Universidade do Minho
Escola de Engenharia

Jorge Filipe Fernandes Abreu

A influência de índices bitmap no desempenho de sistemas de Data Warehousing

Novembro de 2009

A influência de índices bitmap no desempenho de sistemas de Data Warehousing
Jorge Filipe Fernandes Abreu
UMinho | 2009

Universidade do Minho
Escola de Engenharia

Jorge Filipe Fernandes Abreu

A influência de índices bitmap no desempenho de sistemas de Data Warehousing

Tese de Mestrado em Engenharia de Informática

Trabalho efectuado sob a orientação do
Professor Doutor Orlando Manuel de Oliveira Belo

Novembro de 2009

Aos pais e irmãos

Agradecimentos

Ao meu pai e mãe que sempre me apoiaram e incentivaram para a realização de mais esta etapa de formação académica.

Aos meus irmãos, pela compreensão e apoio demonstrados durante a realização desta dissertação, assim como de toda a minha formação académica.

Ao Professor Doutor Orlando Belo, pelo constante e sempre pronto apoio e disponibilidade demonstrada em toda a orientação desta dissertação.

Por fim, aos colegas de licenciatura/mestrado com os quais passei os meus 5 anos de formação académica, por todo o apoio, ajuda e estimulo prestados ao longo de todo o meu percurso académico.

Resumo

A influência de índices bitmap no desempenho de sistemas de Data Warehousing

Nos tempos que correm, são cada vez mais as organizações empresariais que se deparam com grandes volumes de dados, todos eles considerados vitais para o seu negócio. Usualmente, estas organizações têm uma forte necessidade em termos de sistemas de suporte à decisão e, consequentemente, de sistemas de *Data Warehousing*, para suportar de forma eficiente e organizada toda esta informação, para que a sua utilização e análise seja feita de forma mais eficaz. Para tal, é necessário garantir uma forte disponibilidade de toda a informação. Um dos factores mais relevantes a ter em consideração, senão o factor mais crucial desta eficiência, são as estruturas de indexação utilizadas sobre os próprios sistemas de *Data Warehousing*. Na literatura de *Data Warehousing*, os índices *B-Tree* são os mais utilizados, sendo aconselhada a sua utilização sobre atributos de elevada cardinalidade. Os mesmos são vistos e utilizados por vários Sistemas de Gestão de Bases de Dados como estruturas de indexação por omissão. Nos últimos anos, tem-se verificado um crescente interesse e estudo de uma outra estrutura de indexação para sistemas de *Data Warehousing*: os índices do tipo *bitmap*. A eficiência deste tipo de índices é reconhecida na literatura actual para o caso particular de atributos de baixa cardinalidade. Contudo, existem já vários mecanismos propostos (*encode*, *binning* e *compression*) que facultam a possibilidade de se utilizar este tipo de índices sobre atributos de elevada cardinalidade. Actualmente, os Sistemas de Gestão de Bases de Dados líderes de mercado têm já alguma implementação de índices do tipo *bitmap*. Neste sentido, nesta dissertação foi efectuada uma revisão do estado actual de índices *bitmap*, bem como o estado de implementação existente nos Sistemas de Gestão de Bases de Dados actuais. Numa fase mais avançada, essencialmente de demonstração prática, é apresentado que, actualmente, é já possível o uso de índices do tipo *bitmap* recorrendo às implementações dos

próprios motores de bases de dados. Nesse sentido foi analisada a influência dos índices *bitmap* neste tipo de sistemas, através de um conjunto de *queries* típicas de um sistema de processamento analítico, com particular atenção na maximização do desempenho e minimização do espaço dos índices do tipo *bitmap*, tendo como base de comparação os tão reputados índices *B-Tree*.

Abstract

The influence of bitmap indexes on Data Warehousing Systems performance

Nowadays, more and more business organizations are faced with large volumes of data, which are considered vital for their business. Usually, these organizations have a strong need for decision support systems, and consequently for Data Warehousing Systems, in order to efficiently and neatly support all data information so that their use and analysis can be carried out more effectively. For this, it is necessary to ensure a high availability of information. One of the main factors to consider, if not the most crucial factor of this efficiency, is the indexing structures used on such systems. In Data Warehousing literature, the most used indexes are the B-Tree indexes, and their use is recommended for high cardinality attributes. They are seen and used by different database management systems as default indexing structures. In recent years, there has been a growing interest and study in another indexing structure for Data Warehousing systems: bitmap indexes. The efficiency of this kind of indexes is well known and recognized in the current literature, and in particular for the case of low cardinality attributes. However, various mechanisms were already proposed (encode, binning and compression) granting the possibility of using this kind of indexes on high cardinality attributes. Nowadays, the market leaders on Database Management Systems already have some kind of implementation of bitmap indexes. In this way, in this dissertation was carried out a review of the current state of bitmap indexes, as well as the current implementation status on Database Management Systems. In a later stage, essentially of practical demonstration, it is showed that it is already possible to use implementations of bitmap indexes present on Database Management systems. To this end, the influence of bitmap indexes in Data Warehousing Systems is analyzed through a set of typical queries to an analytical processing

system, with particular emphasis on maximizing performance and minimizing the space of the bitmap indices, having the well-known and reputed B-Tree indexes as a base of comparison.

Índice

Índice de Figuras

Índice de Tabelas

Siglas e acrónimos

1RQ	-	One-side Range Queries
2RQ	-	Two-side Range Queries
BBC	-	Byte-Aligned Bitmap Code
BD	-	Base de Dados
DM	-	Data Mining
DML	-	Data Manipulation Language
DW	-	Data Warehouse
DWeb		Data Webhouse
EQ	-	Equality Queries
ETL	-	Extract, Transform and Load
MPP	-	Massive Parallel Processing
OLAP	-	On-Line Analytical Processing
OLTP	-	On-line transaction processing
RLE	-	Run-Length Encode
RQ	-	Range Queries
SDW	-	Sistemas de Data Warehousing
SGBD	-	Sistemas de Gestão de Base de Dados
SGBDR	-	Sistemas de Gestão de Bases de Dados Relacionais
SO	-	Sistemas Operacionais
SSD	-	Sistemas de Suporte à Decisão
WAH	-	Word-Aligned Hybrid code

Capítulo 1

Introdução

1.1 Desempenho em Sistemas de Data Warehousing

Nos nossos dias, a competitividade empresarial deixou de ser algo especulativo, passando a ser um factor de grande peso a considerar nas estratégias empresariais. As empresas têm uma necessidade crescente em terem toda a sua informação, considerada vital ao seu negócio, organizada de uma forma estruturada, de forma a permitir a sua rápida utilização. Este tipo de informação, que tipicamente se encontra em sistemas transaccionais (OLTP - *On-line transaction processing*), encontra-se na maioria dos casos optimizada para operações de escrita. Considere-se, por exemplo, o caso de uma cadeia de hipermercados onde são efectuadas milhões de transacções por dia. Nesta situação, cada produto comprado por um cliente corresponde a uma transacção (um registo) numa *Base de Dados* (BD) operacional, o que gera uma grande carga transaccional sobre si. Maioritariamente, os sistemas OLTP não se encontram preparados para dar suporte a um sistema de suporte à decisão (SSD) típico [Chan & Ioannidis, 1998b; French, 1995] devido aos seus requisitos e cargas operacionais específicas. Desta forma, as empresas sentem cada vez mais a necessidade de adoptarem sistemas de suporte à decisão que lhes permitam fazer processamento analítico (OLAP – *On-Line Analytical Processing*) ou previsões através de métodos de mineração de dados - *Data Mining* (DM) de forma a melhorarem o seu negócio. Para tal, toda a informação considerada vital é armazenada em *Sistemas de Data Warehousing* (SDW), estando organizada de forma a que a sua exploração seja possível de acordo com as principais linhas de suporte à decisão empresariais. Neste tipo de sistemas, a informação é mantida num sistema de

dados específico com características um pouco especiais, sendo normalmente designado por *Data Warehouse* (DW). Este é um repositório de dados não volátil, que usualmente integra informação proveniente de várias fontes de dados e está organizado segundo as várias áreas de negócio da sua instituição de acolhimento [Connolly & Begg, 2005]. Este pode ser elaborado recorrendo a metodologias[1] bem conhecidas e identificadas de forma a fazer com que o DW fique devidamente estruturado para corresponder eficientemente às necessidades da instituição em causa. É, normalmente, povoado com apelo a técnicas[2] de extracção, transformação e carregamento (ETL) de dados, tipicamente provenientes dos diferentes *Sistemas Operacionais* (SO) disponíveis na instituição (figura 1).

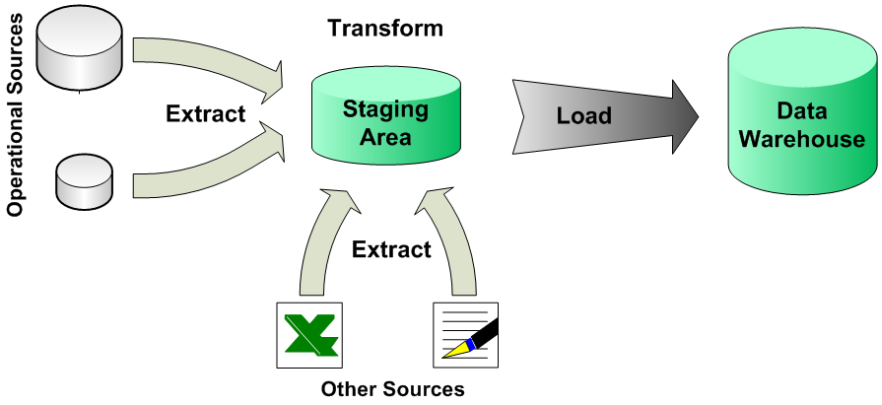


Figura 1 - Representação do processo de ETL

Os sistemas de ETL são, assim, os responsáveis pela recolha da informação proveniente das fontes de dados, pelo seu tratamento e seu posterior carregamento nos DW, executando de forma parcialmente automática esta tantas vezes complicada passagem da informação dos sistemas operacionais (tipicamente organizados segundo um modelo relacional) para os SDW (organizados segundo um modelo dimensional) [Kimball & Caserta, 2004].

O modelo de concepção de um DW, usualmente denominado por modelo dimensional, segue uma abordagem diferente do tradicional modelo relacional e das formas convencionais de projecto dos sistemas operacionais. Normalmente, o modelo dimensional de um DW (e dos seus diversos módulos de dados) desenvolve-se em torno de um esquema em estrela (figura 2(a)) ou de um esquema em floco de neve (figura 2(b)) - *star schema* e *snowflake schema*, respectivamente -,

[1] Metodologia de Kimball. [Kimball et al., 2008]
[2] Técnicas para optimização de ETL. [Kimball & Caserta., 2004]

sendo caracterizado pela existência de uma tabela central, denominada por tabela de factos, com a ligação a N tabelas auxiliares, denominadas por dimensões. A tabela de factos, na maioria dos casos, corresponde tipicamente a transacções realizadas no âmbito de análise da tabela de factos e que foram efectuadas no passado (mesmo que recente) nos SO alvo, e constituem as tabelas de maior cardinalidade localizadas no DW. Nelas também se encontram as principais medidas de suporte às decisões empresariais.

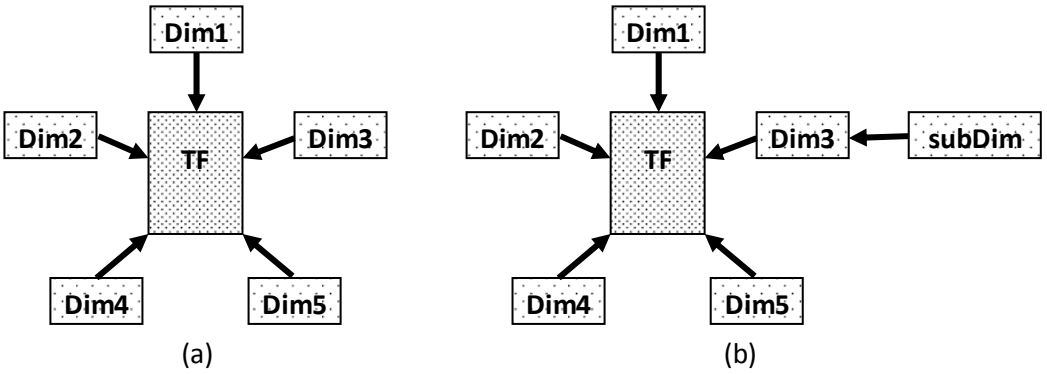


Figura 2 - Representação de esquema em estrela (a) e floco de neve (b)

A elevada cardinalidade das tabelas de factos provoca muitas vezes graves problemas de desempenho nos SDW. A deterioração no desempenho dos SDW, mais propriamente sobre as tabelas de factos, deve-se ao grande número de operações de escrita que ocorre aquando da realização do sistema de ETL e às massivas operações de leitura realizadas pelos utilizadores. Contudo, são as operações de leitura as mais usadas em sistemas de SDW. Na gíria dos SDW são designadas usualmente por *ad hoc*[3] *queries*. Estas *queries*, multidimensionais por natureza, provocam enormes dificuldades ao sistema na sua execução e satisfação de forma eficiente. Este tipo de estrangulamentos provoca variadíssimos problemas, como o descontentamento por parte dos utilizadores, consultas lentas, perdas de capital, diminuição da janela de oportunidade para ETL, entre outras.

Em muitos casos, o SDW é crucial para a vitalidade do próprio negócio, uma vez que é uma das suas fontes de dados vital para o suporte das actividades quotidianas. Um dos factores de maior influência no desempenho dos SDW são sem dúvida as estruturas de indexação utilizadas, sendo

[3] *Ad hoc Queries* – *Queries* realizadas espontaneamente pelos utilizadores [Kimball et al., 2008].

os índices criados sobre cada tabela os que mais contribuem para atenuarem este tipo de estrangulamento.

Como já referido, uma das razões típicas que leva à construção de um DW é a criação de uma plataforma de suporte para sistemas OLAP. Com é sabido, estes sistemas possuem mecanismos típicos de forma a minimizar o tempo de satisfação de uma *query*, tendo toda a informação organizada sobre a forma de um hipercubo, que permite a realização de processos de navegação segundo os vários critérios de agregação e eixos de análise disponíveis no DW. Estes têm normalmente uma parte da informação materializada fisicamente, ou seja, as suas *queries* mais frequentes estão materializadas em disco (materialização de cubos). Desta forma, as *queries* mais frequentes, conhecidas vulgarmente como *Top K queries*, podem ser processadas em tempo próximo do real, uma vez que os dados já se encontram previamente processados. Contudo, na realidade, para a grande maioria das empresas é incomportável materializar todas as perspectivas de um cubo de dados, ou seja, nem todas as *queries* podem ser determinadas a partir dos cubos já materializados. Assim, sempre que uma *query* não pode ser determinada através dos dados pré-calculados, a mesma tem de processada com o recurso às estruturas de indexação existentes nas bases de dados disponíveis, neste caso o DW.

Uma das formas mais directas e simplistas de avaliar uma *query*, passa por fazer um varrimento sequencial de todas as entradas de uma tabela e verificar se cumpre ou não a condição de procura. Tome-se como exemplo uma *query* típica em suporte à decisão: "determinar a quantidade de produtos adquiridos por um cliente C num determinado período de tempo P". Facilmente se deduz que no contexto de um DW a abordagem de pesquisa anteriormente referida não é de todo aconselhada. Este tipo de *query* pode ser normalmente acelerada com a utilização de índices, tipicamente *B-Trees* (ou suas variantes) [Stockinger & Wu, 2006; Comer, 1979; Gaede & Günther, 1998]. Para se poder obter um desempenho máximo deste tipo de índices em *queries* multi-dimensionais, devem ser criadas várias combinações de índices entre atributos de forma a cobrir todo o tipo de *queries* que pudessem eventualmente vir a ser efectuadas. No entanto, estas estruturas de indexação requerem grande espaço em disco, podendo em alguns casos ocupar mais espaço que os próprios dados. Eficiência-espaço é um dos dilemas existentes nas estruturas de indexação com que os investigadores se tem debatido e tentado atenuar ao longo do tempo, sendo conhecido na literatura como "*curse of dimensionality*" [Berchtold et al., 1998; Hinneburg & Keim, 1999].

Na literatura de DW vários são os autores que defendem o uso de índices *bitmap* como sendo uma potencial estrutura de indexação para resolver muitos dos problemas sistemáticos no processamento de *queries* em DW, sendo até apontado por alguns [Stockinger & Wu, 2006] como uma grande promessa para a atenuação da dita "maldição". A eficiência dos *bitmaps* como estrutura de indexação sob atributos de baixa cardinalidade é amplamente reconhecida pelos vários investigadores de *Data Warehousing*. No entanto, existe alguma resistência ao uso deste tipo de índices, sendo vistos muitas vezes, em casos de atributos de elevada cardinalidade, como estruturas mais volumosas que as tradicionais. No entanto, existem vários estudos e publicações que mostram diversas abordagens e técnicas para ultrapassar este tipo de problema, assim como resultados experimentais do seu desempenho em condições muito diversificadas.

1.2 Motivação e Objectivos do Trabalho

Queries complexas, grandes volumes de dados e elevadas frequências de leitura são características que têm grande peso no processamento de *queries* em *Data Warehousing*. Estes são factores que fazem com que as tradicionais técnicas e optimizações realizadas no processamento *queries* sobre os sistemas OLTP não se adeqúem aos ambientes de *Data Warehousing* [Wu & Buchmann, 1998, Chaudhuri & Dayal, 1997, Chaudhuri et al., 2001]. Um bom equilíbrio e optimização das estruturas de indexação são aspectos cruciais de optimização em *Data Warehousing*. Este é um problema com que muitos dos administradores de BD, neste caso de DW, se debatem constantemente. Com a utilização de índices do tipo *bitmap* poderá se obter um equilíbrio de tempo-espaço mais eficaz, proporcionando optimizações bastantes significativas. Desta forma, a presente dissertação tem por objectivo a investigação do impacto dos índices *bitmap* sobre as operações chave de um SDW, mais propriamente, sobre as tão abundantes operações de leitura que ocorrem neste tipo de ambientes. Inicialmente pretende-se realizar o levantamento da informação existente sobre índices *bitmap* ("*state of the art*"), de forma a identificar os vários mecanismos e abordagens de optimização já propostos, mais propriamente nas áreas de *encoding*, *compressing* e *binning* [Stockinger & Wu, 2006].

A literatura de *Data Warehousing* actual conta já com um vasto conjunto de publicações que mostram a análise de desempenho espaço-tempo dos índices *bitmap*. No entanto, os testes e conclusões a que os autores chegam são específicos para um determinado tipo de estratégia de

optimização, sem que se saiba muito bem ao certo o tipo de operações a que as *queries* de teste foram sujeitas. Desta forma, a vertente prática deste trabalho consiste no estudo do impacto e viabilidade das implementações existentes de índices *bitmap* num dos Sistemas de Gestão de Base de Dados (SGBD) líder de mercado. Para tal, um conjunto de *queries* típicas de SSD previamente definidas como base de teste são testadas sobre um *Data Webhouse* (DWeb)[4]. Estas são compostas por um misto de operações típicas (cláusulas de *where* (*equality, range, interval, membership*), *group by, order by, joins*, etc), muitas delas bem conhecidas pelo seu penoso processo de execução. Desta forma, poderá ser realizada uma análise mais concreta da eficiência dos índices *bitmap* para as diferentes operações das *queries*.

No final desta dissertação pretende-se ter adquirido um conjunto de competências nesta área, que permitam de uma forma mais rápida e fiável saber, nomeadamente:

- identificar potenciais tabelas alvo nas quais se possa aplicar este tipo de índices;
- conhecer o tipo de impacto destes índices sobre o desempenho de um SDW;
- conhecer os tipos de atributos normalmente candidatos à aplicação deste tipo de índices;
- identificar o tipo de operações das *queries* para as quais os índices *bitmap* se mostram mais eficientes que as estruturas tradicionalmente usadas.

1.3 Organização da dissertação

Tal como se teve a oportunidade de verificar, neste primeiro capítulo foi, basicamente, feita uma introdução sobre a problemática da introdução de índices do tipo *bitmap* no seio de um SDW, apresentada a motivação base do trabalho e delineados os objectivos de estudo para esta dissertação. No segundo capítulo serão abordados alguns métodos e estratégias de indexação existentes relacionadas com área de estudo em questão. Segue-se o terceiro capítulo, no qual é feita uma análise detalhada e extensiva do "*background*" existente sobre índices *bitmap*, dividida segundo três estratégias distintas que podem ser actualmente encontradas na literatura: *enconding*, *binning* e *compressing*. No fim deste capítulo apresenta-se uma breve análise acerca da implementação dos índices *bitmap* em SGBD actuais. O quarto capítulo corresponde à fase prática desta dissertação. Nele é efectuada uma análise da influência dos índices *bitmap* na

[4] Desenvolvido no departamento de informática da Universidade do Minho para análise de *click stream* de uma plataforma Web da mesma [Marques e Guimarães, 2009].

satisfação de *queries*, bem como a sua avaliação segundo as medidas de análise tempo e espaço utilizado aquando da sua criação. No quinto capítulo, é abordada uma variante de índices *bitmap*, designados de *bitmap join*, onde é feita uma análise similar à do capítulo anterior. A dissertação é concluída no sexto capítulo, no qual são apresentadas as conclusões finais e discutidos algumas linhas de orientação e de trabalho futuro.

Capítulo 2

Indexação de dados

Os índices são um dos componentes chave num sistema de bases de dados. O mesmo acontece num sistema de *Data Warehousing*. Kimball et al. (2008) defendem a elaboração de um plano de indexação efectivo a ser utilizado num DW. No entanto, tal plano deve-se ir adaptando às novas necessidades dos utilizadores do DW que vão surgindo dia após dia. O ideal seria a criação de uma estrutura de indexação capaz de satisfazer todas as necessidades dos utilizadores de um DW. Porém, a complexidade e a quantidade de *queries ad hoc* a que tipicamente um DW está sujeito, faz com que este necessite de estruturas de indexação bastante eficientes de forma a proporcionar um máximo desempenho e satisfação entre os utilizadores, uma vez que a realização de *queries* de uma forma eficiente é cada vez mais uma tarefa complexa de concretizar nas realidades empresariais actuais.

Aquando da elaboração de um plano de indexação para um DW, é importante ter uma noção real dos seus requisitos e saber como é que as *queries* são executadas num determinado SGBD. Tipicamente, todos os SGBD presentes no mercado têm uma implementação de todos os tipos de índices mais conceituados, sendo utilizados para aumentar a performance do processamento de *queries* em SDW. Todavia, deve-se ter em conta que a maioria dos SGBD foram desenvolvidos para dar suporte a sistemas OLTP, o que faz com que os seus requisitos operacionais não sejam propriamente os mesmos de um DW. Vários autores [Graefe, 1993; Chaudhuri & Dayal, 1997; Chan & Ioannidis, 1998b] acreditam que os problemas de optimização em sistemas OLTP já foram extensivamente estudados. Porém, nos últimos anos várias publicações, acompanhadas também pelos mais conceituados SGBD no mercado, têm apresentado estudos e soluções mais adequadas

às funcionalidades usualmente requeridas pelos sistemas de suporte à decisão, com o objectivo de optimizar a eficiência da relação tão complexa tempo-espaço das estruturas de indexação.

Um índice pode ser visto de uma forma análoga ao índice de um livro. Quando desejamos procurar alguma informação num livro, normalmente recorremos ao seu índice para que, de uma forma rápida e simples, possamos encontrar as páginas que abordam o assunto da nossa palavra-chave. Sem a ajuda deste índice, a nossa procura seria um processo demorado, penoso e ineficiente. Os índices de uma base de dados têm o mesmo propósito, isto é, permitirem a localização rápida e eficaz de um ou mais registos correspondentes a uma palavra-chave. Um índice é uma estrutura de dados "que permite aos SGDB localizar registos num ficheiro de uma forma mais rápida, aumentando desta forma a rapidez de resposta das *queries*" [Connolly & Begg, 2005], e está "associada a uma tabela que está logicamente ordenada pelos valores da chave" [Kimball et al., 2008]. Por sua vez, uma estrutura de indexação consiste num conjunto de registos compostos pelo atributo chave e o endereço da localização do registo na BD, correspondente ao atributo chave. De uma forma simplista, um índice, basicamente, contem um apontador para uma linha de uma tabela que contem um valor chave. Este valor chave é utilizado para obter a localização (*rowid*) numa tabela.

Na literatura actual existem vários tipos de índices, de vários tipos. Os principais são, sem dúvida alguma, os índices primários e os índices secundários [Connolly & Begg, 2005]:

- Primários. Um índice primário é um índice que se encontra associado ao atributo, ou conjunto de atributos (no caso de uma chave composta), que constitui a chave primária, na qual está garantida a unicidade de valores (integridade de entidade). Neste caso, o índice e o próprio ficheiro de dados (tabela) encontram-se sequencialmente ordenados pelo atributo chave indexado.
- Secundários. Um índice secundário é também uma estrutura ordenada segundo o atributo indexado, tal como acontece no índice primário. Todavia, o ficheiro de dados (tabela) não está necessariamente ordenado segundo o atributo indexado. Estes índices têm como principal objectivo melhorar o desempenho de *queries* que utilizam outros atributos que não os primários.

Em contrapartida, esta abordagem sobre a forma de índices faz com que as operações de inserção e de actualização de dados sejam mais penosas de realizar, uma vez que os índices também têm de ser actualizados. Desta forma, é necessário que exista um balanceamento adequado à eficiência das *queries* e à quantidade de índices criados para o melhoramento dessa mesma eficiência. De salientar, que um índice secundário difere do índice primário na obrigatoriedade de apenas conter valores únicos no seu atributo indexado, ou seja, o atributo indexado num índice secundário pode ser composto por valores repetidos. Uma estrutura de indexação secundária pode lidar com estes valores repetidos através de várias abordagens, tais como a criação de:

- um índice denso que faça o mapeamento de todos os valores chave existentes no atributo indexado, permitindo assim a repetição de valores na estrutura de indexação;
- um índice com um registo para cada valor chave distinto do atributo indexado e permitindo uma lista de apontadores para cada um dos valores chave repetidos na tabela;
- um índice com um registo para cada valor chave distinto do atributo indexado, como no caso anterior; todavia, este registo apenas terá um apontador para um "*bucket*" de apontadores correspondentes a cada um dos valores chave repetidos na tabela.

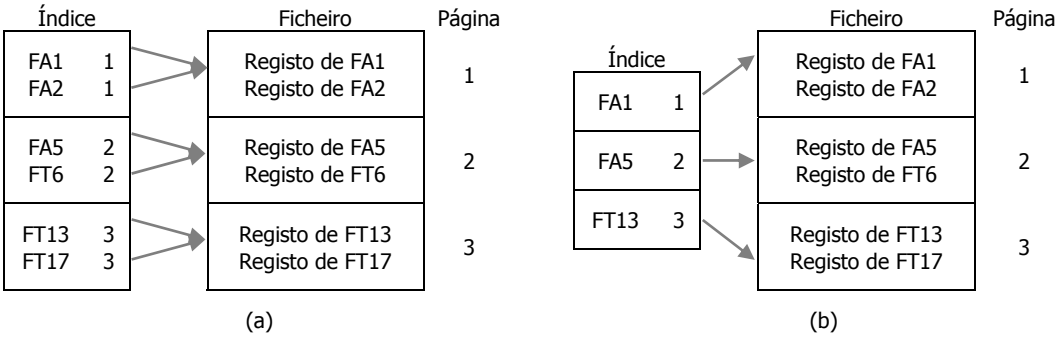


Figura 3 - Exemplo de um índice denso (a) e de um índice disperso (b)

Um índice pode ainda ser classificado como denso (do inglês *dense*) ou disperso (do inglês *sparse*). Um índice primário é um exemplo de um índice denso, em que o índice contém um registo para cada um dos valores do atributo indexado. Um índice disperso é um índice que apenas contém registo para alguns dos valores do atributo indexado. A figura 3 (baseada em [Connolly & Begg, 2005]) mostra um exemplo de um índice denso e de um índice disperso, respectivamente em (a) e (b).

De referir também que uma tabela pode ser indexada apenas por um índice principal, no entanto, pode ser também indexada por vários índices secundários.

O factor espaço é algo também a ter em consideração aquando da criação de uma estrutura de indexação, uma vez que alguns tipos de estruturas se podem tornar demasiado volumosas e incomportáveis de armazenar. Actualmente os SGBD disponibilizam um conjunto de estruturas de indexação que permite um melhor ajuste às necessidades de cada realidade, ou seja, pode-se escolher os tipos de estruturas que melhor se adaptam ao nosso problema.

2.1 B-Trees

As *B-Trees* são o tipo de índices mais populares e mais reconhecidos nos SGBD. A sua forma mais clássica foi a primeira implementação de índices dos SGBD [Kimball et al., 2008], sendo reconhecida a sua eficiência de indexação para atributos de alta cardinalidade. Estes são os índices criados por omissão, aquando da criação de uma chave primária, pela maioria dos SGBD. Actualmente existem uma grande variedade de índices *B-Tree*, como por exemplo *B+-Tree* [Lin, 2008], *R-Tree* [Arge et al., 2004], *R*-Tree* [Beckmann & Seeger, 2009], *UB-Tree* [Comer, 1979; Fenk et al., 2000]. As *B-Trees* e suas variantes são das estruturas de indexação mais utilizadas pela maioria dos SGBD devido à sua eficiência em sistemas OLTP, tipicamente caracterizados pelo maior equilíbrio no número de leituras e escritas nas bases de dados [Comer, 1979; Wu & Buchmann 1998; Stockinger & Wu, 2006]. Um índice do tipo *B-Tree* constrói uma árvore com todos os valores possíveis com uma lista de apontadores (*rowids*) que correspondem ao valor da folha, permitindo desta forma a localização dos registos correspondente numa tabela [Kimball et al., 2008; Price, 2008]. A figura 4, mostra a organização de uma *B-Tree*.

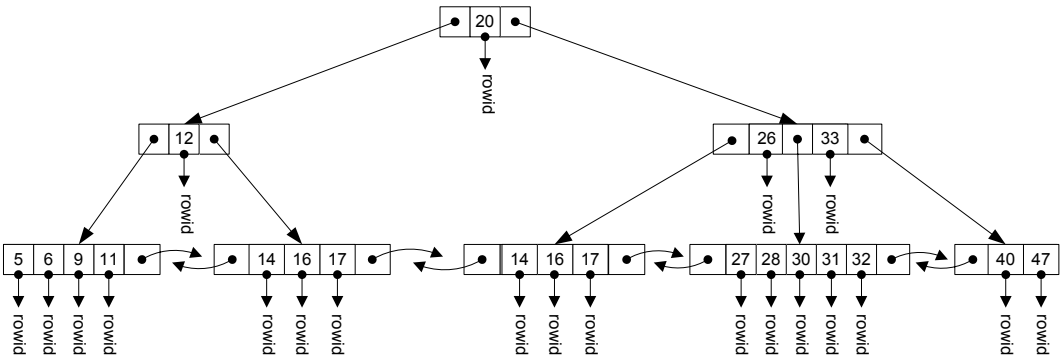


Figura 4 – Exemplo da organização de uma *B-Tree*.

Uma estrutura de indexação com *B-Trees* apenas é eficiente quando utilizada na sua totalidade, ou seja, estas obtêm um máximo desempenho quando todos os atributos indexados são utilizados na mesma *query*, o que na realidade nem sempre acontece. No entanto, como já foi referido antes, os *Data Warehouses* são caracterizados por serem sujeitos aos mais variados tipos de *queries ad hoc*, o que dificulta o planeamento das suas estruturas de indexação. Assim, se fossem criados vários índices compostos numa tabela de n atributos, de forma a cobrir todas as possíveis combinações de *queries*, seriam necessárias $C_1^n + C_2^n + C_3^n + \cdots + C_n^n = 2^n - 1$ *B-Trees*. Neste caso, seria praticamente inaceitável o custo de manutenção de todas estas estruturas de indexação. Por outro lado, as *B-Trees* demonstram uma boa estabilidade e performance em operações de actualização, processo que acontece sempre que são inseridos novos dados nos atributos indexados. Contudo, esta não é uma característica muito relevante no contexto de *data warehousing*, uma vez que tipicamente as actualizações são efectuadas num determinado período de tempo (vulgarmente designado por janela de oportunidade) em grandes conjuntos de dados. Nestas operações de carregamento de dados, muitos autores aconselham a eliminar previamente todas as estruturas de indexação das tabelas alvo e posteriormente proceder à sua reconstrução [Kimball et al., 2008], o que em muitos casos contribui numa optimização de tempo de todo o processo muito importante. Assim, em SDW as *B-Trees* podem não ser a melhor escolha como estrutura de indexação. Por exemplo, segundo Wu & Buchmann (1998) um *bitmap* simples tem vantagens sobre as *B-Trees*, uma vez que normalmente a sua construção e manutenção requer menos tempo para atributos que de baixa actualização/inserção.

2.2 Clustered Index

Um índice cluster é uma das possibilidades de indexação mais vulgar disponibilizada pelos actuais SGBD. Este tipo de índices apenas pode ser criado uma vez por cada tabela. Tipicamente, na grande maioria dos SGBD estes são criados automaticamente aquando da criação de uma chave primária [Connolly & Begg, 2005]. Num índice do tipo cluster os registos estão ordenados e guardados segundo as palavras-chave que o compõe. Por outras palavras, a definição de um índice do tipo cluster faz com que uma tabela seja armazenada em disco pela ordem correspondente à constituição do índice, dos atributos que o compõem. Segundo Kimball et al. (2008), aquando da realização de uma *query* este tipo de índices têm uma grande probabilidade de serem escolhidos pelos optimizadores de *queries*.

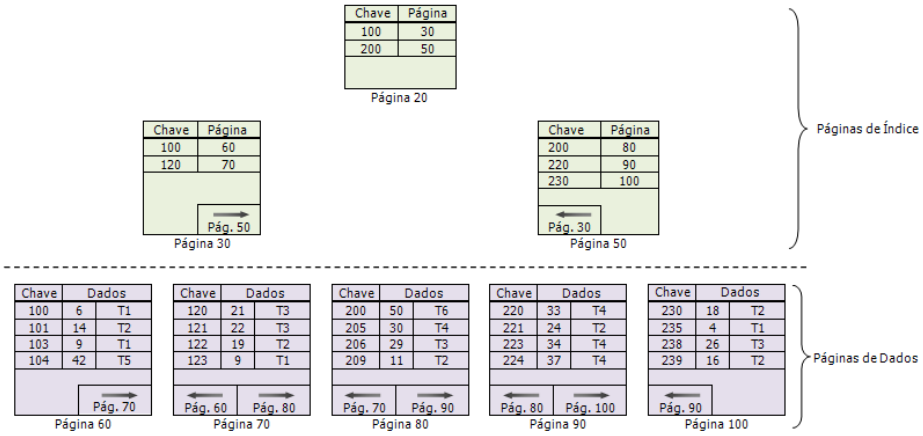


Figura 5 - Representação de um índice Cluster.

2.3 Projection Index

Um índice de projecção é uma das estruturas de indexação mais simples. A Sybase IQ foi o primeiro produto comercial a utilizar este tipo de indexação, denominado de *Fast Projection Index* [French, 1995; O'Neil & Quass, 1997]. Basicamente, este consiste na materialização de todos os valores de um atributo indexados segundo a ordem do seu *rowid*. Por exemplo, considerando um atributo Y de uma tabela, um índice de projecção sobre o mesmo atributo consiste numa sequência de valores do atributo Y, segundo a ordem do número ordinal do registo. O tipo de dimensão de um atributo é um factor a considerar neste tipo de indexação. Segundo O'Neil & Quass (1997), a criação de um índice de projecção sobre um atributo de tamanho variável, torna o índice mais lento. Como forma de resolver este problema pode-se optar por utilizar, como tamanho fixo, o tamanho do valor que tem maior comprimento. Considere-se por exemplo o atributo B (figura 6(a)) com um tamanho fixo de 8 *bytes* e um tamanho de página de 4KB. Neste caso consegue-se armazenar 500 valores Através de um número ordinal "n" de um registo é possível determinar a localização do mesmo em disco, através do número da página "p" e do *slot* "s" (figura 6(b)), onde $p = n/500$ e $s = n\%500$ [O'Neil & Quass, 1997]. O'Neil & Quass (1997) mostram ainda que em operações de SQL como a soma, a média ou a variância, os índices de projecção demonstram um bom ou mesmo "óptimo" desempenho.

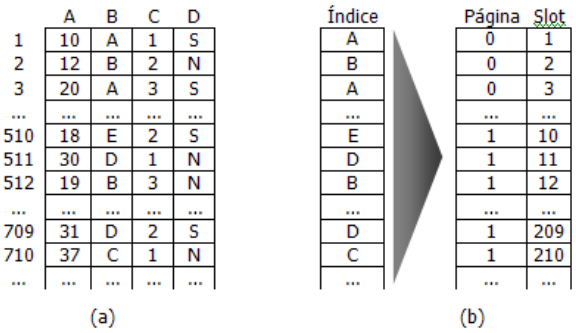


Figura 6 – Representação de um índice de projecção (b),
para um atributo de uma tabela (a).

Capítulo 3

Índices bitmap

Como referido anteriormente, as *B-Trees* e suas variantes são dos tipos de índices mais populares nas actuais implementações de sistemas de bases de dados, sendo que a maioria, senão mesmo todos, têm implementado uma das suas variantes. No entanto, em *Data Warehousing* as necessidades são um pouco diferentes. Em aplicações típicas deste tipo de ambientes, como por exemplo OLAP, as leituras de dados sobre o DW são as operações realizadas com maior frequência [Chaudhuri & Dayal, 1997; Chaudhuri et al., 2001]. Num DW típico, com uma quantidade considerável de dimensões, de elevadas cardinalidades, e, com uma grande variedade de *queries ad hoc*, leva-nos muitas vezes à seguinte questão: "Que atributos é que se devem indexar?".

Segundo Stockinger & Wu (2006), muitas das variantes das *B-Trees*, por muitos defendidas como sendo as mais eficientes, até mesmo a nível académico, não são as mais indicadas neste tipo de condições. Os mesmos consideram que este tipo de estruturas apenas é eficaz para um número modesto de dimensões. No entanto, a sua maior ineficiência prende-se com o facto das mesmas apenas obterem um bom desempenho no caso de todos os atributos indexados serem utilizados em cada *query*, o que na realidade muitas das vezes não acontece. Já os índices *bitmap* demonstram uma grande eficiência conjunta, contribuindo para a redução do espaço de procura, minimizando assim número de acessos a dados em disco.

O'neil & Quass (1997) afirmam que para *ad hoc range*[5] *queries* a maioria dos índices convencionais existentes não consegue obter melhor desempenho que o índice de projecção. Mais tarde, Stockinger & Wu (2006) defenderam que com técnicas de compressão, os índices *bitmap* conseguem obter melhores tempos que os índices de projecção, que muitas das vezes são considerados os melhores métodos para *queries* multi-dimensionais.

Ao longo da sua existência os índices *bitmap* tiveram diferentes aplicações e funcionalidades. Uma estrutura idêntica à de hoje conhecida foi proposta por Wong et al. (1985) como sendo bastante eficaz para ficheiros de dados (*Bit Transposed Files*). *Bitmap index* foi o nome pelo qual O'neil e seus colaboradores divulgaram esta estrutura de indexação que teve a sua primeira implementação comercial no *Model 204*[6] da *Computer Corporation of America*, sendo no entanto referida por muitos investigadores como sendo uma variante dos índices *B-Tree*. [O'neil, 1987; O'neil & Quass, 1997]. Segundo Knuth (1998), chegaram até a ser vistos como uma forma de ficheiros invertidos.

A ideia básica de um índice *bitmap* consiste num conjunto de *bitmaps* (b_0, b_1, …, b_c), correspondentes a cada valor distinto de um atributo com N bits ('0' ou '1') cada, indicando se o atributo em causa é igual ('1') ou não ('0') a um determinado valor. Um atributo de cardinalidade C e um total de valores N (linhas) dará origem a um índice *bitmap* com C *bitmaps* de N *bits* cada. Este tipo de estrutura permite uma eficaz manipulação lógica dos *bitmaps* (*bit-wise*), operações (*AND, OR, XOR, NOT*) que são muito bem suportadas pelo *hardware*.

Tomando a figura 7 (baseada em [Stockinger & Wu, 2006]) como exemplo, temos uma representação de um índice *bitmap* na sua forma mais simples, no qual se pode observar a existência de um *bitmap* para cada valor distinto do atributo. Neste caso, C e N tomam o valor de 7 e 10, respectivamente. Observando, por exemplo, o *bitmap* b_3, verifica-se a existência de apenas dois valores '1', representando os dois valores iguais a 3 existentes no atributo indexado. Uma observação análoga pode ser feita segundo a horizontal. Tomando o valor realçado, temos na estrutura de indexação a representação do atributo de valor igual a 3, onde apenas o *bitmap* b_3 está assinalado a '1' e os restantes *bits* a '0'.

[5] *Queries* com condições por exemplo do tipo "A<20" ou "A>20" realizadas espontaneamente pelos utilizadores.
[6] http://cc.ysu.edu/ComputerServices/Newsletter/link0102/model204.htm
http://sirius-software.com/m204.html

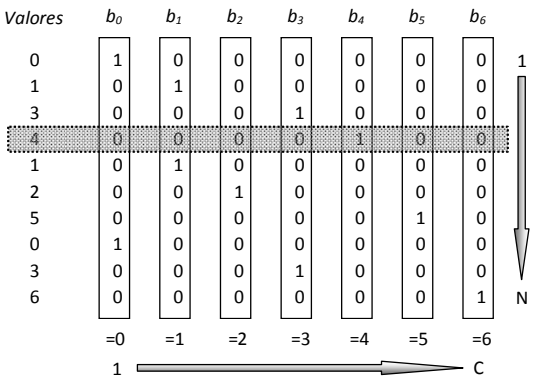


Figura 7- Representação de um índice *bitmap* simples, com 7 *bitmaps* representativos de 7 valores distintos de um atributo.

No caso de maioritariamente se ter operações de leitura e atributos de baixas cardinalidades, os índices *bitmap* são um dos métodos de indexação mais eficientes, possibilitando uma maior rapidez em *queries* multi-dimensionais (O'neil, 1987; Rotem et al., 2005a; Wu et al., 2006a). No entanto para atributos de alta cardinalidade, estes podem-se tornar impraticáveis devido à dimensão que irão ter, gerando algumas questões bastante séries em termos da relação espaço-tempo. Chan & Ioannidis (1998b), analisaram a contrapartida espaço-tempo em índices *bitmap*, onde identificaram analítica e experimentalmente quatro pontos de relevância representados graficamente na figura 8 (extraída de [Chan & Ioannidis, 1998b]):

- (A) Espaço óptimo ocupado por um índice *bitmap*.
- (B) Tempo óptimo de um índice *bitmap* para uma determinada restrição de espaço M.
- (C) Índice *bitmap* com o melhor balanceamento "espaço-tempo".
- (D) Tempo óptimo de um índice *bitmap*.

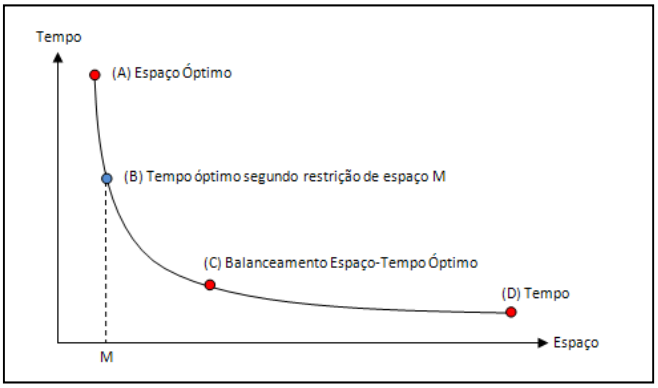


Figura 8 – Contrapartida espaço-tempo.

É com o intuito de ultrapassar este problema, que vários autores ao longo dos últimos anos têm vindo a desenvolver e a propor várias estratégias para maximizar a eficiência e minimizar o espaço dos índices *bitmap* em *Data Warehousing*, independentemente das cardinalidades dos atributos alvo. Actualmente existem já várias técnicas e estratégias apresentadas na literatura de índices *bitmap* para *Data Warehousing*, que basicamente estão divididas em três categorias: *encoding*, *binning* e *compressing*.

3.1 Encoding

Actualmente existem vários métodos de codificação de índices *bitmap* com vista a diminuir, ainda que pouco em algumas abordagens, o espaço ocupado pelas estruturas de indexação, bem como para aumentar a eficiência de determinado tipo de *queries*. Em SSD, condições de *querying* como, por exemplo, "temperatura = 20" (*equality queries*), "temperatura > 20" (*one-side range queries*) ou "15 < temperatura < 30" (*two-side range queries*), são talvez as mais frequentes. Logo, os métodos de *enconding* de *bitmaps* para além de ter como objectivo a minimização de espaço têm também de ser adequados ao tipo de condições como as anteriores.
Refira-se as seguintes abreviaturas a serem usadas EQ, 1RQ, 2RQ, RQ como, respectivamente, *equality queries*, *one-side range queries*, *two-side range queries* e *range queries*, respectivamente [Chan & Ioannidis, 1999].

3.1.1 Equality Encode

A representação básica de um índice *bitmap* (figura 7) é muitas vezes também designada por *equality encoding*. Esta forma de *enconde* é a mais indicada e eficiente para *queries* com condições do tipo "produto = xpto", indicando se o atributo em causa é igual ('1') ou não ('0') a um determinado valor ou condição. Dependendo do tipo de cardinalidade e do tipo de condições a que o atributo está sujeito, esta pode não ser a melhor abordagem, uma vez que o método *equality encoding* apenas satisfaz eficientemente *queries* de igualdade.

Vejamos, agora, o caso de uma *query* com uma condição sobre um atributo "A" do tipo "$v_1 \leq A \leq v_2$". O procedimento de resolução desta *query* em *equality encoding* é dado pela equação (1) [Chan & Ioannidis, 1999]:

$$"v_1 \leq A \leq v_2" = \begin{cases} \bigvee_{i=v_1}^{v_2} E^i, & se\ v_2 - v_1 + 1 < \left\lfloor \frac{C}{2} \right\rfloor \\ \bigvee_{i=0}^{v_1-1} E^i \vee \bigvee_{i=v_2+1}^{C-1} E^i, & caso\ contrário \end{cases} \quad (1)$$

Segundo a equação (1), este método mostra-se ineficiente para *queries* do tipo 1RQ ("temperatura < 30" ou "temperatura >30") e 2RQ ("20 < temperatura < 30"), uma vez que exige a leitura de vários *bitmaps*, no pior dos casos C/2 *bitmaps*.

3.1.2 Range Encode

Queries do tipo 1RQ e 2RQ podem ser efectuadas de uma forma mais eficiente, recorrendo respectivamente às estratégias de *range encoding* e *interval encoding*, desenvolvidas por Chan & Ioannidis (1998b; 1999). Na figura 9 (extraída de [Stockinger & Wu, 2006]) é demonstrada uma comparação entre os índices *equality-encoded* e *range-encoded*, na qual se pode verificar os ganhos de um método sobre o outro. Aparentemente, os ganhos existentes entre estas duas estratégias apenas deverão ser relevantes ao nível da eficiência, uma vez que o espaço requerido por ambos é muito similar, diferindo apenas num *bitmap*. Como o *bitmap* (R^9) apenas é constituído por *bits* '1', este não é guardado, sendo assim apenas necessários C-1 *bitmaps* em *range encoding* minimizando, ainda que pouco, o espaço em relação a *equality encoding*. No entanto, em *Data Warehousing* é frequente ter atributos com elevada repetição de valores, o que pode tornar a simples diferença de 1 *bitmap* armazenado entre os dois métodos significativa.

	$\pi_A(R)$	E^9	E^8	E^7	E^6	E^5	E^4	E^3	E^2	E^1	E^0	R^8	R^7	R^6	R^5	R^4	R^3	R^2	R^1	R^0
1	3	0	0	0	0	0	0	1	0	0	0	1	1	1	1	1	1	0	0	0
2	2	0	0	0	0	0	0	0	1	0	0	1	1	1	1	1	1	0	0	0
3	1	0	0	0	0	0	0	0	0	1	0	1	1	1	1	1	1	1	1	0
4	2	0	0	0	0	0	0	0	1	0	0	1	1	1	1	1	1	1	0	0
5	8	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
6	3	0	0	0	0	0	0	1	0	0	0	1	1	1	1	1	1	0	0	0
7	9	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
9	7	0	0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0
10	5	0	0	0	0	1	0	0	0	0	0	1	1	1	1	0	0	0	0	0
11	6	0	0	0	1	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0
12	4	0	0	0	0	0	1	0	0	0	0	1	1	1	1	1	0	0	0	0
	(a)	(b)										(c)								

Figura 9 - Índice *bitmap* com C=10. (a) Projecção do atributo indexado (duplicados mantidos). (b) Índice *equality-encoded*. (c) Índice *range-encoded*.

Tomando o valor 2, destacado na figura 9, vemos que em *equality encoding* (figura 9(b)) apenas o *bitmap* E^2 é assinalado com um *bit* a '1' e os restantes a '0'. Já em *range encoding* (figura 9(c)) todos os *bitmaps* de valores superiores a 2 também são marcados horizontalmente com '1' para o valores 2, isto é, todos os *bitmaps* de R^2 a R^8 são marcados com um *bit* a '1' e os restantes a '0'. Este tipo de *encode* demonstra ser mais apropriado para RQ. Considere-se a *query* com a condição "A≤5". Em *range encoding* apenas o *bitmap* R^5 necessita ser acedido, uma vez que todos os *bits* assinalados a '1' satisfazem a condição de pesquisa. No caso de *equality encoding*, todos os *bitmaps* entre E^6 e E^9 teriam de ser disjuntos ("*ORed"* através do operador lógico *OR*) e posteriormente negado o resultado, ou seja, teriam de ser efectuadas um conjunto de operações e acessos a vários *bitmaps*, correspondentes à expressão "$\overline{E^6 \vee E^7 \vee E^8 \vee E^9}$". Neste caso, com *equality encoding* seria necessário aceder a 4 *bitmaps* contra apenas 1 em *range encoding*, sendo que com *equality encoding* posteriormente ainda teriam de ser efectuadas 3 operações de disjunção (*OR*) e finalmente a negação do resultado. Em suma, para RQ através de *equality encoding* serão necessários aceder no pior caso c/2 *bitmaps* contra apenas 1 em *range encoding*, em que c é o número de *bitmaps*.

Através dos testes realizados por Chan & Ioannidis, (1998b) conclui-se que para a maioria dos casos através de *range encoding* obtém-se um melhor *tradeoff* espaço-tempo em relação a *equality encoding*. No entanto, como se pode verificar através da equação (2) [Chan & Ioannidis, 1999], para condições do tipo "$v_1 \leq A \leq v_2$", nos casos de v_1e v_2 serem diferentes de 0 ou C-1, com *range encoding* é necessário sempre ler 2 *bitmaps* de forma a satisfazer a pesquisa:

$$"v_1 \leq A \leq v_2" = \begin{cases} R^0, & se\ v_1 = v_2 = 0 \\ R^{v_1} \oplus R^{v_1-1}, & se\ 0 < v_1 = v_2 < C-1 \\ \overline{R^{C-2}}, & se\ v_1 = v_2 = C-1 \\ \overline{R^{v_1-1}}, & se\ 0 < v_1 < C-1, v_2 = C-1 \\ R^{v_2}, & se\ v_1 = 0, 0 < v_2 = C-1 \\ R^{v_2} \oplus R^{v_1-1}, & caso\ contrário \end{cases} \quad (2)$$

Estes dois esquemas, já abordados anteriormente, encontram-se identificados e bem conhecidos há bastante tempo. No entanto, a questão se existe ou não um esquema de *encoding* que tenha uma melhor performance espaço-tempo do que *equality encoding* e *range encoding* permaneceu na literatura. Chan & Ioannidis, (1999) fizeram uma análise de espaço-tempo óptimo sobre os

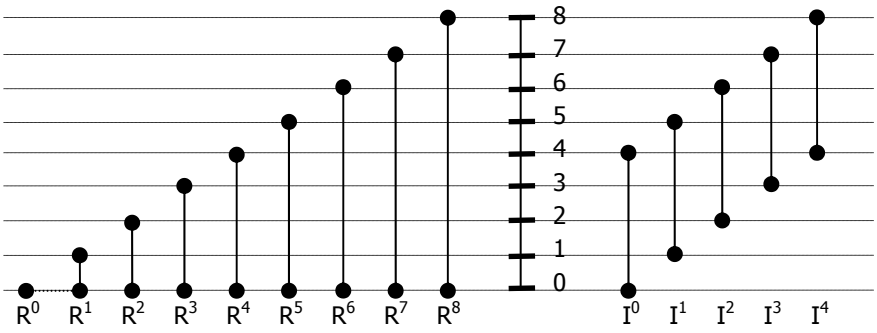
esquemas *equality encoding* e *range encoding*. Esta teve como base a quantidade de *bitmaps* armazenados para analisar a eficiência de espaço e a quantidade de *bitmaps* acedidos para analisar a eficiência de tempo pelos respectivos esquemas de *encode*. O teorema 1[7] apresentado pelos mesmos sugere o tipo de *encode* apropriado para os respectivos tipos de *queries*.

Teorema 1:

1. *Range encoding* é óptimo para EQ sse C≤5.
2. *Range encoding* é óptimo para 1RQ para todo o C.
3. *Range encoding* não é óptimo para 2RQ para qualquer C.
4. *Range encoding* é óptimo para RQ para todo C.
5. *Equality encoding* é óptimo para EQ para todo C.
6. *Equality encoding* não é óptimo para 1RQ, 2RQ, e RQ para qualquer C.

3.1.3 Interval Encode

Através do teorema anteriormente apresentado, verifica-se que nenhum dos métodos de *encode* até aqui abordados é apropriado para *queries* do tipo 2RQ ("20 < temperatura < 30"). Com *range encoding*, uma *query* do tipo anterior é processada através da operação $R^{v_2} \oplus R^{v_1-1}$ da equação (2). Através de um método similar e sempre com o objectivo de minimizar o espaço utilizado, Chan & Ioannidis, (1999) desenvolveram um novo esquema de *encode*, denominado de *interval encoding*.



(a) Range Encoding (b) Interval Encoding

Figura 10 - *Range* vs *Interval enconding*, com C=10.

[7] A prova do teorema 1 pode ser encontrada em [Chan e Ioannidis, 1998a] no seu apêndice C.

Este consiste num conjunto de $\left\lceil \frac{C}{2} \right\rceil$ *bitmaps* $I = \{I^0, I^1, I^2.., I^{\left\lceil \frac{C}{2} \right\rceil}\}$, no qual cada $I^j = R^{j+m} \oplus R^j = [j, j+m]$, com $m = \left\lfloor \frac{C}{2} \right\rfloor - 1$. Para o caso de C ser ímpar[8] temos $m = \left\lfloor \frac{C}{2} \right\rfloor - 1$ ou $m = \left\lfloor \frac{C}{2} \right\rfloor$ dependendo do tipo maioritário de *queries* a que o atributo em causa está sujeito, ou seja, 2RQ ou 1RQ respectivamente.

Na figura 10 (baseada em [Chan & Ioannidis, 1999]) pode-se ver o conjunto de valores que cada *bitmap* consegue alcançar para os respectivos esquemas e a quantidade de *bitmaps* que os mesmos necessitam. Na figura 11 (extraída de [Chan & Ioannidis, 1999]) está representado o esquema de *interval encoding* para os valores da figura 9(a). Esta codificação é obtida através do processo exemplificado abaixo para o *bitmap* e equação destacados na figura 11(c) e figura 11(a), respectivamente.

R^6	1	1	1	1	0	1	0	1	0	1	1	1
R^1	0	0	1	0	0	0	0	1	0	0	0	0
$I^2 = R^6 \oplus R^1$	1	1	0	1	0	1	0	0	0	1	1	1

$\oplus$	0	1
0	0	1
1	1	0

$I = \{I^0, I^1, I^2, I^3, I^4\}, onde$
$I^0 = R^4 = [0,4]$
$I^1 = R^5 \oplus R^0 = [1,5]$
$I^2 = R^6 \oplus R^1 = [2,6]$
$I^3 = R^7 \oplus R^2 = [3,7]$
$I^4 = R^8 \oplus R^3 = [4,8]$

(a)

	$\pi_A(R)$
1	3
2	2
3	1
4	2
5	8
6	2
7	9
8	0
9	7
10	5
11	6
12	4

(b)

I^4	I^3	I^2	I^1	I^0
0	1	1	1	1
0	0	1	1	1
0	0	0	1	1
0	0	1	1	1
1	0	0	0	0
0	0	1	1	1
0	0	0	0	0
0	0	0	0	1
1	1	0	0	0
1	1	1	1	0
1	1	1	0	0
1	1	1	1	1

(c)

Figura 11 - Índice *bitmap* com C=10. (a) Definição de índice *interval encoded*. (b) Projecção do atributo indexado (duplicados mantidos). (c) Índice *interval-encoded*.

[8] [Chan e Ioannidis, 1998a] apêndice B.

Ao nível da rentabilização do espaço, uma estrutura de indexação com *interval encoding* consegue uma optimização para metade do espaço comparativamente ao *range encoding*. Quanto à eficiência de tempo (tempo de leitura dos *bitmaps* envolvidos na satisfação de uma *query*) é similar ao esquema de *range encoding* uma vez que no máximo são necessários 2 *bitmaps* para satisfazer qualquer tipo de condição. Este método de *encode* mantém na mesma a capacidade de resposta aos vários tipos de *queries*, EQ, 1RQ e 2RQ, sendo respectivamente processadas através do procedimento dado pelas equações (3), (4) e (5) [Chan & Ioannidis, 1999].

$$\text{"A}=v\text{"} = \begin{cases} I^0, & se\ v = 0, m = 0 \\ \overline{I^0}, & se\ v = 1, C = 2 \\ I^1, & se\ v = 1, C = 3 \\ I^v \wedge \overline{I^{v+1}}, & se\ v < m \\ I^v \wedge I^0, & se\ v = m, m > 0 \\ I^{v-m} \wedge \overline{I^{v-m-1}}, & se\ m < v < C-1, m > 0 \\ \overline{\left(I^{\lceil \frac{C}{2} \rceil - 1} \vee I^0\right)}, & se\ v = C-1 \end{cases} \tag{3}$$

$$Para\ 0 < v < C-1,$$

$$\text{"A}\leq v\text{"} = \begin{cases} I^0 \wedge \overline{I^{v+1}}, & se\ v < m \\ I^0, & se\ v = m \\ I^0 \vee I^{v-m}, & se\ m < v < C-1 \end{cases} \tag{4}$$

$$Para\ 0 < v_1 < v_2 < C-1,$$

$$\text{"}v_1\text{≤A≤}v_2\text{"} = \begin{cases} I^{v_1} \wedge \overline{I^{v_2+1}}, & se\ v_2 < m \\ I^{v_1} \wedge I^0, & se\ v_2 = m \\ I^{v_1} \wedge I^{v_2-m}, & se\ v_2 < v_1 + m, v_1 < n \\ I^{v_1}, & se\ v_2 = v_1 + m, v_1 < n \\ I^{v_1} \vee I^{v_2-m}, & se\ v_2 > v_1 + m, v_1 < m \\ I^{v_1} \vee I^{v_1+1}, & se\ v_2 = v_1 + m + 1, v_1 = m \\ I^{v_2-m} \wedge \overline{I^{v_1-m-1}}, & se\ v_1 \geq n \end{cases} \tag{5}$$

Dado o novo método de *encode*, Chan & Ioannidis, (1999) apresentam um segundo teorema[9] que identifica o tipo de *queries* para o qual este método é óptimo.

Teorema 2:

1. *Interval-encoding* não é óptimo para EQ se C ≥ 14.
2. *Interval-encoding* é óptimo para 1RQ para todo C.
3. *Interval-encoding* é óptimo para 2RQ para todo C.
4. *Interval-encoding* é óptimo para RQ para todo C.

Após a exposição das três estratégias básicas de *encode* existentes na literatura dos índices *bitmap*, vamos fazer agora um sumário de forma a analisar os seus prós e contras. A tabela 1 (extraída de [Chan & Ioannidis, 1999]) apresenta de uma forma sucinta toda a informação relativa aos teoremas 1 e 2, indicando qual o melhor esquema de *encode* para cada classe de *queries*, ou seja, o melhor par (Q,S), onde Q corresponde a uma classe de *queries* (EQ, 1RQ, 2RQ, RQ) e S ao tipo de *encode*.

Tipo de Query	Esquema de Encoding		
	Equality	Range	Interval
EQ	√	√ sse $C \leq 5$	× se $C \geq 14$
1RQ	×	√	√
2EQ	×	×	√
RQ	×	√	√

Tabela 1 - Optimalidade dos 3 esquemas de *encode* base.

Denote-se que "√" para um par (Q,S) significa que o esquema de *encode* S é óptimo para *queries* da classe Q e que "×" para um par (Q,S) significa que o esquema de *encode* S não é óptimo para *queries* da classe Q. Note-se que para o par ("EQ","Interval") e C<14 existe um caso omitido, o que significa que, para C<14 não se sabe se *interval encoding* é óptimo ou não, para *queries* da classe EQ. Contudo, a tabela mostra que o método de *interval encoding* ostenta ser óptimo para a grande maioria das *queries*.

[9] A prova do teorema 2 pode ser encontrada em [Chan e Ioannidis, 1998a] apêndice C

3.1.4 Comparação das três formas básicas de encode

Pela perspectiva de eficiência de espaço (*space-efficient*), o *interval-encode* é dos três esquemas básicos que tem melhor eficiência de espaço, uma vez que apenas necessita de metade dos *bitmaps* utilizados pelos outros dois esquemas. Para EQ é, sem dúvida, o esquema de *equality enconding* que obtém a melhor eficiência (*query-efficient*), uma vez que acede apenas a um *bitmap* para satisfazer qualquer condição de igualdade. Por outro lado, este é o esquema que obtém piores resultados em RQ. Através de *range encoding* consegue-se obter a melhor eficiência em *queries* do tipo 1RQ, acedendo apenas a um *bitmap* contra os dois normalmente necessitados pelo esquema de *interval encoding*. Este mostra também uma igual eficiência para EQ e 2RQ em relação ao *interval encoding*, sendo acedidos maioritariamente dois *bitmaps*.

No entanto, os DW também estão sujeitos a operações de escrita e actualização, o que obriga a actualizações das estruturas de indexação, ou seja, neste caso actualizações de cada *bitmap* dos esquemas de *encode*. Contudo, em *Data Warehousing* estas operações são tipicamente efectuadas em grandes conjuntos de dados (*batches*), associadas à dita "janela de oportunidade" do processo de ETL. Assim, a eficiência da actualização das estruturas de indexação, neste caso os esquemas de *encode,* está também afectada ao método *batch* associado e à forma que a informação se encontra armazenada em disco. Assumindo que todas as actualizações seriam efectuadas uma a uma, o custo de uma actualização nestas estruturas é dado pela quantidade de *bitmaps* que terão de ser actualizados, ou seja, os *bitmaps* onde têm de ser acrescentados *bits* '1'.
Como mostra a tabela 2, no melhor caso dos três esquemas de *encode* apenas 1 *bitmap* necessita de ser actualizado. O método *equality encoding* tem a melhor eficiência em actualizações, sendo apenas afectado 1 *bitmap* para todos os casos. Com a pior eficiência encontra-se o método de *range encoding,* necessitando de actualizar $\frac{C}{4}$ e $\left\lfloor \frac{C}{2} \right\rfloor$ *bitmaps* para os casos previstos e piores casos, respectivamente.

Esquema de Encoding	Situações		
	Melhor	Previsto	Pior
Equal	1	1	1
Range	1	$\frac{C-1}{2}$	C-1
Interval	1	$\frac{C}{4}$	$\left\lfloor \frac{C}{2} \right\rfloor$

Tabela 2 – Eficiência de actualização dos três esquemas de *encode* base.

3.1.5 Binary Encode

Como referido no inicio deste capítulo, a eficiência dos índices *bitmap* em atributos de baixa cardinalidade é reconhecida na literatura de *Data Warehousing*. Contudo, o tamanho dos índices *bitmap* tem de ser controlados de forma a manter a viabilidade do seu uso em *Data Warehousing*. Note-se que, apesar do método de *interval encoding* ser o mais eficiente dos esquemas base, este apenas reduz o tamanho para metade em relação os restantes. Desta forma é imperativo o uso de outras formas para minimizar o espaço requerido. De todos os mecanismos de *encode* existentes, o que gera o menor número de *bitmaps* é o *binary encoding* [Wong et al. (1985)], sendo mais tarde inserido no contexto de índices *bitmap* por O'neil & Quass (1997) e Wu & Buchmann (1998) (figura 12) (baseada em [Wu & Buchmann, 1998]).

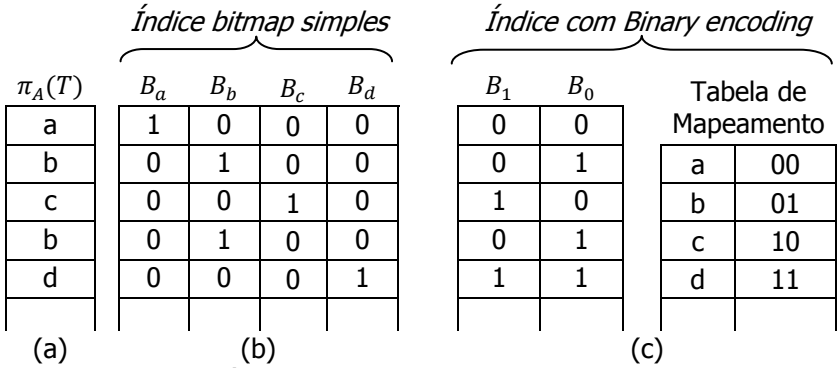


Figura 12 – Comparação de um índice *bitmap* simples (b) e *binary encoding* (c) para o atributo A projectado em (a).

Através deste método, um atributo de cardinalidade C apenas necessita de $\log_2 C$ *bitmaps*, isto é, por exemplo para um C=1000 apenas são necessários 10 *bitmaps*. Contudo, apesar deste mecanismo superar claramente a técnica de *interval encoding* ao nível do espaço usado, para satisfazer uma RQ este mecanismo necessita tipicamente de aceder a todos os *bitmaps* contra apenas dois usados pelo *interval encoding*.

3.1.6 Multi-Component Encode

Várias estratégias de *encode* foram propostas por vários autores com o objectivo de melhorar a eficiência tempo-espaço. Chan & Ioannidis, (1998b;1999) desenvolveram um novo método denominado por *multi-component encoding* que segundo Stockinger & Wu (2006), pode ser visto como uma generalização do *binary encoding*. Esta estrutura pode ser facilmente modelada através

de alguns parâmetros (número de componentes e seus respectivos tamanhos) gerando diferentes optimizações de tempo-espaço. O esquema de *multi-component encode* consiste numa estrutura de obtida em duas fases:

1) Decomposição do valor do atributo. Previamente é necessário decidir o número de componentes da estrutura (n) e os seus respectivos tamanhos ($< b_{n-1}, b_{n-2}, \dots, b_1 >$), sendo $b_n = \left\lceil \frac{C}{\prod_{i=1}^{n-1} b_i} \right\rceil$. Assim, qualquer valor v pode ser decomposto numa sequência de n dígitos $< v_n, v_{n-1}, \dots, v_1 >$ através da seguinte equação (6):

$$\begin{aligned} v &= V_1 \\ &= V_2 b_1 + v_1 \\ &= V_3(b_2 b_1) + v_2 b_1 + v_1 \\ &= V_4(b_3 b_2 b_1) + v_3(b_2 b_1) + v_2 b_1 + v_1 \\ &= \dots\dots\dots \\ &= v_n \left(\prod_{j=1}^{n-1} b_j \right) + \cdots + v_i \left(\prod_{j=1}^{i-1} b_j \right) + \cdots + v_2 b_1 + v_1, \end{aligned} \qquad (6)$$

com $v_i = V_i \% b_i$, $V_i = \left\lfloor \frac{V_{i-1}}{b_{i-1}} \right\rfloor$, $1 < i \leq n$ e $0 \leq v_i < b_i$.

Conforme a escolha do número de componentes (n) e os seus respectivos tamanhos ($< b_{n-1}, b_{n-2}, \dots, b_1 >$), obtém-se um índice diferente e consequentemente com características de eficiência tempo-espaço também diferentes. A sequência $< b_n, b_{n-1}, \dots, b_1 >$ corresponde à base do índice, tipicamente designado de *n-component index* com Base-$< b_n, b_{n-1}, \dots, b_1 >$. Segundo os autores Chan & Ioannidis (1998b), um índice deste tipo está bem definido se $b_i \geq 2$, com $1 \leq i \leq n$. Para o caso de $< b_n = b_{n-1} = \cdots = b_1 >$ a base é dita uniforme, podendo ser representada pela abreviatura Base-*b*.

2) Codificação do *bitmap*. Esta segunda fase corresponde à codificação dos valores da sequência $< v_n, v_{n-1}, \dots, v_1 >$ nos componentes correspondentes. Cada um dos n componentes pode ser codificado com qualquer uma das formas de *encode* base anteriormente apresentadas, *equality encoding, range encoding, interval encoding*. A estrutura de indexação final resulta num conjunto de *bitmaps* B_i^j, onde j representa o j-ésimo *bitmap* do i-ésimo componente com o esquema de codificação B. De salientar que as equações (1) a (5) anteriormente apresentadas permanecem válidas, sendo aplicadas de forma individual em cada um dos n componentes do índice. Note-se

também que as características e vantagens anteriormente apresentadas entre os três métodos base também permanecem válidas para cada componente.

A figura 11 (baseada em [Chan & Ioannidis, 1999]) demonstra a codificação dos valores da figura 9(a) num índice de 2 componentes de base-<3,4> utilizando *equality encoding* (figura 13(b)) e *range encoding* (figura 13(c)). Neste exemplo todos os atributos foram decompostos em dois dígitos através da metodologia anteriormente apresentada com base na fórmula $v = v_2 b_1 + v_1$, onde $v_1 = v\%b_1$ e $v_2 = \left\lfloor \frac{v}{b_1} \right\rfloor$, sendo $v_1 \in [0,3]$ e $v_2 \in [0,2]$ posteriormente codificados nos componentes 1 e 2 respectivamente.

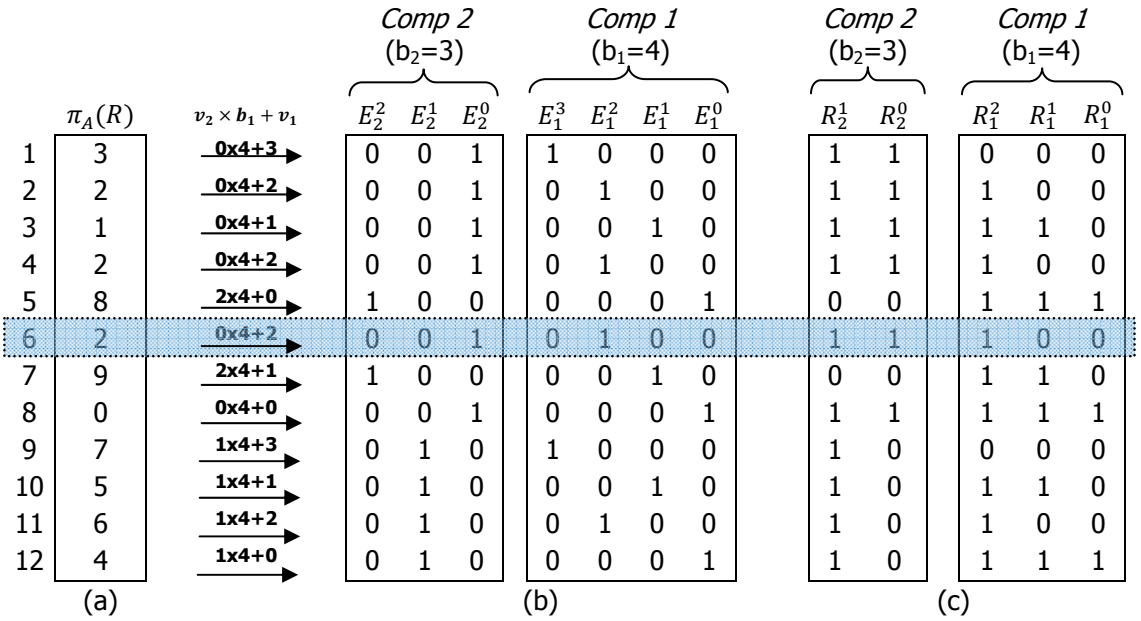


Figura 13 – Exemplo de índice de 2 componentes de Base-<3,4> (C=10). (a) Projecção do atributo indexado (duplicados mantidos). (b) Índice *equality-encoded*. (c) Índice *range-encoded*.

Tomemos o valor 2 destacado na figura 13 para exemplificar a decomposição de um valor para $n = 2$ e $< b_2, b_1 > = < 3,4 >$. Assim, com base nas fórmulas $v_1 = v\%b_1$ e $v_2 = \left\lfloor \frac{v}{b_1} \right\rfloor$, tem-se $v_1 = 2\%4 = 2$ e $v_2 = \left\lfloor \frac{2}{4} \right\rfloor = 0$, que confirma a igualdade $2 = 0 \times 4 + 2$. No segundo passo, $v_1 = 2$ e $v_2 = 0$ são codificados (*bits* a '1') respectivamente nos *bitmaps* E_1^2 e E_2^0 no caso de *equality encoding* e nos *bitmaps* R_1^2, R_2^0 e R_2^1 no caso de *range encoding.*

Aquando do processamento de uma *query,* o valor da condição é decomposto através da equação (6) para determinar a sequência de dígitos $< v_n, v_{n-1}, \dots, v_1 >$ pelo qual o valor é composto, tendo em conta o número de componentes (n) do índice de base $< b_n, b_{n-1}, \dots, b_1 >$. Posteriormente, o processamento da *query* é realizado em duas fases: fase de reescrita da *query* seguida da fase de avaliação da *query* [Chan & Ioannidis, 1999]. É através deste processo que são determinados os *bitmaps* a serem utilizados. Considere-se como exemplo uma *query* com a condição "$A \leq 5$". Como se está perante um índice de 2 componentes, v_1 e v_2 são obtidos por $v_1 = 5\%4 = 1$ e $v_2 = \left\lfloor \frac{5}{4} \right\rfloor = 1$. Assim, a *query* anterior é eficientemente traduzida para "$v_2 \leq 0\ OR\ (v_2 \leq 1\ AND\ v_1 \leq 1)$"[10] utilizando os *bitmaps* R_2^0, R_2^1 *e* R_1^1 no processamento da *query*, correspondentes às condições "$v_2 \leq 0$", "$v_2 \leq 1$" e "$v_1 \leq 1$". Determinados os *bitmaps* a ser usados, a *query* é processada de acordo com sua tradução anterior. Todos os passos do processamento são demonstrados no esquema seguinte (figura 14), onde se encontra destacado o resultado final.

$\pi_A(R)$	3	2	1	2	8	2	9	0	7	5	6	4
R_1^1	0	0	1	0	1	0	1	1	0	1	0	1
R_2^0	1	1	1	1	0	1	0	1	0	0	0	0
R_2^1	1	1	1	1	0	1	0	1	1	1	1	1
$R_2^1 \wedge R_1^1$	0	0	1	0	1	0	0	1	0	1	0	1
$R_2^0 \vee (R_2^1 \wedge R_1^1)$	1	1	1	1	1	1	0	1	0	1	0	1

∧	0	1
0	0	0
1	0	1

∨	0	1
0	0	1
1	1	1

Figura 14 – Esquema de representivo do processamento de uma *query* sobre índice de 2 componentes de Base-<3,4>.

Com este método de *encode* à uma redução de 10 para 7 e de 9 para 5 *bitmaps* comparativamente a *equality encoding* e *range encoding,* respectivamente. Este método continua a usufruir das vantagens dos métodos de *encode* base, ou seja, a sua eficiência de tempo nos respectivos tipos de *queries*. Note-se que apesar de este método ser menos eficiente em termos de espaço do que *binary encoding*, para a *query* anterior, este apenas necessita de três *bitmaps* contra os quatro que seriam utilizados com *binary encoding.* Para um atributo de maior cardinalidade, esta diferença torna ainda mais evidente e distante.

Com o método de *multi-component encoding* é esperada uma redução no tamanho total do índice conforme se vai aumentando o número de componentes. Contudo, quanto mais componentes o

[10] Note-se para "A<5", a *query* seria traduzida para "$v_2 \leq 0\ OR\ (v_2 \leq 1\ AND\ v_1 < 1)$" utilizando os *bitmaps* R_2^0, R_2^1 *e* R_1^0, correspondentes às condições "$v_2 \leq 0$", "$v_2 \leq 1$" e "$v_1 \leq 0$".

índice tiver pior será o tempo de resposta do mesmo, uma vez que terão de ser acedidos um maior número de *bitmaps* para satisfazer uma *query*. Este *trade-off* espaço-tempo foi analisado por Chan & Ioannidis (1999) na sua forma normal, sem recorrer a qualquer tipo de compressão. Os resultados por estes obtidos, sugerem que ponto de inflexão ocorre com o uso de dois componentes, sendo sugerido que os dois componentes sejam aproximadamente do mesmo tamanho, de forma a minimizar o espaço utilizado.

3.1.7 Esquemas de encode híbridos para membership queries

Chan & Ioannidis (1999) propuseram posteriormente quatro esquemas híbridos de diferentes *tradeoffs* espaço-tempo para *membership queries*, ou seja, *queries* com condições do tipo "$A \in \{v_1, v_2, v_3, \dots, v_k,\}$". Uma *query* deste tipo consiste basicamente num conjunto de *equality*, *range* e *interval queries*. Tome-se o exemplo "$A \in \{1,2,3,6,8,9,10,13\}$", com $1 \leq A \leq k$. Esta condição pode ser facilmente traduzida na seguinte disjunção "$(A \leq 3) \vee (A = 6) \vee (8 \leq A \leq 10) \vee (A = 13)$" por forma a minimizar os o número de operações a serem realizadas. Os quatro esquemas são:

1) *Equality-Range Encode*. Este esquema de *encode* denotado de $\mathcal{ER}$, consiste numa simples combinação dos esquemas base de *equality* e *range encoding*, ou seja, $\mathcal{ER} = \mathcal{E} \cup \mathcal{R} = \{\mathrm{E}^0, \mathrm{E}^1, \dots, \mathrm{E}^{\mathrm{C}-1}, \mathrm{R}^0, \mathrm{R}^1, \dots, \mathrm{R}^{\mathrm{C}-2}\}$. Note-se que os *bitmaps* R^0 e $\mathrm{R}^{\mathrm{C}-2}$ não necessitam ser armazenados uma vez $\mathrm{R}^0 = \mathrm{E}^0$ e $\mathrm{R}^{\mathrm{C}-2} = \overline{\mathrm{E}^{\mathrm{C}-1}}$. Com este tipo de esquema, as parcelas da disjunção podem ser processadas com o método de *encode* mais apropriado, ou seja, *equality encoding* para EQ e *range encoding* para RQ. Uma vez que *equality* e *range encoding* são óptimos para EQ e 1RQ respectivamente, este esquema híbrido normalmente irá obter um bom desempenho de tempo, à custa de quase o dobro do espaço dos esquemas base.

2) *OREO Encode*. O esquema de *encode* OREO (*Oscillating Range and Equality Organization*) representado por $\mathcal{O}$, consiste na utilização do melhor dos dois esquemas base que compõem a estratégia anterior. Este esquema necessita da mesma quantidade de espaço que o esquema base *range encoding*. Uma das ideias base deste esquema centra-se na minimização do espaço. O esquema de *encode* OREO consiste numa mistura intercalada de aproximadamente 50-50 dos esquemas base *equality* e *range encoding*.

Assim, o esquema OREO constituído por C-1 *bitmaps* $\mathcal{O} = \{O^1, O^2, \ldots, O^{C-1}\}$, sendo dos seus componentes dados por seguinte equação (7):

$$\text{Para } 1 \leq i \leq C-1,$$
$$O^i = \begin{cases} \bigvee_{i \text{ é par}} E^i, & se\ i = C-1 \\ E^{i-1} \vee E^i, & se\ i \text{ é } par \\ R^i, & caso\ contrátio \end{cases} \qquad (7)$$

Note-se que o *bitmap* O^{C-1} corresponde à disjunção de todos os *bitmaps* par de *equality encoding*, tendo sido esta uma das formas encontradas pelos autores para melhorar o esquema OREO em EQ. Por exemplo, uma *query* do tipo "$A = v$" é resolvida através da expressão "$(O^v \oplus O^{v-2}) \wedge \overline{O^{C-1}}$". A forma de como uma *query* EQ, 1RQ ou 2RQ é processada através deste método é dada respectivamente, pelas equações (10), (11) e (12), em [Chan & Ioannidis, 1998a] apêndice A.

3) <u>*Equality-Interval Encode*</u>. O terceiro esquema de *encode*, é muito semelhante ao primeiro aqui abordado. O *Equality-Interval encoding* é representado por $\mathcal{EI}$ e consiste na combinação dos esquemas base de *equality* e *interval encoding*, ou seja, $\mathcal{EI} = \mathcal{E} \cup \mathcal{I} = \{E^0, E^1, \ldots, E^{C-1}, I^0, I^1, \ldots, I^{\lceil \frac{C}{2} \rceil - 1}\}$. De salientar que para C<3, $\mathcal{EI}$ é dado por $\mathcal{E}$. Este esquema permite que as parcelas da disjunção sejam processadas através *equality encoding* para EQ e *interval encoding* para RQ. Este consegue obter um melhor desempenho de tempo do que $\mathcal{O}$ e ser mais eficiente em espaço do que $\mathcal{ER}$.

4) <u>*Variante de Equality-Interval Encode*</u>. O último esquema proposto por Chan & Ioannidis (1999) é denotado por $\mathcal{EI}^*$, sendo uma melhoria do esquema anterior. Este esquema é baseado na observação da equações (3), (4) e (5), onde se verifica que o *bitmap* $I^0 = \left[0, \left\lfloor \frac{C}{2} \right\rfloor - 1\right]$ é utilizado no processamento da maioria das *queries*. Sucintamente, este esquema consiste num conjunto de $\left(\left\lceil \frac{C}{2} \right\rceil + \left\lceil \frac{C-4}{2} \right\rceil\right)$ *bitmaps*, $\mathcal{EI}^* = \mathcal{I} \cup \{P^0, P^1, \ldots, P^r\}$, com $r = \left\lceil \frac{C-4}{2} \right\rceil$ e $P^i = E^i \cup E^{i+m+1}$, cerca de 2/3 de $\left(C + \left\lceil \frac{C}{2} \right\rceil\right)$ utilizados pelo esquema anterior. Note-se que para $C \leq 4$, $\mathcal{EI}^* = \mathcal{I}$. As equações (4) e (5) continuam válidas para RQ, sendo que as EQ são processadas de acordo com a equação (7) em [Chan & Ioannidis, 1998a].

Com este novo método, a avaliação de uma *query* de igualdade necessita maioritariamente de dois *bitmaps* contra apenas um do esquema $\mathcal{EI}$. No entanto, como o *bitmap* I^0 é tipicamente utilizado em todas as RQ, este apenas necessita de ser lido de disco uma vez. Desta forma, a eficiência de tempo deste mecanismo é aproximadamente a mesma que a de esquema $\mathcal{EI}$ para a maioria das *queries*.

Apesar destes mecanismos híbridos demonstrarem algum potencial, e intuitivamente melhor desempenho, Chan & Ioannidis, (1999) referem que os mesmos raramente conseguem obter melhores resultados que os mecanismos simples, referindo que apenas em algumas situações é que os mecanismos híbridos conseguem obter uma ligeira melhoria de tempo, no entanto, à custa da utilização de maiores volumes de espaço.

3.2 Binning

Na secção anterior foram apresentados vários métodos de codificação com vista à minimização do espaço necessário no uso de índices *bitmap* e o desempenho nos vários tipos de *queries* (EQ, RQ e IQ). Todos os métodos até agora apresentados, demonstram ser capazes e eficientes para o caso de atributos de baixa/média cardinalidade. Assim, para atributos de elevada cardinalidade o espaço requerido e o tempo de computação tornam-se demasiado elevados, o que pode tornar o uso de índices do tipo *bitmap* inviável. Neste sentido, devem ser consideradas outras possibilidades de minimização do espaço utilizado por este tipo de índices para atributos de elevada cardinalidade.

Nesta secção é abordado um outro método de optimização de índices *bitmap* denominado de *binning*. Este visa minimizar o espaço utilizado por estas estruturas de indexação, tornando assim a sua utilização viável para atributos de elevada cardinalidade. Este método pode ser utilizado em conjunto com os mecanismos de *encode* abordados na secção anterior.

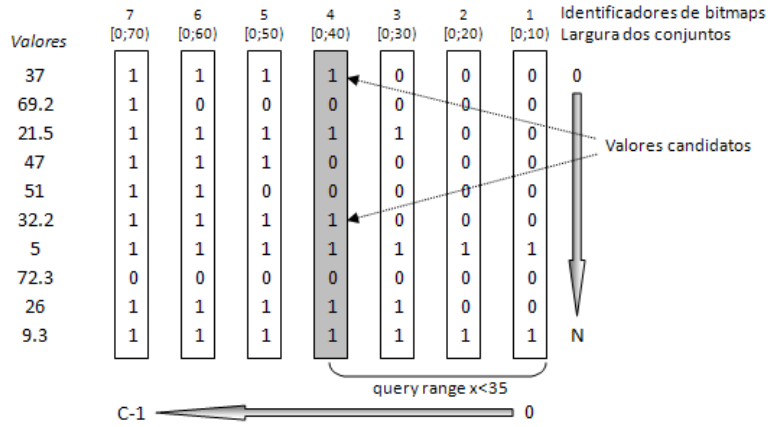


Figura 15 – Range query x<35 sobre *bitmap range encode* com *equi-width binning*.

A ideia base deste mecanismo consiste na construção de conjuntos de valores (*bins*), onde cada um é representado por um *bitmap*. Desta forma, temos um *bitmap* por cada conjunto de valores (*bin*) ao invés de se ter um *bitmap* por cada valor distinto. Este mecanismo permite um maior controlo sobre o tamanho do índice, uma vez que o número de *bitmaps* deixa de ser influenciado pela cardinalidade do atributo indexado. No entanto, apesar deste método permitir uma optimização mais personalizada da sua dimensão, o mesmo acrescenta um grau de incerteza aquando do processamento de uma *query*, uma vez que cada *bitmap* deixa de corresponder a um valor específico, mas sim a um conjunto de valores (*bin*). Com este tipo de mecanismo, para se obter um resultado exacto pode ser necessário para alguns casos a leitura de alguns dados base para se efectuar uma confirmação directa dos dados que correspondem aos requisitos especificados numa *query*. Este mecanismo é especialmente apropriado para atributos decimais.

Considere-se, como exemplo, um atributo X (figura 15) (baseada em [Stockinger et al., 2004]) com uma gama de valores entre 0 e 70. Este encontra-se indexado com um índice *bitmap* sobre a forma de *range encode* com *binning*. Tome-se então uma *query* contendo a seguinte restrição "x<35". Neste caso, o valor de restrição está contido no quarto *bitmap*. No entanto, este *bitmap* corresponde a todos os valores iguais ou inferiores a 40. À partida, sabemos que todos os valores indexados pelo terceiro *bitmap* satisfazem a restrição da *query*. Contudo, ainda é preciso determinar os valores que são indexados pelo quarto *bitmap*. Para tal, através de uma operação "XOR" sobre os *bitmaps* 3 e 4 determina-se o conjunto {37, 32.2} de valores candidatos que possivelmente podem satisfazer a condição de filtragem. Após a leitura dos valores em causa para se proceder uma verificação directa, verifica-se que um dos valores candidatos cumpre os requisitos de filtragem da *query*. Este processo de leitura de dados do disco e posterior análise,

Stockinger et al. (2004) e Rotem et al. (2004; 2005a; 2005b) designam de verificação de candidatos (do inglês *candidate check*).

Segundo Rotem et al. (2004; 2005b) este processo de verificação de candidatos é o principal responsável pelo *bottleneck* no processamento de *queries*. Este problema pode ser atenuado através de estratégias de optimização para a criação dos grupos de valores. Wu & Yu (1996;1998) apresentaram um esquema de *binning* de índices *bitmaps* para atributos de alta cardinalidade denominado *range-based bitmap indexing*. Segundo Rotem et al. (2004;2005b) a ideia base passa pela distribuição dinâmica e uniforme dos valores distorcidos (*skewed*) de um atributo por vários conjuntos de *bitmaps* para que todas as *queries* tenham um comportamento similar.

Na persecução da optimização do processo de verificação, Stockinger et al. (2004) apresentaram três estratégias para minimizar os custos da verificação dos candidatos. Estas estratégias têm como principal objectivo a minimização do acesso aos dados aquando da verificação directa. De uma forma resumida, as estratégias propostas por Stockinger et al. (2004) consiste na manipulação dos índices *bitmap* dos atributos envolvidos numa *query* de forma a determinar que valores terão de ser acedidos para verificação.

Considere-se, agora, uma *query* dimensional de ordem 2 (com dois atributos restritivos). A primeira estratégia utiliza todos os atributos de uma só vez numa primeira fase, de forma determinar o conjunto de valores candidatos para serem avaliados, como exemplifica a interpretação gráfica exposta na figura 16 (extraída de [Stokinger et al., 2004]). Uma série de operações são efectuadas sobre o *bitmaps* dos atributos envolvidos até ser determinada a área candidata (conjunto de valores) a ser verificada. Posteriormente, numa segunda fase todos os valores representados pela área candidata anteriormente determinada são lidos do disco verificando-se se os mesmos satisfazem as restrições de cada atributo da *query* (figura 17, extraída de [Stokinger et al., 2004]).

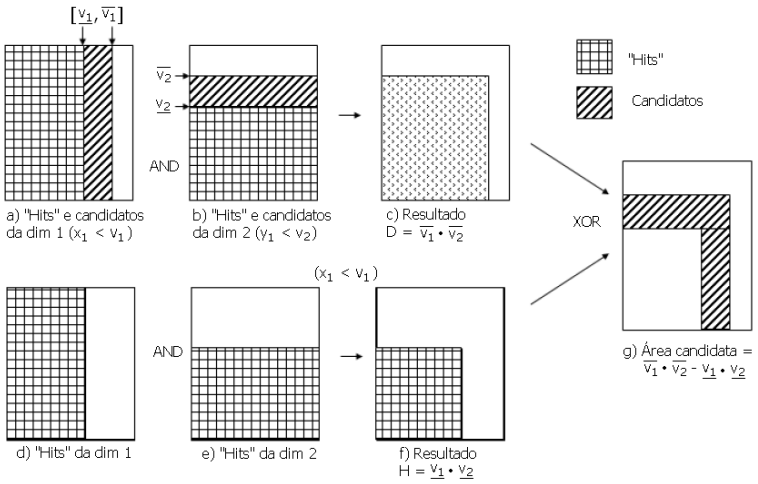


Figura 16 – Interpretação gráfica do cálculo da área candidata para a estratégia 1 e 3.

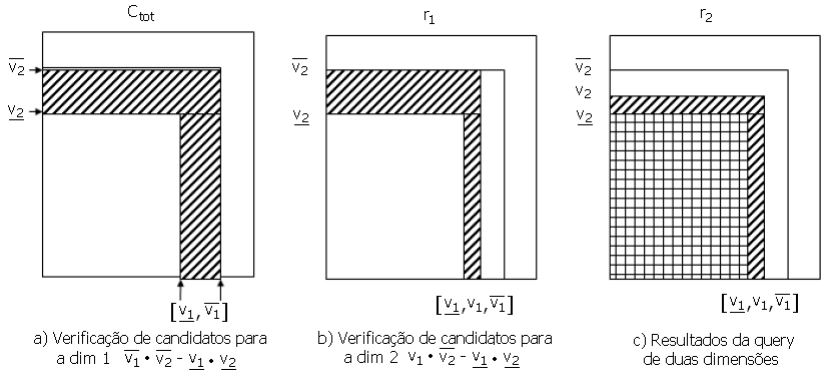


Figura 17 – Representação da avaliação dos índices *bitmap* segundo a estratégia 1.

A segunda estratégia consiste na avaliação de cada dimensão (atributo) de uma forma individual, minimizando em alguns casos a quantidade de valores a serem verificados. Neste caso, o índice *bitmap* do primeiro atributo é avaliado de forma a determinar os valores candidatos. De seguida é efectuada logo a verificação dos valores candidatos do atributo anterior (figura 18, extraída de [Stokinger et al., 2004]). Concluídos estes dois passos, os valores correspondentes à condição do primeiro atributo estão concluídas. Com base no resultado da análise do primeiro atributo (figura 18 d)) um procedimento similar ao anterior é efectuado para o segundo atributo. Concluído o procedimento para o segundo atributo obtém-se o resultado final que satisfaz a *query*.

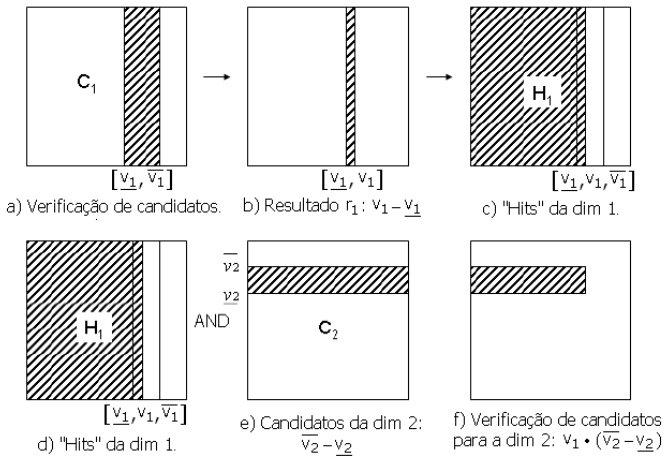


Figura 18 - Representação da avaliação dos índices *bitmap* segundo a estratégia 2.

Por fim, a terceira estratégia consiste numa mistura das duas anteriores (figura 19, extraída de [Stokinger et al., 2004]), avaliando o número mínimo de candidatos. Inicialmente começa-se por determinar a área candidata da mesma forma que na estratégia 1 (figura 16). Posteriormente, ao contrário do que acontece na estratégia 1, apenas é verificada a área candidata correspondente ao primeiro atributo, sendo para tal realizada uma operação lógica entre a área candidata global e a área candidata do primeiro atributo (figura 19 a)-c)). Concluída a verificação da área anteriormente calculada, o mesmo processo é realizado para o segundo atributo tendo em conta a nova área global (figura 19 e)), determinando assim o resultado final que satisfaz a *query*.

As duas primeiras estratégias apenas necessitam de aceder aos índices *bitmap* uma vez, no entanto a terceira estratégia necessita aceder duas vezes a cada *bitmap*. Esta última estratégia minimiza o número de acessos aos registos em disco à custa de um acréscimo de processamento na manipulação dos índices *bitmap*. Stockinger et al. (2004) demonstram que com o uso de índices *bitmap* com *binning* pode aumentar significativamente o desempenho das *queries*. Os mesmos concluíram que apesar da última estratégia ter uma carga computacional superior às anteriores, na generalidade dos casos esta obtém melhores desempenhos que os duas primeiras estratégias, minimizando a quantidade de acessos a registo aquando da verificação. Segundo os mesmos autores, em alguns casos, a última estratégia pode proporcionar uma optimização no tempo de uma *query* na ordem dos 50%.

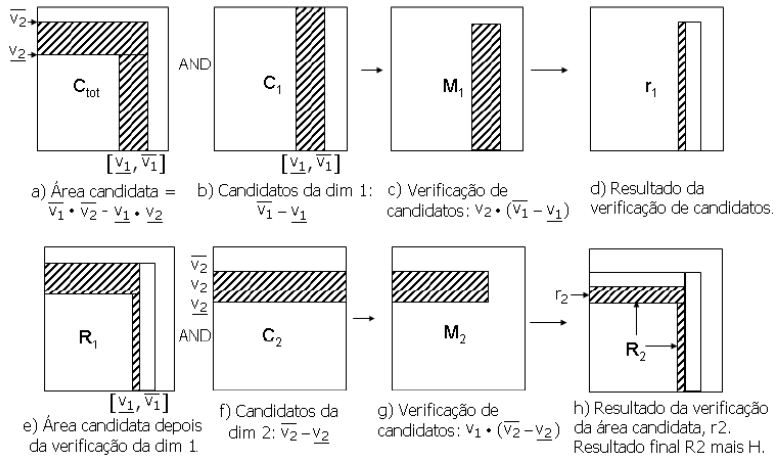


Figura 19 - Representação da avaliação dos índices *bitmap* segundo a estratégia 3.

Uma outra forma de optimização de índices *bitmap* com *binning* prende-se com o método de construção dos conjuntos de valores (*bins*). Estes podem ser construídos através de dois métodos base muito conhecidos: *equi-width* e *equi-depth*. O *equi-width* corresponde a ter todos os grupos de valores com um intervalo fixo, como é o exemplo da figura 15. No caso de *equi-depth* consiste em manter todos os conjuntos com a mesma quantidade de valores. Muitas outras formas de optimização da construção destes conjuntos foram apresentadas [Koudas, 2000; Rotem et al., (2004;2005a;2005b;2006)]. Todos os autores anteriormente referenciados desenvolveram soluções baseadas em programação dinâmica para determinar a dimensão de cada conjunto (*bin*) de forma a minimizar os acessos a disco aquando do processo de verificação. Estas implementações têm em consideração a distribuição de valores dos atributos, assim como o conjunto de *queries* conhecidas mais frequentes.

A implementação de Koudas (2000) teve como propósito encontrar a forma de *binning* mais apropriada para EQ. Outros autores [Rotem et al. (2004;2005a;2005b)] viram as RQ como o principal objectivo. Os resultados obtidos por estes autores demonstram que as suas soluções baseadas em programação dinâmica tornam o processo de verificação dos candidatos mais eficiente, em comparação com os métodos mais convencionais (*equi-width* e *equi-depth*). Contudo, estas abordagens baseadas em programação dinâmica podem não ser as melhores soluções. Tipicamente, o tempo e complexidade em programação dinâmica tende a aumentar de forma quadrática, o que pode tornar estas soluções inviáveis de usar em SDW com elevadas cardinalidades ou elevadas quantidades de *queries* conhecidas [Stockinger & Wu, 2006]. Mais recentemente, Wu et al, (2008) apresentaram um novo método de *binning* com base numa

estrutura *Order-preserving Bin-based Clustering* (OrBiC). Desta maneira, eles conseguem minimizar o custo da verificação de candidatos. Como forma de optimização, os mesmos propõem ainda uma forma híbrida de *binning*, mantendo os valores frequentemente utilizados em *bins* de valor único, eliminando desta forma o processo de verificação de candidatos para estes valores.

3.3 Compressing

A compressão representa a terceira estratégia de optimização de índices *bitmap*. Actualmente estão já bem identificados e comprovados vários esquemas de compressão [Johnson, 1999; Amer-Yahia & Johnson, 2000; Wu et al., (2001a;2001b)]. No entanto, apesar de muitos destes mecanismos serem eficientes na compressão de dados, neste caso, na compressão de índices *bitmap*, não implica que sejam óptimos ou aconselháveis para usar em estruturas de indexação. Isto é, apesar de serem eficientes na minimização de espaço, os métodos de compressão precisam também ser altamente eficientes na sua manipulação (operações lógicas) aquando do processamento de uma *query* [Johnson, 1999]. Em SDW, este factor tem ainda mais peso, uma vez que um DW tipicamente é caracterizado pela sua elevada cardinalidade e necessidade de eficiência em *querying*. Como os *bitmaps* de um índice *bitmap* são muitas vezes utilizados individualmente, tipicamente a compressão é feita de forma individual sobre cada *bitmap* [Stockinger & Wu, 2006]. O tempo de I/O normalmente é o maior responsável pela maioria do tempo gasto no processamento de uma *query*, o que faz com que a grande parte das técnicas de indexação se foque neste aspecto. No caso dos índices *bitmap* a grande maioria da pesquisa efectuada prende-se precisamente com o mesmo aspecto, a minimização do espaço utilizado por estas estruturas [Wu et al., 2006]. Stockinger et al. (2002) demonstram que com a utilização de índices *bitmap* com compressão, o factor espaço não é o principal problema a considerar. Segundo estes, com o uso de compressão a grande parte do tempo de execução de uma *query* prende-se com a computação (tempo de CPU) efectuada sobre os índices comprimidos.

Esquemas de compressão como o LZ77 utilizado no gzip [Ziv & Lempel, 1977; Gailly & Adler, 2002] são soluções já reconhecidas pela sua capacidade e eficácia de compressão. No entanto, vários estudos empíricos [Johnson, 1999; Amer-Yahia & Johnson, 2000; Wu et al., (2001a;2002;2004)] demonstraram que de todos os esquemas de compressão existentes, o *Byte-Aligned Bitmap Code* (BBC) [Antoshenkov, (1994;1995)] e o *Word-Aligned Hybrid code* (WAH) [Wu et al., (2004;2006)] são os que mais se destacam, obtendo os melhores desempenhos em *querying*. Estes

demonstraram ser os esquemas de compressão com melhor desempenho tendo em consideração a contrapartida espaço-tempo. O WAH é normalmente o esquema de compressão que obtém resultados mais rápidos e o BBC demonstra uma capacidade de compressão idêntica ao gzip, tendo também uma capacidade de realização de operações lógicas bastante eficiente. [Wu et al., (2002;2006b)].

Ambos os esquemas apresentados são baseados em *run-length encode* (RLE) representando neste caso sequências de *bits* '0' ou '1'. A ideia base consiste na representação de sequências de *bits* iguais através do tipo de *bit* (*fill bit*) e a sua quantidade. Esta representação é denominada de *fill*. Um *fill* corresponde a um conjunto de *bits* sequenciais do mesmo tipo ('0' ou '1'). Uma sequência de *bits* '0' designa-se de *0-fill* e uma sequência de *bits* '1' designa-se por *1-fill*. Dependendo do método de compressão, cada tipo de *fill* tem um determinado tamanho e limite de *bits*, o que faz que com algumas sequências de *bits* não possam ser comprimidas. Isto é, sempre que uma sequência de *bits* correspondente ao tamanho de um *fill* seja composta por um misto de *bits* '0' e '1', essa sequência terá de ser deixada de fora, tendo de ser armazenada na sua forma literal (o exemplo de codificação com WAH representado na figura 20 (extraída de [Stokinger & WU, 2006]) elucida este acontecimento). A esta sequência de *bits* dá-se o nome de *tail*. Em ambos os esquemas de compressão BBC e WAH, um *bitmap* aquando da compressão é divido em *runs*. Um *run* corresponde a um *fill* seguido de uma *tail*. Segundo Wu et al. (2004), um *bitmap* é incompressível se todos os *bits* tiverem de ser armazenados de forma literal, ou seja, um *bitmap* não está comprimido se todos os *bits* forem armazenados de forma literal.

Byte-aligned Bitmap Code

O *Byte-aligned Bitmap Code* (BBC) foi desenvolvido por Antoshenkov e patenteado pela Oracle [Antoshenkov, (1994;1995); Antoshenkov & Ziauddin, 1996]. Segundo vários estudos, o BBC é um dos esquemas de compressão conhecidos que demonstram maior eficiência [Wu et al., 2002]. Este esquema de compressão existe em duas variantes, *1-sided BBC* e *2-sided BBC*. O *1-sided BBC* foi o primeiro a ser desenvolvido e consiste na compressão de *0-fills*. Posteriormente foi desenvolvido o *2-sided BBC* permitindo a compressão de ambos os tipos de *fills* (*0-fill* e *1-fill*). Segundo Wu et al. (2002), para *bitmaps* esparsos é aconselhado o uso de *1-sided BBC*. Aquando da compressão de um *bitmap* (sequência de *bits*) o BBC começa por dividir o mesmo em grupos de *bytes* e posteriormente em grupos de *runs*. Como referido anteriormente, um *run* é constituído por um *fill* seguido de uma *tail*. Em BBC, um *fill* não a uma sequência de *bits* do mesmo tipo mas sim a uma

sequência de *bytes* que contêm o mesmo tipo de *bits*. Cada *run* usa ainda um cabeçalho (*header*) de um *byte*. Dependendo das sequências de *bits*, um *fill* pode ser considerado de dimensão reduzida[11]. Nestes casos, a sua dimensão (quantidade de *bytes*) é armazenada no cabeçalho. Para um *fill* de maior dimensão, vários *bytes* poderão ser utilizados de forma a representar o total (*counter*) de *bytes* do *fill*. Por fim, o BBC também tem uma forma de identificar um tipo de *tail* especial. Isto é, nos casos onde se tem apenas um *bit* que é diferente da maioria e onde a maioria é do mesmo tipo que o *fill* anterior, a posição do *bit* diferente é armazenada no cabeçalho utilizando três *bits*.

Na sua forma mais simples, o BBC divide os vários *runs* em quatro tipos diferentes (tabela 3, extraída de [Wu et al., 2004]), permitindo assim uma compressão mais eficiente - para uma explicação mais detalhada ver [Wu et al., 2004].

Tipo	Fill	Tail	Encoding
1	curto	normal	header, tail
2	curto	especial	header
3	longo	normal	header, counter, tail
4	longo	especial	header, counter

Tabela 3 - Tipos de *runs* em BBC *encode*.

Word-Aligned Hybrid code

O *Word-Aligned Hybrid code* (WAH) é considerado um dos esquemas de compressão mais eficientes, senão o mais eficiente de para compressão para índices *bitmap*. Numa abordagem redutora, este esquema corresponde a uma abordagem mais simplista da utilizada no método BBC. Este consiste numa solução híbrida baseada em *run-length encoding* e *bitmaps* literais [Wu et al., (2001a;2002)], onde uma sequência de *bits* do mesmo tipo é representada pelo valor do *bit* e a sua quantidade [Stockinger & Wu, 2006]. Tome-se a figura 20 (extraída de [Stokinger & WU, 2006]) como exemplo para mais facilmente se perceber o método de compressão usado pelo WAH. Considere-se ainda que a compressão é realizada numa máquina com uma arquitectura de 32 *bits* e sobre um *bitmap* com uma sequência de 5456 *bits* (figura 20 a)). Inicialmente toda a sequência de *bits* é dividida em vários grupos de 31 *bits* (figura 20 b)). Posteriormente, todos os grupos de 31 *bits* sequenciais do mesmo tipo são juntos, formando um *fill*. No caso do exemplo da figura 20, verifica-se uma sequência de 176 grupos de *bits* a '0' que dá origem a um único *fill*

[11] Um *fill* pequeno corresponde a um máximo de 3 e 7 *bytes*, para *1-sided* BBC e *2-sided* BBC, respectivamente.

(figura 20 c)). Todos os grupos que contenham *bits* mistos são deixados de na sua forma literal, formando uma *tail*. Desta forma, obtém uma compressão em dois *runs* (figura 20 d)). O primeiro *run* é constituído por um *fill* de tamanho nulo e uma *tail* com uma dimensão de 31 *bits*. Como cada palavra tem um tamanho de 32 *bits*, o primeiro *bit* é utilizado como forma de marcação do tipo de palavra, ou seja, se é um *fill, bit* '1' ou uma *tail, bit* '0'. No caso do segundo *run*, este é constituído por duas palavras, um *fill* e a uma *tail*. Como anteriormente referido, note-se que o primeiro *bit* da palavra *fill* indica o tipo da palavra, seguido pelo segundo *bit* que indica o tipo de *fill*, neste caso um *0-fill*. Os restantes 30 *bits* correspondem ao tamanho do *fill*, ou seja, neste caso a um total de 174*31 *bits* a '0'. Por fim, temos a *tail* correspondente aos *bits* armazenados de forma literal.

Vizinhos

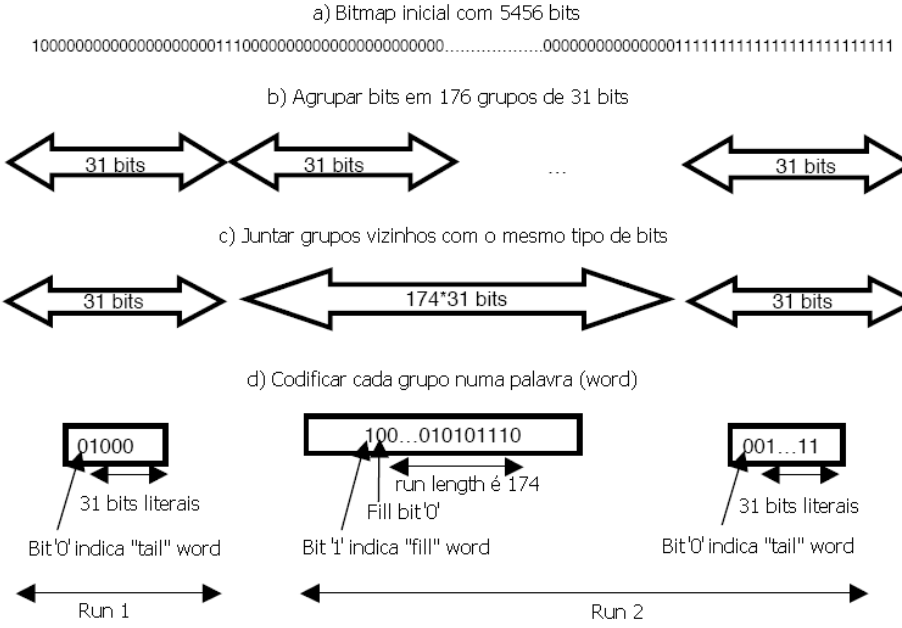


Figura 20 – Exemplo de compressão de uma sequência de 5456 *bits* numa máquina de 32 *bits*, utilizando WAH.

Numa forma global, o tempo de execução de uma *query* pode ser dividido entre o tempo I/O e o tempo de processamento. Existe muito a noção de que o tempo de processamento (tempo de CPU) em muitas operações em BD não é um factor relevante [Stockinger & Wu, 2006]. Assim, como o BBC consegue obter uma melhor compressão, na teoria o BBC deveria obter melhores desempenhos que o WAH. No entanto, um estudo efectuado por Stockinger et al. (2002), realizado com base em várias implementações [Johnson, 1999; Wu et al., 2002] destes dois métodos de compressão, revelam que apesar de o BBC demonstrar uma capacidade de compressão superior ao

WAH, a simplicidade de compressão utilizada pelo último faz com que consiga obter optimizações na ordem dos 50% face ao BBC.

Outros estudos [Wu et al. 2001a, 2002, 2004; Stockinger et al. 2002] sugerem que independentemente da cardinalidade dos atributos, com o uso de *bitmaps* comprimidos com WAH as *queries* são realizadas de forma mais rápida do que com índices *bitmap* simples, índices de projecção, ou *B-Trees*. Wu et al. (2006b) concluíram no seu estudo, que os índices *bitmap* comprimidos com WAH têm desempenho óptimo. Segundo os mesmos, apesar de outras estruturas de indexação similarmente óptimos ao WAH, o facto de o resultado de uma dimensão poder ser combinado para responder a *queries* multi-dimensionais, faz com que o WAH seja óptimo para *queries ad hoc* de grandes dimensões. No que diz respeito ao tamanho dos índices, Wu et al. (2002;2004) provam que no pior dos casos, um índice *bitmap* comprimido com WAH tem um tamanho similar a um índice do tipo *B-Tree*. Goyal et al. (2006) sugerem ainda que os atributos a serem indexados com índices *bitmap* comprimidos devem ser ordenados. Desta forma as sequências de *bits* de cada *bitmap* permitem uma melhor compressão como mostram os resultados obtidos pelos autores. Por fim, O'Neil et al. (2007) analisaram a forma de armazenamento de dados e de índices *bitmap*. Como conclusão do estudo efectuado, os autores sugerem o uso de uma organização vertical para dados e uma organização linear para índices *bitmap* para se obter um bom desempenho em processos de satisfação de *queries*.

3.4 Implementação de sistemas com bitmaps

Até ao momento foram abordadas várias características e estratégias existentes na literatura de *Data Warehousing* para os índices *bitmap*, no entanto, a maioria das soluções já apresentadas nunca chegaram a ser implementadas em SGBD. Ao longo dos anos, os índices *bitmap* tem demonstrado uma grande competência na diminuição do tempo de satisfação de *queries*. Apesar da sua eficiência ser reconhecida na literatura actual de *Data Warehousing*, segundo O'Neil et al. (2007), nos últimos anos os índices *bitmap* não tem sido muito adoptados pelos SGBD comerciais. Os mesmos referem ainda que "apenas alguns SGBD tem implementado índices *bitmap* devido às dificuldades da alteração de pressupostos fundamentais no *design* de baixo nível de um SGBD, bem como às próprias expectativas dos clientes". Outro factor apontado pelos mesmos autores é o facto de não existir um desenho ou estrutura definitiva para os índices *bitmap* e de muitas das implementações existentes para vários SGBD serem para arquitecturas de computadores antigas.

O *Model 204* [O'neil, 1987] da *Computer Corporation of America* foi o primeiro produto a ser lançado para o mercado com a primeira implementação de um índice *bitmap* num SGBD. Actualmente existe uma vasta variedade SGBD comerciais que se têm vindo a destacar da concorrência, demonstrando uma crescente capacidade em dar sustentabilidade a um DW. Alguns destes SGBD têm pelo menos uma solução de indexação que recorre aos *bitmaps* ou a alguma das suas variantes.

Num estudo apresentado pela Gartner[12] (figura 21, extraída de [WWW1]) podemos ver quais foram até 2008 os SGBD mais conceituados do mercado de *Data Warehousing* - o Quadrante mágico de SGBD para *Data Warehousing*. O quadrante apresentado na figura 18 distingue as várias soluções para *Data Warehousing* presentes no mercado por quatro categorias distintas[13]: os líderes, os desafiadores/concorrentes, os visionários e os "*niche players*".

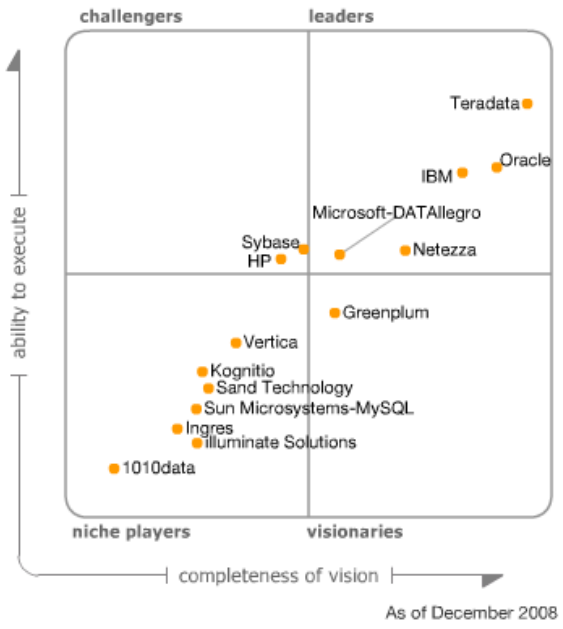


Figura 21 – Quadrante mágico 2008 de SGBD para *Warehouse* 2008 (Gartner).

Os SGBD líderes de mercado são aqueles que demonstram uma maior estabilidade, capacidade e garantia no suporte de qualquer tipo de SDW. Estes são também os SGBD mais conceituados pelos

[12] A Gartner Inc foi fundada em 1979 e é actualmente uma empresa mundialmente líder em pesquisa de informação em tecnologia e em consultoria para empresas.

[13] Critérios de análise em [WWW1].

clientes e que evidenciam maiores garantias de sucesso de um DW a longo prazo. Os desafiadores/concorrentes (*Challengers*) correspondem aos SGBD que demonstram capacidade para dar suporte a SDW de dimensões consideráveis, sendo utilizados por muitos clientes como uma forma de iniciação a SDW. Estes demonstram potencial para entrar para o grupo dos líderes. O grupo dos visionários corresponde aqueles que entram para o mercado com novas soluções e tecnologias que em muitos casos ainda não estão provadas, logo tem uma carência de fiabilidade. Os produtos presentes neste quadrante tem tipicamente uma percentagem de mercado mais reduzida que os dois grupos anteriores. No entanto, são estes que muitas das vezes fazem com que os grandes "*players*" do mercado avancem para novas soluções e abordagens para melhoria de desempenho das suas soluções. Por fim, os "*niche players*" correspondem aqueles que exploram e se especializam em pequenos nichos de mercado/indústria onde o tipo de exigências e necessidades é muito específico. Muitas das soluções mais desenvolvidas no mercado iniciam nesta fase. É neste tipo de mercados que muitas organizações demonstram o potencial da sua solução, levando em muitos casos ao crescimento da organização.

A tabela 4 mostra de uma forma resumida o uso que cada um dos SGBD líderes do mercado de *Data Warehousing* faz ou não, dos índices do tipo *bitmap*. Apesar de vários deles terem já uma forma de índices *bitmap* implementada, outros utilizam os *bitmaps* como uma forma de optimização interna, sendo para tal criados internamente de uma forma dinâmica.

SGBD	**Descrição**
Teradata	Utiliza maioritariamente um mecanismo *hashing* como forma de indexação para os vários tipos de índices, não disponibilizando a possibilidade de criação de índices do tipo *bitmap*. Disponibiliza também uma forma de índices *join* que aparenta também utilizar um mecanismo de *hashing* [WWW3; WWW4].
Oracle	Utiliza índices *bitmap* na sua forma mais simples (*equality encoding*). Permite também a criação de índices do tipo *join* denominados de índices *bitmap join*. De forma a melhorar os gastos de espaço, o Oracle comprime automaticamente este tipo de índices através do método de compressão BBC [Antoshenkov, (1994;1995); Antoshenkov & Ziauddin, 1996].
IBM	Não permite a criação directa de índices *bitmap*. O DB2 cria índices *bitmap* internamente de uma forma dinâmica a partir dos índices existentes. Estes são criados sempre que o optimizador do DB2 "achar" conveniente para a resolução de uma determinada *query*.
Microsoft SQL Server	Não permitem a criação de índices *bitmap* da mesma forma que o Oracle. Segundo a documentação [WWW5], o SQL Server faz um uso dos *bitmaps* similar ao DB2, uma vez que usa os *bitmaps* internamente como forma de optimização de *queries* (*bitmap filtering*).
Microsoft-DATAllegro	O DATAllegro foi adquirido pela Microsoft em Agosto de 2008, sendo integrado no projecto

	da Microsoft *Madison Project*[14]. Este ainda se encontra numa fase de construção, tendo por objectivo o uso de processamento paralelo massivo de forma a permitir uma melhor performance e escalabilidade.
Netezza	Não utiliza qualquer tipo de indexação convencional, não permitindo desta forma a criação de índices *bitmap*. Este utiliza uma estrutura própria denominada de *zone map* que é mantida automaticamente.
Sybase IQ	Utiliza vários tipos de indexação convencionais, tendo implementado *bit-sliced index* [O'Neil & Quass, 1997], o que segundo Stockinger & Wu (2006), na terminologia usada anteriormente neste capítulo, corresponde a uma codificação binária (*binary encode*), sem compressão e sem binarização.

Tabela 4 – Sintese do uso de índices *bitmap* em produtos comerciais lideres de mercado de DW.

O Teradata é um SGBD que visa na sua especialização em SDW de grandes dimensões. Este é caracterizado por ser um sistema de processamento paralelo massivo (do inglês *Massive Parallel Processing* (MPP)). Utiliza um mecanismo de *hashing* na criação dos vários tipos de índices disponibilizados, índices primários, secundários, *join*, entre outros [WWW3], não permitindo criação de índices do tipo *bitmap*.

No caso da IBM, o DB2 não permite a criação directa de índices *bitmap*. Este implementa uma variante de *bitmaps* codificados de uma forma binária (*binary encoded bitmap index*) denominada *Encode Vector Index* [Stockinger & Wu, 2006]. O DB2 utiliza índices do tipo *bitmap* de uma forma mais passiva, ou seja, estes apenas são criados aquando do processamento de uma *query* sempre que o optimizador "achar" conveniente. A criação dos índices *bitmap* é assim efectuada dinamicamente utilizando os índices existentes. Desta forma, o DB2 evita todo o peso de criação e manutenção dos índices *bitmap*, criando-os "*on the fly*" nas situações que achar conveniente [WWW6; WWW7].

O SQL Server da Microsoft utiliza índices do tipo *bitmap* de uma forma similar ao DB2 da IBM. Este também não permite a criação de índices *bitmap*, no entanto faz uso dos mesmos de uma forma interna como um método de optimização de *queries* (*bitmap filtering*). Este mecanismo tem uma estrutura similar a um índice *bitmap*, no entanto tem a diferença de a sua estrutura ser mantida em memória aquando do processamento de uma *query* e de a sua criação ter pouco impacto no tempo de processamento da mesma [WWW5]. Desta forma evita-se também o custo envolvido na

[14] Para mais informação seguir o link: http://www.microsoft.com/Sqlserver/2008/en/us/madison.aspx

manutenção destes índices durante as várias operações de manipulação de dados (*Data Manipulation Language* (DML)). Segundo a documentação referida anteriormente, esta optimização pode ser introduzida no plano de execução após optimização ou pode ser introduzida dinamicamente pelo optimizador de *queries* do SQL Server, sendo denominado neste caso por *optimized bitmap filter*.

O Netezza é um sistema de SGBD que não utiliza qualquer tipo de estruturas de indexação convencionais. Este contém um mecanismo próprio de denominado *zone map* que é mantido automaticamente. Este mecanismo apenas "regista o valor mínimo e máximo de colunas do tipo "inteiro" e "data" por cada 3 megabyte em disco" [WWW8]. Desta forma o Netezza evita a manutenção de chaves primárias e dos vários tipos de indexação convencionais. Este aposta na sua capacidade de processamento paralelo (*Massive Parallel Processing* (MPP)) e no seu mecanismo de leitura de dados do discos que lhes permite efectuar uma filtragem dos registos pretendidos.

No caso da Sybase, o Sybase IQ[15] é um SGBD com uma estrutura diferente do tradicional, seguindo uma arquitectura de armazenamento por colunas (*columnar/column-store architecture*). Este produto possui vários tipos de indexação que fazem uso de dos *bitmaps*. Inicialmente o Sybase IQ começou por apenas fazer uso de índices *bitmap* segundo a sua implementação mais simples (*equality encode*) [WWW9]. Segundo MacNicol & French (2004) "as listas de registos dos índices são armazenadas num formato comprimido como *bitmaps* segmentados". O tipo de índice *Low Fast Index* é um dos tipos de indexação que fazem uso dos *bitmaps* [Howard, 2008], sendo utilizados para atributos de baixa cardinalidade.

O Oracle foi o SGBD escolhido por nós para dar suporte ao estudo do impacto dos índices *bitmap* sobre um SDW. Como mostra o gráfico da figura 21, o Oracle é um dos SGBD mais importantes do mercado em *Data Warehousing* e que já possui uma implementação de índices do tipo *bitmap*. Desta forma, permite que o estudo efectuado fique mais próximo da realidade, recorrendo apenas e exclusivamente a implementações já integradas em SGBD, uma vez que vários dos estudos efectuados com este tipo de índices envolve implementações de índices *bitmap* que não se encontram em nenhum SGBD. Para além disto, esta é uma forma de aprendizagem e de contacto

[15] "Highly optimized business intelligence, analytics and data warehousing" (http://www.sybase.com/products/datawarehousing/sybaseiq)

com este SGBD em particular, uma vez que sem dúvida o Oracle é um dos grandes SGBD líderes de mercado, detendo cerca de 48% [WWW1; WWW2] do mercado de Sistemas de Gestão de Bases de Dados Relacionais (SGBDR) (do inglês *Relational Database Management System Software*).

O Oracle disponibiliza dois tipos de indexação que envolvem *bitmaps*, sendo estas disponibilizadas apenas nas versões *Enterprise Edition* e *Standard Edition*. Este utiliza uma versão de índices *bitmap* comprimidos [Stockinger & Wu, 2006], sendo os índices *bitmap* codificados na sua forma básica (*equality encode*). A compressão é feita de forma dinâmica e automática pelo Oracle aquando da criação do índice, utilizando para tal o método de compressão BBC [Antoshenkov, (1994;1995); Antoshenkov & Ziauddin, 1996], o que permite uma optimização de espaço muito significativa, face aos índices comuns (*B-Tree*). Segundo Bryla & Loney (2008), um índice *bitmap* no Oracle pode ter cerca de 1/10 do tamanho de um índice comum, o que numa forma generalizada pode corresponder a cerca de 2 a 10% do tamanho de um índice comum, para atributos de baixa cardinalidade. O segundo tipo de *bitmap* denominado de índices *bitmap join*, corresponde a uma variante dos índices *bitmap* simples [Bryla & Loney, 2008] que permite a criação de um índice sobre um atributo de uma tabela que é agregada através da operação *join* a uma ou mais tabelas.

Todos estes tipos de *bitmaps* podem ser facilmente utilizados em conjunto com outro tipo de índices devido à capacidade de conversão dos índices *bitmap* para *rowids* por parte do optimizador de *queries* do Oracle. Desta forma, beneficia-se na mesma da cooperação entre os vários tipos estruturas de indexação. A tabela 5 (extraída de [Alapati, 2009]) mostra uma breve comparação entre os índices *B-Tree* e os índices do tipo *bitmap* no Oracle.

Índices *B-tree*	**Índices *Bitmap***
Bom para atributos de elevada cardinalidade	Bom para atributos de baixa cardinalidade
Bom para bases de dados de sistemas OLTP	Bom para aplicações de *Data Warehousing*
Usam elevadas quantidades de espaço	Usam pouco espaço
Fáceis de actualizar	Difíceis de actualizar

Tabela 5 - Comparação de índices *B-Tree* e índices *bitmap* em Oracle.

Capítulo 4

Influência dos Índices bitmap na Satisfação de Queries

4.1 Caracterização e Análise do Sistema Alvo

Para a realização da componente prática desta dissertação foi utilizado como sistema de testes um *data webhouse* implementado em Oracle 11g *Enterprise Edition*, a funcionar sobre uma máquina com um processador Dual Core @ 1.83GHz, com 3GB de memória e com a base de dados sobre um disco externo de 1TB. Este *data webhouse* é constituído por uma tabela de factos, sete dimensões (uma das quais degenerada), uma subdimensão (*outrigger*) e uma tabela ponte. O mesmo foi povoado com dados de *clickstream* de um website organizados segundo o modelo dimensional apresentado na figura 22, de forma a permitir a análise do registo de todas as sessões completas por hora [Marques e Guimarães, 2009]. Na tabela 6 está apresentada uma breve descrição de cada tabela, de forma a elucidar as várias funcionalidades do *data webhouse* referido.

Entidade	**Tabela**	**Descrição**
Tabela de Factos	TF_Sessions	Contém o registo de sessões completas por hora.
Dimensão Computador	ComputadorUtilizador_Dim	Identifica o computador de onde é proveniente o pedido http ao servidor.
Dimensão User Agent	UserAgent_Dim	Identifica o utilizador/aplicação que acedeu ao site (*Web browser*, *search engine bots (crawler)*, etc).
Dimensão Calendário	Data_Dim	Caracteriza o dia de ocorrência do facto contendo

		informação detalhada sobre datas para um período de doze anos.
Dimensão Hora	Tempo_Dim	Guarda a informação referente a cada uma das vinte e quatro horas do dia, caracterizando a mesma para cada ocorrência de um facto.
Dimensão Referenciador	Referrer_Dim	Identifica o local que referenciou o determinado pedido http, podendo ser através de um *link* do próprio site ou externamente ao mesmo.
Dimensão Pedido	Request_Dim	Identifica todos os pedidos feitos ao site, ou seja, toda a informação que é disponibilizada aquando de um *click* num *link.*
Dimensão Sessão	SessionID	É uma dimensão degenerada, permitindo desta forma efectuar agregações por sessão.
SubDimensão Pais	Pais_SubDim	Contém a identificação do país dos utilizadores que acederam ao site.
Tabela Ponte	TP_Session	Contém a informação relativa a cada pedido efectuado durante uma sessão completa.

Tabela 6 - Caracterização das tabelas que constituem o data webhouse

Este *data webhouse* foi construído com base num modelo dimensional, em floco de neve, como se pode verificar pela figura 22. Note-se, também, a existência de uma tabela ponte, que surge a partir da resolução da relação N:M entre a tabela de factos "TF_Sessions" e a dimensão "Request_Dim". Nesta tabela ponte é mantido um número sequencial que permite identificar a ordem pela qual todos os pedidos de uma sessão foram efectuados.

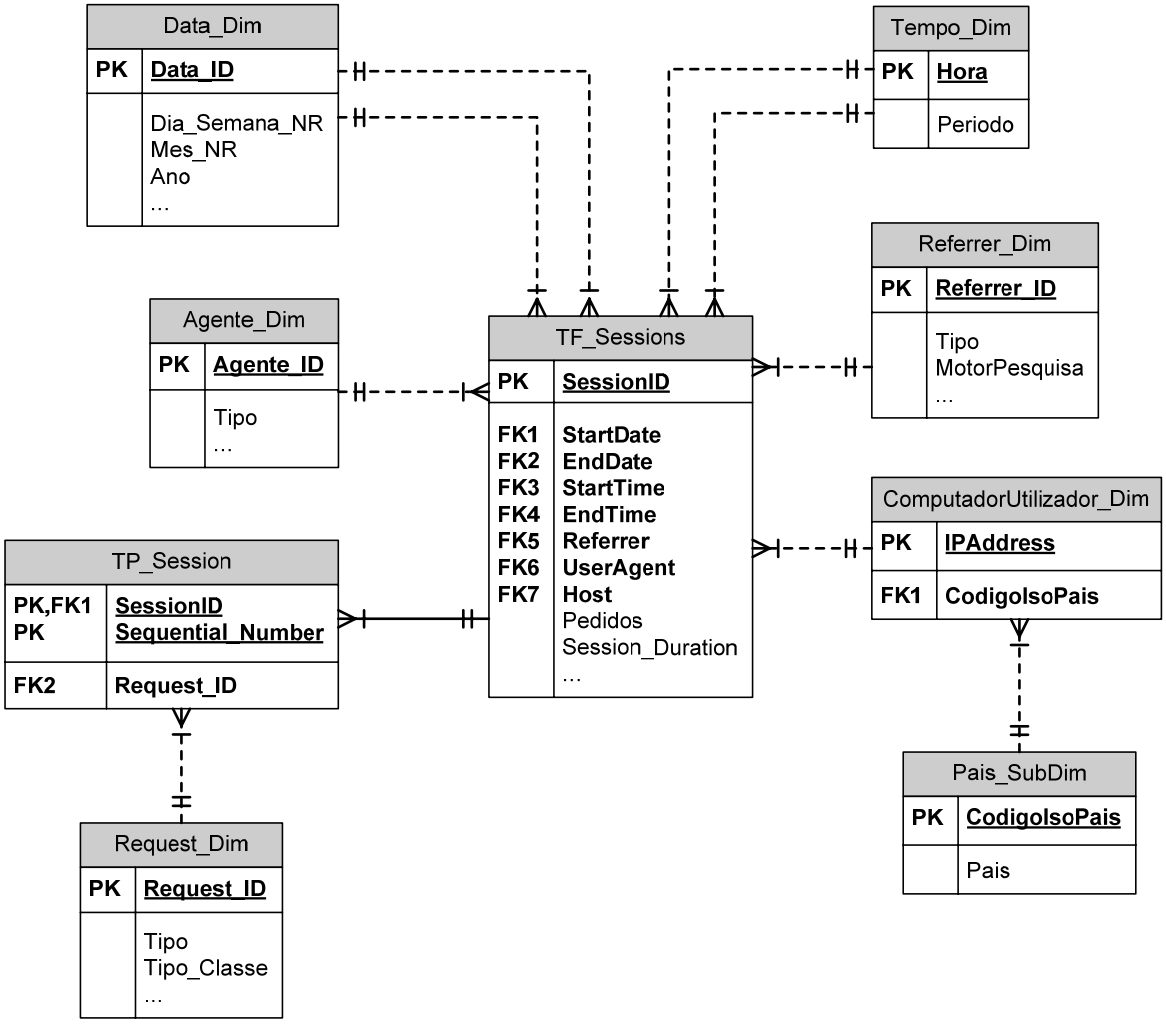


Figura 22 - Esquema dimensional do *data webhouse* alvo.

4.2 Queries de teste

No âmbito deste trabalho pretendeu-se analisar a influência dos índices *bitmap* sobre um DW específico. Após a análise da estrutura base do *data webhouse* e dos seus dados foi necessário definir um conjunto de *queries* típicas a que poderia estar sujeito um DW deste tipo. Na terminologia de OLAP, este conjunto é usualmente denominado por as *top-K queries*. Por outras palavras, existe um conjunto de *queries* que normalmente são mais efectuadas pelos utilizadores. As operações de agregação são as mais utilizadas e importantes para dar suporte à decisão [Li e tal., 2005]. Os utilizadores OLAP, tipicamente, necessitam da informação agregada sobre

determinadas eixos de análise, ou seja, sobre um conjunto de atributos que transpareçam esses mesmos eixos – as dimensões.

O Oracle dispõe de três tipos de agregação bastante úteis para análise [WWW11]: *GROUP BY*, *ROLLUP* e *CUBE*, que correspondem a operações tipicamente efectuadas em ambientes do género dos referidos. O conjunto de *queries* escolhido encontra-se dividido em três tipos, correspondendo cada um deles aos vários tipos de agregação anteriormente referidos. Cada um deles inclui sete categorias diferentes, o que faz como que tenhamos, no final, um conjunto de *queries* de teste constituído por um total de 21 *queries*.

A estrutura mais conhecida e típica de uma *query* OLAP está apresentada na tabela 7 [Li et al., 2005], sendo as *queries* deste tipo classificadas do Tipo 1.

Estrutura	**Exemplo**
SELECT ga1, ..., gam, F FROM R1, ..., Rh WHERE c1 AND ... AND cl GROUP BY ga1, ..., gam ORDER BY F LIMIT k	SELECT deptno, job, count(*), sum(sal) FROM emp GROUP BY deptno, job ORDER BY deptno, job;
Resultado	

```
    DEPTNO JOB         COUNT(*)    SUM(SAL)
---------- --------- ---------   ---------
        10 CLERK               1        1300
        10 MANAGER             1        2450
        10 PRESIDENT           1        5000
        20 ANALYST             2        6000
        20 CLERK               2        1900
        20 MANAGER             1        2975
```

Tabela 7 - Estrutura, exemplo e resultado de uma *query* do tipo 1.

O segundo tipo de *queries* utiliza o comando *ROLLUP* disponibilizado pelo Oracle. Este devolve a mesma informação que o tipo anterior, juntamente com o cálculo dos subtotais dos vários níveis de agregação. Por fim, apresenta-se, também, o total global, ou seja, o total dos subtotais. A tabela 8 mostra um exemplo que permite obter uma fácil percepção do resultado dado por esta

operação. Para mais informação ver [WWW11, WWW12, WWW13]. Em [WWW11] pode-se encontrar ainda um exemplo de uma agregação envolvendo 3 dimensões.

Estrutura	**Exemplo** [WWW12]
SELECT ga1, ..., gam, F FROM R1, ..., Rh WHERE c1 AND ... AND cl GROUP BY ROLLUP (ga1, ..., gam) ORDER BY F	SELECT deptno, job, count(*), sum(sal) FROM emp GROUP BY ROLLUP (deptname,job);

Resultado [WWW12]

DEPTNO	JOB	COUNT(*)	SUM(SAL)
10	CLERK	1	1300
10	MANAGER	1	2450
10	PRESIDENT	1	5000
10		**3**	**8750**
20	ANALYST	2	6000
20	CLERK	2	1900
20	MANAGER	1	2975
20		**5**	**10875**
		8	**19625**

Tabela 8 - Estrutura, exemplo e resultado de uma *query* do tipo 2.

Por fim, o terceiro tipo de *queries* utiliza o comando *CUBE* também disponibilizado pelo Oracle. Este devolve a mesma informação que os dois tipos de *queries* anteriores. Note-se porém, que com o *ROLLUP* não são calculados todos os tipos de subtotais possíveis. Para tal, pode-se utilizar o comando *CUBE* que calcula todos os subtotais, para todas as possíveis combinações de atributos que compõem a agregação. Por fim, apresenta-se também o total global. Na tabela 9 podemos ver um exemplo que permite analisar o resultado dado por esta operação. Para mais informação ver [WWW11, WWW12, WWW13]. Em [WWW11] pode-se encontrar ainda um exemplo utilizando este operador com uma agregação envolvendo 3 dimensões.

Estrutura	**Exemplo** [WWW12]
SELECT ga1, ..., gam, F FROM R1, ..., Rh WHERE c1 AND ... AND cl GROUP BY CUBE (ga1, ..., gam)	SELECT deptno, job, count(*), sum(sal) FROM emp GROUP BY CUBE (deptno,job);

ORDER BY F	

Resultado [WWW12]

DEPTNO	JOB	COUNT(*)	SUM(SAL)
10	CLERK	1	1300
10	MANAGER	1	2450
10	PRESIDENT	1	5000
10		**3**	**8750**
20	ANALYST	2	6000
20	CLERK	2	1900
20	MANAGER	1	2975
20		**5**	**10875**
30	CLERK	1	950
30	MANAGER	1	2850
30	SALESMAN	4	5600
30		**6**	**9400**
	ANALYST	**2**	**6000**
	CLERK	**4**	**4150**
	MANAGER	**3**	**8275**
	PRESIDENT	**1**	**5000**
	SALESMAN	**4**	**5600**
		14	**29025**

Tabela 9 - Estrutura, exemplo e resultado de uma *query* do tipo 3.

Por fim, as categorias de cada tipo de *query* correspondem à quantidade de tabelas nele usadas, ou seja, uma *query* de categoria 4 tem uma junção constituída pela tabela de factos mais três dimensões. Uma *query* do tipo 2 de categoria 4, corresponde a uma *query* que efectua uma agregação através da instrução *ROLLUP* utilizando uma junção de 4 tabelas. Em suma, existem 3 tipos de *queries* com 7 categorias cada.

4.3 O que Indexar e como Indexar

Seguindo a abordagem de Kimball et al. (2008), a criação de índices é um processo que deve ser muito bem planeado, de forma a se poder beneficiar do melhor desempenho possível de um DW. Tipicamente, os índices são criados sobre tabelas onde na maioria dos casos são seleccionadas pequenas quantidades de dados. Segundo Alapati (2009), para *queries* que utilizem mais de 10 a 15% dos dados de uma tabela, pode não ser necessário criar índices sobre a mesma, visto que,

em grande parte dos casos, pode ser mais rápido efectuar um varrimento completo da própria tabela.

A selectividade dos atributos indexados é um dos factores mais importantes a ter em consideração, uma vez que em ambientes *Data Warehousing* se pode encontrar um misto de atributos altamente selectivos e outros de selectividade mínima, ou seja, com apenas 2 valores distintos. Assim, Bryla & Loney (2008) sugerem que se considere a utilização de índices *bitmap* sobre os atributos de menor selectividade, uma vez que estes se mostram mais eficientes para *queries* sobre grandes quantidades de dados com atributos de baixa cardinalidade.

Quando nos deparamos com a tarefa de indexação, a questão que a maioria das pessoas coloca é: "Que atributos se devem indexar?". Neste sentido, Kimball et al. (2008) e Alapati (2009) sugerem um conjunto de casos/atributos, para além dos atributos chave, para serem considerados alvos de indexação:

- Chaves estrangeiras.
- Colunas predicado.
- Atributos utilizados na junção de tabelas.
- Indexar atributos sobre os quais são frequentemente utilizados em operações *ORDER BY* e *GROUP BY*.

Alapati (2009) sugere ainda:

- Evitar indexar atributos do tipo *String* de grande extensão.
- Utilizar, sempre que possível planos de execução que apenas utilizem os índices para satisfazer as *queries*.

Note-se que, este último ponto obriga a um estudo bem detalhado sobre as *queries* que são executadas e também requer a criação de um conjunto de índices compostos, o que em muitos casos pode não ser desejável ou em que os ganhos não são compensatórios.

No caso de índices compostos a Oracle [Loney, 2009] aconselha uma atenção especial. Aquando da criação de um índice composto deve-se ter em consideração a ordem pela qual se efectua a indexação dos atributos. Desta forma, o índice deve ser criado do atributo mais selectivo para o menos selectivo, ou seja, do atributo com a maior quantidade de valores distintos, para o de menor quantidade de valores distintos. Apesar destas considerações, deve-se verificar se estes índices são utilizados na maioria das *queries*, uma vez que são índices que tipicamente têm

maiores períodos de computação e maiores necessidades de armazenamento, quando comparados com um índice simples. Assim, se um índice composto for utilizado pouquíssimas vezes, a Oracle sugere uma ponderação para a sua remoção, sugerindo a criação de um índice individual sobre cada atributo que compõe o índice composto em causa.

Após criação dos índices desejados (ou considerados) necessários é aconselhada uma monitorização do uso de cada índice [Niemiec, 2007], permitindo assim saber quais os índices que realmente estão a ser usados. Desta forma, pode-se não só ganhar espaço em disco, como também tempo na manutenção dos mesmos, tempo este precioso especialmente para sistemas com uma baixa janela de oportunidade para operações de ETL.

No seguimento da análise da influência dos índices *bitmap* sobre um DW e, após uma cuidadosa e elaborada análise do *data webhouse* e do conjunto que *queries* de teste, foi decidida a criação de índices (tabela 10) sobre as chaves estrangeiras, sobre os atributos utilizados em condições de restrição (*where*) e sobre os atributos utilizados em operações de agregação (*group by*) e ordenação (*order by*). Todas estas decisões foram tomadas com base nas recomendações da literatura de *Data Warehousing* [Kimball et al., 2008] e do SGBD [Alapati, 2009].

Tabela	Atributo	FK	"Where"	"Group/Order by"	Selectividade
TF_Sessions	StartDate	×			4383
	EndDate	×			4383
	StartTime	×			24
	EndTime	×			24
	Referrer	×			3936
	UserAgent	×			32679
	Host				398988
	Pedidos		×	×	90
	Bytes_Enviados		×		10000
	Session_Duration		×		1800
Data_Dim	Mes_Nr			×	12
	Ano		×		12
Tempo_Dim	Periodo			×	3

Referrer_Dim	Tipo			×	2
	MotorPesquisa			×	2
Agente_Dim	Tipo		×		100
ComputadorUtilizador_Dim	CodigoIsoPais	×			100
Pais_Subdim	Pais			×	100
TP_Session	SessionID	×			1207600
	Request_ID	×			16287
Request_Dim	Tipo			×	100
	Tipo_Classe		×		50

Tabela 10 – Atributos indexados para efeitos de teste.

Como referido anteriormente, nesta dissertação pretende-se analisar o impacto dos índices *bitmap* em SDW. Para tal, foram elaborados vários tipos de testes considerando várias formas e tipos de indexação. Como base de comparação para os diversos índices considerados neste estudo foram utilizados as *B-Tree*, uma vez que são as estruturas de indexação mais populares e, por omissão, mais utilizadas na maioria dos SGBD. No caso da Oracle, os índices *B-Tree*, muitas vezes referidos como índices *B*Tree*, são os índices que por omissão são considerados, utilizando-os aquando da criação de uma chave-primária de uma tabela, impondo assim a restrição de unicidade de cada valor do atributo chave [Loney, 2009; Alapati, 2009; Price, 2008].

4.4 Análise de resultados

Após os testes terem sido realizados sobre o *data webhouse* procedemos à análise dos resultados segundo dois dos aspectos mais relevantes em *Data Warehousing*: o tempo e o espaço. Assim, começámos por uma análise do espaço necessitado pelos índices *bitmap*, seguida de uma análise do tempo da sua criação e por fim uma análise do tempo de execução das *queries*. Em tudo isto, não nos devemos esquecer que os índices *B-Tree* (índices por omissão do Oracle) são usados como base de comparação dos índices *bitmap*.

Como é característico dos DW a tabela que mais rapidamente tende a aumentar o seu volume de registos é a tabela de factos. Assim, como referido na análise do sistema de testes alvo, inicialmente a tabela de factos encontrava-se povoada com 1,2 milhões de registos, o que devido à natureza do *data webhouse* corresponde a ter cerca de 70 milhões de registos na tabela ponte. Assumindo para efeitos de teste que as dimensões são dimensões de variação lenta, para se poder efectuar uma análise mais viável e rigorosa do impacto no espaço, aquando do uso de índices *bitmap*, foram dobrados sucessivamente o volume de dados da tabela de factos até 153,6 milhões de registos, o que implicitamente resultou em cerca de 9.216 milhões de registos na tabela ponte um valor já significativo.

Volume	Cardinalidade tabela factos	Cardinalidade tabela ponte
1	≈ 1,2 milhões	≈ 72 milhões
2	≈ 2,4 milhões	≈ 144 milhões
3	≈ 4,8 milhões	≈ 290 milhões
4	≈ 9,6 milhões	≈ 576 milhões
5	≈ 19,2 milhões	≈ 1.152 milhões
6	≈ 38,4 milhões	≈ 2.304 milhões
7	≈ 76,8 milhões	≈ 4.608 milhões
8	≈ 153,6 milhões	≈ 9.216 milhões

Tabela 11 - Correspondência de volumes de dados

Seguindo a ordem de trabalhos dos testes realizados, começou-se por analisar os índices *bitmap* segundo o espaço requerido. Os índices *bitmap* para atributos de baixa e média cardinalidade demonstram ser superiores aos índices *B-Tree*. Considerando a média dos dois índices mais volumosos dos vários volumes de dados criados sobre as chaves estrangeiras "sessionID" e "request_ID" da tabela ponte, verificam-se em média reduções de espaço na ordem dos 93% e 87%, respectivamente.

Os gráficos apresentados nas figuras 23 e 24 elucidam bem a diferença de tamanhos dos dois tipos de estrutura de indexação, neste caso sobre os atributos que são chaves estrangeiras, sujeitos ao impacto do aumento do volume de dados da tabela de factos. No caso do volume de dados 5 tem-se uma dimensão da base de dados na ordem dos 24,75 GB. Após a criação das chaves primárias e estrangeiras, a base de dados aumentou em cerca de 95% correspondendo a mais 23,75GB. No entanto, estes são gastos que são à partida necessários, independentemente

dos tipos de indexação utilizados. Assim, para efeitos de comparação, considere-se como espaço base o tamanho dispendido pelos dados das tabelas mais o tamanho dos índices implicitamente criados aquando da criação das chaves primárias. Neste caso ficámos, então, com um tamanho base de 48,5GB.

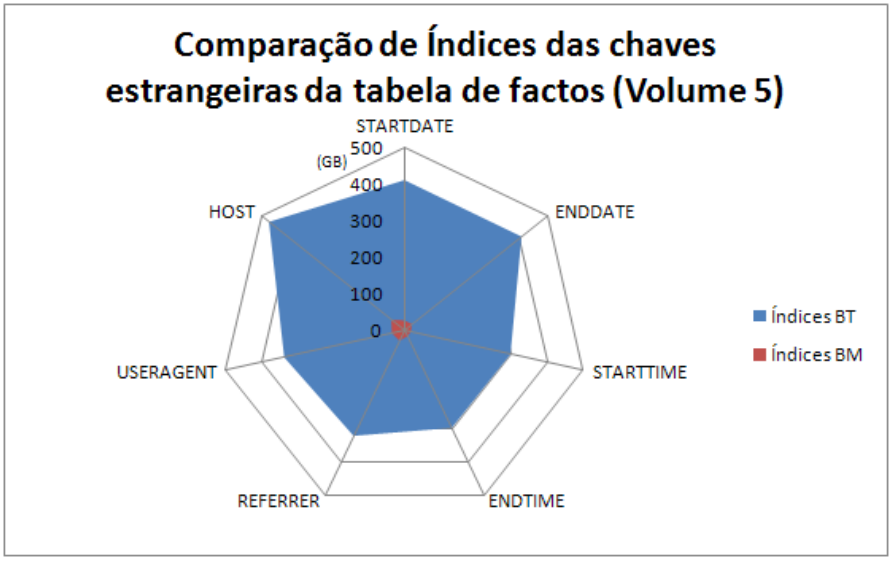


Figura 23 – Comparação do tamanho (MB) dos índices das chaves estrangeiras da tabela de factos, para o volume 5.

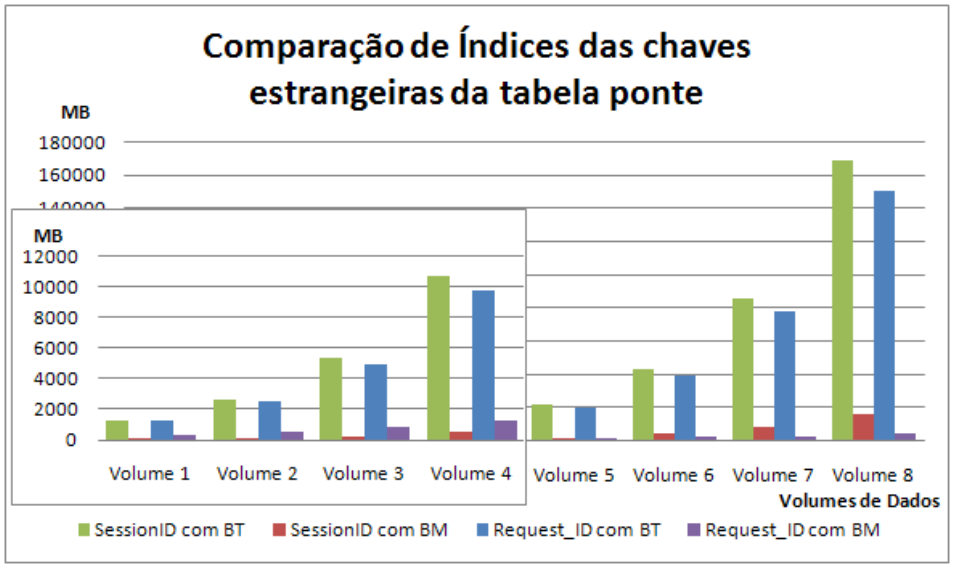


Figura 24 - Comparação do tamanho dos índices das chaves estrangeiras da tabela ponte, para todos os volumes de registos.

Com a criação de índices *B-Tree* sobre os atributos alvo referidos na secção anterior, verificou-se um crescimento de 43,07GB da BD, representando um aumento em 88,8% do tamanho base (dados mais chaves primárias). Com o uso de índices *bitmap* sobre os mesmos atributos apenas tivemos um aumento de 3,39GB da BD, representando neste caso uma redução de cerca de 92% do espaço total utilizado pelas estruturas de indexação. A tabela 12 mostra os ganhos da utilização de índices *bitmap* para os vários volumes de dados da tabela de factos (não esquecer que a tabela ponte, implicitamente, também dobrou a sua dimensão).

Volumes de dados	Dados	Chaves Primárias	Índices BT	Índices BM	Total c/BT	Total c/BM	Ganho
1	1,53 GB	1,47 GB	2,71 GB	0,36 GB	5,72 GB	3,36 GB	41,24%
2	3,05 GB	2,93 GB	5,42 GB	0,70 GB	11,40 GB	6,68 GB	41,40%
3	6,13 GB	5,94 GB	10,86 GB	1,19 GB	22,93 GB	13,26 GB	42,16%
4	12,37 GB	11,88 GB	20,68 GB	1,89 GB	44,93 GB	26,13 GB	41,84%
5	24,75 GB	23,75 GB	43,07 GB	3,39 GB	91,57 GB	51,89 GB	43,33%
6	49,42 GB	47,53 GB	86,20 GB	7,13 GB	183,14 GB	104,07 GB	43,18%
7	98,75 GB	95,19 GB	172,63 GB	11,67 GB	366,57 GB	205,61 GB	43.91%
8	194,00 GB	186,69 GB	336,32 GB	20,24 GB	717,01 GB	400,93 GB	44,08%

Tabela 12 - Comparação de espaço total com os diferentes tipos de indexação

Note-se que, com a utilização de índices *B-Tree*, o espaço necessário para as estruturas de indexação é similar ao espaço necessário para armazenar os dados e índices das respectivas chaves primárias. De salientar que, com a utilização de índices *bitmap* neste caso de estudo, consegue-se obter uma optimização média de espaço superior a 41% para os diferentes volumes de dados testados.

Tome-se como exemplo a tabela ponte com uma cardinalidade total de 576 milhões de registos (volume 4). Neste caso, os atributos sujeitos a indexação com índices *bitmap* são as chaves estrangeiras "sessionID" e "request_ID", como uma cardinalidade de 9,6 milhões e 16287 mil, respectivamente. Assim, o índice do atributo "sessionID" terá 9,6 milhões de *bitmaps* contra apenas 16287 mil do índice do atributo "request_ID".

Um índice *bitmap* criado sobre um atributo de maior cardinalidade ("sessionID") não significa que o mesmo necessita de mais espaço, que um índice *bitmap* criado sobre atributo de menor cardinalidade ("request_ID").

Q: Como é possível que o índice sobre o atributo "sessionID" ocupe menos espaço que o atributo "request_ID"?
R: A resposta a esta questão pode ser dada apenas com uma palavra, compressão. Como referido nesta dissertação, os índices bitmap são estruturas que devido às suas características podem ser bastante compressíveis. Neste caso, aquando a criação do índice, o Oracle automaticamente procede à sua compressão.

Q: Mas com uma diferença tão grande do número de bitmaps entre os dois índices, como é que o índice sobre o atributo "sessionID" consegue ser mais compressível?
R: Como cada sessão tem uma média de 60 pedidos, cada bitmap apenas terá em média 60 bits assinalados a '1', ou seja, cada bitmap é composto por 19,2 milhões de bits '0', com uma excepção dos 60 que em média se encontram a assinalados a '1'. Assim, como cada bitmap é constituído maioritariamente por bits '0', resulta em grande sequências de com bits a '0', o que permite ao Oracle efectuar uma boa compressão. O mesmo já não acontece com o atributo "request_ID", uma vez que cada pedido se encontra ligado a várias sessões, originando um maior equilíbrio na quantidade de bits '0' e '1' em cada bitmap, dificultando assim a compressão.

Em suma, a superioridade em termos de espaço necessitado pelos índices *bitmap*, deve-se em parte à grande capacidade de compressão que estes índices têm.

Considerando agora a análise do tempo de criação dos dois tipos de índices, os índices *bitmap* demonstram na maioria dos casos ser superiores aos índices *B-Tree*, como se pode observar através da figura 25.

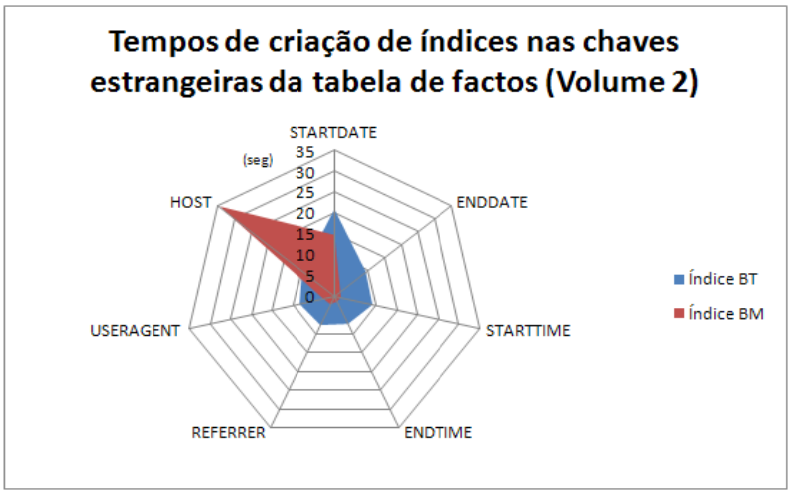


Figura 25 – Comparação do tempo de criação de índices sobre atributos do tipo *String*.

Note-se que, na figura anterior, para o atributo "host" do tipo *String*, que é chave estrangeira na tabela de factos, a criação de índices *bitmap* revelou-se ser mais demorada que a criação do índice *B-Tree*. No entanto, conforme o volume de registos na tabela de factos vai aumentando, existe uma tendência para a diminuição da diferença de tempos, chegando mesmo a inverter posições como mostra o gráfico da figura 26. Este resultado mostra que, para atributos do tipo *String*, a criação de índices *bitmap* pode nem sempre ser mais rápida em comparação com os índices *B-Tree*, podendo mesmo para o caso de atributos do tipo *String* de elevada cardinalidade se tornar uma operação com maior custo. No entanto, no âmbito do espaço utilizado demonstra uma superioridade bem evidente, como é revelado pelo gráfico da figura 23 para o atributo "host".

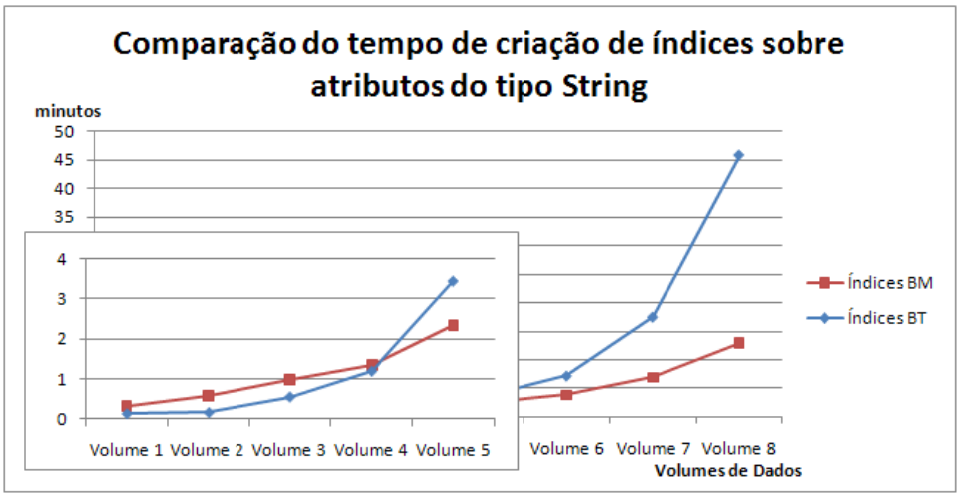


Figura 26 – Comparação do tempo de criação de índices sobre atributos do tipo *String*.

Feita a análise do tempo e do tamanho requisitados pelos índices *bitmap* face aos índices *B-Tree*, para este caso de estudo concreto e até ao fim desta fase de análise, os índices *bitmap* têm-se revelado bastantes superiores aos índices *B-Tree*. Assim, importa agora analisar o seu desempenho em termos do tempo que proporcionam aquando da realização de *queries* sobre o *data webhouse*. Devido ao elevado tempo que todo este processo de testes (e, em particular, de algumas das *queries*) foi decidido testar todas as *queries* do conjunto de testes apenas sobre os três primeiros volumes de dados. Importa referir que, para todos os testes foram geradas todas as estatísticas para os vários tipos de indexação, assim como para todas as tabelas do *data webhouse*. Complementarmente, foram limpas as caches antes da execução de cada *query* de forma a se obter resultados com a máxima fiabilidade possível.

Numa abordagem global sobre o tempo que o conjunto de *queries* demorou a efectuar, uma vez mais, os índices *bitmap* para a maioria dos casos demonstram ser superiores aos índices *B-Tree*. Como mostram as figura 27, 28 e 29, este facto evidencia-se mais claramente à medida que o volume de dados vai aumentando. A mesma figura mostra claramente que os índices *bitmap* são maioritariamente superiores aos índices *B-Tree* para vários volumes de dados. Contudo, no caso das *queries* de categoria 2 e 3, verifica-se uma ligeira desvantagem por parte dos índices *bitmap* no volume de dados 3. Aliás, estas duas categorias de *queries* demonstraram ao longo dos vários testes serem problemáticas a executar - demoram mais tempo do que seria de esperar.

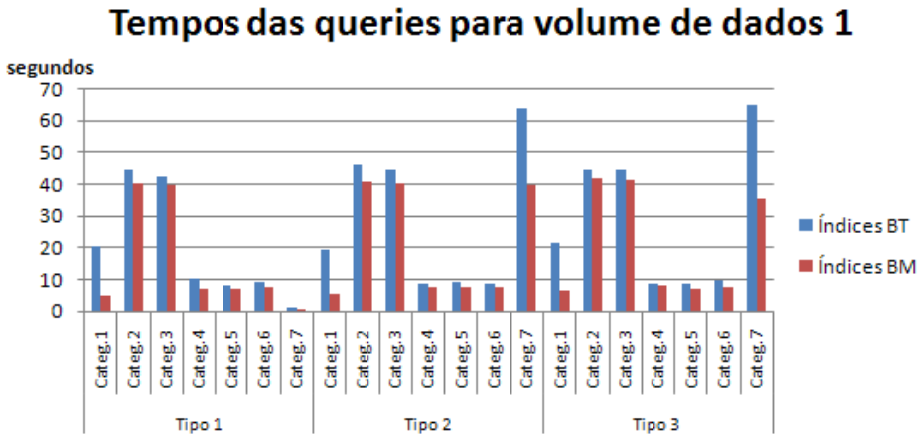


Figura 27 - Tempo das *queries* de teste para volume de dados 1

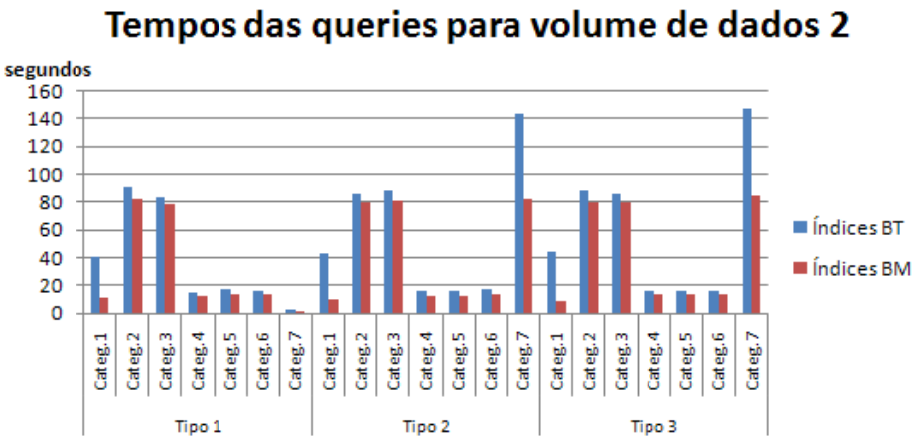


Figura 28 - Tempo das *queries* de teste para volume de dados 2

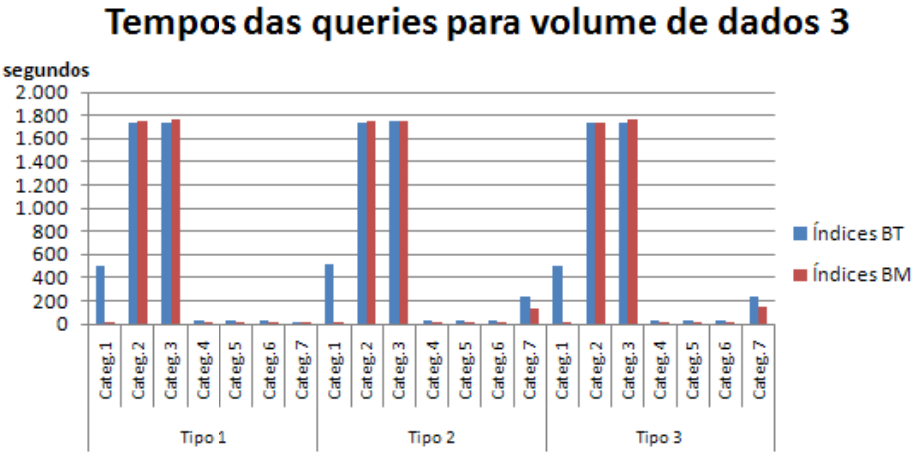


Figura 29 – Tempo das *queries* de teste para volume de dados 3

Abaixo são apresentadas as *queries* para estas categorias do tipo 3. Note-se que as *queries* têm ligações duplas às dimensões que envolvem, o que se crê que faça com que o Oracle opte por um mau plano de execução, provocando assim tempos mais longos de execução.

Categoria 2:

```
SELECT DT.MES_NR, DT.DIA_MES_NR,
       COUNT(*) AS TOTAL_SESSÕES, SUM(PEDIDOS) AS PEDIDOS , SUM(M_GETS) AS M_GETS,
       SUM(M_POST) AS M_POST, SUM(M_HEAD) AS M_HEAD, SUM(M_OUTROS) AS M_OUTROS,
       SUM(STATUS_INFORMACOES) AS STATUS_INFORMACOES, SUM(STATUS_SUCESSO) AS
       STATUS_SUCESSO, SUM(STATUS_REDIRECIONADO) AS STATUS_REDIRECIONADO,
       SUM(STATUS_ERRO_CLIENTE) AS STATUS_ERRO_CLIENTE, SUM(STATUS_ERRO_SERVIDOR) AS
       STATUS_ERRO_SERVIDOR, SUM(BYTES_ENVIADOS) AS BYTES_ENVIADOS,
```

```
        SUM(BYTES_ENVIADOS)/1024 AS KB_ENVIADOS, SUM(BYTES_ENVIADOS)/1024/1024 AS
        MB_ENVIADOS, SUM(SESSION_DURATION) AS DURACAO
FROM TF_SESSIONS TF, DATA_DIM DT
WHERE TF.STARTDATE = DT.DATA_ID
        AND TF.ENDDATE = DT.DATA_ID
        AND DT.ANO=2008
GROUP BY CUBE (DT.MES_NR, DT.DIA_MES_NR)
ORDER BY DT.MES_NR, DT.DIA_MES_NR
```

Categoria 3:

```
SELECT DT.MES.NR, DT.DIA_MES_NR, TMP.PERIODO,
        COUNT(*) AS TOTAL_SESSÕES, SUM(PEDIDOS) AS PEDIDOS , SUM(M_GETS) AS M_GETS,
        SUM(M_POST) AS M_POST, SUM(M_HEAD) AS M_HEAD, SUM(M_OUTROS) AS M_OUTROS,
        SUM(STATUS_INFORMACOES) AS STATUS_INFORMACOES, SUM(STATUS_SUCESSO) AS
        STATUS_SUCESSO, SUM(STATUS_REDIRECIONADO) AS STATUS_REDIRECIONADO,
        SUM(STATUS_ERRO_CLIENTE) AS STATUS_ERRO_CLIENTE, SUM(STATUS_ERRO_SERVIDOR) AS
        STATUS_ERRO_SERVIDOR, SUM(BYTES_ENVIADOS) AS BYTES_ENVIADOS,
        SUM(BYTES_ENVIADOS)/1024 AS KB_ENVIADOS, SUM(BYTES_ENVIADOS)/1024/1024 AS
        MB_ENVIADOS, SUM(SESSION_DURATION) AS DURACAO
FROM TF_SESSIONS TF, DATA_DIM DT, TEMPO_DIM TMP
WHERE TF.STARTDATE = DT.DATA_ID
        AND TF.ENDDATE = DT.DATA_ID
        AND TF.STARTTIME = TMP.HORA
        AND TF.ENDTIME = TMP.HORA
        AND DT.ANO=2008
GROUP BY CUBE (DT.MES_NR, DT.DIA_MES_NR, TMP.PERIODO)
ORDER BY DT.MES_NR, DT.DIA_MES_NR, TMP.PERIODO
```

Todas as *queries* com categoria superior a 3 também contêm estas mesmas ligações, às dimensões Tempo e Data. No entanto, nesses casos, existem outros atributos que restringem o resultado final, fazendo com que o Oracle a siga um plano de execução mais eficiente, como evidenciam os resultados.

> "*The real power of the bitmap index is seen when a table contains multiple bitmap indexes*." [Niemiec, 2007]

Tendo em consideração a citação anterior, vejamos então o comportamento da *query* de categoria 1, do tipo 2. Como se pode ver em baixo, esta *query* apenas utiliza a tabela de factos com restrições sobre os atributos "Session_Duration", "Pedidos" e "Bytes_Enviados".

```
SELECT SESSIONID,
       COUNT(*) AS TOTAL_SESSÕES, SUM(PEDIDOS) AS PEDIDOS , SUM(M_GETS) AS M_GETS,
       SUM(M_POST) AS M_POST, SUM(M_HEAD) AS M_HEAD, SUM(M_OUTROS) AS M_OUTROS,
       SUM(STATUS_INFORMACOES) AS STATUS_INFORMACOES, SUM(STATUS_SUCESSO) AS
       STATUS_SUCESSO, SUM(STATUS_REDIRECIONADO) AS STATUS_REDIRECIONADO,
       SUM(STATUS_ERRO_CLIENTE) AS STATUS_ERRO_CLIENTE, SUM(STATUS_ERRO_SERVIDOR) AS
       STATUS_ERRO_SERVIDOR, SUM(BYTES_ENVIADOS) AS BYTES_ENVIADOS,
       SUM(BYTES_ENVIADOS)/1024 AS KB_ENVIADOS, SUM(BYTES_ENVIADOS)/1024/1024 AS
       MB_ENVIADOS, SUM(SESSION_DURATION) AS DURACAO
FROM TF_SESSIONS
WHERE SESSION_DURATION BETWEEN 450 AND 1350
       AND PEDIDOS BETWEEN 20 AND 80
       AND BYTES_ENVIADOS <=1000
GROUP BY ROLLUP (SESSIONID)
ORDER BY PEDIDOS DESC
```

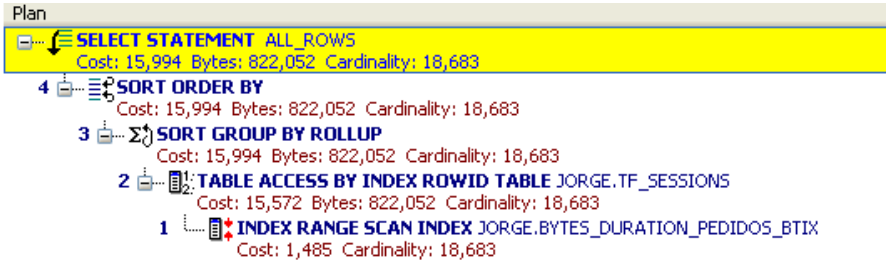


Figura 30 - Plano de execução da *query* do tipo 2 categoria 1 com índices *B-Tree*.

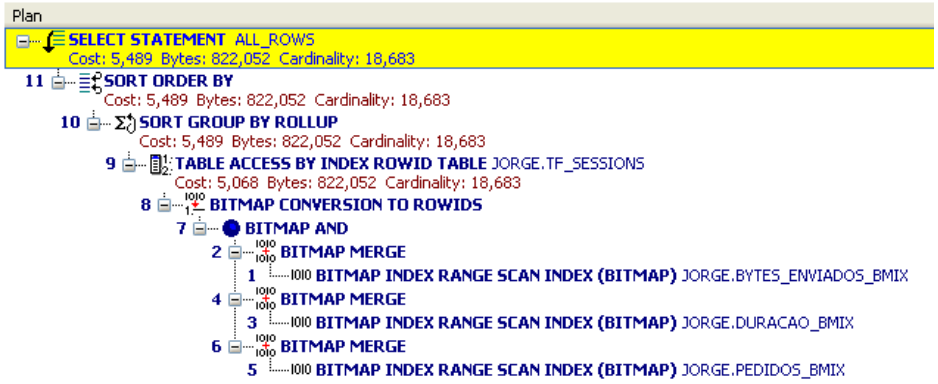


Figura 31 - Plano de execução da *query* do tipo 2 categoria 1 com índices *bitmap*.

As figuras 30 e 31 mostram os planos de execução da *query* anterior para os dois tipos de indexação, índices *B-Tree* e índices *bitmap*, respectivamente. Ao analisar ambos os planos de execução verifica-se que com o uso de índices *B-Tree*, o Oracle utiliza o índice

("Bytes_Duration_Pedidos_BTIX") composto pelos três atributos restritivos da *query*. No caso dos índices *bitmap*, verifica-se que os índices individuais sobre os atributos restritivos são utilizados. Repare-se como o Oracle inicialmente começa por aceder aos índices, efectuando uma junção entre os vários *bitmaps* do índice de cada atributo correspondentes aos valores restritivos e só depois de fazer um "e" lógico (AND) entre os *bitmaps* é que procede ao acesso dos dados da tabela com base no resultado da operação lógica entre os 3 índices *bitmaps* resultantes das junções. Este é um exemplo que demonstra o verdadeiro potencial dos índices *bitmap*, obtendo-se neste caso uma optimização do tempo na ordem dos 97,5% em relação aos índices *B-Tree*, para o caso do volume de dados mais elevado para o qual a *query* foi testada (tabela 13).

Volume	Tempo com índices BT	Tempo com índices bitmap	Ganho
1	19,5 s	5,1 s	73,8%
2	42,8 s	9,4 s	78,0%
3	508,9 s	12,7 s	97,5%

Tabela 13 - Ganhos no tempo da *query* do tipo 2 categoria 1, entre índices *B-Tree* e índices *bitmap.*

No caso particular do Oracle é reconhecida a eficiência e vantagens que os índices *bitmap* proporcionam. Isto porque, o Oracle internamente converte os índices *B-Tree* para poder realizar operações lógicas entre os vários índices. Vejamos então este comportamento, considerando como exemplo a *query* do tipo 1 de categoria 4 que está apresentada de seguida.

```
SELECT DT.MES_NR, DT.DIA_MES_NR, TMP.PERIODO,
       COUNT(*) AS TOTAL_SESSÕES, SUM(PEDIDOS) AS PEDIDOS , SUM(M_GETS) AS M_GETS,
       SUM(M_POST) AS M_POST, SUM(M_HEAD) AS M_HEAD, SUM(M_OUTROS) AS M_OUTROS,
       SUM(STATUS_INFORMACOES) AS STATUS_INFORMACOES, SUM(STATUS_SUCESSO) AS
       STATUS_SUCESSO, SUM(STATUS_REDIRECIONADO) AS STATUS_REDIRECIONADO,
       SUM(STATUS_ERRO_CLIENTE) AS STATUS_ERRO_CLIENTE, SUM(STATUS_ERRO_SERVIDOR) AS
       STATUS_ERRO_SERVIDOR, SUM(BYTES_ENVIADOS) AS BYTES_ENVIADOS,
       SUM(BYTES_ENVIADOS)/1024 AS KB_ENVIADOS, SUM(BYTES_ENVIADOS)/1024/1024 AS
       MB_ENVIADOS, SUM(SESSION_DURATION) AS DURACAO
FROM TF_SESSIONS TF, DATA_DIM DT, TEMPO_DIM TMP, AGENTE_DIM AG
WHERE TF.STARTDATE = DT.DATA_ID
       AND TF.ENDDATE = DT.DATA_ID
       AND TF.STARTTIME = TMP.HORA
       AND TF.ENDTIME = TMP.HORA
       AND TF.USERAGENT = AG.AGENTE_ID
```

```
        AND DT.ANO=2008
        AND AG.TIPO = 'Crawler'
GROUP BY DT.MES_NR, DT.DIA_MES_NR, TMP.PERIODO
ORDER BY DT.MES_NR, DT.DIA_MES_NR, TMP.PERIODO
```

Repare-se nos passos 8 e 14 do plano de execução da query anterior apresentados na figura 32. Aqui, verifica-se que o Oracle converte internamente o índice *B-Tree* num índice *bitmap* para posteriormente fazer um conjunto de operações lógicas sobre os mesmos, como se pode verificar nos passos 9, 10, 15, 16, 17 e 18. Analisemos agora o comportamento da mesma query utilizando índices *bitmap* (figura 33). Note-se que apesar da tabela "Data_dim" ser acedida totalmente duas vezes com uso de índices *bitmap*, isso não afecta a performance da *query*. De salientar ainda o custo de cada plano de execução, verificando-se que o plano de execução dos índices *bitmap* tem um custo mais reduzido, na ordem dos 65%.

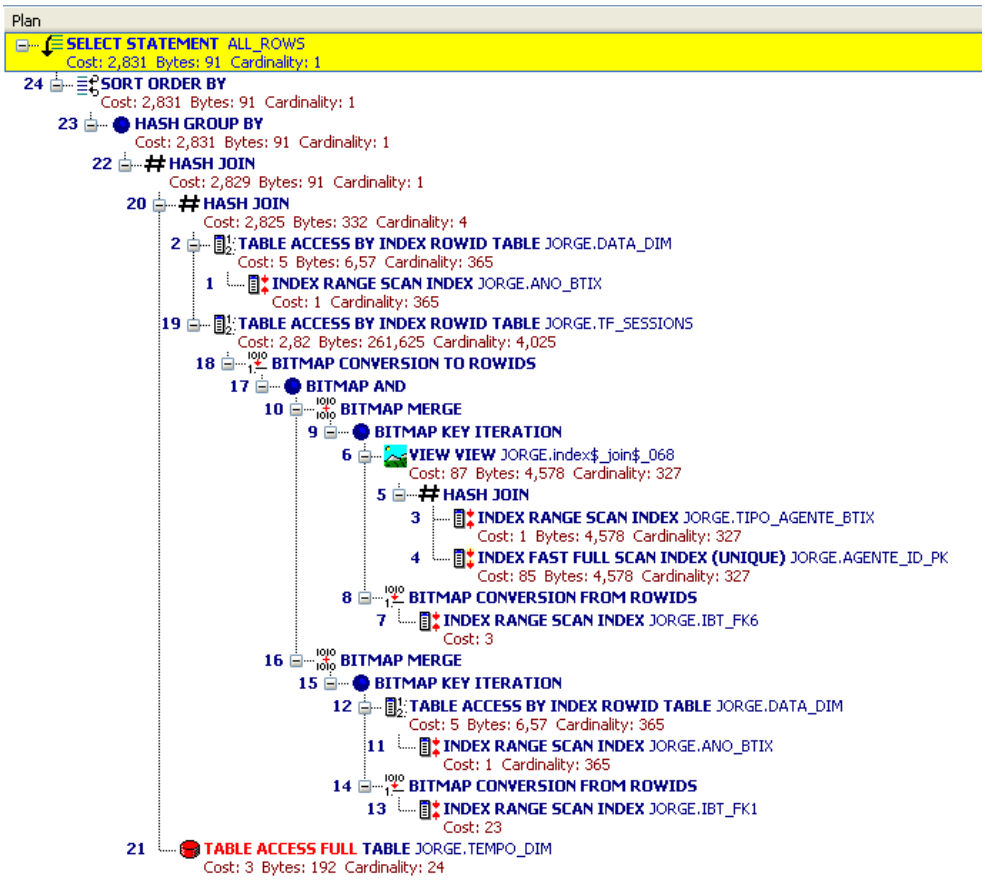


Figura 32 – Plano de execução da *query* do tipo1 categoria 4 com índices *B-Tree*.

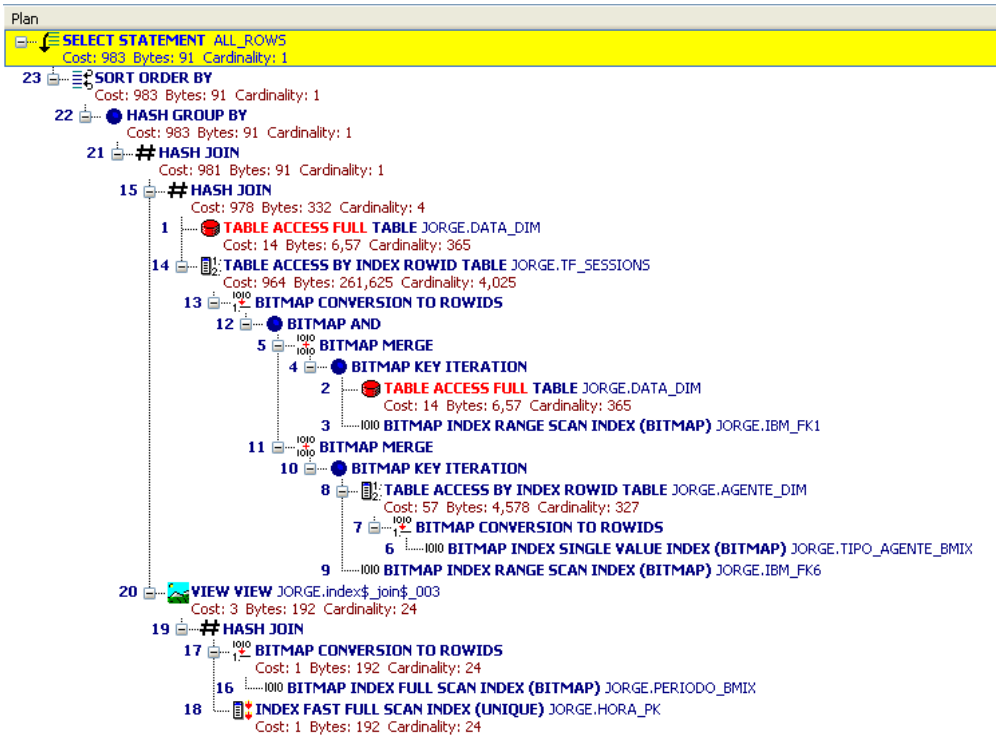


Figura 33 - Plano de execução da *query* do tipo1 categoria 4 com índices *bitmap*.

Por fim, para uma análise mais rigorosa foram efectuados mais alguns testes com volumes de dados mais elevados, contudo foi optado apenas por testar a query do tipo 1 categoria 7 sobre esses volumes. Esta *query* envolve todas as tabelas do *data webhouse*, sendo por isso a *query* que à partida estará mais susceptível a penalizações no tempo de execução, devido ao aumento sucessivo do volume de dados – de seguida apresenta-se a *query*.

```
SELECT
    PA.PAIS, DT.MES_NR, RF.TIPO as TIPO_REFERRER, RQ.TIPO as TIPO_REQUEST, TMP.PERIODO,
    COUNT(*) AS TOTAL_SESSÕES, SUM(PEDIDOS) AS PEDIDOS , SUM(M_GETS) AS M_GETS, SUM(M_POST)
    AS M_POST, SUM(M_HEAD) AS M_HEAD, SUM(M_OUTROS) AS M_OUTROS, SUM(STATUS_INFORMACOES) AS
    STATUS_INFORMACOES, SUM(STATUS_SUCESSO) AS STATUS_SUCESSO, SUM(STATUS_REDIRECIONADO) AS
    STATUS_REDIRECIONADO, SUM(STATUS_ERRO_CLIENTE) AS STATUS_ERRO_CLIENTE,
    SUM(STATUS_ERRO_SERVIDOR) AS STATUS_ERRO_SERVIDOR, SUM(BYTES_ENVIADOS) AS
    BYTES_ENVIADOS, SUM(BYTES_ENVIADOS)/1024 AS KB_ENVIADOS, SUM(BYTES_ENVIADOS)/1024/1024
    AS MB_ENVIADOS, SUM(SESSION_DURATION) AS DURACAO
    FROM TF_SESSIONS TF, DATA_DIM DT, TEMPO_DIM TMP, REFERRER_DIM RF, AGENTE_DIM AG,
    COMPUTADORUTILIZADOR_DIM CU, PAIS_SUBDIM PA, TP_SESSION TP, REQUEST_DIM RQ
    WHERE TF.STARTDATE = DT.DATA_ID
            AND TF.ENDDATE = DT.DATA_ID
            AND TF.STARTTIME = TMP.HORA
```

```
        AND TF.ENDTIME = TMP.HORA
        AND TF.USERAGENT = AG.AGENTE_ID
        AND TF.HOST = CU.IPADDRESS
        AND CU.CODIGOISOPAIS = PA.CODIGOISOPAIS
        AND TP.SESSIONID = TF.SESSIONID
        AND TP.REQUEST_ID = RQ.REQUEST_ID
        AND DT.ANO=2008
        AND AG.TIPO IN ('User','Crawler')
        AND RQ.TIPO_CLASSE = 'G48'
GROUP BY PA.PAIS, DT.MES_NR, RF.TIPO, RQ.TIPO, TMP.PERIODO
ORDER BY PA.PAIS, DT.MES_NR, RF.TIPO, RQ.TIPO, TMP.PERIODO
```

Com os sucessivos aumentos do volume de dados, mais propriamente com as sucessivas duplicações do tamanho da tabela de factos (que implicitamente implica também a duplicação da tabela ponte), verifica-se que a distância entre as linhas de desempenho entre os dois tipos de indexação vai aumentando (figura 34). Estes resultados sugerem que os índices *bitmap* proporcionam um melhor desempenho, mesmo para volumes de dados mais elevados.

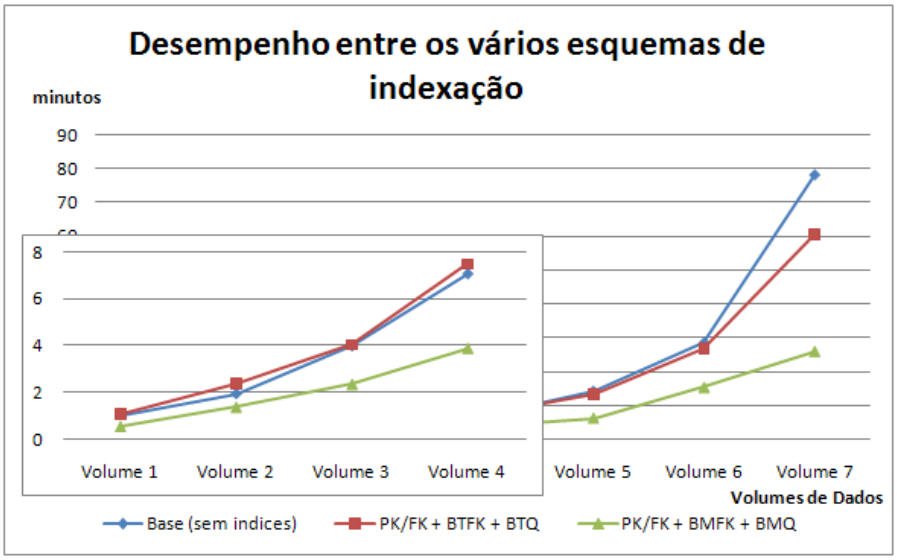


Figura 34 - Gráfico de desempenho entre os vários esquemas de indexação para os diferentes volumes de dados da tabela de factos.

No entanto, note-se que até determinados volumes de dados, verifica-se que a *query* sem a utilização de quaisquer tipo de índices apresenta, para a maioria dos casos, um melhor desempenho do que com a utilização de índices *B-Tree*. A partir do volume de dados 5 (19,2 milhões na tabela de factos), os resultados indiciam que os índices *B-Tree* começam a fazer sentir

algum impacto sobre as *queries*. Estes resultados podem significar que os índices *B-Tree* só para volumes de dados mais elevados demonstram uma melhor eficiência.

> *Q: Como é possível isto acontecer? Como é que uma query deste tipo executa mais rapidamente sem índices do que com o uso de índices B-Tree?*

O Oracle é um SGBD que é dotado de muitas variáveis de configuração que podem influenciar o seu desempenho, originando este tipo de comportamento. No entanto, nenhuma informação foi encontrada nesse sentido. Após algumas indicações online, foi sugerido efectuar um plano de execução mais detalhado para a *query* sem índices (anexo 1), para a *query* com índices *B-Tree* (anexo 2) e para a *query* com índices *bitmap* (anexo 3) para verificar realmente a quantidade de linhas que estavam a ser utilizadas nos vários passos do plano de execução da *query*. Após a análise desses planos, verifica-se uma grande diferença entre o número de linhas que são estimadas (*E-Rows*) e as que realmente são utilizadas (*A-Rows*). No entanto, essa situação acontece nos três planos de execução. Contudo, no caso do plano da *query* com índices *B-Tree*, verifica-se que é utilizado um número bem mais elevado de linhas em algumas das operações, comparativamente os outros dois planos, sendo algumas delas nos acessos às estruturas de indexação, o que ajuda a justificar estes resultados. Note-se que os planos de execução mostram que o Oracle, com índices *B-Tree*, tem de aceder e processar em vários passos dos planos de execução mais registos do que com o uso de índices *bitmap*, e mesmo até sem o uso de qualquer índice. De uma forma global, o Oracle com o uso de índices *B-Tree* tem de aceder e processar mais linhas de informação do que sem qualquer índice.

Em suma, os resultados dos vários testes demonstram claramente que para pequenos e médios *Data Warehouses* com atributos de baixa e média cardinalidade, os índices *bitmap* são uma estrutura de indexação a ter em conta, proporcionando não só melhores tempos de execução das *queries*, como também grandes optimizações de espaço nas estruturas de indexação. Para além destes factores, os testes demonstraram claramente que os índices *bitmap* são criados mais rapidamente que os índices *B-Tree* sobre atributos do tipo inteiro. Como tipicamente todas as chaves estrangeiras da tabela de factos são constituídas por inteiros, atribuídas no processo de substituição de chaves aquando do processo de ETL, o uso de índices *bitmap* tem toda a validade.

Capítulo 5

Introdução de bitmap Joins em Queries Multidimensionais

Como já referido no capítulo anterior, independentemente da cardinalidade dos atributos, os índices *bitmap* são uma opção de indexação a ter sempre em consideração no contexto de um DW. Nos testes efectuados sobre o caso de estudo seleccionado, os índices *bitmap* demonstraram ser mais eficientes que os índices *B-Trees*. Nunca esquecer no entanto que estes resultados não podem ser assumidos como válidos para todos os *Data Warehouses* uma vez que cada DW tem as suas especificidades. Uma variante de índices *bitmap* denominada de *bitmap join* é também disponibilizada pelo Oracle como um mecanismo de indexação. Este tipo de índice consiste na representação da junção de uma ou mais tabelas, que no caso de um DW envolve a junção de uma tabela de factos com uma ou mais dimensões. Desta forma, espera-se que este tipo de estrutura de indexação possa contribuir para uma mais rápida e eficiente satisfação de uma *query*. De uma forma muito simplista, esta estrutura de indexação corresponde a um conjunto de *rowids* armazenados, que são determinados numa pré-computação da junção entre as tabelas envolvidas aquando da criação do índice [Niemiec, 2007]. No entanto, este tipo de estrutura tem um conjunto de condicionantes à sua utilização, nomeadamente [Bryla & Loney, 2008, WWW14]:

- Nenhuma tabela pode ser utilizada mais que uma vez numa junção.
- As colunas utilizadas na junção têm de ser obrigatoriamente chaves primárias ou ter uma restrição de unicidade.

- Apenas uma das tabelas utilizadas nesta forma de indexação pode ser actualizada concorrentemente por diferentes transacções quando o índice *bitmap join* está a ser utilizado.
- Os índices *bitmap join* não podem ser criados sobre uma tabela temporária ou sobre uma tabela organizada sobre um índice (do inglês, *Index-Organized Table*).
- Um índice *bitmap join* apenas é utilizado por *queries* que tenham as mesmas condições sobre a clausula *where*.
- Todas as colunas do índice têm de pertencer unicamente às dimensões.
- Se uma dimensão tiver uma chave primária composta, todos os atributos que compõem a mesma tem de fazer parte da junção.

Considere-se, por exemplo, a *query* do tipo 1 de categoria 4 a seguir apresentada. Esta *query* envolve uma junção entre 4 tabelas: a tabela de factos "TF_Sessions" e as dimensões "Data_dim", "Tempo_dim" e "Agente_dim" - repare-se nas condições de restrição existentes sobre a mesma, nomeadamente, sobre o atributo "ANO" da dimensão "Data_dim" e o atributo "TIPO" da dimensão "Agente_dim".

```
SELECT
        DT.MES_NR, DT.DIA_MES_NR, TMP.PERIODO,
        COUNT(*) AS TOTAL_SESSÕES, SUM(PEDIDOS) AS PEDIDOS , SUM(M_GETS) AS M_GETS,
        SUM(M_POST) AS M_POST, SUM(M_HEAD) AS M_HEAD, SUM(M_OUTROS) AS M_OUTROS,
        SUM(STATUS_INFORMACOES) AS STATUS_INFORMACOES, SUM(STATUS_SUCESSO) AS
        STATUS_SUCESSO, SUM(STATUS_REDIRECIONADO) AS STATUS_REDIRECIONADO,
        SUM(STATUS_ERRO_CLIENTE) AS STATUS_ERRO_CLIENTE, SUM(STATUS_ERRO_SERVIDOR) AS
        STATUS_ERRO_SERVIDOR, SUM(BYTES_ENVIADOS) AS BYTES_ENVIADOS,
        SUM(BYTES_ENVIADOS)/1024 AS KB_ENVIADOS, SUM(BYTES_ENVIADOS)/1024/1024 AS
        MB_ENVIADOS, SUM(SESSION_DURATION) AS DURACAO
FROM TF_SESSIONS TF, DATA_DIM DT, TEMPO_DIM TMP, AGENTE_DIM AG
WHERE TF.STARTDATE = DT.DATA_ID
        AND TF.ENDDATE = DT.DATA_ID
        AND TF.STARTTIME = TMP.HORA
        AND TF.ENDTIME = TMP.HORA
        AND TF.USERAGENT = AG.AGENTE_ID
        AND DT.ANO=2008
        AND AG.TIPO = 'Crawler'
GROUP BY DT.MES_NR, DT.DIA_MES_NR, TMP.PERIODO
ORDER BY DT.MES_NR, DT.DIA_MES_NR, TMP.PERIODO
```

Recorrendo aos índices *bitmap join* é possível criar uma estrutura de indexação capaz de dar uma resposta eficaz a este tipo de *queries*, independentemente dos valores de restrição aplicados. Neste caso, o índice é criado de uma forma um pouco incomum – de seguida apresenta-se a instrução necessária para a sua criação.

```
CREATE BITMAP INDEX BMJ_C4
       ON TF_SESSIONS (DT.ANO, AG.TIPO)
FROM TF_SESSIONS TF, DATA_DIM DT, TEMPO_DIM TMP, AGENTE_DIM AG
WHERE TF.STARTDATE = DT.DATA_ID
       AND TF.STARTIME = TMP.HORA
       AND TF.USERAGENT = AG.AGENTE_ID;
```

Desta forma, o índice *bitmap join* é criado sempre sobre a tabela na qual se encontram as chaves estrangeiras envolvidas na junção, ou seja, sobre a tabela de factos "TF_Sessions". Note-se que, neste caso em particular, a *query* anterior é constituída por uma ligação dupla às dimensões "Tempo_dim" e "Data_dim". Tal não seria possível de fazer na instrução de criação do índice *bitmap join*. No entanto, apesar de ir contra as restrições anteriormente referidas, o Oracle consegue utilizar o índice em causa, mostrando neste caso um melhor desempenho que os índices *B-Tree* e os índices *bitmap*, como mostra o gráfico da figura 35.

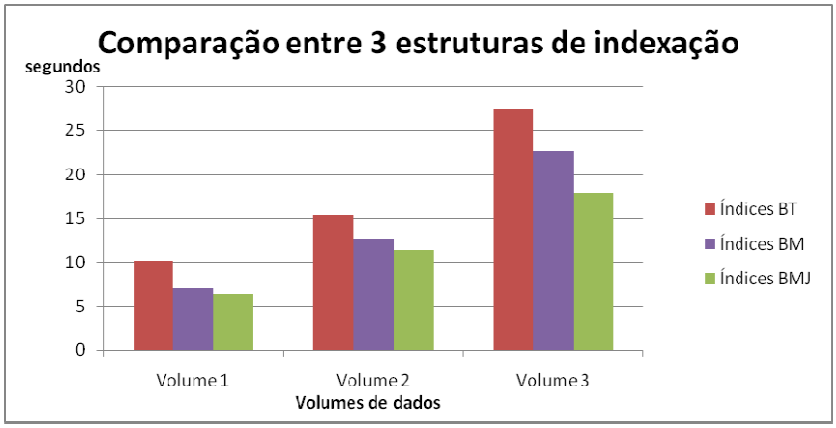


Figura 35 – Comparação dos tempos de para a *query* do tipo 1 de categoria 4 segundo três tipo de indexação, índices *B-Tree*, índices *bitmap* e índices *bitmap join*.

Bryla & Loney (2008) referem que um índice *bitmap join* pode ser visto como uma alternativa às vistas materializadas, uma vez que consegue dar resposta adequada para o mesmo tipo de *queries*. Contudo, os índices *bitmap join* tem a particularidade de tipicamente necessitarem de

menos espaço para armazenamento, já que armazenamento de um conjunto de *rowids* pode em muitos casos, senão mesmo na maioria deles, ser significativamente mais pequeno do que toda a informação de uma vista materializada.

5.1 Análise de resultados

Com o uso de índices *bitmap join*, verifica-se que a performance para grande parte das *queries* se mantém. No entanto, as características próprias deste *data webhouse* obrigam a ter algumas considerações presentes. Como referido na análise do sistema de testes alvo, este sistema de dados tem uma tabela ponte "TP_Session" entre a tabela de factos "TF_Sessions" e a dimensão "Request_dim". Isto obriga a que aquando da criação de um índice *bitmap join* que evolva a tabela ponte, este seja criado sobre a tabela ponte e não sobre a tabela de factos como acima referido. De uma forma simplista, um índice *bitmap join* tem de ser sempre criado sobre a tabela que contém o lado "N" de uma relação.

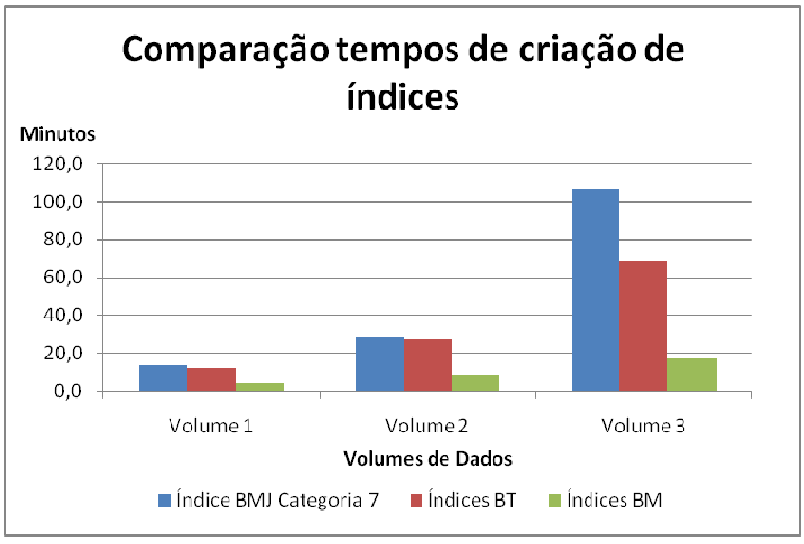


Figura 36 – Comparação dos tempos de criação do índice *bitmap join* para *queries* de categoria 7, com o tempo total de criação de todos os índices *B-Tree* e *bitmap*.

Como referido anteriormente, para alguns casos, este tipo de indexação mostra-se bastante eficiente, contudo é um índice que apenas serve para optimizar um certo grupo de *queries*, bastante específicas. Como neste sistema de testes, as *queries* de categoria 7 são as que mais se encontram sujeitas ao impacto da variação de volumes de dados, foi dada particular atenção a

esta *query*. Os resultados com índices *bitmap join* continuam a demonstrar superioridade face aos índices *B-Tree*, no entanto verifica-se que conforme o volume de dados vai aumentando, o seu tempo de computação aquando da sua criação começa a revelar-se demasiado penoso (figura 36). Como se pode ver pelo gráfico anterior, apesar da *query* se tornar mais eficiente, o tempo necessário à criação deste índice é bastante penalizador, tendo em conta que o mesmo apenas é utilizado para esta *query* (independentemente do tipo de valores dos atributos restritivos). Ou seja, o tempo de criação deste índice pode ser considerado como um tempo adicional, uma vez que todos os outros índices normais terão de ser criados de forma a dar resposta a todas às outras *queries*. Tomando como exemplo o volume de dados 3, verifica-se um aumento no tempo total de criação de índices na ordem dos 155% e 615% face aos índices *B-Tree* e índices *bitmap*, respectivamente.

Uma vez que todos os outros índices terão de ser criados, significa que se terá um misto de estruturas de indexação, o que neste caso resulta numa degradação significativa da performance das *queries* de categoria 7 em causa (figura 37).

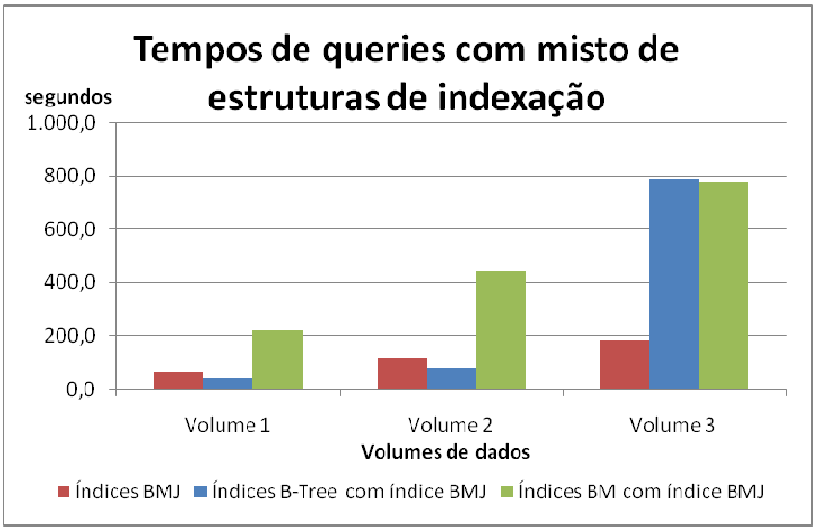


Figura 37 – Comparação dos tempos da *query* do tipo 1 categoria 7 segundo a utilização mista de estruturas de indexação.

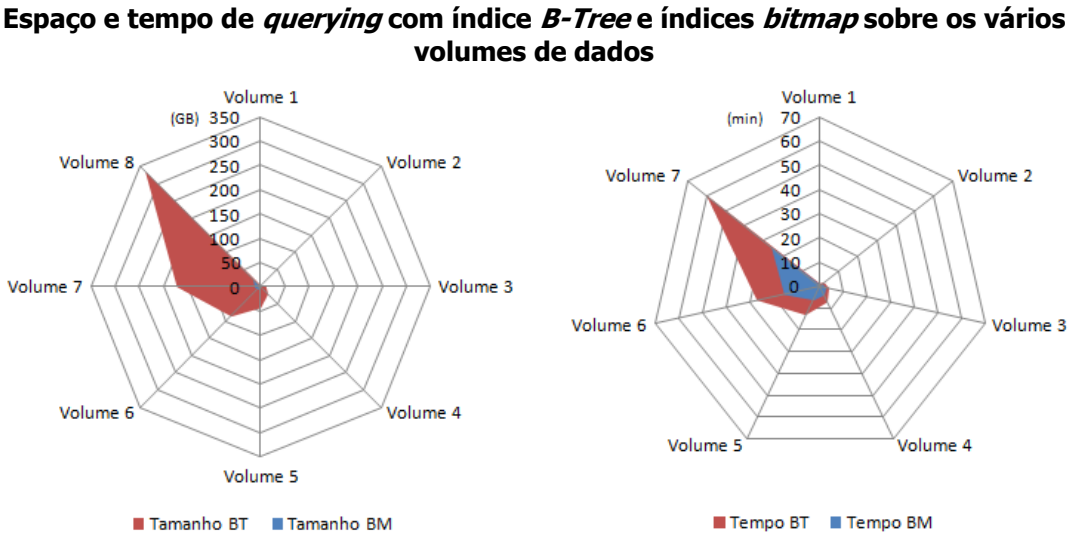
Em suma, apesar dos índices *bitmap join* se revelarem mais eficientes para uma boa parte do tipo de *queries*, estes podem não ser a melhor escolha para utilização em *Data Warehouses* de grandes dimensões. Os resultados demonstram que para volumes de dados mais elevados e para *queries* que envolvam demasiadas tabelas, o tempo de processamento da sua criação é bastante elevado.

Posteriormente, ainda podem surgir problemas na utilização destas estruturas de indexação em conjunto com as estruturas de indexação globais.

Capítulo 6

Conclusões e Trabalho Futuro

Nesta dissertação foram apresentadas várias soluções e estudos existentes na literatura de *Data Warehousing* para a optimização dos índices *bitmap*. Contudo, uma boa parte deles não se encontra implementada nos SGBD actuais, uma vez que cada SGBD aparenta ter a sua própria implementação dos índices *bitmap*. Desta forma, foi demonstrado para um dos SGBD líderes de mercado que os índices *bitmap* são uma opção viável para pequenos e médios *data warehouses*.



a) b)

Figura 38 – Espaço a) e tempo de *querying* b) utilizando pelas respectivas estrutura de indexação sobre os vários volumes de dados.

De salientar que a análise efectuada nesta dissertação centrou-se no impacto dos índices *bitmap* na fase de *querying*, sendo para tal analisados os tempo de criação dos índices, o espaço requisitado e o tempo de *querying* ao os utilizar como suporte. Assim, os resultados obtidos aquando da realização prática desta dissertação mostram que, de uma forma global, a superioridade dos índices *bitmap* sobre os índices *B-Tree*, os índices por omissão do Oracle e os utilizados nesta dissertação como base de comparação.

Os índices *bitmap* demonstraram ser claramente mais eficientes na quantidade de espaço utilizado (figura 38 a)), permitindo obter ganhos muito significativos. Nos testes realizados obtiveram-se ganhos na ordem dos 43% no espaço global da BD e de 91% no espaço global utilizado pelas estruturas de indexação. Independentemente do tipo de dados de cada atributo, os índices *bitmap* sempre demonstraram uma superioridade significativa, face aos índices de omissão. Aquando da sua criação, uma vez mais se verificou a sua superioridade sobre os índices *B-Tree*, mais propriamente sobre atributos do tipo "inteiro" e "data". No entanto, para o caso de atributos do tipo *String*, os índices *bitmap* demonstraram ser mais lentos na criação para determinados volumes de dados. Os resultados sugerem que para atributos de elevada cardinalidade do tipo *String*, a criação de índices *bitmap* podem se tornar penalizadores. Este factor poderá ser explicado pelo tempo dispendido na comparação de palavras para determinar o *bitmap* correspondente aquando da criação do índice. Segundo a análise dos tempos de *querying* (figura 38 b)), uma vez mais os índices *bitmap* se mostraram superiores aos índices *B-Tree*, demonstrando de uma forma generalizada a sua superioridade nos vários tipos de *queries* - *queries* com restrições de igualdade (*equality queries*) e intervalo (*range queries*). De salientar ainda a eficiência destes índices nas operações de junção, em que se verificou que com o uso deste tipo de índices, em muitos casos, o Oracle optou por um plano de execução mais eficiente, originando em alguns casos, ganhos na ordem dos 50%.

No caso dos índices *bitmap join*, estes demonstraram ser eficientes para alguns dos testes realizados. Apesar deste tipo de indexação demonstrar em alguns casos ser superior aos índices *bitmap* normais, estes podem não ser a melhor escolha nos casos em que envolve várias tabelas de grandes dimensões, uma vez que o tempo de processamento aquando da sua criação para alguns casos é demasiado elevado, chegando em alguns casos a ser superior ao tempo de criação de todas as estruturas de indexação do DW. Este tipo de índice apenas serve um determinado tipo de *queries* não inviabilizando a necessidade da criação dos outros tipos de índices. Com base nos

testes realizados, sugerem-se prudência na utilização de índices *bitmap join* para volumes de dados mais elevado. Estes não "combinam bem" com os restantes índices das tabelas, provocando grandes ineficiências. As características de cada DW são também uma condicionante para este tipo de indexação, uma vez que no caso da existência de dimensões duplamente referenciadas, ou da existência de tabelas ponte, estas estruturas de indexação não podem ser criadas da forma normal.

Em suma, a construção de estruturas de indexação não é uma "ciência exacta", sendo necessário um trabalho de monitorização constante, uma vez que os desempenhos são muito dependentes dos volumes de dados existentes num DW. Contudo, conclui-se deste estudo que os índices *bitmap* devem ser sempre uma solução a levar em forte consideração para pequenos e médios *data warehouses*, principalmente sobre atributos do tipo "inteiro". A relação espaço-tempo de *querying* (figura 39) revela a superioridade dos índices *bitmap* sobre os índices *B-Tree* no caso de estudo escolhido. Porém, sabe-se que, os índices *bitmap join*, podem não ser a melhor escolha de optimização de queries que envolvam demasiadas tabelas, com grandes cardinalidades.

Relação espaço-tempo de *querying* entre os dois métodos de indexação

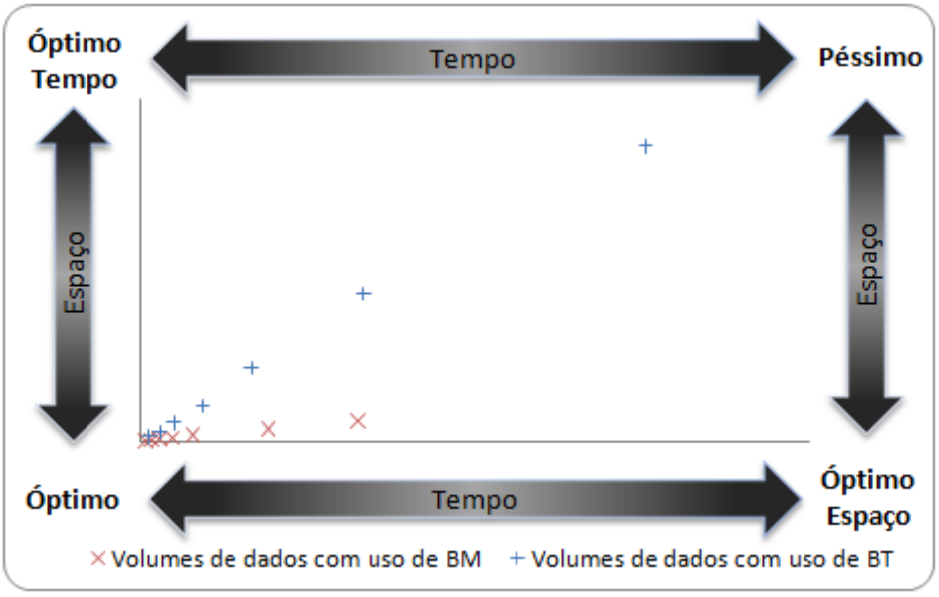


Figura 39 - Relação Espaço-Tempo de *querying* entre os dois metodos de indexação.

Como trabalho futuro, pretende-se continuar o estudo do impacto dos índices *bitmap* sobre diferentes *data warehouses*, com diferentes características, para se tentar identificar possíveis padrões de utilização, bem como eventuais pontos de quebra de desempenho do sistema. Para tal, os testes deverão ser migrados de um ambiente mais académico para um ambiente de aplicação real, onde existam *Data Warehouses* de grande dimensão e com atributos de elevada cardinalidade. Desta forma poderá ter-se uma noção mais real e mais exacta do impacto destes índices sobre um DW. Nestes casos, poderão ser exploradas novas técnicas para viabilizar o uso de índices *bitmap* nestes ambientes, como por exemplo, possíveis discretizações de atributos de elevada cardinalidade e *tuning* segundo várias abordagens - "*Tuning by Cardinality Feedback*" [WWW15] ou "*Tuning by Sampling*" [WWW16]. Estes últimos são mecanismos que permitem fazer com que o Oracle opte por diferentes planos de execução, fazendo com que utilize determinados índices que de outra forma não seriam usados. Por fim, pretende-se analisar também o impacto dos vários tipos de índices *bitmap* no carregamento intensivo de novos dados num DW.

Bibliografia

Alexander Hinneburg & Daniel A. Keim (1999). Optimal Grid-Clustering: Towards Breaking the Curse of Dimensionality in High-Dimensional Clustering. In Proceedings of the 25th international Conference on Very Large Data Bases (September 07 - 10, 1999). M. P. Atkinson, M. E. Orlowska, P. Valduriez, S. B. Zdonik, and M. L. Brodie, Eds. Very Large Data Bases. Morgan Kaufmann Publishers, San Francisco, CA, 506-517.

Bob Bryla & Kevin Loney (2008). Oracle Database 11g DBA Handbook: Administer a Scalable, Secure Oracle Enterprise Database, McGraw-Hill Companies, Inc.

Chee-Yong Chan & Yannis E. Ioannidis (1998a). An Efficient Bitmap Encoding Scheme for Selection Queries. Computer Sciences Department, University of Wisconsin-Madison, 1998. http://www.cs.wisc.edu/Ncychan/interval.ps.

Chee-Yong Chan & Yannis E. Ioannidis (1998b), Bitmap Index Design and Evaluation. SIGMOD, Seattle, Washington, USA, ACM Press.

Chee-Yong Chan & Yannis E. Ioannidis (1999). An Efficient Bitmap Encoding Scheme for Selection Queries, SIGMOD Conference, Philadelphia, Pennsylvania, USA, ACM Press.

Chengkai Li, Kevin Chen-Chuan Chang, Ihab F. Ilyas (2005). Efficient processing of ad-hoc top-k aggregate queries in OLAP. Technical Report UIUCDCS-R-2005-2596, Department of Computer Science, UIUC, June 2005. http://aim.cs.uiuc.edu.

Clark D. French (1995). "One size fits all" database architectures do not work for DSS. In Proceedings of the 1995 ACM SIGMOD international Conference on Management of Data (San Jose, California, United States, May 22 - 25, 1995). M. Carey and D. Schneider, Eds. SIGMOD '95. ACM, New York, NY, 449-450.

Donald E. Knuth (1998). The Art of Computer Programming, Volume 3, Addison Wesley.

Doron Rotem & Kurt Stockinger & Kesheng Wu (2004). Efficient binning for bitmap indices on high-cardinality attributes. Technical Report LBNL-56936. Lawrence Berkeley National Laboratory, Berkeley, California, USA, (November 17, 2004)

Doron Rotem & Kurt Stockinger & Kesheng Wu (2005a). Optimizing Candidate Check Costs for Bitmap Indices, Conference on Information and Knowledge Management (CIKM), Bremen, Germany, November 2005, ACM Press.

Doron Rotem & Kurt Stockinger & Kesheng Wu (2005b). Optimizing I/O Costs of Multi-Dimensional Queries using Bitmap Indices. International Conference on Database and Expert Systems Applications (DEXA) (Copenhagen, Denmark) Springer Verlag. 220-229.

Doron Rotem & Kurt Stockinger & Kesheng Wu (2006). Minimizing I/O Costs of Multi-Dimensional Queries with Bitmap Indices. SSDBM, 2006.

Douglas Comer (1979). The ubiquitous B-Tree, Computing Surveys, 11(2), 121-137.

Elizabeth O'Neil & Patrick O'Neil & Kesheng Wu (2007). Bitmap Index Design Choices and Their Performance Implications. In Proceedings of the 11th international Database Engineering and Applications Symposium (September 06 - 08, 2007). IDEAS. IEEE Computer Society, Washington, DC, 72-84.

Gennady Antoshenkov & Mohamed Ziauddin (1996). Query processing and optimization in ORACLE RDB. The VLDB Journal, 5:229–237, 1996.

Gennady Antoshenkov (1994). Byte-aligned bitmap compression. Technical report, Oracle Corp., 1994. U.S. Patent number 5,363,098.

Gennady Antoshenkov (1995). Byte-aligned bitmap compression. In Proceedings of the Conference on Data Compression (March 28 - 30, 1995). DCC. IEEE Computer Society, Washington, DC, 476.

Goetz Graefe (1993). Query Evaluation Techniques for Large Databases. Computing Surveys, 25(2), 73–170, 1993.

Harry K.T. Wong & Hsiu-Fen Liu & Frank Olken & Doron Rotem & Linda Wong (1985). Bit Transposed Files. International Conference on Very Large Databases (VLDB), Stockholm, Sweden. Morgan Kaufmann.

Hung-Yi Lin (2008). Compressed B+-trees. W. Trans. on Comp. 7, 12 (Dec. 2008), 2001-2010.

Jacob Ziv & Abraham Lempel (1977). A universal algorithm for sequential data compression. IEEE Transactions on Information Theory, 23(3):337--343, 1977.

Jason Price (2008). Oracle Database 11g SQL: Master SQL and PL/SQL in the Oracle Database, McGraw-Hill Companies, Inc.

Jean-loup Gailly & Mark Adler (2002). zlib 1.1.4 Manual, March 11th, 2002. (Código fonte disponivel em http://zlib.net/)

Kesheng Wu & Ekow J. Otoo & Arie Shoshani (2001a). A performance comparison of bitmap indexes. In Proceedings of the Tenth international Conference on information and Knowledge Management (Atlanta, Georgia, USA, October 05 - 10, 2001). H. Paques, L. Liu, and D. Grossman, Eds. CIKM '01. ACM, New York, NY, 559-561.

Kesheng Wu & Ekow J. Otoo & Arie Shoshani and Henrik Nordberg (2001b). Notes on design and implementation of compressed bit vectors. Technical Report LBNL/PUB-3161. Lawrence Berkeley National Laboratory, Berkeley, CA, 2001.

Kesheng Wu & Ekow J. Otoo & Arie Shoshani (2002). Compressing Bitmap Indexes for Faster Search Operations. In Proceedings of the 14th international Conference on Scientific and Statistical Database Management (July 24 - 26, 2002). SSDBM. IEEE Computer Society, Washington, DC, 99-108.

Kesheng Wu & Ekow Otoo & Arie Shoshani (2004). On the performance of bitmap indices for high cardinality attributes. In Proceedings of the Thirtieth international Conference on Very Large Data Bases - Volume 30 (Toronto, Canada, August 31 - September 03, 2004). M. A. Nascimento, M. T. Özsu, D. Kossmann, R. J. Miller, J. A. Blakeley, and K. B. Schiefer, Eds. Very Large Data Bases. VLDB Endowment, 24-35.

Kesheng Wu & Ekow J. Otoo & Arie Shoshani (2006a). An Efficient Compression Scheme for Bitmap Indices. ACM Transactions on Database Systems, v31, pages 1-38.

Kesheng Wu & Ekow J. Otoo & Arie Shoshani (2006b). Optimizing bitmap indices with efficient compression. ACM Trans. Database Syst. 31, 1 (Mar. 2006), 1-38.

Kesheng Wu & Kurt Stockinger & Arie Shoshani (2008). Breaking the Curse of Cardinality on Bitmap Indexes. In Proceedings of the 20th international Conference on Scientific and Statistical Database Management (Hong Kong, China, July 09 - 11, 2008). B. Ludäscher and N. Mamoulis, Eds. Lecture Notes In Computer Science, vol. 5069. Springer-Verlag, Berlin, Heidelberg, 348-365.

Kevin Loney (2009). Oracle Database 11g The Complete Reference: Master the Powerful Features of the Latest Database Release, McGraw-Hill Companies, Inc.

Kun-Lung Wu & Philip S. Yu (1996). Range-Based Bitmap Indexing for High-Cardinality Attributes with Skew. Technical report, IBM Watson Research Center, May 1996.

Kun-Lung Wu & Philip S. Yu (1998). Range-Based Bitmap Indexing for High Cardinality Attributes with Skew. In Proceedings of the 22nd international Computer Software and Applications Conference (August 19 - 21, 1998). COMPSAC. IEEE Computer Society, Washington, DC, 61-67.

Kurt Stockinger & Kesheng Wu (2006), Bitmap Indices for Data Warehouses, Berkeley, California.

Kurt Stockinger & Kesheng Wu & Arie Shoshani (2002). Strategies for processing ad hoc queries on large data warehouses. In Proceedings of the 5th ACM international Workshop on Data warehousing and OLAP (McLean, Virginia, USA, November 08 - 08, 2002). DOLAP '02. ACM, New York, NY, 72-79.

Kurt Stockinger & Kesheng Wu & Arie Shoshani (2004). Evaluation Strategies for Bitmap Indices with Binning. International Conference on Database and Expert Systems Applications (DEXA)(Zaragoza, Spain). Springer-Verlag. 120--129.

Lars Arge & Mark de Berg & Herman J. Haverkort & Ke Yi (2004). The Priority R-tree: a practically efficient and worst-case optimal R-tree. In Proceedings of the 2004 ACM SIGMOD international Conference on Management of Data (Paris, France, June 13 - 18, 2004). SIGMOD '04. ACM, New York, NY, 347-358.

Alice Marques & Nélio Guimarães (2009). Projecto e Implementação de um Sistema de Data Webhousing, Relatório Técnico, Projecto UCE15, Departamento de Informática, Universidade do Minho.

Ming-Chuan Wu & Alejandro P. Buchmann (1998). Encoded Bitmap Indexing for Data Warehouses. International Conference on Data Engineering (ICDE), Orlando, Florida, USA. IEEE Computer Society Press.

Navneet Goyal & Susheel Kumar Zaveri & Yashvardhan Sharma (2006). Improved Bitmap Indexing Strategy for Data Warehouses. In Proceedings of the 9th international Conference on information Technology (December 18 - 21, 2006). ICIT. IEEE Computer Society, Washington, DC, 213-216.

Nick Koudas (2000). Space efficient bitmap indexing. In Proceedings of the Ninth international Conference on information and Knowledge Management (McLean, Virginia, United States, November 06 - 11, 2000). CIKM '00. ACM, New York, NY, 194-201.

Norbert Beckmann & Bernhard Seeger (2009). A revised r*-tree in comparison with related index structures. In Proceedings of the 35th SIGMOD international Conference on Management of Data (Providence, Rhode Island, USA, June 29 - July 02, 2009). C. Binnig and B. Dageville, Eds. SIGMOD '09. ACM, New York, NY, 799-812.

Patrick O'Neil & Dallan Quass (1997). Improved Query Performance with Variant Indexes. International Conference on Management of Data (SIGMOD 1997), Tucson, Arizona, USA. ACM Press.

Patrick O'Neil (1987). Model 204 Architecture and Performance. Workshop in High Performance Transaction Systems, Asilomar, California, USA. Springer-Verlag.

Philip Howard (2008). Sybase IQ technology overview. An InDetail paper by Bloor Research (January, 2008) (http://www.sybase.com/files/White_Papers/Bloor_Sybase_IQ_ Technology_Overview.pdf).

Ralph Kimbal & Joe Caserta (2004). The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming and Delivering Data, Wiley Publishing, Inc.

Ralph Kimbal & Margy Ross & Warren Thornthwaite & Joy Mundy & Bob Becker (2008). The Data Warehouse Lifecycle Toolkit: Practical Techniques for Building Data Warehouse and Business Intelligence Systems, Second Edition, Wiley Publishing, Inc.

Richard Niemiec (2007). Oracle Database 10g Performance Tuning Tips and Techniques, McGraw-Hill Companies, Inc.

Robert Fenk & Akihiko Kawakami & Volker Markl & Rudolf Bayer & Shunji Osaki (2000). Bulk Loading a Data Warehouse Built Upon a UB-Tree. In Proceedings of the 2000 international Symposium on Database Engineering & Applications (September 18 - 20, 2000). B. C. Desai, Y. Kiyoki, and M. Toyama, Eds. IDEAS. IEEE Computer Society, Washington, DC, 179-187.

Roger MacNicol & Blaine French (2004). Sybase IQ multiplex - designed for analytics. In Proceedings of the Thirtieth international Conference on Very Large Data Bases - Volume 30 (Toronto, Canada, August 31 - September 03, 2004). M. A. Nascimento, M. T. Özsu, D. Kossmann, R. J. Miller, J. A. Blakeley, and K. B. Schiefer, Eds. Very Large Data Bases. VLDB Endowment, 1227-1230.

Sam R. Alapati (2009). Expert Oracle Database 11g Administration, Apress.

Sihem Amer-Yahia & Theodore Johnson (2000). Optimizing Queries on Compressed Bitmaps. In Proceedings of the 26th international Conference on Very Large Data Bases (September 10 - 14, 2000). A. E. Abbadi, M. L. Brodie, S. Chakravarthy, U. Dayal, N. Kamel, G. Schlageter, and K. Whang, Eds. Very Large Data Bases. Morgan Kaufmann Publishers, San Francisco, CA, 329-338.

Stefan Berchtold & Christian Böhm & Hans-Peter Kriegel (1998). The Pyramid-Technique: Towards Breaking the Curse of Dimensionality. SIGMOD Record, 27(2), 142-153.

Surajit Chaudhuri & Umeshwar Dayal & Venkatesh Ganti (2001). Database Technology for Decision Support Systems. Computer, 34(12), 48-55.

Surajit Chaudhuri & Umeshwar Dayal (1997). An Overview of Data Warehousing and OLAP Technology, ACM SIGMOD Record, 26(1), 65-74, March 1997.

Theodore Johnson (1999). Performance Measurements of Compressed Bitmap Indices. In Proceedings of the 25th international Conference on Very Large Data Bases (September 07 - 10, 1999). M. P. Atkinson, M. E. Orlowska, P. Valduriez, S. B. Zdonik, and M. L. Brodie, Eds. Very Large Data Bases. Morgan Kaufmann Publishers, San Francisco, CA, 278-289.

Thomas Connolly & Carolyn Begg (2005). Database System, Fourth Edition, Addison Wesley.

Volker Gaede & Oliver Günther (1998). Multidimensional access methods. ACM Comput. Surv. 30, 2 (Jun. 1998), 170-231.

Referências WWW

[WWW1] http://mediaproducts.gartner.com/reprints/microsoft/vol3/article7/article7.html (Último acesso: 04-Out-2009).
Site disponibiliza institucional da Gartner. Estudo sobre mercado de bases de dados para sistemas de *data warehousing*.

[WWW2] http://www.gartner.com/it/page.jsp?id=507466 (Último acesso: 04-Out-2009).
Site disponibiliza institucional da Gartner. Estudo sobre o crescimento do mercado de bases de dados para sistemas de *data warehousing*.

[WWW3] http://teradata.uark.edu/research/wang/indexes.html (Último acesso: 05-Out-2009).
Site institucional da *University of Arkansas*. Informação relativa aos tipos de indexação existentes em Teradata.

[WWW4] http://teradata.uark.edu/research/rusch/td_access_paths.html (Último acesso: 05-Out-2009).
Site institucional da *University of Arkansas*. Informação relacionada com o método de acesso aos dados por parte do Teradata.

[WWW5] http://msdn.microsoft.com/en-us/library/bb522541.aspx (Último acesso: 05-Out-2009).
Site institucional da Microsoft. Informação de optimização de *queries* em *data warehousing* no SQL Server (*Bitmap Filtering*).

[WWW6] http://www.db2ude.com/?q=node/16 (Último acesso: 05-Out-2009).
Informação sobre índices *bitmap* em DB2.

[WWW7] http://www.dwoptimize.com/2007/06/101010-answer-to-life-universe-and.html (Último acesso: 05-Out-2009).
Informação sobre índices *bitmap* em DB2.

[WWW8] http://businessintelligence.ittoolbox.com/groups/vendor-selection/bi-select/netezza-any-experiences-with-this-product-1222959 (Último acesso: 04-Out-2009).
Informação relativa ao SGBD Netezza.

[WWW9] http://www.dbms2.com/2009/08/25/sybase-iq-technical-highlights/ (Último acesso: 05-Out-2009).
Informação sobre a forma de armazenamento de dados utilizado pelo SGBD da Sybase (Sybase IQ).

[WWW10] http://www.cs.umbc.edu/help/oracle8/server.815/a68003/rollup_c.htm#32084 (Último acesso: 05-Set-2009).
Site institucional da *University of MaryLand, Baltimore County*. Informação (sintaxe e exemplos) sobre as instruções de SQL *Rollup* e *Cube* disponibilizadas pelo Oracle.

[WWW11] http://www.remote-dba.net/pl_sql/t_analytic_functions_oracle_rollup_cube.htm (Último acesso: 05-Set-2009).
Informação (sintaxe e exemplos) sobre as instruções de SQL *Rollup* e *Cube* disponibilizadas pelo Oracle.

[WWW12] http://www.dba-oracle.com/t_cube.htm (Último acesso: 05-Set-2009).
Burleson Consulting. Informação (sintaxe e exemplos) relativa à instrução de SQL Cube disponibilizada pelo Oracle.

[WWW13] http://www.scribd.com/doc/2713819/Oracle-bitmap-join-indexes (Último acesso: 20-Set-2009).
Informação relativa aos índices *bitmap join* presentes em Oracle.

[WWW14] http://www.centrexcc.com. "Tuning by Cardinality Feedback by Wolfgang Breitling (Oracle Certified Professional)" (http://www.centrexcc.com/Tuning%20by%20Cardinality%20Feedback.pdf) (Último acesso: 29-Set-2009).
A *Centrex* é uma corporação de consultadoria especializada em Oracle. Neste site podem ser encontrados vários artigos que visam a optimização em Oracle.

[WWW15] http://www.perfvision.com/ "A-Sampling We Will Go by Kyle Hailey" (http://www.perfvision.com/papers/Sampling.pdf) (Último acesso: 29-Set-2009).
Aqui podem ser encontrados vários artigos que visam também a optimização em Oracle, disponibilizados pela *Performance Vision*.

Anexos

Anexo 1

Id	Operation	Name	Starts	E-Rows	A-Rows
1	SORT ORDER BY		1	5	26866
2	HASH GROUP BY		1	5	26866
* 3	HASH JOIN		1	5	52618
* 4	TABLE ACCESS FULL	REQUEST_DIM	1	326	342
* 5	HASH JOIN		1	232	2493K
* 6	HASH JOIN		1	4	41673
* 7	HASH JOIN		1	4	41673
* 8	HASH JOIN		1	4	41673
* 9	HASH JOIN		1	4	41673
* 10	HASH JOIN		1	194	2152K
11	TABLE ACCESS FULL	TEMPO_DIM	1	24	24
* 12	HASH JOIN		1	4650	2839K
* 13	TABLE ACCESS FULL	DATA_DIM	1	365	366
14	TABLE ACCESS FULL	TF_SESSIONS	1	4830K	4830K
* 15	TABLE ACCESS FULL	AGENTE_DIM	1	654	640
16	TABLE ACCESS FULL	REFERRER_DIM	1	3936	3936
17	TABLE ACCESS FULL	COMPUTADORUTILIZADOR_DIM	1	398K	398K
18	TABLE ACCESS FULL	PAIS_SUBDIM	1	100	100
19	TABLE ACCESS FULL	TP_SESSION	1	289M	289M

```
Predicate Information (identified by operation id):
---------------------------------------------------

   3 - access("TP"."REQUEST_ID"="RQ"."REQUEST_ID")
   4 - filter("RQ"."TIPO_CLASSE"='PS')
   5 - access("TP"."SESSIONID"="TF"."SESSIONID")
   6 - access("CU"."CODIGOISOPAIS"="PA"."CODIGOISOPAIS")
   7 - access("TF"."HOST"="CU"."IPADDRESS")
   8 - access("TF"."REFERRER"="RF"."REFERRER_ID")
   9 - access("TF"."USERAGENT"="AG"."AGENTE_ID")
  10 - access("TF"."STARTTIME"="TMP"."HORA" AND "TF"."ENDTIME"="TMP"."HORA")
  12 - access("TF"."STARTDATE"="DT"."DATA_ID" AND "TF"."ENDDATE"="DT"."DATA_ID")
  13 - filter("DT"."ANO"=2008)
  15 - filter(("AG"."TIPO"='Crawler' OR "AG"."TIPO"='User'))
```

Anexo 2

Id	Operation	Name	Starts	E-Rows	A-Rows
1	TEMP TABLE TRANSFORMATION		1		26866
2	LOAD AS SELECT		1		1
* 3	TABLE ACCESS FULL	REQUEST_DIM	1	326	342
4	LOAD AS SELECT		1		1
* 5	HASH JOIN		1	8	41673
6	TABLE ACCESS BY INDEX ROWID	DATA_DIM	1	365	366
* 7	INDEX RANGE SCAN	ANO_BTIX	1	365	366
* 8	TABLE ACCESS BY INDEX ROWID	TF_SESSIONS	1	8050	41673
9	BITMAP CONVERSION TO ROWIDS		1		55508
10	BITMAP AND		1		9
11	BITMAP MERGE		1		12
12	BITMAP KEY ITERATION		1		640
* 13	VIEW	index$_join$_041	1	654	640
* 14	HASH JOIN		1		640
15	INLIST ITERATOR		1		640
* 16	INDEX RANGE SCAN	TIPO_AGENTE_BTIX	2	654	640
17	INDEX FAST FULL SCAN	AGENTE_ID_PK	1	654	32679
18	BITMAP CONVERSION FROM ROWIDS		640		640
* 19	INDEX RANGE SCAN	IBT_FK6	640		94192
20	BITMAP MERGE		1		56
21	BITMAP KEY ITERATION		1		732
22	TABLE ACCESS BY INDEX ROWID	DATA_DIM	1	365	366
* 23	INDEX RANGE SCAN	ANO_BTIX	1	365	366
24	BITMAP CONVERSION FROM ROWIDS		366		732
* 25	INDEX RANGE SCAN	IBT_FK1	366		2869K
26	SORT ORDER BY		1	1	26866
27	HASH GROUP BY		1	1	26866
* 28	HASH JOIN		1	1	52618
* 29	HASH JOIN		1	1	52618
* 30	HASH JOIN		1	1	52618
* 31	HASH JOIN		1	1	52618
* 32	HASH JOIN		1	1	52618
* 33	HASH JOIN		1	1	52618
* 34	HASH JOIN		1	1	52618
35	TABLE ACCESS BY INDEX ROWID	TP_SESSION	1	9	52618
36	BITMAP CONVERSION TO ROWIDS		1		52618
37	BITMAP AND		1		7
38	BITMAP MERGE		1		30
39	BITMAP KEY ITERATION		1		41673
40	TABLE ACCESS FULL	SYS_TEMP_0FD9D6612_DB8FB	1	1	41673
41	BITMAP CONVERSION FROM ROWIDS		41673		41673
* 42	INDEX RANGE SCAN	IBT_FK8	41673		2493K
43	BITMAP MERGE		1		567
44	BITMAP KEY ITERATION		1		1321
45	TABLE ACCESS FULL	SYS_TEMP_0FD9D6611_DB8FB	1	1	342
46	BITMAP CONVERSION FROM ROWIDS		342		1321
* 47	INDEX RANGE SCAN	IBT_FK9	342		6089K
48	TABLE ACCESS FULL	SYS_TEMP_0FD9D6611_DB8FB	1	326	342
49	TABLE ACCESS BY INDEX ROWID	TF_SESSIONS	1	8050	55508
50	BITMAP CONVERSION TO ROWIDS		1		55508
51	BITMAP AND		1		9
52	BITMAP MERGE		1		12
53	BITMAP KEY ITERATION		1		640
* 54	VIEW	index$_join$_074	1	654	640
* 55	HASH JOIN		1		640
56	INLIST ITERATOR		1		640
* 57	INDEX RANGE SCAN	TIPO_AGENTE_BTIX	2	654	640
58	INDEX FAST FULL SCAN	AGENTE_ID_PK	1	654	32679
59	BITMAP CONVERSION FROM ROWIDS		640		640
* 60	INDEX RANGE SCAN	IBT_FK6	640		94192
61	BITMAP MERGE		1		56
62	BITMAP KEY ITERATION		1		732
63	TABLE ACCESS BY INDEX ROWID	DATA_DIM	1	365	366
* 64	INDEX RANGE SCAN	ANO_BTIX	1	365	366
65	BITMAP CONVERSION FROM ROWIDS		366		732
* 66	INDEX RANGE SCAN	IBT_FK1	366		2869K
67	TABLE ACCESS FULL	TEMPO_DIM	1	24	24
68	TABLE ACCESS BY INDEX ROWID	DATA_DIM	1	365	366
* 69	INDEX RANGE SCAN	ANO_BTIX	1	365	366
70	TABLE ACCESS FULL	REFERRER_DIM	1	3936	3936
71	TABLE ACCESS FULL	COMPUTADORUTILIZADOR_DIM	1	398K	398K
72	TABLE ACCESS FULL	PAIS_SUBDIM	1	100	100

```
Predicate Information (identified by operation id):
---------------------------------------------------

   3 - filter("RQ"."TIPO_CLASSE"='PS')
   5 - access("TF"."STARTDATE"="DT"."DATA_ID" AND "TF"."ENDDATE"="DT"."DATA_ID")
   7 - access("DT"."ANO"=2008)
   8 - filter(("TF"."ENDTIME"="TF"."STARTTIME" AND "TF"."ENDDATE"="TF"."STARTDATE"))
  13 - filter(("AG"."TIPO"='Crawler' OR "AG"."TIPO"='User'))
  14 - access(ROWID=ROWID)
  16 - access(("AG"."TIPO"='Crawler' OR "AG"."TIPO"='User'))
  19 - access("TF"."USERAGENT"="AG"."AGENTE_ID")
  23 - access("DT"."ANO"=2008)
  25 - access("TF"."STARTDATE"="DT"."DATA_ID")
  28 - access("CU"."CODIGOISOPAIS"="PA"."CODIGOISOPAIS")
  29 - access("TF"."HOST"="CU"."IPADDRESS")
  30 - access("TF"."REFERRER"="RF"."REFERRER_ID")
  31 - access("TF"."STARTDATE"="DT"."DATA_ID" AND "TF"."ENDDATE"="DT"."DATA_ID")
  32 - access("TF"."STARTTIME"="TMP"."HORA" AND "TF"."ENDTIME"="TMP"."HORA")
  33 - access("TP"."SESSIONID"="TF"."SESSIONID")
  34 - access("TP"."REQUEST_ID"="C0")
  42 - access("TP"."SESSIONID"="C0")
  47 - access("TP"."REQUEST_ID"="C0")
  54 - filter(("AG"."TIPO"='Crawler' OR "AG"."TIPO"='User'))
  55 - access(ROWID=ROWID)
  57 - access(("AG"."TIPO"='Crawler' OR "AG"."TIPO"='User'))
  60 - access("TF"."USERAGENT"="AG"."AGENTE_ID")
  64 - access("DT"."ANO"=2008)
  66 - access("TF"."STARTDATE"="DT"."DATA_ID")
  69 - access("DT"."ANO"=2008)

Note
-----
   - star transformation used for this statement
```

Anexo 3

Id	Operation	Name	Starts	E-Rows	A-Rows
1	TEMP TABLE TRANSFORMATION		1		26866
2	LOAD AS SELECT		1		1
3	TABLE ACCESS BY INDEX ROWID	REQUEST_DIM	1	326	342
4	BITMAP CONVERSION TO ROWIDS		1		342
* 5	BITMAP INDEX SINGLE VALUE	TIPO_CLASSE_REQUEST_BMIX	1		1
6	LOAD AS SELECT		1		1
* 7	HASH JOIN		1	8	41673
* 8	TABLE ACCESS FULL	DATA_DIM	1	365	366
* 9	TABLE ACCESS BY INDEX ROWID	TF_SESSIONS	1	8050	41673
10	BITMAP CONVERSION TO ROWIDS		1		55508
11	BITMAP AND		1		8
12	BITMAP MERGE		1		52
13	BITMAP KEY ITERATION		1		2196
* 14	TABLE ACCESS FULL	DATA_DIM	1	365	366
* 15	BITMAP INDEX RANGE SCAN	IBM_FK1	366		2196
16	BITMAP MERGE		1		11
17	BITMAP KEY ITERATION		1		640
* 18	VIEW	index$_join$_041	1	654	640
* 19	HASH JOIN		1		640
20	INLIST ITERATOR		1		640
21	BITMAP CONVERSION TO ROWIDS		2	654	640
* 22	BITMAP INDEX SINGLE VALUE	TIPO_AGENTE_BMIX	2		2
23	INDEX FAST FULL SCAN	AGENTE_ID_PK	1	654	32679
* 24	BITMAP INDEX RANGE SCAN	IBM_FK6	640		640
25	SORT ORDER BY		1	1	26866
26	HASH GROUP BY		1	1	26866
* 27	HASH JOIN		1	1	52618
* 28	HASH JOIN		1	1	52618
* 29	HASH JOIN		1	1	52618
* 30	HASH JOIN		1	1	52618
* 31	HASH JOIN		1	1	52618
* 32	HASH JOIN		1	1	52618
* 33	HASH JOIN		1	1	52618
34	TABLE ACCESS BY INDEX ROWID	TP_SESSION	1	9	52618
35	BITMAP CONVERSION TO ROWIDS		1		52618
36	BITMAP AND		1		7
37	BITMAP MERGE		1		30
38	BITMAP KEY ITERATION		1		41673
39	TABLE ACCESS FULL	SYS_TEMP_0FD9D6616_DB8FB	1	1	41673
* 40	BITMAP INDEX RANGE SCAN	IBM_FK8	41673		41673
41	BITMAP MERGE		1		522
42	BITMAP KEY ITERATION		1		4478
43	TABLE ACCESS FULL	SYS_TEMP_0FD9D6615_DB8FB	1	1	342
* 44	BITMAP INDEX RANGE SCAN	IBM_FK9	342		4478
45	TABLE ACCESS FULL	SYS_TEMP_0FD9D6615_DB8FB	1	326	342
46	TABLE ACCESS BY INDEX ROWID	TF_SESSIONS	1	8050	55508
47	BITMAP CONVERSION TO ROWIDS		1		55508
48	BITMAP AND		1		8
49	BITMAP MERGE		1		52
50	BITMAP KEY ITERATION		1		2196
* 51	TABLE ACCESS FULL	DATA_DIM	1	365	366
* 52	BITMAP INDEX RANGE SCAN	IBM_FK1	366		2196
53	BITMAP MERGE		1		11
54	BITMAP KEY ITERATION		1		640
* 55	VIEW	index$_join$_074	1	654	640
* 56	HASH JOIN		1		640
57	INLIST ITERATOR		1		640
58	BITMAP CONVERSION TO ROWIDS		2	654	640
* 59	BITMAP INDEX SINGLE VALUE	TIPO_AGENTE_BMIX	2		2
60	INDEX FAST FULL SCAN	AGENTE_ID_PK	1	654	32679
* 61	BITMAP INDEX RANGE SCAN	IBM_FK6	640		640
62	VIEW	index$_join$_003	1	24	24
* 63	HASH JOIN		1		24
64	BITMAP CONVERSION TO ROWIDS		1	24	24
65	BITMAP INDEX FULL SCAN	PERIODO_BMIX	1		3
66	INDEX FAST FULL SCAN	HORA_PK	1	24	24
* 67	TABLE ACCESS FULL	DATA_DIM	1	365	366
68	VIEW	index$_join$_004	1	3936	3936
* 69	HASH JOIN		1		3936
70	INDEX FAST FULL SCAN	REFERRER_ID_PK	1	3936	3936
71	BITMAP CONVERSION TO ROWIDS		1	3936	3936
72	BITMAP INDEX FULL SCAN	TIPO_REFERRER_BMIX	1		2
73	TABLE ACCESS FULL	COMPUTADORUTILIZADOR_DIM	1	398K	398K
74	VIEW	index$_join$_007	1	100	100
* 75	HASH JOIN		1		100
76	INDEX FAST FULL SCAN	CODIGOISOPAIS_PK	1	100	100
77	BITMAP CONVERSION TO ROWIDS		1	100	100
78	BITMAP INDEX FULL SCAN	PAIS_BMIX	1		100

```
Predicate Information (identified by operation id):
---------------------------------------------------

   5 - access("RQ"."TIPO_CLASSE"='PS')
   7 - access("TF"."STARTDATE"="DT"."DATA_ID" AND "TF"."ENDDATE"="DT"."DATA_ID")
   8 - filter("DT"."ANO"=2008)
   9 - filter(("TF"."ENDTIME"="TF"."STARTTIME" AND "TF"."ENDDATE"="TF"."STARTDATE"))
  14 - filter("DT"."ANO"=2008)
  15 - access("TF"."STARTDATE"="DT"."DATA_ID")
  18 - filter(("AG"."TIPO"='Crawler' OR "AG"."TIPO"='User'))
  19 - access(ROWID=ROWID)
  22 - access(("AG"."TIPO"='Crawler' OR "AG"."TIPO"='User'))
  24 - access("TF"."USERAGENT"="AG"."AGENTE_ID")
  27 - access("CU"."CODIGOISOPAIS"="PA"."CODIGOISOPAIS")
  28 - access("TF"."HOST"="CU"."IPADDRESS")
  29 - access("TF"."REFERRER"="RF"."REFERRER_ID")
  30 - access("TF"."STARTDATE"="DT"."DATA_ID" AND "TF"."ENDDATE"="DT"."DATA_ID")
  31 - access("TF"."STARTTIME"="TMP"."HORA" AND "TF"."ENDTIME"="TMP"."HORA")
  32 - access("TP"."SESSIONID"="TF"."SESSIONID")
  33 - access("TP"."REQUEST_ID"="C0")
  40 - access("TP"."SESSIONID"="C0")
  44 - access("TP"."REQUEST_ID"="C0")
  51 - filter("DT"."ANO"=2008)
  52 - access("TF"."STARTDATE"="DT"."DATA_ID")
  55 - filter(("AG"."TIPO"='Crawler' OR "AG"."TIPO"='User'))
  56 - access(ROWID=ROWID)
  59 - access(("AG"."TIPO"='Crawler' OR "AG"."TIPO"='User'))
  61 - access("TF"."USERAGENT"="AG"."AGENTE_ID")
  63 - access(ROWID=ROWID)
  67 - filter("DT"."ANO"=2008)
  69 - access(ROWID=ROWID)
  75 - access(ROWID=ROWID)

Note
-----
   - star transformation used for this statement
```