



Pós-Graduação em Ciência da Computação

*Uma Arquitetura de Software para  
Descoberta e Regras de Associação  
Multidimensional, Multinível e de Outliers  
em Cubos OLAP: um Estudo de Caso com os  
Algoritmos APriori e FP-Growth*

*Por*

*Carla Moreira Tanure*

*Dissertação de Mestrado*



Universidade Federal de Pernambuco  
[posgraduacao@cin.ufpe.br](mailto:posgraduacao@cin.ufpe.br)  
[www.cin.ufpe.br/~posgraduacao](http://www.cin.ufpe.br/~posgraduacao)

RECIFE, MARÇO/2010  
Universidade Federal de Pernambuco  
Centro de Informática  
Mestrado em Ciência da Computação

**Carla Moreira Tanure**

***Uma Arquitetura de Software para Descoberta de Regras  
de Associação Multidimensional, Multinível e de  
Outliers em Cubos OLAP: um Estudo de Caso com os  
Algoritmos APriori e FP-Growth***

Orientador: Prof. Dr. Robson do Nascimento Fidalgo  
Co-orientador: Prof. Dr. Tiago Alessandro Espínola Ferreira

Dissertação apresentada ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Recife

Março de 2010

**Tanure, Carla Moreira**

**Uma arquitetura de software para descoberta de regras de associação multidimensional, multinível e de outliers em cubos OLAP: um estudo de caso com os algoritmos APriori e FP-Growth / Carla Moreira Tanure. - Recife: O Autor, 2010.**

**viii, 101 folhas : il., fig., tab., quadros**

**Dissertação (mestrado) – Universidade Federal de Pernambuco. Cln. Ciência da Computação, 2010.**

**Inclui bibliografia e apêndice.**

**1. Banco de Dados. 2. Mineração de dados. I. Título.**

**025.04**

**CDD (22. ed.)**

**MEI2010 – 068**

## **Agradecimentos**

Primeiramente, agradeço a Deus. Não apenas pela realização deste mestrado, mas por tudo em minha vida. Agradeço aos meus pais, Carlos e Shirley, minha rocha, pelo apoio incondicional, pela confiança e compreensão nos momentos de ausência. Aos meus irmãos Hugo e Samir, agradeço pela amizade e pelos momentos de alegria. A Luiz, meu namorado, agradeço pelo amor e compreensão, a ajuda e o carinho em tantos momentos em que precisei dele. Aos meus amigos, agradeço pelo apoio constante e amizade verdadeira. Não conseguiria chegar aonde cheguei sem todos vocês.

Ao Prof. Robson Fidalgo agradeço pelos ensinamentos, pelo compromisso, por sempre fazer o possível para me ajudar, por ser um modelo de ética e competência para seus alunos. Aos demais professores e funcionários do CIn/UFPE, por desempenharem com tão bem suas funções. À Secretaria da Fazenda do Estado do Ceará agradeço pelo incentivo a minha formação profissional.

## Resumo

O processo tradicional de descoberta de conhecimento em bases de dados (*KDD* – *Knowledge Discovery in Databases*) não contempla etapas de processamento multidimensional e multinível (i.e., processamento *OLAP* - *OnLine Analytical Processing*) para minerar cubos de dados. Por consequência, a maioria das abordagens de OLAM (*OLAP Mining*) propõe adaptações no algoritmo minerador. Dado que esta abordagem provê uma solução fortemente acoplada ao algoritmo minerador, ela impede que as adaptações para mineração multidimensional e multinível sejam utilizadas com outros algoritmos. Além disto, grande parte das propostas de OLAM para regras de associação não considera o uso de um servidor OLAP e não tira proveito de todo o potencial multidimensional e multinível presentes nos cubos OLAP. Por estes motivos, algum retrabalho (e.g., re-implementação de operações OLAP) é realizado e padrões possivelmente fortes decorrentes de generalizações não são identificados.

Diante desse cenário, este trabalho propõe a arquitetura DOLAM (*Decoupled OLAM*) para mineração desacoplada de regras de associação multidimensional, multinível e de *outliers* em cubos OLAP. A arquitetura DOLAM deve ser inserida no processo de *KDD* (*Knowledge Discovery in Databases*) como uma etapa de processamento que fica entre as etapas de Pré-Processamento e Transformação de Dados. A arquitetura DOLAM define e implementa três componentes: 1) Detector de Outliers, 2) Explorador de Subcubos e 3) Expansor de Ancestrais. A partir de uma consulta do usuário, estes componentes são capazes de, respectivamente: 1) identificar ruídos significativos nas células do resultado; 2) explorar, recursivamente, todas as células do resultado, de forma a contemplar todas as possibilidades de combinações multidimensional e multinível e 3) recuperar todos os antecessores (generalizações) das células do resultado. O componente central da arquitetura é o Expansor de Ancestrais - o único de uso obrigatório. Ressalta-se que, a partir desses componentes, o processamento OLAM fica desacoplado do algoritmo minerador e permite realizar descobertas mais abrangentes, as quais, por consequência, podem retornar padrões potencialmente mais fortes. Como prova de conceito, foi realizado um estudo de caso com dados reais de uma empresa de micro-crédito. O estudo de caso foi implementado em *Java*, fez uso do servidor OLAP *Mondrian* e utilizou as implementações dos algoritmos para mineração de regras de associação *APriori* e *FP-Growth* do pacote de *software Weka*.

Palavras-Chave: OLAP, Mineração de Dados, KDD, OLAM, Regras de Associação, *APriori*, *FP-Growth*, mineração multidimensional, mineração multinível, *outlier*.

# Abstract

The traditional KDD (Knowledge Discovery in Databases) process does not observe any multidimensional and multilevel (OLAP) processing for mining datacubes. As a consequence, most of the OLAM (OLAP Mining) approaches propose adaptations to mining algorithms. These approaches are strongly coupled with the mining algorithm, as they do not allow multidimensional and multilevel adaptations to be used with other algorithms. Besides, most of the OLAM proposals do not use an OLAP server, nor do they profit the multilevel and multidimensional potential present in OLAP datacubes. Due to these reasons, some rework (reimplementation of OLAP operations) must be performed and potentially strong patterns resulting from generalizations may not be identified.

In face of this scenario, this work proposes the DOLAM (Decoupled OLAM) architecture for decoupled mining of multidimensional, multilevel and outlier mining of association rules in OLAP datacubes. The DOLAM architecture is to be introduced in the KDD process as a step between Data Pre-processing and Data Transformation. The DOLAM architecture defines and implements three components, namely: 1) Outlier Detector, 2) Cube Explorer and 3) Ancestor Expander. Starting from a user query, these components are able to, respectively: 1) identify significant noise in the result set cells; 2) recursively explore all the result set cells, as to contemplate all the multidimensional and multilevel possible combinations and 3) fetch all the ancestors (generalizations) of the result set cells. The latter is the central component of the proposed architecture, and the only one of mandatory use. We point out that, by using these components, the OLAM processing is totally decoupled from the mining algorithm, which allows one to perform more extensive search, which may return more potentially strong patterns. As a proof of concept, a case study was performed with real data from a micro-credit business company. The case study was implemented in Java and made use of the OLAP server Mondrian and employed the implementations of the APriori and FP-Growth algorithms from the software package Weka.

**Keywords:** OLAP, Data Mining, KDD, OLAM, Association Rules, APriori, FP-Growth, multidimensional mining, multilevel mining, outlier.

# Sumário

<b>1. Introdução .....</b>	<b>1</b>
1.1. Apresentação.....	1
1.2. Motivação .....	2
1.3. Objetivos .....	3
1.4. Estrutura do Trabalho .....	3
<b>2. Conceitos Básicos.....</b>	<b>5</b>
2.1. Data Warehouse e Processamento Analítico de Dados .....	5
2.1.1. MDX .....	9
2.2. Mineração de Dados e Regras de Associação.....	10
2.1.1 O processo de descoberta de conhecimento em bases de dados .....	11
2.2.1.1. Algoritmo APriori .....	17
2.2.1.2. Algoritmo FP-Growth .....	20
2.3. OLAM (OLAP Mining).....	23
2.4. Outlier mining .....	25
2.5. Conclusão.....	27
<b>3. Arquitetura DOLAM .....</b>	<b>28</b>
3.1. Mapeamento entre conceitos de bases de dados transacionais e cubos de dados OLAP28	
3.2. Arquitetura DOLAM .....	30
3.2.1. Componente Expansor de Ancestrais .....	32
3.2.2. Componente Detector de <i>Outliers</i> .....	37
3.2.3. Componente Explorador de Subcubos .....	41
3.3. Funcionamento da Arquitetura .....	48
Cenário 1: SCI com o Componente Expansor de Ancestrais .....	48
3.4. API DOLAM .....	51
3.5. Considerações Finais .....	52
<b>4. Estudo de caso .....</b>	<b>53</b>
4.1. Cubo utilizado.....	53
4.2. Mineração de Dados .....	56
4.3. Comparação entre mineração tradicional e mineração com DOLAM.....	58
4.4. Cenário de maior custo computacional na arquitetura DOLAM.....	61
4.5. Cenário de utilização de todos os componentes da arquitetura DOLAM.....	64
4.6. Considerações Finais .....	69
<b>5. Trabalhos Relacionados.....</b>	<b>70</b>
5.1. Adaptive-FP .....	70
5.2. ML-T2L1 .....	74
5.3. EMARDC .....	78
5.5. An Outlier-based Data Association Method For Linking Criminal Incidents .	86
5.6. Discovering the Association Rules in OLAP Data Cube with Daily Downloads of Folklore Materials .....	87
5.7. Comparação entre os trabalhos .....	89
<b>6. Conclusão e trabalhos futuros.....</b>	<b>92</b>
6.1. Considerações Finais .....	92
6.2. Contribuições .....	92
6.3. Trabalhos Futuros .....	93
<b>Bibliografia.....</b>	<b>94</b>

## **Apêndice I Esquema do cubo de dados utilizado no Estudo de Caso (Capítulo 4) 97**



## Índice de Figuras

Figura 1 - Esboço e exemplo de modelos estrela .....	6
Figura 2 - Cubo de Dados.....	8
Figura 3 - Operadores OLAP .....	9
Figura 4 - Exemplo de uma declaração MDX e seu subcubo retornado .....	10
Figura 5 – Processo de KDD .....	11
Figura 6 - Atividades de Mineração de Dados .....	13
Figura 7 - Exemplo de representação de uma tabela plana com e sem esquemas bem definidos .....	23
Figura 8 - Exemplo de Hierarquia da dimensão Produto .....	24
Figura 9 - Quartis de uma distribuição de dados .....	27
Figura 10 - Representação gráfica de <i>outliers</i> .....	27
Figura 11 - Processo de KDD com a etapa DOLAM .....	31
Figura 12 - Arquitetura DOLAM .....	31
Figura 13 - Hierarquia da dimensão Tempo .....	32
Figura 14 - Célula original .....	35
Figura 15 - Processo de expansão da célula da Figura 14 .....	36
Figura 16 - Célula expandida resultante da célula da Figura 14 .....	36
Figura 17 - Utilização do componente Expansor de Ancestrais.....	36
Figura 18 – Consulta inicial .....	42
Figura 19 - Membros dos subcubos gerados pelo Componente Explorador de Subcubos .....	45
Figura 20 - Diagrama de Atividades do Cenário 1 .....	48
Figura 21 - Diagrama de Atividades do Cenário 2.....	49
Figura 22 - Diagrama de Atividades do Cenário 3.....	50
Figura 23 - Diagrama de Atividades do Cenário 4.....	51
Figura 24 - Diagrama de Classes da API DOLAM .....	52
Figura 25 – Esquema do cubo de dados OLAP Liberacoes .....	54
Figura 26 – Consulta Inicial da definição do SCI .....	55
Figura 27 - Exemplo de células expandidas .....	56
Figura 28 - Arquivo ARFF correspondente às células da Figura 27 .....	57
Figura 29 - Representação de célula no modo Tradicional e expandida pela arquitetura DOLAM .....	58
Figura 30 - Regras mais fortes do Cenário 0.....	59
Figura 31 – 10 regras de associação com 100% de suporte e 100% de confiança geradas pelo Cenário 1.....	59
Figura 32 - Comparação entre a quantidade e força das regras nos cenários tradicional e DOLAM .....	60
Figura 33 - Regras de associação geradas no Cenário 3 .....	62
Figura 34 - Exemplos de regras de associação geradas no cenário 3 .....	63
Figura 35 – Células outliers detectadas .....	66
Figura 36 - Células outliers expandidas .....	66
Figura 37 - Regras de associação geradas no Cenário 4 .....	67
Figura 38 - Exemplos de regras de associação geradas no cenário 4.....	68
Figura 39 - Hierarquia de Produtos .....	71
Figura 40 - Exemplo de base de dados .....	72
Figura 41 - Contagem dos elementos das transações da Figura 40 .....	73

Figura 42 - Hierarquia contendo atributos de interesse definidos pelo usuário .....	76
Figura 43 - Cubo de dados, incluindo dimensões, níveis e membros .....	79
Figura 44 - Exemplo de Hierarquia .....	81
Figura 45 - Hierarquia do cubo gerado.....	88
Figura 46 - Regras de associação geradas .....	89

## Índice de Tabelas

Tabela 1 - Exemplo de base de dados transacional .....	16
Tabela 2 - Contagem dos itens das transações da Tabela 1 .....	16
Tabela 3 - Representação transacional .....	29
Tabela 4 - Representação multidimensional.....	29
Tabela 5 – Mapeamento entre conceitos de bases de dados transacionais e cubos de dados OLAP .....	30
Tabela 6 - Exemplo de SCI .....	34
Tabela 7 - Exemplo de Subcubo.....	39
Tabela 8 - Células selecionadas por execução do cálculo de <i>outliers</i> de linha .....	40
Tabela 9 - Células selecionadas por execução do cálculo de outliers de coluna.....	41
Tabela 10 - Resultado da consulta inicial que define o SCI definido na Figura 18 .....	43
Tabela 11 - Primeiro subcubo gerado.....	46
Tabela 12 - Segundo subcubo gerado.....	46
Tabela 13 - Terceiro subcubo gerado .....	47
Tabela 14 - Resultado da consulta da Figura 26.....	55
Tabela 15 - SCI e outliers detectados .....	65
Tabela 16 - Subcubo resultante após uma operação roll-up na dimensão Tempo partindo do SCI.....	66
Tabela 17 - Base de dados agregada a ser minerada .....	75
Tabela 18 - Exemplo de regras de associação e suportes .....	85
Tabela 19 - Exemplo de dados do cubo.....	88
Tabela 20- Comparação entre os trabalhos.....	91

## Índice de Algoritmos

Algoritmo 1 - Algoritmo APriori .....	19
Algoritmo 2 - Construção da <i>FP-Tree</i> .....	22
Algoritmo 3 - Pseudo-código expandirAncestrais.....	33
Algoritmo 4 - Pseudo-código detectarOutliersLinha .....	38
Algoritmo 5 - Pseudo-código do método explorarSubcubos.....	44
Algoritmo 6 - Pseudo-código do método combinar2a2.....	44
Algoritmo 7 - Algoritmo básico .....	81
Algoritmo 8 - Algoritmo Cumulate .....	83

## **Índice de Quadros**

Quadro 1 - Exemplo de consulta para geração de regras de associação.....	75
Quadro 2 - Exemplo de meta-regra .....	79



## Principais Abreviaturas

<i>API</i>	<i>Application Programming Interface</i>
<i>ARFF</i>	<i>Attribute-Relation File Format</i>
<i>DAG</i>	<i>Directed Acyclic Graph</i>
<i>DM</i>	<i>Data Mart</i>
<i>DOLAM</i>	<i>Decoupled OLAM</i>
<i>DW</i>	<i>Data Warehouse</i>
<i>FP-Growth</i>	<i>Frequent Pattern Growth</i>
<i>FP-Tree</i>	<i>Frequent Pattern Tree</i>
<i>GUI</i>	<i>Graphical User Interface</i>
<i>IIQ</i>	<i>Intervalo Inter-quartil</i>
<i>JDBC</i>	<i>Java DataBase Connectivity</i>
<i>KDD</i>	<i>Knowledge Discovery in Databases</i>
<i>MDX</i>	<i>Multidimensional Expressions</i>
<i>ODBC</i>	<i>Open DataBase Connectivity</i>
<i>OLAM</i>	<i>OnLine Analytical Mining</i>
<i>OLAP</i>	<i>OnLine Analytical Processing</i>
<i>OLTP</i>	<i>OnLine Transaction Processing</i>
<i>OSF</i>	<i>Outlier Score Function</i>
<i>SCI</i>	<i>SubCubo de Interesse</i>
<i>SGBD</i>	<i>Sistema de Gerenciamento de Banco de Dados</i>
<i>SQL</i>	<i>Structured Query Language</i>
<i>UML</i>	<i>Unified Modeling Language</i>
<i>XML</i>	<i>eXtensible Markup Language</i>





# 1. Introdução

Este capítulo apresenta esta dissertação, destacando os motivos que levaram ao seu desenvolvimento e os seus objetivos. Por fim, é apresentada a estrutura em que os demais capítulos estão organizados.

## 1.1. Apresentação

Atualmente existe um grande volume de informação armazenado em bases de dados. No entanto, os tomadores de decisão normalmente não conhecem bem estes dados. Analisá-los não é uma atividade trivial, principalmente devido ao seu volume grande e crescente.

Neste contexto, as ferramentas de suporte à decisão são utilizadas para auxiliar os tomadores de decisão na atividade de análise e compreensão de grandes bases de dados. Estas ferramentas exibem os dados de forma amigável, como gráficos, modelos e sumarizações, o que permite ao tomador de decisão ter sua própria percepção [35].

As ferramentas OLAP (*Online Analytical Processing - OLAP*) [4] se encaixam nesta categoria. São ferramentas de processamento analítico de dados, capazes de realizar consultas contendo cruzamentos de dados (i.e., processamento multidimensional – por exemplo, vendas de produtos por fornecedor e tempo) e análise em diferentes níveis de agregação (i.e., processamento multinível - por exemplo, vendas de produtos por ano, semestre e mês) em uma base de dados chamada de cubo de dados ou cubo OLAP.

Ainda na categoria de ferramentas de suporte à decisão, encontram-se as ferramentas de mineração de dados [14], [7], [29]. O processo de mineração de dados pode ser descrito como o processo automático de extração de padrões potencialmente úteis e previamente desconhecidos, a partir de grandes bases de dados [14]. Dentre as ferramentas de mineração de dados, as voltadas para regras de associação detectam padrões que descrevem dependências significativas entre eventos que ocorrem juntos. Por exemplo, em 70% das vendas de fraldas em um supermercado, foi vendido também leite em pó. Apesar de essas ferramentas realizarem tarefas bem distintas, estas podem ser complementares. Neste sentido, recentemente estas ferramentas vem sendo usadas em conjunto, ao que se denomina de OLAM (*Online Analytical Mining*) [14], [24], [19]. As ferramentas OLAM utilizam a capacidade de exploração de dados multidimensional

e multinível das ferramentas OLAP, além de serem capazes de descobrir padrões nestes dados, como as ferramentas de mineração de dados.

Neste trabalho, as ferramentas de mineração de regras de associação foram escolhidas por se tratarem de ferramentas não supervisionadas e de caráter descritivo, que possibilitam compreensão mais simples que as ferramentas supervisionadas ou de caráter preditivo. No contexto de OLAM para regras de associação, ressalta-se que a maioria dos trabalhos consiste em adaptações do algoritmo de mineração de dados, para que este seja capaz de realizar algum processamento multidimensional e/ou multinível. Note que este tipo de abordagem gera uma solução OLAM específica para um algoritmo. Isto é, gera uma solução OLAM fortemente acoplada. Além disto, grande parte dos trabalhos de OLAM para regras de associação não considera o uso de um servidor OLAP, sendo necessária a implementação das operações OLAP em suas soluções OLAM. Este tipo de abordagem não é interessante, pois 1) além de onerar a solução (i.e., exige retrabalho para implementar as operações OLAP), 2) não faz uso de uma base de dados do tipo cubo OLAP, 3) não oferece garantias que a implementação contempla todas as operações oferecidas por um servidor OLAP e 4) não oferece garantias de que a implementação foi feita de forma correta. De forma a oferecer uma solução para o contexto em questão (i.e OLAM com regras de associação), este trabalho propõe a uma arquitetura desacoplada, denominada DOLAM (*Decoupled-OLAM*), que faz uso de um servidor OLAP para minerar regras de associação em cubos de dados.

Note que a partir da proposta da arquitetura DOLAM, o processo tradicional de KDD [7] passa a ter uma nova etapa de processamento de dados. Esta etapa fica entre as etapas de pré-processamento e transformação de dados e visa oferecer um conjunto de processamentos que irão enriquecer, com características multidimensionais e multiníveis, os dados a serem minerados. Assim, antes de serem minerados, os dados de um cubo OLAP são pré-processados por DOLAM de forma que a mineração se torna multidimensional e multinível sem a necessidade de alteração na etapa de mineração de dados. Ressalta-se que a arquitetura DOLAM foi implementada utilizando tecnologias abertas e extensíveis como a linguagem de programação Java, servidor OLAP *Mondrian* e a API OLAP4J.

## **1.2.      *Motivação***

São três as principais motivações para o desenvolvimento desta dissertação: 1) grande parte das propostas de OLAM para regras de associação não considera o uso de

um servidor OLAP, o que não é uma solução interessante (ver Seção 1.1) e 2) a maioria das abordagens de OLAM para regras de associação não tira proveito de todo o potencial multidimensional e multinível presentes nos cubos OLAP e 3) não foi encontrada nenhuma abordagem de OLAM para regras de associação que fosse desacoplada do algoritmo minerador, impedindo que adaptações feitas para mineração multidimensional e multinível sejam utilizadas com outros algoritmos de mineração.

### **1.3.      *Objetivos***

O objetivo geral deste trabalho é prover uma arquitetura de OLAM para regras de associação. Quanto aos objetivos específicos, deseja-se que esta arquitetura seja:

- desacoplada dos algoritmos de mineração de dados;
- capaz de gerar regras de associação multidimensional e multinível;
- capaz de considerar as especificidades existentes nos cubos OLAP, diferentemente de bases relacionais tradicionalmente mineradas;
- integrada com um servidor OLAP;
- capaz de detectar *outliers* presentes num cubo OLAP;
- flexível para explorar as abstrações presentes num subcubo e expandir seu escopo;
- implementada usando algoritmos de mineração e tecnologias de implementação consolidados e não-proprietários.

É importante ressaltar que o objetivo deste trabalho é prover as funcionalidades de: preparar os dados de cubos OLAP para mineração multidimensional e multinível, detectar *outliers* em um conjunto de células OLAP e expandir subcubos OLAP de forma exploratória. Ou seja, não é objetivo deste trabalho otimizar o desempenho de nenhuma destas funcionalidades.

### **1.4.      *Estrutura do Trabalho***

Visando atingir os objetivos citados na Seção 1.3, bem como apresentar alguns conceitos e tecnologias relacionados a este trabalho, os capítulos restantes desta dissertação estão organizados da seguinte maneira: o Capítulo 2 apresenta os conceitos básicos envolvidos neste trabalho; o Capítulo 3 contém a descrição detalhada da arquitetura DOLAM, proposta neste trabalho; o Capítulo 4 apresenta um estudo de caso

baseado nos algoritmos *FP-Growth* e *APriori*, bem como uma análise dos resultados; o Capítulo 5 discute os trabalhos relacionados a esta proposta; e por fim, o Capítulo 6 consiste nas conclusões e propostas de trabalhos futuros.

## 2. Conceitos Básicos

Neste capítulo serão apresentados os principais conceitos relacionados a *Data Warehouse*, processamento analítico de dados e mineração de dados usando regras de associação. Estes conceitos são fundamentais no entendimento dos próximos capítulos desta dissertação.

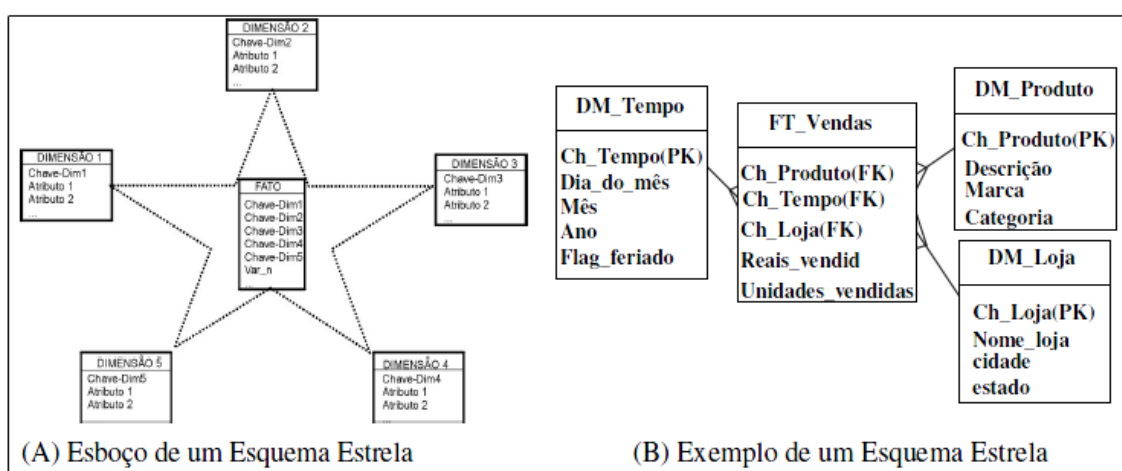
### 2.1. *Data Warehouse e Processamento Analítico de Dados*

*Data Warehouse* (DW) [20], [21], [22] é um banco de dados que integra as várias fontes de dados (banco de dados operacionais) de uma corporação e, por isso, funciona como um grande armazém de dados. A definição de DW mais utilizada na literatura é a de Inmon [20]: “*Data Warehouse* é uma coleção de dados orientada a assunto, integrada, não-volátil, e variante no tempo em suporte a decisões gerenciais”. A partir desta definição, pode-se resumir as principais características de um DW como: 1) orientado ao assunto – deve armazenar dados correspondentes aos fatos e não sobre as transações que geraram os fatos; 2) perfeitamente integrado – trata as heterogeneidades dos dados provenientes de bases localizadas nos diversos sistemas operativos/transacionais da organização, consolidando dados de diferentes origens; 3) não-volátil – os dados do DW raramente são modificados, limitando-se basicamente a cargas e consultas e 4) variante no tempo – o dado é armazenado conforme suas variações ao longo de uma linha de tempo, mantendo-se um histórico dos dados. Nesse contexto, o principal objetivo do DW é transformar o dado operacional, heterogêneo e distribuído pela organização, em informações limpas, estratégicas e consolidadas para suporte a processos de tomada de decisão.

Um DW pode ser dividido em bancos de dados menores denominados *Data Marts* (DM) [20], [21], [22]. Diferentemente dos DW, que correspondem a uma visão global dos dados de uma corporação, os DM são voltados para um assunto específico (e.g., um setor ou departamento), o que torna a sua implementação menos custosa do que a de um DW. Ressalta-se que um DW pode ser implementado a partir da integração dos diversos DM de uma corporação. Esta abordagem se mostra como uma opção com melhor custo-benefício do que construir diretamente o DW, principalmente pelo fato de que os DM

gerados podem funcionar como protótipos para construção incremental do DW [21], [22].

Para se representar os dados do DM comumente é adotado o modelo Estrela [20], [21], [22]. Esse modelo lógico possui tabelas de dois tipos específicos: 1) Tabela de Fatos, a qual possui todas as chaves primárias das dimensões e armazena as medidas numéricas do negócio (i.e., os fatos a serem consultados); e 2) Tabelas de Dimensões, as quais armazenam as descrições textuais sobre os fatos. Tabelas de dimensões são conectadas à tabela de fatos descrevendo uma estrutura de formato semelhante ao de uma estrela. A Figura 1, extraída de [9], ilustra um esboço e um exemplo de um esquema estrela.

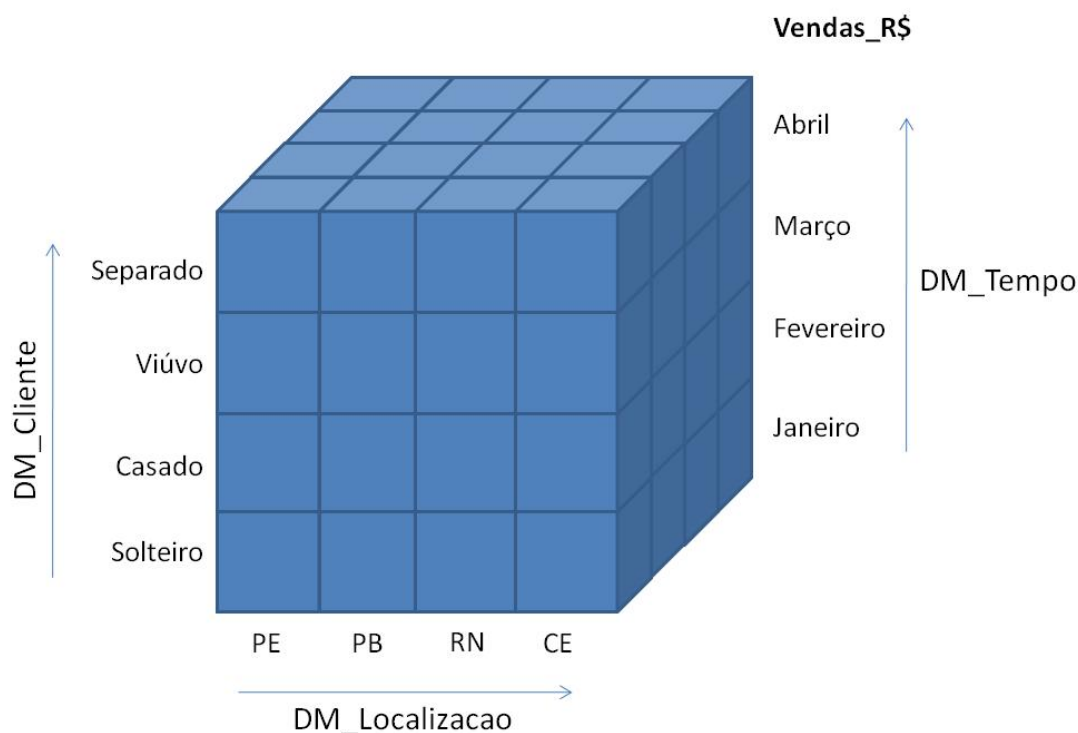


**Figura 1 - Esboço e exemplo de modelos estrela**

Várias ferramentas podem ser usadas para consultar os dados de um DW/DM. Uma dessas ferramentas é a de processamento analítico de dados (*ou Online Analytical Processing - OLAP*) [22], [33]. OLAP é uma ferramenta de consulta que permite realizar cruzamento (i.e., processamento multidimensional) e análise em diferentes níveis de agregação (i.e., processamento multinível) dos dados, preferencialmente, de um DW/DM. O total vendido de roupas femininas por loja nos últimos anos, semestres e meses, é um exemplo típico de consulta OLAP.

A ferramenta OLAP provê uma abstração para o seu banco de dados que é chamada de cubo de dados. O cubo contém sumarizações de um subconjunto dos dados presentes no DW/DM e pode ser visto como um *array* multidimensional. Alguns conceitos básicos sobre cubo de dados são [34]:

- Dimensão – corresponde a um dos eixos do cubo de dados. As dimensões contêm as informações descritivas sobre as quais se deseja realizar as análises. Por exemplo: Tempo, Localização, Cliente, Vendedor;
- Níveis – definem diferentes granularidades de análise para as informações descritivas de uma dimensão. Por exemplo: ano, bimestre, semestre, mês e trimestre;
- Hierarquia – ordena hierarquicamente os níveis de uma dimensão. Por exemplo, uma dimensão Tempo pode ter uma hierarquia que ordene seus níveis da seguinte forma: ano > semestre > trimestre > bimestre > mês;
- Membro – determina um item de dado de um nível de uma dimensão. Por exemplo, para uma dimensão Tempo pode-se ter os seguintes membros: janeiro, tri1 e 2009, os quais são itens de dados dos seguintes níveis: mês, trimestre e ano;
- Medida - corresponde a um atributo (não descritivo), o qual deseja-se analisar quantitativamente. Por exemplo, unidades vendidas, vendas em reais e quantidade de óbitos;
- Fato ou célula – corresponde ao valor de uma medida que é obtido a partir da interseção de membros de diferentes dimensões. Cada fato/célula reflete uma transação/evento registrado no DW/DM. Por exemplo, *mouse Microsoft* em 2009, vendeu 200 unidades;



**Figura 2 - Cubo de Dados**

A Figura 2 ilustra os conceitos descritos em um cubo de dados. Trata-se de um cubo com três dimensões: DM\_Localização, DM\_Cliente e DM\_Tempo. As células correspondentes à intersecção das dimensões contêm valores de medidas. No cubo ilustrado, a medida utilizada é Vendas\_R\$. Os membros ilustrados são:

- Para DM\_Localização: PE, PB, RN, CE (nível Estado)
- Para DM\_Cliente: Solteiro, Casado, Viúvo, Separado (nível Estado\_Civil)
- Para DM\_Tempo: Janeiro, Fevereiro, Março, Abril. (nível Mês)

A partir da estrutura multidimensional de um cubo, várias operações OLAP podem ser aplicadas para análise de dados. Dentre estas operações, podem-se destacar [4]:

- *Roll-up*: agregação ou generalização dos dados para um nível com maior granularidade. Por exemplo, ir de um nível mês para um nível semestre;
- *Drill-down*: desagregação ou especialização dos dados para um nível com menor granularidade. Esta operação é inversa ao *Roll-up*;
- *Slice*: projeção sobre o cubo de dados. Por exemplo, vendas de produtos por marca, ano e tipo de fornecedor;



- *Dice*: seleção sobre o cubo de dados. Por exemplo, vendas em PE, no ano de 2008 e por tipo de cliente;
- *Pivoting* - rotação dos eixos do cubo para visualização dos resultados de uma consulta. Por exemplo, venda de produtos por ano e tipo de cliente ou venda de produtos por tipo de cliente e ano.

A Figura 3, extraída de [31], exibe graficamente o resultado da aplicação dos operadores OLAP mencionados anteriormente.

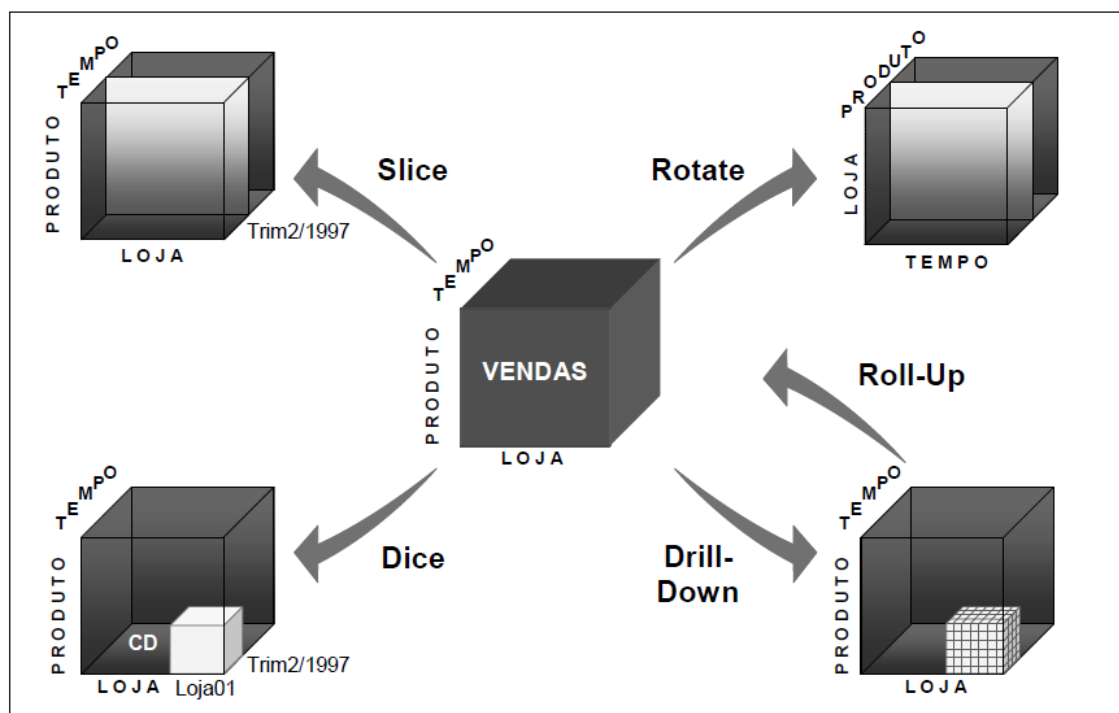


Figura 3 - Operadores OLAP

### 2.1.1.MDX

MDX (*Multidimensional Expressions*) [25] é uma linguagem de consulta para cubos OLAP, com funcionamento semelhante ao de SQL para bases de dados transacionais. MDX provê uma sintaxe especializada em consultas e manipulações de dados da forma multidimensional e multinível armazenados em cubos de dados OLAP e se tornou um padrão de mercado (padrão de fato) nesta área.

A sintaxe de MDX é baseada nas cláusulas *SELECT*, *FROM* e *WHERE* de SQL [6]. A Figura 4 [9] ilustra uma consulta MDX sobre um cubo Vendas para gerar um

subcubo com as unidades vendidas por região, por estado da loja e por tipo do produto. Os termos em maiúsculo na Figura 4 são palavras reservadas da linguagem MDX.

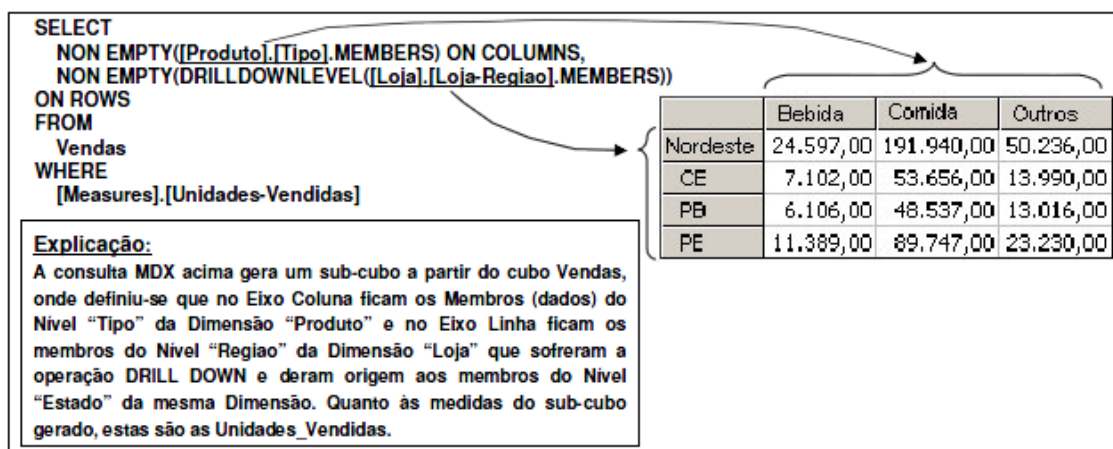


Figura 4 - Exemplo de uma declaração MDX e seu subcubo retornado

### 2.1.2. OLAP4J

OLAP4J [28] é uma interface de programação (*Application Programming Language – API*) que visa prover uma forma de acesso para bases de dados multidimensionais, sendo capaz de recuperar informações como dimensões, membros e níveis de cubos de dados OLAP. Assim, OLAP4J é uma forma de acesso a dados multidimensionais semelhante a JDBC para bases de dados transacionais. A especificação de OLAP4J é aberta e foi implementada em Java. Por ter sido escrita em Java, uma linguagem independente de plataforma, OLAP4J pode ser utilizada com qualquer servidor OLAP e as aplicações desenvolvidas com esta API podem ser migradas de um servidor OLAP para outro com facilidade.

OLAP4J prove um conjunto de métodos e classes capazes de fazer conexões com bases de dados multidimensionais (OLAP), realizar consultas, recuperar e manipular subcubos OLAP, acessar dimensões, membros e níveis da hierarquia de um cubo.

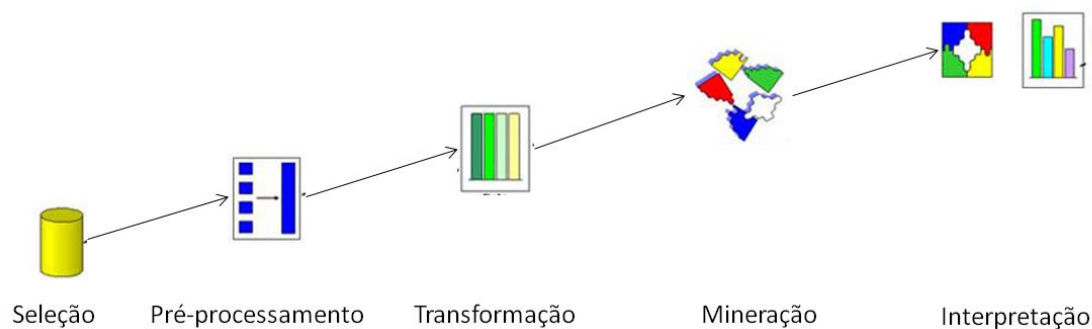
## 2.2. Mineração de Dados e Regras de Associação

Nas últimas décadas, com a adoção de sistemas corporativos por parte de um grande número de empresas, um enorme volume de dados vem sendo criado e

disponibilizado para análise. Entretanto, não é trivial analisar manualmente tamanho volume de dados. Por esta razão, faz-se necessário o desenvolvimento de ferramentas que detectem automaticamente padrões presentes em dados, com o objetivo de analisá-los e permitir tomar ações estratégicas baseadas em informações precisas, ou ainda de prever dados futuros baseando-se em informações atuais. Neste sentido, uma ferramenta que tem atraído a atenção da indústria e da academia nos últimos anos é a de mineração de dados (*data mining*) [7], [14], [29].

### **2.1.1 O processo de descoberta de conhecimento em bases de dados**

A descoberta de conhecimento em bases de dados (KDD – *Knowledge Discovery in Databases*) é um processo de identificação de padrões previamente desconhecidos e potencialmente úteis. Fayyad [7] definiu um conjunto de etapas no processo de KDD, conforme ilustra a Figura 5.



**Figura 5 – Processo de KDD**

O processo de KDD [7] é composto por seis fases. A ordem em que elas são executadas pode ser alterada, tornando o processo iterativo em espiral. As setas da Figura 5 indicam apenas as dependências mais importantes entre as fases.

Antes de se iniciar o processo de KDD, é importante que o usuário, utilizando seu entendimento sobre o domínio do negócio, defina com clareza os objetivos do processo de KDD. O profissional responsável pela descoberta de conhecimento deve compreender o domínio do negócio e os objetivos do usuário.

A etapa de seleção de dados determina o conjunto de dados alvo do processo de KDD, definido através do conhecimento obtido sobre o negócio e os objetivos do

usuário. Ou seja, para atender um objetivo específico, normalmente não é necessário analisar toda a base de dados.

A etapa de pré-processamento de dados envolve todas as atividades realizadas na construção da base final de dados a ser minerada, podendo incluir remoção de ruídos, tratamento de heterogeneidades, tratamento de informações em branco e padronização de formatos. Normalmente corresponde de 60% a 80% do tempo previsto para o projeto de mineração [37]. Segundo Pei et al [30], a chave para o sucesso na mineração de dados é uma boa preparação dos dados.

Após o pré-processamento dos dados, ocorre a etapa de Transformação. Nesta etapa os dados são transformados ou consolidados para formatos apropriados para a mineração de dados. Por exemplo, podem ser executadas operações de agregação, conversões entre formatos de dados e discretização.

A etapa de mineração de dados consiste na escolha da atividade (preditiva ou descritiva) e da tarefa de mineração de dados adequada aos objetivos. A tarefa de mineração de dados escolhida detecta padrões existentes na base de dados.

Finalmente, os padrões detectados na etapa de mineração de dados são analisados e interpretados, verificando-se sua relevância, precisão e adequação ao objetivo. Dependendo do resultado desta análise, pode ser necessário refazer algumas etapas anteriores.

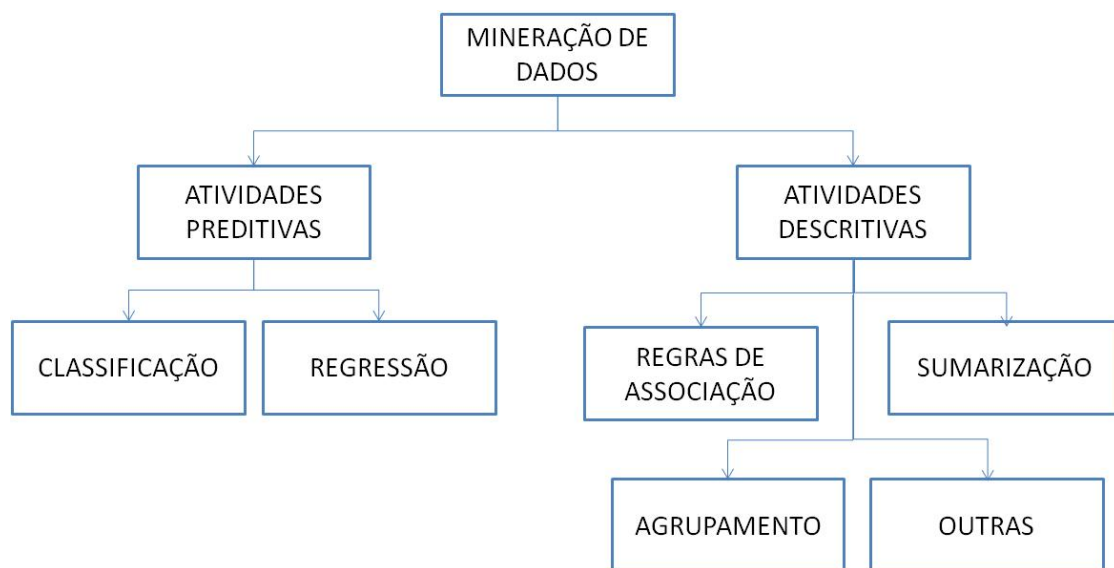
Note que as etapas de seleção, pré-processamento e transformação de dados do processo de KDD coincidem com os passos para a construção de um *data warehouse* (DW). Assim, um DW pode ser utilizado tanto para processamento analítico (OLAP) como para mineração de dados, como ainda para processamento OLAM (*On Line Analytical Mining*).

De acordo com [14], a mineração de dados possui dois grandes grupos de atividades:

- 1) Preditivas: busca de um modelo de conhecimento que permita, a partir de um histórico dos dados, a previsão dos valores de determinados atributos em novas situações;

- 2) Descritivas: busca por um modelo que descreva, de forma compreensível, o conhecimento existente em um conjunto de dados.

Para cada uma dessas atividades existem tarefas de mineração de dados associadas a ela, onde cada tarefa de mineração de dados deve ser entendida como um tipo de problema de descoberta de conhecimento a ser solucionado. As principais tarefas de mineração para atividades preditivas [14] são classificação [14], [7], e regressão [7], [29]. Para atividades descritivas, se destacam a descoberta de associações [2], [1], sumarização [2] e agrupamento (*clustering*) [3], [29], [7], [14]. A Figura 6 resume esta discussão, relacionando as atividades com as suas principais tarefas. Ressalta-se que, independentemente da tarefa, a mineração de dados pode ser realizada sobre um banco de dados operacional (i.e., arquivos ou tabelas) ou sobre um DW/DM. Neste último caso, dada a visão integrada oferecida pelo DW/DM, geralmente, os resultados obtidos são melhores.



**Figura 6 - Atividades de Mineração de Dados**

Dentre as tarefas descritivas, as regras de associação são consideradas uma das tarefas de mineração de dados mais importantes e tem atraído grande atenção tanto da academia, quanto da indústria.

De acordo com [7], a descoberta de regras de associação consiste em encontrar padrões que descrevam dependências significativas entre eventos que ocorrem juntos. Assim, seu objetivo é determinar o quanto a presença de um determinado conjunto de itens nos registros de uma base de dados implica na existência de um outro conjunto

distinto de itens nesta base. As regras de associação surgiram com o objetivo de analisar transações como compras feitas em supermercados (*market-basket analysis*). O objetivo deste tipo de análise é obter regras do tipo “um cliente que compra os produtos  $x_1, x_2$  e  $x_3$  irá comprar também os produtos  $y_1$  e  $y_2$  com probabilidade de  $z\%$ .”

Atualmente regras de associação são aplicadas aos mais diversos contextos, como contratos de seguros, geoprocessamento, padrões de uso da Web (*web usage mining*), detecção de intrusos e bioinformática [14]. Uma regra de associação é representada como uma implicação da forma  $LHS \rightarrow RHS$ , onde  $LHS$  e  $RHS$  são definidos respectivamente como o antecedente (*Left Hand Side*) e o conseqüente (*Right Hand Side*) da regra. Uma regra  $LHS \rightarrow RHS$  implica “se  $LHS$  então  $RHS$ ”, onde  $LHS$  e  $RHS$  são conjuntos disjuntos de itens. Um modelo com base formal para especificar regras de associação foi proposto por Agrawal [2]. Este modelo é apresentado a seguir.

Seja  $D$  uma base de dados transacional composta por um conjunto de itens (atributos binários)  $I = \{i_1, i_2, \dots, i_m\}$  e por um conjunto transações  $T = \{t_1, t_2, \dots, t_n\}$ , onde cada transação  $t_i$  é composta por um conjunto de itens (*itemset*) tal que  $t_i \subset I$ . Cada transação é associada a um identificador único ( $T_{ID}$ ). Seja  $X \subset I$  um conjunto de itens. Uma transação contém  $X$  se  $X \subset t_i$ . Sendo  $X$  o  $LHS$  de uma regra, uma regra de associação é uma implicação da forma  $LHS \rightarrow RHS$ , em que  $LHS \subset I$ ,  $RHS \subset I$ ,  $LHS \cap RHS = \emptyset$ . A regra  $LHS \rightarrow RHS$  ocorre em um conjunto de transações  $T$  com suporte  $sup$  se em  $sup\%$  das transações de  $D$  ocorre  $LHS \cup RHS$ , ou seja, se  $sup\%$  das transações em  $D$  contem  $X$  e  $Y$ . Esta regra possui confiança  $conf$  se em  $conf\%$  das transações em que ocorre  $LHS$ , também ocorre  $RHS$ . Estas métricas são detalhadas nos parágrafos seguintes.

Obviamente, se dois produtos fazem parte da mesma transação em pelo menos uma ocorrência na base de dados, existe uma regra associando estes produtos. No entanto, é necessário verificar a força desta regra, ou seja, quão representativa ela é. Para avaliar a força de uma regra, as medidas mais usadas são suporte e confiança.

O suporte quantifica a porção das transações que satisfazem a regra. Ou seja, mede a relevância da regra no contexto de todo o conjunto de dados. O suporte é a

probabilidade de uma transação D conter  $LHS \cup RHS$ . Sendo assim, o suporte indica a frequência com que o conjunto formado por  $LHS \cup RHS$  ocorre no conjunto total de dados. O suporte de uma regra  $LHS \rightarrow RHS$  é definido por:

$$sup(LHS \cup RHS) = \frac{quantidade(LHS \cup RHS)}{Tamanho(D)}$$

ou

$$sup(LHS \rightarrow RHS) = sup(LHS \cup RHS) = \frac{n(LHS \cup RHS)}{N}$$

onde  $n(LHS \cup RHS)$  é o número de transações nas quais  $LHS$  e  $RHS$  ocorrem simultaneamente e  $N$  é a quantidade total de transações no conjunto de dados minerado.

A regra  $LHS \rightarrow RHS$  é válida no conjunto das transações D com confiança  $conf\%$  se  $conf\%$  das transações do banco de dados que contem  $LHS$  também contem  $RHS$ . A confiança é a probabilidade de  $RHS$  ocorrer em uma transação de D visto que  $LHS$  ocorre. A métrica confiança tem como objetivo verificar a força da implicação da regra. Ela indica a frequência com que  $LHS$  e  $RHS$  ocorrem juntos em relação ao total de transações em que  $LHS$  ocorre. A confiança é representada por:

$$conf(LHS \rightarrow RHS) = \frac{número\ de\ registros\ que\ contem\ a\ associação\ LHS \rightarrow RHS}{número\ de\ registros\ que\ contem\ LHS}$$

ou

$$conf(LHS \rightarrow RHS) = \frac{sup(LHS \cup RHS)}{sup(LHS)} = \frac{n(LHS \cup RHS)}{n(LHS)}$$

onde  $n(LHS)$  representa o número de ocorrências de  $LHS$  no conjunto de dados.

De forma sucinta, o suporte representa a frequência com que o padrão ocorre no conjunto de dados, e a confiança representa a força da implicação da regra. O usuário que deseja localizar regras de associação em um conjunto de dados define valores

mínimos de suporte e confiança. Estes valores expressam a representatividade (suporte) e força (confiança) das regras interessantes ao usuário.

Por exemplo, considere o conjunto de transações ilustrado na Tabela 1. Para calcular o suporte e confiança presente nestas transações, deve-se primeiramente contar a ocorrência dos itens nas transações. O resultado desta contagem é exibido na Tabela 2.

T1	T2	T3	T4	T5
Pão Café Leite	Leite Queijo Pão	Leite Pão Tomate	Pão Café Torrada	Bucha Esponja

**Tabela 1 - Exemplo de base de dados transacional**

	T1	T2	T3	T4	T5
Pão	1	1	1	1	0
Café	1	0	0	1	0
Leite	1	1	1	0	0
Queijo	0	1	0	0	0
Tomate	0	0	1	0	0
Torrada	0	0	0	1	0
Bucha	0	0	0	0	1
Esponja	0	0	0	0	1

**Tabela 2 - Contagem dos itens das transações da Tabela 1**

As colunas da Tabela 2 representam as transações e as linhas representam os itens das transações. A presença do valor “1” no cruzamento de uma transação T com um item I indica que o item I pertence à transação T, enquanto que o valor “0” denota sua ausência. Por exemplo, T1, que contém Pão, Café e Leite, é definida como {1,1,1,0,0,0,0}. A frequência de ocorrência dos itens nas transações é contada a partir da Tabela 2. Por exemplo, uma transação que contém os itens Pão e Leite deve conter ao



menos o *string* {1,0,1,0,0,0,0,0}. Na Tabela 2, as transações T1, T2 e T3 contêm ao menos este *string*. Portanto, num total de 5 transações, onde 3 delas possuem {Pão,Leite}, o suporte (Pão,Leite) =  $3/5 = 60\%$ . Assim o suporte indica a representatividade de ocorrência dos itens na base. Já o conceito de confiança denota a força da implicação. Por exemplo, T5 contém esponja e bucha, e é a única transação que contém esponja. Assim, 100% das transações que contêm esponja também contêm bucha. Assim, a confiança da regra Esponja  $\rightarrow$  Bucha é de 100%.

### ***Processo de geração de regras de associação***

A tarefa de localizar as regras de associação presentes num conjunto de dados pode ser dividida em duas sub-tarefas.

Define-se *itemset* como um conjunto de itens ordenados lexicograficamente (em ordem alfabética). Um *k*-itemset consiste em um *itemset* com *k* itens. Um *k*-itemset frequente é um *k*-itemset com suporte maior que o valor de suporte mínimo definido pelo usuário (*min\_sup*).

A tarefa de localizar as regras de associação presentes num conjunto de dados pode ser dividida em duas sub-tarefas:

1. Encontrar todos os *k*-itemsets frequentes, ou seja, todos os conjuntos que possuam suporte maior que o valor mínimo definido pelo usuário (*min\_sup*);
2. A partir dos *k*-itemsets frequentes, com  $k \geq 2$ , gerar regras de associação. Para cada *itemset* frequente  $S \subseteq I$ , encontrar todos os subconjuntos  $\tilde{I}$  de itens de *S*.

A geração das regras de associação a partir dos *itemsets* frequentes é um problema combinatorial. A maioria dos trabalhos sobre geração de regras de associação, na realidade, tem como objetivo otimizar a busca dos *itemsets* frequentes (primeira etapa).

#### ***2.2.1.1. Algoritmo APriori***

O algoritmo *APriori*, criado por Agrawal e Srikant [2] em 1993, tem como objetivo a geração de regras de associação entre itens contidos em uma base de dados.

Para tal, o algoritmo define dois conceitos: o de conjuntos *itemsets* freqüentes e o de conjuntos *itemsets* candidatos. Conjuntos de *itemsets* freqüentes são aqueles cujos valores de suporte e confiança superam os limites definidos pelo usuário. Conjuntos de *itemsets* candidatos consistem em *itemsets* candidatos a serem freqüentes. Ou seja, trata-se de *itemsets* potencialmente freqüentes, que devem ser explorados para verificar se de fato são freqüentes (i.e., alcançam o suporte mínimo). Na análise de um *itemset* candidato  $I$ , define-se que, se  $I$  não satisfaz o suporte mínimo  $min\_sup$ , então  $I$  não é freqüente. Pois,  $sup(I) < min\_sup$ .

Após gerar um conjunto de *itemsets* candidatos, o algoritmo percorre novamente a base para determinar se estes são freqüentes. Isto é feito comparando-se o suporte dos *itemsets* candidatos com o suporte mínimo definido.

No início do processo de mineração, todos os *itemsets* que podem ser gerados a partir de um conjunto  $I$  de itens distintos são considerados potencialmente freqüentes, ou seja, todos os subconjuntos de  $I$  devem ser calculados (todos os  $X \subset I$ ). Estes subconjuntos, com exceção do conjunto vazio, compõem o espaço de busca do minerador. Mesmo para conjuntos de itens  $I$  pequenos, o espaço de busca resultante é potencialmente grande, devido às possibilidades de combinações entre os itens.

Para evitar este problema, o algoritmo *APriori* define a “propriedade *APriori*” (*APriori property*) [2]. A propriedade *APriori* define que, se um item  $A$  é adicionado a um *itemset*  $I$  não freqüente, o *itemset* resultante  $I \cup A$  não ocorrerá mais frequentemente do que  $I$ . Portanto,  $I \cup A$  também não é freqüente, pois  $sup(I \cup A) \leq min\_sup$ . Por exemplo, seja  $X$  um *itemset* candidato formado pelo conjunto de itens  $\{a,b,c\}$ . Se, após análise, verifica-se que  $X$  não é um *itemset* freqüente, então os *itemsets* formados por  $\{a,b,c,d\}$ ,  $\{a,b,c,e\}$ ,  $\{a,b,c,d,e\}$ , ou qualquer outro conjunto que contenha  $\{a,b,c\}$ , também não será freqüente. Isto porque a frequência de ocorrência (suporte) deles será no máximo igual à de  $X$ . Como o suporte de  $X$  não foi suficientemente alto, então o suporte dos demais *itemsets* relacionados também não o será. Por este motivo, o algoritmo *APriori* trabalha de forma incremental, derivando  $k$ -*itemsets* a partir de  $(k - 1)$ -*itemsets* freqüentes.

### Algoritmo *APriori*

**Requer:** Uma base de dados  $D$  composta por um conjunto de itens  $A = \{a_1, \dots, a_m\}$  ordenados lexicograficamente e por um conjunto de transações  $T = \{t_1, \dots, t_n\}$  na qual cada transação  $t_i \in T$  é composta por um conjunto de itens tal que  $t_i \subseteq A$ .

1.  $L_1 := \{1\text{-itemsets frequentes}\};$
2. **for** ( $k:=2; L_{k-1} \neq \emptyset; k++$ ) **do**
3.    $C_k := \text{apriori-gen}(L_{k-1})$  //Gera novos conjuntos candidatos
4.   **for all** (transações  $t \in T$ ) **do**
5.      $C_t := \text{subset}(C_k, t)$  ; //Conjuntos candidatos contidos em  $t$
6.     **for all** candidatos  $c \in C_t$  **do**
7.        $c.\text{count} ++;$
8.     **end for**
9.   **end for**
10.    $L_k := \{c \in C_k \mid c.\text{count} \geq \text{sup-min}\};$
11. **end for**
12. Resposta  $:= \cup_k L_k;$

### Algoritmo 1 - Algoritmo *APriori*

O algoritmo *APriori* é apresentado no código do Algoritmo 1. Nele, a notação  $C_k$  representa o conjunto de  $k$ -itemsets candidatos e  $L_k$  representa o conjunto de  $k$ -itemsets frequentes. Inicialmente o algoritmo conta a ocorrência de itens, ou seja, de 1-itemsets. Após verificar quais 1-itemsets são frequentes, estes são armazenados em  $L_1$ .

O algoritmo *APriori* funciona de forma incremental, onde a iteração (passo) seguinte utiliza o resultado da iteração (passo) anterior. Assim, o conjunto dos 1-itemsets é utilizado na criação dos 2-itemsets. Na linha 3 do código apresentado no

Algoritmo 1, o conjunto de *itemsets* freqüentes obtidos no passo  $(k - 1)$  é utilizado para gerar o conjunto de  $k$ -*itemsets* candidatos  $C_k$  usando a função *apriori-gen*. O laço *for* presente na linha 2 mostra que esta derivação ocorre para cada  $k$ . A seguir, o algoritmo percorre a base de dados (linhas 4 a 9), gerando assim o conjunto de  $k$ -*itemsets* candidatos  $C_k$  (função *subset* presente na linha 5) e o valor do suporte de cada um deles. Aqueles que alcançam o suporte mínimo (linha 10) são considerados freqüentes e adicionados a  $L_k$ , o conjunto de  $k$ -*itemsets* freqüentes. Em seguida, o valor de  $k$  é incrementado e são geradas

O processo acima ocorre novamente até que todos os valores possíveis de  $k$  sejam explorados. Em seguida, os  $k$ -*itemsets* freqüentes gerados pelo algoritmo *APriori* são utilizados como entrada para um algoritmo que gera regras de associação a partir de *itemsets*.

Analisando-se o algoritmo *APriori* é possível perceber que ele demanda diversas iterações sobre a base de dados para criar o conjunto de *itemsets* candidatos. O processo de geração de conjuntos candidatos é significativamente custoso em termos de tempo e recursos computacionais. Além disto, sua performance é degradada [17] ao trabalhar com limites de suporte baixo e padrões longos ou em grande número. Dada esta constatação, Han et al [17] propuseram o algoritmo *Frequent Pattern Growth (FP-Growth)* - ver Seção 2.2.1.2), o qual obtém o conjunto de *itemsets* freqüentes sem gerar o conjunto de *itemsets* candidatos.

### 2.2.1.2. Algoritmo *FP-Growth*

O algoritmo *FP-Growth* [17],[16] utiliza uma estrutura compacta e funcional, a *FP-Tree (Frequent Pattern Tree)*, para armazenar o conjunto dos 1-*itemsets* freqüentes e evitar varreduras repetitivas da base de dados. O algoritmo requer apenas duas varreduras da base de dados, conforme detalhado nos próximos parágrafos.

Assim como o algoritmo *APriori*, o algoritmo *FP-Growth* gera, primeiramente, o conjunto dos 1-*itemsets* freqüentes. O armazenamento destes 1-*itemsets* é feito numa estrutura denominada *FP-Tree*. Se várias transações compartilham um conjunto de itens, eles podem ser agrupados nos mesmos nós da árvore *FP-Tree*, que registram junto ao nó o número de ocorrências em uma variável *count*.

Para se criarem regras de associação, devem-se verificar quais transações compartilham itens frequentes. Esta comparação se torna mais fácil se os itens estiverem ordenados conforme um critério pré-fixado. Após esta ordenação, se duas transações compartilham um prefixo, este prefixo pode ser armazenado em apenas uma estrutura, contabilizando-se os valores de frequência de ocorrência de cada item. No algoritmo *FP-Growth*, o critério de ordenação definido foi o suporte dos itens frequentes (que pode ser expresso por seu *count*, numa ordenação descendente, pois esta ordem aumenta as chances de compartilhamento de prefixo).

Dado um banco de dados transacional e um suporte mínimo, a *FP-Tree* correspondente contém toda informação necessária para a mineração das regras de associação nesta base, conforme demonstrado em [17]. A seguir são apresentados o algoritmo de construção da *FP-Tree* e de derivação das regras de associação a partir da árvore.

#### **Algoritmo de construção de FP-Tree**

**Entrada:** Base de transações DB e limite mínimo de suporte *min\_sup*

**Saída:** FP-Tree, a árvore de padrões frequentes de DB

A *FP-Tree* é construída da seguinte forma:

1. Percorra a base de transações DB uma vez. Colete F, o conjunto de itens frequentes, e o suporte de cada item frequente. Crie *FList*, a lista de itens frequentes, que recebe F ordenada em ordem descendente de suporte.
2. Crie a raiz da árvore *FP-Tree*, T, cujo valor é *null*

Para cada transação *Trans* em DB, faça o seguinte:

Selecione os itens frequentes em *Trans* e os ordene de acordo com a ordem em *FList*.

Seja [p|P] a lista ordenada de itens frequentes em *Trans*, onde p é o primeiro elemento e P é o resto da lista. Chame *insert\_tree*([p|P],T).

A função *insert\_tree*([p|P],T) é executada da seguinte forma.

Se T tem um filho N tal que  $N.item\_name = p.item\_name$ , então incremente o *count* de N em 1;

Senão crie um novo nó N com seu contador inicializado em 1 e cujo nó pai é ligado a T, e link é ligado aos nós com mesmo *item\_name* através da estrutura de *node\_link*.

Se P não é vazia, chame *insert\_tree*(P,N) recursivamente.

### Algoritmo 2 - Construção da *FP-Tree*

A construção da *FP-Tree* consiste em uma série de passos, ilustrados no Algoritmo 2. Primeiramente, a base de dados *DB* é varrida, coletando-se os *1-itemsets* freqüentes e seu respectivo suporte, que são armazenados na lista *F*. A seguir os itens desta lista são ordenados em ordem descendente do suporte.

A raiz da árvore *T* é então criada, e recebe o valor *null*. Para cada transação *Trans* em *DB*, seus itens devem ser ordenadas de acordo com a lista *F*. Seja a lista de itens freqüentes em *Trans*[*p*|*P*], onde *p* é o primeiro elemento e *P* o restante da lista. Então, chama-se *insert\_tree*(*[p|P]*, *T*).

A função *insert\_tree*(*[p|P]*, *T*) comporta-se da seguinte forma: se *T* possui um filho *N* tal que o valor de *T* é igual ao valor de *p*, então o valor do *count* de *N* é incrementado. Se não, um novo nó *N* é criado, o valor inicial de seu *count* é 1, e ele é posicionado como filho de *T*. Esta função é chamada recursivamente enquanto houverem itens a serem inseridos na árvore, ou seja, enquanto *P* não é vazio. A saída do algoritmo é a árvore *FP-Tree T* contendo os padrões de *DB*.

Para a construção da *FP-Tree* são necessárias, portanto, apenas duas varreduras da base de dados *DB* (uma para detectar os *1-itemsets* freqüentes e outra para preenchimento dos nós da árvore). O custo para inserir uma transação *Trans* em *FP-Tree* é  $O(|Trans|)$ , onde  $|Trans|$  é o número de itens freqüentes em *Trans*.

Desconsiderando-se a raiz, o tamanho de uma *FP-Tree* é limitado ao total de ocorrências dos itens freqüentes em *DB* e sua altura é limitada pela transação com maior número de itens freqüentes. Após a geração da *FP-Tree*, a derivação das regras de associação pode ser feita simplesmente enumerando-se todas as combinações entre os itens no caminho da árvore, com o suporte sendo o valor mínimo entre os itens contidos na combinação.

Foi provado [17] que o algoritmo *FP-Growth* é mais eficiente computacionalmente que o algoritmo *APriori*.

### 2.3. OLAM (OLAP Mining)

As ferramentas OLAP, apesar de serem poderosos instrumentos de auxílio à tomada de decisão, não são capazes de extrair conhecimento dos dados, como a descoberta de padrões e relacionamentos. Para este propósito, as técnicas de mineração de dados são as ferramentas mais apropriadas [7], [14]. Estas técnicas são tradicionalmente aplicadas a tabelas planas, sem relacionamentos (i.e., *flat files*). Por este motivo, se não houver nenhum pré-processamento, elas não são capazes de lidar com aspectos multidimensional e multinível dos cubos de dados. Por consequência, padrões potencialmente relevantes não são descobertos.

Por exemplo, considere a venda de três laptops, sendo um da marca *Compaq*, um HP e um *Toshiba*. Em tabelas planas, a informação de cada produto vendido consiste apenas de suas descrições, as quais, apesar de poderem ter um esquema bem definido, não têm a definição explícita de níveis hierárquicos, como ilustra a Figura 7-A. Num pior caso (Figura 7-B), as tabelas planas podem condensar algumas informações, desprezando o esquema que as descreve.

Produto			
Marca	Tipo	Sub-tipo	Categoria
Compaq	Computador	Laptop	Eletrônico
HP	Computador	Laptop	Eletrônico
Toshiba	Computador	Laptop	Eletrônico

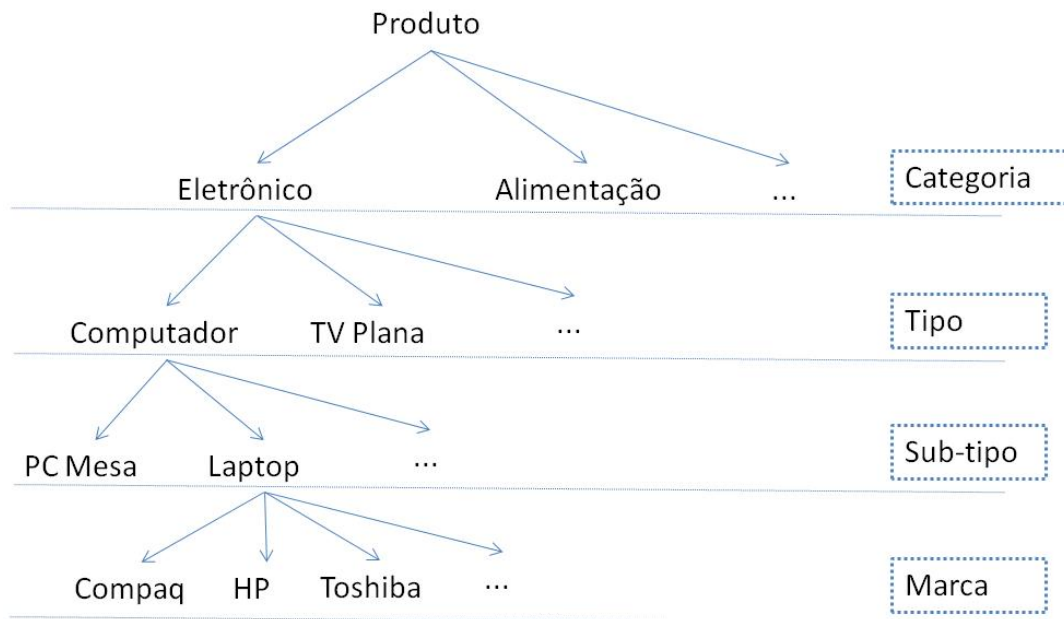
A – Tabela plana com esquema bem definido

Produto			
Compaq Computador Laptop Eletrônico			
HP Computador Laptop Eletrônico			
Toshiba Computador Laptop Eletrônico			

B – Tabela plana sem esquema bem definido

**Figura 7 - Exemplo de representação de uma tabela plana com e sem esquemas bem definidos**

Diferente de uma tabela plana, as informações (i.e. membros) de um cubo de dados são, intrinsecamente, organizadas em níveis hierárquicos. Na Figura 8, é exibida uma hierarquia dos membros de uma dimensão Produto, onde estes são organizados em níveis (destacados nos retângulos pontilhados).



**Figura 8 - Exemplo de Hierarquia da dimensão Produto**

A área OLAM (*OLAP Mining*) visa unir esforços nas áreas de mineração de dados e processamento OLAP para obter melhores resultados. Com o objetivo de utilizar plenamente a flexibilidade da informação presente nos cubos OLAP, os métodos de mineração de dados vêm sendo adaptados e aplicados a bases multidimensionais. Esta adaptação deve ser capaz de utilizar o aspecto multidimensional e multinível presentes nos cubos OLAP.

As técnicas de mineração de dados irão descobrir um conjunto  $P$  de padrões, onde cada padrão  $p_i$  é analisado quanto ao seu aspecto multidimensional (inter-dimensional ou intra-dimensional) e multinível (inter-nível ou intra-nível).

Para avaliar o aspecto multidimensional e multinível de um padrão  $p_i$  que possui ao menos dois membros  $l_a$  e  $l_b$ , verifica-se primeiramente se  $l_a$  e  $l_b$  pertencem à mesma dimensão ou a dimensões diferentes. Se os membros  $l_a$  e  $l_b$  pertencem a dimensões diferentes, o padrão  $p_i$  é classificado quanto ao seu aspecto multidimensional como inter-dimensional. Como cada dimensão possui uma hierarquia própria e os membros  $l_a$  e  $l_b$  pertencem a dimensões diferentes, logo eles pertencem a hierarquias (e níveis) diferentes. Portanto, quanto ao aspecto multinível, o padrão  $p_i$  é inter-nível. Se os



membros  $t_a$  e  $t_b$  pertencem à mesma dimensão, o padrão  $p_i$  é classificado quanto ao seu aspecto multidimensional como intra-dimensional. Segundo a hierarquia da dimensão de  $t_a$  e  $t_b$ , deve-se avaliar se eles pertencem ao mesmo nível ou a níveis diferentes. Se ambos os membros pertencerem ao mesmo nível da hierarquia, quanto ao aspecto multinível, o padrão  $p_i$  é classificado como intra-nível. Do contrário, é classificado como inter-nível. Assim, existem três tipos de padrões, a saber: 1) inter-dimensional inter-nível, 2) intra-dimensional intra-nível e 3) intra-dimensional inter-nível. Desta forma, deseja-se que os algoritmos mineradores sejam capazes de gerar padrões inter e intra-nível, assim como inter e intra-dimensionais.

## 2.4. *Outlier mining*

Estatisticamente, um *outlier* é definido como um ponto de observação numericamente distante dos demais dados. Grubbs [12] define *outlier* como uma observação que parece ter sido propositalmente desviada dos outros membros da amostra que os contém. Hawkins [18] afirma que *outliers* são tão diferentes das demais observações que levantam dúvidas se foram gerados por um mecanismo diferente.

*Outliers* podem ocorrer por acaso em qualquer distribuição, mas normalmente indicam:

- erros de medição ou
- falhas ou modificações no sistema gerador de dados ou
- comportamento fraudulento.

Cabe ao analista identificar em qual destas situações o caso estudado se aplica. Caso seja identificado erro de medição, normalmente os registros de *outliers* são descartados, ou usam-se métodos estatísticos que são robustos a *outliers*. Neste último caso, deve-se tomar muito cuidado ao se utilizar ferramentas que assumam distribuição normal dos dados. A detecção de *outliers* é importante em várias aplicações, como em detecção de fraude de crédito, detecção de intrusos em redes, análises de mercado, sistemas de monitoramento e em análises de eventos raros em geral. A mineração de *outliers* (*outlier mining*) [23] aplica técnicas de mineração de dados aos *outliers* detectados numa massa de dados, extraíndo padrões existentes nos *outliers*. Através dos padrões de ocorrência de *outliers*, os analistas podem verificar suas causas. Por exemplo, cada cliente de uma operadora de cartão de crédito possui um perfil de

compras. Quando é feita uma compra muito diferente do perfil do cliente (ou seja, quando ocorre um *outlier*), as operadoras normalmente bloqueiam a operação até que ela seja confirmada junto ao titular do cartão. Suponha que uma compra se trate de uma fraude, onde o cartão de um cliente foi clonado. Neste caso, a detecção do *outlier* tornou possível a detecção da fraude. Provavelmente mais clientes foram vítimas da mesma fraude. A operadora de cartão de crédito, juntamente com a polícia local, pode então realizar *outlier mining* a partir das compras detectadas como fraudes. Assim, são identificados padrões existentes entre as compras fraudulentas, sendo possível extrair informações potencialmente úteis na identificação dos autores da fraude.

De forma semelhante, no contexto da análise de incidentes criminosos [23], a mineração de *outliers* provê relações entre os crimes, de forma a facilitar a identificação de seus autores. Por exemplo, em várias ocorrências de roubo, o valor preenchido para o campo “arma” é de “arma de fogo”. Não se pode assumir que eles foram cometidos pelo mesmo criminoso apenas por causa do valor deste campo. No entanto, se houver registros de vários incidentes cometidos usando-se uma arma incomum, “espada japonesa”, por exemplo, é mais provável que estes crimes tenham sido cometidos pela mesma pessoa. Como “espada japonesa” não é um registro comum, isto a torna mais facilmente distinguível das outras armas, por exemplo, “arma de fogo”. Utilizando *outliers*, o trabalho é capaz de relacionar registros de crimes, além de distinguir um pequeno grupo de registros dos demais incidentes.

Existem diversas formas de identificação de *outliers*. Algumas são baseadas em modelos matemáticos que assumem que os dados seguem uma distribuição normal e utilizam as métricas de média e desvio-padrão para localizar os *outliers*. Outras abordagens utilizam a medida estatística de intervalo inter-quartil.

Para calcular esta medida, os dados são ordenados e divididos em quatro partes com a mesma quantidade de dados. Os quartis correspondem ao valor que divide as quatro partes, como mostra a Figura 9. Assim, o segundo quartil (Q2) equivale à mediana. O primeiro quartil equivale ao limite dos 25% menores dados, e o terceiro quartil equivale ao limite dos 75% maiores dados. O intervalo inter-quartil (IIQ) corresponde ao intervalo entre Q1 e Q3.

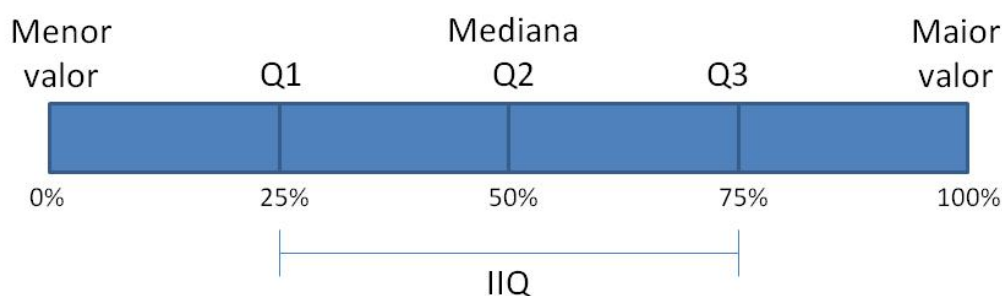


Figura 9 - Quartis de uma distribuição de dados

Um valor é considerado *outlier* se ele estiver fora do intervalo formado por  $[Q1 - k(Q3 - Q1), Q3 + k(Q3 - Q1)]$ , para uma constante  $k$ . Normalmente define-se que para *outlier* suave como  $k=1,5$  e para um *outlier* extremo,  $k=3$ . A Figura 10 exibe um gráfico box-plot com uma representação gráfica da localização dos *outliers* em uma distribuição. Os valores marcados como *outliers* suaves são 1,5 vezes superiores a  $(Q3 + IIQ)$  ou 1,5 vezes inferiores a  $(Q1 - IIQ)$ . Os *outliers* extremos são 3 vezes superiores a  $(Q3 + IIQ)$  ou 3 vezes inferiores a  $(Q1 - IIQ)$ .

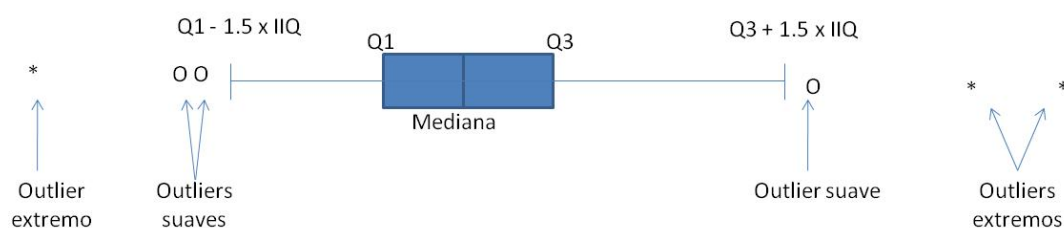


Figura 10 - Representação gráfica de outliers

## 2.5. Conclusão

Neste capítulo foram apresentados os conceitos básicos que fundamentam esta dissertação. Inicialmente, foram apresentados os conceitos de *Data Warehouse* e OLAP, a linguagem MDX e a API OLAP4J. Em seguida, conceitos de mineração de dados, regras de associação e os algoritmo *APriori* e *FP-Growth* foram apresentados. Por fim, a área OLAM, que contextualiza esta dissertação, é discutida em seguida, juntamente com a área de *outlier mining*.

### 3. Arquitetura DOLAM

Este capítulo apresenta a arquitetura DOLAM (*Decoupled OnLine Analytical Mining*), a qual, de forma desacoplada, permite a mineração de regras de associação multidimensional, multinível e de *outliers* em um cubo de dados. A arquitetura DOLAM propõe uma etapa adicional no processo de KDD. O processo tradicional de KDD [8] define as fases de: Seleção, Pré-processamento, Transformação, Mineração de Dados e Interpretação. Este capítulo apresenta a arquitetura DOLAM como uma etapa intermediária entre as fases de Pré-processamento de Dados e Transformação de Dados. A arquitetura DOLAM define e implementa três componentes, a saber: 1) Detector de *Outliers*, 2) Explorador de Subcubos e 3) Expansor de Ancestrais. A partir de uma consulta do usuário, estes componentes são capazes de, respectivamente: 1) identificar ruídos significativos nas células do resultado; 2) explorar, recursivamente, todas as células do resultado, de forma a contemplar todas as possibilidades de combinação multidimensional e multinível e 3) recuperar todos os antecessores (generalizações) das células do resultado. O componente central da arquitetura é o Expansor de Ancestrais - o único de uso obrigatório. As próximas seções apresentam o mapeamento entre os conceitos de bases de dados transacionais e cubos de dados OLAP, bem como os componentes da arquitetura, o funcionamento da arquitetura e a API DOLAM.

#### 3.1. *Mapeamento entre conceitos de bases de dados transacionais e cubos de dados OLAP*

Os algoritmos tradicionais de regras de associação operam em bases de dados transacionais, realizando associações entre itens de transações. Considere o exemplo da compra de um *Laptop Compaq* por um cliente do sexo masculino no ano de 2007. Se esta compra for armazenada em uma base de dados transacional, seria representada como o registro  $R_{11}$  na Tabela 3. Se esta mesma compra for armazenada em um cubo OLAP, seria representada como a célula  $C_{1,1}$  da Tabela 4. Nota-se que, tanto a representação transacional (Tabela 3), quanto a representação OLAP (Tabela 4) armazenam a mesma informação, mas em formatos diferentes.

Registro	Tipo	Marca	Ano	Sexo do Cliente
11	Laptop	Compaq	2007	Masculino
12	Laptop	Compaq	2008	Masculino
13	Laptop	Compaq	2009	Masculino
21	Laptop	Compaq	2007	Feminino
22	Laptop	Compaq	2008	Feminino
23	Laptop	Compaq	2009	Feminino
31	Laptop	Toshiba	2007	Masculino
32	Laptop	Toshiba	2008	Masculino
33	Laptop	Toshiba	2009	Masculino
41	Laptop	Toshiba	2007	Feminino
42	Laptop	Toshiba	2008	Feminino
43	Laptop	Toshiba	2009	Feminino

**Tabela 3 - Representação transacional**

Produto		Cliente	Tempo		
Tipo	Marca		2007	2008	2009
Laptop	Compaq	Masculino	$C_{1,1}$	$C_{1,2}$	$C_{1,3}$
		Feminino	$C_{2,1}$	$C_{2,2}$	$C_{2,3}$
	Toshiba	Masculino	$C_{3,1}$	$C_{3,2}$	$C_{3,3}$
		Feminino	$C_{4,4}$	$C_{4,2}$	$C_{4,3}$

$C_{1,1} = \{\text{Compaq, M, 2007}\}$

$C_{1,2} = \{\text{Compaq, M, 2008}\}$

...

$C_{4,3} = \{\text{Toshiba, F, 2009}\}$

**Tabela 4 - Representação multidimensional**

Como os algoritmos de regras de associação foram propostos para minerar bases de dados transacionais, é necessário criar um mapeamento entre os conceitos de bases de dados transacionais e cubos OLAP (multidimensionais), para que estes algoritmos

sejam capazes de minerar dados a partir de um servidor OLAP. Para isto, este trabalho propõe duas premissas: 1) o conceito de registro de um fato é equivalente ao conceito de célula de dados, uma vez que uma célula de um cubo OLAP reflete a ocorrência de um fato. Isto é, uma célula é o registro de um fato em um cubo de dados. Por exemplo, a célula  $C_{1,1}$  é equivalente ao registro 11; 2) o conceito de dados de um registro é equivalente ao conceito de membros de uma célula, pois um membro de uma célula corresponde a um dado de um registro. Por exemplo, a célula  $C_{1,1}$  tem os membros *Laptop*, *Compaq*, *Masculino* e *2007*, que são equivalentes aos dados, de mesmo nome, do registro 11. Desta forma, em vez de minerar registros de fatos associando seus dados, a mineração OLAM minera células associando seus membros. A Tabela 5 resume estas premissas.

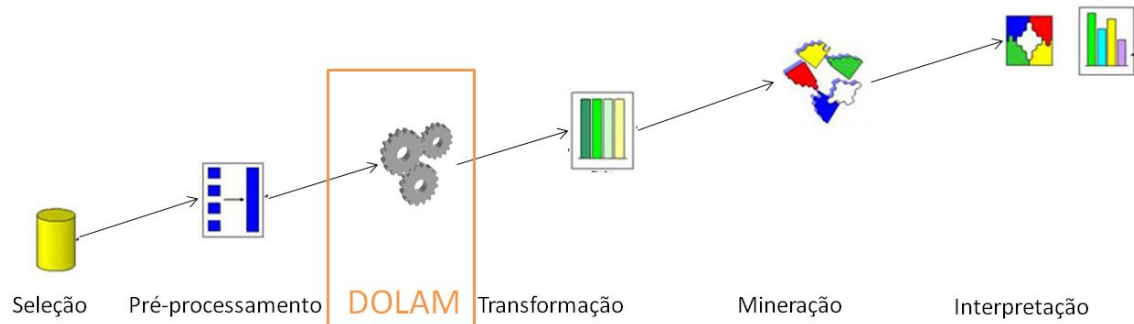
Representação transacional	Representação OLAP
Registro	Célula
Dados de um registro	Membros de uma célula

Tabela 5 – Mapeamento entre conceitos de bases de dados transacionais e cubos de dados OLAP

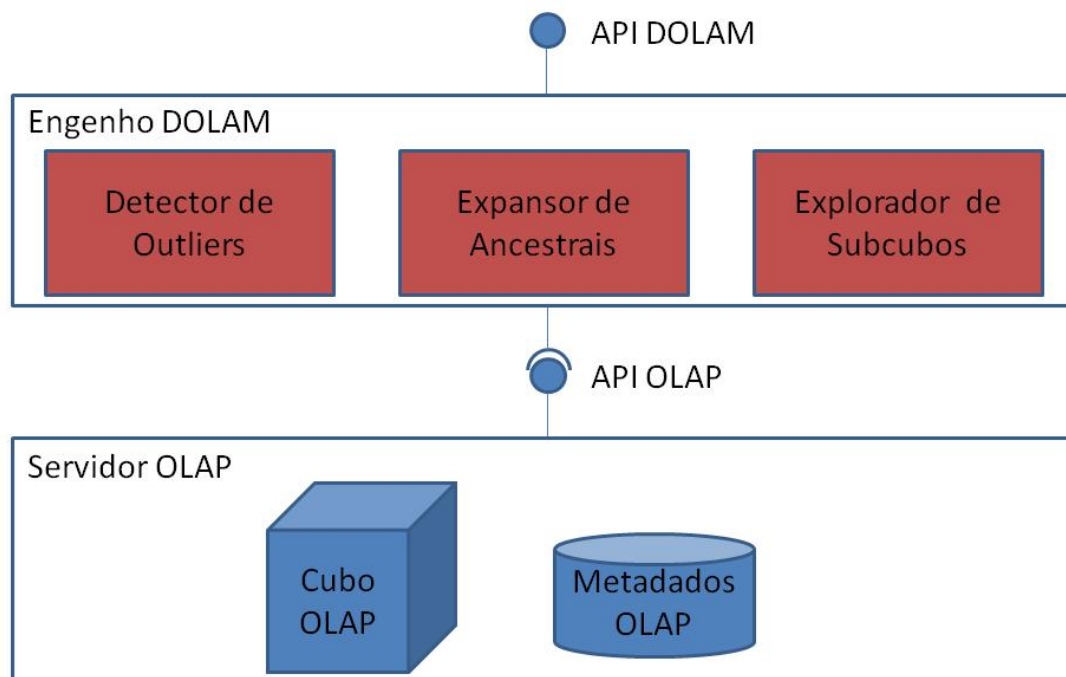
### 3.2. *Arquitetura DOLAM*

A arquitetura DOLAM combina as funcionalidades analíticas providas por um servidor OLAP com o potencial de descoberta de padrões oferecido pelos algoritmos de regras de associação. Desta forma, DOLAM propõe uma etapa intermediária no processo de KDD, adicionando processamento multidimensional e multinível entre as etapas de Pré-processamento de Dados e Transformação de Dados. Assim, após o Pré-Processamento de Dados, os componentes da Arquitetura DOLAM são executados. Em seguida, alguma Transformação pode ser realizada nos dados, por exemplo, conversão de dados para o formato de uma ferramenta de mineração de dados, como o Weka. Em seguida, a etapa de Mineração de Dados é realizada. Assim, DOLAM possibilita que a mineração multidimensional e multinível de cubos OLAP seja realizada sem adaptações nos algoritmos de mineração de dados, tornando possível o reuso das inúmeras soluções de mineração de dados existentes atualmente. Na Figura 11, as etapas em preto

correspondem ao processo tradicional de KDD. A etapa destacada corresponde a DOLAM, proposta neste trabalho. A arquitetura DOLAM é apresentada na Figura 12.



**Figura 11 - Processo de KDD com a etapa DOLAM**



**Figura 12 - Arquitetura DOLAM**

Na Figura 12, tem-se: 1) a camada Servidor OLAP que empacota o Cubo OLAP e seus respectivos metadados; 2) a API OLAP que provê conexão ao servidor OLAP, permite o processamento de consultas e a requisição de metadados; 3) a camada Engenho DOLAM que empacota os Componentes Expansor de Ancestrais, Detector de *Outliers* e Explorador de Subcubos e 4) a API DOLAM que provê o acesso aos componentes do Engenho DOLAM. Ressalta-se que os componentes do Engenho

DOLAM podem ser utilizados separadamente ou de forma integrada, sendo que o Componente Expansor de Ancestrais é o único obrigatório. A seguir, os três componentes da camada Engenho DOLAM e a API DOLAM são detalhados e exemplificados. Para os exemplos utilizou-se o cubo de dados *Sales*, disponível para testes do servidor OLAP Mondrian [27].

### 3.2.1. Componente Expansor de Ancestrais

As células de um cubo de dados são formadas por um conjunto de membros, que se encontram em um determinado nível da hierarquia de sua dimensão. Por exemplo, os membros Janeiro/2009 e Julho/2009 encontram-se no nível Mês da dimensão Tempo de um cubo, conforme ilustra a hierarquia da Figura 13.

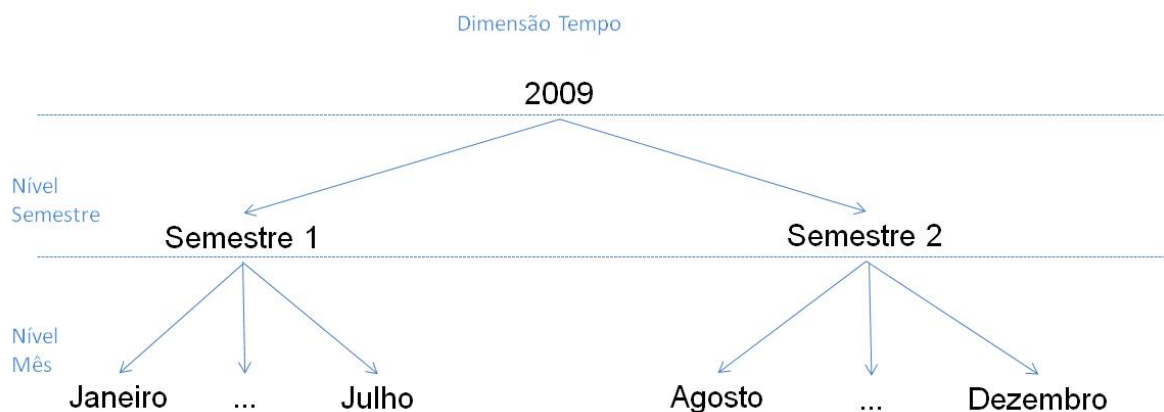


Figura 13 - Hierarquia da dimensão Tempo

A partir da Figura 13, considere um conjunto de células  $C$  a ser minerado, composto pelas células  $C_1$  e  $C_2$ , onde  $C_1$  contém o membro Janeiro e  $C_2$  contém o membro Julho. No processo de KDD convencional, após a etapa de Transformação, estas células seriam mineradas (etapa de Mineração de Dados). Neste caso, somente são associados os membros exatamente iguais que ocorrerem em mais de uma célula. Como Janeiro é diferente de Julho, nenhuma associação entre eles seria detectada. No entanto, é importante perceber que um fato ocorrido no mês de Janeiro ocorreu no ano de 2009, em seu primeiro semestre. Na hierarquia da Figura 13, os membros Semestre 1 e 2009 são ancestrais do membro Janeiro. Da mesma forma, o membro Julho também possui



como ancestrais Semestre 1 e 2009. Ou seja, há uma relação entre os membros Janeiro e Julho, pois ambos possuem Semestre 1 e 2009 como ancestrais. Esta relação é ignorada pela mineração de dados tradicional.

Para criar associações multinível, a arquitetura DOLAM propõe que as células a serem mineradas sejam expandidas pelo Componente Expansor de Ancestrais, de forma que, para cada um de seus membros, todos os seus ancestrais na hierarquia são recuperados e adicionados ao conjunto dos membros da célula original, criando assim uma célula expandida. Desta forma, uma célula expandida consiste em um conjunto de membros de um cubo de dados, contendo os membros da célula original e também todos os ancestrais destes membros. Dado um subcubo inicial definido pelo usuário, denominado subcubo de interesse (SCI), o Algoritmo 3 exibe o pseudo-código do principal método (`expandirAncestrais`) do Componente Expansor de Ancestrais. Note que não existe no cubo de dados uma célula correspondente aos membros de uma célula expandida.

```
1. expandirAncestrais()  
2.   para cada célula de um SCI  
3.     dimensão = SCIMetadados.getDimension(célula)  
4.     para cada dimensão de uma célula do SCI  
5.       membro = SCIMetadados.getMember(dimensão)  
6.       célulaExpandida = expansorAncestrais.addMembro(membro)  
7.       enquanto (nãoÉMembroRaiz(membro))  
8.         membro = datacubeMetadados.getAncestral(membro)  
9.         célulaExpandida = expansorAncestrais.addMember(membro)
```

**Algoritmo 3 - Pseudo-código `expandirAncestrais`**

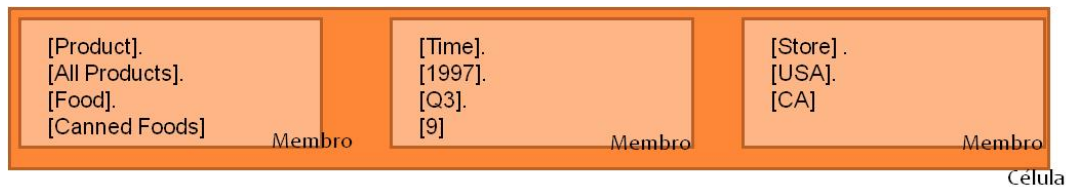
O método `expandirAncestrais` funciona da seguinte forma. Para cada célula do SCI (linha 2), inicialmente suas dimensões são recuperadas (linha 3). Em seguida, para cada uma dessas dimensões (linha 4), é recuperado o membro correspondente no SCI (linha 5). Este membro é adicionado à célula expandida (linha 6) e, enquanto o membro de nível mais alto da dimensão não é alcançado (linha 7), os ancestrais são recuperados sucessivamente (linha 8) e adicionados à célula expandida (linha 9). Assim, para cada

célula original, é criada uma célula expandida, que contem os membros da célula original e todos os ancestrais de cada um destes membros.

A seguir é apresentado um exemplo passo-a-passo do funcionamento do Componente Expansor de Ancestrais. Dada a consulta “quantidade vendida de tipo de produto por estado da loja e trimestre do ano”, a Tabela 6 mostra o resultado desta consulta (ou seja, um SCI). Por sua vez, a Figura 14 exhibe a representação dos membros da célula destacada na Tabela 6 (i.e., *[Product].[All Products].[Food].[Canned Foods]* da dimensão *Product*, *[Time].[1997].[Q3].[9]* da dimensão *Time* e *[Store].[USA].[CA]* da dimensão *Store*).

		Product		
Time	Store	+Dairy	+Canned Foods	+Meat
7	+CA	81	343	36
	+OR	146	550	49
	+WA	149	695	78
8	+CA	113	535	33
	+OR	44	259	45
	+WA	133	780	63
9	+CA	98	435	47
	+OR	82	386	30
	+WA	147	693	75
10	+CA	126	432	40
	+OR	82	277	30
	+WA	161	623	66
11	+CA	151	553	45
	+OR	76	375	60
	+WA	153	984	51
12	+CA	124	570	52
	+OR	104	458	57
	+WA	211	893	61

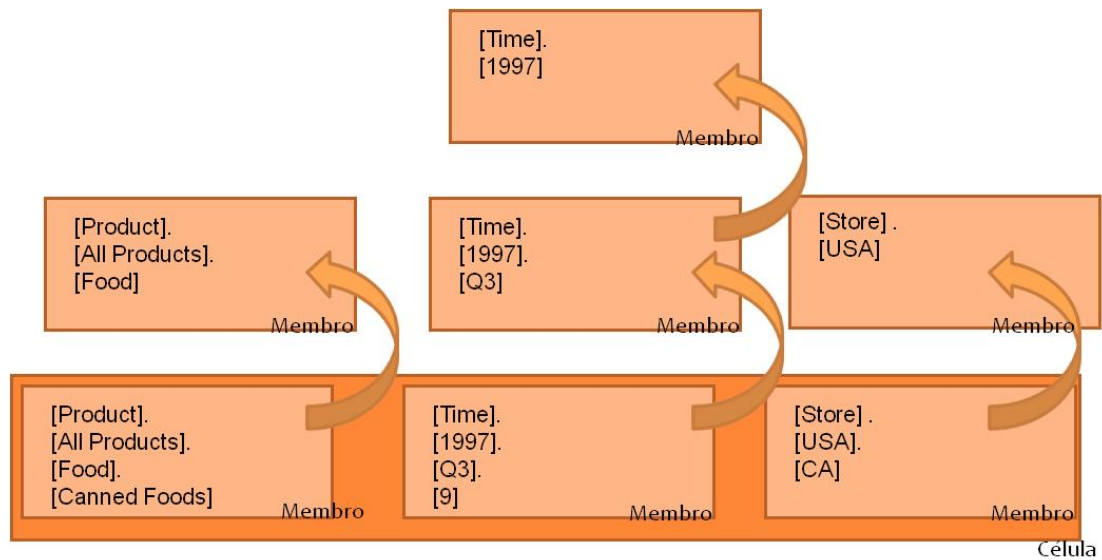
Tabela 6 - Exemplo de SCI



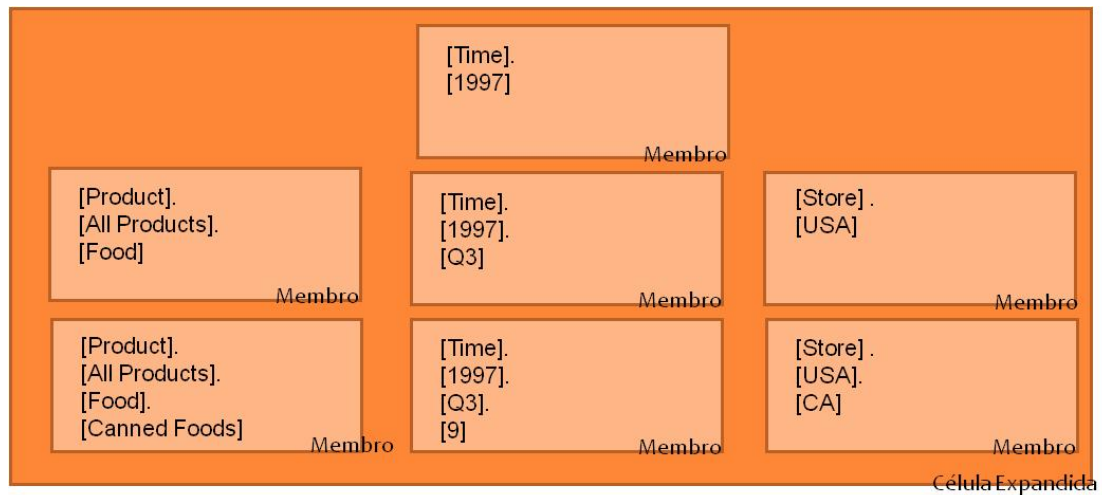
**Figura 14 - Célula original**

Primeiramente, o componente Expansor de Ancestrais realiza uma operação *roll-up* sobre um membro da célula, por exemplo, *[Product].[All Products].[Food].[Canned Foods]*, resultando no membro *[Product].[All Products].[Food]*. O componente Expansor de Ancestrais verifica se é possível realizar uma operação *roll-up* sobre o membro resultante. Ou seja, o componente Expansor de Ancestrais verifica se o membro ainda possui ancestral na hierarquia da dimensão *Product*. Ao verificar que o membro *[Product].[All Products].[Food]* está no nível mais alto da hierarquia, o componente Expansor de Ancestrais segue com o processamento de um outro membro, ainda não expandido, da célula original, por exemplo, o membro *[Time].[1997].[Q3].[9]*. Como este membro possui ancestral, uma operação *roll-up* é executada sobre ele, resultando em *[Time].[1997].[Q3]*.

Da mesma forma, o membro *[Time].[1997].[Q3]* possui ancestral e portanto uma operação *roll-up* é executada sobre este membro, resultando em *[Time].[1997]*. Como este membro não possui ancestral, o componente Expansor de Ancestrais segue para outro membro, ainda não expandido, da célula original, neste caso só restou o membro *[Store].[USA].[CA]*. Uma operação *roll-up* é realizada sobre ele, e como resultado tem-se o membro *[Store].[USA]*. Como este membro não possui ancestrais e não existe mais nenhum membro da célula original para ser expandido, a expansão da célula da Figura 14 é finalizada, resultando em uma célula expandida contendo todos os membros recuperados através de operações *roll-up*, além dos membros da célula original. A Figura 15 exibe uma representação das operações *roll-up* executadas na expansão da célula da Figura 14. A Figura 16 exibe a célula expandida resultante.



**Figura 15 - Processo de expansão da célula da Figura 14**



**Figura 16 - Célula expandida resultante da célula da Figura 14**

Em resumo, a arquitetura DOLAM propõe que as células a serem mineradas sejam previamente processadas pelo componente Expansor de Ancestrais. O resultado obtido consiste em células expandidas, que são utilizadas pelo algoritmo minerador de regras de associação, como ilustra a Figura 17.



**Figura 17 - Utilização do componente Expansor de Ancestrais**

Note que uma célula de cubo OLAP nunca contém dois membros do mesmo nível de uma dimensão, já que, por definição, uma célula OLAP consiste na interseção de um membro de cada dimensão. Uma célula expandida contém os ancestrais de todos os membros de uma célula não expandida. Ou seja, uma célula expandida também nunca pode conter dois membros do mesmo nível de uma dimensão. Como dois membros do mesmo nível não ocorrem juntos em uma célula, seja ela expandida ou não, não podem existir associações entre dois membros do mesmo nível. Assim, dentre os três tipos existentes de regras – inter-dimensional inter-nível, intra-dimensional inter-nível e intra-dimensional intra-nível – é impossível gerar regras do tipo intra-dimensional intra-nível ao se minerar as similaridades entre membros de células de cubos OLAP. Desta forma, no presente contexto, é possível gerar dois tipos de regras de associação: inter-dimensional inter-nível e intra-dimensional inter-nível.

É importante observar que a execução do Componente Expansor de Ancestrais é de complexidade exponencial. Assim, o usuário deve escolher cuidadosamente o nível dos membros do SCI, levando em conta a informação obtida e os custos computacionais envolvidos. No pior caso, onde o SCI possui  $d$  dimensões e os membros de cada dimensão possuem  $n$  ancestrais, a complexidade do Componente Expansor de Ancestrais é  $O(d \cdot n^d)$ .

### 3.2.2. Componente Detector de *Outliers*

O Componente Detector de *Outliers* é responsável por detectar os valores discrepantes em um SCI. Estes valores discrepantes, numericamente distantes do resto dos dados, são denominados *outliers*. Utilizando o Componente Detector de *Outliers*, o usuário é capaz de realizar *outlier mining* (Seção 2.4). Ou seja, obter regras de associação entre os *outliers* presentes em um SCI. Na arquitetura DOLAM, este componente é de uso opcional.

Para detectar outliers em um SCI, a forma tradicional de cálculo de outliers (seção 2.4) foi adaptada ao contexto OLAP. Nos cubos OLAP existe um conjunto de células. Entretanto, não faz sentido comparar células que não possuem relação, como por exemplo, as vendas de um produto A no mês Janeiro com as vendas de um produto B no mês de Fevereiro, pois não há relação direta entre estas células. Como os *outliers* consistem em valores fora do padrão de um conjunto de dados, devem-se comparar dados que tenham alguma relação entre si. Neste sentido, como este trabalho é baseado em servidores OLAP e na linguagem MDX, a qual gera SCI organizados em linhas e colunas, definiram-se os conceitos de *outlier* de linha e *outlier*

de coluna. A Tabela 6 ilustra um SCI definido por uma consulta MDX. Nela, as células da mesma linha correspondem ao mesmo produto (dimensão *Product*), enquanto que as células da mesma coluna correspondem ao mesmo mês (dimensão *Time*) e estado (dimensão *Store*). Ou seja, as células pertencentes à mesma linha ou à mesma coluna são relacionadas. Uma célula é considerada um *outlier* de linha se seu valor for discrepante no contexto de sua linha. Da mesma forma, uma célula é considerada um *outlier* de coluna se seu valor for discrepante no contexto de sua coluna. O Algoritmo 4 exibe o pseudo-código do método para detecção de outliers de linha. Neste algoritmo, para cada linha do SCI (linha 2), o componente Detector de Outliers, obtém as suas células (linha 3), calcula os valores do primeiro e terceiro quartil (linhas 4 e 5). Posteriormente, são calculados o intervalo inter-quartil (IIQ) (linha 6) e os limites inferior e superior dos *outliers* (linhas 7 e 8), segundo a fórmula detalhada na Seção 2.4. Então cada célula da linha é percorrida (linha 9) e caso seu valor não esteja entre os limites inferior e superior calculados para aquela linha (linha 10), a célula é considerada um *outlier* (linha 11). A função de detecção de *outliers* de coluna é semelhante, tendo como entrada o conjunto dos valores de uma coluna.

```

1. detectarOutliersLinha( )
2.   para cada linha do SCI
3.     células = linha.getCélulas()
4.     primeiroQuartil = detectorOutliers.calculaPrimeiroQuartil(células)
5.     terceiroQuartil = detectorOutliers.calculaTerceiroQuartil(células)
6.     IIQ = terceiroQuartil - primeiroQuartil
7.     limiteInferior = primeiroQuartil - 1,5* IIQ
8.     limiteSuperior= terceiroQuartil + 1,5 * IIQ
9.     para cada célula em células
10.       se célula<= limiteInferior OU célula >= limiteSuperior
11.         detectorOutlier.adicionaAListaOutliersDeLinha(célula)

```

**Algoritmo 4 - Pseudo-código detectarOutliersLinha**

		Product		
Time	Store	+Dairy	+Canned Foods	+Meat
7	+CA	81	343	36
	+OR	146	550	49
	+WA	149	695	78
8	+CA	113	535	33
	+OR	44	259	45
	+WA	133	780	63
9	+CA	98	435	47
	+OR	82	386	30
	+WA	147	693	75
10	+CA	126	432	40
	+OR	82	277	30
	+WA	161	623	66
11	+CA	151	553	45
	+OR	76	375	60
	+WA	153	984	51
12	+CA	124	570	52
	+OR	104	458	57
	+WA	211	893	61

**Tabela 7 - Exemplo de Subcubo**

Para minerar *outliers* de linha, a função de detecção de *outliers* de linha é executada  $n$  vezes, onde  $n$  é o número de linhas do subcubo. Em cada execução, os parâmetros de entrada são os valores das células de uma linha. Por exemplo, considere o SCI ilustrado na Tabela 7. Para calcular os *outliers* de linha, a função de cálculo de *outliers* será executada 18 vezes, pois há 18 linhas. Em cada execução, os parâmetros de entrada serão as células contidas em um retângulo alaranjado da Tabela 8.

		Product		
Time	Store	+Dairy	+Canned Foods	+Meat
7	+CA	81	343	36
	+OR	146	550	49
	+WA	149	695	78
8	+CA	113	535	33
	+OR	44	259	45
	+WA	133	780	63
9	+CA	98	435	47
	+OR	82	386	30
	+WA	147	693	75
10	+CA	126	432	40
	+OR	82	277	30
	+WA	161	623	66
11	+CA	151	553	45
	+OR	76	375	60
	+WA	153	984	51
12	+CA	124	570	52
	+OR	104	458	57
	+WA	211	893	61

Tabela 8 - Células selecionadas por execução do cálculo de *outliers* de linha

De forma semelhante, para o cálculo de *outliers* de coluna, o algoritmo será executado 3 vezes, pois há 3 colunas. Em cada execução os parâmetros de entrada são as células contidas em um dos retângulos destacados na Tabela 9Tabela 8. Na arquitetura DOLAM, os usuários devem escolher se desejam detectar *outliers* de linha, de coluna ou ambos os tipos.



Time	Store	Product		
		+Dairy	+Canned Foods	+Meat
7	+CA	81	343	36
	+OR	146	550	49
	+WA	149	695	78
8	+CA	113	535	33
	+OR	44	259	45
	+WA	133	780	63
9	+CA	98	435	47
	+OR	82	386	30
	+WA	147	693	75
10	+CA	126	432	40
	+OR	82	277	30
	+WA	161	623	66
11	+CA	151	553	45
	+OR	76	375	60
	+WA	153	984	51
12	+CA	124	570	52
	+OR	104	458	57
	+WA	211	893	61

Tabela 9 - Células selecionadas por execução do cálculo de outliers de coluna

Como qualquer célula do SCI pode conter um *outlier*, as regras de associação geradas podem conter membros em diferentes níveis, sendo assim regras de associação multinível.

### 3.2.3. Componente Explorador de Subcubos

O Componente Explorador de Subcubos tem como objetivo processar todas as combinações entre os membros do SCI e seus ancestrais na hierarquia. Como resultado, diversos subcubos são gerados, todos eles excedem os limites do SCI definido pelo usuário, mas mantêm uma forte conexão com o SCI, pois os membros destes subcubos possuem relação de ancestralidade com os membros do SCI. Na arquitetura, este componente também é de uso opcional.

Pelo fato do Componente Explorador de Subcubos gerar subcubos que são generalizações do SCI, este é especialmente proveitoso para a descoberta de padrões ocultos e úteis (um dos objetivos da mineração de dados), pois este componente expande as fronteiras do SCI (i.e., descobre subcubos ocultos) e seus subcubos descobertos possuem um forte relacionamento de ancestralidade com o SCI (i.e., os subcubos descobertos são potencialmente úteis). Note que o componente de Exploração de Subcubos expande as fronteiras do SCI, mas não inclui membros não relacionados com os escolhidos pelo usuário, por exemplo, membros de uma dimensão não contemplada no SCI. Desta forma, o usuário tem controle sobre o que deve ser explorado. Note também que a geração dos subcubos a partir da generalização dos níveis dos membros do SCI permite a obtenção de padrões mais fortes, pois quando se analisam as células em níveis mais altos existem mais similaridades entre elas do que numa análise em níveis mais baixos. Assim, são detectadas similaridades entre mais células e conseqüentemente os padrões gerados envolvem um número maior de células, sendo, portanto, mais fortes.

Para exemplificar o funcionamento do Componente Explorador de Subcubos, considere a consulta MDX apresentada na Figura 18. A consulta seleciona as informações sobre vendas (*Sales*) de três produtos (*[Product].[AllProducts].[Drink].[Dairy]*, *[Product].[AllProducts].[Food].[CannedFoods]*, *[Product].[AllProducts].[Food].[Meat]*), realizadas nos meses compreendidos no 3º e 4º trimestres (*Quarter*) nos estados do país *USA*. A consulta da Figura 18 define o SCI, que é exibido na Tabela 10.

```
SELECT
    {[Product].[All Products].[Drink].[Dairy],
     [Product].[All Products].[Food].[Canned Foods],
     [Product].[All Products].[Food].[Meat]}
ON COLUMNS,
    Crossjoin({[Time].[1997].[Q3].Children,
               [Time].[1997].[Q4].Children},
              {[Store].[All Stores].[USA].Children})
ON ROWS
FROM [Sales]
```

**Figura 18 – Consulta inicial**

		Product		
Time	Store	+Dairy	+Canned Foods	+Meat
7	+CA	81	343	36
	+OR	146	550	49
	+WA	149	695	78
8	+CA	113	535	33
	+OR	44	259	45
	+WA	133	780	63
9	+CA	98	435	47
	+OR	82	386	30
	+WA	147	693	75
10	+CA	126	432	40
	+OR	82	277	30
	+WA	161	623	66
11	+CA	151	553	45
	+OR	76	375	60
	+WA	153	984	51
12	+CA	124	570	52
	+OR	104	458	57
	+WA	211	893	61

**Tabela 10 - Resultado da consulta inicial que define o SCI definido na Figura 18**

O Algoritmo 5 exibe o pseudo-código do principal método (explorarSubcubos) do Componente Explorador de Subcubos. Neste pseudo-código, se o SCI tiver apenas 1 dimensão (linha 2), cria-se uma lista de níveis a partir de sucessivos *roll-ups* na única dimensão do SCI (linha 3) e faz-se a lista de combinações igual a lista de níveis da única dimensão do SCI (linha 4). Caso contrário (linha 5), isto é, o SCI tem 2 ou mais dimensões, criam-se as listas de níveis da primeira dimensão (linha 6) e da segunda dimensão (linha 7) a partir de sucessivos *roll-ups* nestas dimensões. Em seguida chama-se o método combinar2a2, o qual irá, de forma recursiva, fazer o produto cartesiano entre os níveis de dimensões diferentes (não há combinação entre níveis de uma mesma dimensão). Por fim, são escritas e processadas as consultas MDX que geram os subcubos definidos a partir das combinações realizadas.

```

1. explorarSubcubos()
2.   se (exploradorSubcubos.getQtdDimensõesSCI() = 1)
3.       listaNíveisÚnicaDim.addRollups(exploradorSubcubos.getDimensão(0))
4.       listaNíveisCombinação = listaNíveisÚnicaDim
5.   senão
6.       listaNíveisPriDim.addRollups(exploradorSubcubos.getDimensão(0))
7.       listaNíveisSegDim.addRollups(exploradorSubcubos.getDimensão(1))
8.       exploradorSubcubos.combinar2a2(listaNíveisPriDim, listaNíveisSegDim)
9.   exploradorSubcubos.processarMDXSubcubos()

```

**Algoritmo 5 - Pseudo-código do método explorarSubcubos**

Dada a sua importância, a seguir é apresentado o Algoritmo 6 que mostra o pseudo-código do método combinar2a2.

```

1. combinar2a2 (listaNíveisA, listaNíveisB)
2.   para cada A da listaNíveisA
3.       para cada B da listaNíveisB
4.           listaNíveisCombinação.addCombinação(A, B)
5.   proxDim = exploradorSubcubos.getProxDimensão()
6.   se proxDim != -1
7.       listaNíveisProxDim.addRollups(exploradorSubcubos.getDimensão(proxDim))
8.       combinar2a2(listaNíveisCombinação, listaNíveisProxDim)

```

**Algoritmo 6 - Pseudo-código do método combinar2a2**

O método combinar 2a2 recebe duas listas de níveis de dimensões do SCI (linha 1) de forma a fazer todas as combinações (produto cartesiano) entre os níveis de dimensões diferentes e gerar uma lista com estas combinações (linha 4). Se existir dimensão faltando ser combinada (linha 6), cria-se uma lista de níveis a partir de sucessivos *roll-ups* nos níveis da dimensão a ser combinada (linha 7) e, de forma recursiva (até fazer todas as combinações entre os níveis de dimensões diferentes), faz-se uma nova combinação entre a lista de combinações anterior e a lista de níveis da dimensão que falta ser combinada (linha 8).

A Figura 19 exibe os membros dos subcubos gerados. Cada linha corresponde a um cubo gerado, cada coluna corresponde aos membros de uma dimensão. As setas laranja indicam a dimensão sobre a qual a operação de *roll-up* foi realizada, ligando o membro antigo ao novo. A linha 1 da Figura 19 corresponde ao SCI.

12	[Product].[All Products].[Drink] [Product].[All Products].[Food]	[Time].[1997]	[Store].[All Stores].[USA]
11	[Product].[All Products].[Drink] [Product].[All Products].[Food]	[Time].[1997]	[Store].[All Stores].[USA].Children
10	[Product].[All Products].[Drink] [Product].[All Products].[Food]	[Time].[1997].[Q3] [Time].[1997].[Q4]	[Store].[All Stores].[USA]
9	[Product].[All Products].[Drink] [Product].[All Products].[Food]	[Time].[1997].[Q3] [Time].[1997].[Q4]	[Store].[All Stores].[USA].Children
8	[Product].[All Products].[Drink] [Product].[All Products].[Food]	[Time].[1997].[Q3].Children [Time].[1997].[Q4].Children	[Store].[All Stores].[USA]
7	[Product].[All Products].[Drink] [Product].[All Products].[Food]	[Time].[1997].[Q3].Children [Time].[1997].[Q4].Children	[Store].[All Stores].[USA].Children
6	[Product].[All Products].[Drink].[Dairy] Product].[All Products].[Food].[Can Food] [Product].[All Products].[Food].[Meat]	[Time].[1997] [Time].[1997]	[Store].[All Stores].[USA]
5	[Product].[All Products].[Drink].[Dairy] Product].[All Products].[Food].[Can Food] [Product].[All Products].[Food].[Meat]	[Time].[1997] [Time].[1997]	[Store].[All Stores].[USA].Children
4	[Product].[All Products].[Drink].[Dairy] Product].[All Products].[Food].[Can Food] [Product].[All Products].[Food].[Meat]	[Time].[1997].[Q3] [Time].[1997].[Q4]	[Store].[All Stores].[USA]
3	[Product].[All Products].[Drink].[Dairy] Product].[All Products].[Food].[Can Food] [Product].[All Products].[Food].[Meat]	[Time].[1997].[Q3] [Time].[1997].[Q4]	[Store].[All Stores].[USA].Children
2	[Product].[All Products].[Drink].[Dairy] Product].[All Products].[Food].[Can Food] [Product].[All Products].[Food].[Meat]	[Time].[1997].[Q3].Children [Time].[1997].[Q4].Children	[Store].[All Stores].[USA]
1	[Product].[All Products].[Drink].[Dairy] Product].[All Products].[Food].[Can Food] [Product].[All Products].[Food].[Meat]	[Time].[1997].[Q3].Children [Time].[1997].[Q4].Children	[Store].[All Stores].[USA].Children

Figura 19 - Membros dos subcubos gerados pelo Componente Explorador de Subcubos

Como componente Explorador de Subcubos funciona de forma recursiva, o primeiro *roll-up* é realizado sobre a última dimensão definida na consulta do SCI, que, no exemplo, é a dimensão *Store*. O primeiro subcubo gerado (linha 2 da Figura 19) conterá, para as dimensões *Product* e *Time*, os mesmos membros do SCI. Para a dimensão *Store*, o membro será o ancestral do membro do SCI (ou seja, *[Store].[All Stores].[USA]*). O subcubo definido pela consulta da linha 2 da Figura 19 é exibido na Tabela 11.

		Product		
Time	Store	+Dairy	+Canned Foods	+Meat
7	+USA	376	1,588	163
8	+USA	290	1,574	141
9	+USA	327	1,514	152
10	+USA	369	1,332	136
11	+USA	380	1,912	156
12	+USA	439	1,921	170

Tabela 11 - Primeiro subcubo gerado

O Componente seguiria realizando operações *roll-up* sobre a dimensão *Store*, mas o membro *[Store].[All Stores].[USA]* se encontra no topo da hierarquia de sua dimensão. Como o componente é recursivo, a execução segue para a próxima dimensão: *Time*. Os demais membros permanecem como na consulta original, pois este era o contexto no momento da chamada recursiva. A linha 3 da Figura 19 exibe os membros resultantes após a operação *roll-up* realizada sobre a dimensão *Time*. A Tabela 12 exibe o subcubo definido pela consulta da linha 3 da Figura 19.

		Product		
Time	Store	+Dairy	+Canned Foods	+Meat
+Q3	+CA	292	1,313	116
	+OR	272	1,195	124
	+WA	429	2,168	216
+Q4	+CA	401	1,555	137
	+OR	262	1,110	147
	+WA	525	2,500	178

Tabela 12 - Segundo subcubo gerado

Neste ponto, o contexto consiste nos membros da linha 3. Em seguida, uma nova chamada recursiva é feita com a dimensão *Store*. Partindo do contexto atual, os membros da dimensão *Product* e *Time* permanecem os mesmos da linha 3 da Figura 19, e uma nova operação de *roll-up* é executada sobre a dimensão *Store*, resultando nos membros exibidos na linha 4 da Figura 19. O subcubo definido pela consulta da linha 4 da Figura 19 é exibido na Tabela 13.

		Product		
Time	Store	+Dairy	+Canned Foods	+Meat
+Q3	+USA	993	4,676	456
+Q4	+USA	1,188	5,165	462

**Tabela 13 - Terceiro subcubo gerado**

O componente segue assim sucessivamente, até que o último subcubo gerado contenha apenas membros no nível mais alto das hierarquias de suas respectivas dimensões. No exemplo dado, são gerados 11 subcubos, que somados ao SCI resultam em 12 subcubos para exploração pelo minerador (Figura 19).

Note que o Componente Explorador de Subcubos realiza combinações através de busca cega na hierarquia das dimensões, começando a partir do nível definido no SCI. Este componente poderia ser implementado de forma menos custosa computacionalmente, utilizando heurísticas, por exemplo. Contudo, devido à limitação de tempo, a atual implementação deste componente demonstra adequadamente a sua funcionalidade.

É importante perceber que o Componente Explorador de Subcubos possui custo exponencial de acordo com o número de dimensões e o nível dos membros definidos no SCI. O usuário deve estar atento a este custo, tomando decisões de compromisso entre a quantidade e a abrangência dos subcubos gerados e o custo computacional associado. Ou seja, o usuários devem ter consciência das implicações de selecionar um SCI com muitas dimensões ou com níveis muito baixos na hierarquia.

### 3.3. Funcionamento da Arquitetura

Esta seção apresenta as formas como os três componentes propostos podem ser combinados para prover as diferentes funcionalidades disponíveis através da API DOLAM. Dado que os componentes Detector de *Outliers* e Explorador de Subcubos são de uso opcional, os cenários de utilização dos componentes são listados abaixo e detalhados a seguir.

1. SCI com o Componente Expansor de Ancestrais;
2. SCI com os Componentes Expansor de Ancestrais e Detector de *Outliers*;
3. SCI com os Componentes Expansor de Ancestrais e Explorador de Subcubos;
4. SCI com os Componentes Expansor de Ancestrais, Detector de *Outliers* e Explorador de Subcubos.

#### Cenário 1: SCI com o Componente Expansor de Ancestrais

Neste cenário, evoca-se o método *executaConsulta* da classe Expansor de Ancestrais da API DOLAM com a consulta de definição do SCI como parâmetro de entrada. A saída deste método é então passada como entrada para o método *recuperaAncestrais* para requisitar que todas as células do SCI sejam expandidas pelo componente Expansor de Ancestrais. A Figura 20 exibe o diagrama de atividades do Cenário 1. Posteriormente, quando as células expandidas são enviadas para o algoritmo minerador, regras de associação multinível são geradas entre os membros das células do SCI.

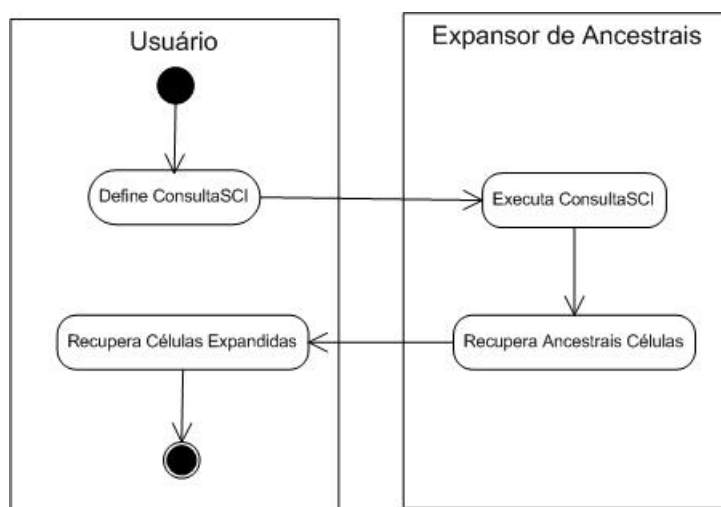


Figura 20 - Diagrama de Atividades do Cenário 1



## Cenário 2: SCI com os Componentes Expansor de Ancestrais e Detector de Outliers

Neste cenário, o usuário escolhe se deseja minerar *outliers* de linha ou de coluna, e então, de acordo com a escolha do usuário, utilizando a consulta de definição do SCI como entrada, são executados os métodos *detectaOutliersLinha* ou *detectaOutliersColuna* da classe Detector de *Outliers* da API DOLAM, que detectam *outliers* presentes no SCI. Em seguida, executa-se método *recuperaAncestrais* do Componente Expansor de Ancestrais apenas sobre as células detectadas como *outliers*. A Figura 21 exibe o diagrama de atividades do Cenário 2. Assim, a partir deste cenário (ou seja, da evocação dos métodos *detectaOutliersLinha* e *detectaOutliersColuna* da classe Detector de *Outliers* da API DOLAM), é possível realizar *outlier mining* multinível.

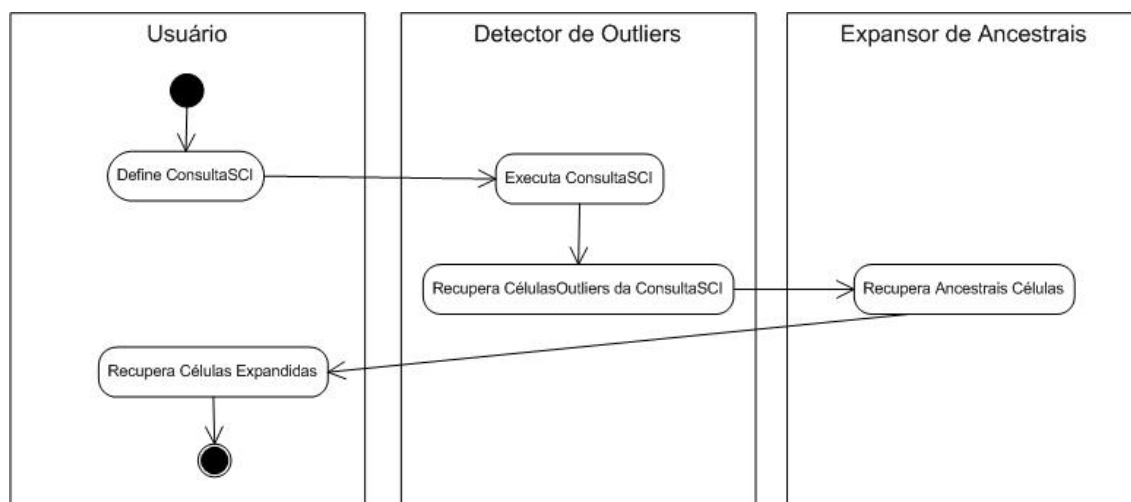


Figura 21 - Diagrama de Atividades do Cenário 2

## Cenário 3: SCI com os Componentes Expansor de Ancestrais e Explorador de Subcubos

No cenário 3, chama-se o método *gerarSubCubos* da classe Explorador de Subcubos da API DOLAM, passando como entrada a consulta de definição do SCI, para gerar os subcubos (consultas) relacionados ao SCI em níveis mais altos. Então, após executar estas consultas, o método *recuperaAncestrais* do componente Expansor de Ancestrais é executado sobre todas as células de cada subcubo gerado pelo Componente Explorador de Subcubos. A Figura 22 ilustra o diagrama de atividades do Cenário 3.

Ressalta-se que este é o cenário de maior custo computacional, pois todas as células de todos os cubos gerados são expandidas. Por este motivo, neste cenário há mais chances de serem encontrados padrões fortes.

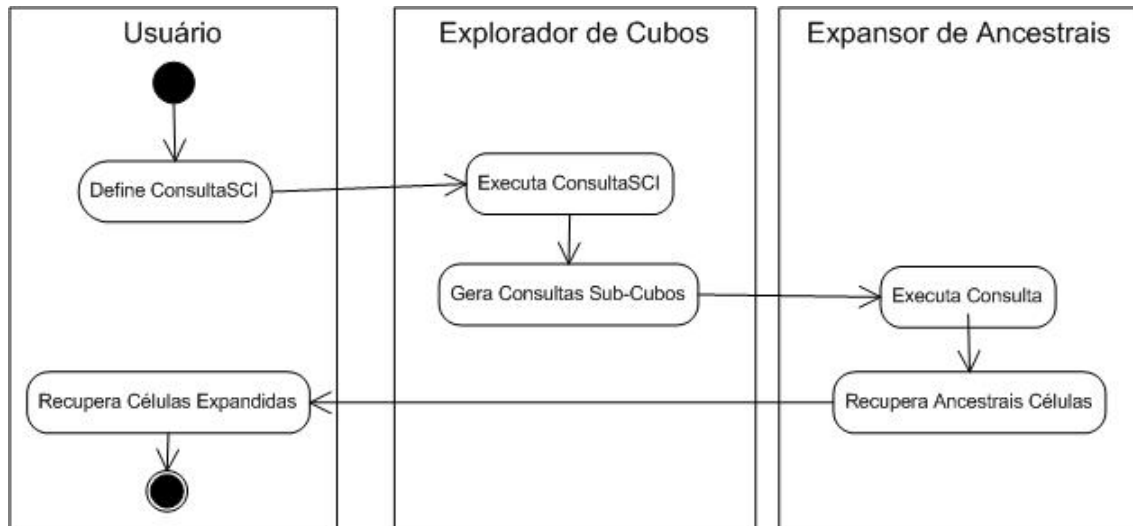
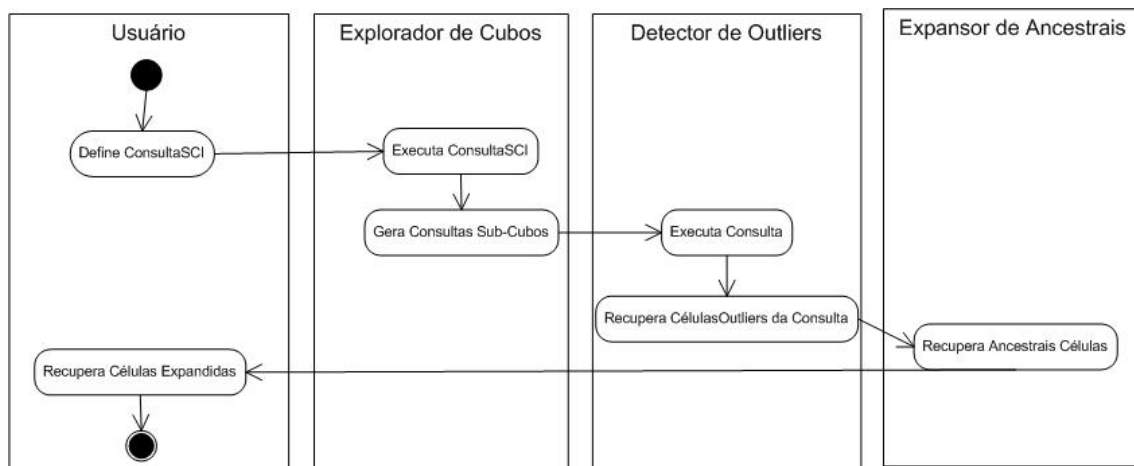


Figura 22 - Diagrama de Atividades do Cenário 3

#### Cenário 4: SCI com os Componentes Expansor de Ancestrais, Detector de *Outliers* e Explorador de Subcubos

Neste cenário, primeiramente, a classe Explorador de Subcubos gera diversos subcubos relacionados ao SCI (método *gerarSubCubos*). Em seguida, os métodos *detectaOutliersLinha* e *detectaOutliersColuna* da classe Detector de *Outliers* são executados sobre cada subcubo gerado pelo Componente Explorador de Subcubos. Após esta execução, são obtidos todos os *outliers* presentes em cada subcubo gerado. Estes *outliers* são expandidos pelo método *recuperaAncestrais* da classe Expansor de Ancestrais. O Diagrama de Atividades do Cenário 4 é exibido na Figura 23. As regras de associação geradas relacionarão os *outliers* dos subcubos gerados.



**Figura 23 - Diagrama de Atividades do Cenário 4**

### **3.4. API DOLAM**

Esta seção apresenta a API DOLAM, a qual deve ser utilizada para acessar os componentes apresentados na arquitetura DOLAM. A Figura 24 exibe o diagrama de classes da API DOLAM. A classe *Membro* contém informações sobre os membros de uma célula OLAP. Uma célula expandida é formada por um conjunto de membros. A classe *Expansor de Ancestrais* possui o método *recuperaAncestrais*, que recebe como entrada um conjunto de células (ou seja, as células de um SCI) e gera como saída um conjunto de células expandidas. A classe *Expansor de Ancestrais* possui também o método *executaConsulta*, que recebe como entrada uma consulta escrita em MDX e a executa, gerando como saída um conjunto de células do resultado da consulta. Esta saída pode ser usada como entrada do método *recuperaAncestrais*, por exemplo.

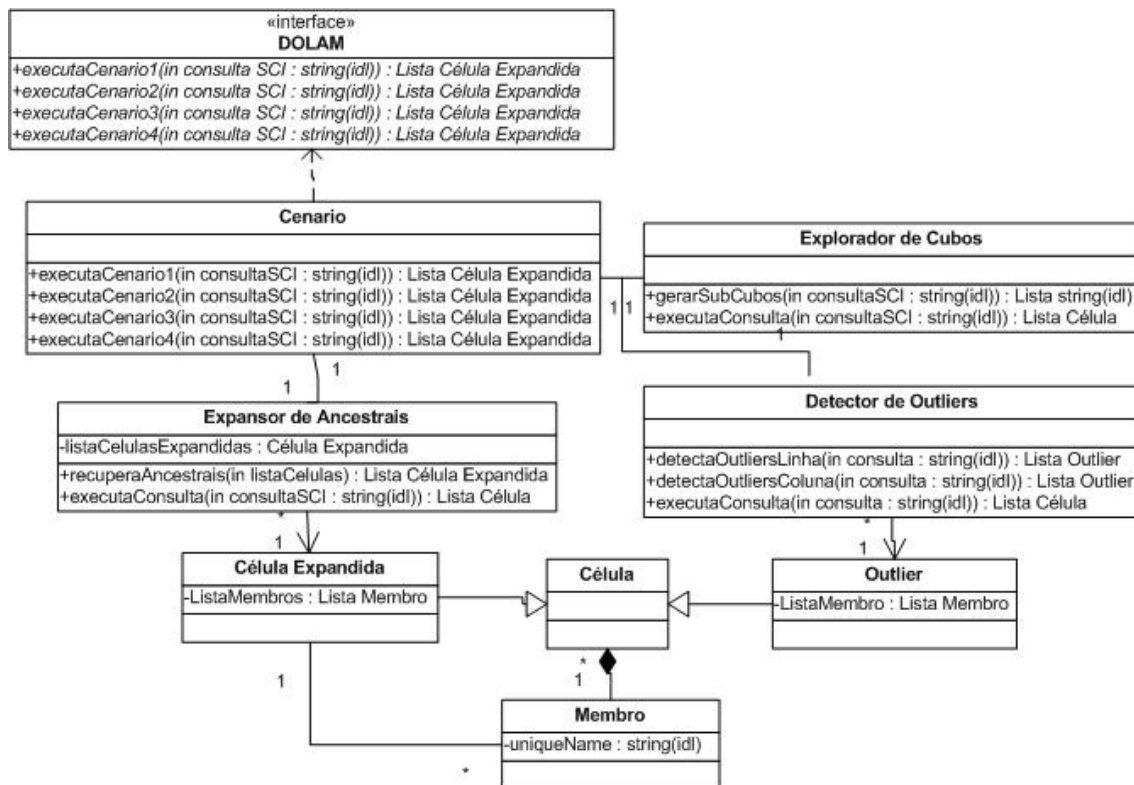


Figura 24 - Diagrama de Classes da API DOLAM

Um *outlier* corresponde a uma célula do cubo de dados OLAP e é formado por um conjunto de membros. A classe *Detector de Outliers* possui dois métodos: *detectaOutliersLinha* e *detectaOutliersColuna*. Ambos recebem como entrada uma consulta de definição do SCI e geram como saída o conjunto de células cujos valores são *outliers* de linha e de coluna, respectivamente.

A classe *Explorador de Subcubos* possui o método *gerarSubCubos* que recebe como entrada uma consulta de definição do SCI e gera como saída um conjunto de consultas que definem os subcubos gerados.

### 3.5. Considerações Finais

Neste capítulo foi apresentada a arquitetura DOLAM. Inicialmente, foi fornecido um mapeamento entre os conceitos de bases transacionais e cubos de dados OLAP. Em seguida, apresentam-se e discutem-se os três componentes que formam a arquitetura DOLAM. Em seguida, o funcionamento da arquitetura é explicado na forma de cenários de uso dos componentes propostos pela arquitetura DOLAM. Por fim, a API DOLAM é descrita.

## 4. Estudo de caso

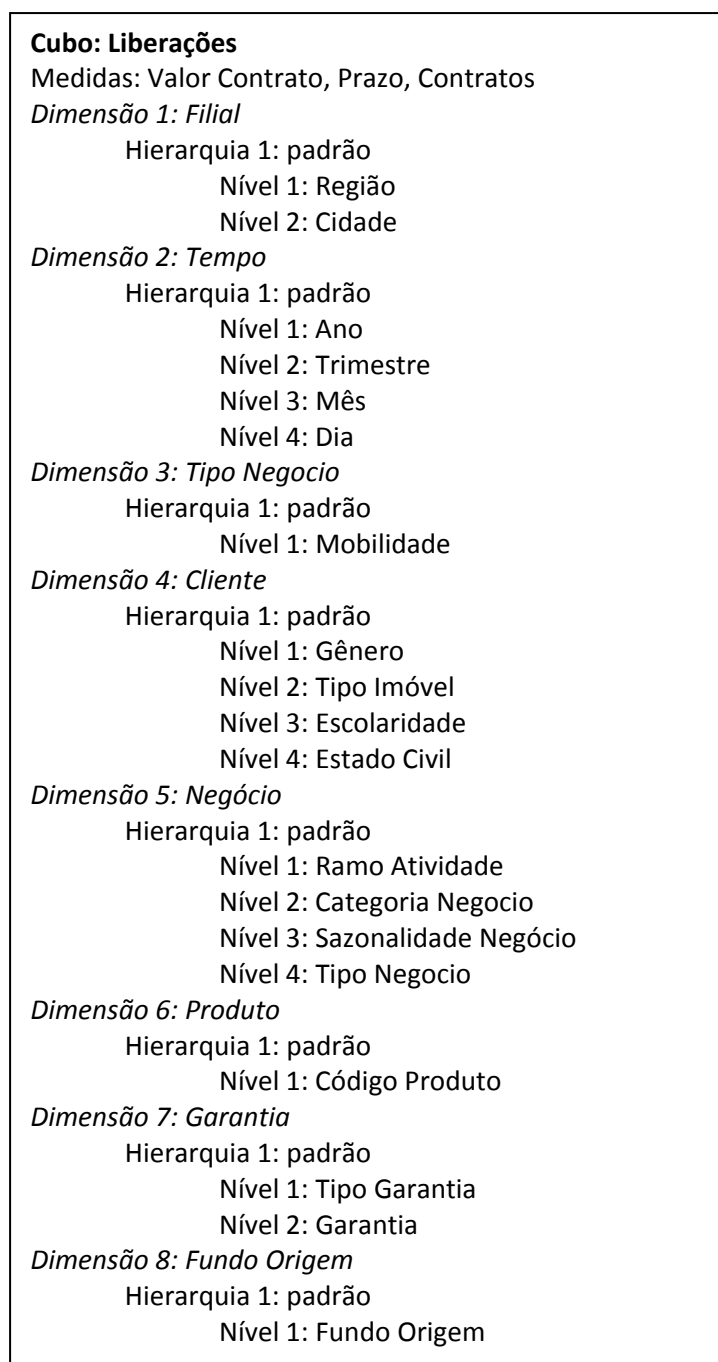
Foi elaborado um estudo de caso que apresenta indícios de que a arquitetura DOLAM cumpre os objetivos propostos. A arquitetura DOLAM se propõe a gerar células expandidas, a partir das quais podem ser geradas regras de associação multidimensional e multinível. Os diferentes cenários de utilização propostos pela arquitetura DOLAM diferem na porção das células do cubo de dados OLAP que serão mineradas. Todos os cenários de uso da arquitetura DOLAM geram como saída um conjunto de células expandidas. Estas células expandidas, ao serem mineradas por um algoritmo de regras de associação, possibilitam que as regras de associação geradas sejam da forma multidimensional e multinível.

Neste estudo de caso, as células expandidas geradas pela arquitetura DOLAM foram utilizadas para criar arquivos que são dados como entrada para aos algoritmos *APriori* (por ser clássico) e *FP-Growth* (por ter bom desempenho) do pacote de software *Weka*. A escolha do *Weka* foi motivada pelo fato deste pacote de *software* ser de código aberto, bem aceito pela comunidade acadêmica e não realizar nenhum processamento multidimensional ou multinível.

A máquina utilizada para o estudo de caso é um *laptop* Dell XPS com processador Intel Core 2 Duo 2.0 Ghz, 3Gb de memória RAM, executando o sistema operacional Windows Vista. A arquitetura DOLAM foi implementada em Java, utilizando a API OLAP4J para acessar o servidor OLAP *Mondrian*. A cada experimento, para que o estudo de caso não sofra influência de alguma *cache*, o computador foi reiniciado, tanto na execução da arquitetura DOLAM quanto na execução dos algoritmos de regras de associação no *Weka*.

### 4.1. *Cubo utilizado*

Utilizou-se um cubo de dados OLAP com informações reais sobre micro-crédito de empréstimos. O cubo de dados OLAP utilizado possui 14,5 Mb e sua tabela de fatos contém 39877 registros. O esquema do cubo é ilustrado na Figura 25. O Apêndice I contém esquema XML deste cubo.



**Figura 25 – Esquema do cubo de dados OLAP Liberacoes**

O SCI considerado neste estudo de caso é apresentado na Tabela 14. O SCI reflete as liberações de crédito nas cidades de Pacajus, Caucaia, Maracanaú e Fortaleza (localizadas na região metropolitana de Fortaleza), por mobilidade do tipo de negócio (se fixo ou ambulante) por mês do ano de 2009. O resultado da consulta é exibido na Tabela 14.

```

select {
    [Filial].[All Filials].[FORTALEZA          ].[PACAJUS          ],
    [Filial].[All Filials].[FORTALEZA          ].[CAUCAIA          ],
    [Filial].[All Filials].[FORTALEZA          ].[MARACANAU        ],
    [Filial].[All Filials].[FORTALEZA          ].[FORTALEZA        ]
} ON COLUMNS,
Crossjoin(
{
    [Tempo].[All Tempos].[2009].[1].[Janeiro],
    [Tempo].[All Tempos].[2009].[1].[Fevereiro],
    [Tempo].[All Tempos].[2009].[1].[Março],
    [Tempo].[All Tempos].[2009].[2].[Abril],
    [Tempo].[All Tempos].[2009].[2].[Maio],
    [Tempo].[All Tempos].[2009].[2].[Junho],
    [Tempo].[All Tempos].[2009].[3].[Julho],
    [Tempo].[All Tempos].[2009].[3].[Agosto],
    [Tempo].[All Tempos].[2009].[3].[Setembro],
    [Tempo].[All Tempos].[2009].[4].[Outubro],
    [Tempo].[All Tempos].[2009].[4].[Novembro],
    [Tempo].[All Tempos].[2009].[4].[Dezembro]
},
{
    [Tipo negocio].[All Tipo negocios].[Ambulante],
    [Tipo negocio].[All Tipo negocios].[Fixo]
}
) ON ROWS
from [Liberacoes]

```

Figura 26 – Consulta Inicial da definição do SCI

Tempo			Tipo negocio	Filial			
Ano	Trimestre	Mes	Tipo de negocio	FORTALEZA			
2009	1	+Janeiro	Ambulante	8.754,00	9.436,00	24.471,00	74.266,00
			Fixo	50.223,00	25.526,00	28.397,00	155.644,00
		+Fevereiro	Ambulante	4.898,00	7.509,00	9.214,00	36.391,00
			Fixo	21.147,00	31.902,00	23.009,00	134.150,00
		+Março	Ambulante	19.000,00	11.948,00	38.120,00	78.166,00
			Fixo	17.298,00	14.250,00	70.793,00	240.219,00
	2	+Abril	Ambulante	3.971,00	6.047,00	9.345,00	83.188,00
			Fixo	13.817,00	37.053,00	36.898,00	153.126,00
		+Maio	Ambulante	17.001,00	16.203,00	18.400,00	105.544,00
			Fixo	15.979,00	12.423,28	37.462,00	155.501,00
		+Junho	Ambulante	13.486,00	8.165,00	28.730,00	32.432,00
			Fixo	56.022,00	43.267,00	43.649,00	193.766,00
	3	+Julho	Ambulante	12.222,00	5.284,00	19.410,00	64.031,00
			Fixo	42.310,00	27.258,00	50.030,00	198.879,00
		+Agosto	Ambulante	5.491,00	1.362,00	24.545,00	85.907,00
			Fixo	24.775,00	5.518,00	60.720,50	149.376,00
		+Setembro	Ambulante	15.386,00	8.904,00	20.144,00	100.438,00
			Fixo	10.347,00	17.439,00	56.486,00	112.255,00
	4	+Outubro	Ambulante	21.052,00	13.918,00	11.665,00	58.451,00
			Fixo	32.950,00	53.367,40	47.473,00	161.620,00
		+Novembro	Ambulante	18.590,00	23.060,00	12.772,00	121.654,00
			Fixo	13.696,00	26.422,00	34.258,00	243.224,00
		+Dezembro	Ambulante	16.015,00	8.852,00	13.636,00	12.706,00
			Fixo	3.343,00	16.086,00	20.603,00	111.352,00

Tabela 14 - Resultado da consulta da Figura 26

Analisando-se o SCI e a hierarquia do cubo de dados OLAP, verifica-se que o cubo é composto por três dimensões (Filial, TipoNegocio e Tempo). Na dimensão Filial, existe 1 nível (Região) acima do nível definido no SCI. Por isto, o Componente Explorador de Subcubos é capaz de explorar 2 níveis da dimensão Filial: o nível Cidade (definido no SCI) e o nível Região (nível acima de Cidade). Na dimensão Tempo, os níveis Trimestre e Ano são mais altos que o nível Mês definido no SCI. Por isto, o Componente Explorador de Subcubos é capaz de explorar 3 níveis da dimensão Tempo: o nível Mês (definido no SCI) e os níveis Trimestre e Ano (níveis acima de Mês). Na dimensão TipoNegocio, o nível Mobilidade, utilizado no SCI, é o único e portanto o nível mais alto da hierarquia. Assim, o nível Mobilidade é o único nível da dimensão TipoNegocio explorado pelo Componente Explorador de Subcubos. Nota-se, portanto, que o Componente Explorador de Subcubos é capaz de gerar 6 (2 x 3 x 1) cubos a partir do SCI, combinando os níveis descritos acima.

## 4.2. Mineração de Dados

Para realizar a mineração de dados, os arquivos contendo as células expandidas geradas pela arquitetura DOLAM foram convertidos para o formato ARFF (*Attribute-Relation File Format*), utilizado pelo Weka. O formato ARFF define que cada membro seja definido como um valor de atributo. A definição dos dados a serem minerados consiste em definir a presença ou a ausência do membro na célula. Considere as três células expandidas exibidas na Figura 27.

<p>[Tempo].[All Tempos].[2009].[1].[Janeiro] , [Tempo].[All Tempos].[2009].[1] , [Tempo].[All Tempos].[2009] , [Tipo negocio].[All Tipo negocios].[Fixo] , [Filial].[All Filiais].[FORTALEZA ] , [Filial].[All Filiais].[FORTALEZA ]</p> <p>[Tempo].[All Tempos].[2009].[2].[Junho] , [Tempo].[All Tempos].[2009].[2] , [Tempo].[All Tempos].[2009] , [Tipo negocio].[All Tipo negocios].[Fixo] , [Filial].[All Filiais].[FORTALEZA ] , [Filial].[All Filiais].[FORTALEZA ]</p> <p>[Tempo].[All Tempos].[2009].[3].[Julho] , [Tempo].[All Tempos].[2009].[3] , [Tempo].[All Tempos].[2009] , [Tipo negocio].[All Tipo negocios].[Fixo] , [Filial].[All Filiais].[FORTALEZA ] , [Filial].[All Filiais].[FORTALEZA ]</p>
--

Figura 27 - Exemplo de células expandidas

Para converter as células da Figura 27 para o formato ARFF, primeiramente os membros que compõem as células devem ser identificados. Eles estão listados como elementos *@attribute* na Figura 28. O bloco *@data* se refere aos dados a serem



minerados, sendo assim, cada linha do bloco *data* se refere a uma célula expandida. Em cada linha é definida a presença ou ausência dos atributos na ordem em que foram definidos no bloco *@attribute*. Neste estudo de caso, o elemento {p} indica a presença do atributo na célula, o elemento {?} indica sua ausência. No arquivo ARFF da Figura 28, a primeira coluna de cada linha do bloco *@data* representa a presença do membro [Tempo].[AllTempos].[2009].[1].[Janeiro] na célula em questão. A segunda linha corresponde ao segundo atributo do bloco *@attribute*; no caso, [Tempo].[AllTempos].[2009].[2].[Junho], e assim por diante. Considere a primeira célula expandida da Figura 27. Ela contém os membros [Tempo].[AllTempos].[2009].[1].[Janeiro], [Tempo].[AllTempos].[2009].[1], [Tempo].[AllTempos].[2009], [Tipo negocio].[AllTipoNegocios].[Fixo], [Filial].[AllFilials].[FORTALEZA], [Filial].[AllFilials].[PACAJUS], [Filial].[AllFilials].[FORTALEZA]. Assim, a linha de dados no bloco *@data* correspondente à primeira célula da Figura 27 é representada no arquivo ARFF (Figura 28) como *p,?,?,p,?,?,p,p,p,p*. A Figura 28 apresenta o arquivo ARFF correspondente às três células expandidas da Figura 27.

```
@relation liberacoes

@attribute [Tempo].[AllTempos].[2009].[1].[Janeiro] {p}
@attribute [Tempo].[AllTempos].[2009].[2].[Junho] {p}
@attribute [Tempo].[AllTempos].[2009].[3].[Julho] {p}
@attribute [Tempo].[AllTempos].[2009].[1] {p}
@attribute [Tempo].[AllTempos].[2009].[2] {p}
@attribute [Tempo].[AllTempos].[2009].[3] {p}
@attribute [Tempo].[AllTempos].[2009] {p}
@attribute [Tiponegocio].[AllTiponegocios].[Fixo] {p}
@attribute [Filial].[AllFilials].[FORTALEZA].[PACAJUS] {p}
@attribute [Filial].[AllFilials].[FORTALEZA] {p}

@data
p, ?, ?, p, ?, ?, p, p, p, p
?, p, ?, ?, p, ?, p, p, p, p
?, ?, p, ?, ?, p, p, p, p, p
```

Figura 28 - Arquivo ARFF correspondente às células da Figura 27

### 4.3. *Comparação entre mineração tradicional e mineração com DOLAM*

Para analisar o ganho em quantidade das regras de associação geradas utilizando-se a arquitetura DOLAM, foram elaborados dois cenários. No Cenário 0, as células do SCI da Tabela 14 foram mineradas da forma tradicional, sem o uso da arquitetura DOLAM. Ou seja, nenhum processamento multidimensional ou multinível foi feito. No Cenário 1 (Seção 3.3), as células do Cenário 0 foram expandidas pelo Componente Expansor de Ancestrais definido pela arquitetura DOLAM. Note que os Cenários 0 e 1 utilizam as mesmas células, sendo que, no Cenário 1, elas foram expandidas pela arquitetura DOLAM. Por exemplo, considere a primeira célula superior, esquerda do SCI da Tabela 14 (i.e. mês de Janeiro, cidade de Pacajus e Tipo de Negócio ambulante). No Cenário 0, apenas os 3 membros, referentes às 3 dimensões do SCI, serão minerados, como mostra a Figura 29. No Cenário 1, a arquitetura DOLAM, através do Componente Expansor de Ancestrais, recupera os ancestrais dos 3 membros, resultando em 6 membros, como também mostra a Figura 29. No Cenário 1, todas as 96 células do SCI (Tabela 14) passam por este processo de expansão.

Tradicional	Arquitetura DOLAM (Componente Expansor de Ancestrais)
[Tempo].[All Tempos].[2009].[1].[Janeiro] [Tipo negocio].[All Tipo negocios].[Ambulante] [Filial].[All Filiais].[FORTALEZA].[PACAJUS]	[Tempo].[All Tempos].[2009].[1].[Janeiro] [Tempo].[All Tempos].[2009].[1] [Tempo].[All Tempos].[2009] [Tipo negocio].[All Tipo negocios].[Ambulante] [Filial].[All Filiais].[FORTALEZA].[PACAJUS] [Filial].[All Filiais].[FORTALEZA]

**Figura 29 - Representação de célula no modo Tradicional e expandida pela arquitetura DOLAM**

Para recuperar as células do SCI (Cenário 0), foram gastos 2146 ms, ou 2,146 segundos. Para recuperar e expandir as células do SCI (Cenário 1), foram gastos 2348 ms, ou 2,348 segundos. Ao minerar as células do Cenário 0, nota-se que são geradas apenas regras de associação com suporte e confiança muito baixos. As regras de associação mais fortes geradas no Cenário 0 possuem suporte de 10% e confiança de 50% e são exibidas na Figura 30.

1.	[Filial].[AllFiliais].[FORTALEZA].[PACAJUS]=x 24 ==> [Tiponegocio].[AllTiponegocios].[Ambulante]=x 12	conf:(0.5)
2.	[Filial].[AllFiliais].[FORTALEZA].[CAUCAIA]=x 24 ==> [Tiponegocio].[AllTiponegocios].[Ambulante]=x 12	conf:(0.5)
3.	[Filial].[AllFiliais].[FORTALEZA].[MARACANAU]=x 24 ==> [Tiponegocio].[AllTiponegocios].[Ambulante]=x 12	conf:(0.5)
4.	[Filial].[AllFiliais].[FORTALEZA].[FORTALEZA]=x 24 ==> [Tiponegocio].[AllTiponegocios].[Ambulante]=x 12	conf:(0.5)
5.	[Filial].[AllFiliais].[FORTALEZA].[PACAJUS]=x 24 ==> [Tiponegocio].[AllTiponegocios].[Fixo]=x 12	conf:(0.5)
6.	[Filial].[AllFiliais].[FORTALEZA].[CAUCAIA]=x 24 ==> [Tiponegocio].[AllTiponegocios].[Fixo]=x 12	conf:(0.5)
7.	[Filial].[AllFiliais].[FORTALEZA].[MARACANAU]=x 24 ==> [Tiponegocio].[AllTiponegocios].[Fixo]=x 12	conf:(0.5)
8.	[Filial].[AllFiliais].[FORTALEZA].[FORTALEZA]=x 24 ==> [Tiponegocio].[AllTiponegocios].[Fixo]=x 12	conf:(0.5)

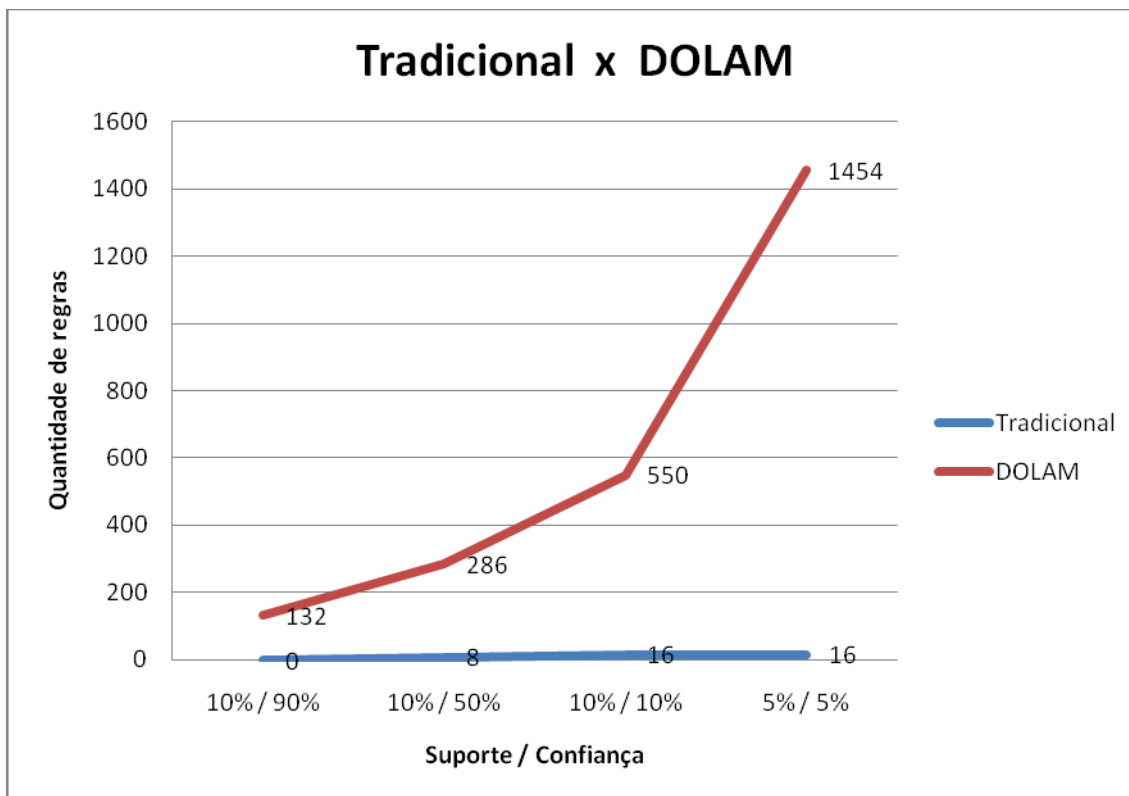
**Figura 30 - Regras mais fortes do Cenário 0**

As regras geradas no Cenário 1 são mais fortes que as do Cenário 0. Existem, por exemplo, 132 regras com suporte de 100% e confiança de 100%, como as exibidas na Figura 31. Por exemplo, a primeira regra de associação da Figura 31 expressa que todas as células mineradas que se encontram na região de Fortaleza pertencem ao ano de 2009.

1.	[Filial].[AllFiliais].[FORTALEZA]=x 96 ==> [Tempo].[AllTempos].[2009]=x 96	conf:(1)
2.	[Tempo].[AllTempos].[2009]=x 96 ==> [Filial].[AllFiliais].[FORTALEZA]=x 96	conf:(1)
3.	[Tiponegocio].[AllTiponegocios].[Ambulante]=x 48 ==> [Tempo].[AllTempos].[2009]=x 48	conf:(1)
4.	[Tiponegocio].[AllTiponegocios].[Fixo]=x 48 ==> [Tempo].[AllTempos].[2009]=x 48	conf:(1)
5.	[Tiponegocio].[AllTiponegocios].[Ambulante]=x 48 ==> [Filial].[AllFiliais].[FORTALEZA]=x 48	conf:(1)
6.	[Tiponegocio].[AllTiponegocios].[Fixo]=x 48 ==> [Filial].[AllFiliais].[FORTALEZA]=x 48	conf:(1)
7.	[Tiponegocio].[AllTiponegocios].[Ambulante]=x [Filial].[AllFiliais].[FORTALEZA]=x 48 ==> [Tempo].[AllTempos].[2009]=x 48	conf:(1)
8.	[Tempo].[AllTempos].[2009]=x [Tiponegocio].[AllTiponegocios].[Ambulante]=x 48 ==> [Filial].[AllFiliais].[FORTALEZA]=x 48	conf:(1)
9.	[Tiponegocio].[AllTiponegocios].[Ambulante]=x 48 ==> [Tempo].[AllTempos].[2009]=x [Filial].[AllFiliais].[FORTALEZA]=x 48	conf:(1)
10.	[Tiponegocio].[AllTiponegocios].[Fixo]=x [Filial].[AllFiliais].[FORTALEZA]=x 48 ==> [Tempo].[AllTempos].[2009]=x 48	conf:(1)

**Figura 31 – 10 regras de associação com 100% de suporte e 100% de confiança geradas pelo Cenário 1**

Além disto, no Cenário 1, através da utilização do Componente Expansor de Ancestrais, as informações de todos os níveis de hierarquia das células são associadas, gerando regras multinível, enquanto que no Cenário 0 apenas o nível mais baixo da hierarquia é associado. Note que no Cenário 0 (Figura 30) não são geradas regras multinível. Os Cenários 0 (tradicional) e 1 (que faz uso de DOLAM) foram executados com diversos valores de suporte e confiança, como apresenta a Figura 32.



**Figura 32 - Comparação entre a quantidade e força das regras nos cenários tradicional e DOLAM**

Através dos dados apresentados na Figura 32, é possível perceber que, ao se utilizar a arquitetura DOLAM, é gerada uma maior quantidade de regras, e estas são significativamente mais fortes. Por exemplo, para 10% de suporte e 90% de confiança, a mineração com DOLAM gera 132 regras, enquanto que a mineração tradicional não é capaz de gerar regra alguma. A mineração tradicional só é capaz de gerar regras com no máximo 10% de suporte e 50% de confiança, o que não configura regras muito significativas. Para estes parâmetros, a mineração tradicional gera 8 regras, enquanto que a mineração com DOLAM gera 286 regras. A mineração tradicional gera 16 regras com 10% de suporte e 10% de confiança, o que configura regras extremamente fracas, enquanto que a mineração com DOLAM gera 550 regras. Para valores de 5% de suporte e 5% de confiança, a mineração tradicional gera as mesmas 16 regras, enquanto que a mineração com DOLAM gera 1454 regras.

A grande diferença na força e na quantidade das regras da mineração tradicional e da mineração com DOLAM (Cenários 0 e 1, respectivamente) ocorre porque a arquitetura DOLAM é capaz identificar as relações multinível existentes entre as células do SCI, enquanto que a forma tradicional não é capaz de fazê-lo. Por exemplo, as regras

apresentadas na Figura 31 mostram que a arquitetura DOLAM (Cenário 1) foi capaz de identificar que todas as células do SCI se referem à região de Fortaleza e ao ano de 2009, enquanto que a forma de mineração de dados tradicional, utilizada no Cenário 0 não foi capaz de detectar esta similaridade entre as células. Desta forma, demonstra-se que, no processo de mineração de células de cubos de dados OLAP, ao se utilizar o Componente Expansor de Ancestrais, as regras de associação geradas pelos algoritmos mineradores são capazes de captar semelhanças entre os diferentes níveis de abstração das células.

Para mostrar a utilidade dos demais componentes propostos pela arquitetura DOLAM, dois cenários de utilização da arquitetura apresentados na Seção 3.3 são detalhados a seguir. O Cenário 3 foi escolhido pois é o cenário de maior custo computacional, uma vez que este considera todas as células de todos os subcubos gerados pelo Componente Explorador de Subcubos. Por sua vez, o Cenário 4 foi escolhido pois corresponde ao cenário mais completo, uma vez que este utiliza todos os componentes da arquitetura DOLAM.

#### **4.4. Cenário de maior custo computacional na arquitetura DOLAM**

No Cenário 3, conforme visto anteriormente (Seção 3.3), o subcubo inicial definido pelo SCI (Tabela 14) é expandido pelo Componente Explorador de Subcubos em 6 subcubos que, somados, possuem 866 células. Para a geração das 866 células foram necessários 229001 ms, ou seja, cerca de 4 minutos. Todas estas células foram expandidas pelo Componente Expansor de Ancestrais e então foram mineradas através do Weka. Tanto o algoritmo *APriori* quanto o *FP-Growth* geraram 132 regras. Analisando-se as 10 primeiras regras geradas pelo algoritmo *APriori*, verificou-se que todas também foram geradas pelo algoritmo *FP-Growth*. Algumas regras foram geradas em ordem diferente devido ao funcionamento interno dos algoritmos, mas as regras eram idênticas, inclusive nas métricas suporte e confiança. A Figura 33 exibe as 10 primeiras regras geradas pelo algoritmo *APriori* e as regras respectivas geradas pelo algoritmo *FP-Growth*.

### APriori

1. [Filial].[AllFilials].[FORTALEZA]=x 866 ==> [Tempo].[AllTempos].[2009]=x 866 conf:(1)
2. [Tempo].[AllTempos].[2009]=x 866 ==> [Filial].[AllFilials].[FORTALEZA]=x 866 conf:(1)
3. [Tiponegocio].[AllTiponegocios].[Ambulante]=x 433 ==> [Tempo].[AllTempos].[2009]=x 433 conf:(1)
4. [Tiponegocio].[AllTiponegocios].[Fixo]=x 433 ==> [Tempo].[AllTempos].[2009]=x 433 conf:(1)
5. [Tiponegocio].[AllTiponegocios].[Ambulante]=x 433 ==> [Filial].[AllFilials].[FORTALEZA]=x 433 conf:(1)
6. [Tiponegocio].[AllTiponegocios].[Fixo]=x 433 ==> [Filial].[AllFilials].[FORTALEZA]=x 433 conf:(1)
7. [Tiponegocio].[AllTiponegocios].[Ambulante]=x [Filial].[AllFilials].[FORTALEZA]=x 433 ==> [Tempo].[AllTempos].[2009]=x 433 conf:(1)
8. [Tempo].[AllTempos].[2009]=x [Tiponegocio].[AllTiponegocios].[Ambulante]=x 433 ==> [Filial].[AllFilials].[FORTALEZA]=x 433 conf:(1)
9. [Tiponegocio].[AllTiponegocios].[Ambulante]=x 433 ==> [Tempo].[AllTempos].[2009]=x [Filial].[AllFilials].[FORTALEZA]=x 433 conf:(1)
10. [Tiponegocio].[AllTiponegocios].[Fixo]=x [Filial].[AllFilials].[FORTALEZA]=x 433 ==> [Tempo].[AllTempos].[2009]=x 433 conf:(1)

### FP-Growth

2. [[Filial].[AllFilials].[FORTALEZA]=x]: 866 ==> [[Tempo].[AllTempos].[2009]=x]: 866 <conf:(1)> lift:(1) lev:(0) conv:(0)
1. [[Tempo].[AllTempos].[2009]=x]: 866 ==> [[Filial].[AllFilials].[FORTALEZA]=x]: 866 <conf:(1)> lift:(1) lev:(0) conv:(0)
4. [[Tiponegocio].[AllTiponegocios].[Ambulante]=x]: 433 ==> [[Tempo].[AllTempos].[2009]=x]: 433 <conf:(1)> lift:(1) lev:(0) conv:(0)
3. [[Tiponegocio].[AllTiponegocios].[Fixo]=x]: 433 ==> [[Tempo].[AllTempos].[2009]=x]: 433 <conf:(1)> lift:(1) lev:(0) conv:(0)
14. [[Tiponegocio].[AllTiponegocios].[Ambulante]=x]: 433 ==> [[Filial].[AllFilials].[FORTALEZA]=x]: 433 <conf:(1)> lift:(1) lev:(0) conv:(0)
13. [[Tiponegocio].[AllTiponegocios].[Fixo]=x]: 433 ==> [[Filial].[AllFilials].[FORTALEZA]=x]: 433 <conf:(1)> lift:(1) lev:(0) conv:(0)
28. [[Filial].[AllFilials].[FORTALEZA]=x, [Tiponegocio].[AllTiponegocios].[Ambulante]=x]: 433 ==> [[Tempo].[AllTempos].[2009]=x]: 433 <conf:(1)> lift:(1) lev:(0) conv:(0)
27. [[Tempo].[AllTempos].[2009]=x, [Tiponegocio].[AllTiponegocios].[Ambulante]=x]: 433 ==> [[Filial].[AllFilials].[FORTALEZA]=x]: 433 <conf:(1)> lift:(1) lev:(0) conv:(0)
26. [[Tiponegocio].[AllTiponegocios].[Ambulante]=x]: 433 ==> [[Tempo].[AllTempos].[2009]=x, [Filial].[AllFilials].[FORTALEZA]=x]: 433 <conf:(1)> lift:(1) lev:(0) conv:(0)
25. [[Filial].[AllFilials].[FORTALEZA]=x, [Tiponegocio].[AllTiponegocios].[Fixo]=x]: 433 ==> [[Tempo].[AllTempos].[2009]=x]: 433 <conf:(1)> lift:(1) lev:(0) conv:(0)

Figura 33 - Regras de associação geradas no Cenário 3

Note que as regras de associação geradas são da forma multidimensional e multinível, como proposto pela arquitetura DOLAM. Na Figura 33, a regra de associação 3 gerada pelo algoritmo APriori relaciona o membro [Tiponegocio].[AllTiponegocios].[Ambulante] da dimensão TipoNegocio, com o membro [Tempo].[AllTempos].[2009] da dimensão Tempo. Portanto, a regra 3 é multidimensional do tipo inter-dimensional, pois relaciona dimensões diferentes, por consequência, é inter-nível. Isto é, se a regra relaciona dimensões diferentes, por consequência, também relaciona níveis diferentes. Resumindo, se a regra é inter-dimensional ela é necessariamente inter-nível.

A Figura 34 exibe duas regras de associação geradas pelos algoritmos APriori e FP-Growth. A regra 13 do algoritmo APriori corresponde à regra 8 do algoritmo FP-Growth. Da mesma forma, a regra 93 do algoritmo APriori corresponde à regra 65 do algoritmo FP-Growth.

<p><b>APriori</b></p> <p>13. [Tempo].[AllTempos].[2009].[1]=x 212 ==&gt; [Tempo].[AllTempos].[2009]=x 212 conf:(1)</p> <p>93. [Tiponegocio].[AllTiponegocios].[Ambulante]=x [Filial].[AllFilials].[FORTALEZA].[PACAJUS]=x 90 ==&gt; [Tempo].[AllTempos].[2009]=x 90 conf:(1)</p> <p><b>FP-Growth</b></p> <p>8. [[Tempo].[AllTempos].[2009].[1]=x]: 212 ==&gt; [[Tempo].[AllTempos].[2009]=x]: 212 &lt;conf:(1)&gt; lift:(1) lev:(0) conv:(0)</p> <p>65. [[Tiponegocio].[AllTiponegocios].[Ambulante]=x, [Filial].[AllFilials].[FORTALEZA].[PACAJUS]=x]: 90 ==&gt; [[Tempo].[AllTempos].[2009]=x]: 90 &lt;conf:(1)&gt; lift:(1) lev:(0) conv:(0)</p>
---

**Figura 34 - Exemplos de regras de associação geradas no cenário 3**

Note que a regra 13 gerada pelo algoritmo *APriori* (Figura 34) relaciona os membros [Tempo].[AllTempos].[2009].[1] e [Tempo].[AllTempos].[2009]. Ambos pertencem à dimensão Tempo e se encontram em níveis diferentes (nível Trimestre e nível Ano, respectivamente). Portanto, a regra 13 gerada pelo algoritmo *APriori* é multidimensional do tipo intra-dimensional, pois relaciona membros da mesma dimensão, e multinível do tipo inter-nível, pois relaciona membros de níveis diferentes.

Assim, no Cenário 3, foram gerados os dois tipos de regras de associação possíveis: inter-dimensional inter-nível e intra-dimensional inter-nível.

#### **4.5. Cenário de utilização de todos os componentes da arquitetura DOLAM**

No Cenário 4, definido na Seção 3.3, foi utilizado o mesmo SCI, que foi expandido em 6 subcubos pelo Componente Explorador de Subcubos. Cada um destes subcubos é explorado pelo Componente Detector de *Outliers*. Os *outliers* detectados são expandidos pelo Componente Expansor de Ancestrais, e em seguida os *outliers* (células) expandidos são minerados utilizando o Weka.

Após a exploração de todos os 6 subcubos, foram identificados 3 *outliers*, todos ocorreram no SCI. Estas operações levaram 4666 ms, ou seja, cerca de 4,6 segundos. Note que a diferença de tempo de execução dos Cenários 3 e 4 é expressiva. Isto se deve ao número de células a serem expandidas. Em ambos os cenários, o SCI é expandido em 6 subcubos. No Cenário 3, todas as células dos 6 subcubos são expandidas, num total de 866 células. Enquanto que no Cenário 4, o Componente Detector de *Outliers* é aplicado aos 6 subcubos e apenas as células detectadas como *outliers* são expandidas, num total de 3 células. Portanto, a diferença de tempo de execução dos cenários 3 e 4 se deve ao fato de que, no Cenário 3, 866 células são expandidas, enquanto que no cenário 4, apenas 3 células são expandidas.

A Tabela 15 exibe o SCI definido pelo usuário, as células destacadas em laranja correspondem aos *outliers* detectados.



				Filial			
Tempo			Tipo negocio	FORTALEZA			
Ano	Trimestre	Mes	Tipo de negocio	PACAJUS	CAUCAIA	MARACANAU	FORTALEZA
2009	1	+Janeiro	Ambulante	8.754,00	9.436,00	24.471,00	74.266,00
			Fixo	50.223,00	25.526,00	28.397,00	155.644,00
		+Fevereiro	Ambulante	4.898,00	7.509,00	9.214,00	36.391,00
			Fixo	21.147,00	31.902,00	23.009,00	134.150,00
		+Março	Ambulante	19.000,00	11.948,00	38.120,00	78.166,00
			Fixo	17.298,00	14.250,00	70.793,00	240.219,00
	2	+Abril	Ambulante	3.971,00	6.047,00	9.345,00	83.188,00
			Fixo	13.817,00	37.053,00	36.898,00	153.126,00
		+Maio	Ambulante	17.001,00	16.203,00	18.400,00	105.544,00
			Fixo	15.979,00	12.423,28	37.462,00	155.501,00
		+Junho	Ambulante	13.486,00	8.165,00	28.730,00	32.432,00
			Fixo	56.022,00	43.267,00	43.649,00	193.766,00
	3	+Julho	Ambulante	12.222,00	5.284,00	19.410,00	64.031,00
			Fixo	42.310,00	27.258,00	50.030,00	198.879,00
		+Agosto	Ambulante	5.491,00	1.362,00	24.545,00	85.907,00
			Fixo	24.775,00	5.518,00	60.720,50	149.376,00
		+Setembro	Ambulante	15.386,00	8.904,00	20.144,00	100.438,00
			Fixo	10.347,00	17.439,00	56.486,00	112.255,00
	4	+Outubro	Ambulante	21.052,00	13.918,00	11.665,00	58.451,00
			Fixo	32.950,00	53.367,40	47.473,00	161.620,00
		+Novembro	Ambulante	18.590,00	23.060,00	12.772,00	121.654,00
			Fixo	13.696,00	26.422,00	34.258,00	243.224,00
		+Dezembro	Ambulante	16.015,00	8.852,00	13.636,00	12.706,00
			Fixo	3.343,00	16.086,00	20.603,00	111.352,00

Tabela 15 - SCI e outliers detectados

Todos os *outliers* detectados pertencem ao SCI. Isto se deve à natureza de agregação dos dados presentes nos cubos de dados OLAP. Por exemplo, a Tabela 16 exibe o subcubo gerado pelo Componente Explorador de Subcubos resultante de uma operação *roll-up* na dimensão Tempo, indo do nível Mês para o nível Trimestre. Note que as disparidades de valores ocorridas nos meses de Janeiro, Junho e Julho, que contem *outliers* no nível do SCI, não ocorrem quando se analisam seus respectivos Trimestres, porque ao agregar os dados é provável que as discrepâncias se diluam em meio aos demais membros. Desta forma, a ocorrência de um *outlier* em um nível mais alto consiste em um grande alerta. Como o cubo analisado neste estudo de caso é razoavelmente homogêneo, todos os *outliers* detectados pertencem ao SCI.

A Figura 35 exibe as células da Tabela 15 detectadas como *outliers*. A Figura 36 exibe as células da Figura 35 após expandidas pelo Componente Expansor de Ancestrais.

		Filial				
Tempo		Tipo negocio	FORTALEZA			
Ano	Trimestre	Tipo de negocio	PACAJUS	CAUCAIA	MARACANAU	FORTALEZA
2009	+1	Ambulante	32.652,00	28.893,00	71.805,00	188.823,00
		Fixo	88.668,00	71.678,00	122.199,00	530.013,00
	+2	Ambulante	34.458,00	30.415,00	56.475,00	221.164,00
		Fixo	85.818,00	92.743,28	118.009,00	502.393,00
	+3	Ambulante	33.099,00	15.550,00	64.099,00	250.376,00
		Fixo	77.432,00	50.215,00	167.236,50	460.510,00
	+4	Ambulante	55.657,00	45.830,00	38.073,00	192.811,00
		Fixo	49.989,00	95.875,40	102.334,00	516.196,00

Tabela 16 - Subcubo resultante após uma operação roll-up na dimensão Tempo partindo do SCI

[Tempo].[All Tempos].[2009].[1].[Janeiro]	
[Tipo negocio].[All Tipo negocios].[Fixo]	
[Filial].[All Filials].[FORTALEZA	].[PACAJUS
[Tempo].[All Tempos].[2009].[2].[Junho]	
[Tipo negocio].[All Tipo negocios].[Fixo]	
[Filial].[All Filials].[FORTALEZA	].[PACAJUS
[Tempo].[All Tempos].[2009].[3].[Julho]	
[Tipo negocio].[All Tipo negocios].[Fixo]	
[Filial].[All Filials].[FORTALEZA	].[PACAJUS

Figura 35 – Células outliers detectadas

[Tempo].[All Tempos].[2009].[1].[Janeiro]	
[Tempo].[All Tempos].[2009].[1]	
[Tempo].[All Tempos].[2009]	
[Tipo negocio].[All Tipo negocios].[Fixo]	
[Filial].[All Filials].[FORTALEZA	].[PACAJUS
[Filial].[All Filials].[FORTALEZA	]
[Tempo].[All Tempos].[2009].[2].[Junho]	
[Tempo].[All Tempos].[2009].[2]	
[Tempo].[All Tempos].[2009]	
[Tipo negocio].[All Tipo negocios].[Fixo]	
[Filial].[All Filials].[FORTALEZA	].[PACAJUS
[Filial].[All Filials].[FORTALEZA	]
[Tempo].[All Tempos].[2009].[3].[Julho]	
[Tempo].[All Tempos].[2009].[3]	
[Tempo].[All Tempos].[2009]	
[Tipo negocio].[All Tipo negocios].[Fixo]	
[Filial].[All Filials].[FORTALEZA	].[PACAJUS
[Filial].[All Filials].[FORTALEZA	]

Figura 36 - Células outliers expandidas

As células expandidas da Figura 36 foram mineradas pelos algoritmos *APriori* e *FP-Growth* através do Weka, após o processamento detalhado anteriormente. Ambos os algoritmos geraram 1121 regras de associação. As 10 primeiras regras de associação geradas por cada algoritmo são iguais, conforme ilustra a Figura 37.

<p><b>APriori</b></p> <ol style="list-style-type: none"> <li>1. [Tiponegocio].[AllTiponegocios].[Fixo]=p 3 ==&gt; [Tempo].[AllTempos].[2009]=p 3 conf:(1)</li> <li>2. [Tempo].[AllTempos].[2009]=p 3 ==&gt; [Tiponegocio].[AllTiponegocios].[Fixo]=p 3 conf:(1)</li> <li>3. [Filial].[AllFilials].[FORTALEZA].[PACAJUS]=p 3 ==&gt; [Tempo].[AllTempos].[2009]=p 3 conf:(1)</li> <li>4. [Tempo].[AllTempos].[2009]=p 3 ==&gt; [Filial].[AllFilials].[FORTALEZA].[PACAJUS]=p 3 conf:(1)</li> <li>5. [Filial].[AllFilials].[FORTALEZA]=p 3 ==&gt; [Tempo].[AllTempos].[2009]=p 3   conf:(1)</li> <li>6. [Tempo].[AllTempos].[2009]=p 3 ==&gt; [Filial].[AllFilials].[FORTALEZA]=p 3   conf:(1)</li> <li>7. [Filial].[AllFilials].[FORTALEZA].[PACAJUS]=p 3 ==&gt; [Tiponegocio].[AllTiponegocios].[Fixo]=p 3   conf:(1)</li> <li>8. [Tiponegocio].[AllTiponegocios].[Fixo]=p 3 ==&gt; [Filial].[AllFilials].[FORTALEZA].[PACAJUS]=p 3   conf:(1)</li> <li>9. [Filial].[AllFilials].[FORTALEZA]=p 3 ==&gt; [Tiponegocio].[AllTiponegocios].[Fixo]=p 3 conf:(1)</li> <li>10. [Tiponegocio].[AllTiponegocios].[Fixo]=p 3 ==&gt; [Filial].[AllFilials].[FORTALEZA]=p 3 conf:(1)</li> </ol> <p><b>FP-Growth</b></p> <ol style="list-style-type: none"> <li>1. [[Tiponegocio].[AllTiponegocios].[Fixo]=p]: 3 ==&gt; [[Tempo].[AllTempos].[2009]=p]: 3 &lt;conf:(1)&gt; lift:(1) lev:(0) conv:(0)</li> <li>2. [[Tempo].[AllTempos].[2009]=p]: 3 ==&gt; [[Tiponegocio].[AllTiponegocios].[Fixo]=p]: 3 &lt;conf:(1)&gt; lift:(1) lev:(0) conv:(0)</li> <li>8. [[Filial].[AllFilials].[FORTALEZA].[PACAJUS]=p]: 3 ==&gt; [[Tempo].[AllTempos].[2009]=p]: 3 &lt;conf:(1)&gt; lift:(1) lev:(0) conv:(0)</li> <li>7. [[Tempo].[AllTempos].[2009]=p]: 3 ==&gt; [[Filial].[AllFilials].[FORTALEZA].[PACAJUS]=p]: 3 &lt;conf:(1)&gt; lift:(1) lev:(0) conv:(0)</li> <li>10. [[Filial].[AllFilials].[FORTALEZA]=p]: 3 ==&gt; [[Tempo].[AllTempos].[2009]=p]: 3 &lt;conf:(1)&gt; lift:(1) lev:(0) conv:(0)</li> <li>9. [[Tempo].[AllTempos].[2009]=p]: 3 ==&gt; [[Filial].[AllFilials].[FORTALEZA]=p]: 3 &lt;conf:(1)&gt; lift:(1) lev:(0) conv:(0)</li> <li>4. [[Filial].[AllFilials].[FORTALEZA].[PACAJUS]=p]: 3 ==&gt; [[Tiponegocio].[AllTiponegocios].[Fixo]=p]: 3   &lt;conf:(1)&gt; lift:(1) lev:(0) conv:(0)</li> <li>3. [[Tiponegocio].[AllTiponegocios].[Fixo]=p]: 3 ==&gt; [[Filial].[AllFilials].[FORTALEZA].[PACAJUS]=p]: 3   &lt;conf:(1)&gt; lift:(1) lev:(0) conv:(0)</li> <li>6. [[Filial].[AllFilials].[FORTALEZA]=p]: 3 ==&gt; [[Tiponegocio].[AllTiponegocios].[Fixo]=p]: 3 &lt;conf:(1)&gt; lift:(1) lev:(0) conv:(0)</li> <li>5. [[Tiponegocio].[AllTiponegocios].[Fixo]=p]: 3 ==&gt; [[Filial].[AllFilials].[FORTALEZA]=p]: 3 &lt;conf:(1)&gt; lift:(1) lev:(0) conv:(0)</li> </ol>
---

**Figura 37 - Regras de associação geradas no Cenário 4**

Na Figura 37, a regra de associação 1 gerada pelo algoritmo *APriori* relaciona o membro [Tiponegocio].[AllTiponegocios].[Fixo] da dimensão TipoNegocio, com o membro [Tempo].[AllTempos].[2009] da dimensão Tempo. Portanto, a regra 1 é multidimensional do tipo inter-dimensional, pois relaciona membros de dimensões diferentes, e, conseqüentemente, multinível do tipo inter-nível, pois relaciona membros de hierarquias diferentes, e, portanto, níveis diferentes. A Figura 38 exibe duas outras regras de associação geradas através do Weka. A regra 51 do algoritmo *APriori* corresponde à regra 42 do algoritmo *FP-Growth*. Da mesma forma, a regra 61 do algoritmo *APriori* corresponde à regra 27 do algoritmo *FP-Growth*.

<b>APriori</b>	
51. [Tempo].[AllTempos].[2009].[1]=p 1 ==>	
[Tempo].[AllTempos].[2009].[1].[Janeiro]=p 1	conf:(1)
61. [Tempo].[AllTempos].[2009].[2].[Junho]=p 1 ==>	
[Filial].[AllFilials].[FORTALEZA].[PACAJUS]=p 1	conf:(1)
<b>FP-Growth</b>	
42. [Tempo].[AllTempos].[2009].[1]=p: 1 ==>	
[Tempo].[AllTempos].[2009].[1].[Janeiro]=p: 1	<conf:(1)> lift:(3) lev:(0.22) conv:(0.67)
27. [Tempo].[AllTempos].[2009].[2].[Junho]=p: 1 ==>	
[Filial].[AllFilials].[FORTALEZA].[PACAJUS]=p: 1	<conf:(1)> lift:(1) lev:(0) conv:(0)

**Figura 38 - Exemplos de regras de associação geradas no cenário 4**

Note que a regra 51 gerada pelo algoritmo *APriori* (Figura 38) Tempo].[AllTempos].[2009].[1] e [[Tempo].[AllTempos].[2009].[1].[Janeiro]. Ambos pertencem à dimensão Tempo e se encontram em níveis diferentes (nível Trimestre e nível Mês, respectivamente). Portanto, a regra 51 gerada pelo algoritmo *APriori* é multidimensional do tipo intra-dimensional, pois relaciona membros da mesma dimensão, e multinível do tipo inter-nível, pois relaciona membros de níveis diferentes.

Assim, no Cenário 4, foram gerados os dois tipos de regras de associação possíveis: inter-dimensional inter-nível e intra-dimensional inter-nível.

#### **4.6.      *Considerações Finais***

Ao se comparar os Cenários 0 (tradicional) e 1 (DOLAM), quanto à quantidade e força (suporte e confiança) das regras de associação geradas, nota-se que, ao se utilizar a arquitetura DOLAM, são detectadas associações em diversos níveis, fazendo com que a quantidade e força das regras seja superior.

Em qualquer cenário, os algoritmos *APriori* e *FP-Growth* geraram as mesmas regras de associação, provando assim que a arquitetura DOLAM trabalha de forma desacoplada. Em todos os cenários foram gerados os dois tipos de regras de associação possíveis: inter-dimensional inter-nível e intra-dimensional inter-nível.

A seguir são apresentados alguns trabalhos relacionados a este, onde são destacados seus principais pontos.

## 5. Trabalhos Relacionados

Este capítulo discute os principais trabalhos relacionados à mineração multidimensional e multinível de regras de associação em cubos de dados. São apresentadas análises de trabalhos com propostas semelhantes, destacando seu objetivo, vantagens e limitações.

Além de trabalhos relacionados que mineram regras de associação, foi analisado também um trabalho que minera *outliers* em cubos OLAP usando uma função de pontuação de *outliers*. Um trabalho que utiliza MDX com o objetivo de minerar um cubo OLAP também foi analisado.

Ao final, um quadro comparativo das funcionalidades e características é apresentado.

### 5.1. *Adaptive-FP*

*Adaptive-FP* [24] visa criar regras de associação multidimensionais e multinível a partir de uma base de dados transacional. Os autores, após realizarem um estudo comparativo entre os algoritmos *A-priori* [1] e *FP-Growth* [30], optaram por utilizar o *FP-Growth* devido ao fato de ele ser cerca de uma ordem de magnitude mais rápido que o *A-priori*.

Na criação das regras de associação, os autores utilizaram o conceito de suportes flexíveis [15] na aplicação do algoritmo *FP-Growth*. Este conceito visa combater um problema em potencial de mineração em múltiplos níveis. O conceito de suportes flexíveis dita que, em alguns casos, não se deve utilizar o mesmo parâmetro de suporte para todos itens. A escolha do suporte pode ser baseada em 1) na diferença esperada de ocorrência dos itens ou 2) no nível da hierarquia em que eles se encontram.

Utiliza-se o conceito de suportes flexíveis na associação de itens em níveis diferentes na hierarquia porque a confiança de uma regra A que associa itens de nível mais baixo na hierarquia (por exemplo, *Código de Produto*) é certamente menor que a de uma regra B entre elementos no topo da hierarquia (por exemplo, *Marca de Produto*). Mesmo que A seja uma regra forte, com alto suporte, como uma marca envolve produtos com vários códigos, existem muito mais registros com uma dada

*Marca* do que com um dado *Código de Produto*. Além disto, não é interessante gerar regras que expressam o senso comum, uma informação óbvia (como associar a venda de computadores e acessórios para estes.)

Outro aspecto a ser levado em conta na decisão do uso de suportes flexíveis é a diferença esperada de ocorrência dos itens, utilizada por *Adaptive-FP*. Assim, itens cujo padrão de vendas é naturalmente diferente são tratados de forma especial, como leite e anel de diamante. O trabalho sugere que se classifiquem os objetos de uma dimensão, como *Produto*, por frequência esperada de ocorrência. Por exemplo, é comum que 1% das vendas diárias de uma loja X contenham leite. Em contrapartida, a ocorrência de mais de uma venda diária de um anel de diamante constitui uma situação excepcional. Neste caso, haveria de se definir dois valores de suporte: um para produtos normais (leite) e outro para os excepcionais (anel de diamante), onde o primeiro é bem maior que o segundo.

Os autores advogam que em bases de dados transacionais reais, os itens aparecem em diferentes níveis de abstração. Por isto, *Adaptive-FP* recebe como entrada um conjunto de transações que contém itens em diversos níveis da hierarquia. Dada a hierarquia da Figura 39 [24], os autores apresentam a base de dados multidimensional da Figura 40 [24] como exemplo.

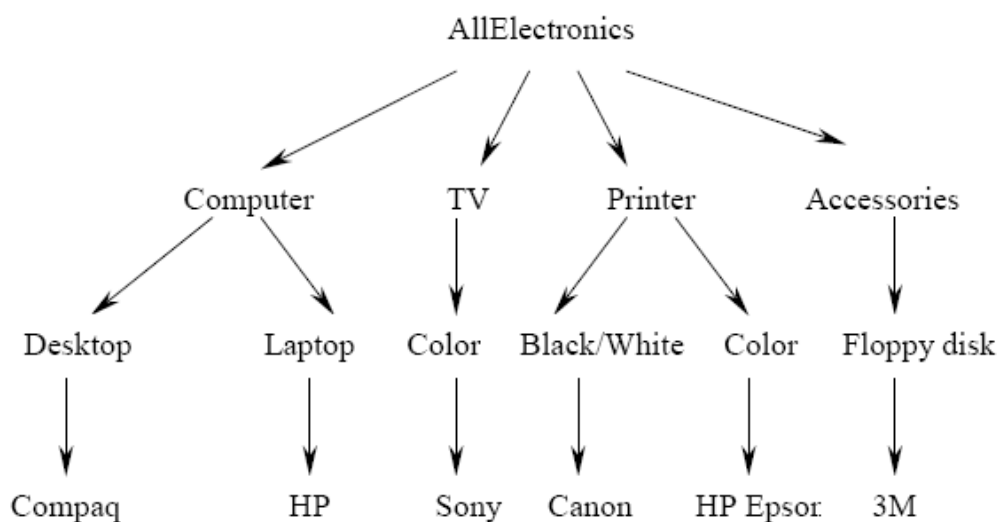


Figura 39 - Hierarquia de Produtos

TID	Store_Location	Items
T <sub>1</sub>	BC	{Compaq Desktop, Color TV, Canon b/w printer, Accessories}
T <sub>2</sub>	ON	{Desktop, Accessories}
T <sub>3</sub>	BC	{Compaq Desktop, Canon b/w printer}
T <sub>4</sub>	ON	{Desktop, Accessories}
T <sub>5</sub>	BC	{Sony Color TV}

**Figura 40 - Exemplo de base de dados**

Entretanto, note que a base de dados multidimensional da Figura 40 não corresponde à realidade do registro de dados neste tipo de base de dados. Em bases de dados multidimensionais, uma vez definida a granularidade de cada dimensão, o registro de um fato ocorre segundo esta granularidade. Ou seja, o registro da venda de um produto sempre considera o menor nível de detalhe sobre o produto. Por exemplo, a hierarquia da Figura 39 apresenta, do mais alto para o mais baixo, os níveis Categoria, Sub-Categoria e Marca. O registro da venda de produto sempre ocorre no nível mais baixo da hierarquia, no exemplo, no nível Marca.

Como na hierarquia Desktop é ancestral de Compaq, isto significa que um Compaq Desktop é uma especialização de Desktop, ou seja, um Compaq Desktop é um tipo de Desktop. Note que cada transação da base de dados da Figura 40 [24] contém itens em vários níveis da hierarquia (Figura 39). Por exemplo, as transações T<sub>1</sub> e T<sub>3</sub> contêm o item Compaq Desktop, que se encontra no nível mais baixo da hierarquia. Na mesma base de dados, transações T<sub>2</sub> e T<sub>4</sub> contêm o item Desktop, que se encontra no nível imediatamente superior ao item Compaq Desktop. Sendo assim, apesar de os itens Compaq Desktop e Desktop pertencerem a níveis diferentes na hierarquia, ambos estão representados na base de dados da mesma forma - como itens. Além disto, existe uma relação entre estes itens, pois de acordo com a hierarquia um Compaq Desktop é um Desktop. Note, portanto, que a entrada considerada pelo trabalho é uma de dados transacional (Figura 40) que possui elementos em diferentes níveis (por exemplo, Compaq Desktop e Desktop). Utilizando a hierarquia da Figura 40 e a base de dados da Figura 39, a técnica propõe a contagem de itens apresentada na Figura 41, adaptada de [24].



	counts
Compaq Desktop	2
Color TV	1
Sony Color TV	1
Canon b/w printer	2
Desktop	2
Accessories	3
BC Stores	3
ON Stores	2

**Figura 41 - Contagem dos elementos das transações da Figura 40**

Note que o item *Desktop* aparece duas vezes na base de dados da Figura 40, nas transações  $T_1$  e  $T_3$ . O item *Compaq Desktop* aparece também duas vezes na base de dados da Figura 40, nas transações  $T_2$  e  $T_4$ . Através da hierarquia, sabe-se que *Compaq Desktop* é uma especialização de *Desktop*. Semanticamente, um *Compaq Desktop* é um *Desktop*. Neste contexto, ocorreu a venda de quatro *Desktops*. Dois foram registrados como *Compaq Desktop* ( $T_1$  e  $T_3$ ), e outros dois foram registrados como *Desktop* ( $T_2$  e  $T_4$ ). Para se considerar uma solução multinível correta, a contagem do item *Desktop* deveria ser 4, e não 2, como foi processado por *Adaptive-FP*. Ao desconsiderar a hierarquia entre *Desktop* e *Compaq Desktop*, o trabalho compromete a contagem de itens (counts). Como a contagem (counts) é a principal métrica usada para o cálculo do suporte, o valor de suporte gerado por *Adaptive-FP* é questionável.

De acordo com esta discussão, pode-se observar que o *Adaptive-FP* minera os dados da maneira convencional, sem analisar os níveis em que os itens se encontram. As regras geradas por *Adaptive-FP* apresentam itens em diversos níveis apenas devido a entrada do algoritmo ser composta por itens de diversos níveis. Ou seja, os itens *Compaq Desktop* e *Desktop* são minerados da forma convencional, sem observar a relação de hierarquia presente entre eles, e ao final geram-se regras que contêm estes itens. Como eles pertencem a níveis diferentes, os autores dizem que geram regras multinível. De fato, as regras geradas são multinível, pois contêm itens em diferentes níveis da hierarquia. Entretanto, isto se deve unicamente ao fato da entrada do algoritmo

conter transações cujos itens se encontram em vários níveis. Adaptive-FP não realiza nenhum tratamento multinível nos dados, tornando sua abordagem multinível questionável. Além disto, a técnica proposta gera regras multidimensionais, pois, faz cruzamento de dados considerando diferentes dimensões.

Concluem-se como pontos positivos do trabalho a adoção de suportes flexíveis e a capacidade de gerar regras de associação multidimensional. Entretanto, sua abordagem multinível é questionável, e a estrutura da base de dados utilizada não corresponde à realidade.

## **5.2.ML-T2L1**

Este trabalho [13] tem como objetivo minerar regras de associação multinível em bases de dados transacionais multinível. A mineração de dados é guiada pelo usuário, o qual define um sub-conjunto da base de dados a ser minerado a partir dos atributos de interesse. ML-T2L1 não se propõe a minerar bases de dados multidimensionais.

O trabalho assume que os usuários de mineradores de dados normalmente estão mais interessados em uma parte específica da base de dados. Por este motivo, o usuário deve selecionar uma porção da base de dados. Esta abordagem, além de tornar a busca mais focada no objetivo, restringe o espaço de busca, tornando a execução da mineração mais rápida. Além de restringir a base de dados, ML-T2L1 restringe também a busca de itens multinível. Para isto, os autores partem da premissa de que o usuário tem mais interesse pelos itens cujos correspondentes de nível mais alto (“pais”, “antecessores”) possuem suporte relativamente alto. Usando uma abordagem *top-down*, ML-T2L1 não aprofunda itens cujos descendentes não alcançam o suporte mínimo, uma vez que nem seu ancestral (correspondente de nível mais alto) o alcançou. Ao eliminar ancestrais pouco freqüentes, as regras geradas se tornam mais fortes.

Para facilitar a manipulação de grandes conjuntos de dados transacionais, ML-T2L1 adotou um modelo relacional estendido, que permite que o valor de um atributo seja um conjunto de valores (ou seja, não obedecendo a primeira forma normal). Segundo os autores, ML-T2L1 pode ser aplicado a bases de dados relacionais com alterações mínimas. Esta afirmação é questionável, uma vez que um banco de dados relacional deve obedecer à primeira forma normal.

ML-T2L1 trabalha da seguinte forma. Primeiramente, a base de dados é restrita ao objeto de exploração do usuário. Para isto, o usuário seleciona através de uma linguagem *SQL-like* [13] uma porção da base de dados e também um conjunto de atributos de interesse. Para criar as regras, ML-T2L1 trabalha apenas com os atributos de interesse selecionados. O Quadro 1 [13] ilustra um exemplo de seleção da base de dados para geração de regras de associação. Esta consulta seleciona as transações e seus respectivos itens (tabelas *Sales\_transactions* e *Sales\_items*), onde a categoria (*category*) do item deve ser “food” e o período de armazenamento (*storage\_period*) deve ser menor que 21 dias. Os atributos de interesse são *category*, *content* e *brand*.

```

discover association rules
from sales_transactions T, sales_item I
where T.bar_code = I.bar_code and I.category = “food” and
        I.storage_period < 21
with interested attributes category, contend, brand

```

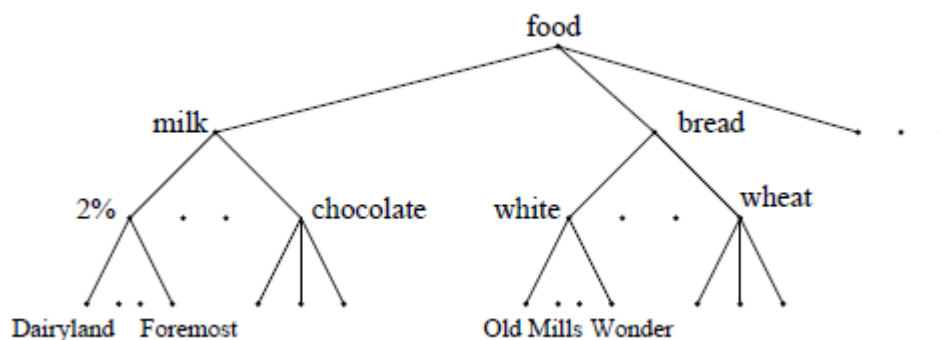
**Quadro 1 - Exemplo de consulta para geração de regras de associação**

Em seguida, a tabela que contém os itens definidos na cláusula “*with interested attributes*” é agregada segundo estes atributos. Os itens agregados são codificados de acordo com sua hierarquia. Após serem agregados, os itens são codificados de acordo com a hierarquia dos itens. A Figura 42 estabelece uma hierarquia entre os itens presentes na base de dados agregada (Tabela 17). Note que a hierarquia possui três níveis, e que a agregação ocorre no nível mais baixo. Por este motivo, os itens codificados contêm três caracteres. O primeiro caractere de um item codificado corresponde ao seu valor para o atributo de nível mais alto na hierarquia.

TID	Items
$T_1$	{111, 121, 211, 221}
$T_2$	{111, 211, 222, 323}
$T_3$	{112, 122, 221, 411}
$T_4$	{111, 121}
$T_5$	{111, 122, 211, 221, 413}
$T_6$	{211, 323, 524}
$T_7$	{323, 411, 524, 713}

**Tabela 17 - Base de dados agregada a ser minerada**

Considere o exemplo de tabela agregada na Tabela 17 e a hierarquia da Figura 42. Todos os itens da categoria “milk” são codificados começando pelo caractere 1. Os itens da categoria “bread” começam com o caractere 2. A codificação é feita de forma semelhante para os demais níveis. Assim, os itens da categoria “milk” que tiverem como sub-categoria “2%” começam com os caracteres “11”. Aqueles que são da sub-categoria “chocolate” começam com “12”. Esta codificação ocorre para todos os itens da tabela agregada, de forma que é possível obter a codificação de um item na árvore hierárquica (Figura 42) fazendo uma leitura em pré-ordem. Por exemplo, o item codificado 111 corresponde a “*food-milk-2%-Dairyland*”.



**Figura 42 - Hierarquia contendo atributos de interesse definidos pelo usuário**

Devido à forma como a codificação foi feita, a similaridade entre os caracteres dos itens codificados denota que eles compartilham parte da hierarquia. Por exemplo, os itens codificados 111 e 121 possuem em comum o ancestral de nível mais alto da hierarquia, pois ambos começam com “1” (referente a “*milk*”). Os itens 111 e 112 possuem em comum os dois ancestrais imediatamente superiores a eles, uma vez que ambos começam com “11” (referente a “*milk-2%*”). Nota-se, portanto, que a codificação de um item contém toda a sua hierarquia.

Os itens codificados presentes nas transações da Tabela 17 são a entrada do algoritmo de criação de regras de associação. Primeiramente, é analisado o primeiro caractere de cada item, correspondente ao nível mais alto (nível 1). Segundo esta análise, na base de dados da Tabela 17 existem três tipos de itens: aqueles que começam com 1, 2 ou 3, representados respectivamente por  $\{1^{**}\}$ ,  $\{2^{**}\}$  e  $\{3^{**}\}$ . Existem itens do tipo  $\{1^{**}\}$  em cinco transações ( $T_1$ ,  $T_2$ ,  $T_3$ ,  $T_4$  e  $T_5$ ) desta base de dados. Existem itens do tipo  $\{2^{**}\}$  em cinco transações ( $T_1$ ,  $T_2$ ,  $T_4$ ,  $T_5$  e  $T_6$ ), e itens do tipo  $\{3^{**}\}$  em 2

transações ( $T_5$  e  $T_6$ ). ML-T2L1 trabalha com suportes flexíveis de acordo com o nível hierárquico. Ou seja, cada nível possui um suporte mínimo definido. Considere que o suporte (contagem) mínimo definido para o nível 1 como 4. Desta forma, apenas os itens dos tipos  $\{1^{**}\}$  e  $\{2^{**}\}$  alcançam este suporte e portanto continuarão a ser explorados no próximo nível (nível 2). Os itens do tipo  $\{3^{**}\}$  não serão mais explorados por ML-T2L1, pois não alcançaram o suporte mínimo. Os itens que alcançam o suporte mínimo são combinados em *itemsets*, como define o algoritmo *APriori*. ML-T2L1 adota uma abordagem de aprofundamento progressivo, onde após explorar um nível, ele segue para o nível imediatamente inferior.

Analisando-se ML-T2L1, verifica-se que ele realiza contagem e descarte dos *itemsets* de níveis mais altos com o objetivo de minimizar a geração de *itemsets* do nível mais baixo que não alcançam suporte mínimo. Por exemplo, a criação e contagem de *itemsets* como  $\{1^{**}\}$  e  $\{11^*\}$  tem como objetivo evitar a geração de *itemsets* de nível 3 como  $\{122\}$  ou  $\{112\}$  que não alcançarão o suporte mínimo. Para tal, a abordagem *top-down* com descarte de *itemsets* não freqüentes foi adotada. Desta forma, ao final do processo, obtem-se apenas *itemsets* que obedecem às restrições de suporte. Os *itemsets* resultantes possuem apenas itens no nível mais baixo da hierarquia. As regras de associação são geradas a partir dos *itemsets* resultantes, seguindo o tradicional método do algoritmo *A-Priori*. Note que todas as regras de associação geradas por ML-T2L1 contêm elementos do nível mais baixo da hierarquia. Ou seja, apesar de percorrer todos os níveis, ML-T2L1 não gera regras do tipo inter-nível, como por exemplo  $\{12^*, 112\} \rightarrow \{2^{**}\}$  que denota a associação entre um item de nível 2 ( $12^*$ ), um item de nível 3 ( $112$ ) e um item de nível 1 ( $2^{**}$ ).

ML-T2L1 restringe o espaço de busca a partir de uma consulta feita pelo usuário. Nesta consulta o usuário informa a porção dos dados na qual está interessado, além dos atributos que devem ser levados em conta na criação de regras de associação. Note que se outros atributos influenciarem as associações reais entre os itens, eles não serão considerados, uma vez que é o usuário define os atributos que serão levados em conta nas regras de associação. Dado que a mineração de dados se propõe a localizar padrões não triviais e muitas vezes desconhecidos entre os dados, estes atributos de interesse restringem a busca aos padrões triviais, previstos pelo usuário. Além disto, o fator multidimensional não é considerado.

A criação das regras de associação é baseada no algoritmo *APriori*, apesar de atualmente se saber que o algoritmo *FP-Growth* produz regras de forma mais eficiente. Concluem-se como pontos positivos do trabalho a adoção de suportes flexíveis de acordo com o nível hierárquico, além da utilização da hierarquia reduzir o número de *itemsets* gerados. Entretanto, o trabalho gera apenas regras de associação entre itens do nível mais baixo da hierarquia.

### 5.3. **EMARDC**

EMARDC (*Enhanced Mining of Association Rules from Data Cubes*) [26] visa minerar cubos de dados para definir regras de associação inter-dimensional a partir de meta-regras definidas pelo usuário. Estas meta-regras guiam a mineração de dados. Elas definem um subcubo a ser minerado, além de selecionar os predicados de dimensão. Os predicados de dimensão definem os níveis cujos membros devem compor o antecedente e conseqüente das regras de associação. O trabalho permite que os predicados de dimensão pertençam a dimensões diferentes, gerando assim regras inter-dimensionais. Os aspectos multinível do cubo de dados não são explorados. Desta forma, o trabalho gera regras de associação inter-dimensionais em apenas um nível.

Por exemplo, suponha que o cubo apresentado na Figura 43 [26], além das três dimensões apresentadas na figura – *Shop* ( $D_1$ ), *Product* ( $D_2$ ) e *Time* ( $D_3$ ) – possua ainda as dimensões *Profile* ( $D_4$ ), *Profession* ( $D_5$ ), *Gender* ( $D_6$ ) e *Promotion* ( $D_7$ ). O Quadro 2 (13) apresenta um exemplo de meta-regra que define que as regras de associação serão mineradas no subcubo definido pelos membros (*Student*, *Female*). Ou seja, o subcubo se refere à população de mulheres (na dimensão *Profile* –  $D_4$ ) estudantes (na dimensão *Profession* –  $D_5$ ). As demais dimensões não interferem no processo de mineração. A meta-regra define os predicados de dimensão que devem compor o antecedente (LHS) e o conseqüente (RHS) das serem geradas. A meta-regra apresentada no Quadro 2 define, portanto, que o antecedente das regras geradas deve conter apenas membros da dimensão *Shop* no nível *Continent* e membros da dimensão *Time* no nível *Year*. Da mesma forma, o conseqüente deve conter apenas membros da dimensão *Product* no nível *Article*.

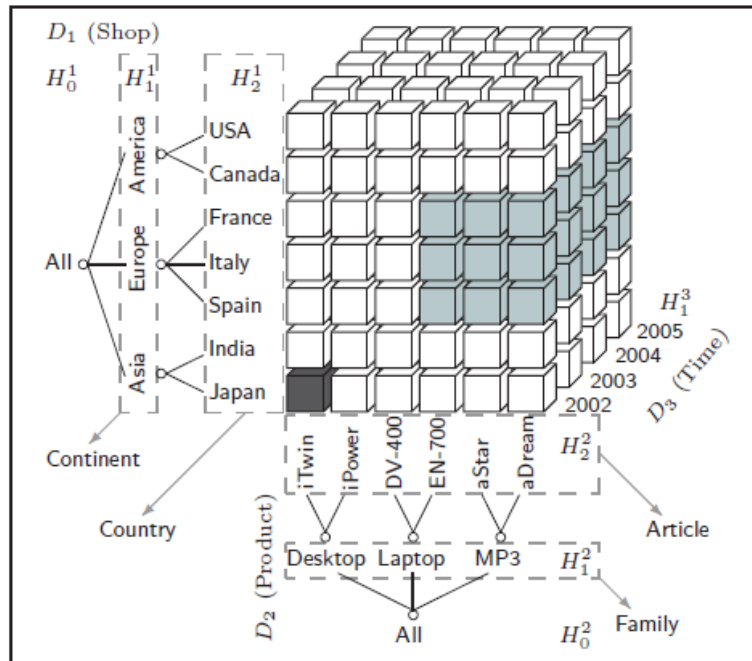


Figura 43 - Cubo de dados, incluindo dimensões, níveis e membros

In the context (Student, Female)  
 $\langle a_1 \in \text{Continent} \rangle \wedge \langle a_3 \in \text{Year} \rangle \Rightarrow \langle a_2 \in \text{Article} \rangle$

Quadro 2 - Exemplo de meta-regra

Note que o minerador deve gerar apenas regras de associação que obedecem às restrições de antecedente e conseqüente definidas na meta-regra. Assim o processo de mineração é focado no interesse do usuário, mas pode deixar de detectar regras fortes cujos membros não foram previstos pelo usuário quando ele definiu a meta-regra a ser explorada. Uma abordagem semelhante foi proposta por [13].

Além de mineração baseada em meta-regras, o trabalho redefiniu também os conceitos de suporte e confiança. O cálculo tradicional leva em conta a frequência de ocorrência dos itens (métrica *COUNT*). Em EMARDC, o usuário tem a opção de utilizar a função de agregação SUM sobre fatos do cubo. Isto é de grande utilidade, pois muitas vezes é necessário avaliar não apenas quantas vezes um fato ocorreu, mas qual o montante envolvido, como por exemplo, o lucro gerado a partir daquelas transações. Ou seja, um padrão de ocorrência frequente (ou seja, com alto *COUNT*) e baixa representatividade financeira (ou seja, valor baixo do SUM de uma medida que representa valores monetários) pode não ser tão interessante quanto um padrão pouco

menos freqüente porém bem mais representativo quantitativamente (no exemplo, financeiramente). Além do suporte e confiança, o trabalho oferece critérios de avaliação de regras mais significativos, como *Lift* e *Loevinger*.

Conclui-se que o trabalho se limita a gerar regras de associação inter-dimensional e não aborda o aspecto multinível. Um ponto positivo deste trabalho é a utilização da métrica *sum* em vez de *count* na definição do suporte e confiança das regras de associação

#### **5.4. .MGAR**

MGAR (*Mining Generalized Association Rules*) [32] visa localizar associações entre itens de transações de qualquer nível de uma taxonomia, através da generalização dos itens e transações. Além disto, MGAR utiliza a hierarquia de itens para eliminar regras redundantes.

O processo de minerar regras de associação generalizadas pode ser dividido em três partes:

1. Descobrir todos os *itemsets* cujo suporte seja maior que o definido pelo usuário. Estes são denominados *frequent itemsets*.
2. Gerar regras de associação a partir dos *frequent itemsets*. As regras válidas serão aquelas cuja confiança é maior que a definida pelo usuário
3. Excluir todas as regras não interessantes ou redundantes

MGAR apresenta alguns algoritmos para a geração de *frequent itemsets* cujos itens podem ser de qualquer nível da taxonomia. Os algoritmos são capazes de trabalhar com múltiplas taxonomias, modeladas como um grafo direcionado acíclico (DGA – *directed acyclic graph*). Uma vez gerados os *frequent itemsets*, o algoritmo em [2] pode ser usado para gerar regras de associação.

##### Algoritmo 1 – Algoritmo básico

A abordagem básica proposta pelo trabalho para se gerar regras multinível é, durante a contagem de frequência dos itens do algoritmo *APriori*, para todo item de transação, consideram-se também como participantes desta transação todos os ancestrais deste item.



Sendo assim, durante a contagem da frequência dos itens, para cada transação  $T$ , é criada uma transação estendida  $T'$  contendo  $T$  e todos os ancestrais de cada item de  $T$ . A geração das transações estendidas é, portanto, um passo inserido dentro do algoritmo *APriori*. Por exemplo, considere a taxonomia da Figura 44. Em cada transação que contenha o item *Jackets*, a respectiva transação estendida contém os itens *Jackets*, *Outerwear* e *Clothes*, ou seja, participam da transação estendida o item e todos seus ancestrais na hierarquia.

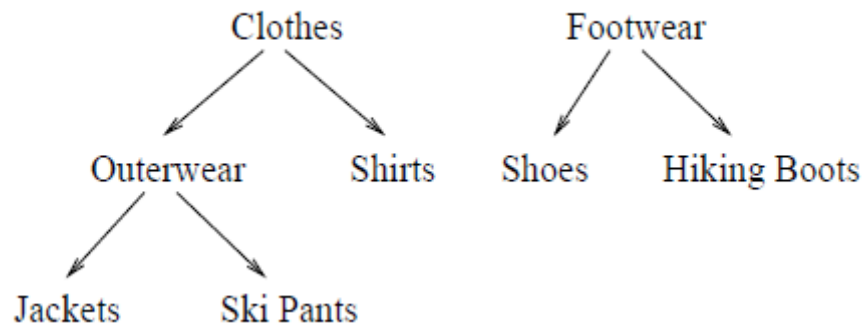


Figura 44 - Exemplo de Hierarquia

Note que o algoritmo Básico insere a criação da transação estendida dentro do algoritmo *APriori*. Portanto, a abordagem de MGAR no algoritmo Básico é fortemente acoplada ao algoritmo *APriori* e não pode ser facilmente adaptada a outro algoritmo de regras de associação. O Algoritmo 7 [32] exibe o pseudo-código do algoritmo básico.

```

 $L_1 := \{\text{frequent 1-itemsets}\};$ 
 $k := 2$  //  $k$  represents the pass number
while ( $L_{k-1} \neq \emptyset$ ) do
  Begin
     $C_k :=$  New candidates of size  $k$  generated from  $L_{k-1}$ .
    forall transactions  $t \in D$  do
      Begin
        Add all ancestors of each item in  $t$  to  $t$ , removing any duplicates
        Increment the count of all candidates in  $C_k$  that are contained in  $t$ .
      end
     $L_k :=$  All candidates in  $C_k$  with minimum support
     $k := k+1$ 
  end
  Answer :=  $\bigcup_k L_k$ 

```

Algoritmo 7 - Algoritmo básico

## Algoritmo 2 – Algoritmo Cumulate

O algoritmo *Cumulate* apresenta três otimizações em relação ao algoritmo básico. São elas:

1. Filtragem dos ancestrais adicionados às transações
2. Pré-processamento dos ancestrais
3. Exclusão dos *frequent itemsets* que contenham um item e seu ancestral

O algoritmo *Cumulate*, diferentemente do algoritmo Básico, não inclui todos os ancestrais dos itens nas transações estendidas. Em cada passe (*pass*) do algoritmo APriori, são inseridos nas transações estendidas apenas os ancestrais que pertencem a um (ou mais) itemsets candidatos que estão sendo contados no passe corrente. Portanto, as transações expandidas do algoritmo *Cumulate* não contem todos os itens da transação original e seus ancestrais.

Por exemplo, considere que o único *itemset* candidato detectado no passe corrente é *{Clothes, Shoes}*. Dada a taxonomia da Figura 44, onde *Clothes* é ancestral de *Jackets*. Segundo o algoritmo *Cumulate*, todas as transações contendo *Jacket*, ao serem estendidas, conterão apenas o item *Clothes*, pois não existe *itemset* candidato que contenha *Jacket* e existe um *itemset* candidato (*{Clothes, Shoes}*) que contem o item *Clothes*. Não é necessário sequer incluir o item *Jacket* na transação estendida, uma vez que este item não trará regras significativas pois não está presente em nenhum *itemset* candidato.

As otimizações apresentadas pelo algoritmo *Cumulate* tem como objetivo evitar processar itens que com certeza não alcançarão o suporte mínimo necessário, evitando assim o desperdício de recursos de processamento e tempo. O Algoritmo 8 ilustra o pseudo-código do algoritmo *Cumulate*.

Note que, da mesma forma que o algoritmo Básico, a abordagem utilizada pelo algoritmo *Cumulate* é fortemente acoplada ao algoritmo APriori.

```

Compute  $T^*$ , the set of ancestors of each item from  $T$  //Optimization 2
 $L_1 := \{\text{frequent 1-itemsets}\}$ ;
 $k := 2$  //k represents the pass number
while ( $L_{k-1} \neq \emptyset$ ) do
  Begin
     $C_k :=$  New candidates of size k generated from  $L_{k-1}$ .
    if ( $k = 2$ ) then
      Delete any candidate in  $C_2$  that consists of an item and its ancestors
    //Optimization 3
      Delete any ancestors in  $T^*$  that are not present in any of the
      candidates in  $C_k$  //Optimization 1
    forall transactions  $t \in D$  do
      begin
        foreach item  $x \in t$  do
          Add all ancestors of  $x$  in  $T^*$  to  $t$ .
          Remove any duplicates from  $t$ .
          Increment the count of all candidates in  $C_k$  that are contained in  $t$ .
        end
       $L_k :=$  All candidates in  $C_k$  with minimum support
       $k := k+1$ 
    end
  Answer  $:= \bigcup_k L_k$ 

```

**Algoritmo 8 - Algoritmo Cumulate**

### Algoritmo 3 – Algoritmo Stratify

O algoritmo *Stratify* usa fundamentos de regras de associação e conhecimento sobre a taxonomia para otimizar a geração de conjuntos candidatos. Considerando que a profundidade de um *itemset*  $X$  que não contem ancestrais é zero (0) e a dos demais *itemsets* é definida por  $\text{depth}(X) = \{\max(\{\text{depth}(X^A) \mid X^A \text{ is a parent of } X\}) + 1$ , o algoritmo *Stratify* propõe que os *itemsets* de menor profundidade sejam contados primeiro, ou seja, que a contagem se dê do topo para a base da hierarquia.

Por exemplo, para a hierarquia da Figura 44, considere que os candidate *itemsets* a serem contados são  $\{\text{Clothes}, \text{Shoes}\}$ ,  $\{\text{Outerwear}, \text{Shoes}\}$ ,  $\{\text{Jacket}, \text{Shoes}\}$ . Se o *itemset*  $\{\text{Clothes}, \text{Shoes}\}$  não alcançar o suporte mínimo, com certeza os *itemsets*  $\{\text{Outerwear}, \text{Shoes}\}$  e  $\{\text{Jacket}, \text{Shoes}\}$  também não o alcançarão, pois ambos os *itemsets* são descendentes de  $\{\text{Clothes}, \text{Shoes}\}$ , uma vez que *Jackets* é descendente de *Outerwear*, que por sua vez é descendente de *Clothes*.

Sendo assim, no exemplo acima, primeiro deve-se contar o suporte do *itemset* {Clothes, Shoes}. Se este alcançar o suporte mínimo, conta-se o suporte de {Outerwear, Shoes}. Se não, nem {Outerwear, Shoes} nem {Jacket, Shoes} serão contados. No caso de {Outerwear, Shoes} ser contado, se ele alcançar o suporte mínimo conta-se também {Jacket, Shoes}.

Entretanto, note que para estratificar as leituras em níveis de profundidade, é possível que o número de passes sobre o banco de dados aumente consideravelmente. No exemplo acima, se todos os *itemsets* alcançarem o suporte mínimo, serão necessários três passes sobre a base de dados, enquanto que apenas um passe é necessário no algoritmo básico. Existe, portanto, um *tradeoff* entre o número de *itemsets* contados (consumo de tempo de CPU) e o número de passes sobre a base de dados (consumo de recursos de IO e tempo de CPU).

Conclui-se que o algoritmo *Stratify* evita a contagem do suporte de *itemsets* que certamente não alcançarão o suporte mínimo, enquanto que o algoritmo *Cumulate* evita a expansão de transações cujos itens certamente não alcançarão o suporte mínimo.

MGAR apresenta ainda os algoritmos *Estimate* e *EstMerge* que trabalham com probabilidades para estimar o suporte de *itemsets* candidatos.

### Criação de Regras interessantes

Após criados os *itemsets* candidatos com um dos algoritmos propostos por MGAR – algoritmo *Básico*, *Cumulate* ou *Stratify* - regras de associação devem ser geradas. MGAR propõe uma forma de eliminar regras redundantes. Uma regra de associação é considerada redundante se a regra, seu suporte e confiança podem ser derivados a partir de outra regra.

Para eliminar regras não interessantes ou redundantes, MGAR utiliza a taxonomia dos itens. O suporte da regra redundante acompanha a proporção entre as vendas e o suporte da regra original. Considere o seguinte exemplo de regra de associação: Leite -> Cereal (8% suporte, 70% confiança). Considere que Leite é ancestral de Leite Desnatado, e que cerca de  $\frac{1}{4}$  das vendas de leite é de leite desnatado. Assim, espera-se que a regra Leite Desnatado -> Cereal tenha suporte 2% e confiança de 70%, pois  $8\% * \frac{1}{4} = 2\%$ . Se de fato estas estimativas de suporte e confiança se

concretizarem, a regra pode ser considerada redundante, pois não prove nenhuma informação diferente da regra original.

Outro critério de definição de regras redundantes é, dada uma regra  $X \rightarrow Y$ , se existir uma regra equivalente que relacione os ancestrais diretos de seus itens, a regra  $X \rightarrow Y$  é considerada redundante. Neste caso, a regra que contem os ancestrais é considerada original e a que contem os descendentes é considerada redundante. Por exemplo, seguindo a taxonomia da Figura 44, considere as regras de associação, seus respectivos suportes e o suporte de seus itens apresentados na Tabela 18.

Rule #	Rule	Support	Item	Support
1	"Clothes $\Rightarrow$ Footwear"	10	Clothes	5
2	"Outerwear $\Rightarrow$ Footwear"	8	Outerwear	2
3	"Jackets $\Rightarrow$ Footwear"	4	Jackets	1

**Tabela 18 - Exemplo de regras de associação e suportes**

Nenhum dos itens da regra 1 da Tabela 18 possui ancestrais, uma vez que tanto o item *Clothes* quanto o item *Footwear* estão no tipo da hierarquia da Figura 44. Como *Clothes* e *Footwear* não possuem ancestrais, não existe regra que relacione seus ancestrais. Por isto, a regra 1 é interessante.

No quadro da porção direita da Tabela 18, o suporte do item *Outerwear* é dado como 2/5 do suporte do item *Clothes*. No quadro da porção esquerda da Tabela 18, a regra 1, onde *Clothes*  $\rightarrow$  *Footwear*, possui suporte 10. Como o RHS, ou conseqüente, das regras 1 e 2 são iguais (*Footwear*), e o suporte de uma regra redundante acompanha a proporção da regra original, o suporte esperado da regra 2 é de  $2/5 * 10 = 4$ . Entretanto, o suporte real da regra é 8 - o dobro do esperado. Como o suporte esperado é diferente do suporte real, a regra 2 é considerada interessante.

Quanto à regra 3, note que seu RHS, ou conseqüente, é igual ao das regras 1 e 2. Na hierarquia da Figura 44, o item *Jackets* é descentente direto de *Outerwear* (regra 2), mas também é descendente de *Clothes* (regra 1). Por este motivo, o suporte esperado da regra 3 pode ser calculado a partir da regra 1 ou da regra 2.

Calculando-se o suporte da regra 3 a partir da regra 1, deve-se levar em conta que o suporte de *Jackets* é  $1/5$  do suporte de *Clothes*. Assim, o suporte esperado da regra 3 é  $1/5 * 10 = 2$ . Segundo esta análise, o suporte real da regra 3 é o dobro do suporte esperado quando calculado a partir da regra 1. Por outro lado, analisando-se a regra 3 a partir da regra 1, como o suporte de *Jackets* é  $1/2$  do suporte de *Outerwear*, seu suporte esperado da regra é  $1/2 * 8 = 4$ . Assim, o suporte real é igual ao previsto a partir da regra 2. Neste caso, a regra 3 é considerada redundante pois, apesar de seu suporte ser o dobro do esperado a partir da regra 1, seu suporte pode ser previsto a partir da regra 2.

Conclui-se que os algoritmos Básico, *Cumulate* e *Stratify* propostos MGAR são capazes de gerar regras de associação multinível através do uso de transações estendidas. Entretanto, sua abordagem é fortemente acoplada, o que torna difícil sua utilização com outros algoritmos de mineração de dados. MGAR utiliza a hierarquia para estimar os valores de suporte e confiança de uma regra, possibilitando assim eliminar regras redundantes, ou seja, regras que podem ser derivadas a partir de outras, denominadas regras interessantes.

### **5.5. *An Outlier-based Data Association Method For Linking Criminal Incidents***

O trabalho “*An Outlier-based Data Association Method for Linking Criminal Incidents*” (AOBDAMFLCI) [23] se propõe a criar regras de associação entre dados de crimes. Ao associar os crimes e seus autores, é possível compreender padrões que ocorrem nos crimes, fazer previsões sobre crimes futuros e detectar seus autores.

A mineração de dados funciona bem quando as informações sobre os incidentes de crime e suspeitos são perfeitamente observadas e armazenadas. Entretanto, isto raramente ocorre na prática. Normalmente a informação disponível é insuficiente para diferenciar criminosos. Por exemplo, em várias ocorrências de roubo, o valor preenchido para o campo “arma” é de “arma de fogo”. Não se pode assumir que eles foram cometidos pelo mesmo criminoso apenas por causa do valor deste campo. No entanto, se houver registros de vários incidentes cometidos usando-se uma arma

incomum, “arma japonesa”, por exemplo, é mais provável que estes crimes tenham sido cometidos pela mesma pessoa. Como “arma japonesa” não é um registro comum, isto a torna mais facilmente distinguível das outras armas, por exemplo, “arma de fogo”. Utilizando outliers, AOBDAMFLCI é capaz de relacionar registros de crimes e distinguir um pequeno grupo de registros dos demais incidentes.

Foi definida uma função  $f$  de pontuação de outliers (*outlier score function - OSF*). Esta função é usada para medir quão extrema é a célula. A associação baseada na OSF é definida através de dois conceitos. O conceito de união (*Union*) é definido da seguinte forma. Sejam  $c_1$  e  $c_2$  duas células. Chama-se  $c$  a união de  $c_1$  e  $c_2$  quando ambos estão contidos em  $c$ , e para cada célula pai  $c'$  contendo  $c_1$  e  $c_2$ ,  $c'$  contem  $c$ . A função  $\text{Union}(c_1, c_2)$  retorna a união de  $c_1$  e  $c_2$ . Para dois incidentes, define-se  $\text{Union}(z_1 \text{ e } z_2) = \text{Union}(\text{Cell}(z_1), \text{Cell}(z_2))$ .

A seguinte regra é usada para associar incidentes: para dois incidentes  $z_1$  e  $z_2$ , diz-se que  $z_1$  e  $z_2$  são associados entre si se, e somente se  $f(\text{Union}(z_1, z_2)) \geq T$ , onde  $T$  é um valor de limite (*threshold*) definido. A complexidade computacional do algoritmo é exponencial de acordo com o número de atributos para o pior caso.

AOBDAMFLCI não tem como objetivo considerar os aspectos multidimensionais ou multiníveis dos cubos OLAP para minerá-los. Seu objetivo detectar outliers através de uma função de pontuação de outliers e encontrar associações existentes entre os outliers detectados.

## **5.6. Discovering the Association Rules in OLAP Data Cube with Daily Downloads of Folklore Materials**

DARODCDDFM (*Discovering the Association Rules in OLAP Data Cube with Daily Downloads of Folklore Materials*) [10] apresenta uma aplicação que faz uso de MDX para descobrir regras de associação em um cubo de dados criado a partir das informações de downloads diários de material do *Bulgarian Academy of Sciences Folclore Institute*. O sistema original contempla uma base de dados OLTP, que mantém informações detalhadas dos documentos e materiais que podem ser baixados do site. Um cubo de dados multidimensional e multinível foi criado a partir da base OLTP. O

processo de construção do cubo encontra-se em [11]. O cubo gerado possui quatro dimensões, e sua hierarquia é apresentada na Figura 45. Um exemplo dos dados do cubo é exibido na Tabela 19.

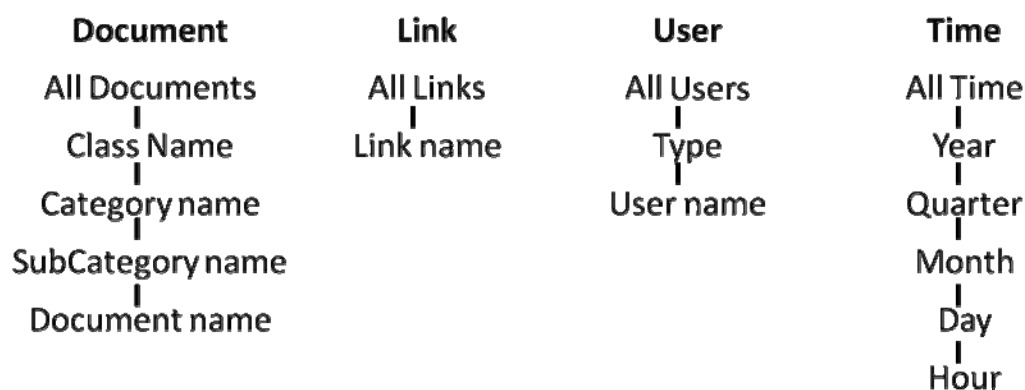


Figura 45 - Hierarquia do cubo gerado

Document	Link	User	Time
Кръщение	songlink062	User0001	19/02/2005
Годеж	songlink013	User0001	09/01/2005
Имен ден	songlink031	User0001	27/11/2004
Имен ден	songlink032	User0001	27/11/2004
Имен ден	songlink031	User0002	20/11/2004
Кръщение	songlink061	User0002	20/11/2004
Семейство	songlink071	User0002	20/11/2004
Годеж	songlink013	User0002	29/12/2004
Годеж	songlink012	User0002	01/03/2005
Годеж	textlink011	User0002	28/05/2005

Tabela 19 - Exemplo de dados do cubo

A medida relacionada aos dados armazena a contagem das linhas correspondentes na base relacional. Os valores de níveis mais altos são obtidos através de sumarizações de seus descendentes. A aplicação descobre regras de associação no cubo de dados usando operações OLAP. Para isto, as linguagens MDX e *Visual Basic* foram utilizadas. Os autores não dão detalhes sobre o processo de mineração, apenas apresentam os procedimentos que o usuário deve realizar para minerar o cubo. Em nenhum momento é possível selecionar o algoritmo minerador a ser utilizado, nem há menção ao acoplamento da arquitetura, o que leva a crer que sua arquitetura é fortemente acoplada.



A aplicação requer que o usuário escolha as dimensões e níveis a serem analisados. Em seguida, os atributos de interesse (membros) devem ser selecionados. Após a definição do valor mínimo de suporte desejado (*min\_sup*), os *k-itemsets* freqüentes são gerados a partir das dimensões e níveis selecionados através de uma consulta MDX.

Ao final são geradas as regras de associação com seus respectivos valores de suporte e confiança. As regras de associação são multidimensionais, como pode ser visto na Figura 46. Não há processamento multinível dos membros minerados.

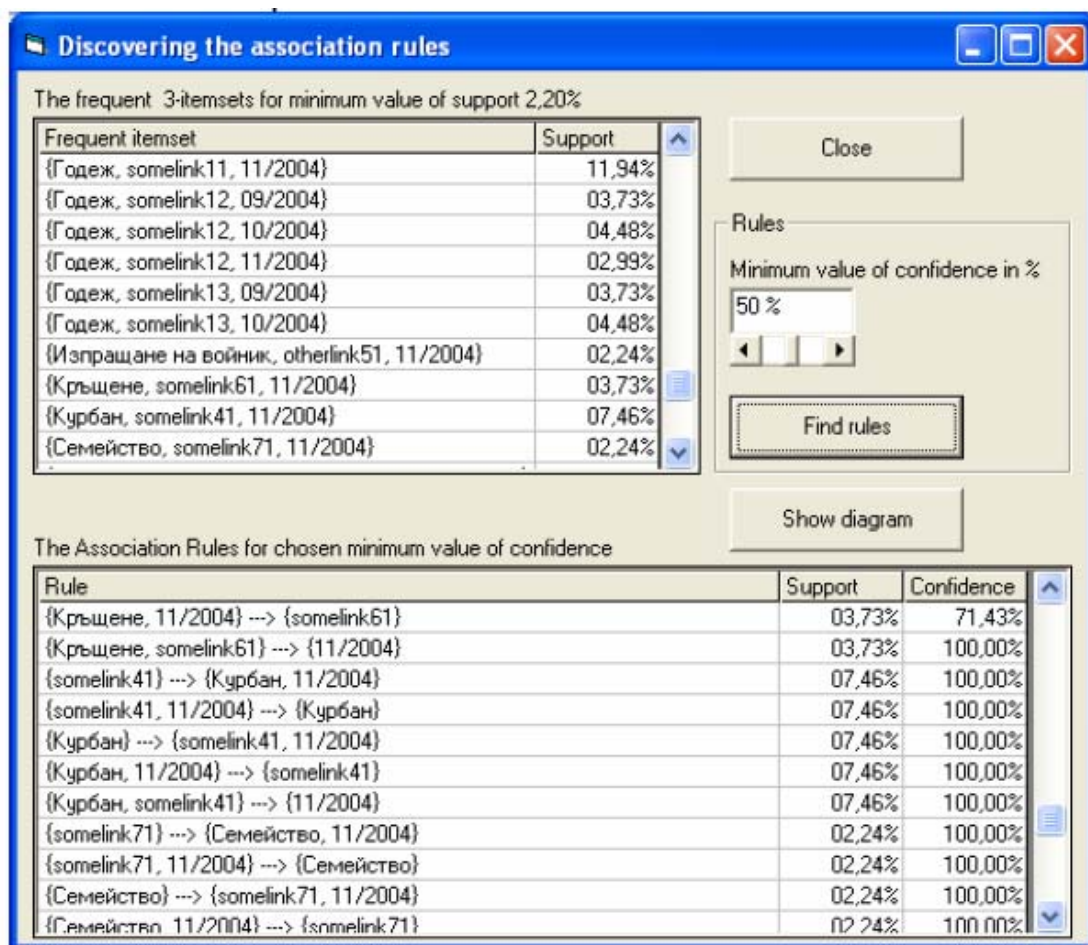


Figura 46 - Regras de associação geradas

## 5.7. Comparação entre os trabalhos

A Tabela 20 exibe um resumo comparativo dos trabalhos relacionados, além do trabalho proposto nesta dissertação. Os trabalhos foram comparados de acordo com as seguintes características:

- Tipo da base de dados minerada (Transacional ou Cubo de dados);
- Algoritmo de mineração (*APriori* ou *FP-Growth*) ;
- Acoplamento ao algoritmo de mineração;
- Aspecto multidimensional;
- Aspecto multinível;
- Características específicas dos trabalhos, em que se baseiam seus processos de mineração de dados.

Apesar de alguns trabalhos operarem sobre bases de dados transacionais, eles apresentam inovações no processamento multidimensional ou multinível. Por exemplo, Adaptive-FP, apesar de não obter sucesso no processamento multinível, é capaz de utilizar suportes flexíveis. ML-T2L1 opera a partir da premissa de que os descendentes de ancestrais fortes provavelmente são também fortes. MGAR define o conceito de regras interessantes com o objetivo de evitar redundância entre regras de associação.

Todos os trabalhos utilizam a métrica *count* para cálculo do suporte e confiança, exceto EMARC, que propõe que a métrica *sum* também seja utilizada, abrindo assim outra forma de análise de força de regras de associação.

Exceto pela arquitetura DOLAM aqui proposta, todos os trabalhos possuem forte acoplamento com o algoritmo minerador, o que impede ou dificulta alterações neste sentido.

Trabalho	Tipo BD	Algoritmo	Acoplamento	Multidimensional	Multinível	Características específicas
<b>Adaptive-FP</b>	Transacional	FP-Growth	Forte	Sim	Não realiza processamento multinível	Utiliza suportes flexíveis
<b>ML-T2L1</b>	Transacional	Apriori	Forte	Não	Realiza processamento multinível Regras geradas apenas no nível mais baixo	Indicação dos atributos a minerar Minera apenas os descendentes de ancestrais fortes
<b>EMARC</b>	Cubo	Apriori	Forte	Inter-Dimensional	Não	Meta-regras: indicação do antecedente e conseqüente das regras Suporte e confiança redefinidos utilizando a métrica SUM
<b>MGAR</b>	Transacional	Apriori	Forte	Não	Sim	Definição de regras interessantes Inserção da transação estendida no algoritmo minerador
<b>AOBDAMFLCI</b>	Cubo	Próprio	Forte	Não informa	Não	Minera outliers através de uma função de pontuação
<b>DARODCDDFM</b>	Cubo	Apriori + MDX	Forte	Sim	Não	Utiliza consultas MDX
<b>DOLAM</b>	Cubo	APriori FP-Growth	Fraco	Sim	Sim	Arquitetura desacoplada <i>Outlier mining</i> Exploração de cubos

Tabela 20- Comparação entre os trabalhos

## 6. Conclusão e trabalhos futuros

Este capítulo apresenta as considerações finais deste trabalho, bem como os objetivos que foram transformados em contribuições e algumas indicações para trabalhos futuros.

### 6.1. Considerações Finais

Esta dissertação apresentou arquitetura DOLAM (*Decoupled OLAM*) para mineração desacoplada de regras de associação multidimensional, multinível e de outliers em cubos OLAP. A arquitetura DOLAM deve ser inserida no processo de KDD entre as etapas de Pré-Processamento e Transformação de Dados. O estudo de caso dá indícios de que os objetivos propostos foram alcançados, pois ao se fazer uso da arquitetura DOLAM, uma quantidade maior de regras é gerada, e estas possuem suporte e confiança mais fortes, uma vez que a abstração multidimensional e multinível das células OLAP foi considerada. As regras geradas a partir do uso da arquitetura DOLAM são da forma multidimensional e multinível, conforme proposto inicialmente. O estudo de caso exemplifica o desacoplamento da arquitetura DOLAM, ao utilizar dois algoritmos diferentes para minerar regras de associação, a saber: *APriori* e *FP-Growth*. A funcionalidade de detecção de *outliers* foi implementada, possibilitando *outlier mining* multidimensional e multinível. O processamento realizado pela arquitetura DOLAM mostrou-se viável quanto ao tempo de execução e útil quanto à quantidade das regras de associação e ao desacoplamento de seus componentes.

### 6.2. Contribuições

As principais contribuições oriundas deste trabalho são:

- Definição da arquitetura DOLAM de forma desacoplada do algoritmo minerador;
- Processamento provido pela arquitetura DOLAM possibilita gerar regras de associação multidimensional e multinível, em quantidade superior à mineração tradicional;
- Capacidade da arquitetura DOLAM de observar as abstrações existentes nos cubos OLAP, através do Componente Expansor de Ancestrais;

- Integração da arquitetura DOLAM com um servidor OLAP. No estudo de caso, foi utilizado o servidor OLAP *Mondrian*;
- Capacidade da arquitetura DOLAM de detectar *outliers* presentes em um cubo OLAP, através do Componente Detector de *Outliers*;
- Capacidade de expandir escopo de busca a partir de um subcubo inicial definido pelo usuário (SCI), provendo flexibilidade para explorar as abstrações presentes nos cubos OLAP, implementada através do Componente Explorador de Subcubos;
- Utilização de tecnologias e ferramentas consolidadas e não-proprietárias, como o servidor OLAP *Mondrian*, a linguagem Java, a API OLAP4J e o pacote de software Weka.

### **6.3.      *Trabalhos Futuros***

Como continuação deste trabalho, pretende-se criar um mecanismo de prevenção de regras redundantes através da seleção das células expandidas geradas pela arquitetura DOLAM.

Pretende-se também adaptar a arquitetura DOLAM para que ela trabalhe com outras tarefas de mineração de dados, como agrupamento (*clustering*). Pretende-se também utilizar a métrica *sum* para cálculo do suporte e confiança, conforme utilizado por EMARC [26].

A arquitetura DOLAM realiza processamento para mineração de regras de associação entre membros de células de cubos de dados OLAP. Como uma célula é formada por um membro de cada dimensão, não é possível gerar regras de associação da forma intra-dimensional e intra-nível. Para fazê-lo, pretende-se, no futuro, minerar associações entre conjuntos de células, para que seja possível obter regras que associam membros do mesmo nível de uma dimensão, como Janeiro e Fevereiro, por exemplo.

# Bibliografia

- [1] **Agrawal, R. and Srikant, R..** *Fast Algorithms for Mining Association Rules in Large Databases*. s.l. : Proceedings of 20th International Conference on Very Large Data Bases, Morgan Kaufmann, pp. 487-499, 1994.
- [2] **Agrawal, R., Imielinski, T. and Swami, A.** *Mining Association Rules between Sets of Items in Large Databases*. Proceedings of the 1993 ACM SIGMOD international conference on Management of data, pp. 207-216, 1993.
- [3] **Bigues, J. P.** *Data mining with Neural Networks*. s.l. : McGrawHill, 1996.
- [4] **Chaudhuri, S.t and Dayal, U..** *An overview of data warehousing and OLAP technology*. s.l. : ACM, 1997, SIGMOD Rec., Vol. 26, pp. 65-74.
- [5] **Bogdanova, G.and Georgieva, T.** *Discovering the Association Rules in OLAP Data Cube with Daily Downloads of Folklore Materials*. International Conference on Computer Systems and Technologies- CompSysTech ' pp. IIIB.23-1–IIIB.23-6. 2005.
- [6] **Elmasri, R. and Navathe, S. B.** *Fundamentals of Database Systems*. s.l. : Pearson Addison Wesley, 2003.
- [7] **Fayyad, S. P.** *The KDD Process for Extracting Useful Knowledge from Volumes of Data*. 1996. Vol. 29, pp. 27-34.
- [8] **Fayyad, U. M., Piatetsky-Shapiro, G.y and Smyth, P..** *From Data Mining to Knowledge Discovery: An Overview*. *Advances in Knowledge Discovery and Data Mining*. s.l. : AAAI Press, Menlo Park, CA, 1996, pp. 1-34.
- [9] **Fidalgo, R. N.** *Uma Infra-estrutura para Integração de Modelos, Esquemas e Serviços Multidimensionais e Geográficos*. Tese de Doutorado. Universidade Federal de Pernambuco. 2005.
- [10] **Georgieva, T.** *Discovering the Association Rules in OLAP Data Cube with Daily Downloads of Folklore Materials*. International Conference on Computer Systems and Technologies- CompSysTech ' pp. IIIB.23-1–IIIB.23-6. 2005.
- [11] **Georgieva, T.** *Using the Fractal Dimension of Sets to Discover the Distribution Intervals of Association Rules in OLAP Data Cubes*. *Proceedings of the First International Conference on Information Systems and DataGrids*, pp. 88-98. 2005.
- [12] **Grubbs, F. E.** *Procedures for Detecting Outlying Observations in Samples*. *Technometrics*, vol. 11, no. 1, pp. 1—21. 1969
- [13] **Han, J. and Fu, Y..** *Discovery of Multiple-Level Association Rules from Large Databases*. Proceedings of the 1995 Int. Conf. Very Large Data Bases Conference, pp. 420-431. 1995.

- [14] **Han, J. and Kamber, M.** *Data Mining: Concepts and Techniques (The Morgan Kaufmann Series in Data Management Systems)*. 1st. s.l. : Morgan Kaufmann, 2000.
- [15] **Han, J., Cai, Y. and Cercone, N.** *Data-Driven Discovery of Quantitative Rules in Relational Databases* IEEE Transactions on Knowledge and Data Engineering, Vol. 5, pp. 29-40. 1993.
- [16] **Han, J., Pei, J. and Yin, Y.** *Mining frequent patterns without candidate generation*. s.l. : ACM Press, 2000. pp. 1-12.
- [17] **Han, J., Pei, J., Yin, Y. and Mao, R.** *Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach*. Data Mining and Knowledge Discovery. Volume 8 , pp. 53 – 87. 2004
- [18] **Hawkins, D.** *Identifications of Outliers*. s.l. : Chapman and Hall, 1980.
- [19] **Hong, Z. and Zhang, B.** *Generalized Association Rule Mining Algorithms based on Data Cube*. s.l. : IEEE Computer Society, 2007, Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, ACIS International Conference on, Vol. 2, pp. 803-808.
- [20] **Inmon, W. H.** *Como construir um Data Warehouse*. s.l. : Editora Campus, 1997.
- [21] **Kimball, R.** *The Data Warehouse Toolkit*. s.l. : John Wiley & Sons, Inc, 1996.
- [22] **Kimball, R. and Ross, M.** *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling*. s.l. : John Wiley & Sons, Inc., 2002.
- [23] **Lin, S. and Brown, D. E.** *An outlier-based data association method for linking criminal incidents*. 2006, Decision Support Systems, Vol. 41, pp. 604-615.
- [24] **Mao, R.** *Adaptive-FP: An Efficient And Effective Method For Multi-Level Multi-Dimensional Frequent Pattern Mining*. Dissertação de Mestrado. Simon Frasier University. 2005.
- [25] **MDX Reference**. Disponível em: <http://msdn.microsoft.com/en-us/library/ms145506.aspx>. Acessado em 12 de fevereiro de 2010
- [26] **Messaoud, R., Rabaséda, S. Boussaid, O. and Missaoui, R.** *Enhanced mining of association rules from data cubes*. Proceedings of the 9th ACM international workshop on Data warehousing and OLAP, pp. 11-18. 2006.
- [27] **Mondrian Pentaho Reference**. Disponível em: <http://mondrian.pentaho.org/>. Acessado em: 12 de fevereiro de 2010
- [28] **OLAP4J Reference**. <http://www.olap4j.org/>. [Online] [Cited: ]
- [29] **Passos, R.** *Data Mining: Um guia prático*. s.l. : Elsevier, 2005.
- [30] **Pei, J., Han, J., Mortazavi-Asl, B., Wang, J., Pinto, H., Chen, Q., Dayal, U. and Hsu, M.** *Mining Sequential Patterns by Pattern-Growth: The PrefixSpan Approach*. s.l. : IEEE

Computer Society, 2004, IEEE Transactions on Knowledge and Data Engineering, Vol. 16, pp. 1424-1440.

- [31]**Silva, F. R..** *Uma metodologia para análise de requisitos em sistemas Data Warehouse*. Dissertação de Mestrado. Universidade Federal de Pernambuco. 2003.
- [32]**Srikant, R. and Agrawal, R.** *Mining generalized association rules*. Research Report RJ 9963, IBM Almaden Research. 1995.
- [33]**Thomsen, E.** *OLAP Construindo Sistemas de Informações Multidimensionais*. s.l. : Editora Campus, 2002.
- [34]—. *OLAP Solutions – Building Multidimensional Information Systems*. s.l. : John Wiley & Sons, Inc, 1997.
- [35]**Turban, E.** *Decision Support and Expert Systems: Management Support Systems*. s.l. : Prentice Hall PTR, 1993.
- [36]**Ross, W. M.** *The Data warehouse Lifecycle Toolkit*. s.l. : John Wiley & Sons, Inc., 1998.
- [37]**Romano, D.** *Data Mining Leading Edge: Insurance & Banking*. s.l. : Proceedings of Knowledge Discovery and Data Mining, Unicom, Brunel University, 1997.



# Apêndice I

## Esquema do cubo de dados utilizado no Estudo de Caso (Capítulo 4)

```
<Schema name="ANDE">

  <Cube name="Liberacoes" defaultMeasure="Valor contrato" cache="true" enabled="true">

    <Table name="fact_liberacao">

    </Table>

    <Dimension type="StandardDimension" foreignKey="id_filial" name="Filial">

      <Hierarchy hasAll="true" allMemberName="All Filials" primaryKey="id_filial">

        <Table name="dim_filial">

        </Table>

        <Level name="Regiao" column="cod_filial" nameColumn="desc_filial" type="String"
uniqueMembers="false" levelType="Regular" hideMemberIf="Never">

        </Level>

        <Level name="Cidade" column="cod_posto" nameColumn="desc_posto" type="String"
uniqueMembers="false" levelType="Regular" hideMemberIf="Never">

        </Level>

      </Hierarchy>

    </Dimension>

    <Dimension type="StandardDimension" foreignKey="id_cliente" name="Cliente">

      <Hierarchy hasAll="true" allMemberName="All Clientes" primaryKey="id_cliente">

        <Table name="dim_cliente">

        </Table>

        <Level name="Genero" column="genero" nameColumn="genero" type="String"
uniqueMembers="false" levelType="Regular" hideMemberIf="Never">

        </Level>

        <Level name="Tipo Imovel" column="tipo_imovel" nameColumn="tipo_imovel"
type="String" uniqueMembers="false" levelType="Regular" hideMemberIf="Never">

        </Level>
```

```

        <Level name="Escolaridade" column="escolaridade"
nameColumn="escolaridade" type="String" uniqueMembers="false" levelType="Regular"
hideMemberIf="Never">

        </Level>

        <Level name="Estado Civil" column="estado_civil" nameColumn="estado_civil"
type="String" uniqueMembers="false" levelType="Regular" hideMemberIf="Never">

        </Level>

    </Hierarchy>

</Dimension>

<Dimension type="StandardDimension" foreignKey="id_negocio" name="Negocio">

    <Hierarchy hasAll="true" allMemberName="All Negocios" primaryKey="id_negocio">

        <Table name="dim_negocio">

        </Table>

        <Level name="Ramo Atividade" column="ramo_atividade"
nameColumn="ramo_atividade" type="String" uniqueMembers="false" levelType="Regular"
hideMemberIf="Never">        </Level>

        <Level name="Categoria Negocio" column="categoria_negocio"
nameColumn="categoria_negocio" type="String" uniqueMembers="false" levelType="Regular"
hideMemberIf="Never">        </Level>

        <Level name="Sazonalidade Negocio" column="sazonalidade_negocio"
nameColumn="sazonalidade_negocio" type="String" uniqueMembers="false"
levelType="Regular" hideMemberIf="Never"> </Level>

        <Level name="Tipo Negocio" column="tipo_negocio" nameColumn="tipo_negocio"
type="String" uniqueMembers="false" levelType="Regular" hideMemberIf="Never">

        </Level>

    </Hierarchy>

</Dimension>

<Dimension type="StandardDimension" foreignKey="id_tempo" name="Tempo">

    <Hierarchy hasAll="true" allMemberName="All Tempos" primaryKey="id_tempo">

        <Table name="dim_tempo">

        </Table>

```

```

    <Level name="Ano" column="ano" type="Integer" uniqueMembers="false"
levelType="Regular" hideMemberIf="Never">

    </Level>

    <Level name="Trimestre" column="trimestre" type="String" uniqueMembers="false"
levelType="Regular" hideMemberIf="Never">

    </Level>

    <Level name="Mes" column="mes_numero" nameColumn="mes_nome" type="String"
uniqueMembers="false" levelType="Regular" hideMemberIf="Never">

    </Level>

    <Level name="Dia" column="dia_numero" type="Integer" uniqueMembers="false"
levelType="Regular" hideMemberIf="Never">

    </Level>

</Hierarchy>

</Dimension>

<Dimension type="StandardDimension" foreignKey="id_produto" name="Produto">

    <Hierarchy name="Produtos" hasAll="true" allMemberName="Todos os produtos"
primaryKey="id_produto">

        <Table name="dim_produto">

        </Table>

        <Level name="Produto" column="cod_produto" nameColumn="desc_produto"
type="String" uniqueMembers="false" levelType="Regular" hideMemberIf="Never">

            <Property name="Codigo Produto" column="cod_produto" type="String">

            </Property>

        </Level>

    </Hierarchy>

</Dimension>

<Dimension type="StandardDimension" foreignKey="id_negocio" name="Tipo negocio">

    <Hierarchy hasAll="true" allMemberName="All Tipo negocios" primaryKey="id_negocio">

        <Table name="dim_negocio">

        </Table>

```

```

    <Level name="Mobilidade" column="tipo_negocio" type="String" uniqueMembers="false"
levelType="Regular" hideMemberIf="Never">

    </Level>

</Hierarchy>

</Dimension>

<Dimension type="StandardDimension" foreignKey="id_gar1" name="Garantia">

    <Hierarchy name="Garantias" hasAll="true" allMemberName="Todas as garantias"
primaryKey="id_garantia">

        <Table name="dim_garantia">

        </Table>

        <Level name="Tipo garantia" column="tipo_garantia" type="String"
uniqueMembers="false" levelType="Regular" hideMemberIf="Never">

        </Level>

        <Level name="Garantia" column="cod_garantia" nameColumn="desc_garantia"
type="String" uniqueMembers="false" levelType="Regular" hideMemberIf="Never">

            <Property name="Cod.Garantia" column="cod_garantia" type="String">

            </Property>

        </Level>

    </Hierarchy>

</Dimension>

<Dimension type="StandardDimension" foreignKey="id_fundo" name="Fundo origem">

    <Hierarchy name="Fundos origem" hasAll="true" allMemberName="Todos os fundos
origem" primaryKey="id_fundo">

        <Table name="dim_fundo">

        </Table>

        <Level name="Fundo origem" column="cod_fundo" nameColumn="desc_fundo"
type="String" uniqueMembers="false" levelType="Regular" hideMemberIf="Never">

        </Level>

    </Hierarchy>

</Dimension>

```

```
<Measure name="Valor contrato" column="valor_contrato" datatype="Numeric"
formatString="#.###.00" aggregator="sum" visible="true">

</Measure>

<Measure name="Prazo" column="quantidade_parcelas" datatype="Numeric"
aggregator="avg" visible="true">

</Measure>


<Measure name="Contratos" column="contrato" datatype="Integer" aggregator="count"
visible="true">

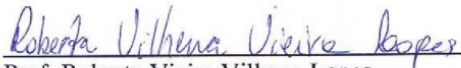
</Measure>

</Cube>

</Schema>
```


Dissertação de Mestrado apresentada por **Carla Moreira Tanure** à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título **“Uma Arquitetura de Software para Descoberta de Regras de Associação Multidimensional, Multinível e de Outliers em Cubos OLAP: Um Estudo de Caso com os Algoritmos APriori e FP-Growth”**, orientada pelo **Prof. Robson do Nascimento Fidalgo** e aprovada pela Banca Examinadora formada pelos professores:

  
\_\_\_\_\_  
Prof. George Darmiton da Cunha Cavalcanti  
Centro de Informática / UFPE

  
\_\_\_\_\_  
Prof. Roberta Vieira Vilhena Lopes  
Instituto de Computação / UFAL

  
\_\_\_\_\_  
Prof. Robson do Nascimento Fidalgo  
Centro de Informática / UFPE

Visto e permitida a impressão.  
Recife, 01 de março de 2010.

  
\_\_\_\_\_  
**Prof. Nelson Souto Rosa**  
Coordenador da Pós-Graduação em Ciência da Computação do  
Centro de Informática da Universidade Federal de Pernambuco.