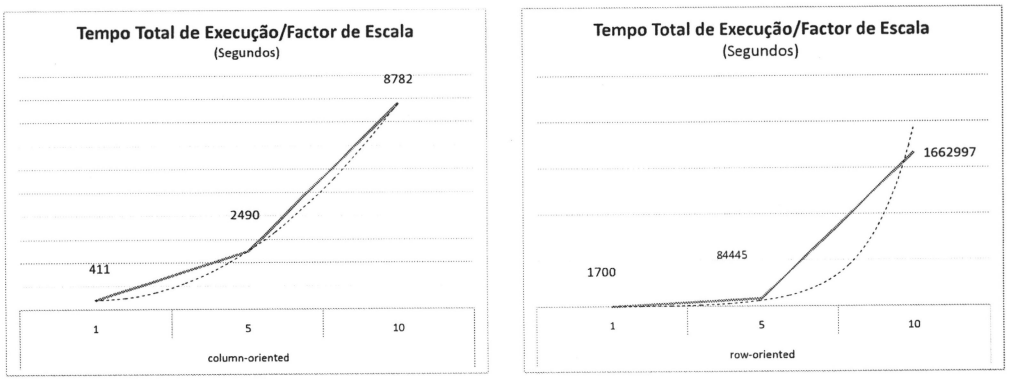
RAM (Mhz)	Tempo total execução (seg)	Ganho face a 677Mhz
677	136456	-
800	145913	0,935
1066	135005	1,010

Tabela 5.4: Velocidade da RAM na arquitectura *row-oriented*

Nesta análise foram considerados apenas os testes #1, #4 e #5.

5.6 Factor de escala

Analisando a resposta relativamente ao factor de escala obtemos o resultado da figura 5.5.



a) b)

Figura 5.5: Tempo de execução/Factor de Escala

Podemos verificar na arquitectura *column-oriented* (figura 5.5 a) que o tempo de execução parece apresentar uma tendência de crescimento polinomial em função do factor de escala. Já na arquitectura *row-oriented* essa tendência parece ser exponencial. Isto sugere-nos que para factores de escala superiores, e.g. 50, o tempo de execução na arquitectura *row-oriented* pode ser proibitivo, o que revela uma má escalabilidade desta arquitectura. Considerando o SGBD (figura 5.6) confirma-se o resultado.

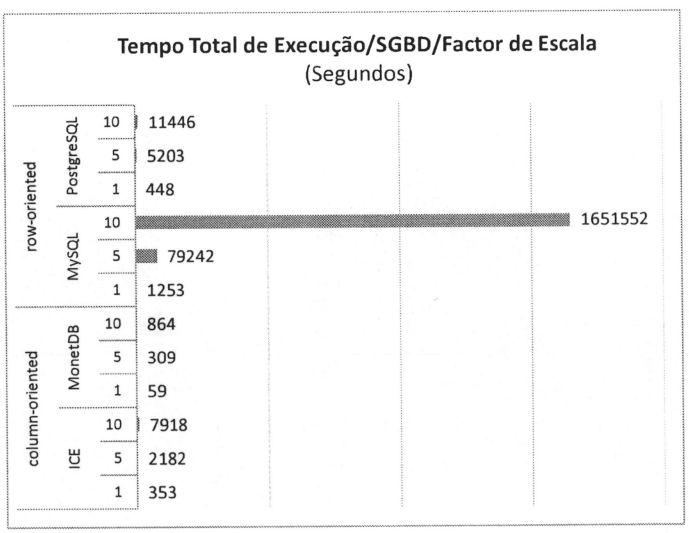


Figura 5.6: Tempo de execução/SGBD/Factor de Escala

5.7 Tamanho da BD

Analisando o tamanho das bases de dados obtemos o resultado da figura 5.7.

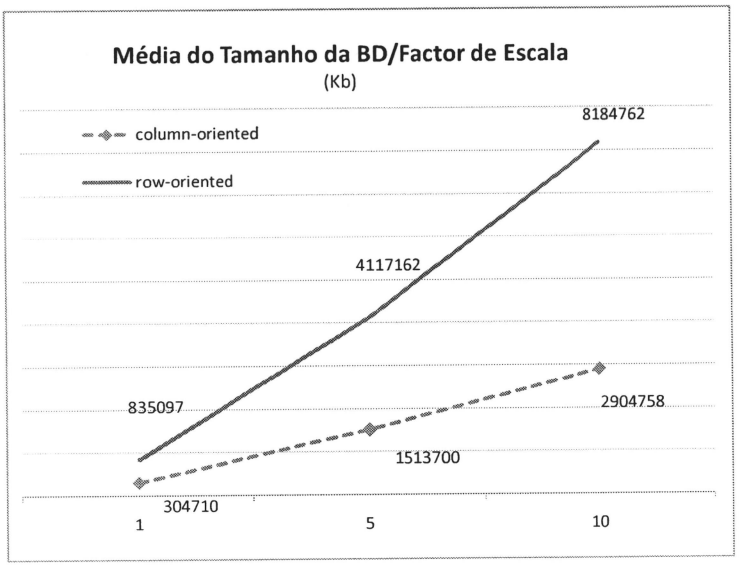


Figura 5.7: Tamanho da BD/Factor de Escala

Podemos verificar que em ambas as arquitecturas o crescimento é linear no entanto na arquitectura *row-oriented* este é mais acentuado. Verifica-se ainda que uma base de dados nesta arquitectura ocupa cerca de 2,75 vezes o espaço ocupado por uma base de dados na arquitectura *column-oriented*.

5.8 Tempo médio de execução das consultas

A tabela 5.8 apresenta o tempo médio de execução de cada uma das treze consultas, agrupado por arquitectura e SGBD. A coluna R/C apresenta a razão entre a média na arquitectura *row-oriented* e a média na arquitectura *column-oriented*, e.g., a consulta Q1.1 demora, em média, 4 vezes mais tempo a executar na arquitectura *row-oriented* do que na arquitectura *column-oriented* (razão 35,9/9).

Em todos os casos verifica-se um R/C superior a 1, em alguns casos muito superior, o que demonstra uma melhor performance da arquitectura *column-oriented* em todas as consultas do teste. A maior diferença regista-se nas consultas Q2.3 e Q2.2.

Consultas	Média	R/C	Consultas	Média	R/C	Consultas	Média	R/C
Q1.1	22,4		**Q2.3**	2069,8		**Q4.1**	2095,7	
column	9		*column*	6,4		*column*	37,4	
ICE	13,8		ICE	12		ICE	65,5	
MonetDB	4,1		MonetDB	0,8		MonetDB	9,3	
row	35,9	4	*row*	4133,3	645,8	*row*	4154	111,1
MySQL	31,3		MySQL	8229,5		MySQL	8262,5	
PostgreSQL	40,4		PostgreSQL	37,1		PostgreSQL	45,4	
Q1.2	16,3		**Q3.1**	2067,7		**Q4.2**	2112,2	
column	3,4		*column*	15,6		*column*	36,8	
ICE	5,6		ICE	28,1		ICE	68,9	
MonetDB	1,1		MonetDB	3,1		MonetDB	4,6	
row	29,2	8,6	*row*	4119,7	264,1	*row*	4187,7	113,8
MySQL	20,6		MySQL	8189,8		MySQL	8336,4	
PostgreSQL	37,9		PostgreSQL	49,6		PostgreSQL	39,1	
Q1.3	16		**Q3.2**	420,4		**Q4.3**	426,3	
column	3,1		*column*	6,9		*column*	19,1	
ICE	5		ICE	12,7		ICE	33,4	
MonetDB	1,2		MonetDB	1		MonetDB	4,7	
row	28,9	9,3	*row*	833,9	120,9	*row*	833,5	43,6
MySQL	20,3		MySQL	1630,6		MySQL	1629,5	
PostgreSQL	37,5		PostgreSQL	37,2		PostgreSQL	37,5	
Q2.1	1903,1		**Q3.3**	148,1				
column	13,1		*column*	11,1				
ICE	21,5		ICE	21,4				
MonctDB	4,7		MonetDB	0,7				
row	3793,2	289,6	*row*	285,2	25,7			
MySQL	7542,4		MySQL	533,5				
PostgreSQL	43,9		PostgreSQL	36,9				
Q2.2	1930,8		**Q3.4**	110,7				
column	7,1		*column*	8,2				
ICE	13,1		ICE	15,6				
MonetDB	1,2		MonetDB	0,7				
row	3854,5	542,9	*row*	213,2	26			
MySQL	7670,6		MySQL	389,2				
PostgreSQL	38,4		PostgreSQL	37,1				

Figura 5.8: Tempo médio de execução das consultas

5.9 Distribuição das consultas por classe

Cada medição de cada consulta foi classificada com um número de 0 a 19, calculado com o *Log*(Tempo de Execução), e discretizado para 20 intervalos de igual dimensão. A utilização da escala logarítmica justifica-se pelo facto da distribuição dos tempos de execução estar muito concentrada nos valores mais baixos. Esta classe representa o tempo de execução, discreto, numa escala logarítmica, logo, valores mais baixos significam melhores performances, e valores mais elevados, piores performances.

A tabela 5.9 apresenta a distribuição das classificações agrupada por consulta / arquitectura. Verifica-se que a melhor classificação, a classe 0, foi obtida apenas na arquitectura *column-oriented*, e mesmo a classe 1 foi obtida apenas em duas medições, das consultas Q3.3 e Q3.4, na arquitectura *row-oriented*. Por outro lado, todas a medições na arquitectura *column-oriented* foram classificadas abaixo da classe 13, sendo as classes 14 a 19 ocupadas apenas por consultas realizadas na arquitectura *row-oriented*.

Podemos verificar ainda que a classe mais frequente na arquitectura *column-oriented* é sempre inferior ou igual à classe mais frequente na arquitectura *row-oriented*.

		Classe																			
Consulta	Arquitectura	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Q1.1	column		1	1	9	9	11	8	11	11	5										
	row						1	19	2	7	18	17	2								
Q1.2	column		11	6	11	11	16		6	5											
	row					6	15	1	9	8	12	13	2								
Q1.3	column		11	6	11	11	16	2	8	1											
	row					12	10		11	6	12	13	2								
Q2.1	column		1	1	9	4	10	13	12	11	5										
	row						6	5	5	8	8	18	4	1	1				1	6	3
Q2.2	column	2	9		6	21	5	7	5	6	5										
	row					6	5		8	8	5	20	4						1	7	2
Q2.3	column	11		6	11	15	1	6	5	6	5										
	row					6	5	2	7	7	5	20	4						1	7	2
Q3.1	column		1	8	2	12	14	4	8	6	6	5									
	row						6	5	7	8	6	16	8						1	7	2
Q3.2	column	4	7		15	14	3	7	5	6	5										
	row					7	15	1	7	9	4	11	2				1	7	2		
Q3.3	column	11		6	8	14	5		6	5	9	2									
	row		1	6	4	7	14	1	2	3	5	11	2	1		2	7				
Q3.4	column	11		6	8	14	5		9	7	6										
	row		1	6	4	7	14	3	1	3	4	11	2	1		7	2				
Q4.1	column			1	3	7	1	20	5	7	9	8	5								
	row						6	5	7	8	6	18	6						1	6	3
Q4.2	column			6	5		12	15	4	12	1	2	6	3							
	row						6	5	8	8	5	19	5						1	6	3
Q4.3	column			6	5		17	10	10	6	5	6	1								
	row					6	13	3	9	7	5	11	2				1	7	2		

Figura 5.9: Distribuição das consultas por classe

5.10 Classificação com árvore de decisão

Utilizando o algoritmo de aprendizagem de árvores de decisão para classificação de dados nominais, fornecido pela aplicação *Rapid Miner*, e utilizando a classe determinada para cada medição, obtemos o resultado apresentado na figura 5.10. Foram incluídas apenas as variáveis mais influentes.

A árvore mostra-nos, em cada folha, a classe mais frequente considerando o caminho desde a raíz, o SGBD, até à folha, correspondendo aos diferentes ambientes de execução. Podemos verificar facilmente que a melhor classificação (valor de C mais baixo) está presente no MonetDB, factor de escala 1, e processador *high-end*. Por outro lado, a pior classificação vai para o MySQL, factor de escala 10 e com 1024Mb de RAM disponível.

5.11 Número de colunas acedidas pela consulta

Apesar das consultas serem diferentes, fazendo variar outros parâmetros que não apenas o número de colunas acedidas, genericamente constata-se (figura 5.11) que nos sistemas *column-oriented* o tempo de resposta degrada-se à medida que o número de colunas acedidas aumenta.

5.12 Ranking

Se considerar o tempo médio de execução de uma consulta dividido pelo factor de escala obtemos uma medida de Tempo mais genérica que não é especifica de uma consulta em particular, mas de todas, e não é especifico de uma factor de escala mas de todos. Compilando estes tempos e ordenando de forma ascendente obtemos a tabela 5.5. Facilmente se observa que a metade superior da tabela, onde estão os melhores resultados, é dominada pela arquitectura *column-oriented*, sendo a parte inferior dominada pela arquitectura *row-oriented*. Relativamente ao SGBD é também clara a separação, verificando-se a superioridade do MonetDB relativamente ao ICE, e do PostgreSQL relativamente ao MySQL.

Se observarmos a coluna do Processador no caso do MonetDB e do ICE há uma separação clara em duas áreas, a primeira com o processador *high-end* e a segunda com o processador *low-end*, revelando novamente uma maior influência do processador na performance desta arquitectura face à arquitectura *row-oriented* onde há uma intermitência entre *high-end* e *low-end*.

Relativamente à RAM disponível verifica-se também a sua maior influência no caso da arquitectura *row-oriented*, e, observando o tempo de 3690 do PostgreSQL, que o aumento da RAM pode levar a desempenhos que se aproximam dos valores registados

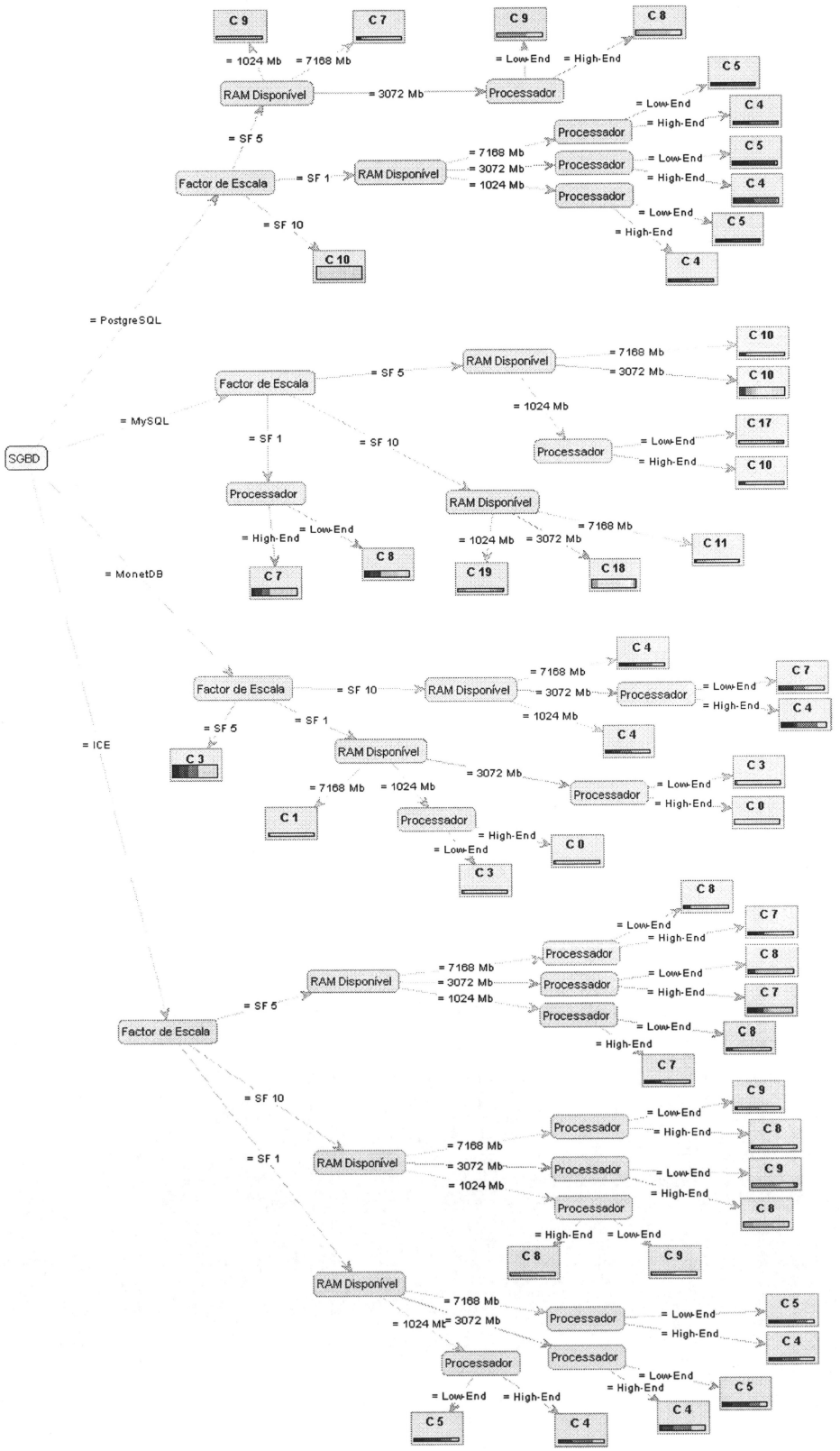
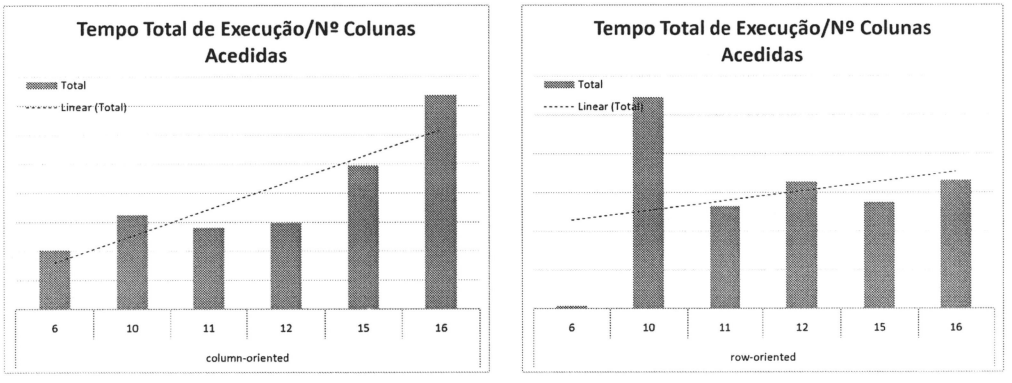


Figura 5.10: Árvore de decisão

Processador	Disco	RAM(Mb)	RAM(Mhz)	Arquit.	SGBD	Tempo
High-End	SATA II	7168	800	column	MonetDB	325
High-End	SATA II	3072	800	column	MonetDB	328
High-End	SATA II	3072	1066	column	MonetDB	337
High-End	SATA II	3072	677	column	MonetDB	379
High-End	ATA	3072	800	column	MonetDB	452
High-End	SATA II	1024	800	column	MonetDB	517
Low-End	SATA II	7168	800	column	MonetDB	535
Low-End	SATA II	3072	677	column	MonetDB	572
Low-End	SATA II	3072	800	column	MonetDB	579
Low-End	ATA	3072	800	column	MonetDB	600
Low-End	SATA II	1024	800	column	MonetDB	676
High-End	SATA II	7168	800	column	ICE	2701
High-End	SATA II	3072	1066	column	ICE	2718
High-End	SATA II	3072	800	column	ICE	2751
High-End	ATA	3072	800	column	ICE	2755
High-End	SATA II	3072	677	column	ICE	2761
High-End	SATA II	1024	800	column	ICE	2995
High-End	SATA II	7168	800	row	PostgreSQL	3690
Low-End	SATA II	7168	800	column	ICE	4590
Low-End	SATA II	3072	800	column	ICE	4650
High-End	SATA II	3072	1066	row	PostgreSQL	4698
Low-End	ATA	3072	800	column	ICE	4706
Low-End	SATA II	3072	677	column	ICE	4739
High-End	SATA II	3072	677	row	PostgreSQL	4974
High-End	SATA II	3072	800	row	PostgreSQL	5019
High-End	SATA II	1024	800	row	PostgreSQL	5081
Low-End	SATA II	1024	800	column	ICE	5170
Low-End	SATA II	7168	800	row	PostgreSQL	5210
Low-End	SATA II	3072	800	row	PostgreSQL	6133
Low-End	SATA II	1024	800	row	PostgreSQL	6194
Low-End	SATA II	3072	677	row	PostgreSQL	6269
High-End	SATA II	7168	800	row	MySQL	7378
High-End	ATA	3072	800	row	PostgreSQL	10007
Low-End	ATA	3072	800	row	PostgreSQL	10231
Low-End	SATA II	7168	800	row	MySQL	12776
Low-End	SATA II	3072	800	row	MySQL	340760
High-End	SATA II	3072	1066	row	MySQL	346383
High-End	SATA II	3072	677	row	MySQL	350273
Low-End	SATA II	3072	677	row	MySQL	374057
High-End	SATA II	3072	800	row	MySQL	374257
High-End	ATA	3072	800	row	MySQL	405797
Low-End	ATA	3072	800	row	MySQL	436679
High-End	SATA II	1024	800	row	MySQL	823014
Low-End	SATA II	1024	800	row	MySQL	1201870

Tabela 5.5: Ranking



a) b)

Figura 5.11: Tempo de execução/Número de Colunas Acedidas

para sistemas *column-oriented*.

De notar que a pior configuração, o último caso da tabela, é 3.698 vezes mais lento que a melhor configuração, e no entanto a diferença de custo monetário entre estas duas soluções anda na ordem das 3 a 4 centenas de euros. Não sendo o custo uma restrição maior torna-se claro qual a solução a adoptar.

5.13 Regras de classificação

Como é natural as bases de dados de menor dimensão têm melhores tempos de resposta, dada a menor quantidade de dados a processar. Tendo em conta este pressuposto podemos eliminar a variável factor de escala (que mede a dimensão da base de dados), reflectindo o seu valor numa nota atribuída a cada medição, e calculada da seguinte forma:

Para cada consulta/factor de escala foram calculados os valores:

$$\text{Min} = Min(Log(\text{Tempo de Execução}))$$

$$\text{Max} = Max(Log(\text{Tempo de Execução}))$$

Para cada medição foi calculada a nota com

$$Nota = 5 - Truncate\left(\frac{Log(\text{Tempo de Execução}) - \text{Min}}{(\text{Max} - \text{Min})/5}\right)$$

Obtendo-se assim uma nota, de 1 a 5, que mede a execução da consulta tendo em conta o factor de escala associado, e.g., se a medição da consulta Q2.1 no factor de escala 5 obtiver a nota 1 significa que foi a pior performance tendo em conta todas as outras medições dessa consulta nesse factor de escala.

Esta nota, juntamente com as principais variáveis, foram submetidas ao algoritmo *Apriori* afim de descobrir regras de classificação. Estas regras, do tipo $X \Rightarrow Y$, permitem determinar que, com um determinado grau de confiança, se X acontece então Y também acontece. Neste caso Y será a nota (a classe) e X qualquer combinação das variáveis consideradas.

O algoritmo foi aplicado com um suporte mínimo de 8% (correspondendo a 137 instancias) e com uma confiança mínima de 33%. O conjunto de regras resultante apresenta-se na tabela 5.6.

Analisando em primeiro lugar as regras mais genéricas, ou seja, aquelas que envolvem menos variáveis, verifica-se que com 47% de confiança se utilizarmos um sistema *column-oriented* iremos obter a nota máxima de 5 (regra 24). No caso da *row-oriented* só resultaram três regras, 33, 34 e 35, e todas elas levam à pior nota.

Com duas variáveis, e relativamente ao SGBD, verifica-se que no caso do MySQL existe 42% de confiança de que a nota será 1, no ICE uma confiança de 50% de que a nota será 4, e no MonetDB uma confiança de 93%,quase total, de que a nota será 5.

Verifica-se novamente a grande influência do processador na arquitectura *column-oriented* pelo facto das regras se dividirem claramente por esta variável, ficando o processador *High-End* na metade superior e o *Low-End* na metade inferior.

De realçar ainda a regra 1 que nos diz que um sistema MonetDB, com um processador High-End e com um disco rápido irá obter, com 96% de confiança, a nota máxima 5.

Regra #	Confiança	Arquitectura	SGBD	Processador	Disco	RAM(Mb)	RAM(Mhz)	⇒	Nota
1	96%	column	MonetDB	High-End	SATA II			⇒	5
2	96%	column	MonetDB	High-End		3072 Mb		⇒	5
3	95%	column	MonetDB	High-End				⇒	5
4	94%	column	MonetDB	High-End			800 Mhz	⇒	5
5	94%	column	MonetDB		SATA II	3072 Mb		⇒	5
6	93%	column	MonetDB					⇒	5
7	93%	column	MonetDB			3072 Mb		⇒	5
8	93%	column	MonetDB		SATA II			⇒	5
9	92%	column	MonetDB				800 Mhz	⇒	5
10	92%	column	MonetDB		SATA II		800 Mhz	⇒	5
11	92%	column	MonetDB			3072 Mb	800 Mhz	⇒	5
12	90%	column	MonetDB	Low-End				⇒	5
13	90%	column	MonetDB	Low-End			800 Mhz	⇒	5
14	90%	column	MonetDB	Low-End	SATA II			⇒	5
15	51%	column	ICE			3072 Mb		⇒	4
16	50%	column	ICE					⇒	4
17	50%	column	ICE		SATA II			⇒	4
18	49%	column		High-End				⇒	5
19	49%	column		High-End	SATA II			⇒	5
20	49%	column		High-End		3072 Mb		⇒	5
21	49%	column	ICE				800 Mhz	⇒	4
22	48%	column		High-End			800 Mhz	⇒	5
23	48%	column			SATA II	3072 Mb		⇒	5
24	47%	column						⇒	5
25	47%	column				3072 Mb		⇒	5
26	47%	column			SATA II			⇒	5
27	47%	column					800 Mhz	⇒	5
28	47%	column			SATA II		800 Mhz	⇒	5
29	47%	column				3072 Mb	800 Mhz	⇒	5
30	45%	column		Low-End				⇒	5
31	45%	column		Low-End			800 Mhz	⇒	5
32	45%	column		Low-End	SATA II			⇒	5
33	42%	row	MySQL					⇒	1
34	40%	row	MySQL		SATA II			⇒	1
35	35%	row				3072 Mb		⇒	1

Tabela 5.6: Regras de classificação

Capítulo 6

Conclusão e trabalho futuro

As extensas medições efectuadas nos quatro SGBD's permitem concluir que para cargas típicas de *data warehouses*, com pesquisas *ad-hoc* e acessos de leitura intensiva, a arquitectura de sistemas de bases de dados *column-oriented* mostra-se mais adequada, apresentando níveis de performance superiores. Não só o armazenamento coluna a coluna mas também as outras tecnologias associadas, tais como a materialização tardia, a iteração de blocos ou a compressão, fazem desta arquitectura a escolha certa para a carga de trabalho em causa.

Os resultados revelam que esta arquitectura faz uma utilização mais eficiente do dispositivo de armazenamento, e transfere o peso para o processador, tornando-se mais dependente deste, logo mais sensível à sua performance. Na arquitectura *column-oriented* verifica-se também uma melhor escalabilidade havendo um crescimento polinomial do tempo de resposta versus um crescimento exponencial verificado na arquitectura *row-oriented*.

Das várias variáveis em análise em todas elas se verificou uma melhor performance na arquitectura *column-oriented*, com um dos SGBD's a realçar-se pela positiva, o MonetDB, que em 93% dos casos testados garante uma nota máxima.

Cada vez mais os sistemas de gestão de bases de dados estão a adoptar arquitecturas próprias em função do tipo de carga de trabalho que vão gerir, especializando-se na resolução de problemas específicos, como é o caso dos *data warehouses*.

A metodologia utilizada neste trabalho revelou-se muito válida permitindo uma recolha de dados organizada, e uma análise de resultados relevante, rápida, e facilmente escalável a um conjunto de dados mais vasto.

O alargamento a outros SGBD's, comerciais ou não, e a realização de mais testes incluindo parâmetros tais como a utilização da memória cache, das novas memórias em estado sólido, da paralelização, ou da compressão de dados nas colunas, são algumas ideias para trabalho futuro.

Apêndice A

SSB - Definição das Tabelas

```
-- DIMENSÕES

CREATE TABLE PART (
    P_PARTKEY     INTEGER NOT NULL,
    P_NAME        VARCHAR(22) NOT NULL,
    P_MFGR        CHAR(6) NOT NULL,
    P_CATEGORY    CHAR(7) NOT NULL,
    P_BRAND1      CHAR(9) NOT NULL,
    P_COLOR       VARCHAR(11) NOT NULL,
    P_TYPE        VARCHAR(25) NOT NULL,
    P_SIZE        SMALLINT NOT NULL,
    P_CONTAINER   CHAR(10) NOT NULL
);

CREATE TABLE DATE (
    D_DATEKEY            INTEGER NOT NULL,
    D_DATE               CHAR(18) NOT NULL,
    D_DAYOFWEEK          CHAR(9) NOT NULL,
    D_MONTH              CHAR(9) NOT NULL,
    D_YEAR               SMALLINT NOT NULL,
    D_YEARMONTHNUM       INTEGER NOT NULL,
    D_YEARMONTH          CHAR(7) NOT NULL,
    D_DAYNUMINWEEK       SMALLINT NOT NULL,
    D_DAYNUMINMONTH      SMALLINT NOT NULL,
    D_DAYNUMINYEAR       SMALLINT NOT NULL,
    D_MONTHNUMINYEAR     SMALLINT NOT NULL,
    D_WEEKNUMINYEAR      SMALLINT NOT NULL,
    D_SELLINGSEASON      CHAR(12) NOT NULL,
    D_LASTDAYINWEEKFL    BIT NOT NULL,
    D_LASTDAYINMONTHFL   BIT NOT NULL,
```

```
    D_HOLIDAYFL          BIT NOT NULL,
    D_WEEKDAYFL          BIT NOT NULL
);

CREATE TABLE SUPPLIER (
    S_SUPPKEY   INTEGER NOT NULL,
    S_NAME      CHAR(25) NOT NULL,
    S_ADDRESS   VARCHAR(25) NOT NULL,
    S_CITY      CHAR(10) NOT NULL,
    S_NATION    CHAR(15) NOT NULL,
    S_REGION    CHAR(12) NOT NULL,
    S_PHONE     CHAR(15) NOT NULL
);

CREATE TABLE CUSTOMER (
    C_CUSTKEY      INTEGER NOT NULL,
    C_NAME         VARCHAR(25) NOT NULL,
    C_ADDRESS      VARCHAR(25) NOT NULL,
    C_CITY         CHAR(10) NOT NULL,
    C_NATION       CHAR(15) NOT NULL,
    C_REGION       CHAR(12) NOT NULL,
    C_PHONE        CHAR(15) NOT NULL,
    C_MKTSEGMENT   CHAR(10) NOT NULL
);


-- TABELA DE FACTOS

CREATE TABLE LINEORDER (
    LO_ORDERKEY         INTEGER NOT NULL,
    LO_LINENUMBER       SMALLINT NOT NULL,
    LO_CUSTKEY          INTEGER NOT NULL,
    LO_PARTKEY          INTEGER NOT NULL,
    LO_SUPPKEY          INTEGER NOT NULL,
    LO_ORDERDATE        INTEGER NOT NULL,
    LO_ORDERPRIORITY    CHAR(15) NOT NULL,
    LO_SHIPPRIORITY     CHAR(1) NOT NULL,
    LO_QUANTITY         SMALLINT NOT NULL,
    LO_EXTENDEDPRICE    INTEGER NOT NULL,
    LO_ORDTOTALPRICE    INTEGER NOT NULL,
    LO_DISCOUNT         SMALLINT NOT NULL,
    LO_REVENUE          INTEGER NOT NULL,
    LO_SUPPLYCOST       INTEGER NOT NULL,
    LO_TAX              SMALLINT NOT NULL,
    LO_COMMITDATE       INTEGER NOT NULL,
```

```
    LO_SHIPMODE         CHAR(10) NOT NULL
);
```

Apêndice B

SSB - Definição das Restrições

```
-- CHAVES PRIMÁRIAS

ALTER TABLE PART ADD CONSTRAINT
 part_pkey PRIMARY KEY(P_PARTKEY);

ALTER TABLE DATE ADD CONSTRAINT
 date_pkey PRIMARY KEY(D_DATEKEY);

ALTER TABLE SUPPLIER ADD CONSTRAINT
 supplier_pkey PRIMARY KEY (S_SUPPKEY);

ALTER TABLE CUSTOMER ADD CONSTRAINT
 customer_pkey PRIMARY KEY (C_CUSTKEY);

ALTER TABLE LINEORDER ADD CONSTRAINT
 lineorder_pkey PRIMARY KEY (LO_ORDERKEY, LO_LINENUMBER);


-- CHAVES ESTRANGEIRAS

ALTER TABLE LINEORDER ADD CONSTRAINT
 lo_c_fkey FOREIGN KEY (LO_CUSTKEY) REFERENCES CUSTOMER(C_CUSTKEY);

ALTER TABLE LINEORDER ADD CONSTRAINT
 lo_p_fkey FOREIGN KEY (LO_PARTKEY) REFERENCES PART(P_PARTKEY);

ALTER TABLE LINEORDER ADD CONSTRAINT
 lo_s_fkey FOREIGN KEY (LO_SUPPKEY) REFERENCES SUPPLIER(S_SUPPKEY);

ALTER TABLE LINEORDER ADD CONSTRAINT
```

```
 lo_d_fkey FOREIGN KEY (LO_ORDERDATE) REFERENCES DATE(D_DATEKEY);

ALTER TABLE LINEORDER ADD CONSTRAINT
 lo_d2_fkey FOREIGN KEY (LO_COMMITDATE) REFERENCES DATE(D_DATEKEY);
```

Apêndice C

SSB - Consultas

```
------- FLIGHT 1 -------------------------------------

-- Q1.1
SELECT SUM(LO_EXTENDEDPRICE*LO_DISCOUNT) AS REVENUE
FROM LINEORDER, DATE
WHERE LO_ORDERDATE = D_DATEKEY
  AND D_YEAR = 1993
  AND LO_DISCOUNT BETWEEN 1 AND 3
  AND LO_QUANTITY < 25;

-- Q1.2
SELECT SUM(LO_EXTENDEDPRICE*LO_DISCOUNT) AS REVENUE
 FROM LINEORDER, DATE
 WHERE LO_ORDERDATE = D_DATEKEY
  AND D_YEARMONTHNUM = 199401
  AND LO_DISCOUNT BETWEEN 4 AND 6
  AND LO_QUANTITY BETWEEN 26 AND 35;

-- Q1.3
SELECT SUM(LO_EXTENDEDPRICE*LO_DISCOUNT) AS REVENUE
 FROM LINEORDER, DATE
 WHERE LO_ORDERDATE = D_DATEKEY
  AND D_WEEKNUMINYEAR = 6
  AND D_YEAR = 1994
  AND LO_DISCOUNT BETWEEN 5 AND 7
  AND LO_QUANTITY BETWEEN 26 AND 35;

------- FLIGHT 2 -------------------------------------
```

```
-- Q2.1
SELECT SUM(LO_REVENUE), D_YEAR, P_BRAND1
FROM LINEORDER, DATE, PART, SUPPLIER
WHERE LO_ORDERDATE = D_DATEKEY
  AND LO_PARTKEY = P_PARTKEY
  AND LO_SUPPKEY = S_SUPPKEY
  AND P_CATEGORY = 'MFGR#12'
  AND S_REGION = 'AMERICA'
GROUP BY D_YEAR, P_BRAND1
ORDER BY D_YEAR, P_BRAND1;

-- Q2.2
SELECT SUM(LO_REVENUE), D_YEAR, P_BRAND1
FROM LINEORDER, DATE, PART, SUPPLIER
WHERE LO_ORDERDATE = D_DATEKEY
  AND LO_PARTKEY = P_PARTKEY
  AND LO_SUPPKEY = S_SUPPKEY
  AND P_BRAND1 BETWEEN'MFGR#2221' AND 'MFGR#2228'
  AND S_REGION = 'ASIA'
GROUP BY D_YEAR, P_BRAND1
ORDER BY D_YEAR, P_BRAND1;

-- Q2.3
SELECT SUM(LO_REVENUE), D_YEAR, P_BRAND1
FROM LINEORDER, DATE, PART, SUPPLIER
WHERE LO_ORDERDATE = D_DATEKEY
  AND LO_PARTKEY = P_PARTKEY
  AND LO_SUPPKEY = S_SUPPKEY
  AND P_BRAND1 = 'MFGR#2221'
  AND S_REGION = 'EUROPE'
GROUP BY D_YEAR, P_BRAND1
ORDER BY D_YEAR, P_BRAND1;

------- FLIGHT 3 -------------------------------------

-- Q3.1
SELECT C_NATION, S_NATION, D_YEAR, SUM(LO_REVENUE) AS REVENUE
FROM CUSTOMER, LINEORDER, SUPPLIER, DATE
WHERE LO_CUSTKEY = C_CUSTKEY
  AND LO_SUPPKEY = S_SUPPKEY
  AND LO_ORDERDATE = D_DATEKEY
  AND C_REGION = 'ASIA' AND S_REGION = 'ASIA'
  AND D_YEAR >= 1992 AND D_YEAR <= 1997
GROUP BY C_NATION, S_NATION, D_YEAR
ORDER BY D_YEAR ASC, REVENUE DESC;
```

```
-- Q3.2
SELECT C_CITY, S_CITY, D_YEAR, SUM(LO_REVENUE) AS REVENUE
FROM CUSTOMER, LINEORDER, SUPPLIER, DATE
WHERE LO_CUSTKEY = C_CUSTKEY
  AND LO_SUPPKEY = S_SUPPKEY
  AND LO_ORDERDATE = D_DATEKEY
  AND C_NATION = 'UNITED STATES'
  AND S_NATION = 'UNITED STATES'
  AND D_YEAR >= 1992 AND D_YEAR <= 1997
GROUP BY C_CITY, S_CITY, D_YEAR
ORDER BY D_YEAR ASC, REVENUE DESC;

-- Q3.3
SELECT C_CITY, S_CITY, D_YEAR, SUM(LO_REVENUE) AS REVENUE
FROM CUSTOMER, LINEORDER, SUPPLIER, DATE
WHERE LO_CUSTKEY = C_CUSTKEY
  AND LO_SUPPKEY = S_SUPPKEY
  AND LO_ORDERDATE = D_DATEKEY
  AND (C_CITY='UNITED KI1' OR C_CITY='UNITED KI5')
  AND (S_CITY='UNITED KI1'  OR S_CITY='UNITED KI5')
  AND D_YEAR >= 1992 AND D_YEAR <= 1997
GROUP BY C_CITY, S_CITY, D_YEAR
ORDER BY D_YEAR ASC, REVENUE DESC;

-- Q3.4
SELECT C_CITY, S_CITY, D_YEAR, SUM(LO_REVENUE) AS REVENUE
FROM CUSTOMER, LINEORDER, SUPPLIER, DATE
WHERE LO_CUSTKEY = C_CUSTKEY
  AND LO_SUPPKEY = S_SUPPKEY
  AND LO_ORDERDATE = D_DATEKEY
  AND (C_CITY='UNITED KI1' OR C_CITY='UNITED KI5')
  AND (S_CITY='UNITED KI1' OR S_CITY='UNITED KI5')
  AND D_YEARMONTH = 'Dec1997'
GROUP BY C_CITY, S_CITY, D_YEAR
ORDER BY D_YEAR ASC, REVENUE DESC;

------- FLIGHT 4 -------------------------------------

-- Q4.1
SELECT D_YEAR, C_NATION, SUM(LO_REVENUE - LO_SUPPLYCOST) AS PROFIT
FROM DATE, CUSTOMER, SUPPLIER, PART, LINEORDER
WHERE LO_CUSTKEY = C_CUSTKEY
  AND LO_SUPPKEY = S_SUPPKEY
  AND LO_PARTKEY = P_PARTKEY
```