Universidade do Minho
Escola de Engenharia

Raquel Gouveia Ribeiro

Sistemas de Bases de Dados Orientados por Colunas

Outubro de 2013

Universidade do Minho
Escola de Engenharia
Departamento de Informática

Raquel Gouveia Ribeiro

Sistemas de Bases de Dados Orientados por Colunas

Dissertação de Mestrado
Mestrado em Engenharia Informática

Trabalho realizado sob orientação de
Professor Doutor Orlando Manuel de Oliveira Belo

Outubro de 2013

À minha família.

Agradecimentos

Agradeço em primeiro lugar ao meu orientador Professor Orlando Belo, por toda a colaboração e dedicação que me disponibilizou na realização desta dissertação. Foi a pessoa que sempre me ensinou a dar o passo seguinte e que me fez crescer. Muito obrigada pela paciência demonstrada em diversos assuntos e pelos seus conselhos, aos quais muitas vezes recorri, pela sua vasta experiência e respeito pelo seu trabalho e pessoa.

Queria também agradecer à minha família, por toda a dedicação, apoio e por todos os momentos em que me fizeram ver a realidade, fosse ela dura ou não. Por todas as alturas que me proporcionaram as condições de trabalho necessárias, por todo o encorajamento e palavras que me fizeram acreditar. É graças sobretudo a eles, que encarei esta fase da minha vida sempre com confiança que era capaz, e com o lema de que se deve olhar em frente e nunca para trás.

Muito obrigado!

Resumo

Sistemas de Bases de Dados Orientados por Colunas

Nos últimos tempos os sistemas de bases de dados orientados por colunas têm atraído muita atenção, quer por parte de investigadores e modeladores de base de dados, quer por profissionais da área, com particular interesse em aspectos relacionados com arquiteturas, desempenho dos sistemas e sua escalabilidade ou em aplicações de suporte à decisão, nomeadamente em *Data Warehousing* e *Business Inteligence*. Ao contrário dos sistema de bases de dados mais tradicionais ("orientados à linha"), neste tipo de sistemas cada coluna de uma tabela de uma base de dados é armazenada separadamente. Deste modo, em vez de se armazenar uma linha seguida de outra, todos os valores de um atributo pertencente à mesma coluna são continuamente comprimidos e armazenados num pacote um pouco mais denso. A aplicação deste tipo de sistemas de bases de dados permite, principalmente, minimizar o tempo das *queries* típicas de um ambiente *data warehousing*, que através de sistemas de bases de dados mais convencionais seriam difíceis de minimizar. Neste trabalho, além da abordagem genérica ao tema, desenvolveram-se trabalhos especificamente orientados para a sua aplicação a um caso de estudo real. As técnicas abordadas seguiram de perto a metodologia apresentada por Kimball et al. (2008), tendo-se dado particular enfâse ao modelo de representação dos dados. Após o estudo necessário ter sido realizado, este trabalho focou-se a análise da influência e utilidade dos sistemas de bases de dados orientados por colunas num sistema de *data warehousing*. Tendo em consideração dois sistemas de bases de dados distintos, um relacional e outro não relacional, aplicou-se um conjunto de *queries* típicas de um ambiente de data warehousing sobre o mesmo conjunto de dados, apontando as diferenças em nível de tempo. Desta forma, a importância dada à sua estrutura base, funcionalidades, linguagens de descrição, manipulação e controlo, sistemas de gestão, entre outros, acabaria por facilitar o processo de conversão da base de dados em questão, do seu povoamento e a própria exploração das *queries* no Sistema de *Data Warehousing* implementado.

Palavras-chave: Data Warehousing, Sistemas de Bases de Dados Orientados por Colunas, Sistemas de Bases de Dados, Povoamento de Data Warehouses (ETL), Sistemas de Processamento Analítico (OLAP).

Abstract

Column–Oriented Database Systems

Recently systems-oriented database columns have drawn the attention from researchers, database modelers and professionals interested in subjects such as architectures, systems performance and its scalability or decision support applications, including Data Warehousing and Business Intelligence. In this type of systems each table's column of a database is stored separately, unlike the traditional system databases ("line oriented"). Thus instead of storing one line after another, all the values of an attribute belonging to the same column are continuously compressed and stored in a slightly more dense package. The application of this type of database systems enables to reduce the time of queries normally used in a data warehousing environment. This would be unlikely to achieve in conventional database systems. In this project besides the generic approach to the subject applied their approach to a real case study. The techniques used in this project follow Kimball's methodology [Kimball et al., 2008], putting the stress in the data's model representation. After a first study has been conducted, this work focused the analysis of the influence and usefulness of database systems oriented for columns in a data warehousing system. Considering two different database systems, i.e. a relational and non-relational system, a set of queries typical of a data warehousing environment was applied on the same data set highlighting the time differences. Thus, the importance given to its basic structure, features, description languages, manipulation and control, management systems, among others, would eventually facilitate the process of converting the database and its settlement and exploration of queries in the implemented data warehousing system.

Keywords: Data Warehousing, Column-Oriented Database Systems, Database Systems, Populating the Data Warehouses (ETL), Online Analytical Processing (OLAP)

Índice

Índice de Figuras

Índice de Tabelas

Lista de Siglas e Acrónimos

TI	Tecnologias de Informação
SDW	Sistemas de Data Warehousing
DW	Data Warehouse
SI	Sistemas de Informação
BD	Base de Dados
SQL	Structured Query Language
DBMS	Database Management Systems
CPU	Central Processing
I/O	Input/Output
ETL	Extract, Transformation and Loading
OLAP	Online Analytical Processing
DBA	Administrador de Base de Dados
SI	Sistemas de Informação

Capítulo 1

Introdução

1.1 Contextualização

Numa sociedade em constante crescimento e numa era considerada como sendo a era da informação, a vantagem competitiva de uma empresa é alcançada de acordo com a habilidade de adquirir e manusear a maior quantidade de informação útil dos dados existentes, de modo adequado ao uso e aos objetivos de cada empresa.

A subjetividade associada à tomada de decisões por parte dos responsáveis de cada organização, regida por suspeitas ou vontades, tem dirigido, na sua maioria, as organizações para o insucesso. Isto só evidencia que o sucesso de uma qualquer organização não se concilia com más decisões ou decisões vagas. Assim, apenas uma aposta forte nas *Tecnologias de Informação* (TI) garante vantagens competitivas reais no esforço da restruturação e dinamização das empresas. Hoje, tal trabalho está mais simplificado, uma vez que os avanços na tecnologia dos sistemas de informação possibilitam novas maneiras (e mais fáceis) de recolher e analisar a informação que se pretende, bem como novas formas para a armazenar.

Correntemente, assistimos a uma mudança de paradigma no mundo organizacional, uma nova fase, na qual, através do recurso às novas tecnologias, a chave do sucesso está essencialmente na forma como as empresas se organizam e operam para uma resposta mais eficaz. Mais organização

traduz-se em mais eficiência e mais eficácia. No entanto, num mercado cada vez mais competitivo e exigente, será que isso basta? Como se poderá racionalizar os serviços?

O *Sistemas de Data Warehousing* (SDW) dão-nos algumas respostas nesse sentido. Adoptando um papel de relevo em relação aos sistemas de suporte à decisão anteriormente disponíveis, estes sistemas disponibilizam um conjunto de meios para atingir as necessidades, ambições e preocupações mais essenciais de cada organização. Suplantando a subjetividade e o risco na tomada de decisões, a diferenciação estratégica é, cada vez mais, resultado das opções tomadas em momentos chave, na camuflagem dos pontos fracos da empresa, na exploração das debilidades dos concorrentes ou simplesmente no aproveitamento de oportunidades do mercado.

Num mercado cada vez mais competitivo, os SDW tornam o processo de tomada de decisão mais rápido e eficiente, garantindo maior confiança aos agentes sobre a credibilidade da informação. No entanto, construir um SDW, não basta para garantir o sucesso da sua implementação. Tal como se pode comprovar em diversos estudos, podem existir inúmeras razões para o insucesso dos SDW, nomeadamente:

- A incorreta modelação dimensional dos dados existentes.
- A ocorrência de erros graves no levantamento dos requisitos dos utilizadores.
- A existência de processos pouco eficientes na extração e refrescamento dos dados.
- A má escolha nas estruturas tecnológicas de suporte à operacionalidade do sistema.
- O mau planeamento do projeto.
- As deficiências verificadas na implementação do sistema.

A este conjunto de possíveis razões para o insucesso, acresce também a preocupação com a qualidade dos dados processados pelo próprio sistema. Apesar de negligenciada por parte das organizações, esta, tem vindo assumir um papel cada vez mais importante no fracasso do desenvolvimento desses sistemas.

Habitualmente, a perda de desempenho tem sido associada com a ineficácia das ferramentas de interrogação ou com a pouca capacidade de memória ou fraca execução por parte do processador, entre outras questões de hardware, por exemplo. No entanto, a permanência no sistema de dados, nunca ou raramente utilizados, designados como dados dormentes [Inmon et al., 1998], associado

às elevadas taxas de crescimento dos repositórios de *Data Warehouse* (DW), degrada os tempos de resposta na entrega dos dados aos consumidores.

Em termos gerais, podemos dizer que um DW é um repositório de dados, construído sobre uma perspetiva de armazenamento de informações a longo prazo, isto é, mantém a informação histórica à medida que existem alterações nas base de dados, dando a possibilidade às organizações de tomarem decisões de acordo com a informação existente [Connolly & Begg, 1998]. Apesar dos avanços da tecnologia dos sistemas de informação serem significativos nos dias que correm, existe uma maior dificuldade em melhorar a performance dos DW. Cerca de 40% das organizações dizem que o volume de dados está aumentar cerca de 50% ao ano, enquanto que 18% delas indicam um aumento para o dobro do tamanho (Kimball & Caserta (2004) apresentam referência a tabelas contendo na ordem dos 200 milhões de linhas). Para lidar com este crescimento anual, os sistemas de base de dados têm, naturalmente, que sofrer ajustamentos ao longo da sua vida. Muitas empresas, de médio e de grande porte, já enfrentarem regularmente várias atualizações nos seus DW [Abadi, 2010], isto porque, possivelmente, se debatem no seu quotidiano com inúmeros problemas de desempenho e de armazenamento de dados que continuam a crescer de forma significativa, sendo cada vez maior a sua importância na diferenciação num mercado considerado por muitos como hostil. Felizmente, alguns eventos atuais e tendências correntes mostram uma reviravolta importante, em prol da emergência e aplicação de um novo tipo de sistema de armazenamento de dados: os sistemas de bases de dados orientados por colunas.

Os sistemas de base de dados orientados por colunas começaram a ser mencionados, pela primeira vez, por volta do ano de 1970. No entanto, e após uma primeira tentativa, no decurso da década de 80, na qual foram apresentadas e documentadas as vantagens que estes novos sistemas apresentavam em relação aos sistemas mais tradicionais, só no ano 2000 é que estes começaram a ser tidos em maior conta no mercado [Abadi et al.,2009]. Depois, já por volta do ano 2005, os sistemas de bases de dados orientados por colunas renasceram com novas aproximações, tanto em termos de soluções hardware como de software. Assim, surgiram algumas propostas *open-source* tais como o *MonetDB/X100* [MonetDB, 2010], o *Infobright* [Infobright, 2010], o *LucidDB* [LucidDB,2010] ou o *Calpont´s InfiniDB* [InfiniDB,2010]. Durante muitos anos, apenas o *Sybase IQ* estava disponível comercialmente – e a mudar rapidamente ao longo dos últimos anos.

Assim sendo, ferramentas como o Sybase IQ [Sybase Inc, 2010], o *Addamark* [SenSage,2010] e o *KDB* foram comprovando a eficácia deste tipo de sistemas [Abadi et al.,2009].

Contrariamente aos sistemas mais tradicionais, cada coluna de uma tabela de base de dados é armazenada separadamente. Isto é, em vez de se armazenar uma linha seguida de outra, num pacote mais denso e comprimido são armazenados todos os valores de um atributo pertencente a uma coluna [Abadi et al.,2009]. Ao se colocarem dados semelhantes juntos, minimiza-se o tempo das queries, em especial aquelas que são típicas de um ambiente de *data warehousing* [Computerworld, 2007]. Sabendo que estas *queries* são orientadas a um assunto, apenas leem as colunas pretendidas na sua execução, evitando deste modo o tempo gasto em percorrer todas as colunas das linhas e por sua vez, evitando o processamento de dados dormentes. Acresce que, se um agregado necessita ser processado sobre muitas linhas, mas para um subconjunto inferior de colunas de dados, a sua execução é mais rápida do quer ler todos os dados. Outra das vantagens deste tipo de sistemas de base de dados é a compressão dos dados [Abadi et al.,2009]. No entanto, estes apresentam duas desvantagens nas operações de escrita que ocorre na construção de tuplos, nomeadamente, quando uma *query* necessita de mais do que um atributo por entidade, encontrando-se toda a informação referente a essa entidade em locais diferentes do disco [Abadi et al.,2009].

O armazenamento dos dados num *data warehouse* orientado por colunas é geralmente mais lento do que num orientado por linhas. No entanto, num *data warehouse* existem mais leituras do que escritas o que faz com que os sistemas orientados por colunas tenham vantagens e propiciem melhorias significativas no processo de satisfação das *queries* [Computerworld, 2007]. Por exemplo, no armazenamento de endereços, em vez de estes serem armazenados ao longo de uma linha, são armazenados de forma consecutiva ao longo de uma única coluna, permitindo assim obter uma melhoria satisfatória aquando da leitura dos endereços.

Ora o problema coloca-se a nível da implementação mais eficaz do SDW. Que razões para orientar DW a colunas? A conversão dos modelos convencionais deve ser feita de modo completo ou parcialmente? Segundo a metodologia Kimball (Figura 1), que *queries* é que devem ser executadas? Como será feito o povoamento do DW orientado por colunas? Trará benefícios visíveis em relação aos DW convencionais? É sobre esta área de conhecimento que se pretende desenvolver a presente dissertação.

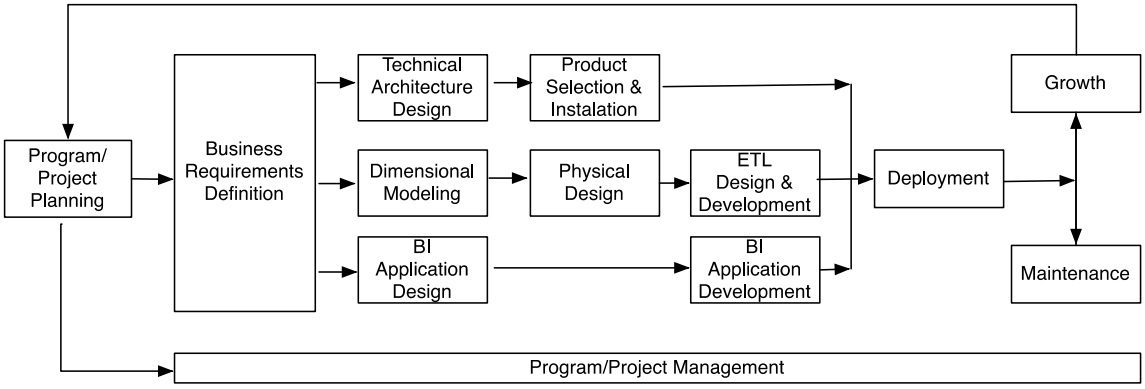

Figura 1 – Metodologia Kimball [Kimball et al., 2008]

1.2 Motivação e Objectivos

Na tentativa de abranger novos mercados que garantam a sobrevivência e o crescimento ao longo dos anos, assistimos diariamente a uma procura desmesurada por parte das organizações em encontrar novas soluções que permitam melhorar a sua "imagem" nos mercados em que desenvolvem as suas funções. Este crescimento trás a si associado um aumento significativo do volume de informação gerada e armazenada, implicando, na sua maioria, um maior tempo de tratamento e de resposta às solicitações dos consumidores de cada organização. Sendo o tempo de resposta um factor cada vez mais importante no meio empresarial, e um dos principais impulsionadores para o crescimento e eficácia nas organizações, o desempenho dos SDW é crucial para determinar o seu sucesso e, consequentemente, o das organizações associadas.

A realização deste trabalho teve como especial interesse avaliar a utilidade dos sistemas de base de dados orientados por colunas num SDW. O que se pretendia era estudar, particularmente e de uma forma detalhada, a sua estrutura base, as suas funcionalidades, as linguagens de descrição, entre outras características. Outro dos objectivos deste trabalho era a comparação entre os dois tipos de sistemas de base de dados envolvidos neste trabalho. Dessa forma, poder-se-á analisar as vantagens e desvantagens de cada um face à sua aplicação a um cenário típico de *data warehousing*, na perspetiva de encontrar novas soluções para melhorar o desempenho das organizações.

1.3 Organização da dissertação

Além do presente capítulo, esta dissertação está organizada em mais outros cinco capítulos, nomeadamente:

- **Capítulo 2 - Sistemas de Armazenamento de Dados.** Este capítulo apresenta uma breve introdução do tema do armazenamento de dados, abordando questões diversas acerca dos sistemas ditos convencionais até aos sistemas de base de dados orientados por colunas, o tema desta dissertação.

- **Capítulo 3 - Base de Dados Orientadas por Colunas. Aqui** são apresentadas as características, bem como as diferenças em relação aos sistemas de base de dados tradicionais, abordando-se em particular os modelos de dados, os motores e as ferramentas disponíveis em ambientes de base de dados orientados por colunas.

- **Capítulo 4 - Data Warehouses Baseados por Colunas.** Neste capítulo apresenta-se um caso de um sistema de *data warehouse* orientado por colunas e os principais pontos que se devem ter em conta na sua implementação. De forma complementar, apresentam-se algumas das razões que podem orientar um SDW por colunas (ou não), os vários tipos de conversão (parcial e completa) que podem ocorrer, e a forma como são feitas as diversas *queries* orientadas para o povoamento de um DW.

- **Capítulo 5 - Um Sistema de Data Warehouse Orientado por Colunas - Validação e Análise.** É apresentado um cenário típico de aplicação, de forma a se criar o suporte necessário para se proceder à comparação dos dois tipos de sistemas de base de dados envolvidos neste trabalho: orientados à linha e orientados à coluna. Este capítulo termina com o estudo dos resultados obtidos e contributos para o melhoramento do sistema.

- **Capítulo 6 - Conclusões e Trabalho Futuro**. Neste último capítulo tecem.se alguns comentários sobre o trabalho realizado, apresentam.se as conclusões das várias ações levadas a cabo, bem como se traçam algumas linhas de orientação para trabalhos futuros.

A metodologia seguida na realização desta dissertação, cobriu, essencialmente, as etapas mais relevantes apresentadas em Data Warehouse Lifecycle Tolkit, Kimball R. na aplicação de um DW a

um determinado caso de estudo real, tentando-se, em cada passo dado, referenciar de forma detalhada de cada um dos tópicos abordados.

Capítulo 2

Sistemas de Armazenamento de Dados

2.1 Preservação e Exploração de Dados

Desde os tempos mais longínquos que a vida em sociedade é própria da natureza humana. O homem nunca viveu isoladamente. A necessidade de viver em grupo manifestou-se desde muito cedo. Ao estabelecerem relações de afeto com grupos de pessoas, usualmente com algum tipo de parentesco entre elas, os humanos criaram, no seu ponto mais elementar, a família. Este grupo social primário influencia e é influenciado por outras pessoas ou instituições.

Através da partilha de gostos, preocupações, propósitos ou costumes, consegue-se estabelecer redes de relacionamentos entre pessoas, constituindo aquilo que reconhecemos como comunidades organizadas. Agrupando-se em aldeias (estabelecendo relações de vizinhança e cooperação) ou em organizações, os ser humanos compartilham interesses e preocupações sobre um dado objetivo comum. Isso beneficia-os a todos sempre de alguma maneira. Na sociedade dos dias de hoje são criados grupos sociais, nos quais existe uma divisão funcional de trabalho através da qual se visa atingir determinados objetivos e cujos membros são, eles próprios, indivíduos coprodutores desses mesmos objetivos.

Ao longo do tempo, a sociedade sofreu um processo gradual de transformação. A evolução das diversas formas de transmitir informação, como a televisão, o computador ou o rádio, fez com que se despoletassem uma série de alterações sociais, politicas e económicas que alteraram,

profundamente, a visão do mundo, resultando, como fator dominante, naquilo que hoje vulgarmente designamos por globalização. Desta forma, um dos fenómenos que a humanidade tem vindo assistir é o formidável desenvolvimento do numero, do tamanho e da complexidade das organizações.

Essencialmente, as organizações têm ao seu dispor 3 tipos de recursos: humanos, técnicos e financeiros. O *inputs* técnicos e financeiros que as organizações recebem do meio em que estão inseridas são elementos essenciais para a produção dos bens e serviços que constituem o seu *output*. Cada vez mais, o futuro dos países assenta na eficácia das suas organizações e na diferença entre si, em termos de desenvolvimento ou de pobreza, que se explica, muitas vezes, através das suas habilidades organizacionais como consequência das tecnologias utilizadas.

O sucesso na gestão organizacional está relacionada e dependente da capacidade de gerar sinergias, conjugar interesses, ou articular recursos, desempenhando um papel estruturante. No entanto, esta capacidade depende do poder de integração dos fenómenos de mudança, quer da própria empresa quer do meio envolvente em que esta se insere, bem como da aptidão de assegurar as ações de gestão através de um sistema de informação suficientemente capaz de produzir informação fiável, oportuna e seletiva, ou de vincular decisões e impactos associados a essas mesmas decisões.

A incidência sobre parâmetros objetivos, tais como a rentabilização do capital investido, não é suficiente para avaliar a performance da gestão organizacional. É importante ter em conta parâmetros mais subjetivos, parâmetros que não são avaliados matematicamente, mas que têm tanto ou mais impacto que a rentabilização monetária, tais como a satisfação dos clientes, dos fornecedores ou da sociedade envolvente à organização. A organização precisa de clientes para sobreviver e estes necessitam de produtos para satisfazer as suas necessidades. Consequentemente, a nível externo, um sistema de gestão deve basear-se na dependência existente entre as organizações e os seus clientes, procurando-se aumentar a capacidade comercial das próprias organizações. Para levar a cabo este tipo de gestão organizacional, os gestores deparam-se frequentemente com diversos tipos de problemas e de dificuldades, necessitando do apoio de informação que lhes permita suportar com detalhe as suas situações e detetar pontos fortes e fracos, ameaças e oportunidades de negócio, no sentido de poder definir estratégias, estabelecer prioridades e controlar a sua implementação. Por exemplo, para satisfazer

um cliente, um decisor precisa saber qual a quantidade de produto existente em stock (informação gerada internamente), mas para planear a próxima campanha promocional, ele necessita conhecer algumas das características que levam os consumidores a escolher o produto, ou seja, os seus hábitos de consumo e suas preferências, de forma a adaptar os seus processos de tomada de decisão ao seu mercado alvo.

Tal como se pode ver no gráfico da figura 2, em termos gerais, a informação para tomada de decisão é interna e externa à organização, enquanto que a informação para a decisão operacional é detalhada e apropriada ao decisor, tal como os produtos em stock ou eventos, como por exemplo, o número de vendas.

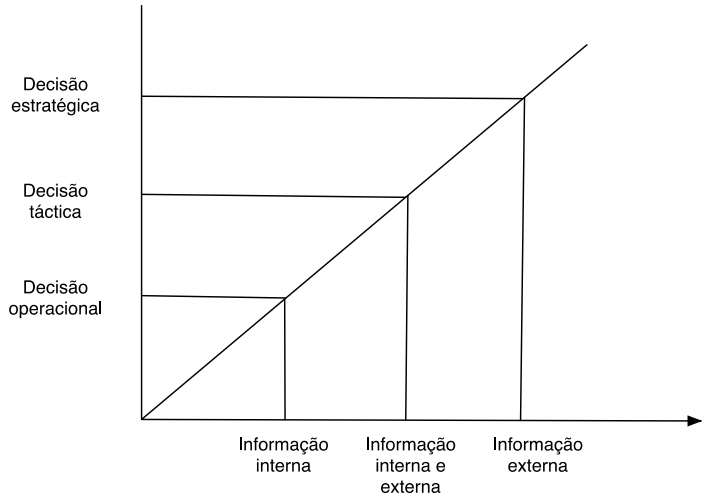

Figura 2 – Caraterísticas da informação para a tomada de decisão [Rascão, 2001]

Vejamos então o seguinte exemplo: para saber se é possível satisfazer hoje um pedido de um cliente, é necessário ter informação sobre a quantidade existente desse produto em stock. Também, para se tomar a decisão estratégica de substituir um produto, o gestor necessita de saber o total das vendas e a sua evolução, digamos, nos últimos 2 ou 3 anos, verificando-se se o produto se encontra, efetivamente, em "declínio" ou não. Assim, os gestores devem transformar a incerteza em informação, incorporando-a sob a forma de conhecimento nos seus processos de tomada de decisão. É a partir desta perspetiva que se considera a informação como cerne do progresso de uma organização. Não é uma grande novidade, com certeza. Desde sempre foi inerente ao ser humano a necessidade de preservar a informação que lhe permitisse obter quantidades adicionais de conhecimento, sobre um determinado assunto, evento, fenómeno ou

acontecimento, quer seja no seu dia-a-dia profissional ou num qualquer processo de aprendizagem requerendo algum tipo de retenção de informação.

A importância da informação dos sistemas de informação (SI) e das tecnologias de informação para o desenvolvimento organizacional está, hoje, universalmente aceite. Para além de encarada como um recurso económico, a informação é cada vez mais considerada como um fator estruturante e uma "arma" indispensável para a obtenção de vantagens competitivas.

"Vivemos numa época de acelerada mutação, é certo. No entanto, essa não é uma característica exclusivamente contemporânea. O que pode ser exclusivo é o ritmo e são os alcances."
Jorge Sampaio, QueroDizer-vos, 2000.

Quem dispõe de informação de boa qualidade, e em quantidade adequada, tem vantagens competitivas em relação aos seus parceiros. Informação é tudo aquilo que influencia as nossas decisões, independentemente da forma como é possível obtê-la. É, pois, essencial que se produza informação útil, fiável e credível em tempo oportuno e de fácil acesso a todos os interessados, para que os gestores possam tomar as decisões de forma correta, no sentido de ser possível prosseguir as estratégias definidas pelas organizações.

Ter um sistema de informação (SI) é ter a capacidade de recolher dados, selecioná-los, tratá-los e armazená-los, produzindo informação relacionada com os processos de tomada de decisão. Em tempos já remotos foram criados os sistemas de gestão de ficheiros [Martins, 2006]. Este tipo de sistemas de armazenamento, como o próprio nome indica, tem como principal característica a associação de aplicações a estruturas de ficheiros de dados (figura 3).

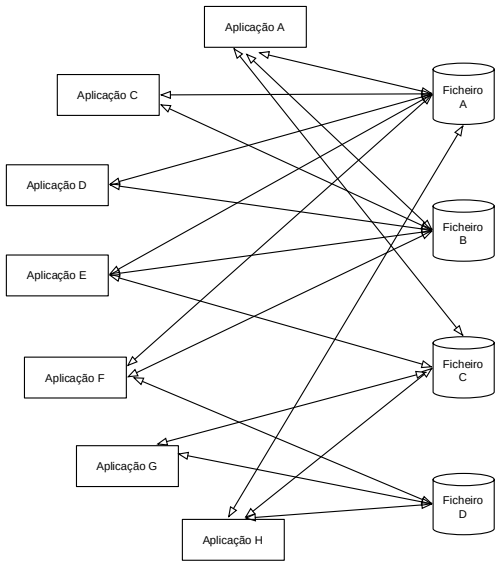

Figura 3 – Ilustração de um sistema de armazenamento de dados baseado em ficheiros

Como cada sistema é tratado de forma autónoma, não se relacionando com outros sistemas já existentes, os mesmos dados podem ser armazenados e recolhidos por várias aplicações em momentos diferentes. Assim, o mesmo documento pode ser tratado várias vezes, por vários processos, o que pode levar, em alguns casos, à replicação dos dados. Também a atualização dos dados de forma independente pode originar a ocorrência de incoerências. De igual modo, o facto das aplicações terem uma ligação direta com os próprios dados que processam, torna a manutenção deste tipo de sistemas mais complexa, em situações em que o mesmo ficheiro seja usado por mais que uma aplicação. Desta forma, uma qualquer alteração a esse ficheiro bastaria para que esta se propagasse por todas as aplicações que o utilizam.

No entanto, com o passar dos anos, e com uma quantidade de informação cada vez maior, novas necessidades surgiram obrigando a otimizar a forma como os sistemas de armazenamento de dados atuavam. Limitações como o consumo de recursos, quer humanos quer temporais, as inconsistências e a redundância dos dados, a falta de integração dos dados de diferentes aplicações, o isolamento e a dificuldade de aceder aos dados, a falta de atomicidade, de integridade e os problemas no acesso concorrente, tiveram que ser resolvidos ao nível das aplicações. Tal acelerou muito esse processo.

A má organização dos dados pode impedir o acesso a grande parte da informação que os ficheiros de dados possam conter. Tal situação levou à substituição dos ficheiros ditos convencionais por bases de dados. Em grande parte, isto foi a razão pela qual foram criados os sistemas de bases de dados. Na figura 4 podemos verificar a evolução dos sistemas de armazenamento como resposta às necessidades das organizações.

Figura 4 – Evolução dos sistemas de armazenamento de dados

2.2 Sistemas Convencionais de Armazenamento

Em termos gerais, uma base de dados (BD) consiste numa coleção de dados organizados, estruturados e armazenados de forma persistente, por uma ou mais aplicações informáticas, que permite que estes sejam acedidos e utilizados, sempre que necessário, por parte de organizações para processos de suporte operacional e de tomada de decisão.

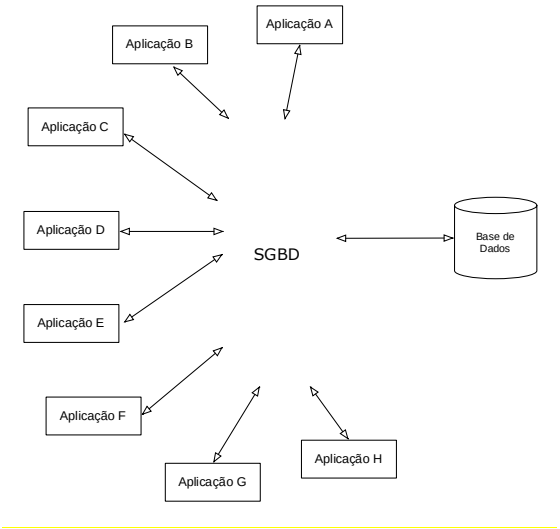

Figura 5 – Ilustração de um ambiente de um sistemas de bases de dados [4Information, 2008]

A maioria das aplicações pioneiras que utilizavam os sistemas da base de dados, mantinham os registos das organizações intactos, muitos deles com uma estrutura semelhante. Por exemplo, a informação relativa às vendas de um hipermercado poderia ser armazenada tendo em consideração os dados relativos ao cliente, ao produto, às quantidades vendidas, etc. (figura 6). Além disso, a probabilidade de existirem muitos tipos de registos e inter-relacionamentos entre eles é bastante grande.

Nome	Morada	Telefone	Produto	Modelo	Preço	Quantidade
Raquel	Monção	29547285	Leite	L4	3	12
José	Braga	29133424	Leite	L4	3	18
Raquel	Monção	29547285	Massa	M1	10	3
António	Lisboa	25560921	Tesoura	T2	5	1
António	Lisboa	25560921	Queijo	Q1	15	1
Paula	Porto	28172381	Massa	M1	10	2

Figura 6 – Uma estrutura de dados de um sistemas de base de dados pioneiro

A maioria destes sistemas de base de dados começaram a ser implementados desde a década de 70 e 80, em grandes e caros computadores, vulgarmente designados por *mainframes*, que assentavam as suas estruturas em modelos hierárquicos e em modelos de rede. Um dos principais problemas com o sistema de base de dados baseados nestes dois tipos de modelos, era a mistura entre o armazenamento físico, a localização dos registos no disco e os seus relacionamentos.

Apesar de permitirem o acesso a consultas originais e transações de forma mais eficiente não ofereciam, porém, a flexibilidade suficiente para acesso aos registos aquando de novas *queries* e transações, principalmente quando se tratassem de consultas que precisavam de um armazenamento diferente, mais eficiente e menos problemático. Para atendimento de novos requisitos, especialmente oriundos das aplicações cliente, a reorganização das bases de dados revelou-se uma tarefa extremamente complicada. O facto deste tipo de sistemas pioneiros de base de dados fornecer, apenas, interfaces para a linguagem de programação, o processo de implementação de novas consultas e transações provocava um consumido de recursos demasiado alto.

Para obtenção da tal flexibilidade, perdida pelos outros modelos, surgiram na década de 80 sistemas de base de dados relacionais. Inicialmente projetadas com o objetivo de separar o armazenamento físico dos dados da sua representação conceptual, a sua popularidade, desde que surgiram, não têm parado de crescer até hoje. Para além da simplicidade do modelo em questão, modelo relacional, também introduziram uma linguagem de consulta de alto nível, SQL (*Structured Query Language*). Desta forma, a escrita de novas consultas tornou-se um processo muito mais rápido e eficaz.

Apesar de, numa fase inicial, estes sistemas de base de dados serem muito lentos, com o desenvolvimento de novas técnicas de armazenamento e indexação e com o processamento de otimizações e de novas consultas, o seu desempenho melhorou significativamente. Constituído somente por relações entre as diversas entidades (organizações, funcionários, clientes, produtos, alunos, etc.), este tipo de base de dados existe na maioria dos computadores, desde os de uso pessoal até grandes servidores. Por exemplo, no caso que apresentámos anteriormente quando aplicado o modelo relacional ao sistema base de dados ficaria tal como as figuras 7, 8 e 9 ilustram, dando estas, em conjunto, uma visão do esquema final obtido.

Codigo_Cliente	Nome	Morada	Telefone
1	António	Lisboa	25560921
2	Raquel	Monção	29547285
3	José	Braga	29133424
4	Paula	Porto	28172381

Figura 7 – Clientes da mercearia

Codigo_Produto	Produto	Modelo	Preço
1	Massa	M1	10
2	Tesoura	T2	5
3	Queijo	Q1	15
4	Leite	L4	3

Figura 8 – Produtos comercializados na mercearia

Codigo_Venda	Codigo_Cliente	Codigo_Produto	Quantidade
1	2	4	12
2	3	4	18
3	2	1	3
4	1	2	1
5	1	3	1
6	4	1	2

Figura 9 – Relacionamento entre os clientes e produtos (vendas)

Desta forma, o relacionamento entre as duas principais entidades, nomeadamente clientes (figura 7) e produtos (figura 8) seria possível através da utilização de uma terceira tabela, a de vendas (figura 9), que como facilmente verificamos estabelece um relacionamento entre as duas primeiras tabelas.

Os sistemas relacionais apresentam a informação de forma mais compacta o que permite o acesso simultâneo e o compartilhamento dos dados, evitando a duplicação da informação e permitindo uma recuperação e modificação mais rápida dos dados. Ao se armazenar informação relacionada com os dados operacionais de uma organização e com a descrição dos dados (dicionário de dados ou metadados), as BD podem também definir alguns tratamentos sobre eles, podendo-os, restringir através de regras de negócio, definir o modo como a informação deve estar relacionada entre si, estabelecer regras para a normalização dos dados, simplificando a sua estrutura (sem duplicação de informação) e regras de segurança, definindo direitos e procedimentos, para cada um dos utilizadores do sistema, diferenciando-os de acordo com o papel que cada um desempenha no sistema [Connolly & Begg, 1998].

2.3 Das Linhas às Colunas

Para além dos tradicionais sistemas de armazenamento, existe outra forma de mapear as tabelas das bases de dados: fazendo o seu armazenamento coluna-por-coluna. Num sistema de armazenamento linha-por-linha, isto é, cada linha de uma tabela de uma base de dados é armazenada separadamente, seguida de outra linha e assim consecutivamente, até todas as linhas serem armazenadas. Desta forma, toda a informação sobre uma dada entidade ou relacionamento encontra-se num único "bloco" (figura 10).

	Coluna1	Coluna2	Coluna3	Coluna4	Coluna5	Coluna6
Linha1						
Linha2						
Linha3						
Linha4						
Linha5						
Linha6						

Figura 10 – Uma organização de dados orientada à linha [Loshin, 2010]

Imaginando-se, por exemplo, que na tabela apresentada na figura 10 temos uma tabela com informação sobre clientes, em cada uma das suas linhas teremos, então: o nome do cliente (coluna1), a sua idade (coluna2), o seu email (coluna3), a sua data de nascimento (coluna4), a sua morada (coluna5) e o código postal relativo ao se endereço postal (coluna6). Neste exemplo, a informação sobre o primeiro cliente (linha1) está armazenada na primeira linha, seguindo-se o segundo cliente, na segunda linha (linha2), e assim sucessivamente. Este armazenamento consecutivo dos valores de diferentes atributos para um mesmo tuplo é preferível, se a carga de trabalho do sistema de bases de dados, ou seja, se o tipo de *queries* normalmente efetuado no seu ambiente, tende a aceder aos dados na totalidade do grau das entidades. Por exemplo, no caso anterior, encontrar, adicionar ou apagar um cliente da base de dados, em que é bastante útil ter a informação de um dado registo toda junta, evitando-se o acesso a mais do que um objeto de dados. Este tipo de sistema é usado por aplicações transacionais que, em determinado momento, geram ou modificam um ou um pequeno número de registos. No entanto, num sistema de

armazenamento coluna-por-coluna, cada coluna de uma tabela de base de dados é armazenada separadamente, mantendo-se junta toda a informação referente a um dado atributo (figura 11).

	Coluna1	Coluna2	Coluna3	Coluna4	Coluna5	Coluna6
Linha1						
Linha2						
Linha3						
Linha4						
Linha5						
Linha6						

Figura 11 - Uma organização de dados orientada à coluna [Loshin, 2010]

Neste exemplo (figura 11), e seguindo o mesmo padrão de informação que o anterior, respeitante a um conjunto de clientes de uma dada loja, todos os nomes dos clientes (coluna1) existentes são armazenados juntos, seguido das suas idades (coluna2), depois dos seus emails (coluna3), e assim por diante até termos todos os atributos armazenados. Este armazenamento consecutivo dos valores sucessivos de um atributo é preferível, se o tipo de *queries* que queremos lançar sobre um dado sistema de armazenamento de dados tende a aceder apenas a uma pequena parte da totalidade dos atributos de uma entidade - por exemplo, uma *query* para encontrar o domínio mais comum dos endereços de email, ou obter a idade mais comum dos clientes. Nestes casos, não há a necessidade de aceder a dados, considerados irrelevantes para a *query* em questão. Apenas se pretende aceder às colunas pretendidas para a sua execução. Deste modo, o tempo que se poderia gastar em percorrer todas as colunas das linhas é evitado.

A grande maioria dos sistemas de bases de dados comerciais, entre eles os mais populares (Oracle, Microsoft SQL Server e IBM DB2), escolhe o sistema de bases de dados mais tradicional, isto é, aquele que apresenta um sistema de armazenamento linha-por-linha [Abadi,2008]. Apesar disso e nos dias que correm, as empresas preocupam-se mais com a obtenção de informação em tempo-real e em tarefas de análise mais detalhados, visando os seus processos de tomada de decisões e de planeamento do negócio. Um caso prático deste tipo de estudo e respeitante aos clientes de uma loja, o administrador pode querer procurar através da informação que possui quais os clientes que deveriam receber tratamento mais especial. Um simples sistema de base de dados nem sempre é suficiente para situações no qual a quantidade de dados cresce muito rapidamente. Adicionalmente, as *queries* necessárias para este estudo, de carácter mais analítico, para além de não processarem, geralmente, os mesmos dados que as das aplicações ditas transacionais,

consomem mais recursos (tempo e memória) na sua execução que as *queries* transacionais de escrita mais curtas, sendo necessário, não raras vezes, fazer o bloqueio das transacionais.

Um sistema de *Data Warehouse* é um sistema informático direcionado para agentes de decisão (gestores, administradores, etc.) permitindo-lhes um acesso rápido e eficaz à informação contida na base de dados, sem prejudicar o desempenho do sistema operacional. Para além de permitir interrogações constantes e pesadas ao sistema, também permite usufruir de resultados que resultam da combinação de dados provenientes de diferentes fontes de informação [Golfarelli & Rizzi, 2009]. A criação de *data warehous*es, para a execução de *queries* analíticas, está a tornar-se uma prática de negócio cada vez mais comum e imprescindível. Contudo, as *queries* típicas deste tipo de sistema são mais imprevisíveis (*queries ad-hoc*), exigindo mais recursos computacionais e sendo orientadas para um tema específico, isto é, direcionadas mais para um dado atributo do que para uma entidade em termos gerais. Igualmente, a análise de dados é um esforço naturalmente orientado à leitura de informação, implicando maior número de *queries* orientadas à leitura do que à escrita. Em parte como consequência destas características, os sistemas de armazenamento de dados coluna-por-coluna têm vindo a assumir, cada vez mais, um papel preponderante em ambientes de *data warehousing.*

Capítulo 3

Bases de Dados Orientadas por Colunas

3.1 Caracterização Geral

Uma das propriedades físicas das BD é a sua orientação. A nível físico, uma tabela precisa de ser mapeada numa estrutura de uma dimensão no espaço de endereços de memória do computador antes de ser armazenada. O mapeamento das tabelas numa estrutura unidimensional pode ser feito de duas maneiras: por linhas ou por colunas. Desta forma, a escolha do *design* a implementar é feito de acordo com o tipo de consultas que são esperadas. Caso o tipo de análise seja feita ao longo de um ou mais atributos, deve-se ter a noção que armazenar as colunas de forma consecutiva traz vantagens e, por conseguinte, uma concepção orientada às colunas revela-se como uma melhor opção em termos de desempenho em sistemas cujas *queries* trabalhem sobre a informação de um ou mais atributos.

Um conceito importante para as BD orientadas por colunas é a vectorização (figura 12). Este tipo de operação é frequentemente utilizada já por alguns sistemas de BD atuais, como é o caso do sistema MonetDB/X100 [Boncz et al., 2005]. Todas as operações básicas são relativamente fáceis de executar do ponto de vista vetorial. Por conseguinte, torna-se ideal, combinar operadores lógicos, como AND, OR, entre outros, por poderem ser realizados sobre os vetores de bits resultantes das operações.

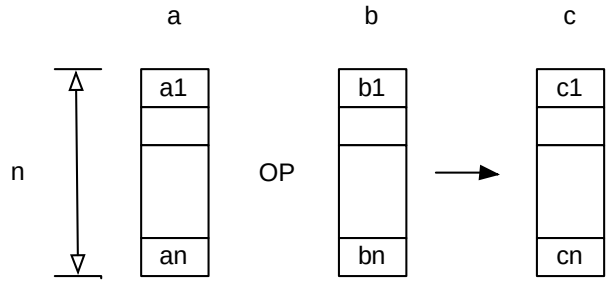

Figura 12 – Vectorização em BD orientadas por colunas [Jonsoon, 2009]

As operações de inserção e exclusão de informação são típicas em sistemas de BD orientados por linhas. Porém, são mais difíceis de lidar, de forma eficiente, em sistemas de BD orientados por colunas. Neste último tipo de BD, tais processos são mais lentos que em BD orientadas por linhas. Para além de cada tuplo ser dividido em diferentes campos (colunas), antes de ser inserido, cada um deles é armazenado separadamente e em diferentes locais de um sistema de armazenamento. Por último, caso uma coluna seja comprimida e ordenada, a carga de trabalho numa operação de inserção torna-se bastante maior. Isto requer que cada tuplo seja dividido e que sejam executadas mais operações de inserção que o habitual. Assim sendo, a abordagem comum é tentar atenuar o efeito causado pelas operações de inserção e, só depois, proceder à inserção em massa da informação.

Alguns sistemas de BD orientados por colunas, tais como o KDB e o Addamark [SenSage, 2010], mantêm as colunas no modo de entrada, inserindo novos dados no final de cada coluna. No que diz respeito à operação de exclusão, existem duas formas distintas para abordar a sua problemática. A primeira abordagem, e mais simples, trata de remover os campos das colunas correspondentes e reorganizar os dados, preenchendo a lacuna existente. Contudo, para além de dispendioso, este processo implica bloquear grandes estruturas, ao mesmo tempo que reduz a possibilidade de concorrência. Numa segunda abordagem, contudo, pode fazer-se a inclusão de um índice de exclusão, adicionando um marcador em cada um dos índices nas colunas correspondentes ao tuplo excluído. Apesar de mais tarde ou mais cedo isto permitir fazer a exclusão dos registos da estrutura por lotes, amortizando consequentemente os custos associados, esta solução acrescenta complexidade aos processos relacionados com as operações de leitura. Por último, as operações de atualização são facilmente efetuadas, isto se não existir nenhum tipo de ordenação dos dados aplicada. Desta forma, bastaria encontrar os registos que se pretendem atualizar e depois substitui-los por novos. No entanto, numa situação que integre colunas comprimidas e ordenadas, é necessário descomprimir,

atualizar, ordenar, compactar e só então armazenar os novos dados. Se necessário, um dado atributo ordenado pode ser definido como falso, o que permite fazer atualizações no lugar, sempre que as atualizações violarem a ordenação dos dados. Essencialmente, existem três tipos de otimizações usadas para melhorar o desempenho deste tipo de sistemas [Abadi et al., 2008], nomeadamente: por materialização, por boco de iteração e por junção invisível (*invisible join*).

Optimização por Materialização. Em grande parte das ocasiões, as consultas efetuadas englobam informação proveniente de várias colunas, sendo necessário fazer assim a combinação entre esses dados, o que torna a materialização das junções de tuplos numa operação extremamente comum [Abadi et al., 2007]. Assim, as colunas que estão armazenadas em partes do disco distintas, devem ser convertidas em linhas. Apesar de toda a arquitetura orientada a colunas apresentar um mecanismo para construção de tuplos, anexando posições a valores das colunas ou identificadores físicos ou virtuais de tuplos, a altura ideal para se efetuar essa conversão não muito óbvia, nem percetível, existindo uma série de fatores que condicionam esse processo. Para construir os tuplos resultantes o sistema de gestão das bases de dados necessita encontrar a correspondência entre as posições das várias colunas a combinar. Este processo, porém, é facilitado pela ordenação das colunas, isto é, para se reconstruir a quarta linha (ou o quarto tuplo) utiliza-se a quarta posição de cada coluna, o que acelera a reconstrução do tuplo. Existem dois mecanismos distintos para fazer a materialização dos tuplos. O *Early Materialization*, que procede à construção dos tuplos, o mais cedo possível, às vezes mesmo antes de ser solicitada, e o *Late Materialization* que reconstrói os tuplos o mais tardiamente possível, por vezes no fim das consultas. Deste modo, o *Late Materialization* permite operações de compressão com altos desempenhos que, com o adiamento da construção de tuplos, permita, porventura, construir menos tuplos, enquanto que no *Early Materialization*, logo que uma coluna é acedida por uma consulta, os seus valores são adicionados a um tuplo de resultados intermédios, eliminando a necessidade de aceder novamente a essa informação. Em conclusão, o *Late Materialization* permite a realização de optimizações de desempenho. No entanto e ao contrário do *Early Materialization*, o ato de aceder a colunas por várias vezes, para fazer a reconstrução dos tuplos, gasta mais tempo e, por conseguinte, aumenta os custos de desempenho.

Bloco de iteração. A implementação de blocos de iteração permite que vários valores de uma coluna sejam passados como um bloco, de um operador para outro. Em sistemas de base de dados tradicionais, sempre que se pretende processar uma série de tuplos, cada tuplo é iterado primeiramente e, só depois, lhe seriam extraídos os atributos pretendidos para a realização da *query*.

No entanto, em sistemas de base de dados orientados por colunas, e uma vez que cada atributo é independente, os blocos de valores das mesmas colunas são enviados para um operador de uma só vez. Além disso, a extração dos atributos não é necessária. Se a coluna é de largura fixa, estes valores poderão ser iterados diretamente, como se de um *array* se tratasse. Por conseguinte, operar os dados como um *array,* para além de explorar o paralelismo no CPU moderno, minimiza a sobrecarga por tuplo.

Junção Invisível (Invisible Join). Esta otimização foi criada com o objetivo de juntar várias projeções entre as várias colunas. Surgiu como resposta às desvantagens apresentadas pelos planos de execução tradicional, que consistiam em redirecionar as *joins* à ordem da seletividade do predicado, e também pelo plano de *late materialization join* (materialização tardia de junções) [Abadi et al., 2007]. Esta otimização é executada em três fases distintas. No entanto, para serem descritas e devidamente explicadas, precisamos de utilizar um exemplo prático para a sua demonstração. Veja-se então o seguinte [Abadi et al., 2008]:

```
/* Total da receita dos clientes que vivem na Ásia e que compraram produtos
num fornecedor asiático entre o ano 1992 e o ano 1997, agrupados por
nacionalidade do cliente, nacionalidade do fornecedor e ano de transação. */

SELECT c_nacionalidade, v_nacionalidade, d_ano, sum(lo_receita) as receita
FROM clientes c, ordem_pedidos lo, vendedores v, data d
WHERE lo_codigo_cliente = c_codigo_cliente
    AND lo_codigo_vendedor = v_codigo_vendedor
    AND lo_orderdate = d_id_data
    AND c_regiao = 'ASIA'
    AND v_regiao = 'ASIA'
    AND d_ano >= 1992 and d_ano <=1997
GROUP BY c_nacionalidade, v_nacionalidade, d_ano
ORDER BY d_ano asc, receita desc;
```

Numa primeira fase, como se pode verificar na figura 13, cada predicado é aplicado há tabela de dimensões apropriada, fazendo-se a extração da lista de chaves primárias que satisfazem o predicado. Essas chaves serão usadas depois para construir uma tabela de hash [Michael, 2002]. Por conseguinte, podem ser usadas para testar se um determinado valor de chave satisfaz o predicado em questão.

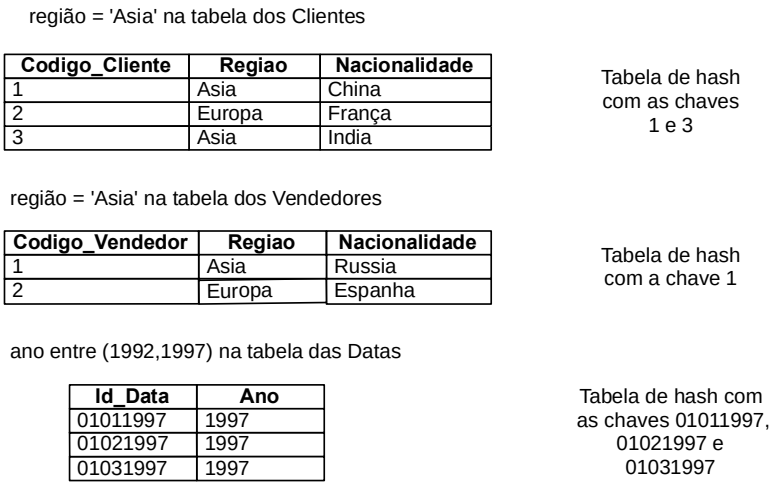

Figura 13 – Fase I – Junção (*join*) [Abadi et al., 2008]

De seguida, cada tabela de hash é utilizada para extrair as posições dos registos que satisfazem o correspondente predicado, na tabela de factos. Para isso, é feita uma pesquisa na tabela de hash com cada valor da chave estrangeira da tabela de factos. Após esta extração, esses resultados são intersectados, gerando a lista com as posições que satisfazem o predicado em questão (figura 14).

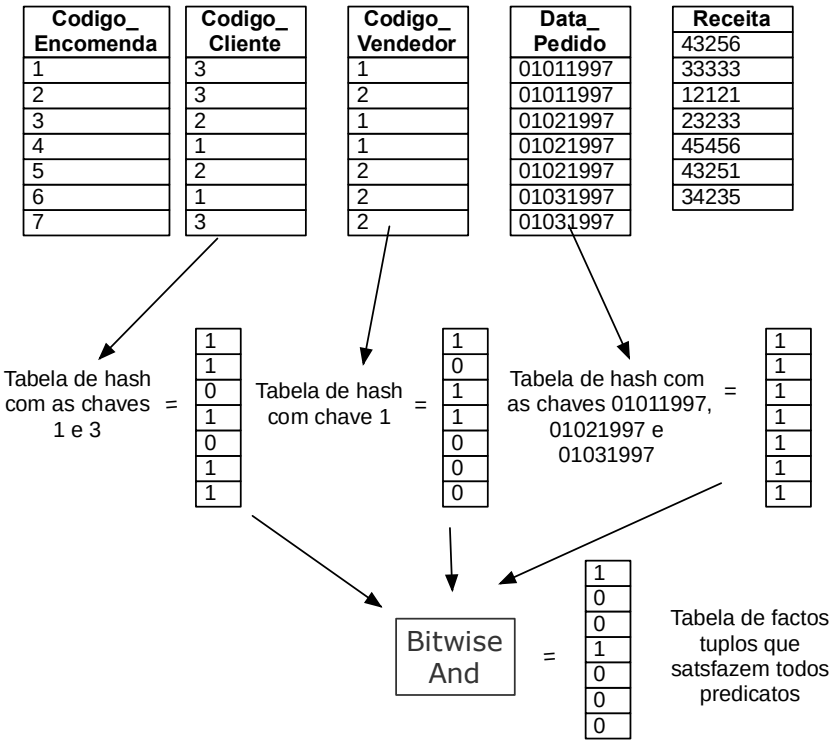

Figura 14 – Fase 2 – Junção (*join*) [Abadi et al., 2008]

Por último, na terceira fase do *join* necessitamos de executar a *query* inicialmente referenciada. Por conseguinte, por cada coluna na tabela de factos, que contém uma chave estrangeira referente à tabela das dimensões, é necessário responder à *query,,* por exemplo, quais as colunas referenciadas na lista de seleções, *group by* ou cláusulas de agregação. A todos os valores extraídos, de cada coluna pertencente à chave estrangeira da tabela de factos, é feita uma pesquisa (*looked up ())* na tabela de dimensões correspondente, de forma a obter o resultado esperado (figura 15).

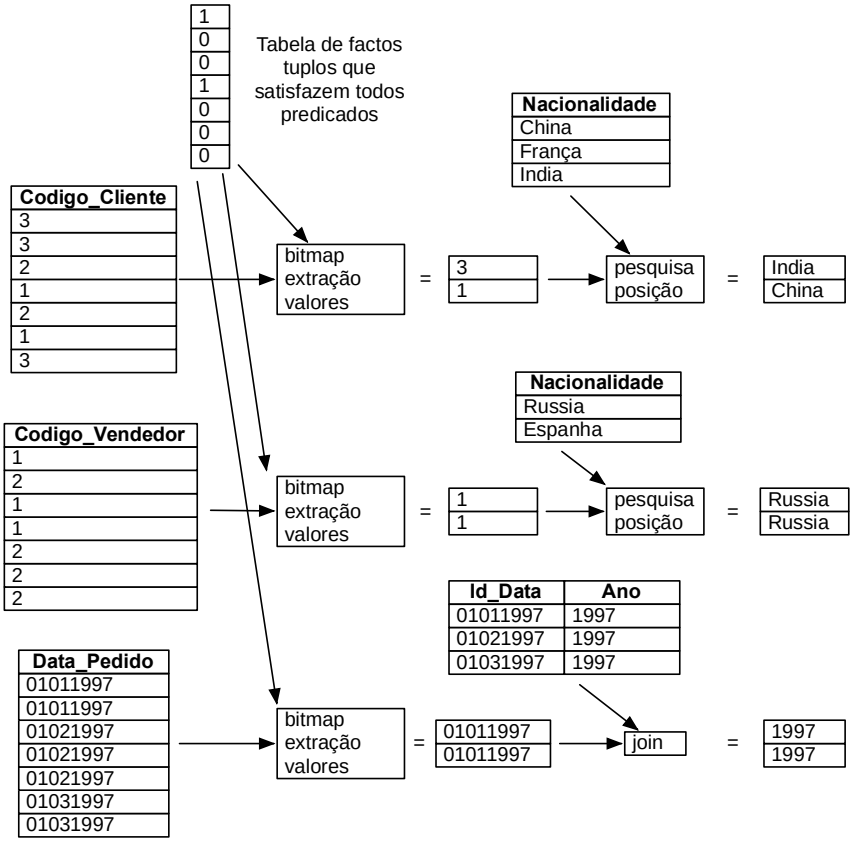

Figura 15 – Fase 3 – Junção (*join*) [Abadi et al., 2008]

Compressão. Com a utilização de algoritmos de compressão, os dados são compactados e mantidos no formato comprimido durante o processo de pesquisa de dados [Abadi et al., 2006]. Intuitivamente os dados armazenados em colunas são mais compreensíveis do que em linhas. Para além das técnicas tradicionais, o armazenamento em colunas é conveniente para esquemas de compressão que comprimem simultaneamente os valores de várias linhas. Desta forma, comprimir dados com reduzida entropia, apresenta melhores resultados. Geralmente, o ratio de compressão é maior neste tipo de armazenamento, dado que as entradas de uma coluna são por norma similares, enquanto que os atributos adjacentes de um tuplo não o são. Para além disso, sobrecarga de trabalho do processador (CPU *overhead*), resultante da iteração sobre uma página de valores de uma coluna tende, a ser menor que a iteração sobre uma página de tuplos, principalmente quando todos os valores da coluna apresentam o mesmo tamanho. Por conseguinte, isto resulta num aumento na velocidade de descompressão.

Por último, as base de dados orientadas por colunas têm a capacidade de armazenar colunas com diferentes categorias de ordenação, o que aumenta as possibilidades de compressão, uma vez que os dados quando ordenados, por norma, facilitam a compressão. Para tal, existem vários esquemas de compressão que podem ser escolhidos dependendo do tipo de compressão que melhor se adaptar à situação em causa, além de inúmeras variações de técnicas de supressão de nulos, zeros e campos vazios sucessivos. Vejamos, então, as seguintes [Abadi et al., 2006]:

1. **Supressão de valores nulos (*Null Supression*)**, zeros e campos vazios, que são substituídos por uma nota com o número de ocorrências e o local em que se encontravam. Esta técnica é eficiente em casos em que aparecem zeros e campos vazios com alguma frequência, obviamente.

2. **Codificação baseada em dicionários (*Dictionary Encoding*)**, que é talvez a técnica mais utilizada nos dias de hoje nas BDs. Esta técnica permite substituir padrões frequentes por códigos mais reduzidos. Todos os esquemas baseados em dicionários orientados às linhas apenas conseguem mapear valores de atributos de um único tuplo para entradas do dicionário, uma vez que são incapazes de juntar atributos de vários tuplos numa única entrada se outros atributos desses tuplos não estiverem incluídos na mesma entrada. Nesta técnica, começa-se por calcular o número de bits, X, necessários para codificar um único atributo da coluna (calculando diretamente através do número de valores únicos para esse atributo) e de seguida quantos desses X bits codificados podem ser contidos em 1,2,3 ou 4 bytes. Por exemplo, se um atributo tem 32 valores pode ser codificado em 5 bits. Supondo que escolhemos 3 valores em 2 bits, é feito um mapeamento entre todos os conjuntos de valores 3 5-bits possíveis e os 3 valores iniciais. Por exemplo, se o valor 1 for codificado pelos 5 bits: 00000; o valor 25 pelos 5 bits: 00001; e o valor 31 pelos 5 bits: 00010; então o dicionário teria como entrada `X000000000100010 -> 31 25 1` (lê-se da direita para a esquerda), em que X indica um bit que não é utilizado. O algoritmo de descodificação para este exemplo é intuitivo: lêem-se 2 bytes e procura-se a entrada no dicionário de forma a obter-se 3 valores de cada. Descobriu-se desta forma, que compensa recorrer a entradas de dicionários alinhadas por bytes para conseguir modestos ganhos no processador (CPU).
    - **Optimização consciente de cache (*Cache-Conscious Optimization*)**, para decidir se os valores devem ser agrupados em 1,2,3 ou 4 bytes é requisitado ao dicionário para os ajustar na cache L2. No exemplo acima, usando 2 bytes para

conter cada entrada, e o número de entradas no dicionário é 32^3 = 32 768. Portanto, o tamanho do dicionário é 524 288 bytes, metade do tamanho da cache L2 das nossas máquinas (1MB). Claro que é necessário ter em atenção que são utilizadas exclusivamente caches de 1 ou 2 bytes nas arquiteturas atuais.

- **Transformação em valores únicos (*Parsing Into Single Values*)**, este esquema é fácil adaptá-lo de forma que mapeie um atributo por uma única entrada, o que é útil para operar diretamente sobre dados comprimidos. Por exemplo, em vez de se descodificar a entrada de 16 bits nos 3 valores iniciais, podiam-se aplicar 3 máscaras e as permutações de bits correspondentes, obtendo-se os valores do dicionário para os 3 atributos:

  (X000000000100010 & 0000000000011111) > > 0 = 00010

  (X000000000100010 & 0000001111100000) > > 5 = 00001

  (X000000000100010 & 0111110000000000) > > 10 = 00000

  Em vários casos pode-se operar diretamente sobre estes valores do dicionário e atrasar a descompressão para a raiz da árvore de planeamento de *queries.*

3. Codificação de comprimento de sequência (*Run-Lenght Encoding*), que faz a compressão de sequências de valores repetidos de uma coluna numa única representação. É ideal para colunas ordenadas ou que contenham um número razoável de sequências de valores repetidos. Estas sequências são substituídas por triplos (valor, posição inicial, comprimento da sequência), atribuindo a cada elemento do triplo um valor fixo de bits. Em sistemas orientados por linhas, este esquema de compressão é apenas utilizado em atributos do tipo *string,* que tenham vários caracteres em branco ou repetidos. Porém, em sistemas orientados por colunas, este pode ser utilizado em situações em que são comuns sequências de valores repetidos e em que os atributos são armazenados consecutivamente.

4. Codificação baseada num vector de bits (*Bit-Vector Encoding*), que é útil quando cada coluna tem um número limitado de valores possíveis. Associa uma *string* a cada valor. No caso de um valor ter aparecido numa posição, associa-se um "1" nessa mesma posição, e um "0" no caso contrário. Por exemplo: a sequência **112231** seria representada por 3 *strings* de bits, nomeadamente: uma string de bits para o valor "1": 110001; uma string de bits para o valor "2": 001100; e uma string de bits para o valor "3": 000010.

5. **Esquemas de compressão de grande carga (*Heavy Weight Compression Schemes*)** - Codificação Lempel-Ziv -, que substitui padrões de tamanho variável por códigos de tamanho fixo. Consiste, basicamente, em reconhecer e transformar a sequência de entrada (input) em blocos de diferentes tamanhos, sem interceções, enquanto constrói um dicionário de blocos. Por conseguinte, caso o mesmo bloco ocorra mais que uma vez, uma ocorrência posterior é substituída por apontadores para uma anterior ocorrência.

6. **Quadro de referência (*Frame of Reference*)**, utilizando um valor base, um quadro de referência armazena todos os valores como compensações a partir dos valores de referência. Para além disso, introduz o conceito de exceções para lidar com a vulnerabilidade dos algoritmos ao existirem valores muito divergentes da média. Assim, com a redução da gama de valores diferentes é possível uma maior taxa de compressão.

Alguns esquemas de compressão, permitem operações rápidas e eficazes sobre dados compactados. O facto de não existir a necessidade em descompactar previamente os dados, reduz os requisitos de I/O, bem como pode reduzir os de CPU. No entanto, a este processo acresce a complexidade na sua implementação. Basicamente, o optimizador tem que estar ciente dos custos e implicações na eficiência das operações em operar diretamente sobre dados comprimidos. Para além disso, modelos que só tenham em consideração custos de operações de *I/O* terão, provavelmente, um fraco desempenho no contexto dos sistemas orientados por colunas, visto que os custos do processador são geralmente o fator dominante.

Uma das abordagens mais comuns, quando se executam operações em dados comprimidos, é a descompressão dos dados num *buffer* da memória principal seguida da operação de seleção. No entanto, e como se pode ver na Figura 16 a), a cache não intervêm diretamente nesta abordagem. Outra possível abordagem - Figura 16 b) - passa pela descompressão dos dados em pequenos blocos, que como são de tamanho reduzido cabem na cache, não sendo necessário recorrer ao buffer intermédio. Desta forma, não só economizamos I/O como economizamos espaço de armazenamento na memória principal.

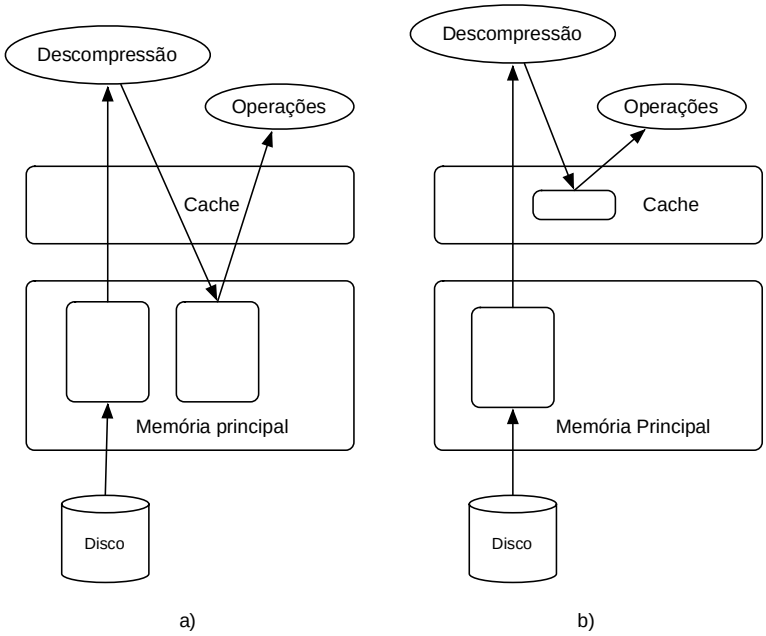

Figura 16 – Abordagens de descompressão [Jonsoon, 2009]

Alguns estudos anteriores realçam a importância de todas estas técnicas. No entanto, a compressão é geralmente alcançada através da descompressão dos dados antes de se chegar aos operadores. Apesar disso, em alguns casos, a eficiência do processo pode ser melhorada se, no processamento das *queries*, os operadores puderem operar diretamente sobre os dados comprimidos [Abadi et al., 2006], o que poderá justificar as diferenças alcançadas entre estes últimos operadores e os operadores relacionais. De referir, então, os seguintes operadores:

- **Operadores de seleção,** que produzem colunas de bits (*bit-column*) que podem ser combinados de um modo eficiente. É utilizado um operador "máscara" para se chegar a um subconjunto de valores de uma coluna e de um bitmap.
- É utilizado um **operador permutador especial**, que é responsável por reordenar uma coluna que utilize um índice de junção (*Join Index*).
- Uma projeção não apresenta qualquer custo visto não requerer alterações ao nível dos dados. Além disso duas projeções na mesma ordem podem ser concatenadas sem qualquer custo associado.
- As junções (*Join*) produzem "posições" em vez de valores.

3.2 Porquê Orientar as BDs às colunas

Antigamente, e de acordo com a evolução dos tempos, os prédios eram de menor estrutura e construídos com um tipo de material completamente diferente dos dias que correm. Antes do aço se tornar abundante e barato, os arranha-céus eram, sobretudo, prédios altos e desengonçados, limitados em altura e construídos à base de tijolos, apoiados horizontalmente pelas paredes. Um método, igualmente, seguido pelas bases de dados tradicionais. Os mais altos, nesses tempos, não atingiam alturas superiores a 20 andares. Contudo, arranha-céus mais modernos, como podemos ver em Taiwan, o Taipeh 101, apoiados verticalmente por uma estrutura de aço, tem cerca de 1.474 pés de altura, distribuídos por 101 andares. Estes, assemelham-se aos *data warehouses* orientados por colunas, que como mencionou Philip Howard em Março de 2008, proporcionam melhores desempenhos a um custo menor com um espaço menor. Segundo Howard, é difícil entender uma empresa que pretenda melhorar o desempenho das *queries* sem que para isso considere uma solução baseada em colunas [16]. De facto, e segundo muitos especialistas, os *data warehouses* baseados em colunas estão no sentido ascendente no mercado e o uso das abordagens são cada vez mais um assunto a resolver.

No núcleo do *data warehouse* maior do mundo, criado com o Sun SPARC Enterprise M9000 Server [Oracle, 2009] e BMMsoft Server, encontra-se um *data warehouse* baseado em colunas – Sybase IQ [Sybase Inc, 2010]. Projetado, sobretudo, para suportar um grande número de intervenientes a aceder simultaneamente ao sistema com *queries* ad-hoc, o processo de projeção do Sybase IQ deu mais prioridade ao desempenho das *queries* e depois à velocidade à qual a atualização dos dados em massa deve ser realizada. Contudo, o desempenho da atualização dos dados de menor porte teve uma prioridade muito menor para a definição dessa arquitetura. Com cerca de 1500 clientes a utilizarem o Sybase IQ, entre os quais se encontra a Prémalliance, um dos principais prestadores de serviços sociais da França, podemos dizer que os *data warehouses* baseados em colunas dificilmente se encontram nas sombras nos dias que correm. Cada vez mais são tidos em conta na hora de decidir qual o sistema de base de dados a utilizar no suporte de um *data warehouse*.

A grande vantagem da utilização da Sybase IQ por parte da Prémalliance foi a sintetização de cerca de 30 milhões de linhas de dados em menos de 5 segundos, que evitou uma enorme carga de trabalho ao servidor. Segundo o administrador da Prémalliance, antes, só para preparar um relatório com informações contidas na base de dados para uma reunião do conselho a empresa demorava

cerca de 2 a 3 semanas a reunir e a organizar toda a informação relevante. Nos dias que correm, e com a implementação deste novo sistema, esse tempo foi reduzido drasticamente, recebendo, inclusive, destaque pela realização tecnológica baseada no Sybase IQ.

Segundo Geoffrey Moore, a Sybase, com esta ideia radical, proporcionou um paradigma completamente novo e inovador, criando um novo tipo de oportunidades de negócio e de exploração de dados no mercado. Este novo paradigma revolucionou a forma como encarar a arquitetura deste tipo de sistema, providenciando novas técnicas e meios para uma nova arquitetura que permite extrair dados de forma 100 vezes mais rápida que as tradicionais e que, além disso, permite a partilha do sistema, em tempo real, por muitas mais pessoas [Edge, 2012]. Efetivamente, um sistema de armazenamento em colunas proporciona melhorias significativas no desempenho das *queries*, principalmente em *queries* típicas de um ambiente de *data warehousing*, *queries* estas que, tal como referido anteriormente, usualmente estão orientadas para um assunto especifico.

Sempre que se efetua uma *query* de seleção para recuperação dos dados contidos numa coluna em sistemas de base de dados orientados por colunas, o número de dados recuperados é sempre menor que o número de dados recuperados em sistemas tradicionais. Assim, tabelas que apresentem um grande número de atributos não são um problema para *queries* que acedem a um número fixo de atributos por tabela, tenham elas "3 ou 4 milhões" de colunas. Neste tipo de armazenamento, apenas os atributos necessários para a realização da *query* são acedidos, lidos e processados. Contudo, no armazenamento em linhas, existem colunas que simplesmente não podem ser ignoradas, visto que, sempre que um atributo precisa ser lido, normalmente os dados do mesmo tuplo no qual os dados estão inseridos são igualmente acedidos. Desta forma, em sistemas de base de dados tradicionais, para uma *query* que acede apenas a uma pequena parte dos atributos, em tabelas com uma grande quantidade de informação e implicando o acesso aos atributos envolventes a quando da sua realização, esta resulta num grande desperdício de performance. Por outro lado, em sistemas de base de dados orientados por colunas, e uma vez que os valores pertencentes ao mesmo atributo são armazenados todos juntos e de forma consecutiva, isto não é um problema, tendo até como vantagem uma melhor utilização da largura de banda [Abadi, 2007]. Por conseguinte, para *queries* típicas de uma ambiente de *data warehousing*, estes apresentam melhores desempenhos, acedendo apenas aos atributos necessários para a execução da *query*.

Do mesmo modo, o armazenamento dos dados em colunas apresenta um maior número de oportunidades para melhorar a eficiência do sistema através dos algoritmos de compressão, melhorando inclusive a compressão dos dados, quando comparado com o que acontece em arquiteturas orientadas por linhas. Numa base de dados orientada por colunas os esquemas de compressão que codificam múltiplos valores são bastante normais, podendo-se até escolher para cada atributo um esquema de compressão distinto.

O esquema de compressão é escolhido com base na dispersão da coluna em questão. O facto de cada coluna ser armazenada separadamente das restantes, e em distintas posições do disco, facilita muito este processo [Abadi et al., 2006]. Desta forma, o tratamento dos NULLs torna-se mais fácil, não impondo grande sobrecargas ao seu desempenho. Se a coluna a comprimir é dispersa, os NULLs podem ser codificados ao longo da execução.

Neste trabalho realizámos algumas comparações entre o desempenho de um sistema de gestão de base de dados tradicionais com um sistema de base de dados *open-source* orientadas às colunas - o C-Store – tendo em conta o nível da compressão dos dados. Sabendo que todas as opções relativas ao tratamento de valores nulos num sistema de gestão de base de dados tradicional desperdiçam espaço ou impedem melhores desempenhos, nos sistemas orientados por colunas, para uma grande sobrecarga de espaço, tais valores ocupam um espaço mínimo, sendo omitidos na coluna.

Dependendo do espaçamento das colunas a ser comprimidas, existem diferentes técnicas de compressão. Contudo em base de dados tradicionais, este tipo de esquemas não se enquadra tão bem, uma vez que cada atributo é armazenado como parte de um tuplo. Desta maneira, combinar o mesmo atributo de diferentes tuplos num só valor iria exigir uma carga extra [Abadi et al., 2006]. Outro dos motivos para utilizar este tipo de sistemas de base de dados é a melhor localização da sua cache, uma vez que cada linha da cache tende a ser maior que um tuplo, estas acabam por conter atributos irrelevantes num qualquer armazenamento de linhas [Ailamaki et al., 2001]. Por último os dados de cada atributo, neste tipo de armazenamento, podem ser iterados diretamente sem ser por via indireta através de uma interface de tuplos, providenciando assim alta eficiência em termos de IPC (instruções por ciclo) e do código que pode tirar vantagens das propriedades super-escalares dos CPUs modernos. Com isto, os sistemas de base de dados orientados por colunas apresentam melhorias no código *pipeling*, apresentando assim melhor desempenho.

Apesar de tudo isso, como qualquer sistema, os sistemas de base de dados orientados por colunas apresentam algumas desvantagens em relação aos sistemas mais tradicionais. No que diz respeito às *queries*, se a informação requerida por uma *query* contiver mais que um atributo, 2 ou mesmo o número total de atributos de uma entidade, e estando cada coluna armazenada separadamente em diferentes locais do disco, a construção da informação necessária para a execução da *query* é mais demorada do que num sistema tradicional. Isto deve-se ao facto de que num sistema de armazenamento de dados convencional a informação está armazenada toda junta. Todos os valores dos atributos relativos a uma mesma "entidade" estão num único registo. Tal circunstância faz com que haja um aumento dos custos da satisfação da query, uma vez que é necessário fazer a reconstrução dos tuplos envolvidos nos resultados da query [Abadi et al., 2009]. Por seu lado, os custos de inserção de dados também aumentam nos sistemas orientados à coluna. Com a atualização de cada tuplo inserido e, tendo em conta que cada atributo do tuplo está inserido em partes distintas do disco, para aceder a todos os locais do disco pretendidos para realização da *query*, é necessário realizar um maior conjunto de operações de relacionamento e, como tal, os custos de satisfação da query aumentam. Além disso, a procura nas diversas partes do disco entre cada bloco de leitura, por vezes pode implicar que múltiplas colunas sejam lidas em paralelo, o que aumenta o tempo de procura no disco [Abadi, 2007].

No momento em que se tenha de decidir entre uma base de dados tradicional ou uma base de dados orientada por colunas, as vantagens e desvantagens de umas em relação às outras devem ser, obviamente, tidas em consideração [Abadi, 2007]. Para isso, é necessário fazer um estudo detalhado às várias abordagens e arquiteturas existentes, ter noção dos diversos motores e ferramentas disponíveis para os sistemas de base de dados orientados por colunas e uma ideia bastante concreta (e prática) das várias *queries* que poderão vir a ser executadas no sistema.

3.3 Abordagens e Arquiteturas Existentes

Uma das primeiras abordagens aos sistemas de base de dados orientados por colunas foi realizada com uma simulação deste tipo de armazenamento num sistema de base de dados tradicional aplicando o particionamento vertical das tabelas. Permitindo apenas a leitura das colunas necessárias para responder a uma consulta, o tipo de abordagem realizada particionava uma tabela dividindo cada coluna em coleções de duas, isto é, em pares (chave da tabela, atributos). Esta abordagem

necessitava de um mecanismo que ligasse todas as colunas, colocando os campos de uma mesma linha todos juntos. Para facilitar este processo e tendo em conta que, tipicamente, no armazenamento por colunas, as colunas são armazenadas na mesma ordem, a abordagem mais simples seria adicionar como chave a posição na qual se encontra cada valor da coluna na tabela. Por conseguinte, era criado uma tabela física para cada coluna no modelo lógico, com uma coluna indicando a posição e outra contendo o valor correspondente na posição da coluna. Apesar de ser um mecanismo bastante percetível e de fácil implementação, esta abordagem exige um atributo posição em cada coluna, o que desperdiça espaço e largura de banda. O facto de este mecanismo ser implementado sobre um sistema de armazenamento por linhas e tendo em conta que a maioria destes sistemas armazena um cabeçalho em cada tuplo relativamente grande, também provoca desperdício de espaço de armazenamento.

Para amenizar os problemas levantados pela anterior abordagem, surgiram duas novas formas de implementação de um sistema de armazenamento por colunas: o modelo de indexação único e materialização das vistas.

Uma abordagem, modelo de indexação único, cria um conjunto de índices que abrangem todas as colunas necessárias para responder a uma determinada consulta e as relações base são armazenadas usando o modelo orientado por linhas. Este processo baseia-se na construção de listas de pares (id,valor) que satisfaçam os predicados das *queries* em cada tabela e pela fusão dessas listas aquando da existência de múltiplos predicados na mesma tabela. No entanto, caso os campos obrigatórios na execução das *queries* não contenham predicados, este processo pode produzir uma lista de todos os pares, obrigando a digitalização dos índices para que os valores sejam extraídos. Uma possível otimização para esta abordagem pode ser, por exemplo, fazer a criação de índices com chaves compostas, nos quais as chaves são os predicados. Por exemplo:

```
select conta_cliente
       from empregados
       where anos_trabalho > 10;
```

Se for criado um índice composto com a chave (anos_trabalho, conta_cliente), pode-se responder diretamente à *query* a partir do índice criado. Não obstante, caso sejam criados índices separados, far-se-á inicialmente uma pesquisa com base nos identificadores (id) correspondentes aos valores que satisfazem a idade de trabalho e, só depois, se procede à fusão com a lista de pares (id, conta_cliente), o que torna o processo muito menos eficaz.

A aplicação da materialização de vistas pode ser uma abordagem alternativa. Esta permite a uma *query* aceder apenas aos dados que forem necessários, evitando o armazenamento de identificadores dos registos ou das posições envolvidas e o armazenamento dos cabeçalhos dos tuplos. No entanto, a criação de um conjunto de visualizações materializadas por uma dada *query*, contendo apenas as colunas necessárias para a sua realização, apenas é ideal em algumas situações, face à carga de trabalhos que uma *query* implica.

Obviamente, quanto mais próximo for o *layout,* ou seja, o esboço, de um sistema de base de dados aos dados específicos, melhores serão os resultados. Desta forma, e uma vez que os resultados seriam mais fracos nestas abordagens do que na abordagem tradicional, foram estudados duas novas abordagens, diferindo no processo de execução das *queries*. Apesar da execução das *queries* se manter idêntica à utilizada num sistema de base de dados convencional e de todas as definições das tabelas e de SQL permanecerem as mesmas, o modo como as tabelas são armazenadas é diferente. Quando se armazena uma tabela, os identificadores dos tulos necessários para se unir os vários atributos de uma tabela não são explicitamente armazenados. Em vez disso, utiliza-se as posições das colunas para reconstruir tabelas, isto é, o valor que se encontra na posição X de cada uma das colunas, corresponde ao tuplo X na tabela [Abadi, 2008].

Ao contrário do armazenamento linha a linha, os cabeçalhos dos tuplos são armazenados separadamente nas suas colunas correspondentes. Assim, as colunas armazenadas contêm apenas a informação dessas colunas, ao contrário da abordagem de particionamento vertical, na qual contêm o cabeçalho do tuplo e o respetivo identificador, junto com os dados da coluna. Consequentemente, esta abordagem resolve uma das limitações da abordagem convencional, em que o armazenamento de uma coluna de inteiros da tabela é muito menor que na abordagem de particionamento vertical. No entanto, e uma vez que mantém o plano de execução das *queries*, esta abordagem necessita de realizar o processo de fusão dos tuplos antes da execução da própria consulta. Normalmente, apenas um pequeno subconjunto de colunas precisa ser acedido. Logo, apenas esse subconjunto será lido do sistema de armazenamento e fundido numa linha. Este processo, apesar de envolver um algoritmo de fusão extremamente simples, no qual cada valor da posição de uma coluna corresponde ao valor da posição do tuplo, traz algumas contrapartidas. Como tal, as colunas têm que ser armazenadas de forma ordenada (o valor na posição X é sempre armazenado, em cada uma das colunas, antes do valor na posição X-1), senão a junção das colunas, sem um identificador, não será possível. Assim,

modificando a camada de armazenamento e garantindo a ordenação das colunas, obtém-se um mecanismo de junção mais rápido, no que diz respeito à união das colunas em conjuntos. Apesar das vantagens significativas esta abordagem implica algum esforço extra. Por isso existe uma segunda abordagem, na qual quer a camada de armazenamento quer o plano de execução das *queries* é modificado. A diferença mais relevante desta abordagem em relação à anterior, é que o processo de fusão pode ser retardado e que a *query* especifica para as consultas pode ser utilizada na execução da própria *query*. Tomemos como exemplo o seguinte caso:

```
select X
       from tabela_exemplo
       where Y<constante;
```

Na primeira abordagem, as colunas X e Y seriam lidas do sistema de armazenamento e fundidas num único tuplo de dois atributos, obrigando à fusão de todos os valores contidos em X e em Y. No entanto, além de alguns desses valores serem futuramente descartados aquando da aplicação da constante, também será necessário efetuar a separação das colunas anteriormente fundidas, já que se pretende apenas extrair os valores do atributo X. No que concerne à segunda abordagem, esse esforço extra é completamente desnecessário, subdividindo o trabalho. A coluna Y é lida e é-lhe aplicado o predicado fora do sistema de armazenamento, obtendo-se um conjunto de valores de posições (na coluna Y) que passaram a condição, que, neste caso, foram todos os valores inferiores à constante. Por sua vez, a coluna X é digitalizada e os valores correspondentes são extraídos ao conjunto de posições que deram sucesso. Em suma, esta abordagem evita custos desnecessários de separação e de fusão de tuplos, assim como os dados movidos em torno da memória são muito menores, passando apenas os valores da coluna Y para o processador.

Em relação à arquitetura destes sistemas de base de dados, cada tabela é representada fisicamente como um conjunto de projeções. A projeção é um subconjunto da tabela original, composto por um subconjunto das suas colunas. Cada coluna é armazenada separadamente, mas com uma ordem de classificação comum. Cada coluna de cada tabela é representada em pelo menos uma projeção. Isto permite que o optimizador de consultas escolha uma das possíveis ordens de classificação disponíveis para uma determinada coluna. As colunas dentro de uma projeção podem ser ordenadas com a prioridade que assim o executador o entender [Abadi, 2008]. Um exemplo de uma projeção com quatro colunas poderá ser:

```
(data_envio, quantidade, flag, chave_compra | data_envio, quantidade, flag)
```

na qual a projeção é classificada por "data_envio", estando secundariamente ordenada por "quantidade" e de seguida ordenados por "flag". Por exemplo, a tabela a seguir:

```
(1/1/07, 1, Falso, 12), (1/1/07, 1, Verdadeiro, 4), (1/1/07, 2, Falso, 19),
(1/2/07, 1, Verdadeiro, 2)
```

Estes níveis de secundariedade aumentam a localização dos dados, ou seja, relacionam dados que são frequentemente acedidos ou que apresentam o mesmo tipo de valor, melhorando o desempenho da maioria dos algoritmos de compressão.

As projeções podem relacionar-se entre si através de *join indexes*, mantendo os relacionamentos originais entre os tuplos, quando estes são particionados e classificados em diferentes ordens. No entanto, a reconstrução de um tuplo, contendo várias colunas de múltiplas projeções usando *join indexes*, é bastante lenta. Consequentemente, uma projeção que contém todas as colunas de uma tabela é tipicamente mantida e caso uma consulta não possa ser respondida a partir de uma diferente projeção então a projeção completa é utilizada.

3.4 Motores e Ferramentas Disponíveis

Os sistemas de base de dados orientados por colunas começaram a ser mencionados, pela primeira vez, por volta do ano de 1970. No entanto, e após uma primeira tentativa no decurso da década de 80, e depois de serem documentadas as vantagens que apresentavam em relação aos sistemas mais tradicionais, só no ano de 2000 é que estes começaram a ser tidos em maior conta no mercado [Abadi et al., 2009]. Por volta do ano de 2005 renasceram com novo software e nova abordagem combinando várias técnicas inovadoras na altura. Assim, surgiram algumas propostas *open-source* tais como o MonetDB [MonetDB, 2008], o MonetDB/X100, o Infobright [Infobright, 2010], o LucidDB [LucidDB, 2010], o Calpont's InfiniDB, o Sybase IQ [Sybase Inc, 2010], o Addmark [SenSage, 2010], o KDB e o C-Store.

Pioneiro na construção de sistemas orientados por colunas, o MonetDB ao oferecer meios para a redução dos pedidos de I/O, mostrou como superar a performance dos *open-source* das BD tradicionais, apresentando melhorias no CPU e no desempenho [Abadi et al., 2008]. Utilizado com o

objectivo de processar grandes cargas de trabalho sobre BD de grandes dimensões de forma mais eficiente, por exemplo, fazendo a combinação de tabelas com milhares de colunas e centenas de milhares de linhas, este sistema apresenta inúmeras diferenças em termos de arquitetura quando comparado com as das BD tradicionais. Para além do mecanismo de execução ser baseado no processamento de uma coluna de cada vez, também, os algoritmos de processamento, que permitem a minimização da cache, a indexação implícita, que acontece como um subproduto da execução da *query*, a optimização de consultas, que é feita em tempo de execução, e o gerenciamento das transações, implementado usando tabelas adicionais, são alguns dos aspectos principais que fazem a diferença deste sistema. Aplicado com sucesso em aplicações de alto desempenho para *Data Mining*, *OLAP, GIS, XMLQuery*, texto e recuperação multimédia, a sua representação interna dos dados confia nos espaços de endereçamento de memória dos CPUs contemporâneos. Confiando na grande capacidade da memória dos sistemas em que está instalado, usando *demanding paging* dos ficheiros de memória mapeada e assim partindo de desenhos tradicionais de SGBD envolvendo a gestão de dados de grande dimensão em memória limitada. Desta forma, é um dos primeiros sistemas de BD orientados por colunas a concentrar os seus esforços na optimização das consultas sobre a exploração das caches do CPU.

O MonetDB segue uma arquitetura *front-end* e *back–end*, sendo o *front-end* responsável pela obtenção de informação das BD e por manter uma ilusão semelhante àquela que nós encontramos perante o armazenamento dos dados num modelo de dados lógico. Na Figura 17 podemos ver uma representação da arquitetura básica do sistema MonetDB e a organização geral dos seus diversos componentes. De referir:

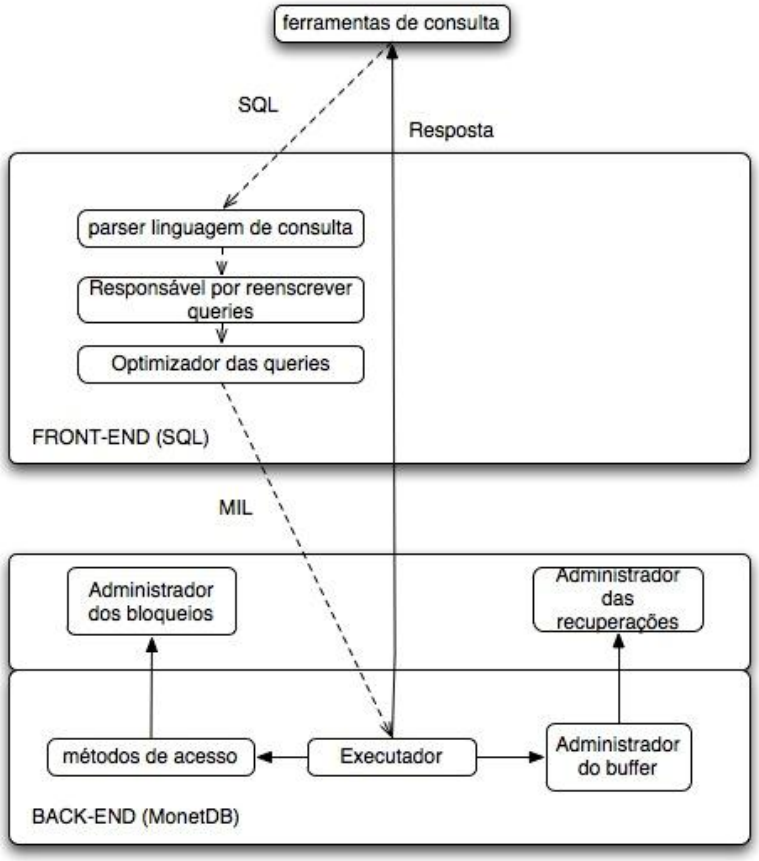

Figura 17 – Arquitectura básica do MonetDB [Venkat & Rakesh, 2007]

1. o *parser* da linguagem das *queries*, que lê as queries SQL lançadas pelo utilizador e verifica se a sua sintaxe está correta.
2. o responsável pela reescrita das queries.
3. o optimizador das *queries*, que traduz a descrição lógica das *queries* num plano de consulta que seja interpretado pelo MonetDB.
4. o executador da *query*, que é o responsável por executar as *queries* físicas e produzir os seus resultados.
5. os métodos de acesso, que são os serviços que permitem aceder aos dados através das tabelas.
6. o administrador do *buffer*, que manipula a cache das tabelas nas quais os dados estão armazenados.
7. o administrador dos bloqueios, que realiza os serviços responsáveis por bloquear transações.

8. o administrador da reparação/recuperação, que é o responsável por garantir que os dados são coerentes, assim que é feito um *commit* (transação concluída);

A arquitetura do MonetDB garante que vários *front-ends* se liguem a *back-ends*. Criado com o objetivo de permitir o armazenamento e a consulta de dados, não só relacionais como também orientados aos objetos, XML e por proporcionar a representação gráfica dos mesmos, o *front-end* do MonetDB permite o uso de várias linguagens de manipulação de dados, traduzindo as *queries* finais (ex. SQL, XML, entre outras) em álgebra BAT, que utiliza para fazer a apresentação dos resultados. Desta forma, podemos ter *queries* orientadas a objetos como *front-ends* e o MonetDB como *back-end*. A linguagem intermédia entre o *front-end* e o *back-end* é o MIL *(Monet Interpreter Language)* e contêm estruturas de controlo padrão, como if-then-else, estruturando, um tuplo, construído numa linguagem de consulta habitualmente utilizada, em blocos.

Tal como mencionado anteriormente, o processamento de cada coluna é feito através da utilização de BATs (*Binary Association Tables*) (Figura 18). As BATs são estruturas que contêm o mapeamento de cada uma das colunas na BD. Desta forma, cada coluna é representada por uma tabela binária, que integra duas colunas: um identificador virtual da linha e um valor real para a coluna (id_virtual,valor).

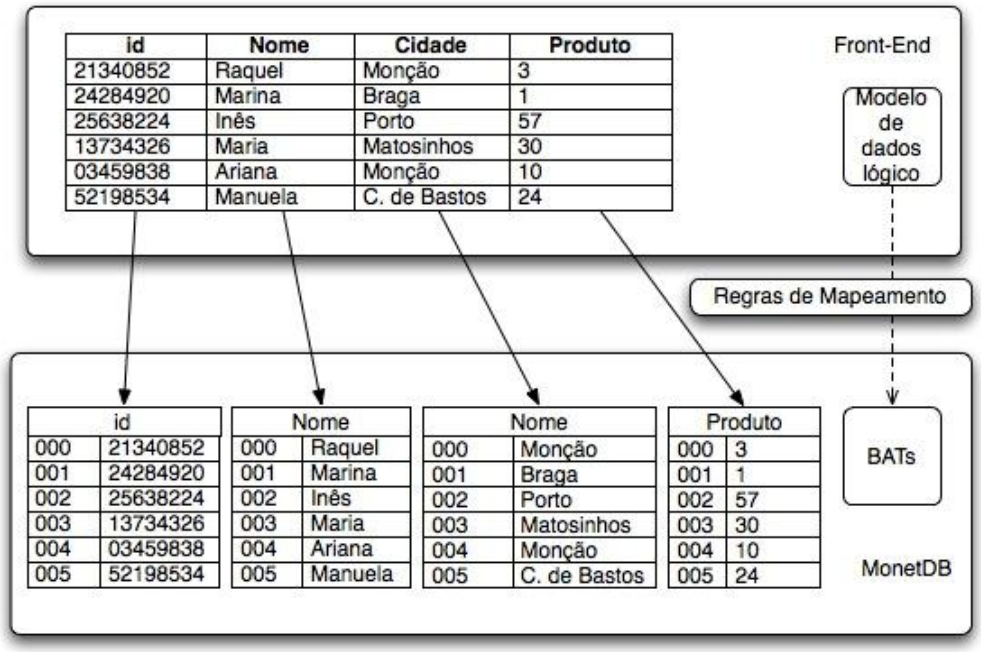

Figura 18 – As BATs do MonetDB [Venkat & Rakesh, 2007]

Por conseguinte, os dados durante a execução são sempre armazenados em BATs e, por sua vez, o seu resultado também. Por exemplo, considerando o seguinte conjunto de tabelas, relacionadas entre si:

- clientes (int id; varchar(100) Nomeq; int venda);
- vendas (int venda; int produto, float price; date day);

e decompondo-as nas seguintes BATs (Figura 19) obtemos para cada coluna da tabela de BD utilizada uma tabela binária independente:

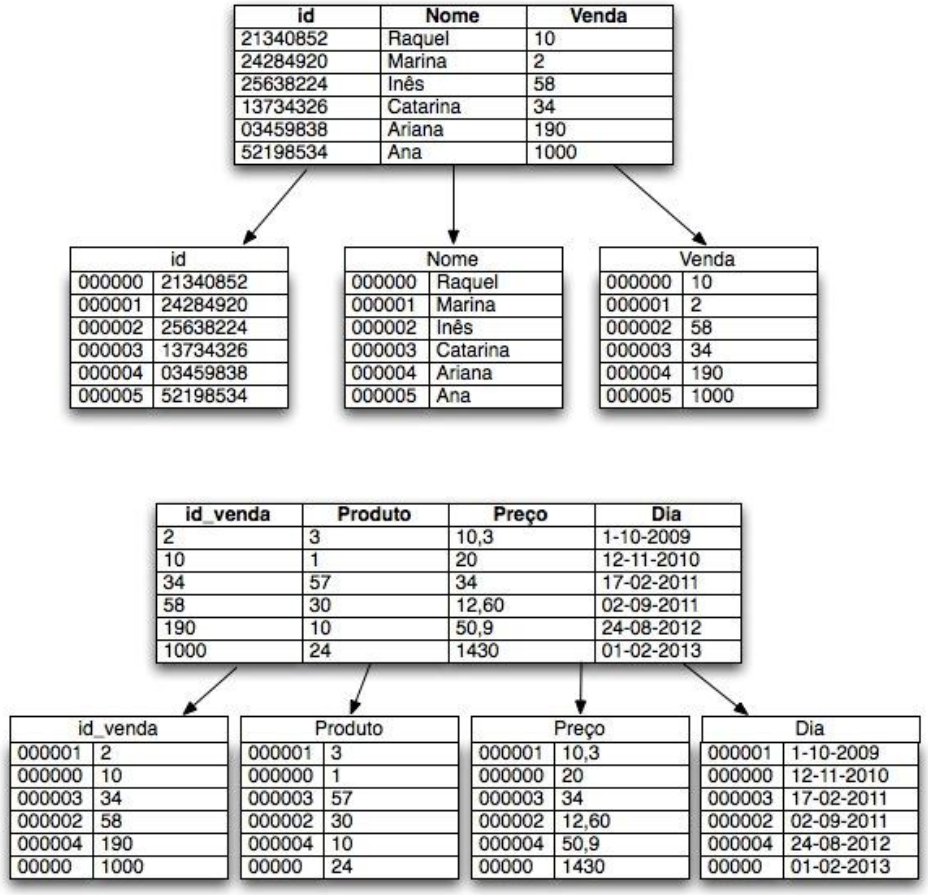

Figura 19 – Exemplo de uma implementação de BATs num modelo de dados relacional [Venkat & Rakesh, 2007]

Verificamos assim, que um qualquer modelo de dados relacional pode ser armazenado no MonetDB, fazendo a separação de cada tabela relacional em colunas e transformando cada uma delas numa BAT. Os tuplos das tabelas relacionais podem ser sempre reconstruídos tirando todos os valores da cauda de cada uma das BATs necessárias com o mesmo "id" (identificador virtual da linha).

Deste modo, usando a representação de dados interna (BATs) para manipular os dados, a falta de reconstrução de tuplos é possível, levando ao extremo a materialização de tuplos tardia (*late materialization*).

O armazenamento orientado por colunas baseadas em BATs e a independência de *front-ends* são as principais características do MonetDB. A sua orientação à coluna permite diminuir a quantidade de

operações de I/O geradas durante as operações sobre as BD. Realizando operações simples sobre colunas, para além de executar uma maior carga de trabalho com menos instruções permite uma maior eficiência em termos dos CPUs. Executando uma *query* de cada vez, o MonetDB gera, sem qualquer problema, um tuplo por cada consulta. No entanto, a política de materialização das colunas implementada pelo MonetDB proporciona, na sua totalidade, grandes fluxos de dados no período de execução da *query*. Com uma largura de banda de memória condicionada a eficiência do CPU cai drasticamente. Desta forma, e como maneira de combater as limitações apresentadas pelo MonetDB, surgiu um novo mecanismo de consulta para este open-source, o X100. O X100, partindo da linguagem original do MonetDB e através de primitivas vectorizadas, proporciona a execução de várias *queries* em simultâneo. Assim, o MonetDB/X100, além de executar *queries* com alta eficiência, também tencionava intensificar conjuntos de dados para o disco (non main memory).

Durante muitos anos, apenas o Sybase IQ estava disponível comercialmente - e a mudar rapidamente ao longo dos últimos anos [Abadi et al., 2009]. Inspirado na versão inicial do MonetDB, este sistema é conhecido pelo uso de vários tipos de índices *bitmap*. Estes aceleradores de pesquisas têm–se revelado uma ajuda enorme, sobretudo em predicados de seleção que envolvam atributos com baixa cardinalidade. Permitindo a criação automática de um índice de projeção em cada atributo, estes índices são como "fatias" de colunas verticais, utilizados durante a fase de projeto. O que torna possível ao Sybase IQ a aplicação destas estruturas de indexação é a fragmentação vertical que este realiza sobre as bases de dados que estão sobre a sua gestão.

Para além dos índices, também a compressão é um argumento chave para o bom funcionamento do Sybase IQ.

Quando um conjunto de *queries* é executado paralelamente, podem surgir problemas na consistência dos dados. Com a abordagem utilizada pelo Sybase IQ, designada por *versioning*, garante-se a integridade dos dados, evitando o bloqueio das tabelas principais aquando das transações de inserção, atualização e exclusão, atualizando essas tabelas no commit. Esta atualização é feita com base num marcador, identificando as linhas alteradas e procedendo à sua actualização na tabela principal assim que a transação for efetuada com sucesso. Assim, as restantes *queries* só terão acesso aos novos resultados existentes aquando da sua conclusão.

A constante necessidade de informação útil em tempo real e em grandes montantes por parte das empresas, para ajudar na tomada de decisões, implica uma crescente necessidade por parte dos DW em manipular os dados em tempo real e de forma paralela entre vários utilizadores do sistema. Por conseguinte, torna-se imprescindível a manipulação de várias colunas ao mesmo tempo na execução de distintas *queries*. No entanto, e apesar do Sybase IQ oferecer paralelismo nas inserções e exclusões de linhas, só permite a execução paralela se existirem recursos suficientes para o efeito [McKnight, 2002].

O C-Store é o mais recente SGBD orientado por colunas. Mesmo tendo adotado muitas das características do MonetDB/X100, este sistema apresenta otimizações na operação direta de dados comprimidos [Abadi et al., 2006]. Alcançando simultaneamente um elevado desempenho em *queries* típicas de *data warehousing* e uma velocidade bastante razoável na realização de transações tipicamente OLTP, o C-Store foi projetado com o objetivo de reduzir o número de acessos a disco que cada *query* pode implicar. Assim, com uma arquitetura híbrida inovadora, combina num sistema de *software* dois componentes distintos (Figura 20), que são responsáveis, respetivamente, pelos serviços de:

- armazenamento de escrita orientado à atualização/inserção (*Writeable Store*/WS);
- armazenamento por colunas orientado à leitura optimizada (*Read-optimized Store*/RS).

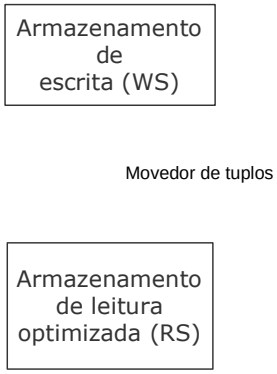

Figura 20 – Arquitetura do C-Store [Stonebraker et al., 2005]

Enquanto, que a arquitetura do WS suporta elevados desempenhos em termos de operações de inserção e de atualização de dados, o componente maior, o RS, suporta grandes montantes de

informação optimizado à leitura. Esta abordagem permite o armazenamento de várias colunas (coleção de colunas), ordenadas e classificadas num atributo, designando-se por "projeção". Desta forma, a mesma coluna poderá existir em múltiplas "projeções", ordenadas em diferentes atributos. Estes dois componentes são conectados por um movedor de tuplo (*tuple mover)*, responsável pela passagem dos registos do WS para o RS. Por conseguinte, o armazenamento dos elementos redundantes de uma tabela em distintas "projeções" com diferentes ordens proporciona novas oportunidades para optimizações, possibilitando o uso da "projeção" mais vantajosa.

Para além destes recursos inovadores, o C-Store inclui o uso de isolamento instantâneo para evitar o 2PC e o bloqueio das *queries*, elevada disponibilidade e melhorias no desempenho do k-safety, usando um número suficiente de projeções sobrepostas e colunas comprimidas, e utilizando um dos vários esquemas de codificação descritos anteriormente.

Por fim, devemos referir que existem motores que utilizam outro tipo de mapeamento de dados, o multi-dimensional, em vez de um modelo de dados relacional. Motores como o HBase, o Cassandra, o BigTable ou o HyperTable, mapeiam parte dos dados das entidades (ou "linhas" de uma tabela) em colunas por separado ("colunas família"). Apesar deste mapeamento ser bastante positivo para consultas que apenas necessitam de aceder a uma parte das linhas (1 subconjunto da informação de uma entidade), por separado é impossível aceder a um subconjunto dessas entidades. Embora permita melhorar consultas a entidades de forma individual, este tipo de base de dados, utilizam um conjunto de dados mais disperso, implicando que diferentes linhas possam ser armazenadas em conjuntos de colunas muito diferentes. Agregações de dados apresentam cargas de trabalho mais pesadas.

Além destes sistemas devemos referir o Hadoop, que foi criado pela Yahoo em 2005 e é hoje um dos projetos da comunidade Apache que vem sendo utilizado por empresas que tratam grandes volumes de dados não estruturados. Na prática, o Hadoop é uma combinação de dois outros projetos autónomos, o *Hadoop MapReduce* (HMR), *framework* para processamento paralelo e o *Hadoop Distributed File System* (HDFS), sistema de arquivos distribuídos optimizados para atuar sobre dados não estruturados. Em torno do Hadoop existem vários projetos relacionados, entre eles o HBase, uma base de dados NoSQL que trabalha sobre o sistema HDFS. Um dos seus maiores cliente é o Facebook, que o utiliza para suporte aos seus serviços de mensagens e de informação analítica em tempo real.

Com licença Apache open-source, e disponível para vários sistemas operativos, ao contrário dos motores anteriormente apresentados, este sistema apresenta uma API para Java.

Estas diferenças existentes entre as bases de dados orientadas por colunas utilizando modelos relacionais e as bases de dados que apresentam mapeamento multi-dimensional, sendo benéficas para aplicações alvo completamente distintas. Ou seja, motores de BD como MonetDB, LucidDB, KDD, SybaseIQ, entre outros, são ideias para sistemas de DW, pois são optimizados para consultas, principalmente para cargas de trabalho analíticas, suportando tempos de execução razoavelmente rápidos.

Como sistema de armazenamento por colunas, surgiu uma abordagem, designada *Fractured Mirrors* [Ramamurthy et al., 2002] (espelhos partidos), propondo desta forma um sistema híbrido, ou seja, um sistema situado entre linhas e colunas. Este esquema combina da melhor forma os aspectos mais vantajosos dos dois tipos de armazenamento. Com um processo intermédio, responsável pela migração dos dados do armazém de linhas para o armazém de colunas, este tipo de abordagem utiliza para processamento de *updates; inserts* e *deletes* o armazenamento em linhas e para processamento das leituras o armazenamento em colunas. Pensava-se, que desta forma, se obteria melhores resultados, no entanto, uma má implementação pode levar a resultados incrivelmente maus, mesmo se tratando de poucos atributos.

De seguida (Tabela 1) apresenta-se uma tabela com uma breve análise comparativa sobre alguns sistemas orientados por colunas, indicando-se as principais diferenças entre os motores e ferramentas disponíveis para os sistemas de base de dados orientados por colunas, direcionados para modelos de dados relacionais.

	MonetDB	LucidDB	Sybase IQ
Licença do Software	• MonetDB Public Licence v1.1	• GPL v2	• BSD
Sistema Operativo	• Linux • MAC OS X • Unix • Windows	• Linux • MAC OS X • Windows	• BSD • Linux • MAC OS X • Z/OS • Unix • Windows
Interface	• Mapi Client • JDBC Client	• SQL	• SQL
Características	• ACID • Unicode • XML Format Support • Java Support • Referential Integrity • Transactions	• ACID • Unicode	• ACID • Unicode • Referential Integrity • Transactions
Índices	• Hash	• Bitmap	-
Capacidade da BD	• Client-server support • Except • Inner Joins • Inner Selects • Intersect • Outer Joins • Parallel query • Union • Common table expressions	• Except • Inner Joins • Inner Selects • Intersect • Merge Joins • Outer Joins • Union	• Blobs and Clobs • Common table expressions • Except • Inner Joins • Inner Selects • Intersect • Outer Joins • Union
Particionamento	• Hash • Range (Série) • Composto (Range+ Hash)	• Hash	-
Tabelas ou vistas	• Tabelas temporárias	-	• Tabelas temporárias
Outros objetos	• External Routine • Function • Procedure • Trigger • Views	• Cursor • External Routine • Function • Procedure	• Data Domain • External Routine • Function • Procedure • Trigger
Suporte das características	• Email	• FAQ • Mailing List	• Fóruns • FAQ • Mailing List
Máximo de colunas por linhas	• Ilimitado		• Ilimitado

Tabela 1 - Tabela comparativa entre SGBD orientados por colunas [FindTheBest, 2012]

3.5 Uma Possível Seleção para um Possível Caso

Nesta altura é necessário escolher um motor de base de dados orientado por colunas, que nos permite demonstrar as características e funcionalidades mais pertinentes deste tipo de programa. Utilizado mundialmente por empresas, instituições de ensino e de investigação o MonetDB proporciona um ambiente flexível aos seus utilizador, possibilitando a sua personalização em distintas plataformas. Compatível com diversos sistemas operativos, tais como o Unix, o Linux, o Mac OS X, o Windows XP, ou o Windows Vista, e baseado no padrão SQL 2003, este sistema suporta funções, *triggers, joins, views*, procedimentos, chaves estrangeiras, entre outras muitas coisas habituais de um qualquer sistema de gestão de bases de dados. Construído sobre a representação canónica de relações da base de dados como colunas, o MonetDB destaca-se em aplicações em que algumas colunas da tabela relacional são mais que suficientes para responder às consultas efectuadas. Este sistema, considerado de alto desempenho, foi projetado para permitir a execução paralela de processos em *multi-core*, reduzindo o tempo de execução das *queries* usualmente mais complexas. De igual modo, a versatilidade algébrica do seu kernel adapta-se a diferentes linguagens, tornando este sistema de gestão de bases de dados open-source de licença pública uma mais valia. O tamanho máximo suportado pela BD do MonetDB depende da plataforma de processamento. O número máximo de colunas por tabela é praticamente ilimitado, sendo cada coluna mapeada para um ficheiro, no qual o limite é imposto pelo sistema operativo e plataforma de hardware em utilização. Durante a parte seguinte desta dissertação veremos então a forma como este motor poderá ser utilizado e explorado, em particular, num ambiente típico de data warehousing.

Capítulo 4

Data Warehouses Baseados por Colunas

4.1 Orientar ou não Orientar por Colunas

O desenvolvimento de um SDW tem como um dos seus objetivos colmatar as deficiências dos sistemas de apoio à decisão e a inadequação dos sistemas transacionais para dar resposta às questões relacionadas com as atividades de gestão e à tomada de decisões das organizações. Por vários motivos, mas sobretudo por sucesso gerar sucesso, cada vez mais, as companhias percebem a utilidade dos dados, e a necessidade em planear e reagir de imediato à evolução do mercado, através da implementação de estratégias e inovação da área, estabelecendo politicas mais flexíveis que permitam tirar proveito das oportunidades que o mercado proporciona. A maioria dos sistemas de *data warehousing* disponíveis são baseados em tecnologias relacionais orientadas às linhas. Apesar de pretenderem colmatar algumas das deficiências dos sistemas de suporte à decisão, os SDW não têm evitado que estas tenham vindo a aumentar muito nos últimos anos - estima-se que cerca de 40% das organizações, tenha um aumento de volume de dados superior a 50% por ano e 18% apresente um aumento de mais do dobro.

Muitos autores têm revelado formas de melhorar o desenvolvimento de *queries* típicas de *OLAP (On Line Analytical Processing*) para aplicações construídas em cima de sistemas de BD orientados por colunas uma vez que as *queries* que mais são executadas nesses ambientes requerem agregações de grandes quantidades de dados usando apenas um pequeno grupo de atributos. Desta forma, o armazenamento por colunas é bastante adequado para sistemas OLAP. Nestes sistemas realizam-se

essencialmente muitos processos de leitura sobre campos de dados específicos, como, por exemplo, quando se pretende explorar o número de vendas por código-postal e por produtos vendidos numa mercearia. Neste caso, bastaria apenas selecionar as colunas relativas a esses dois campos.

As base de dados orientadas por colunas trouxeram um novo impulso na produtividade desse tipo de sistema, mas nem sempre eles e respectivas aplicações são adequadas para substituir os *data warehouses* convencionais nas empresas. Estas alternativas são mais utilizadas para *data marts* que são mantidos a partir do descarregamento de dados intensivos a partir de *data warehouses*.

Nem sempre, porém, se sabe ou se tem uma noção concreta da quantidade de dados que um dado problema ou aplicação pode envolver. Por exemplo, aquando da construção das declarações de IRS da população, tal como referido no artigo "Reinvent the Data Warehouse With Column-Store Databases And Appliances" [InformationWeek, 2013] isso aconteceu, o que foi motivo suficiente para se basear a sua construção numa base de dados orientada por colunas. Quando se avalia uma aplicação ou base de dados orientada por colunas temos que saber se estamos a substituir o DW ou se estamos a atrasar uma empresa em relação a uma nova atualização no seu DW, substituindo *queries* sobre dados de estrutura complexa em *queries* analíticas com vista a melhorar a sua performance.

Será que um DW terá que suportar milhares de clientes com uma mistura de vários tipos de consultas? Ou o tipo dessas consultas será sempre o mesmo? Será necessário recorrer a agregações complexas? Se o objectivo é o de encontrar um novo *data mart* analítico ou o "próximo" *data warehouse*, temos que procurar para além do preço, a escalabilidade e o desempenho do próprio *data warehouse*. As BD orientadas por colunas são óptimas para DW, suportando consultas mais complexas e com volumes de dados extraordinários. Por exemplo, segundo Steve Hirisch, no caso da NYSE Euronext, a controladora da bolsa em Nova York, com cerca de 3 DW de 100 TB, *queries* que demoram cerca de 26 horas a executar em DW convencionais, tem agora o seu sistema baseado numa solução que lhes permite realizar os mesmos trabalhos agora em cerca de 2 a 30 minutos, enquanto que as consultas mais simples demoram cerca de 5 segundos. Apesar destas vantagens, estes não são cenários típicos de um DW, apresentando cargas de consulta limitadas e um número máximo de 20 utilizadores simultâneos por aparelho. Ainda assim, apresenta bastantes cálculos analíticos complexos e com excelentes resultados a nível de desempenho, para uma tão grande quantidade de dados [InformationWeek, 2013]. Por este motivo, e só depois de se fundamentar e

definir os objetivos do nosso *data warehouse*, quer em termos de profundidade quer em termos da quantidade de utilizadores, diversidade e volume de consultas, é que se poderá fazer uma análise no sentido de estabelecer a "melhor" implementação. Apesar desse estudo inicial, muitos autores confiam que só colocando os dados no DW e executando as consultas desejadas é que se poderá ter uma melhor percepção de qual a melhor e mais rápida alternativa.

4.2 Conversão dos Modelos Convencionais

A modelação dimensional é uma técnica relacionada com o desenho lógico, caracterizada pela capacidade de apresentar dados de uma forma estruturada e intuitiva, com o objectivo de possibilitar que o acesso a estes seja efectuado de uma forma bastante eficiente. Desta forma, a modelação dimensional é consensualmente reconhecida como a plataforma mais adequada para a conceptualização dos dados existentes num DW [Kimball et al., 1998]. A justificação por esta opção reside na facilidade de concretização de análises complexas aos dados de um modo muito mais intuitivo.

Quando um modelo de dados é projetado para um sistema de base de dados orientado por linhas, uma das maiores preocupações durante a modelação é a carga de I/O, sendo as tabelas concebidas de forma a reduzir o "peso" das suas linhas. Com base nessa "orientação", normalmente, os modelos de dados são normalizados. Um modelo dimensional divide um DW em medidas e contextos. As medidas são capturadas pelos processos do negócio e pelos sistemas operacionais, sendo vulgarmente referidas como factos. Estes encontram-se ligados a diferentes contextos que se verificam no momento em que determinado facto ocorre. Por seu turno, os contextos são denominados por dimensões. Em suma, as tabelas de factos têm uma granularidade associada, isto é, nível de detalhe armazenado numa tabela. Assim a cada tabela de factos está associada uma unidade de medida, que se chama de grão. Este não é mais do que o elemento de dados mais refinado, a peça mais elementar de análise do negócio, do evento que se pretende avaliar. Quanto maior é a granularidade, maior é o grão e menor é o nível de detalhe. Todos os modelos dimensionais são compostos no mínimo por uma tabela de factos que integra uma chave, normalmente composta, resultante da combinação de um conjunto de chaves estrangeiras provenientes das tabelas de dimensões, e de um conjunto de atributos, designados por medidas que atuam como métricas de análise.

Existem dois tipos de esquemas de modelos dimensionais, nomeadamente: estrela e flocos de neve. Enquanto que no esquema em estrela, as tabelas das dimensões encontram-se relacionadas diretamente com a tabela de factos, sendo que os dados de cada uma das tabelas de dimensão contem todas as descrições que são necessárias para definir uma classe, por exemplo, Produtos, Tempo, entre outras, no esquema em flocos de neve, as tabelas de dimensões relacionam-se com a tabela de factos, no entanto, algumas delas apenas se relacionam entre si, por exemplo, tabelas como Ano, Dia e Mês apenas se iriam relacionar com a tabela de dimensão: Tempo. Ao contrário do esquema em estrela, as tabelas de dimensões deste ultimo esquema encontram-se normalizadas. Esta normalização das tabelas de dimensão obrigou ao aparecimento de novas dimensões, diminuindo o espaço ocupado pelas mesmas. Assim, com elementos de dados semelhantes é possível obter diferentes esquemas dimensionais. Como tal, é importante optar pelo esquema que oferece melhores resultados e garantias em termos de eficácia, eficiência e compreensão das estruturas envolvidas.

A complexidade de um modelo de dados afecta a capacidade de execução dos pedidos ad-hoc dos utilizadores sobre os dados. Por vezes, estes pedidos são tão complexos que os utilizadores tem imensas dificuldades em definir as suas consultas sem comprometer os resultados e, consequentemente, sem comprometer a capacidade do sistema em transformar os dados em conhecimento empresarial credível. Para colmatar esta adversidade foram desenvolvidas várias técnicas, tais como as vistas materializadas, os cubos e os quadros de resumo, que permitem um acesso mais simples aos dados disponíveis.

Outro dos problemas que os DW orientados por linhas têm vindo a enfrentar ao longo do tempo é o caso das dimensões de variação lenta. Com base nestas limitações, sempre que as consultas de informação implicam operações de junção através de dimensões de variação lenta, nomeadamente, dimensão tempo as BD orientadas por colunas podem apresentar melhores resultados, oferecendo maior flexibilidade na modelação dimensional dos dados.

Uma das questões mais pertinentes quando se pretende implementar um sistema de *data warehouse* orientado por colunas é a conversão do modelo de dados. A desnormalização [Copeland & Khoshafian, 1985] pode ser total ou parcial, dependendo do aspecto do modelo de dados que precisa de ser simplificado - a escolha do *design* é baseado no tipo de consultas que são esperadas. Decompor uma tabela de factos em várias tabelas de dimensões, como seria implementado num DW orientado por linhas, introduzirá mais operações de junção em processos de combinação de dados e,

por conseguinte, diminuirá o seu desempenho na execução das consultas solicitadas. Ao fazermos a desnormalização do modelo de dados podemos juntar a uma tabela de factos colunas das tabelas de dimensão que possam ser mais apropriadas aos processos de consulta, eliminando o tempo de execução que teríamos no caso de termos que realizar tais junções "adicionais". Tal como Stonebraker et al. (2007) referiram esta é uma táctica muito utilizada para melhorar o desempenho dos DWs.

Outra das vantagens da modelação de um esquema orientando-nos por dados desnormalizados é a redução da complexidade em administrar tabelas de dimensão de variação lenta (*slowly changing dimensions*). Se fizermos a desnormalização do modelo de dados de forma a atenuar os efeitos dessas variações das dimensões, integraremos os atributos com variação, associados às tabelas de dimensão envolvidas, na própria tabela de factos. Desta forma, os atributos estariam sempre com os valores corretos em termos temporais e o número de operações de junção seria, mais uma vez, menor. No casos de existência de uma tabela de dimensão tempo, correspondente às datas em que cada facto ocorre, e podendo-se consultar informação com base em datas, por exemplo, consulta respeitante às encomendas efectuadas no dia 22-03-2011, a integração das colunas desta dimensão na respectiva tabela de factos permitiria um menor numero de junções, redução da complexidade em administrar a tabela de dimensão tempo, e consequentemente um melhor desempenho nas consultas.

O modelo de armazenamento orientado por colunas, também designado por DSM (*Decomposed Storage Model*), ao contrário do NSM (*N-array Storage Model*), correspondente ao modelo das BD tradicionais. Quer a implementação das BD orientadas por linhas quer das BD orientadas por colunas, pode ser realizada ao nível do armazenamento ou do processamento dos dados. Isso permite categorizar os motores de SQL em quatro grupos [Sorjonen, 2012], a saber, o modelo de armazenamento orientado:

1) às linhas com processamento de linhas;
2) às colunas com processamento de linhas;
3) às colunas com processamento de colunas;
4) às linhas com processamento de colunas.

De referir que, ao contrários dos grupos 2 e 4, os grupos 1 e 3 são grupos nativos, com o mesmo modelo de armazenamento e processamento dos dados. Caso se opte por um modelo de armazenamento orientado às linhas podem-se introduzir índices de armazenamento de colunas

(*columnstore índex)* para acelerar *os processos de consulta*. Um índice de armazenamento em colunas, armazena cada coluna em separado em diferentes páginas do disco, em vez de armazenar várias linhas por página [Hanson, 2008]. O facto de apenas as colunas necessárias para a satisfação das consultas serem obtidas a partir do disco, a maior facilidade em compactar os dados, devido à redundância dentro das colunas, e as taxas de acerto no *buffer* serem melhoradas, pela compactação dos dados e pelas colunas menos utilizadas serem paginadas, são alguns dos benefícios dos índices. Assim, para melhorar o desempenho dos processos de consulta, tudo o que é necessário é fazer a construção de um índice de armazenamento por colunas, nas tabelas de factos de um DW. Caso estejam associadas tabelas de dimensão, com muitos registos (mais de 10 milhões de linhas, por exemplo) poder-se-á proceder à construção de um índice sobre cada uma dessas tabelas.

O processamento de *queries* nos índices de armazenamento por colunas é mais optimizado para *queries* de junção a realizar sobre um esquema em configuração em estrela. Todavia, todos os outros tipos de modelação podem beneficiar deste tipo de estratégia de otimização.

Os índices de armazenamento nem sempre contribui para melhorar o desempenho das consultas num DW. Quando isso sucede o optimizador das *queries* opta pela utilização dos índices *B-Tree* para aceder de forma mais expedita aos dados do DW. Sempre que novos dados são inseridos em tabelas com este tipo de índice, estas não podem ser atualizadas diretamente, isto é, não se podem realizar operações de INSERT, UPDATE, DELETE ou MERGE, ou fazer operações de carregamento de dados diretamente. Sempre que se tenha que realizar uma dessas operações, é necessário proceder primeiro à desativação do índice antes de fazer a atualização da tabela e depois voltar a fazer a reconstrução do índice.

Outra possibilidade de otimização é a de fazer a criação de uma *view* que usa uma instrução UNION ALL para combinar uma tabela com um índice de armazenamento por colunas. Isto permite inserções dinâmicas de novos dados numa tabela de factos, enquanto que mantém ao mesmo tempo os benefícios do armazenamento por colunas [Hanson, 2008].

Os índices de armazenamento de colunas podem reduzir a carga sobre os sistemas e reduzir os tempos de processamento do sistema de ETL, diminuindo também a dependência dos agregados construídos. A concepção e a manutenção dos agregados tornam-se por vezes tarefas complicadas e dispendiosas, uma vez que um único índice de armazenamento de colunas pode substituir dezenas de

agregados. Assim sempre que uma query é ligeiramente alterada, o armazenamento por colunas com índices de armazenamento pode apresentar melhores resultados do que os agregados.

De certa forma, apesar do modelo de armazenamento orientado por colunas apresentar vantagens sobre o armazenamento orientado por linhas, vantagens que advém do facto de serem aplicadas técnicas no armazenamento orientado por linhas, tais como, um plano de indexação único, a verdade é que o modelo de armazenamento pode imitar a abordagem orientada por colunas. Ainda que o desempenho do armazenamento por linhas mostre melhores resultados, obter todos os benefícios das abordagens por colunas implica que seja necessário usar um SGBD construído do zero, com uma abordagem por colunas. Por sua vez, o modelo de processamento orientado por colunas também pode ser utilizado a partir de um modelo de armazenamento convencional, seguindo para a conversão dos tuplos num modelo orientado por colunas (DSM). Em suma, a conversão de um modelo de armazenamento orientado por linhas (NSM) para um modelo de armazenamento orientado por colunas (DSM) pode melhorar o desempenho de uma BD. O mesmo pode acontecer em sentido contrário.

O modelo de dados orientado por colunas pode ser implementado ao nível do armazenamento dos dados ou ao nível do processamento dos dados. As BD orientadas por colunas podem usar várias técnicas de optimização, tais como, materialização tardia, compressão, iteração em blocos e junções invisíveis, que facilitam a implementação de um DW orientado por colunas. Porém estas técnicas usualmente não estão disponíveis nas BD orientadas por linhas. Há uma série de experiências em BD por colunas que comparam o desempenho destas, para queries OLAP, aquando da utilização de diferentes técnicas de optimização. Por exemplo, a técnica DDTA-Join (Directly Dimensional Tuple Accessing) visa especificamente melhorar o desempenho de um sistema, usando o processamento orientado à linha para a tabela de factos e o processamento oriendado à coluna nas tabelas de dimensão. As técnicas como o DDTA-Join, que apenas extraem uma única vez os dados da tabela de factos do modelo dimensional implementado, costumam apresentar melhorias significativas no desempenho de consultas neste tipo de BD [Sorjonen, 2012].

As BD orientadas por colunas não são mais que base de dados relacionais que armazenam e recuperam os dados de colunas em vez de linhas, usando SQL e álgebra convencional. Conceptualmente idênticas às BD convencionais, a utilização de um produto com uma BD orientada à coluna não requer nenhuma conversão especial, mesmo quando os seus utilizadores apresentem

novos requisitos na sua implementação. Em suma, converter completamente um modelo convencional ou torná-lo num modelo de dados híbrido, com uma conversão parcial, é um dos pontos chaves para a implementação de um sistema de *data warehousing* adequado às necessidades das organizações.

4.3 Exploração dos dados

William H.Inmon é considerado pela comunidade informática como o "pai" do conceito de *Data Warehousing*. Ralph Kimball desenvolveu uma metodologia para o desenvolvimento de projetos de SDW designada por Ciclo de Vida Dimensional do Negócio [Kimball et al., 1998]. Esta metodologia desenvolve-se em três vertentes distintas: a definição do projeto do SDW, a gestão do projeto do SDW e, por último, as tarefas a desempenhar durante o processo de implementação do SDW. Como pudemos ver na Figura 1, anteriormente apresentada no capítulo 1, em que está esquematizada a metodologia apresentada por Kimball, o primeiro passo para o desenvolvimento de um SDW é a criação de um plano de projeto. Seguida do levantamento de requisitos, modelação dimensional do sistema e análise das fontes de dados. Assim que a gestão do projeto e estruturas de dados estão definidas de acordo com os requisitos de utilizador, é sempre implementado o desenho da arquitetura do sistema, medidas de segurança e instalação do projeto, escolhendo o software e hardware necessários para acolher o projeto em desenvolvimento. Por último, o desenho lógico é implementado fisicamente e posteriormente povoado com a informação existente nas fontes de dados.

Relativamente à implementação do SDW, Kimball define que este processo pode ser executado recorrendo-se a 3 frentes de atuação distintas: a linha de ação tecnológica, a linha de ação dos dados e por último, a linha de ação das aplicações. Desta forma, contribui-se para uma redução do tempo total de conclusão do projeto, uma vez que as tarefas podem, assim, ser executadas em paralelo, especialmente as que requerem perfis de trabalho diferentes.

Os esquemas em estrela, dada a sua estrutura, permitem melhor desempenhos ao nível das *queries* relacionais e simplificam o próprio processo de ETL, uma vez que existem menos tabelas envolvidas no povoamento comparativamente ao que sucede com os esquemas em floco de neve. No entanto, quando o modelo de dados está bastante simplificado, ao ponto de se tornar um modelo bastnate desnormalizado, as tarefas dos administrador das BDs ficam mais facilitadas, em virtude de não ser necessário realizar tantas operações de junção (e outras operações de combinação de dados) para manipular os dados e, posteriormente, definir as consultas ad-hoc [Russom, 2009].

O conceito de decomposição das colunas torna muito mais fácil o processo de adição ou remoção de atributos numa tabela ou mesmo a alteração do seu tamanho. Estas alterações podem ocorrer sem os encargos de reorganizar uma tabela ou descarregar/carregar novamente os dados. Finalmente, uma vez que estão armazenadas separadamente, todas as colunas de uma tabela podem ser tratadas independentemente do sistema envolvido no processo, não havendo a necessidade de o reajustar às mudanças executadas. Apesar da desnormalização do modelo de dados transformar o processo de consulta mais eficaz, com o modelo de dados normalizado é possível obter melhores resultados na execução de *queries* típicas de um sistema de *data warehousing*. Para além de consultarem apenas as colunas solicitadas nas consultas, tal como referido anteriormente, este tipo de modelo de organização permite uma maior compressão dos dados, por motivos de uniformidade de informação, e consequentemente, proporciona a manipulação de mais dados num mesmo bloco e nas operação de leitura que tenham que ser realizadas [Matei, 2010].

Uma BD orientada por colunas consegue lidar com consultas SQL, sem que seja necessário fazer qualquer modificação. No entanto, os seus administradores têm que pensar de maneira diferente na forma como manipular os dados, sabendo que nestes sistemas têm que o fazer em termos de coleções de registos similares derivados de um conjunto de transações executadas em vez das transações de forma individual. Logo, não têm que se preocupar com o seu processamento interno, nem com a sua sintaxe da consulta, uma vez que todas as modificações serão feitas sobre a árvore de formação das *queries;* isto é, antes do planeamento das *queries*. Assim e partindo do *TPC-H benchmark*, identificaram-se duas características importantes na definição deste tipo de queries:

1. O número de atributos por tabela tem que ser elevado.
2. O tamanho do conjunto de dados deve estar num intervalo de milhares de registos.

Desta forma, pode-se concluir que, quanto maior for o número de atributos e quanto maior for o conjunto de dados, menor será o tempo de execução em DW orientados por colunas, quando comparados com os DW orientados por linhas. Ao se armazenar cada atributo em tabelas separadas, fazendo-se assim o armazenamento dos valores do atributo em questão de forma sequencial, operações de agregação como a MIN, MAX, AVG, SUM ou COUNT, são realizadas de forma mais rápida em DW orientados por colunas do que seriam em DW tradicionais [Matei, 2010].

Este tipo de modelo é, pois, mais eficiente em operações como SELECT e PROJECT, já que o processamento de consultas orientadas às colunas não precisa de se preocupar com colunas desnecessárias, apenas acedendo às colunas desejadas para realização da consulta. No entanto, para a realização de operações de junção, e dependendo do cenário aplicacional em questão, o modelo orientado por linhas parece obter melhores desempenhos. Normalmente o DW orientado por colunas acede aos valores das colunas em bloco, diminuindo o número de chamadas das funções, o que aumenta a eficiência, ao contrário do modelo orientado por linhas, que acede 1 a 2 vezes por cada tuplo.[Sorjonen, 2012].

Como não existe nenhuma restrição que impossibilite o armazenamento das colunas de forma separada uma das outras, o armazenamento orientado por colunas permite que os acessos e as operações de agregação possam ser executadas de forma paralela, melhorando consequentemente o desempenho global das consultas. Apesar destas vantagens se refletirem diretamente nos processos de negócio e nos seus custos de operação, melhorando assim o retorno total do investimento feito em DW, estas mesmas vantagens podem implicar o aumento da complexidade dos respetivos processos de ETL (*Extract, Transformation and Loading*).

4.4 Povoamento do *Data Wareh*ouse

O povoamento de um DW é uma das atividades mais importantes de um sistema de *data warehousing*. A criticidade desta atividade está muitas vezes associada ao volume de dados envolvidos. Quanto maior é o volume de dados, mais tempo é necessário para o seu processamento e carregamento no DW. O processo de povoamento de um *data warehouse* considera todo o trabalho que foi realizado em tarefas como a extração, transformação e carregamento dos dados operacionais num repositório central. Os sistemas de ETL (*Extract, Transformation and Loading*) são os responsáveis pela recolha de informação proveniente das fontes de dados, pelo seu tratamento e posterior carregamento nos sistemas de *data warehousing*. Estes sistemas executam a passagem de informação dos sistemas operacionais para os sistemas de *data warehousing* [Kimball & Caserta, 2004a]. De facto, a fase de transformação da informação é a mais complexa em todo o processo, uma vez que está dependente das regras de negócio envolvidas nos modelos do DW. Como consequência, e considerando a optimização desta etapa, existe uma grande carga de trabalho ao nível do fluxo de dados. Em contraste, as fases de extração e carregamento são facilmente aplicáveis a qualquer negócio, apresentando estratégias de optimização mais genéricas. A fase final, tida

também como uma das operações criticas do processo de ETL, pertence ao carregamento dos dados para as estruturas dimensionais. Apesar deste processo não ocorrer de forma contínua e de envolver o carregamento de um grande volume de informação num curto espaço de tempo, este é muitas vezes menosprezado aquando da construção dos sistemas de ETL. Muitas vezes, a simplicidade desta tarefa, que consiste em passar os registos tratados numa área de retenção (ARD) para o DW, pde conduzir a este tipo de situação.

Grande parte dos sistemas de data warehousing optam pela estratégia convencional de povoamento, armazenando os registos de forma individual nas estruturas do DW após o seu tratamento. Esta estratégia é típica de um DW povoado em tempo real (ou próximo de tempo real), não se verificando uma diferenciação entre as fases de transformação e carregamento, integrando cada registo no DW logo após as transformações serem realizadas. Não é de estranhar, pois, a importância que as operações de inserção assumem neste tipo de carregamento, assemelhando-se nesta fase aos sistemas operacionais nos quais cada registo é carregado de uma forma similar. Apesar do carregamento dos dados ser trivial, a grande parte do trabalho assenta em simples inserções nas tabelas de factos e nas tabelas de dimensão, a complexidade desta estratégia relaciona-se com a necessidade do motor de base de dados efetuar, previamente, um conjunto bastante diverso de operações para determinar a melhor maneira de executar os processos de consulta que serão mais tarde solicitados.

Como forma de colmatar este problema, muitos autores defendem o povoamento dos DWs recorrendo à estratégia de "*Prepared Statements*". Através desta estratégia, sempre que uma *query* é usada muitas vezes, podemos de alguma forma tentar optimizar o seu desempenho [Parsian, 2005] [Shirazi, 2003]. Com isto, pretende-se reutilizar o plano de execução em todas as situações em que a *query* é utilizada. Por exemplo, numa situação em que tivéssemos de realizar um conjunto de 100 inserções, em vez de se executarem 100 planos de execução distintas apenas seria executado um único plano de execução. Não obstante esta vantagem em termos de desempenho, o motor de base de dados tem que neste caso disponibilizar os recursos suficientes para guardar o plano de execução e garantir que este se mantem válido aquando da ocorrência de mudanças na própria BD [Shirazi, 2003]. No entanto, para um DW cujo o número de tabelas de factos seja reduzido, este factor não é assim tão negativo, uma vez que ao serem efectuadas poucas inserções distintas o seu plano de execução será válido até ao final da operação de carregamento.

Uma das estratégias mais eficazes de inserção intensiva de dados, implementada nativamente na maioria dos motores de BD mais conhecidos, é a estratégia de *bulk load*, uma alternativa à inserção de um registo de cada vez, permitindo carregar os registos todos de uma só vez, evitando a sobrecarga em operações de leitura/escrita. Mas, apesar dessa vantagem, as operações de inserção de dados que utilizam *bulk loads* levantam algumas questões. Segundo Kimball & Caserta (2004b), "As ferramentas de *bulk load* são vistas como caixas negras que geram programas que não são facilmente costumizáveis" [Amer-Yahia & Cluet, 2004]. Esta falta de informação deve-se às diferentes formas de implementação utilizadas por cada empresa responsável por desenvolver um SGBD.

Qualquer estratégia de bulk load utiliza como fonte de dados um ficheiro de texto armazenado no sistema de ficheiros de um sistema origem. Esses ficheiros podem ter dois formatos: comprimento fixo ou delimitado. No primeiro destes tipos são especificados os campos existentes no ficheiro, bem como o nome do ficheiro, a posição onde começa cada campo, o seu comprimento e o seu tipo de dados. Por vezes, é também fornecida a posição final dos campos. Já os ficheiros de texto que seguem o segundo formato incluem separadores (normalmente a vírgula, o ponto-e-vírgula, a tabulação ou *pipes*) entre cada um dos campos, como alternativa às posições de início e fim de cada um dos campo. A partir do momento em que que consegue abrir o ficheiro, respeitando cada uma das delimitações impostas, o processador da operação de *bulk load* lê os registos presentes no ficheiro e efetua a sua integração no DW [Kimball & Caserta, 2004b].

A não permissão de manipulação/tratamento dos dados durante o processo de inserção é uma das principais limitações, pelo que todos os tratamentos a realizar sobre os registos devem ser efectuados em tarefas específicas de transformação do sistema de ETL implementado. Por ultimo, o *bulk insert*, independentemente do tamanho do ficheiro, carrega todos os dados adquiridos como se de uma única transação se tratasse [Microsoft, 2012], ou seja num ficheiro com 1 milhão de linhas, por exemplo, a confirmação da inserção dos dados (*commit*) só será efectuada no final do carregamento desse milhão de registos no DW. No caso de ocorrer algum erro no processo de inserção e apesar deste permitir evitar sucessivos *commits*, todos os dados voltam a ser retirados do sistema (*rollback*), regressando-se ao estado do sistema no início da operação.

Frequentemente, a redução dos tempos de processamento em inserções de grande volume de dados em tabelas passa pela eliminação temporária dos índices que foram criados sobre uma tabela e a sua

posterior reconstrução, conseguindo-se, desta forma, ganhos de tempo consideráveis durante os processo de carregamento dos dados.

Conforme já referido, as tarefas que precedem a divulgação dos dados aos consumidores finais são usualmente desenvolvidas em áreas de tratamento dos dados, também designadas vulgarmente por áreas de estágio, de retenção ou de *back-room* [Kimball & Caserta, 2004a]. Estas tarefas são definidas em 4 áreas de tratamento de dados bem conhecidas, nomeadamente: extração, limpeza, integração e carregamento de dados (Figura 21). Estas tarefas, em conjunto, são comummente designadas sistema de ETL, tendo um importância muito grande nos SDWs, em especial.

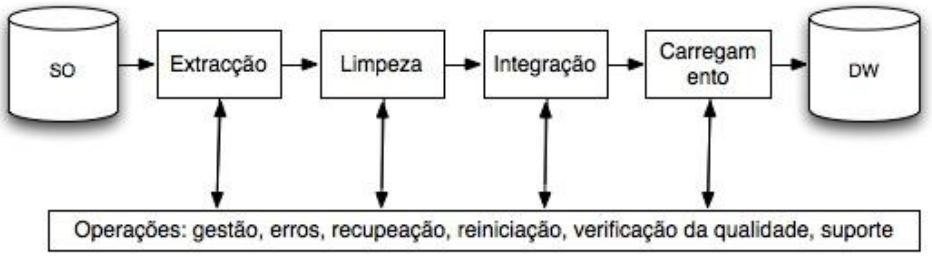

Figura 21 – Fases do *back-room* dum SDW [Kimball & Caserta, 2004a]

Um processo de ETL deve ter um conjunto de características que lhe garantam a sua relevância, seja compreensivo, apresente o nível de detalhe adequado ao problema em causa e seja de fácil interpretação [Redman, 2004]. Os dados de alta qualidade resultam de processos bem definidos e bem geridos que são capazes de criar, armazenar, manipular, mover, processar e usar adequadamente os dados disponíveis. Em [Kimball & Caserta, 2004a] são tratados vários dos assuntos relacionados com as tarefas de um sistema de ETL, que condicionam o resultado das suas ações aos requisitos impostos pela organização ou aos meios que esta dispõe. Requisitos como as exigências do negócio, a rapidez e a frescura com que os dados tem que ser distribuídos aos consumidores finais, o perfil dos dados nos sistemas operacionais, as ações de armazenamento de dados, a linhagem dos dados ou os processos de preparação de dados, podem ser encarados como constrangimentos ao funcionamento do sistema e, como tal, condicionarem a construção dos processos de ETL. Para tal, e de forma a evitar o aparecimento de custos demasiado elevados no seio de uma organização, os sistemas tem que se adaptar às realidades impostas pelas próprias organizações e modelos de negócios por elas utilizados. Noutras palavras, podemos revelar que a

qualidade dos dados de um DW, em todas as suas dimensões, é diretamente influenciado pelas ações de ETL e, consequentemente, pelos seus requisitos operacionais.

Os dados serão relevantes se forem úteis aquando da tomada de decisão. Os dados, na concretização da tarefa em mãos, devem ser aplicáveis e úteis [Pipino et al, 2002]. Assim, os dados disponibilizados pelos DWs devem ter aplicação efetiva, caso contrário, deixaria de fazer sentido ter um DW com informação que não é relevante. O povoamento de um DW com dados inúteis e desnecessários gera o aparecimento de dados dormentes e irrelevantes para o cumprimento das atividades e estes dados são os principais responsáveis pelo desinteresse pelos sistemas e pelas obstruções na execução de interrogações sobre o DW [Immon el al., 1998].

Para que um plano de extração, transformação e integração de dados seja bem definido e bem estruturado é necessário que ao ser realizado o povoamento do DW todas as restrições de integridade definidas sejam aplicadas. Após a análise dos metadados de cada uma das fontes é necessário definir um plano que permita a extração e conjugação dos dados que irão integrar as estruturas do DW. O povoamento de um DW pode ser realizado em duas fases: um primeiro povoamento, no qual se faz a consolidação de todos os factos provenientes das fontes de informação envolvidas e armazenados no DW; e um segundo povoamento, usualmente designado por povoamento periódico ou regular, no qual se realiza, periodicamente, a atualização da informação no DW. Dependendo dos requisitos do negócio, este último povoamento poderá ser realizado, por exemplo, de hora em hora, diariamente ou semanalmente.

Todas as estruturas orientadas por colunas suportam o mesmo modelo de carregamento dos dados, no entanto, a forma de lidar com cada atributo por separado implica que os SGBD orientados por colunas carreguem os dados de forma mais lenta que SGBD tradicionais. Apesar desta controvérsia, e da necessidade em aplicar um processamento adicional na análise de cada valor de entrada nos respectivos campos, este tipo de sistema de base de dados é capaz de atingir velocidade elevadas no momento de carregamento os dados. A velocidade com que é efectuado um carregamento dos dados, pode ser um problema quando se trata de DW de grandes dimensões [Bathia & Patil, 2011].

O armazenamento de dados orientados por colunas, não só elimina o armazenamento de vários índices, como também de várias views ou de agregações, podendo resultar numa redução adicional de armazenamento, com melhorias significativas ao nível de compressão dos dados. Tal como

referido, o processo típico de armazenamento de dados num DW envolve a extração, transformação e carregamento dos novos dados nas tabelas de factos e de dimensões correspondentes. Ao nível do armazenamento dos dados, as BD orientadas por colunas usam uma estrutura de armazenamento baseada nos valores que são passados em cada uma das operações de *inserts,* logo, cada uma das colunas (atributos) pode estar inserida em milhões de entradas (linhas). Assim, o carregamento dos mesmos, além de ser um processo mais demorado, implica a necessidade de mais espaço livre no disco [Bathia & Patil, 2011].

Existem muitas formas de inserir novos dados nas tabelas dos sistemas de *data warehousing* orientados por colunas. Porém, na generalidade todas essas formas seguem os mesmos modelos que se usam nas ditas BD tradicionais. Nos SGBD como o MonetDB [MonetDB, 2008] existem esquemas de inserção mais simples, como instruções independentes de INSERT ou inserções agrupadas numa única transação. No entanto, e apesar de um servidor poder conter vários CPUs, tal como em BD tradicionais, apresentam desvantagens, utilizando um único CPU de cada vez no processo de inserção e consequentemente inserindo um registo de cada vez. De forma a combater este problema permitindo a utilização dos servidores de forma mais eficaz, este tipo de SGBD apresenta outro método de carregamento, o *bulk load*. Tal como em SGBD tradicionais trata-se de uma única *query* (COPY INTO) que agarra em registos que se encontram num ficheiro inserindo-os de uma só vez nas tabelas de BD criadas. Pode ser utilizado qualquer tipo de delimitador para além do delimitador padrão de cada campo (‘,’) e do delimitador padrão dos registos (ou seja, das linhas) e, para isso, basta identificar corretamente essa alteração no processo implementado. Por exemplo, caso se pretenda alterar o delimitador entre cada um dos campos para ‘|’, de acordo com o conteúdo do ficheiro, podemos definir essa implementação de insert através da seguinte query:

```
COPY INTO nome_tabela FROM ‘ficheiro’ USING DELIMETERS ‘|’, ‘\n’;
```

Este procedimento também pode incluir a leitura de “ ” nos campos ou até mesmo de NULLs.

Quanto ao SGBD Vertica [Vertica, 2010], existem 3 métodos de carregamento dos dados: carregamento em “conta gotas” (*trickle load*), cujos dados são inseridos conforme a ordem de chegada; carregamento em massa para a memória (*bulk load to memory*), ideal para carregamentos rápidos de volumes de dados menores (apenas alguns gigabytes); e o carregamento em massa para o disco (*bulk load disk*), ideal para grandes volumes de dados (terabytes). Com a arquitetura shared-

nothing do Vertica, o carregamento pode ser realizado paralelamente e permite análises em tempo real. Para além da paralelização, também a utilização da cache da memória principal e o processamento em lote permitem alcançar carregamentos mais rápidos. O Vertica, armazena novos dados na memória principal no WOS (*Write Optimized Store*), no entanto, os dados não são comprimidos nem ordenados. Um processo assíncrono, Mover Tuple, move o conjunto de dados do WOS para o ROS (*Read Optimazed Store*), baseado no disco, comprimindo e ordenando as colunas no processo.

Não fugindo à regra, os dados podem ser carregados usando SQL *INSERT* ou o comando *COPY*. No processo "conta gotas" (*trickle load*), o SQL *INSERT* carrega os dados necessários para análise em tempo real para o WOS (*Write Optimized Store*). Para pequenos lotes de dados, os dados são carregados diretamente no WOS. Para grandes quantidades a abordagem recomendada é usar o *COPY* com a opção *DIRECT*. Desta forma, os dados serão carregados diretamente no disco do ROS (*Read Optimazed Store*).

Em conclusão, o processo bulk loading, pode utilizar o COPY com DIRECT dependendo do volume dos dados e do facto de pretender que sejam inseridos diretamente no disco. Já o trickle load, utiliza o COPY sempre que possível, no entanto, para inserção de menos de 1000 linhas utiliza SQL INSERT. Ambos garantem o máximo de utilização de memória disponível e carregam múltiplos fluxos de nós diferentes para o objectivo. No caso do SGBD C-Store existe um processo, designado por, SColumnExtracter, responsável pela leitura de um ficheiro ASCII do disco e por particioná-lo em colunas, convertendo os dados no formato de blocos binários para ai ser possível operar usando o C-Store Operators e carrega-los oficialmente no C-Store.

No entanto, e como os DW são orientados à leitura de informação, a velocidade do carregamento dos dados é uma questão secundária. A questão chave não é o tempo de carregamento dos dados, mas sim o tempo de análise. Isto é, é mais importante o tempo total decorrido, desde o recebimento de novos dados brutos, para serem adicionados ao DW em questão, até se encontrarem disponíveis para uso.

Capítulo 5

Um sistema de *Data Warehousing* orientado por colunas – sua validação e análise

5.1 Da Teoria à Prática – Um Caso de Estudo

Uma vez concluída a investigação e avaliadas as propostas dos mais variados motores para *data warehousing* foi necessário validar a informação recolhida tendo em conta as atuais práticas no mercado neste domínio, e avaliar quais as metodologias e técnicas a utilizar na implementação deste tipo de sistemas e fazer uma análise comparativa dos dois tipos de sistemas de base de dados mencionados usualmente em projetos de *data warehousing*.

Para implementação do projeto escolheu-se a AdventureWorkDW2008R2, cujo *data warehouse* contém um subconjunto das tabelas base de um sistema de dados OLTP, além da informação financeira que foi extraída a partir de uma fonte de dados externa. Desta forma, os dados são mantidos sincronizados com a base de dados OLTP, permitindo cenários típicos de carregamento e atualização dos *data warehouses* [Microsoft, 2013]. Este *data warehouse*, para além de suportar os processos de tomada de decisão sobre a comercialização de produtos, envolve também informação sobre organizações, empresas, logótipos, nomes de pessoas, lugares, eventos, endereços de email, entre outros. Apesar desta seleção e como este DW contém duas áreas distintas, finanças e vendas, optou-se apenas por uma delas: a área de vendas.

Esta área está organizada em 4 esquemas distintos e apresenta um modelo de dados menos complexo, mais intuitivo, com um número suficiente de registos e com atributos associados às tabelas de dimensão com alteração lenta. Daí selecionámos o esquema de vendas pela internet, algo que nos é muito próximo nos dias que correm e que, cada vez mais, tem maior impacto na sociedade em que estamos inseridos. Destinado a prestar serviços aos seus consumidores multilíngue apresenta, não só línguas universais como o francês, o inglês e o espanhol, como também, o chinês, o árabe, o alemão e o japonês. Este sistema de dados tem também um público alvo muito abrangente, logo a diversidade da sua oferta é grande: desde bicicletas, roupa desportiva para mulher e para homem, acessórios de montanha a roupa clássica, entre outros. Assim, pretendia-se que, a partir da sua análise, pudessem ser respondidas, entre outras, questões de negócio como as seguintes:

- Qual é a providência geográfica dos seus consumidores?
- Quais os meses do ano com mais vendas pela Internet?
- Quais os produtos vendidos num determinado período?
- Qual foi o número de vendas realizadas a cada cliente, por mês?
- Quais os locais dos clientes que compram +/- produtos pela Internet?
- Quais os produtos que são vendidos mais frequentemente?
- Quanto tempo fica, em média, um produto em espera até ser enviado?

Para se poder responder a estas questões, temos que conhecer melhor as características dos nossos sistemas de dados, quer estes sejam o DW utilizado, quer seja a parte relativa, especificamente, às vendas realizadas através da Internet.

A parte do modelo dimensional do AdventureWorkDW2008R2 (Figura 22) no qual incidiu este trabalho é composta por uma tabela de factos, FactInternetSales, acompanhada por nove dimensões que ajudam a definir o contexto dos factos em questão: Produtos (DimProduct), Categorias dos produtos (DimProductCategory), Sub-categorias dos produtos (DimProductSubCategory), Promoção (DimPromotion), Geografia (DimGeography), Vendas por território (DimSalesTerritory), Clientes (DimCustomer), Data (DimDate), Moeda (DimCurrency). Isto permite suportas medidas de análise de vendas de serviços pela Internet e contém as ordens de vendas pela Internet por cliente, por detalhe dos dados, bem com informação relativa às encomendas enviadas.

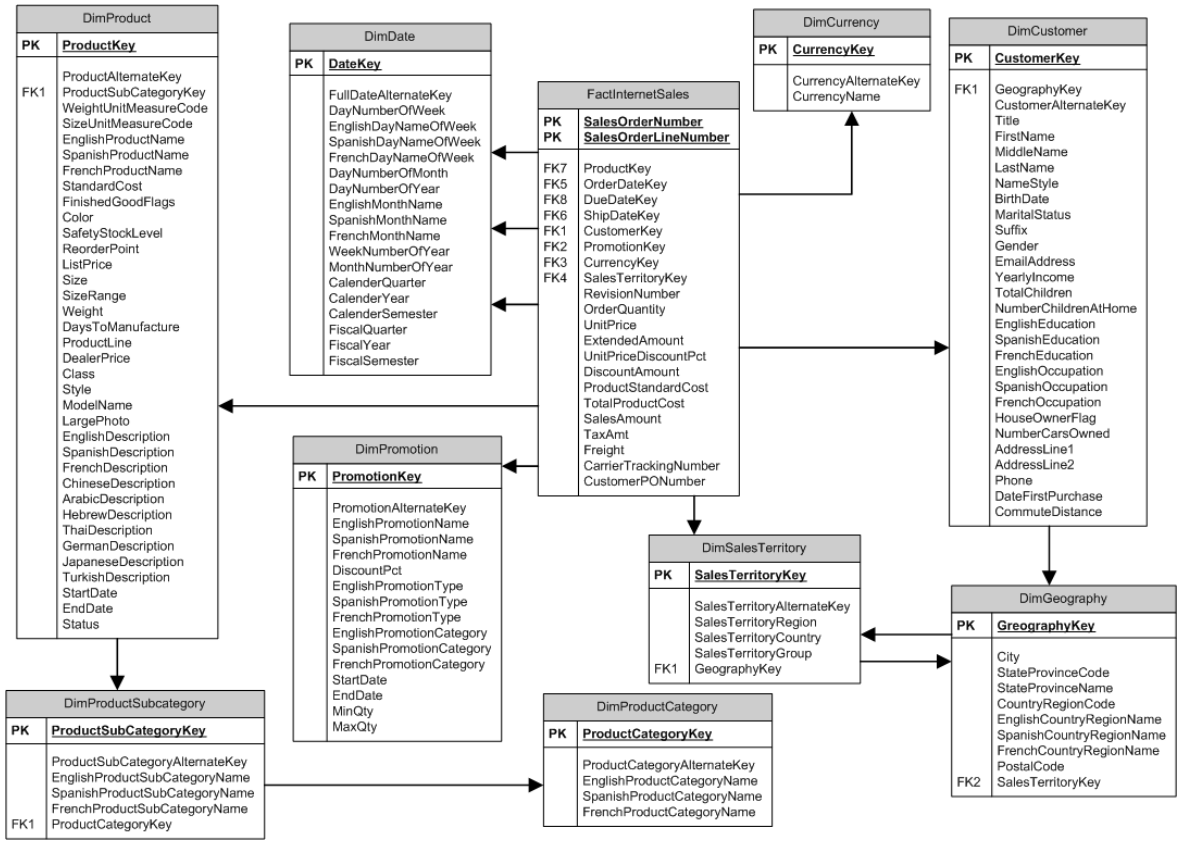

Figura 22 – Fragmento do Modelo Dimensional da base de dados AdventureWorksDW2008R2 [Microsoft, 2013]

O SDW do AdventureWorksDW2008R2 seguiu uma metodologia de implementação assente nos princípios da arquitetura em bus pura [Kimball et al., 1998]. Em especial, no que diz respeito, à configuração do DW e à reunião dos diversos DMs que sustentam as várias atividades organizacionais. Na versão atual, o volume de dados do DW anda em torno dos 60 TB de informação.

O principal objectivo deste trabalho visava melhorar o desempenho dos Sistemas de Data Warehousing, de forma a compreender melhor as distintas abordagens existentes a nível de motores de BD e, como tal, melhorar o desempenho das consultas efectuadas neste tipo de sistemas, de acordo com a melhor metodologia. Assim pretendia-se que fossem respondidas questões mais direcionadas à vertente que se analisou ao longo desta dissertação, tais como:

- Qual dos sistemas de BD apresenta o melhor desempenho em processos de seleção, aumentando-se gradualmente o número de colunas?
- Qual dos sistemas de BD apresentam melhor desempenho na execução de consultas que apresentem operações de filtragem, de junção, de ordenação e de agrupamento?
- Qual a melhor modelação dimensional para BD orientados por colunas?
- Qual o tipo de índices que poderiam ser implementados e que melhoramentos estes apresentariam na performance dos processo de satisfação das consultas em sistemas de BD orientados por colunas?
- Quais as melhorias que poderiam ser feitas na modelação dimensional dos sistemas de BD orientadas por colunas?

Para tal foi necessário entender o modelo dimensional mais apropriado a aplicar no motor de BD orientado por colunas, neste caso o MonetDB, e entender a problemática associada à conversão total, ou não, do esquema em estrela do modelo de dados da AdventureWorkDW2008R2, anteriormente apresentado.

Após a respectiva conversão do modelo dimensional, foi necessário executar algumas *queries* para análise e avaliação da informação e do desempenho dos dois tipos de sistemas. O processo de avaliação será baseado em cenários da vida real, usando como referência de apoio à decisão o TPC-H Benchmark [TPC, 2001]. O teste de referência TPC-H é utilizado por muitos fabricantes para reivindicar alguma superioridade no desempenho do DW. Esse teste baseia-se num negócio orientado a *queries* ad-hoc (em tempo real e inesperadas) e a modificações sobre os dados executadas de forma concorrente. As *queries* e o preenchimento dos dados da base de dados foram escolhidos de forma a conseguir analisar e comparar diferentes domínios, possibilitando desta forma uma visão global de toda a área de vendas pela internet do negócio.

Para consultas que implicam aceder a um menor número de atributos, considera-se que estas são direcionadas para BD orientadas por colunas. Desta forma, serão apresentadas várias tipos de *queries*, que vão desde simples esquemas de seleção, de um ou mais atributos, até agrupamentos, passando por seleções com filtros simples, sobre um ou mais campos, junções e ordenações dos dados. Por conseguinte, será possível determinar o desempenho das BD orientadas por colunas à medida que se aumenta gradualmente o número de colunas envolvidas numa *query*, tentando, assim, demonstrar se este tipo de BD é ou não apropriado para acolher um DW.

Em suma, espera-se que para grandes conjuntos de dados, as BD orientadas por colunas apresentem melhores resultados no desempenho de consultas que envolvam um pequeno grupo de atributos, apesar de se esperar piores resultados em *queries* que envolvam operações de INSERT, UPDATE ou DELETE, quando comparadas com a sua execução num ambiente de BD tradicionais.

5.2 Análise do Sistema Implementado

Ao longo desta dissertação tem-se pretendido revelar a problemática relacionada com o desempenho dos DW e, assim, introduzir os DW orientados por colunas, identificando as suas principais vantagens para *queries* típicas de sistemas OLAP. Deste modo, procurou-se abranger os aspectos mais importantes em termos de quebra de rendimento na execução de determinadas consultas ad-hoc.

Como já mencionado, o AdventureWorkDW2008R2 segue um modelo dimensional com configuração em estrela, no qual todas as tabelas de dimensão estão relacionadas com a tabela de factos. Além disso os dados das tabelas de dimensão estão desnormalizados. Um dos principais desafios era perceber se o motor de base de dados orientado por colunas que se selecionou, permitia um modelo de dados em estrela e se requeria algum tipo de conversão parcial ou total do modelo em questão.

Após o estudo das abordagens e técnicas existentes e mencionadas ao longo desta dissertação, foi necessário escolher a mais apropriada para suportar esta análise nas melhores condições.

Para este estudo comparativo sobre o desempenho dos sistemas na satisfação de *queries*, escolheu-se o sistema MonetDB, cujo sistema suporta o modelo dimensional em estrela e implementando-se o mesmo modelo de dados do AdventureWorkDW2008R2.

O modelo de dados do "AdventureWorkDW2008R2" foi implementado no sistema MonetDB como se de um motor de base de dados tradicional se tratasse. Todas as tabelas criadas seguiram a mesma estrutura e os mesmos tipos de dados (mas de acordo com as especificações específicas do MonetDB). Os dados foram carregados com a mesma estrutura que tinham no seu sistema original.

Esta conversão em termos de armazenamento de dados implicou o mesmo tratamento, quer nas operações de consulta efectuadas quer nas operações de inserção de registos, o que significa que não se teve a necessidade de reconverter junções, agregações, ou ordenações, nem mesmo em manipular previamente os dados para inserir nas tabelas criadas no MonetDB. Ou seja, caso se procedesse à conversão parcial do modelo de dados teríamos certamente implicações ao nível das *queries* a executar, mas também ao nível das inserções de novos dados no sistema, uma vez que implicaria o tratamento prévio dos dados do modelo original. De referir que, não foi instalada nenhuma interface compatível com o MonetDB, tendo-se optado por fazer a implementação de toda a BD e respectivas *queries,* de forma a que fosse possível fazer uma análise do desempenho deste tipo de motor de BD na plataforma mclient. De qualquer forma, isto é equivalente a ter-se uma interface que permite a execução de todo o tipo de comandos SQL.

Durante a primeira fase da implementação da BD para análise no MonetDB surgiu um entrave, muito recorrente, para quem manipulasse este motor de BD na plataforma mclient. Assim que se começou a executar os primeiros comandos na aplicação, esta apresentou um erro indicando uma falha no reconhecimento da libraria ibmapi.lib. Para resolver essa situação, e tal como implementado por outros utilizadores o fizeram e documentaram, recorreu-se à batch disponibilizada pelo MonetDB para realizar a implementação pretendida. Para além deste entrave inicial, outros obstáculos apareceram ao nível da implementação da BD. Sempre que se executava o comando de criação da BD, o sistema informava o sucesso da operação realizada, mas sem nunca reconhecer a BD que foi criada, invocando a BD de demonstração sempre que se pretendia efetuar algum tipo de transação. Desta forma, e após algumas pesquisas, recorreu-se à alteração da batch do mserver5, acrescentando a informação relativa a base de dados que se pretendia criar e explorar, em particular:

--dbname = "adventurework" /* neste caso optou-se por adicionar uma base de dados designada por "*adventurework*"), o que originou a seguinte linha de informação no ficheiro batch já referido:

```
"C:\ProgramFiles\MonetDB\MonetDB5\mserver5.exe" -set "prefix=C:\Program
Files\MonetDB\MonetDB5" --set "exec_prefix=%MONETDB%" %MONETDBFARM% %* --
dbname="adventurework"
```

Assim, estas ações evitaram a criação da forma mais convencional, isto é, fazendo a criação da BD em questão através de instruções de SQL.

Após ter sido criada a BD, tratou-se da criação de cada uma das tabelas de dimensão e, por último, da tabela de factos. Tal como em sistemas de BD tradicionais, e seguindo a metodologia proposta por R. Kimball, foi necessário ter em conta as dependências que se verificavam entre as várias tabelas. Nesse sentido, começou-se por criar as tabelas cujas chaves primárias seriam chaves estrangeiras nos esquemas de outras tabelas. Em suma, o armazenamento de cada uma das tabelas num sistema orientado à coluna em nada influenciou a criação do modelo de dados pretendido. Todavia, teve-se sempre presente a preocupação da passagem, de uma forma compatível, do tipo de dados associado com a BD "AdventureWorkDW2008R2" para o MonetDB. A diferença entre os tipos de dados suportado no MonetDB e no SQL Server, implicou a redefinição de alguns campos em termos de domínio nas distintas tabelas do "AdventureWorksDW2008R2" (Tabela 2).

AdventureWorkDW2008R2	**MonetDB**
DATETIME	TIME
NVARCHAR	VARCHAR
BIT	TINYINT
FLOAT	REAL
MONEY	REAL

Tabela 2 – Correspondência entre tipos de dados

Por exemplo, e após a implementação de cada uma das tabelas de dimensão necessárias criou-se a tabela de factos "FactInternetSales" no MonetDB procedendo-se ao seguinte create na BD:

```
CREATE TABLE FactInternetSales(
       ProductKey INT,
       OrderDateKey INT,
       DueDateKey INT,
       ShipDateKey INT,
       CustomerKey INT,
       PromotionKey INT,
       CurrencyKey INT,
       SalesTerritoryKey INT,
       SalesOrderNumber VARCHAR(40),
       SalesOrderLineNumber TINYINT,
       RevisionNumber TINYINT,
       OrderQuantity SMALLINT,
       UnitPrice REAL,
       ExtendedAmount REAL,
       UnitPriceDiscountPct DOUBLE,
       DiscountAmount DOUBLE,
       ProductStandardCost REAL,
       TotalProductCost REAL,
       SalesAmount REAL,
       TaxAmt REAL,
       Freight REAL,
       CarrierTrackingNumber VARCHAR(50),
```

```
CustomerPONumber VARCHAR(50));
```

De seguida, procedeu-se à criação da chave primária e das chaves estrangeiras associadas a cada tabela, desenvolvendo-se e executando-se as *queries* que apresentamos de seguida, cujos comandos mantêm o mesmo padrão das BD tradicionais:

- Criação da chave primária:

```
ALTER TABLE FactInternetSales
       ADD PRIMARY KEY(SalesOrderNumber,SalesOrderLineNumber);
```

- Criação das chaves estrangeiras:

```
ALTER TABLE FactInternetSales
       ADD FOREIGN KEY(ProductKey)
       REFERENCES DimProduct(ProductKey);
ALTER TABLE FactInternetSales
       ADD FOREIGN KEY (OrderDateKey)
       REFERENCES DimDate(DateKey);
ALTER TABLE FactInternetSales
       ADD FOREIGN KEY (DueDateKey)
       REFERENCES DimDate(DateKey);
ALTER TABLE FactInternetSales
       ADD FOREIGN KEY (ShipDateKey)
       REFERENCES DimDate(DateKey);
ALTER TABLE FactInternetSales
       ADD FOREIGN KEY (CustomerKey)
       REFERENCES DimCustomer(CustomerKey);
ALTER TABLE FactInternetSales
       ADD FOREIGN KEY (PromotionKey)
       REFERENCES DimPromotion(PromotionKey);
ALTER TABLE FactInternetSales
       ADD FOREIGN KEY (CurrencyKey)
       REFERENCES DimCurrency(CurrencyKey);
ALTER TABLE FactInternetSales
       ADD FOREIGN KEY (SalesTerritoryKey)
       REFERENCES DimSalesTerritory(SalesTerritoryKey);
```

Na sequência desta linha de ação, e depois de se desenhar e construir o modelo de dados de suporte às necessidades de informação dos utilizadores finais - os utilizadores da parte relativa às vendas através da Internet da BD "AdventureWorkDW2008R2", foram também desenvolvidos os vários procedimentos de extração, transformação, limpeza e alimentação dos dados a incorporar o sistema de dados a incorporar no MonetDB.

Para evitar inserir os registos nas respectivas tabelas através do comando INSERT do SQL, o que implicaria inserir os registos um a um, processo bastante moroso em muitas situações, optou-se pela inserção dos dados da BD através do *bulk load* que o MonetDB disponibiliza. Para isso foi necessário criar um utilizador com as credenciais adequadas para aceder ao mclient e assim fazer a inserção dos

registos de uma forma bastante mais expedita. Este processo de povoamento foi dividido em 3 fases distintas:

1. Criação do utilizador: mclient -u USERNAME -p PASSWORD;
2. Lançamento do processo Java que utiliza o jdbcclient.jar, que permite a execução de futuras importações através da shell do Windows: java -jar C:\jdbcclient.jar -dadventurework –umonetdb;
3. Colocar os registos nas tabelas correspondentes: mclient -d database -s "COPY INTO table FROM STDIN USING DELIMITERS ',','\\n','\"" - < file.

Todo este processo de importação implicou uma prévia exportação dos dados de cada uma das tabelas da base de dados do AdventureWorkDW2008R2 para os respectivos ficheiros (anteriormente identificados por "f*ile*"). No entanto, nesta situação, foi inevitável ter em atenção a formatação de cada um dos ficheiros utilizados para importar os dados, nomeadamente, a separação de cada uma dos campos das tabelas entre vírgulas, as plicas nos registos e o enter entre os vários tuplos para garantir a correcta inserção de cada tuplo. Contudo, o comando import tem várias formas de ser implementado, dependendo isso do tipo de dados com que o utilizador tem que lidar nos processos de extração. Por exemplo, os registos de dados em questão poderão ter pipes ("|") em vez da tão comum vírgula para separar cada um dos atributos da tabela.

Por último foi possível passar à fase da execução das *queries* típicas de um sistema OLAP. Estas *queries* foram necessárias para realizar a análise do desempenho dos dois tipos de sistemas de BD em causa e, consequentemente, avaliá-los comparativamente. É isso que trataremos nas secções seguintes.

5.3 *Queries* sobre Colunas – Análise de Desempenho

O processamento de *queries* é um dos métodos mais utilizados na extração da informação dos sistemas de BD tradicionais e nos sistemas orientados por colunas. No caso do sistema MonetDB uma interrogação, geralmente escrita numa linguagem de alto nível, é transformada numa outra interrogação equivalente escrita numa linguagem de nível inferior: a linguagem MIL. A nova interrogação deve ser eficiente e traduzir fielmente a questão inicial, de forma a se poder obter a

informação desejada. Para que fosse possível analisar o comportamento de vários sistemas de BD foram desenvolvidas 15 *queries* específicas, abrangendo desde cenários de interrogação bastante simplistas até outros mais complexos, envolvendo operações de agrupamento, típicas de cenários de suporte à decisão. Vejamos, então, de seguida essas *queries* bem como as suas respetivas descrições.

***Queries* simples**

Query 1 – Identificação do número da encomenda e do número de linha de encomenda de todas as vendas registadas pela Internet.

```
SELECT salesordernumber, salesorderlinenumber
      FROM FactInternetSales;
```

```
SQL: 1359 ms
MonetDB: 1300 ms
```

Query 2 – Identificação dos produtos disponíveis para venda através da Internet, com apresentação do modelo do produto, ultimo preço registado e estado no sistema.

```
SELECT ProductKey, EnglishProductName, ModelName, ListPrice, Status
      FROM DimProduct;
```

```
SQL: 109 ms
MonetDB: 100 ms
```

Query 3 – Identificação dos dados gerais das vendas pela internet.

```
SELECT *
      FROM FactInternetSales;
```

```
SQL: 2812 ms
MonetDB: 3323 ms
```

***Queries* com critérios de filtragem**

Query 4 – Informação das vendas pela Internet, com valores de venda superiores a '2000'€.

```
SELECT *
      FROM FactInternetSales
      WHERE salesamount > 2000;
```

```
SQL: 187 ms
MonetDB: 5448 ms
```

Query 5 – Média dos valores de venda para encomendas realizadas entre '01-01-2007' até '2008'.

```
SELECT AVG(SalesAmount)
       FROM FactInternetSales
       WHERE OrderDateKey >= 20070101 AND OrderDateKey < 20080000;

SQL: 15 ms
MonetDB: 10 ms
```

Query 6 – Vendas registadas através da Internet desde o dia '2005-07-01', sem desconto efetuado.

```
SELECT *
       FROM FactInternetSales
       WHERE DiscountAmount = '0'AND OrderDateKey > 20050701;

SQL: 1078 ms
MonetDB: 3011 ms
```

***Queries* envolvendo operações de junção**

Query 7 – Para cada produto vendido pela Internet, num determinado período de tempo, apresente a informação do último preço estipulado, categoria geral e sub-categoria de produtos a que pertence, bem como a listagem de todas as suas encomendas, quantidade vendida e preço total.

```
SELECT PROD.EnglishProductName, PROD.ProductAlternateKey, PROD.ListPrice,
PSC.EnglishProductSubcategoryName
, PC.EnglishProductCategoryName, S.salesordernumber, S.OrderQuantity,
S.SalesAmount
       FROM FactInternetSales S, DimProduct PROD,
       DimProductSubcategory PSC, DimProductCategory PC, DimDate d
       WHERE S.ProductKey = PROD.ProductKey
       AND PROD.ProductSubcategoryKey = PSC.ProductSubcategoryKey
       AND PSC.ProductCategoryKey = PC.ProductCategoryKey
       AND s.OrderDateKey = d.datekey
       AND d.FullDateAlternateKey <= '2007-01-01'
       AND d.FullDateAlternateKey > '2005-07-01';

SQL: 859 ms
MonetDB: 789 ms
```

Query 8 – Apresentar as vendas pela Internet, sem promoções, cujas encomendas foram feitas antes do ano de '2007' e entre os meses de 'Janeiro' a 'Junho'.

```
SELECT f.salesordernumber, f.salesamount, p.EnglishPromotionName,
cust.customeralternatekey, cust.firstname, cust.lastname, cust.addressline1,
cust.addressline2, dat.calendaryear, dat.monthnumberofyear
       FROM FactInternetSales f, DimCustomer cust, DimDate dat, DimPromotion
```

```
        p
        WHERE f.customerkey = cust.customerkey
        AND f.orderdatekey = dat.datekey
        AND f.PromotionKey = p.PromotionKey
        AND dat.calendaryear < '2007'
        AND dat.monthnumberofyear <= '6'
        AND f.TotalProductCost > 500
        AND f.DiscountAmount = '0';
```

```
SQL: 187 ms
MonetDB: 170 ms
```

Query 9 – Apresentar as vendas realizadas pela Internet cujos produtos têm um custo standard inferior ou igual a 30 e os clientes com um rendimento anual inferior a 50000, incluindo alguma informação dos produtos e clientes envolvidos.

```
SELECT f.salesordernumber, d1.productalternatekey, d2.customeralternatekey
        FROM FactInternetSales F, DimProduct D1, DimCustomer D2
        WHERE F.ProductKey = D1.ProductKey
        AND F.CustomerKey = D2.CustomerKey
        And D1.StandardCost <= 30 AND D2.YearlyIncome <= 50000;
```

```
SQL: 4046 ms
MonetDB: 3969 ms
```

Queries envolvendo operações de ordenação

Query 10 - Identificação dos clientes que encomendaram produtos pela Internet, apresentando-os ordenados descendentemente por número de encomenda, ou seja, apresentar uma lista com informação desde o cliente com o maior número de encomendas até ao cliente que efetuou menos encomendas, a fim de se poder definir estratégias de marketing para cada tipo de clientes.

```
SELECT CustomerKey, COUNT(customerkey)
        FROM FactInternetSales
        GROUP BY CustomerKey
        ORDER BY COUNT(CustomerKey) DESC;
```

```
SQL: 859 ms
MonetDB: 405 ms
```

Query 11 - Identificação dos produtos mais encomendados, para se determinar qual o mercado que a empresa deve dar mais atenção de forma a melhorar o seu lucro.

```
SELECT ProductKey, COUNT(productkey)
```

```
FROM FactInternetSales
GROUP BY ProductKey
ORDER BY COUNT(ProductKey) DESC;
```

```
SQL: 859 ms
MonetDB: 400 ms
```

Query 12 – Quais foram as vendas que foram registadas pela Internet sem promoção; apresentar essa informação ordenada da encomenda mais cara para a encomenda mais barata e pela identificação do cliente; esta query pode ser utilizada para avaliar uma possível recompensa para os melhores clientes.

```
SELECT f.salesordernumber, f.salesamount, p. EnglishPromotionName,
       cust.customeralternatekey, cust.firstname, cust.lastname,
       cust.addressline1, cust.addressline2,
       dat.calendaryear, dat.monthnumberofyear
       FROM FactInternetSales f, DimCustomer cust,
              DimDate dat, DimPromotion p
       WHERE f.customerkey = cust.customerkey
              AND  f.orderdatekey = dat.datekey
              AND f.PromotionKey = p.PromotionKey
              AND f.DiscountAmount = '0'
       ORDER BY f.SalesAmount desc, cust.customeralternatekey;
```

```
SQL: 859 ms
MonetDB: 289 ms
```

***Queries* envolvendo operações de agrupamento**

Query 13 - Em média, quanto tempo demora um produto demora a ser enviado ao cliente, desde o momento em que ele fez a sua encomenda.

```
SELECT DISTINCT f.ProductKey, AVG(DATEDIFF(D,d.FullDateAlternateKey,
       sd.FullDateAlternateKey)) as media
       FROM FactInternetSales f, DimDate d, DimDate sd
       WHERE f.OrderDateKey = d.DateKey AND f.ShipDateKey = sd.datekey
       AND DATEDIFF(D,d.FullDateAlternateKey, sd.FullDateAlternateKey) >= 0
       GROUP BY f.ProductKey
       ORDER BY f.ProductKey;
```

```
SQL: 78 ms
MonetDB: 68 ms
```

Query 14 – Apresentar a identificação do número de encomendas por localidade, média do preço das encomendas e soma total das encomendas, cuja contagem das encomendas é

superior a 100 registos, a fim de conhecer quais as localidades que mais estão referenciadas nas encomendas através dos clientes.

```
SELECT g.city AS localidade, COUNT(f.SalesOrderNumber) AS encomendas,
       AVG(f.SalesAmount) AS media_vendas, SUM(f.SalesAmount) AS soma_vendas
       FROM FactInternetSales f, DimSalesTerritory t, DimGeography g
       WHERE f.salesterritorykey = t.salesterritorykey
             AND g.SalesTerritoryKey = t.SalesTerritoryKey
       GROUP BY g.city, f.OrderQuantity
       HAVING COUNT(f.OrderQuantity) > 100;

SQL: 109 ms
MonetDB: 80 ms
```

Query 15 – Apresentar o maior, o menor e a média do valor das vendas por categoria de produto, para decidir até que ponto aquele produto está a ser rentável ou não para o negócio.

```
SELECT pc.EnglishProductCategoryName AS "Category"
  ,SUM(f.SalesAmount) AS "Total Sales"
  ,MIN(f.SalesAmount) AS "Lowest Sale Amount"
  ,MAX(f.SalesAmount) AS "Highest Sale Amount"
  ,AVG(f.SalesAmount) AS "AVG Sales"
       FROM FactInternetSales f, DimProduct p, DimProductSubcategory psc,
       DimProductCategory pc
       WHERE f.ProductKey = p.ProductKey
       AND p.ProductSubcategoryKey = psc.ProductSubcategoryKey
       AND psc.ProductCategoryKey = pc.ProductCategoryKey
       GROUP BY pc.EnglishProductCategoryName
       ORDER BY SUM(f.SalesAmount) desc;


SQL: 46 ms
MonetDB: 68 ms
```

Na seguinte tabela são apresentados de uma forma resumida os resultados obtidos, quer nas BD tradicionais quer nas BD orientadas por colunas para cada uma das *queries* anteriormente implementadas, indicando percentualmente a melhoria (ou não) do MonetDB, assim como, a respectiva deslocação a nível de desempenho para cada umas das queries:

Tipo de Query	Query	Execução da Query (ms)		Desempenho	Ganho (%)
		SQL	MonetDB		
Simples	1	1359	1300	↑	4%
	2	109	100	↑	8%
	3	2812	3323	↓	-18,17%
Com critérios de filtragem	4	187	5448	↓	-2813%
	5	15	10	↑	33,3%
	6	1078	3011	↓	-179%
Com junções	7	859	789	↑	8,15%
	8	187	170	↑	9,09%
	9	4046	3969	↑	1,90%
Com ordenações	10	859	405	↑	52,9%
	11	859	400	↑	53,4%
	12	859	289	↑	66%
Com agrupamentos	13	78	68	↑	12,8%
	14	109	80	↑	26,6%
	15	46	68	↓	-47,8%
	Observações:				

Tabela 3 – Métricas de desempenho entre SQL e MonetDB

Em termos globais, e com base na tabela anterior, a deslocação a nível de desempenho para o sistema MonetDB é maioritariamente positivo em relação ao sistema de BD tradicional utilizado. Aparte de algumas *queries* simples, com filtragens ou com agrupamentos que registaram piores resultados no sistema de BD orientado por colunas, conclui-se que operações com junções e ordenações, apresentam na sua totalidade melhores desempenhos. No entanto, é importante referir que as perdas de performance que foram registadas no MonetDB em relação ao sistema de BD tradicional, com critérios de filtragem e agrupamentos registaram uma perca de performance muito

maior que os ganhos nos outros tipos de query. Dependendo do tipo de pesquisas que se pretendem, é necessário avaliar na fase de planeamento qual o tipo de sistema de BD adequado.

Os testes que foram desenvolvidos, tendo em conta a avaliação do desempenho do sistema na realização das *queries* apresentadas nesta secção, foram realizados numa instalação do sistema operativo Windows 7. Sob o ponto de vista de hardware utilizado, os testes foram efectuados num computador com as seguintes características: processador 2.7 GHz Intel Core i7 e com memória 4 GB.

Para a realização deste trabalho foram utilizadas, maioritariamente, ferramentas *open-sourc*e, com exceção do editor de texto e a ferramenta utilizada na exploração gráfica. No que diz respeito às ferramentas *open-source*, foi utilizado para gestão de base de dados o SQL Server 2008 R2 Enterprise, bem como todas as ferramentas gráficas associadas. A opção da utilização destas ferramentas recaiu, no facto de serem gratuitas e já conhecidas.

5.4 Análise do comportamento

Atendendo apenas à análise de desempenho, à leitura de consultas típicas de sistemas de DW e apoiada no suporte de decisão padrão anteriormente mencionado - o TPC-H -, dividiu-se as *queries* para análise em 5 grupos distintos, aumentando gradualmente a complexidade das *queries* em questão com a introdução de validações adicionais.

Divididas em grupos de 3 *queries* por tipo de consulta, as primeiras *queries* implementadas para avaliação e comparação da performance dos dois tipo de sistemas de BD eram destinadas há identificação da BD cujas consultas variavam apenas no número de atributos acedidos. Pôde-se verificar que, tanto a *query* 1 como a *query* 2 registaram melhores resultados em termos de desempenho no sistema de BD orientado por colunas. Porém, na terceira *query*, cuja consulta selecionava todos os atributos da tabela de factos, verificou-se uma situação contrária.

Posteriormente, executaram-se 3 *queries* com critérios de filtragem definidos. A *query* 5, utilizada para detectar a média da quantidade de venda de todas encomendas registadas num determinado espaço de tempo, registou melhores resultados. Esta *query* como acede apenas a 2 colunas da tabela de factos, uma para retirar a média dos valores solicitados e outra para as datas das encomendas,

não necessitou de aceder a várias partes do disco para encontrar os diferentes atributos, que ao contrário da BD tradicional necessitou de extrair toda a linha de informação e só posteriormente fez a seleção dos atributos necessários. No entanto, as *queries* 4 e 6 deste grupo posicionam-se na situação anteriormente apresentada, ou seja, necessitam de aceder a diferentes partes do disco para construção do tuplo com a filtragem solicitada. Desta forma, apresentaram piores desempenhos em MonetDB quando comparadas com a BD tradicional.

Aumentando o grau de complexidade e por *queries* típicas de um ambiente de DW implicarem na sua maioria a fusão de colunas de distintas tabelas, sendo necessário aceder a mais que uma tabela do modelo de dados para restringir a informação ao pedido solicitado, foram construídas 3 *queries* que implicassem junção entre distintas tabelas do modelo de dados do AdventureWorkDW2008R2. Por conseguinte pode-se verificar que, apesar das junções entre as várias tabelas do modelo de dados em cada uma das consultas, sempre se registaram melhores desempenhos com o sistema MonetDB. Isto talvez tenha sido causado por se terem implementado junções com critérios de filtragem, o que impediu a junção total dos atributos das entidades entre todas as tabelas solicitadas. Quanto às *queries* 7, 8 e 9, estas apresentaram melhores desempenhos no sistema MonetDB, mas com uma margem de diferença pequena, o que pode implicar que caso se aumentasse o número de atributos a aceder ou até mesmo se aumentasse o número de filtragens, estas poderiam apresentar um comportamento diferente.

De seguida, seguiram-se as *queries* de ordenação e de agrupamento de informação. A *queries* 10, 11 e 12, permitem obter, respectivamente, os clientes que encomendaram produtos pela Internet, ordenados por numero descendente do número de encomendas, os produtos mais e menos encomendados através da Internet e as vendas aí também registadas sem promoção, ordenadas da encomenda mais cara para a encomenda mais barata e pela identificação do cliente. Todas estas *queries* apresentaram sempre melhores desempenhos no sistema MonetDB. Tal circunstância, permite-nos identificar uma certa melhoria nos processos de consulta típicos de sistemas OLAP. Desta forma, a identificação dos cliente com mais vendas ou dos produtos com mais vendas registam melhores desempenhos, o que, obviamente, para as organizações se torna uma mais valia.

No entanto, no que diz respeito às ultimas 3 *queries* implementadas, que envolvem para além de filtragens, junções e ordenações, operações de agregação associadas aos agrupamentos de informação, o que requer o processamento de médias e de somas de valores. Por exemplo, na *query*

13 pretende-se saber, em média, quanto tempo um produto demora a ser enviado, desde o momento em que é feita a encomenda até à data do seu envio. Por sua vez, na *query* 14 requer-se o número de encomendas por localidade, a média do preço das encomendas e a soma total das encomendas, cuja contagem das encomendas seja superior a 100 registos. A última *query* tem como objetivo uma análise global de cada um dos produtos existentes para venda na Internet, ou seja, identificar o maior, o menor e a média do valor das vendas por categoria de produto. Assim, será possível detectar quais os melhores produtos que disponibilizam, quais os mais apelativos, quais os que obtêm maior lucro, e com isso decidir até que ponto aquele produto está a ser ou não rentável para o negócio, bem como permite saber as localidades dos clientes que mais encomendas registam. Com essa informação pode-se decidir inúmeras questões acerca de eventuais promoções ou sobre novas formas de captação das atenção para novos produtos. Ao contrário das restantes queries deste último grupo, a query 15 registou piores resultados de desempenho, uma vez que necessitou de aceder de forma paralela às mesmas colunas "salesamount" da maior tabela do sistema de BD criado: a FactInternetSales.

Analisando o comportamento de cada *query* apercebemo-nos que à medida que se aumentava o número de colunas consultadas pelas *queries,* sobre uma mesma tabela, quer as base de dados tradicionais como as bases de dados orientadas por colunas, o tempo de processamento também ia aumentando – nada de surpreendente. No entanto, caso se pretendesse aceder a um grande número de atributos de uma tabela estas apresentavam piores registos de desempenho que as BD tradicionais, já nessas bases de dados é necessário aceder a distintas partes do disco. Além disso, em algumas outras situações, também se verificou que o facto das consultas implicarem o acesso em paralelo a muitas colunas, apresentou uma redução a nível de performance de execução, em relação aos sistemas de BD tradicionais. Apesar disso, as base de dados orientadas por colunas apresentaram, na sua totalidade, melhores resultados nos tempos de execução das consultas orientadas a sistemas de DW.

5.5 Contributos para o melhoramento do sistema

Antes da realização dos testes comparativos de desempenho, pensou-se que sempre que o número de atributos acedidos pelas consultas fosse reduzido, estas obteriam sempre melhores resultados em termos de desempenho. No entanto o número de junções integradas nas consultas e o número de consultas paralelas sobre muitas colunas, cujo número de registos era muito grande, fez com que se

obtivessem piores resultados do que aquilo que era expectável. Uma possível solução para casos como estes seria a introdução de índices. Com eles poderíamos obter melhores resultados na implementação de determinadas *queries,* nomeadamente em *queries* que envolvessem operações de ordenação e de agrupamento.

Com a promoção de algum tipo de indexação, a *query* 15, por exemplo, beneficiaria com a criação de um índice sobre a soma da quantidade de vendas, por ordem decrescente, e sobre o produto em questão. Dessa forma, poderíamos registar que teríamos um relacionamento mais rápido entre os dados relacionados com o produto e com as vendas. Todavia, em *queries* como as duas primeiras da secção de ordenação (*query* 10 e *query* 11), poderíamos ter ainda melhores resultados se se indexasse os clientes e os produtos, respectivamente, ordenando-os de forma decrescente. Em suma, sempre que se pretenda efetuar qualquer tipo de ordenação de dados em processos de consulta, mesmo que as BD orientadas por colunas apresentem melhores registos que as BD tradicionais, poderíamos ainda aumentar a diferença de tempo de resposta entre estes dois sistemas com criação de índices, já com a ordenação das colunas a consultar. A isto acresce o facto de melhor performance, uma vez que não haveria necessidade em percorrer toda a coluna para obter a informação desejada.

Outra das possíveis melhorias passa pela conversão do modelo de dados, obtendo-se em determinadas consultas melhores resultados a nível de desempenho. No entanto, apesar dessa melhoria a nível da performance das consultas, a nível de povoamento do *data warehouse* o cenário seria outro. Desta forma, algumas das tabelas de dimensão convertiam-se, principalmente as tabelas de dimensões com alteração lenta, em campos da tabela de factos do mesmo modelo.

Desta forma, a desnormalização tem sido utilizada como uma técnica de melhoramento de desempenho de consultas, pela redução do número de junções que são necessárias realizar entre as tabelas envolvidas na consulta. No entanto, a desnormalização não é muito útil em sistemas de BD orientados por colunas (pelo menos quando acolhem esquemas com configuração em estrela). Tendo em conta que a principal característica do esquema em estrela passa pela presença de dados altamente redundantes, melhorando o desempenho, e se as BD orientadas por colunas acedem apenas às colunas que necessitam para realizar as consultas desejadas, a desnormalização deste esquema poderia implicar valores de execução piores. Dependendo do tipo de consultas a efectuar esta técnica pode melhorar ou não o desempenho nos sistemas de BD orientados por colunas. Sendo assim, e caso se implementasse este tipo de técnica, por exemplo, caso se implementasse a tabela de dimensões correspondente a datas (DimDate) na tabela de factos a performance das *queries* como a

query 5, 6 e 8, cujas consultas se restringem a um determinado espaço de tempo apresentariam piores tempos de execução, porque necessitariam aceder a mais que uma coluna da tabela de factos (maior tabela em quantidade de registos).

Desta forma e para ser possível registar o melhor tempo possível na execução de consultas típicas de sistemas de DW, é necessário efetuar previamente um estudo sobre o tipo de dados que temos nas tabelas de factos e de dimensões, bem como ter uma noção clara do tipo de consultas que se realizam usualmente no contexto do negócio da organização em causa.

Capítulo 6

Conclusões e Trabalho Futuro

6.1 Conclusões

Numa sociedade em constante crescimento a vantagem competitiva de uma empresa é alcançada de acordo com a sua habilidade de adquirir e manusear a maior quantidade de dados existentes de modo adequado ao uso e aos objectivos de cada empresa. Para que isto se torne possível, é necessário que as organizações disponham de mais e melhores meios de tratamento e exploração de informação. O desenvolvimento de um Sistema de *Data Warehousing* permite às organizações atingir tais objectivos disponibilizando meios e serviços para analisar, planear e reagir de imediato às várias necessidade de informação impostas pela natural evolução do mercado.

Neste trabalho foram descritos e analisados dois tipos distintos de sistemas de base de dados: os sistemas de BD tradicionais e os sistemas de BD orientados por colunas. A grande diferença, como sabemos, reside na forma como é realizado o mapeamento das tabelas. Enquanto, que num sistema de BD tradicional existe um sistema de armazenamento linha-por-linha, isto é, cada linha de uma tabela de base de dados é armazenada separadamente, seguida de outra linha e assim consecutivamente até todas as linhas da tabela estarem armazenadas, num sistema de BD orientado por colunas, cada atributo de uma tabela de base de dados é armazenado separadamente, mantendo-se assim toda a informação relativa a um dado atributo junta. Ao se colocarem dados semelhantes juntos, minimiza-se, consequentemente, o tempo das queries que sejam lançadas sobre tais atributos, situação que é bastante típica de um ambiente de *data warehousing.*

Neste trabalho de dissertação foram abordados três tipos distintos de sistemas de gestão de base de dados orientados por colunas, com o objectivo final de ajudar na escolha do melhor motor de BD, consoante as características dos dados, para obtenção e concretização dos testes planeados para análise de desempenho. Foi com o sistema MonetDB que, para além de ser software *open-source* e de proporcionar um ambiente flexível de utilização, possibilitou a implementação dos nossos testes em distintas plataformas. Além disso foi o motor de BD que melhor se adaptou às nossas necessidades.

Neste trabalho de dissertação identificou-se quatro tipos de optimizações que podem ser usadas para melhorar o desempenho deste tipo de sistemas, nomeadamente: a compressão, a materialização, os blocos de iteração e as junções invisíveis (*invisible joins*). Depois, foram analisados vários modelos de dados de DW que foram aplicados sobre um sistema de BD orientado por colunas. Com isso, verificou-se que um dos pontos mais pertinentes quando se pretende implementar um Sistema de *Data Warehouse* orientado por colunas é a conversão do modelo dimensional. Tal conversão influencia o processo de exploração e povoamento dos dados contidos num DW. Visto este tipo de sistema oferece maior flexibilidade na modelação dimensional dos dados, em particular porque o peso da linha não é mais um problema na exploração dos dados e do povoamento do DW, registou-se que o sistema que se analisou mantinham os mesmos padrões que os Sistemas de *Data Warehouse* orientados por linhas. No entanto, neste tipo de processo temos que ter sempre em conta o modelo dimensional implementado e que apenas a representação interna dos dados difere. De seguida, foi realizada uma análise com base no modelo de dados da AdventureWorksDW2008R2. Mantendo-se o modelo de dados inicial, isto é, mantendo a mesma lógica do modelo, com as mesmas tabelas de dimensão e a mesma tabela de factos, no mesmo esquema (esquema em estrela), foram aplicadas 15 *queries* para avaliação da performance entre os dois sistemas analisados. Por fim, após a obtenção e explicação dos resultados finais, foi apresentada uma breve conclusão acerca dos resultados obtidos. De notar que, par cada uma das *queries* estudadas, a escolha dos atributos variou consoante as necessidades de trabalho em causa, tendo todas elas sido realizadas com base nos testes TPC-H, que, como sabemos, fornecem um conjunto de *queries* padrão especialmente desenvolvidas para testes de desempenho de SDW.

Divididas em grupos de 3 *queries,* por tipo de consulta, as primeiras três *queries* implementadas para a análise pretendida apresentavam como principal objectivo identificar a diferença entre os sistemas de BD orientados por colunas e os sistemas de BD comuns em termos de desempenho das consultas

pretendidas, cujo número de atributos ía aumentando gradualmente de query para query. De seguida, analisou-se o desempenho do sistema com queries envolvendo critérios de filtragem. Como estes dois tipos de análise apenas envolveram uma tabela de cada vez, e como a maioria dos DW usualmente registam consultas sobre mais que uma tabela, centrou-se a nossa atenção para a análise de consultas que exigissem a aplicação de operações de junção. Apesar da realização destes três tipos de consultas terem permitido identificar algumas vantagens e desvantagens de um sistema em relação ao outro, não se pode descartar neste estudo processos como a ordenação de resultados ou a realização de agrupamentos de dados. As consultas com requisitos de ordenação são um dos processos mais importantes para distinguir este tipo de sistemas de BD em sistemas de DW. Frequentemente as empresas, para analisar os seus negócios, necessitam de extrair informação das suas fontes de informação numa certa ordem, crescente ou decrescente, segundo este ou aquele parâmetro, por exemplo, quando precisam de extrair informação sobre as vendas registadas numa determinada altura ou quais os clientes que fizeram mais encomendas num determinado período. Por último, e como as *queries* típicas de um sistema de DW apresentam normalmente muitas agregações, como sendo o caso de consultas que solicitam uma média de vendas por produto, analisou-se as funções relacionadas com o agrupamento de dados, com vista à obtenção de valores estatísticos sobre os valores formados. De seguida, foi feita uma análise às melhorias de desempenho obtidas e a possíveis contributos para o melhoramento do desempenho do sistema na execução de cada uma das *queries* implementadas.

No final de todo o processo, foi possível retirar algumas conclusões acerca dos padrões dos sistemas de BD orientados por colunas e como estes se comportam quando acolhem um sistema de DW. Mais concretamente, foi possível verificar uma grande diferença nos tempos de execução entre os sistemas, quando se tratava da leitura de muitos atributos sobre uma mesma tabela. Sempre que se pretendia aceder a um grande número de atributos por tabela, ou mesmo aceder a todo o seu esquema, o MonetDB registou piores tempos de execução nas consultas realizadas. Isto deve-se ao facto de, num sistema de armazenamento de dados convencional, a informação ser armazenada toda junta, enquantom que num sistema orientado por colunas os diversos atributos estão dispersos por diferentes estruturas. Desta forma, registou-se um aumento do tempo de execução na reconstrução dos tuplos das tabelas envolvidas nas consultas no sistema de BD orientado por colunas.

No entanto, a recuperação dos dados, aquando de uma seleção, contidos numa única coluna em sistemas de BD orientados por colunas é quase sempre mais rápida, do que em sistemas tradicionais.

Neste tipo de armazenamento, apenas os atributos necessários para a realização da query são acedidos, lidos e processados. Contudo, em sistemas de BD orientados por linhas, sempre que é realizada uma consulta a apenas um campo nenhuma coluna é ignorada pois toda a linha é carregada em memória antes de selecionar o campo que é pretendido. Por esse motivo e como *queries* típicas de um ambiente de *data warehousing*, apresentam valores de I/O menores, *queries* onde se verificou consultas a um menor número de colunas registou melhores resultados de performance na execução das consultas.

Apesar disso, consultas que acediam paralelamente a muitas colunas de distintas tabelas em alguns casos registaram piores resultados nos sistemas de BD orientados por colunas e noutros registaram melhores, no entanto com uma margem de diferença muito pequena. Isto está relacionado com o facto de que a procura nas diversas partes do disco entre cada bloco de leitura, por vezes, pode implicar que várias colunas sejam lidas em paralelo, o que aumenta o tempo de procura no disco e consequentemente piora o desempenho da consulta. A conversão a que o modelo de dados do AdventureWorkDW2008R2 foi submetido foi total, tendo-se registado piores tempos do que espectável nos processos de inserção dos dados, principalmente em tabelas cuja carga de registos, nomeadamente na tabela de factos "FactInternetSales" era superior (número de registos superior às restantes tabelas do modelo de dados).

O objectivo deste estudo era descobrir qual o melhor tipo de sistema de BD a aplicar num sistema de DW e em que situações se poderia fazê-lo. Supostamente os resultados obtidos poderiam ter sido melhores, caso as consultas acedessem a um menor número de atributos com uma grande carga de registos.

6.2 Trabalho Futuro

Apesar dos resultados obtidos nos diversos testes realizados com as consultas que se apresentou ao longo do capítulo 5 serem bastante interessantes, pensa-se que a realização de um estudo mais aprofundado permitiria retirar outras ilações sobre os dois sistemas avaliados. Por exemplo, relativamente aos contributos para melhoramento do sistema, poderiam ser aplicados índices sobre as estruturas de dados utilizadas ou até mesmo proceder-se à desnormalização do modelo dimensional implementado no MonetDB. Desta forma, no caso dos dados se encontrarem desnormalizados, seria

possível testar o desempenho de todas as consultas sem ser necessário recorrer a "inúmeros" processos de junção entre as tabelas que integram o DW. Recorrendo-se a estruturas de índices poder-se-ia testar o desempenho das consultas usando uma ou mais colunas de uma tabela da BD, fornecendo-se, assim, a base para um acesso mais eficiente aos registos pretendidos.

Outra perspetiva interessante seria a criação, tal como mencionado anteriormente, de índices de armazenamento por colunas no sistema de BD tradicional e ver até que ponto uma implementação híbrida não faria mais sentido para determinados sistemas de DW e, consequentemente, para determinados ambientes de negócio. Para finalizar, as consultas foram, apenas, aplicadas a uma área de trabalho da BD selecionada: as vendas efectuadas pela Internet. Como tal, seria interessante no futuro aplicar a mesma análise às restantes áreas da AdventureWorksDW2008R2, alargando o âmbito do trabalho. As propostas de estudos apresentados são apenas algumas das possíveis na "imensidão" de possíveis estudos existentes com o conjunto de dados facultado, que deverão depender essencialmente das necessidades dos utilizadores e das consultas exigidas pelo negócio.

Bibliografia

[Abadi et al., 2006] Daniel J. Abadi, Samuel R.Madden e Miguel C. Ferreira: "Integrating Compression and Execution in Column-Oriented Database Systems". SIGMOD 2006 (Chicago, Illinois), USA. Junho 2006.

[Abadi, 2007] Daniel J. Abadi: "Column-Stores for Wide and Sparse Data". 3rd Biennal Conference on Innovative Data Systems Research (CIDR), (Asilomar, Califórnia), USA. Janeiro, 2007.

[Abadi et al., 2007] Daniel J. Abadi, Daniel S. Myers, David J. DeWitt e Samuel R. Madden: "Materialization Strategies in a Column-Oriented DBMS". ICDE, (Istambul), Turquia. 2007.

[Abadi et al., 2008] Daniel J. Abadi, Samuel R. Madden e Nabil Hachem: "Column-Stores vs. Row-Stores: How Different Are They Really?". SIGMOD'08, (Vancouver, BC), Canada. Junho, 2008.

[Abadi, 2008] Daniel J. Abadi: "Query Execution in Column-Oriented Database Systems". Massachusetts Institute of Technology. Fevereiro, 2008.

[Abadi et al., 2009] Daniel J. Abadi, Peter A. Boncz e Stavros Harizopoulos: "Column-oriented Database Systems". in Proc. VLDB'09 (Lyon), France. Agosto, 2009.

[Ailamaki et al., 2001] Anastassia Ailamaki, David J. DeWitt, Mark D. Hill, Marios Skounakis: "Weaving Relations for Cache Performance". 27th VLDB Conference, Roma, Italy. 2001.

[Amer-Yahia & Cluet, 2004] Amer-Yahia, S. e Cluet, S.: "A declarative approach to optimize bulk loading into databases." ACM Trans. Database Syst., vol.29, pp.233-281. 2004.

[Bathia & Patil, 2011] Anuradha Bathia e Shefali Patil: "Column Oriented DBMS an Approach". International Journal of Computer Science & Communication Network, vol1(2). Outubro-Novembro, 2011.

[Boncz et al., 1999] Peter A. Boncz, Martin L. Kersten: "Mil primitives for querying a fragmented world". Amesterdam, Netherlands. 1999.

[Boncz et al., 2005] Peter A. Boncz, Marcin Zukowski e Niels Nes: "MonetDB/X100: Hyper-Pipelining Query Execution". CIDR Conference, Amesterdam, Netherlands. 2005.

[Connolly & Begg, 1998] Thomas M. Connolly e Carolyn Begg: "Database Systems: A Practical Approach to Design, Implementation, and Management, 2nd Ed". Addison-Wesley Longman Publishing Co., Inc, 1998.

[Copeland & Khoshafian, 1985] George P. Copeland e Setrag N. Khoshafian: "A decomposition storage model" . Microelectronica and Techonoly Computer Corporation. Austin, Texas. 1985.

[DeWitt et al., 2002] David J. DeWitt, Ravishankar Ramamurthy e Qi Su: "A Case for Fractured Mirrors". 28th VLDB Conference, Hong Kong, China. 2002.

[Golfarelli & Rizzi, 2009] Matteo Golfarelli e Stefano Rizzi:"Data Warehouse Design: Modern Principles and Methodologies". The McGraw-Hill Companies, S.r.l.-Publishing Group Itália.

[Halverson et al., 2006] Alan Halverson, Jennifer L. Beckmann, Jeffrey F. Naughton e David J. DeWitt: "A Comparison of C-Store and Row-Store in Common Framework". 32nd VLDB Conference, (Seoul), Korea. 2006.

[Hanson, 2008] Eric N. Hanson: "Columnstore for fast Data Warehouse Query". Microsoft Corporation. Novembro, 2010.

[Harizopoulos et al., 2006] Stavros Harizopoulos, Velen Liang, Daniel J. Abadi e Samuel Madden: "Performance Tradeoffs in Read-Optimized Databases". VLDB'06, (Seoul), Korea. Setembro, 2006.

[Hérman et al., 2010] Sandor Hérman, Marcin Zukowski, Niels Nes, Lefteris Sidirourgos, Peter Boncz: "Positional Update Handling in Column Stores". SIGMOD'10, (Indianápolis, Indiana), USA. Junho, 2010.

[Idreos et al., 2009] Stratos Idreos, Martin L. Kersten, Stefan Manegold: "Self-organizing Tuple Reconstuction in Column-Stores". SIGMOD' 09, Providence, Rhode Island, USA. June 29- July 2, 2009.

[Immon el al., 1998] Inmon, W., Rudin, K., Buss, C. e Sousa, R.: "Data Warehouse Performance". John Wiley & Sons, Inc. New York, EUA, 1998.

[Jonsoon, 2009] Johan Jonsson: "Codbase – A Column-Oriented In Memory Database". Umea University, Department of Computing Science. Suécia. Fevereiro, 2009.

[Kimball & Caserta, 2004a] Kimball, R. e Caserta, J. "The Data Warehouse ETL Toolkit: Practical Techniques for Extracting and Cleaning". John Wiley & Sons, Inc. New York, EUA, 2004.

[Kimball & Caserta, 2004b] Kimball, R. e Caserta, J. "The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming and Delivering Data". John Wiley & Sons, Indianapolis, 2004.

[Kimball et al.,2008] Ralph Kimball, Margy Ross, Warren Thornthwaite, Joy Mundy e Bob Becker: "The Data Warehouse ETL Toolkit:Practical Techniques for Bulding Data Warehouse and Business Intelligence Systems". 2nd Edition. Wiley Publishing, Inc. Indianapolis, Indiana, 2008.

[Loshin, 2010] David Loshin: "Gaining the Performance Edge Using a Column- Oriented Database Management System". Sybase, 2010.

[Martins, 2006] Victor Manuel Moreira Martins: "Integração de Sistemas de Informação – Perspectivas, Normas e Abordagens". Edição Silabo, Lda., 1ª Edição. Lisboa, 2006.

[Matei,2010] Gheorghe Matei: "Column-Oriented Databases, an Alternative for Analytical Environment". Romania Commercial Bank, Bucharest, Romania. Database System Journal vol.I, no 2/2010.

[McKnight, 2002] William McKnight: "Choosing a DBMS for Data Warehousing". McKnight Associates, Inc. 2002.

[Michael, 2002] Michael, Maged M. "High performance dynamic lock-free hash tables and list-based sets." Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures. ACM, 2002.

[Parsian, 2005] Parsian M.: "JDBC Recipes: A Problem-Solution Approach". Apress, USA, 2005.

[Pipino et al., 2002] Pipino, L., Lee Y. e Wang, R. : "Data Quality Assessment". Communications of ACM, vol.45, no4, pp.211-218. Abril, 2002.

[Ramamurthy et al., 2002] Ravishankar Ramamurthy, David J. DeWitt e Qi Su: "A case for Fractured Mirrors". 28th VLDB Conference, Hong Kong, China. 2002.

[Rascão, 2001] José Rascão: "Sistemas de Informação para as Organizações – A Informação chave para a tomada de decisão". Edições Sílabo, Lda., 1ª Edição. Lisboa, Janeiro, 2001.

[Redman, 2004] Redman, T.: "Data: An Unfolding Quality Disaster". DM Review Magazine. Agosto, 2004.

[Russom, 2009] Philip Russom: "Next Generation Data Warehouse Plataforms". 1105 Media, Inc. 2009.

[Shirazi, 2003] Shirazi, J.: Java™ Performance Tuning, 2nd Edition. O'Reilly & Associates, Inc., California, 2003.

[Sorjonen, 2012] Sami Sorjonen: "OLAP Query performance in Column Orientde Databases (December 2012)". Seminar: Column Databases, December 2012.

[Stonebraker et al., 2005] Mike Stonebraker, Daniel J. Abadi, Adam Batkin, Xuedong Chen, Mitch Cherniack, Miguel Ferreira, Edmond Lau, Amerson Lin, Sam Madden, Elizabeth O'Neil, Pat O'Neil, Alex Rasin, Nga Tran e Stan Zdonik: "C-Store: A Column-oriented DBMS". 31th VLDB Conference, (Trondheim), Norway. 2005.

[Stonebraker et al., 2007] Stonebraker, M., Bear, C., Çetintemel, U., Cherniack, M., Ge, T., Hachem, N., Harizopoulos, S., Lifter, J., Rogers, J. e Zdonik, S.: "One Sizes fit all? – Part 2: Benchmarking Results". 3rd Biennal Conference on Innovative Data Systems Research (CIDR), Asilomar, California, USA, 2007.

[Venkat & Rakesh, 2007] Venkat K. e Rakesh K.: "Column Oriented Databases vs Row Oriented Databases. Special Interest Activity". 2007.

Referências WWW

[SenSage, 2010] [online] Available at: <http://www.addamark.com/> [Accessed 10]

[BMMsoft, 2009] [online] Available at: <http://www.bmmsoft.com/>

[ComputerWorld, 2007] Relational database Pioneer says technology is obsolete. [online] Available at: <http://www.computerworld.com/s/article/print/9034619/Relational_database_pioneer_says_technology_is_obsolete>

[InfiniDB, 2010] Scalable. Fast. Simple. [online] Available at: <http://www.infinidb.org/>

[Infobright, 2010] Infobright. [online] Available at: <http://www.infobright.com/>

[LucidDB, 2010] LucidDB. [online] Available at: <http://www.luciddb.org/>

[MonetDB, 2008] The column-store pioneer. [online] Available at: <http://monetdb.cwi.nl/>

[Sybase Inc, 2010] Available at: <http://www.sybase.com/products/datawarehousing/sybaseiq>

[Oracle, 2009] SPARC Enterprise M9000 Server. [online] Available at: <http://www.oracle.com/us/products/servers-storage/servers/sparc-enterprise/m-series/m9000/overview/index.html>

[Abadi, 2010] Distinguishing Two Major Types of Column-Stores. [online] Available at: <http://dbmsmusings.blogspot.pt/2010/03/distinguishing-two-major-types-of_29.html>

[Microsoft, 2012] Bulk Insert (Transact - SQL). [online] Available at: <http://msdn.microsoft.com/en-us/library/ms188365.aspx>

[InformationWeek, 2013] Reinvent the Data Warehouse with Column-Store databases and Appliances. [online] Available at: <http://www.informationweek.com/software/information-management/reinvent-the-data-warehouse-with-column-/207801688>

[TPC, 2001] TPC Transaction Processing Performance Councill. [online] Available at: <http://www.tpc.org/tpch/>

[Edge, 2012] Why Column Based Data Warehouses are Rising Higher and Higher. [online] Available at: <http://edge.sybase.com/TomMorris2?elqPURLPage=357>

[FindTheBest, 2012] Find the best: unbiased, Database Management Systems. [online] Available at: <http://database-management-systems.findthebest.com/compare/21-28/LucidDB-vs-MonetDB>

[Microsoft, 2013] Data Warehouse de exemplo Adventure Works. [online] Available at: <http://msdn.microsoft.com/pt-br/library/ms124623(v=sql.105).aspx>

[Vertica, 2012] Vertica. [online] Available at: <http://www.vertica.com/>

[4Information, 2008] SGBD - Sistema Gerenciador de Banco de Dados. [online] Available at: <http://4information.wordpress.com/2008/08/23/sgbd-sistema-gerenciador-de-banco-de-dados/>