UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE CIÊNCIAS E TECNOLOGIA
COORDENAÇÃO DE PÓS-GRADUAÇÃO EM INFORMÁTICA

DISSERTAÇÃO DE MESTRADO

INTEGRAÇÃO MINERAÇÃO DE DADOS – SGBD NÃO É UMA PANACÉIA: ESTUDO DA INTEGRAÇÃO DO ALGORITMO *APRIORI* QUANTITATIVO AO *ORACLE9I*

MARIA DE FÁTIMA ALMEIDA SANTOS

Campina Grande – PB
Agosto de 2002

UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE CIÊNCIAS E TECNOLOGIA
COORDENAÇÃO DE PÓS-GRADUAÇÃO EM INFORMÁTICA

INTEGRAÇÃO MINERAÇÃO DE DADOS – SGBD NÃO É UMA PANACÉIA: ESTUDO DA INTEGRAÇÃO DO ALGORITMO *APRIORI* QUANTITATIVO AO *ORACLE9I*

MARIA DE FÁTIMA ALMEIDA SANTOS

Dissertação apresentada à Coordenação de Pós-graduação em Informática – COPIN – da Universidade Federal de Campina Grande – UFCG, como requisito parcial para a obtenção do grau de Mestre em Informática.

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Sistemas de Informação e Banco de Dados

Orientador: Marcus Costa Sampaio

Campina Grande – PB
Agosto de 2002

"A chave para ter sucesso nos negócios
é ter informação que ninguém mais tem."
(Aristóteles Onassis)

Agradecimentos

A Deus, pelas graças que me concede a cada dia.

A meus pais Manoel e Valdeci, pelo incentivo para seguir este caminho.

A meus irmãos Fábio, Fabiano e Flaviano, pelo estímulo e paciência nas horas mais difíceis. Em especial a ‘Flavi’, conselheiro de todas as horas e companhia constante para uma boa conversa nos intervalos de estudo.

A meu orientador Prof. Doutor Marcus Costa Sampaio, pela seriedade com que conduziu este trabalho e pelo profundo conhecimento na área, que resultaram em uma excelente orientação.

Ao ex-professor, hoje colega de profissão e amigo, Methanias Júnior, por ter me apresentado os primeiros conceitos da fascinante área de banco de dados e por fazer questão de lembrar-me sempre que seria possível chegar até aqui. Não poderia deixar de agradecer ainda pela companhia nas várias noites de estudo.

Ao amigo Mateus que, mesmo distante, auxiliou e incentivou este trabalho.

Aos amigos Cristiane d’Ávila, Cristiane Nunes, Lucianne e Yuri, pelo apoio constante e pelo encorajamento, principalmente na tão difícil reta final.

Aos professores, colegas e funcionários do Programa de Mestrado que, com certeza, contribuíram de alguma forma.

Dedicatória

Dedico este trabalho a meus queridos pais, verdadeiras fontes de amor, carinho e admiração e meus maiores incentivadores.

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

Inicialmente os algoritmos de Mineração de Dados eram utilizados em arquivos de dados especialmente preparados com esta finalidade, sem a gerência de SGBD's. Entretanto, pesquisas recentes começaram a aparecer com o objetivo de integrar esses dois mundos, visando aliar o potencial das técnicas de mineração às conhecidas vantagens dos SGBD's como, por exemplo, controle e atomicidade de transações, segurança, robustez e otimização de consultas.

Este trabalho é principalmente sobre a implementação de um algoritmo de mineração de dados considerado estado-da-arte em modelos de regras de associação, denominado *Apriori* Quantitativo [4], estreitamente integrado com o SGBDOR *Oracle9i*. O algoritmo praticamente inexiste além da literatura especializada, isto é, nunca tinha sido testado e usado em outros trabalhos. Os resultados obtidos foram então comparados com o clássico algoritmo de regras de associação *Apriori* [2], também integrado ao *Oracle9i*, visando avaliar em que medida o primeiro seria superior ao último. Os resultados, de uma certa forma desconcertantes, permite-nos concluir que integrar algoritmos de mineração de dados a SGBD's não é uma panacéia. Os problemas que tivemos com a integração do *Apriori* Quantitativo ao *Oracle9i* são discutidos em detalhes, as conclusões gerais da experiência sendo inferidas.

As bases de dados utilizadas pelos algoritmos foram modeladas de duas formas distintas: a primeira, puramente relacional; a segunda, objeto-relacional, fazendo uso do conceito de tipo coleção de dados. Desta forma, um outro objetivo desta dissertação foi verificar a influência das estruturas de dados relacional e objeto-relacional sobre o desempenho dos algoritmos, à procura de conclusões gerais e úteis sobre a superioridade de uma sobre a outra.

Abstract

Most early data mining algorithms were developed largely on specialized file systems, without the DBMS management. However, modern researches begun appeared fitting to couple the database systems and data mining techniques and summing the mining approaches up DBMS's known advantages as transactions control and atomicity, security, robustness and query optimization.

This work analyzes the Quantitative Apriori [4] algorithm, state of the art in association rules generation completely integrated to Oracle 9i Object-Relational DBMS. The algorithm almost never there is over the specialized literature, so it wasn't be tested and used. The results (execution time) of it were compared to Apriori [2] classical association rules generation algorithm, either coupled to Oracle 9i,looking at analyses how first one would be higher than the last one. The results, although disturbing, let us conclude that couple the data mining algorithms to DBMS's isn't an enough solution. The problems that we had with the Apriori Quantitative integration to Oracle9i are discussed in details and the general tests conclusions are being deduced.

The databases used for algorithms were modeled by two distinct forms: first, purely relational; second, with object-relational extensions, wearing, for example, collection types. Then, an other work purpose was analyze the data structures relational and object-relational influence about the algorithms performance, looking for general and useful conclusions about superiority of one then other.

Capítulo 1

Introdução

A evolução do hardware, propiciando meios de armazenamento e recuperação de informação cada vez maiores a um custo mais baixo e o crescimento avassalador da Internet, cuja ubiqüidade permite que as empresas guardem dados sobre clientes e transações, entre outros fatores, têm feito com que bases de dados gigantescas surjam e cresçam a cada dia. Aliada a este fato, a competição entre as empresas torna-se cada vez mais acirrada, o que exige uma constante melhoria na qualidade dos produtos e serviços oferecidos. Todo este cenário faz surgir a necessidade de novas estratégias de negócio. Os tomadores de decisão modernos precisam de novas ferramentas para enfrentar essas profundas mudanças.

Por outro lado, existe uma enorme quantidade de informação guardada nos bancos de dados, informação esta que é potencialmente importante, mas que ainda não foi descoberta, ou seja, está escondida na massa de dados e é raramente tornada explícita.

Surgem, então, técnicas para descobrir informação, as quais consistem em descobrir *padrões* ("*patterns*") ou *conhecimento* em dados. Um conhecimento descoberto pode ser utilizado como uma previsão a respeito de dados futuros, que continuariam seguindo o mesmo padrão, dentro de uma margem de erro que pode ser quantificada. É importante, no entanto, que os padrões descobertos não sejam nem óbvios e nem desinteressantes. Em outras palavras, o conhecimento deve ser útil ao processo de tomada de decisão [9].

1.1 Um Rápido Panorama de Mineração de Dados, com Ênfase em Regras de Associação

O processo de descobrir padrões em dados é conhecido como Mineração de Dados ("*Data Mining*", em inglês). O processo deve ser semi-automático, isto porque é indispensável a interação com o usuário, que participará do processo desde a definição dos dados a serem analisados, até a análise do conhecimento gerado, de maneira a verificar se este é realmente útil e previamente desconhecido. Mais precisamente, o processo semi-automático

de mineração de dados visa extrair, de grandes bases de dados, sem nenhuma formulação prévia de hipóteses, informações desconhecidas, válidas e acionáveis, úteis para a tomada de decisão [1].

Existem vários modelos de padrão ou de conhecimento. Dois modelos bem disseminados são *Classificação em Forma de Árvore de Decisão* ([20], [27]) e *Regras de Associação* ([1], [2]).

Um algoritmo de classificação pode gerar uma árvore de decisão como parcialmente apresentada na Fig. 1 [20]. A interpretação é que uma pessoa que possui mestrado e ganha acima de dez mil reais por ano paga seus empréstimos, sendo, portanto, uma confiável. O atributo renda foi dividido em faixas. Já uma pessoa com o título de mestrado e com uma renda anual menor que 10 mil reais não é uma pessoa confiável para se conceder um empréstimo. Tal conhecimento extraído da massa de dados de uma empresa permite ao gerente tomar a decisão de fazer novos empréstimos com uma maior segurança.

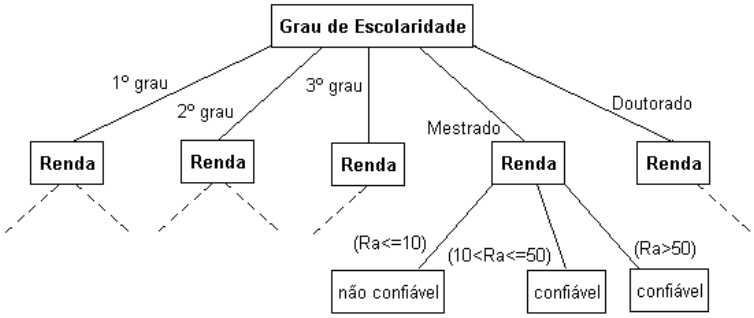


Figura 1 – Árvore de decisão

Os ramos da árvore podem crescer de maneira diferente. Isto pode ocorrer porque, por exemplo, para um determinado grau de escolaridade como o 1º grau, é possível que não existam empréstimos realizados a clientes com tal atributo. Além disso, todas as regras geradas a partir de uma árvore terão que conter o atributo raiz em seu antecedente. No exemplo, como o grau de escolaridade é o atributo raiz escolhido, não há como se ter uma regra do tipo: *Se Ra > 50 então confiável*.

Existem outros algoritmos de classificação que, ao invés de montarem uma árvore de decisão, expressam o conhecimento extraído através de regras do tipo *Se condição então classe* ou $X, Y \Rightarrow Z$. Regras nestas formas são chamadas de *regras de classificação stricto sensu*, ou simplesmente *regras de classificação*. Na Fig 2, são exemplificadas as mesmas regras da Fig. 1, sob a forma de regras de classificação[20]:

... Se (Grau de Escolaridade = Mestrado), (Ra <= 10) $\Rightarrow$ não confiável

Se (Grau de Escolaridade = Mestrado), (10 < Ra <= 50) $\Rightarrow$ confiável

Se (Grau de Escolaridade = Mestrado), (Ra > 50) $\Rightarrow$ confiável ...

Figura 2 – Regras de classificação

Uma regra de classificação terá sempre no seu conseqüente uma resposta ao fato das condições satisfazerem ou não a uma determinada classe previamente definida.

Regras de Associação consistem em exprimir correlações entre dados. Um exemplo clássico é o da rede de supermercados americana que descobriu que o aumento do consumo de fraldas infantis também causava o aumento do consumo de cerveja. Esta informação não é de modo algum trivial e é inviável de ser descoberta a partir de uma inspeção manual ao banco de dados, ou mesmo com uma consulta SQL.

Uma regra de associação é definida como: Se X então Y ou X $\rightarrow$ Y, onde X e Y são conjuntos de termos e $X \cap Y = \varnothing$. Diz-se que X é o antecedente da regra, enquanto Y é o seu conseqüente ([1], [2], [3], [14]). Daqui por diante a palavra item poderá ser usada com o mesmo sentido de termo. Uma regra pode ter vários itens tanto no antecedente quanto no conseqüente. Um algoritmo baseado em regras de associação consiste em descobrir regras desse tipo entre os dados preparados para a mineração.

Uma regra de associação é então uma regra de classificação generalizada. A generalização consiste no fato de que Y na regra de associação é uma conjunção de termos com quaisquer atributos, enquanto que, nas regras de classificação, este é só um termo envolvendo unicamente o atributo de classificação. A expressividade de uma regra de associação gerada é medida pela correlação entre os seus antecedente e conseqüente: quanto mais forte a correlação, melhor ou mais expressiva (confiável) é a regra.

É importante salientar que um algoritmo de regras de associação pode gerar uma explosão de regras, uma vez que muitas combinações de itens são possíveis. Para evitá-la, dois parâmetros são passados para o algoritmo: suporte mínimo e confiança mínima.

Seja a tabela 1, que representa dados de uma mercearia para mineração. A primeira coluna representa o código da transação e as demais, os itens ou produtos de venda. Cada linha representa uma transação feita para um cliente, e é identificada a partir da primeira coluna: o identificador de transação. Se um cliente comprou um item, o valor da coluna

correspondendo ao item é 1, caso contrário, 0. Por exemplo, a primeira transação indica que um cliente comprou leite e manteiga, mas não comprou pão.

Transação	Leite	Manteiga	Pão
1	1	1	0
2	1	0	1
3	1	1	1
4	1	1	1
5	0	1	1
6	1	1	1
7	1	1	1
8	1	0	1
9	1	1	1
10	1	1	1

Tabela 1 – Transações de vendas de uma mercearia

Considere a regra:

{pão, leite} ➔ {manteiga} /* Se pão e leite então manteiga */

Pode-se constatar que os itens Pão, Leite e Manteiga foram comprados juntos em 6 das 10 transações da tabela 1. Diz-se que a regra tem suporte ("*coverage*") de 0,60 (6/10). Por outro lado, o antecedente aparece em 8 transações; das 8, 6 contêm o conseqüente; diz-se então que 0,75 (6/8) é a confiança da regra. Sintetizando, uma regra é expressiva se: (1) for freqüente, ou tiver um suporte acima de um mínimo considerado aceitável; e (2) se a correlação entre antecedente e conseqüente, medida pela confiança, for forte ou acima de um mínimo considerado aceitável. A representação completa da regra é então:

{pão, leite} ➔ {manteiga} [0,60 0,75]

que é lida assim: *60% dos clientes compram pão, leite e manteiga; 75% dos clientes que compram pão e leite também compram manteiga.*

As definições de suporte e confiança são mostradas nas figuras 3 e 4, respectivamente.

Formalmente, para uma base de dados *T*, o suporte de um conjunto de itens *X* é o percentual de transações que verifica *X*, conforme figura 3:

$$\text{Suporte}(X) = \frac{\text{Nº de transações em } T \text{ que verifica } X}{\text{Nº total de transações em } T}$$

Figura 3 – Definição do Suporte de uma Regra

A definição formal de confiança é trazida na figura 4, onde X denota o antecedente e Y o conseqüente da regra:

$$\text{Confiança}(R) = \frac{\text{Nº de transações em } T \text{ que verifica } X \text{ e } Y}{\text{Nº de transações em } T \text{ que verifica } X}$$

Figura 4 – Definição da Confiança de uma Regra

Utilizando o exemplo da mercearia, algumas perguntas podem ser respondidas a partir dessas regras como: Quais regras possuem pão como antecedente? Isso pode ajudar a entender, por exemplo, que itens poderão ter suas vendas influenciadas caso o estabelecimento deixe de vender pão. Outra pergunta pode ser: Quais regras possuem manteiga como conseqüente? Essa informação é uma forma de descobrir como aumentar as vendas deste produto através de sua comercialização associada a outros produtos. Para se ter uma idéia, técnicas de mineração de dados, ainda no contexto da mercearia, podem levar a conclusões como qual a melhor organização das prateleiras de maneira que produtos que geralmente são comprados juntos sejam disponibilizados também juntos para os clientes, estimulando um possível aumento nas vendas.

Muitos algoritmos foram desenvolvidos com o objetivo de descobrir regras de associação entre itens. Podem ser citados: *LargeKItemSets* [1], *Apriori* [2], *AprioriTid* [2]. Desses, o mais disseminado é o seminal *Apriori*, pois deu origem a vários outros. Sendo pioneiro, o *Apriori* padece de vários problemas, sobretudo ligados às questões de desempenho — sua idéia básica, entretanto, permanece inalterável. Levando isso em conta, um grande esforço de pesquisa ([2],[4], [12], [26]) está concentrado na necessidade de otimizar mais e

mais o algoritmo *Apriori*, com a finalidade de viabilizá-lo para a mineração de grandes bases de dados, com muitos itens. O algoritmo *Apriori* Quantitativo [4] aparece como uma proposta muito recente de melhoria do desempenho do *Apriori* clássico. Na versão quantitativa, passa a ser importante saber a quantidade adquirida de cada item, informação que até então não era usada. É relevante saber agora que um cliente comprou 2 litros de leite e 10 pães e não apenas leite e pão. Essa informação extra é então utilizada em proveito do desempenho do algoritmo.

Como se pode perceber, diferentemente das tradicionais consultas a bancos de dados nas quais o usuário expressa exatamente tudo o que quer consultar e como os resultados devem ser apresentados (*consulta fechada*), um algoritmo de mineração de dados é capaz de descobrir informações 'escondidas' na massa de dados (*consulta aberta*).

Exemplificando, para descobrir quantas transações tiveram a venda dos produtos fralda e cerveja juntos, a consulta SQL precisaria ser escrita explicitamente. Com a utilização de um algoritmo de mineração, o máximo que precisa ser feito é especificar o tipo de técnica que deverá ser utilizada. Neste caso, o algoritmo de regras de associação seria executado recebendo os valores de suporte e confiança mínimos. Desta forma, todas as possíveis associações de itens (regras) são geradas automaticamente, sem necessidade de especificação de nenhuma consulta prévia. Cabe ressaltar que o objetivo dos algoritmos de mineração é justamente descobrir informações não previstas. Muito dificilmente um gerente de um supermercado iria imaginar que a venda de fraldas pudesse ter alguma relação com a de cervejas. Assim, é praticamente impossível descobrir as associações entre os itens sem a utilização de um algoritmo de regras de associação.

As pesquisas iniciais em mineração de dados concentraram-se em definir novas técnicas de mineração e desenvolver algoritmos para as mesmas. Esses algoritmos geralmente lêem dados a partir de arquivos isolados não normalizados ("*flat files*").

A teoria convencional de bancos de dados relacionais é apoiada no processo de normalização, visando a eliminação das chamadas anomalias de inclusão, atualização e exclusão, entre outros motivos. Entretanto, a *não normalização* pode ser útil para encontrar associações entre os dados. Para explicar melhor esta questão, seja o esquema completamente normalizado da Fig. 5.

Cod	Item
001	Açúcar
002	Leite
003	Pão

Venda	Data
001	28/10/2001
002	29/10/2001

Venda	Item	Qte
001	002	3
002	001	1
002	003	2
002	002	6

Figura 5 – Base puramente normalizada

Contrapondo-se ao esquema normalizado, segue-se o esquema desnormalizado da figura 6.

Venda	Data	Item	Qtd
001	28/10/2001	Leite	3
002	29/10/2001	Açúcar	1
		Pão	2
		Leite	6

Figura 6 – Base não normalizada – "*flat file*"

A desnormalização pode vir a facilitar muito mais a mineração, visto que todos os itens de uma determinada venda já estão agrupados, dispensando o acesso a várias tabelas. Assim, a questão da normalização deve ser analisada, pois pode influenciar em muito e negativamente no desempenho dos algoritmos de mineração.

Até recentemente, não havia integração entre algoritmos de mineração de dados e Sistemas Gerenciadores de Banco de Dados (SGBD's). Toda a mineração era feita sobre arquivos isolados ou "*stand-alone*", desnormalizados. Entretanto, cada vez mais as empresas armazenam seus dados sob a gerência de robustos SGBD's normalizados. A preparação de dados para mineração (normalizado $\rightarrow$ desnormalizado) pode tornar-se então uma tarefa

complexa e demorada. Atualmente, esforços estão também se concentrando em integrar mineração de dados e SGBD's, visando somar as vantagens desses dois mundos.

Ao mesmo tempo, uma nova geração de SGBD's — os chamados SGBD's extensíveis — começa a surgir, motivada pelas limitações da modelagem relacional, que só suporta restritos tipos de dados, além de presa às amarras da chamada primeira forma normal (tabelas somente com colunas atômicas). Com a evolução do escopo das aplicações, começou a surgir a necessidade de definição de tipos de dados complexos, muitas vezes definidos pelo próprio usuário. Neste cenário, aparecem duas vertentes de novos SGBD's: os puramente orientados a objeto (SGBDOO), e os SGBD´s Objeto-Relacionais (SGBDOR), estes estendendo a tecnologia relacional, sem o intuito de desembaraçar-se dela. Em ambos os casos, tipos de dados não nativos podem ser definidos, o que os faz extensíveis. As estruturas agora não são necessariamente normalizadas na primeira forma normal. A título de ilustração, os bancos objeto-relacionais suportam o conceito de coleção, ou seja, o tipo de uma coluna pode ser um vetor ou um outro tipo coleção. Todo este novo cenário abre novas possíveis perspectivas de integração SGBD – Mineração de Dados.

1.2 Objetivos da Dissertação

O primeiro e principal objetivo deste trabalho foi a análise do desempenho do algoritmo *Apriori* Quantitativo[1] frente ao *Apriori* básico, ambos comparados dentro de um mesmo ambiente de testes. Tornou-se essencial analisar em que medida o *Apriori* Quantitativo que se apresenta como um promissor estado-da-arte em algoritmos de regras de associação no que concerne ao desempenho, o é na prática, isto é, quando implementado e testado. Por exemplo, não está claro se a otimização proposta não acarretaria efeitos colaterais negativos para o desempenho global do algoritmo, e quais seriam esses efeitos negativos.

Para fins de avaliação, ambos os algoritmos foram convertidos para a linguagem *Oracle Object PL/SQL*, que é a linguagem de desenvolvimento de aplicações sobre o SGBDOR *Oracle9i* ([22], [23], [24]). Programas PL/SQL são integrados ao núcleo do servidor *Oracle*, beneficiando-se de todos os recursos de otimização do servidor [18]. Desta

[1] O algoritmo foi o tema de um artigo científico premiado no Simpósio Brasileiro de Banco de Dados 2000.

forma, os tempos de execução de cada algoritmo puderam ser medidos, analisados e comparados.

É preciso atentar para o fato de que o algoritmo *Apriori* Quantitativo utiliza estruturas de dados do tipo "*kd-tree*" [8], que não são nativas do *Oracle*. Fez-se então necessário escrever, em PL/SQL, um programa de criação, manutenção e recuperação de dados de uma "*kd-tree*". Desta forma, novos tipos e operações foram criados para manipular esse tipo de árvore. O segundo objetivo é a integração de mineração de dados com SGBD's, no caso *Apriori* e *Apriori* Quantitativo com o *Oracle9i*. De maneira desconcertante, o desempenho do *Apriori* Quantitativo é bastante inferior ao do *Apriori* básico, por causa das condições especialíssimas do *Apriori* Quantitativo, como o emprego de pesadas estruturas auxiliares do tipo *"kd-trees"*, que tiveram que ser simuladas sob a forma de tabelas. A experiência evidenciou que a integração mineração de dados – SGBD's não é uma panacéia.

O terceiro e último objetivo é a comparação das tecnologias relacional e objeto-relacional utilizando o mesmo ambiente (*Oracle9i*) e os mesmos algoritmos de mineração, fazendo mudanças apenas nas estruturas de dados. Ao contrário do que se poderia esperar, pelo menos no que diz respeito à experiência com o algoritmo *Apriori* Quantitativo, as estruturas relacionais — tabelas normalizadas — se revelaram mais eficientes que as objeto-relacionais — tabelas desnormalizadas. As causas para isto, absolutamente não evidentes, são explicadas em detalhes, no decorrer da dissertação. Conclusões gerais a respeito desta comparação entre as tecnologias relacional e objeto-relacional também são obtidas.

1.3 Relevância da Dissertação

A integração de tecnologias de mineração de dados com SGBD's promete unir as vantagens desses dois mundos. As pesquisas têm se concentrado em regras de associação e bancos de dados relacionais. O algoritmo de regras de associação que tem sido utilizado é o seminal *Apriori*, o qual, infelizmente, apresenta sérias restrições no que diz respeito ao seu desempenho. Por sua vez, os bancos de dados relacionais, sendo normalizados (primeira forma normal), podem dificultar a descoberta de regras de associação.

Essa pesquisa analisou uma integração do algoritmo *Apriori* Quantitativo, pretendida evolução do clássico algoritmo *Apriori*, com o *Oracle* 9i. Tal integração permitiu verificar em que medida a otimização proposta por este algoritmo seria útil na solução de problemas práticos de mineração de dados.

Outro ponto de extrema relevância foi a avaliação da tecnologia objeto-relacional, em comparação com a tecnologia relacional. Isto veio cobrir uma lacuna existente na literatura, que é a falta de uma análise crítica da tecnologia objeto-relacional.

1.4 Estrutura da Dissertação

A dissertação está estruturada como segue.

O capítulo 1 é esta introdução.

O capítulo 2 concentra-se na discussão dos conceitos e utilidade prática das técnicas de mineração de dados, com ênfase especial na geração de regras de associação. Os Algoritmos *Apriori* e *Apriori* Quantitativo são explicados em detalhes.

No terceiro capítulo é abordada a questão da integração de algoritmos de regras de associação com os SGBD's, comentando suas vantagens e desvantagens potenciais. O capítulo menciona ainda o estado-da-arte em pesquisas de integração.

O capítulo 4 descreve o funcionamento e a implementação do *Apriori* Quantitativo no *Oracle*, abordando todos os recursos e detalhes utilizados.

O quinto capítulo descreve os experimentos realizados, detalhando a quantidade de dados utilizada, a variação de parâmetros e medidas tomadas para a otimização das consultas no SGBD *Oracle.* Ainda neste capítulo, os resultados, não esperados, são interpretados e analisados.

O capítulo 6 apresenta as conclusões e as perspectivas do trabalho. Completam o documento a bibliografia e os anexos.

Capítulo 2

Mineração de Dados Através de Regras de Associação

Como exposto no capítulo 1, algoritmos de mineração de dados extraem, de grandes bases de dados e sem prévia formulação de hipóteses, tendências, padrões e correlações entre os dados — *conhecimento* —, os quais devem ser úteis à tomada de decisão.

Existem diversos modelos de conhecimento, e a escolha de um modelo — juntamente com um algoritmo que extrai conhecimento segundo o modelo — depende de um particular problema. Entre os diversos modelos de conhecimento, aqueles que geram *regras de associação* são bastante poderosos e flexíveis, além de provavelmente os mais usados em problemas práticos. Regras de associação dizem diretamente respeito ao nosso trabalho, razão pela qual serão tratadas em detalhes, neste capítulo.

2.1. Regras de Associação

O problema de mineração através de regras de associação foi introduzido em [1]. Esta técnica tem sido intensamente pesquisada e consiste em encontrar correlações entre os dados minerados.

Para isso, um algoritmo muito disseminado é o seminal *Apriori* [2], dando origem a vários outros — família de algoritmos *Apriori*. O crescimento da 'família' é devido ao fato de que persistem problemas, principalmente ligados a questões de desempenho.

2.2. O Algoritmo *Apriori*

O algoritmo *Apriori* foi introduzido por *Agrawal* e *Srikant* em [2]. O problema de descobrir regras de associação pode ser decomposto em dois subproblemas:

1. Encontrar todos os conjuntos de itens (*itemsets*) que têm suporte acima do suporte mínimo (*MinSup*), os quais são chamados de *Large Itemsets* (*grandes conjuntos*);

2. Usar os grandes conjuntos para gerar as regras, com confiança acima da confiança mínima (*MinConf*).

Todo o esforço de otimização é concentrado na geração dos grandes conjuntos, visto que pode existir uma combinação muito grande de itens. De uma maneira geral, a otimização consiste em descartar o quanto antes todos conjuntos candidatos sem suporte mínimo. Após esta etapa, a geração das regras propriamente ditas torna-se uma tarefa extremamente simples.

A tabela 2 contém algumas notações importantes utilizadas pelos diversos algoritmos de regras de associação.

Notação	Descrição
k-itemset	Um conjunto de itens com *k* itens
L_k	Uma união de grandes conjuntos de tamanho *k*. Cada grande conjunto tem dois atributos: i) conjunto de itens ii) suporte
C_k	União de conjuntos candidatos (potencialmente grandes) de tamanho *k*. Cada conjunto candidato tem dois atributos: i) conjunto de itens ii) suporte

Tabela 2 - Algumas notações usadas no *Apriori*

O algoritmo aparece na figura 7. Nas próximas três subseções, explicamos em detalhes seu funcionamento.

```
1)      L1 := {grandes conjuntos de tamanho 1};
2)      para (k = 2; Lk-1 <> 0; k++) faça
3)          Ck = GerarCandidatos(Lk-1);
4)          para toda transação t ∈ D faça
5)              Ct := subconjunto(Ck,t);
6)              para todo candidato c ∈ Ct faça
7)                  c.count++;
8)          fim
9)          Lk = {c ∈ Ck | c.count >= MinSup}
10)     fim;
11)     GerarRegras;
```

Figura 7 – O algoritmo *Apriori*

2.2.1. Geração de Conjuntos Candidatos

A tarefa de descobrir os grandes conjuntos passa por duas etapas antes da contagem do suporte: geração e poda. A primeira gera as possíveis combinações de itens, os chamados *Conjuntos Candidatos* e, na segunda, aqueles que não atingirem o suporte mínimo são descartados. A geração dos candidatos será mostrada nesta seção e a poda logo em seguida.

O algoritmo para descobrir os grandes conjuntos faz muitas passagens pelos dados. Na primeira, o suporte para cada item individual existente é contado e, a partir deste valor, verifica-se quais conjuntos de tamanho 1 são grandes.

Utilizando a base de dados mostrada na tabela 1, o algoritmo geraria os conjuntos candidatos mostrados na figura 8(a). A partir daí, aqueles que obedecerem ao suporte mínimo são considerados grandes. Usando aqui um suporte de 0,7, todos os candidatos seriam considerados grandes. Esta fase está representada na figura 7 através da linha 1.

A partir da segunda passagem pelos dados, cada passo inicia com um conjunto semente de itens o qual irá gerar novos conjuntos potenciais, ou seja, novos candidatos. Assim, após a geração dos grandes conjuntos de tamanho 1, deve-se gerar os de tamanho 2, 3, 4,..n. Para gerar os candidatos de tamanho 2, é preciso produzir todas as combinações possíveis de grandes conjuntos de tamanho 1, encontrados na fase anterior. O processo continua até que nenhum grande conjunto seja encontrado. Na segunda fase serão gerados os candidatos mostrados 8(b), todos considerados grandes. Os grandes conjuntos de tamanho 3 estão sendo mostrados na figura 8(c).

Conjunto	Suporte
{Leite}	0,9
{Manteiga}	0,8
{Pão}	0,9

(a)

Conjunto	Suporte
{Leite, Manteiga}	0,7
{Leite, Pão}	0,8
{Manteiga, Pão}	0,7

(b)

Conjunto	Suporte
{Leite, Manteiga, Pão}	0,6

(c)

Figura 8 – Candidatos de um, dois e três elementos

Essa estratégia de fazer várias passagens pelos dados está representada no algoritmo a partir do laço na linha 2, figura 7. Nela está sendo mostrado que o processo continua até que nenhum grande conjunto seja encontrado na passagem anterior. Dentro do laço é feita a geração dos candidatos e posterior contagem do suporte de cada um deles (linhas 3 a 8). A linha 9 indica que os candidatos de tamanho k que atingirem o suporte passarão a compor os grandes conjuntos deste mesmo tamanho denominados de L_k.

Cabe ressaltar que a ordem dos itens não faz diferença para o algoritmo. Por exemplo, o conjunto {Leite, Pão} é idêntico ao {Pão, Leite}. Justamente por isso que só um candidato de tamanho 3 foi gerado, qual seja, {Leite, Manteiga, Pão}.

A geração dos conjuntos candidatos é feita a partir da função GerarCandidatos, que recebe L_{k-1}, o conjunto de todos os grandes (k-1)-*itemsets* como argumento e retorna um superconjunto de todos os k-*itemsets* candidatos, grandes conjuntos potenciais. A primeira parte desta função é o chamado passo de junção (ou *join step*), no qual é feita uma junção de L_{k-1} com L_{k-1}, como mostrado a seguir:

INSERT INTO C_k
SELECT p.item$_1$, p.item$_2$, ..., p.item$_{k-1}$, q.item$_{k-1}$
FROM L_{k-1} p, L_{k-1} q
WHERE p.item$_1$ = q.item$_1$, ..., p.item$_{k-2}$ = q.item$_{k-2}$, p.item$_{k-1}$ < q.item$_{k-1}$;

O conjunto de candidatos de tamanho k, C_k, recebe o resultado de um *self join* (junção de uma tabela com ela mesma) de L_{k-1}. Seja L_2 = {Leite, Pão},{Leite, Manteiga}, a partir do qual será gerado C_3. Traduzindo a consulta anterior para este caso, fica algo do tipo:

INSERT INTO C_3
SELECT p.item$_1$, p.item$_2$,q.item$_2$
FROM L_2 p, L_2 q
WHERE p.item$_1$ = q.item$_1$,p.item$_2$ < q.item$_2$;

No início do *join*, a consulta tentará montar um candidato a partir de {Leite, Pão} com esse mesmo conjunto. Neste ponto, p.item1 = q.item1 (Leite = Leite). Entretanto, a cláusula p.item2 < q.item2 vai falhar (Pão é igual e não menor que Pão). Tomando agora como exemplo a junção dos conjuntos {Leite, Manteiga} e {Leite, Pão}, neste caso, p.item1 =

q.item1 (Leite = Leite) e p.item2 < q.item2 (Manteiga < Pão), considerando a ordem alfabética). Assim, o conjunto {Leite, Manteiga, Pão} (que corresponde a p.item1, p.item2, q.item2) será inserido em C_3.

Vale ressaltar que o "*where*" na consulta é justamente para garantir duas coisas: primeiro que o conjunto resultante vai ser sempre uma extensão de exatamente um item, de dois conjuntos já considerados grandes anteriormente. Segundo, a última condição garante que o algoritmo nunca irá gerar conjuntos do tipo {Leite, Manteiga, Pão} e {Leite, Pão, Manteiga}, visto que esses dois conjuntos são idênticos. A segunda etapa da função GerarCandidatos é a fase de poda ("*prune step*"), mostrada na próxima seção.

2.2.2. Fase de Poda

Alguns candidatos poderão ser descartados ou, seguindo a terminologia do algoritmo, podados, antes mesmo de terem seu suporte contado. Isso acontece porque qualquer subconjunto de um grande conjunto será necessariamente grande. Ora, para que {Leite, Manteiga} obedeça ao suporte mínimo, os itens leite e manteiga individualmente também devem satisfaze-lo. Assim, se um candidato tem algum subconjunto que não é grande terá sua poda antecipada. O passo de poda então remove todos os *itemsets* $c \in C_k$ que contém algum (k-1)-subconjunto de c que não está em $L_{k-1.}$

Exemplificando, seja L_3 = {{1 2 3}, {1 2 4}, {1 3 4}, {1 3 5}, {2 3 4}}. Depois da junção, C_4 será {{1 2 3 4}, {1 3 4 5}}. O passo de poda irá apagar o conjunto {1 3 4 5} porque o conjunto {1 4 5} não está em L_3. Dessa forma, C_4 conterá apenas {1 2 3 4}. Para cada candidato restante, o suporte será então contado, sendo descartado ou não posteriormente, a depender do suporte mínimo aceitável. A fase de poda é mostrada a seguir:

```
FORALL itemsets c ∈ Ck DO
    FORALL (k-1)-subsets s OF c DO
      IF (s ∉ Lk-1) THEN
            DELETE c FROM Ck;
```

Cabe ressaltar, no entanto, que, para cada conjunto, todos os subconjuntos são verificados, exceto os dois a partir dos quais o conjunto analisado foi gerado que, com certeza, já são grandes. No exemplo anterior, para o candidato {1, 2, 3, 4} gerado a partir de {1, 2, 3} e {1, 2, 4}, somente os subconjuntos {1, 3, 4} e {2, 3, 4} precisam ser conferidos. Isto não está sendo exibido no trecho de código anterior, mas é implementado na prática.

2.2.3. Contagem de Suporte

Esta fase resume-se a passar por todas as transações, verificando a existência ou não do candidato. Caso exista, seu suporte é incrementado (linhas 4 a 7 do algoritmo na figura 7).

2.2.4. Geração de Regras

A última parte do algoritmo *Apriori* é a geração das regras (linha 11, figura 7), partindo dos grandes conjuntos com seus respectivos suportes já encontrados, não exigindo, portanto, passagem pelos dados. Aqui, a confiança mínima aceitável passa a ser considerada.

O processo de geração é visto como:

> Para todo *Large Itemset l*, encontre todos os subconjuntos não vazios de *l*. Para cada subconjunto *a*, gerar a regra no formato $a \rightarrow (l - a)$, se a divisão do suporte(*l*) pelo suporte(*a*) for pelo menos igual à confiança mínima estabelecida (*MinConf*).

Por exemplo, para o grande conjunto {Leite, Manteiga, Pão}, considerando os subconjuntos {Leite, Manteiga} e {Pão} com respectivos suportes de 0,7 e 0,9, e considerando ainda que o suporte de {Leite, Manteiga, Pão} é de 0,6, a confiança da regra seria de 0,6/0,7 = 0,85. Estabelecendo uma confiança mínima aceitável de 0,8, somente as regras sombreadas são geradas.

Regra	Fator de Confiança
{Leite} ➔ {Manteiga}	0,77
{Manteiga} ➔ {Leite}	0,88
{Leite} ➔ {Pão}	0,77
{Pão} ➔ {Leite}	0,88
{Manteiga} ➔ {Pão}	0,88
{Pão} ➔ {Manteiga}	0,77
{Leite, Manteiga} ➔ {Pão}	0,85
{Pão} ➔ {Leite, Manteiga}	0,66
{Leite, Pão} ➔ {Manteiga}	0,75
{Manteiga} ➔ {Leite, Pão}	0,75
{Manteiga, Pão} ➔ {Leite}	0,85
{Leite} ➔ {Manteiga, Pão}	0,66

Tabela 3 – As regras geradas pelo *Apriori*

A técnica de geração de regras de associação é bem flexível e poderosa, na medida em que muitas associações entre os itens podem ser descobertas e através dos parâmetros de suporte e confiança as regras podem ser filtradas. Entretanto, o número de conjuntos candidatos pode ser enorme, podendo até impedir a execução do algoritmo e a explosão de regras. Na prática, o custo do algoritmo depende criticamente da base de dados utilizada e do suporte mínimo especificado. A confiança tem menos influência, pois, para calculá-la, não é mais necessária a varredura das transações. Todo o esforço está praticamente concentrado na necessidade de otimizar mais e mais o algoritmo *Apriori*, com a finalidade de viabilizá-lo para a mineração de grandes bases de dados, com muitos itens. Uma das principais preocupações é tentar descobrir cada vez mais cedo que um candidato pode ser podado, antes mesmo de ter seu suporte contado.

2.3. O Algoritmo *Apriori* Quantitativo

Insistamos: o grande problema do algoritmo básico *Apriori* é que o número de conjuntos candidatos produzidos pode ser muito grande, onerando muito os custos para contar o suporte de todos esses conjuntos. No entanto, muitos desses conjuntos podem não atingir o

suporte mínimo e serão, desta forma, descartados adiante. Sendo assim, grande parte das pesquisas em algoritmos de regras de associação concentra-se em tentar descobrir, cada vez mais cedo, quais conjuntos candidatos podem ser descartados ou podados.

Uma versão bem atual do seminal *Apriori*, *Apriori* Quantitativo [4], é o tema desta seção.

A idéia básica do algoritmo é a de tratar também com a quantidade dos itens. Recordemos que, de acordo com o *Apriori* original, uma transação de um cliente que comprou 2 litros de leite e 10 pães é idêntica à de outro cliente que comprou apenas 1 litro de leite e 3 pães. Com o *Apriori* Quantitativo, as quantidades são informações relevantes e as duas transações são distinguíveis.

A otimização consiste então em descobrir, o mais cedo possível, quais conjuntos candidatos serão descartados, usando as informações quantitativas. Em suma, o algoritmo explora as propriedades quantitativas dos dados, permitindo combinar essa informação adicional como um critério de corte [4]. Seguem-se os detalhes.

2.3.1. Regras Quantitativas

Os domínios dos atributos que descrevem os dados armazenados em um banco de dados podem ser numéricos (quantitativos), como idade e salário, ou categóricos, como valores *booleanos* e códigos postais ([4], [29]). À guisa de simplificação do discurso, todos os atributos serão tratados como numéricos ou quantitativos[2]. Desta forma, associada a um item, sempre existirá uma quantidade caracterizando o que é chamado de regras quantitativas.

Formalmente, seja $A = \{a_1, a_2,..., a_n\}$ o conjunto de atributos de uma tabela, e V o conjunto de inteiros não negativos. V_a é o conjunto de valores inteiros adquiridos para um atributo *a*. Um item i é definido como um par $<a, q_a>$, onde *a* é um atributo e $q_a \in V_a$, seu valor quantitativo. Exemplos de itens: <Leite,2>, <Pão, 5> [4].

Um *itemrange* (intervalo para um item) é uma faixa contígua para um atributo *a*, representado por uma tupla $<a: l_a – h_a>$, onde $l_a \in V_a$, $h_a \in V_a$, e $l_a \leq h_a$ são os limites inferior e superior do intervalo. Por exemplo, <Pão: 2-6> é o *itemrange* para o atributo Pão, que indica que a quantidade comprada de pão variou entre 2 e 6. Em [4], para cada atributo apenas

[2] Para que o algoritmo pudesse tratar de atributos alfanuméricos, estes precisariam ser discretizados e transformados em numéricos.

uma faixa é considerada, ou seja, o *itemrange* conterá as quantidades mínima e máxima de aquisição do produto.

Uma transação *T* é representada por um conjunto $\{t_1, t_2, ..., t_n\}$ de itens, e *D* é o conjunto de todas as transações. Uma transação *T* satisfaz um conjunto de *itemranges I* se para cada $<a_I : l_a – h_a> \in I$ existe um $<a_T, q_a> \in T$ com $a_I = a_T$ e $l_a \leq q_a \leq h_a$ [4].

Uma regra de associação quantitativa é uma expressão da forma X ➔ Y, onde $X \subset I$, $Y \subset I$, $X \cap Y = \varnothing$ e I é um conjunto de *itemranges* [4].

Considerando o estudo de caso do mercearia, uma regra quantitativa é: *70% das pessoas que compram entre 1 e 3 litros de leite, também compram de 5 a 10 pães*.

As medidas de suporte e confiança continuam sendo importantes no *Apriori* Quantitativo. O problema, de uma maneira geral, ainda é encontrar todas as regras que satisfazem ao suporte e confiança mínimos exigidos.

Descobrir regras quantitativas pode ser dividido em três problemas menores: o primeiro passo é enumerar o suporte para os conjuntos de *itemranges*; depois, encontrar todos os conjuntos de itens que obedecem ao suporte mínimo especificado; por fim, as regras são geradas, com confiança igual ou superior ao mínimo exigido. Como se pode perceber, os passos básicos são semelhantes aos do *Apriori* tradicional, mas a informação extra das quantidades adiciona uma nova dimensão nas regras geradas. O algoritmo é mudado basicamente no passo de poda, onde as informações quantitativas são levadas em conta, para que o corte seja mais eficiente.

2.3.2. Estruturas de Dados Utilizadas

O *Apriori* Quantitativo usa dois tipos de estruturas de dados que auxiliam no processo de geração e poda dos conjuntos candidatos, bem como na geração das regras: as árvores de conjuntos e as árvores de intervalos. As primeiras guardam os *itemsets* (conjuntos de itens). Cada nó da árvore representa um *itemset*, formado pelo item armazenado no nó corrente e todos os seus ancestrais. Considerando o nível 0 como sendo aquele que armazena o nó raiz, o nível 1, guarda todos os grandes conjuntos de tamanho 1. Da mesma forma, o nível *k* da árvore guarda os k-*itemsets*. A figura 9 representa os conjuntos de itens{{Açúcar},{Leite},{Pão},{Açúcar, Leite},{Açúcar, Pão},{Leite, Pão},{Açúcar, Leite, Pão}}.

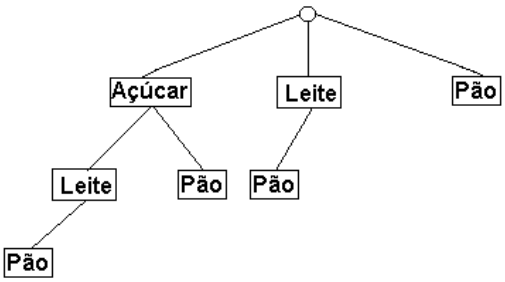


Figura 9 – Árvore de Conjuntos

Cada nó da árvore de conjuntos faz referência a uma árvore de intervalos, que é similar a uma *kd-tree* [8], armazenando informações sobre *itemranges*. Um nó da árvore de intervalos contém um conjunto de *itemranges*, denominado *j-rangeset*, que armazena as regras quantitativas do *itemset*.

Um exemplo desse tipo de estrutura é mostrado na figura 10, para o *itemset* {Açúcar, Leite}. Os *itemranges* são representados dentro dos nós. O contador de ocorrência é mostrado como "S:n", onde n é seu valor. O discriminante de um nó é uma chave que vai indicar o caminho a ser seguido no momento de se percorrer a árvore. Para cada nó, o filho da esquerda contém valores menores que o discriminante e, o da direita, valores maiores ou iguais. Este valor é escolhido baseando-se nos comprimentos dos intervalos dos *rangesets*. O discriminante será um valor intermediário no intervalo de aquisição.

Pela figura 10 pode-se perceber, por exemplo, a partir do nó marcado, que foram feitas 3 compras, cuja quantidade de item Açúcar estava entre 2 e 3 e a do item Leite, entre 4 e 8.

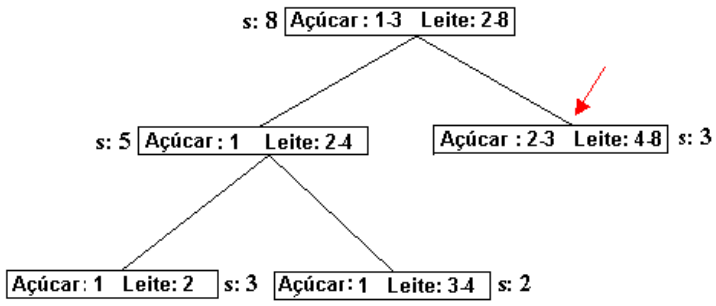


Figura 10 - Árvore de Intervalos do *itemset* {Açúcar,Leite}

As árvores de intervalos satisfazem a duas propriedades: acumulação ancestral e inclusão ancestral. A acumulação ancestral diz que o valor de um contador de ocorrência armazenado em um nó é igual à soma dos contadores dos nós filhos. Por exemplo, para o nó

raiz cujo contador é igual a 8, a soma dos contadores dos filhos também é igual a este valor. Na inclusão ancestral, os *itemranges* dos nós filhos são subintervalos dos *itemranges* do pai. Mais uma vez observando o nó raiz, a faixa para o item Açúcar é igual a 1-3. Para o filho da esquerda os valores mínimo e máximo são iguais a 1. Para o da direita, a faixa varia de 2 a 3.

Resumindo, cada nó de uma árvore de intervalos contém um conjunto de *itemranges*, chamado de *rangeset*, um contador de ocorrências e um discriminante, que é um item *a* de seu *rangeset* e um valor $d_a \in V_a$, isto é, uma quantidade adquirida do item. Dado um nó da árvore, a subárvore da esquerda contém *itemranges* onde todas as quantidades são menores que d_a, enquanto a subárvore da direita contém valores maiores ou iguais a d_a.

Essas árvores ainda possuem outra característica: os contadores de todos os nós folhas respeitam um limite especificado em tempo de execução. Sempre que a capacidade de um nó é atingida, um item é escolhido como discriminante, dois nós filhos são criados e os *rangesets* dos nós filhos são calculados com base no discriminante.

O *Apriori* simples também faz uso de uma árvore de conjuntos, e a diferença básica entre este algoritmo e o *Apriori* Quantitativo é justamente com relação à geração das árvores de intervalos. Estas árvores são utilizadas na fase de poda (seção 2.3.3) do *Apriori* Quantitativo, e são construídas enquanto o suporte do *itemset* está sendo contado. Para ilustrar, seja um conjunto candidato {Pão, Leite}. Para este candidato, a árvore deve ser construída e, durante a varredura dos dados, ela vai sendo atualizada, levando em consideração as propriedades de acumulação e inclusão ancestrais das *kd-trees*.

Apesar do uso dessas árvores ser considerado em [4] como uma otimização, na medida em que elas podem possivelmente eliminar um candidato mais cedo, dá para perceber que a criação e manutenção destas estruturas devem ser tarefas bastante onerosas, como nossa implementação do algoritmo veio a confirmar.

2.3.3. A Fase de Poda

Segundo [4], a grande melhoria do *Apriori* Quantitativo é a fase de poda, que reduz o número de candidatos gerados a serem verificados posteriormente.

Como já mencionado, o *Apriori* tradicional poda um candidato C de tamanho k sempre que qualquer subconjunto de tamanho k-1 de C não for freqüente. Para os conjuntos não podados o suporte será contado. Vale ressaltar que o grande objetivo das melhorias propostas pelos algoritmos de Regras de Associação é podar cada vez mais cedo um

candidato, justamente porque a contagem do suporte é uma fase muito onerosa, visto que todas as transações precisam ser analisadas.

Para o *Apriori* Quantitativo, a poda sugerida pelo *Apriori* simples continua a ser feita. Entretanto, após a aplicação deste critério, uma segunda fase de poda é aplicada. Isto é feito porque, apesar da poda original ser correta no sentido de que nenhum grande conjunto é descartado, essa estratégia pode levar à geração de candidatos que mais tarde mostrar-se-ão infreqüentes porque a sobreposição entre as transações dos itens nos conjuntos de tamanho k-1 não é grande o suficiente para garantir o suporte de C. A nova estratégia leva em consideração as informações quantitativas dos dados para estimar mais precisamente esta sobreposição em termos de transações.

Para ilustrar como isto funciona, seja a base de dados abaixo, apresentada em [4], a qual mostra 10 transações, cada uma delas com os itens adquiridos e a respectiva quantidade. Por exemplo, a linha 1 indica que foi adquirida 1 unidade do produto A, 1 do B, 3 do C e 4 do D. Já a segunda, mostra que a cliente adquiriu 2 unidades de A, 1 de B, 2 de C e não comprou o produto D.

1.	A	1	B	1	C	3	D	4
2.	A	2	B	1	C	2	-	-
3.	A	3	B	2	-	-	D	4
4.	A	2	B	3	C	3	-	-
5.	A	2	B	1	-	-	-	-
6.	A	3	B	2	C	3	-	-
7.	A	4	-	-	-	-	D	4
8.	-	-	B	2	C	1	D	3
9.	-	-	B	4	C	3	-	-
10.	-	-	B	1	-	-	D	1

Figura 11 – Exemplo de base de transações

Sendo considerado um suporte de 3, são encontrados cinco conjuntos freqüentes de tamanho 2: AB, AC, AD, BC e BD, com suportes 6, 4, 3, 6 e 4 respectivamente. Seguindo a estratégia de geração de conjuntos candidatos já explicada anteriormente, o *Apriori* original

geraria os conjuntos candidatos ABC e ABD de tamanho 3. Entretanto, a partir de uma simples inspeção visual da base de dados, percebe-se que somente o conjunto ABC é freqüente.

Neste ponto, aparece a proposta de otimização da fase de poda do *Apriori* Quantitativo. A partir das árvores de intervalos de AB, AD e BD mostradas na figura 12 e montadas na passagem anterior pelos dados, é possível descobrir que ABD não é um grande conjunto. Isto acontece porque o intervalo (A : 4 D : 4), destacado na figura 12, não combina com nenhum intervalo da árvore de AB, já que não existe nenhum nó no qual A está associado ao valor 4. Desta forma, as transações consideradas em AD não são consideradas em AB, como pode ser visto a partir da base de dados, onde a transação 7 não inclui B. Neste caso, ABD é insatisfatório em relação a AD. Justamente por isso que foi comentado anteriormente que a sobreposição dos intervalos é verificada. Neste caso, o intervalo {A: 4} da árvore de AD não se sobrepõe aos intervalos {A: 1-2} e {A: 3} da estrutura correspondente ao conjunto AB.

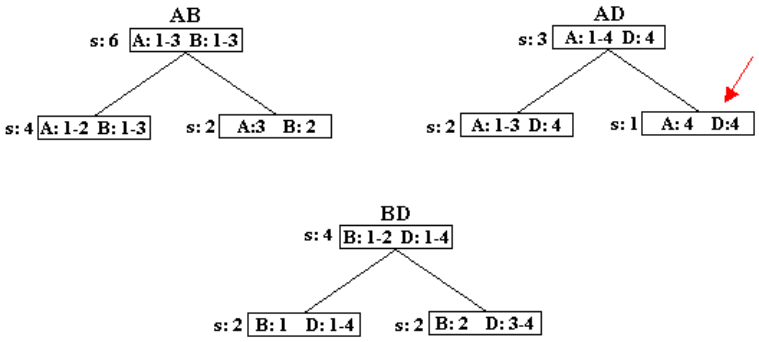


Figura 12 – Árvores de Intervalos para os *itemsets* AB, AD e BD

O algoritmo que generaliza este procedimento e se propõe a melhorar o processo de poda é mostrado na figura 13. Os intervalos são escolhidos baseando-se numa estratégia gulosa. Já que o objetivo é descartar um k-*itemset* candidato o mais cedo possível, o algoritmo concentra-se no (k-1)-*itemset* com menor suporte, que presumivelmente será o mais fácil de ser considerado insatisfatório. Além disso, começa-se checando as folhas que têm as menores faixas em todas as dimensões, chamado de *rangeset coverage*.

Por definição, dois *rangesets* sobrepõem-se (≈) quando algum de seus *itemranges* se sobrepõem (“*overlap*”). Mais especificamente, dados dois *rangesets* R = r_1, r_2, ..., r_n e S = s_1,

s_2, ..., s_m onde r_i e s_j são *itemranges*. R (≈) S se ∃ r, s | r ∈ R, s ∈ S, $r_a = s_a$, $r_l <= s_h \wedge s_l <= r_h$ [4].

Por exemplo, para a figura 12, analisando as árvores de AD e de BD, o nó marcado na estrutura relacionada a AD tem o valor 4 para o atributo D. Desta forma, sobrepõe-se a qualquer uma das folhas de BD, posto que os valores de D são {1-4} e {3-4}[4].

A partir das considerações sobre árvores de intervalos, fica claro que esse tipo de estrutura só deve ser montado para os candidatos com tamanho maior ou igual a dois. Pela definição de sobreposição de intervalos fica claro que não tem como testar esta sobreposição entre dois candidatos de tamanho 1.

O algoritmo proposto em [4] é descrito na figura 13:

```
1.      para cada candidato C
2.        enumere o conjunto P de k-1 itemsets de C
3.        se ∃ p ∈ P | support(p) < minsupp então
4.           C não é grande conjuntos
5.        senão
6.           encontre pmin | pmin ∈ P e not ∃ p | support(p) < support(pmin)
7.           ∀ folha l de Pmin
8.          overlapped_support := support(l)
9.          para cada p ∈ P e p ≠ pmin
10.                  K é o conjunto de todas as folhas de k onde k (≈) l
11.           se overlapped_support > ∑ support(k) então
12.                       overlapped_support := ∑ support(k)
13.                  se (overlapped_support = 0) então
14.                       support(pmin) := support(pmin) – support(l)
15.        se (support(pmin) < minsupp) então
16.             C não é um grande conjunto
```

Figura 13 - Fase de Poda do *Apriori* Quantitativo [4]

A parte inicial deste algoritmo mostra a estratégia de poda tradicional, na qual um candidato é eliminado se algum de seus subconjuntos não é freqüente (linhas 2 a 4, figura 13).

A segunda fase (linhas 5 a 16) explora as informações quantitativas contidas nas árvores de intervalos. O primeiro passo da poda encontra o subconjunto do candidato analisado (que tem tamanho k) com menor suporte, chamado de *pmin* (linha 6) para posterior avaliação das árvores de intervalos de todos os outros k-1 subconjuntos. Esta avaliação leva em conta todos os nós de intervalo (l) de *pmin* (linha 7). O *overlapped_support* inicial é o suporte do próprio l. Então é verificado se este suporte é válido para todos os k-1 subconjuntos. Neste ponto o algoritmo é guloso, já que começa com o subconjunto de menor suporte e verifica se ele é válido para todos os subconjuntos. Então, para cada nó considerado, o algoritmo determina quais folhas (k) nas árvores de intervalos restantes se sobrepõem às folhas na árvore de intervalos associada a *pmin* (linha 10). Então *overllaped_support* é atualizado se a soma do suporte de todo k é menor que o valor corrente do suporte (linhas 11 e 12). Deve-se enfatizar que esta soma de suportes é um limite superior no suporte que l pode ter em p e, se a faixa é menor que o valor corrente total, então ele torna-se o novo suporte para aquele *itemrange*. Se, depois de verificados todos os nós, o *overllaped_support* resultante for zero, o suporte total para *pmin* é decrementado pelo suporte de l (linhas 13 e 14), significando que l abrange um *itemrange* que não está presente em todos os subconjuntos necessários para o novo candidato. Finalmente, se o suporte para *pmin* depois das verificações for menor que MinSup então C é dito insatisfatório (linhas 15 e 16) e, portanto, descartado.

Exemplificando todo este processo, seja o candidato ABD, aqui chamado de candidato C, cujos subconjuntos AB, AD e BD são todos grandes, considerando um suporte mínimo igual a 3. Portanto, o candidato não será descartado na primeira fase de poda, aquela utilizada tanto pelo *Apriori* simples quanto pelo Quantitativo. Partindo para a poda proposta em [4], o algoritmo irá buscar *pmin*, o subconjunto de C com menor suporte (linha 6). Neste caso *pmin* = AD, cujo suporte é igual a 3. Então cada folha *l* de *pmin* será buscada (linha 7). A variável *overlapped_support* recebe o suporte da folha em análise (linha 8). Escolhendo a da direita, o valor da variável será então 1. Para cada um dos outros subconjuntos de C (exceto *pmin*), K será o conjunto de folhas que sobrepõem *l* (linhas 9 e 10). No caso do subconjunto BD, as duas folhas se sobrepõem a *l*. O somatório dessas duas folhas é 4. Então *overllaped_support* é menor que este valor e não sofre alteração (linha 11). Entretanto, para o subconjunto AB, nenhuma folha se sobrepõe a *l*, *overlapped_support* recebe então o valor 0, somatório das folhas que sobrepõem àquela que está sendo analisada, visto que o valor anterior é menor que

este somatório (linhas 11 e 12). Por fim, se *overllaped_support* for zero, a folha não se sobrepõe a ninguém. Então, o suporte de *pmin* é decrementado do suporte da folha (linhas 13 e 14). AD passa então a ter suporte 2, menor que o mínimo. Desta forma, ABD passa a ter um subconjunto que não é grande e é, então, descartado (linhas 15 e 16).

2.4. Crítica aos Algoritmos da Família *Apriori*

À guisa de conclusão, o principal problema dos algoritmos da família *Apriori* é quanto ao enorme número de candidatos a grandes conjuntos que terão seus suportes contados. Desta forma, o grande esforço das pesquisas tem sido para tentar antecipar cada vez a poda.

O algoritmo *Apriori* básico tem uma fase de poda bastante simples, baseando-se apenas no fato de que todos os subconjuntos de um grande conjunto com suporte maior que o mínimo têm também suporte maior ou igual ao mínimo. Já a versão quantitativa tenta elaborar a poda fazendo uso de estruturas de dados auxiliares. O problema, no entanto, é verificar até que ponto o uso dessas estruturas é bom, no sentindo de melhorar a poda sem prejudicar o desempenho global do *Apriori* Quantitativo. O esclarecimento desta questão aberta é o principal objetivo do nosso trabalho.

Capítulo 3

Integração de Algoritmos de Regras de Associação com SGBD's

Muito investimento em pesquisa e desenvolvimento tem sido feito para o contínuo aprimoramento dos SGBD's. Entre suas numerosas características importantes, destacamos: robustez, escalabilidade, controle de acessos concorrentes e otimização de consultas. Em contrapartida, as primeiras aplicações de mineração de dados foram desenvolvidas em sistemas de arquivos com estruturas de dados e estratégias de gerenciamento de memória especializadas. Grande parte das aplicações atuais de mineração de dados ainda tem uma fraca conexão com SGBD's: na melhor das hipóteses, eles são utilizados apenas como repositórios, a partir dos quais os dados são extraídos e preparados para mineração [5]. Isto contraria a lógica elementar de que SGBD's e algoritmos de mineração de dados deveriam se complementar, os primeiros tratando da gerência dos dados a minerar, enquanto os últimos cuidariam da mineração propriamente dita.

Felizmente, trabalhos recentes analisam alternativas para a integração de técnicas de mineração com SGBD's([5], [6], [11], [13], [15]). O objetivo principal deste capítulo é abordar algumas propostas para esta integração.

3.1. Abordagens de Integração SGBD's – Mineração de Dados

Propostas para integração de mineração de dados e SGBD's foram estudadas em ([5], [6], [11], [25]). Entretanto, em todos os trabalhos, a classificação das abordagens divide-se basicamente em: convencional, fortemente acoplada e caixa preta.

Na categoria *convencional*, também chamada de fracamente acoplada, não existe nenhum tipo de integração entre o SGBD e a aplicação. A maioria das atividades de mineração de dados reside fora do controle do SGBD. Este serve, na melhor das hipóteses,

como repositório para armazenamento dos dados. A implementação pode ocorrer de duas formas. Na primeira, os dados sob um SGBD são lidos tupla a tupla, através de um cursor SQL[3] — enfoque "*loose coupling*" ([6], [11]). O maior atrativo da abordagem é a flexibilidade de programação, visto que o algoritmo de mineração é implementado completamente no lado aplicação. Além disso, qualquer aplicação "*stand alone*" de mineração de dados pode facilmente passar a utilizar dados sob a gerência de um SGBD. Em contrapartida, um problema potencial com esta alternativa é o alto custo da troca de contexto entre o SGBD e o processo de mineração, já que eles estão em diferentes espaços de endereçamento [5]. Tem mais problemas: nenhuma das facilidades para grandes bancos de dados oferecidas por um SGBD (paralelismo, otimização de consultas, escalabilidade, controle de acesso concorrente, etc) é aproveitada [25], apesar de serem de extrema importância para aplicações de mineração, visto que estas geralmente lidam com grandes volumes de dados.

A segunda alternativa da categoria convencional é uma variação da primeira, na qual os dados provenientes de um SGBD são armazenados temporariamente em um arquivo fora do banco ("*flat file*"), especialmente preparado para mineração. Essa estratégia é denominada "*Cache-Mine*" ([6], [11]). Comparada com a alternativa anterior, pode significar uma melhora no desempenho, porém continua sem tirar proveito das facilidades do SGBD [25], além de requerer espaço extra de armazenamento.

Na categoria *fortemente acoplada*, parte do algoritmo de mineração pode ser codificado e processado no núcleo de um SGBD — lado servidor —, através de *stored procedures* e funções definidas pelo usuário. Desta forma, as operações de acesso a dados, que consomem mais tempo, são mapeadas para SQL e executadas no lado servidor. A aplicação de mineração de dados comporta-se então como uma aplicação cliente, ficando com a tarefa de visualizar o conhecimento minerado no lado servidor. Fica então evidenciado que, desta forma, a aplicação de mineração como um todo pode se beneficiar de vantagens próprias de um SGBD, como otimização de consultas e todas as demais vantagens anteriormente mencionadas [25].

Finalmente, na categoria caixa preta [25], aos olhos do usuário, o algoritmo de mineração é completamente encapsulado dentro do SGBD. A aplicação envia uma simples requisição solicitando a extração de algum conhecimento e recebe o resultado final como

[3]Um cursor recebe o resultado da consulta permitindo que o mesmo seja tratado linha a linha.

resposta. Note que se trata também de uma integração fortemente acoplada. Pode-se até dizer que, do ponto de vista do desempenho, esta é a melhor alternativa, visto que todos os recursos do SGBD são explorados. A desvantagem potencial é que a integração é escrava do particular algoritmo de mineração implementado e sabe-se que nenhum algoritmo de mineração é o mais adequado para todos os conjuntos de dados a minerar.

3.2 A Ubiqüidade do Algoritmo *Apriori* de Regras de Associação

Em praticamente todos os trabalhos de integração de mineração de dados sob a forma de regras de associação com SGBD's, o algoritmo analisado é sempre o clássico *Apriori*.

Ao mesmo tempo, o modelo das bases de dados pode influenciar e muito na execução do algoritmo. Relembrando a base trazida no capítulo 1, tabela 1, esta modela as transações de uma mercearia. Para cada produto existe uma coluna e, caso o item tenha sido adquirido em uma dada transação, a coluna referente àquele item recebe o valor 1. Caso contrário, 0. Esta é uma base especialmente preparada para mineração e seria ideal para a execução dos algoritmos, visto que nela os dados de uma mesma transação já estão agrupados. Entretanto, na prática seria praticamente impossível fazer uso de base uma semelhante, justamente porque o número de itens pode variar muito e muito espaço seria gasto com os dados, além de ser necessária uma coluna para cada item.

Nos trabalhos de integração propostos, as bases de dados mineradas são puramente relacionais e normalizadas, como ilustrado na figura 14, na qual a tabela de transações contém duas colunas: identificador da transação (tid) e identificador do item (item). É evidente que o número de itens adquiridos por transação é variável; este fato foi a justificativa dada em ([6], [11]) para se usar estruturas puramente normalizadas.

Tid	*Item*
001	Cerveja
001	Coca-cola
002	Cerveja
002	Coca-cola
003	Batata frita
003	Cerveja
003	Coca-cola
004	Batata frita
004	Cerveja
004	Coca-cola
005	Batata frita
005	Cerveja
005	Coca-cola

Tid	*Item*
006	Batata frita
006	Cerveja
006	Coca-cola
007	Batata frita
007	Cerveja
008	Batata frita
008	Cerveja
008	Coca-cola
009	Batata frita
009	Coca-cola
010	Batata frita
010	Cerveja
010	Coca-cola

Figura 14 - Base de dados puramente normalizada

No entanto, esta abordagem pode sofrer muito visto que será ainda necessário agrupar os dados de uma mesma transação. Com o surgimento dos SGBD's objeto-relacionais passa a ser permitida a desnormalizacão através do uso de coleções. A figura 15 traz então uma proposta de modelagem de dados na qual os itens de uma mesma transação já se encontram agrupados. As transações são idênticas às da figura 14:

Transação	Itens Adquiridos
001	Cerveja, Coca-cola
002	Cerveja, Coca-cola
003	Batata frita, Cerveja, Coca-cola
004	Batata frita, Cerveja, Coca-cola
005	Batata frita, Cerveja, Coca-cola
006	Batata frita, Cerveja, Coca-cola
007	Batata frita, Cerveja
008	Batata frita, Cerveja, Coca-cola
009	Batata frita, Coca-cola
010	Batata frita, Cerveja, Coca-cola

Figura 15 - Base de dados desnormalizada

Assim, a base da figura 15 seria ideal para algoritmo de regras de associação. Nela, ao contrário, da figura 14, os dados de uma mesma transação estão agrupados e, ao mesmo tempo, não existe desperdício de espaço como na proposta da tabela 1.

Desta forma, o presente trabalho de pesquisa concentrou-se em duas inovações em relação à integração de mineração de dados e SGBD's: a implementação do *Apriori* Quantitativo e a análise comparativa de seu desempenho em relação ao *Apriori* original, ambos completamente integrados a um SGBD; e comparações efetivas dos resultados com o uso de estruturas relacionais e com extensões objeto-relacionais.

3.2. Ferramentas para Mineração de Dados no *Oracle*

O SGBD *Oracle* traz desde a sua versão 8i ferramentas integradas para mineração de dados. Na versão 9i, a ferramenta apresentada é denominada "*Oracle 9i Data Mining*". Esta seção tem o objetivo de analisar alguns detalhes desta ferramenta, como técnicas e algoritmos utilizados.

O *Oracle9i Data Mining* suporta a implementação de algoritmos para algumas técnicas de mineração de dados. Para regras de classificação, os algoritmos *Adaptive Bayes Network, Naive Bayes* e *Model Seeker* podem ser utilizados ([16], [17], [26]). A escolha da utilização de cada um deles vai depender dos requisitos da aplicação em análise.

No caso de regras de associação, apenas o *Apriori* foi utilizado. Entretanto, não se sabe se algum tipo de otimização foi utilizada. Desta forma, fica ressaltado aqui um dos principais problemas dos pacotes prontos: a incapacidade de descoberta da real implementação e, conseqüentemente, a falta de parâmetros para análise de possíveis melhorias.

Desta forma, o *Oracle* até oferece uma ferramenta já pronta para mineração de dados, entretanto, sua utilização fica completamente presa ao que foi definido pelo fornecedor. Além disso, no caso de regras de associação, o algoritmo utilizado é o seminal *Apriori.*

Capítulo 4

Integração do Algoritmo *Apriori* Quantitativo com o SGBDOR *Oracle9i*

Este capítulo tem como objetivo apresentar os detalhes da implementação do algoritmo *Apriori* Quantitativo, integrado ao SGBDOR *Oracle 9i* — doravante *Oracle*, simplesmente.

Inicialmente, são mostrados alguns detalhes de como a infraestrutura de otimização utilizada pelo *Apriori* Quantitativo foi implementada. Todas as estruturas auxiliares utilizadas pelo algoritmo foram simuladas em tabelas e criadas em tempo de execução. Em seguida, cada uma das fases do algoritmo tem sua implementação descrita e explicada em detalhes. São feitas ainda considerações sobre os modelos das bases de dados utilizadas como entrada para os algoritmos e suas implementações. O capítulo é finalizado com um exemplo passo a passo de execução do *Apriori* Quantitativo, detalhando a montagem e manutenção de todas as estruturas dinâmicas.

4.1. Utilização de SQL Dinâmico no *Oracle*

Nas primeiras versões da linguagem de desenvolvimento de aplicações do *Oracle*, PL/SQL, os únicos comandos SQL suportados eram os chamados comandos DML ("*Data Manipulation Language*") – SELECT, INSERT, UPDATE, DELETE —– e comandos de controle de transações. Comandos DDL ("*Data Definition Language*") - utilizados para definir e criar objetos – não eram suportados. Também não havia suporte a comandos dinâmicos: a estrutura de todos os comandos dentro de um programa PL/SQL tinha que ser conhecida em tempo de compilação, sendo impossível criar um objeto qualquer, como uma simples tabela, em tempo de execução.

A restrição de não permitir comandos DDL diretamente em uma aplicação PL/SQL está intimamente relacionada ao projeto da linguagem. Com o objetivo de deixar a execução

mais rápida, a compilação foi projetada para ser mais demorada, em benefício da execução. A grande desvantagem desta abordagem é sua inflexibilidade, o que pode levar a restrições inaceitáveis. Considere, por exemplo, um comando de criação de uma tabela: muitas vezes, a estrutura da tabela (número e definição das colunas) só pode ser conhecida em tempo de execução, inviabilizando comandos estáticos, ou impondo comandos dinâmicos.

Com SQL dinâmico, um comando SQL só é detalhado em tempo de execução. A criação dinâmica de uma tabela passa a ser possível. Em tempo de compilação, um comando dinâmico é completamente invisível, ou seja, o compilador simplesmente ignora determinados comandos ao 'perceber' que se trata de SQL dinâmico. Assim, o comando só é analisado de fato no momento de sua execução. Da mesma forma, comandos DML podem ser definidos dinamicamente.

PL/SQL 2.1 (*Oracle7* release 7.1) trouxe a primeira implementação de SQL dinâmico — o pacote DBMS_SQL [19], que foi melhorado para o *Oracle8*. Com o pacote, um comando SQL deve ser montado em uma "string" interpretada em tempo de execução. Procedimentos como o DBMS_SQL.PARSE, que analisa gramaticalmente o comando, o DBMS_EXECUTE, para executar o comando, e o DBMS_SQL.VARIABLE_VALUE, para recuperar valores de variáveis, são disponibilizados no pacote.

O passo seguinte foi tornar os comandos dinâmicos parte integrante da linguagem, significativamente superior ao pacote DBMS_SQL. Comandos dinâmicos SQL ainda são montados em uma "string". A partir daí, os comandos podem ser executados diretamente com comando *EXECUTE IMMEDIATE*, que imediatamente analisa e executa o comando embutido na "string" .

Toda a infraestrutura necessária ao funcionamento do algoritmo de mineração de dados *Apriori* Quantitativo foi implementada por meio de SQL dinâmico.

4.2. Árvores de Grandes Conjuntos

Uma árvore de grandes conjuntos foi representada através de uma tabela *Oracle*. As próximas seções detalham toda a implementação, inclusive com a descrição das operações possíveis sobre a estrutura.

4.2.1. Simulação da Árvore de Conjuntos em Tabela *Oracle*

Seja a estrutura da figura 16, com cada um dos nós numerados, para facilitar a compreensão do leitor.

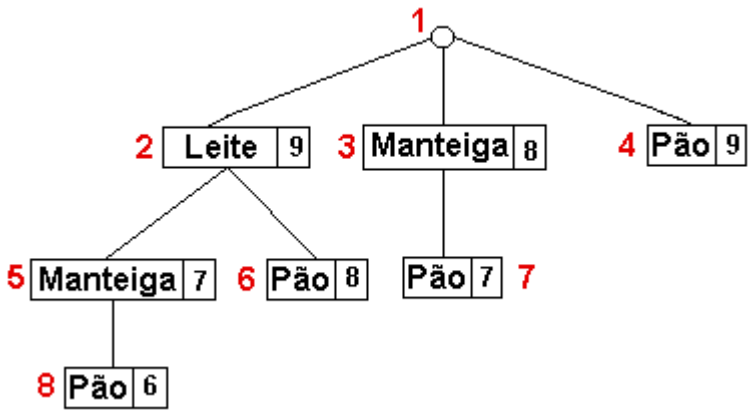


Figura 16 – Árvore de Conjuntos com numeração dos nós

A árvore de conjuntos é utilizada pelo algoritmo *Apriori* com a função de armazenar os grandes conjuntos, ou seja, aqueles que já tiveram seu suporte contado e que obedecem ao suporte mínimo estabelecido. Da mesma forma, a estrutura é necessária à execução do *Apriori* Quantitativo.

Cada linha da tabela (figura 17) que simula uma árvore de grandes conjuntos representa um nó da árvore e contém o código do item que o nó representa e a quantidade de transações que contém o conjunto (suporte). Cada conjunto é formado pelo nó corrente e todos os seus ancestrais. Desta forma, no nível K estão sendo representados os grandes conjuntos de tamanho K, bem como o respectivo suporte. Assim, o nó 8 representa o grande conjunto {Leite, Manteiga, Pão} e tem seu suporte associado. Os valores do suporte estão baseados nos dados contidos no exemplo de base de transações trazido na tabela 1, capítulo 1.

Para representar a árvore em forma tabular e permitir o caminhamento pelos seus nós, a modelagem da simulação da árvore foi feita da seguinte forma: cada linha da tabela representa um nó da árvore e os apontadores para os nós filhos são simulados através de *ROWID's* (identificador único para uma determinada linha de uma tabela). Assim, no nó pai existem os endereços dos filhos, descritos por seus *ROWID's*. O nó raiz, rotulado com o número 1 na figura 16, deve armazenar o identificador (*ROWID*) dos nós 2, 3 e 4. Além disso, como o número de nós filhos pode variar de nó para nó, cada nó contém, na verdade, um campo do tipo *Oracle* "*Varray*" — coleção unidimensional e homogênea de elementos; *array*

de tamanho variável, porém com um limite máximo definido previamente —, em que cada elemento é o *ROWID* de um nó filho.

No processo de busca e caminhamento pela árvore, tudo se inicia a partir do nó raiz. Este é reconhecido pelo valor do suporte que foi definido em "0" como padrão. Então, sempre que é necessário recuperar um conjunto, a busca é iniciada no nó raiz e os *apontadores* são percorridos a partir do *ROWID*. A implementação de ponteiros através de *ROWID´s* é de suma relevância, visto que a recuperação de uma linha de uma tabela através de seu *ROWID* é um recurso ultraeficiente [21].

A figura 17 mostra a estrutura da tabela que simula uma árvore de conjuntos como a da figura 16.

Suporte	**Código do Item**	**Coleção de ROWID's (*Varray*)**

Figura 17 – A estrutura da tabela que simula uma árvore de conjuntos

A figura 18 é a instanciação da tabela que representa a árvore de conjuntos da figura 16. Consideram-se 1, 2 e 3 os códigos dos itens Leite, Manteiga e Pão, respectivamente. Somente no momento de geração das regras é que são buscadas as descrições correspondentes aos códigos dos itens.

**	Suporte	Item	Coleção de ROWIDs	Grande Conjunto**
Linha 1	0	000	{ROWID Linha2, ROWID Linha3, ROWID Linha4}	Nó raiz
Linha 2	9	001	{ROWID Linha5, ROWID Linha6}	{Leite}
Linha 3	8	002	{ROWID Linha7}	{Manteiga}
Linha 4	9	003	Array sem nenhuma posição alocada*	{Pão}
Linha 5	7	002	{ROWID Linha8}	{Leite, Manteiga}
Linha 6	8	003	Array sem nenhuma posição alocada*	{Leite, Pão}
Linha 7	7	003	Array sem nenhuma posição alocada*	{Manteiga, Pão}
Linha 8	6	003	Array em nenhuma posição alocada*	{Leite, Manteiga, Pão}

* - Linhas referentes aos nós folha. Nenhuma posição do array foi alocada até então.

** - Coluna que na realidade não existe: serve apenas para facilitar o entendimento do leitor.

Figura 18 – Simulação da árvore da figura 16 por meio de tabela

A primeira ‘coluna’ está representando a informação implícita dos *ROWID’s* das linhas representando os nós da árvore: *Linha 1*, *Linha 2*, etc. Na verdade , um *ROWID* é um endereço *Oracle* composto por 18 caracteres, como: ‘AUIPKLJNHXYZ6O7PZC’.

Pode parecer estranho à primeira vista que um mesmo item apareça várias vezes em uma tabela. Na figura 18, existem 4 linhas representando o item “Pão”. Voltando à figura 16, pode-se perceber que nela também aparecem 4 nós relativos ao mesmo item. A princípio, poder-se-ia ter a impressão de que tal repetição não seria necessária, bastando apenas alguns cuidados com os apontadores. Entretanto, tal abordagem pode falhar e isso pode ser explicado. Sejam {Leite, Manteiga} e {Manteiga, Pão} dois grandes conjuntos. Se não houvesse a repetição de nós, o nó referente ao item Leite estaria apontando para aquele ligado ao item {Manteiga}. Este, por sua vez, seria o mesmo que apontaria para {Pão}. Desta forma, o conjunto {Leite, Manteiga, Pão} também seria tido como grande, o que poderia não ser verdade. Isto comprova a necessidade de repetição dos valores para a correta representação dos grandes conjuntos.

4.2.1.1. Rotinas de Construção e Manipulação de Tabelas para Árvores de Conjuntos

Inicialmente foram definidos alguns tipos (“*Object Types*”) como base para a implementação dos demais, conforme a tabela 4. Um *“Object Type”* é uma construção de banco de dados *Oracle* e é equivalente a uma classe de objetos, com atributos e métodos.

A tabela 4 descreve os tipos criados e utilizados nas rotinas da tabela-árvore de conjuntos.

NOME DO TIPO	ATRIBUTOS	DESCRIÇÃO
APONT_T	VARRAY DE CHAR(18)	Serve para implementar os apontadores para os nós filhos.
CONJUNTO_T	VARRAY DE NUMBER	Representa um conjunto de itens (*itemset*), seja ele candidato ou grande, através de seus códigos. Geralmente serve para ser passado como parâmetro das rotinas.
NO_T	PRODUTO number, SUPORTE number, APONT APONT_t, APONTPAI CHAR(18)	Representa um nó da árvore com o código do item referente àquele nó, o suporte do conjunto e o array de apontadores, ressaltando que o atributo APONT_PAI foi criado para auxiliar na geração das regras.

Tabela 4 – Tipos utilizados pela tabela árvore de conjuntos

Onde existe array variável, o tipo de cada posição é *CHAR(18)* porque o *Oracle* não permite um *VARRAY* cujos elementos sejam do tipo *ROWID*. Entretanto, a conversão de *CHAR* para aquele outro tipo é implícita. Desta forma, sempre que um atributo servir para guardar um *ROWID*, ele será definido como *CHAR(18)*.

As várias operações possíveis sobre uma árvore de conjuntos são descritas a seguir. A definição completa de todas elas e dos tipos criados pode ser encontrada no Anexo 1.

4.2.1.2. Criação da Árvore

Esta é a rotina mais simples e encarrega-se de criar uma nova tabela representando a árvore de conjuntos. O procedimento tem como parâmetro o nome que a tabela deve receber e cria uma "*object table*" com cada objeto do tipo NO_T. Uma *object table* pode ser definida

como uma tabela criada através de um *Object Type*. Suas colunas consistem exatamente dos atributos definidos no tipo e cada linha é em si um objeto (instância do *object type*).

Além de criar a tabela, a rotina insere o nó raiz com o "*flag*" (código do item = 000) que o identifica. Como os apontadores são feitos através de um *VARRAY*, as posições deste array vão sendo criadas somente quando for necessário. Desta forma, para um novo nó, o array não conterá valor algum e nem posições alocadas.

O processo de criação da árvore é sempre feito no início da execução. No final do algoritmo, a tabela é apagada.

4.2.1.3. Inserção de um Conjunto

Este procedimento permite que um novo grande conjunto seja inserido na tabela árvore de conjuntos. São passados como parâmetros o nome da tabela e uma variável do tipo *Conjunto_t* com o conjunto a ser inserido. O procedimento é executado sempre ao final de cada fase para a inserção dos grandes conjuntos encontrados naquela etapa. Exemplificando, inicialmente são adicionados à árvore os conjuntos de tamanho 1, como {Leite} e {Manteiga}; depois de tamanho 2, semelhantes a {Leite, Manteiga} e assim por diante.

Cabe ressaltar que a ação que sempre ocorrerá através da execução desta rotina é a criação de apenas um novo nó na árvore, independente do tamanho do conjunto a ser inserido. Isso acontece porque, caso o conjunto tenha apenas 1 item, ou seja, um grande conjunto de tamanho 1, um novo filho do nó raiz será criado. No entanto, caso seja necessário inserir um conjunto como {Açúcar, Leite, Pão} necessariamente o subconjunto {Açúcar, Leite} já existe, visto que qualquer subconjunto de um grande conjunto também é grande. Desta forma, a partir do nó raiz, a árvore será percorrida em busca de {Açúcar, Leite}. Ao encontrar tal conjunto um novo nó será criado e um apontador no nó correspondente a "Leite" será direcionado para o novo nó. O procedimento é descrito na figura 19:

```
InsereNo(Tabela varchar2,CONJUNTO CONJUNTO_T)
    Início
      RECUPERA O NO RAIZ;
      Se tamanho CONJUNTO <> 1 então
        Percorra a árvore procurando o conjunto até o penúltimo elemento
      END IF;
      Crie um novo nó;
      Atualize apontadores;
    Fim;
```

Figura 19 – Rotina de Inserção de um novo Conjunto na Árvore de Conjuntos

4.2.1.4. Pesquisa por um Conjunto

Esta função recebe um determinado conjunto como parâmetro e verifica se ele existe ou não na tabela árvore de grandes conjuntos.

A rotina é utilizada no primeiro critério de poda do algoritmo, aquele que diz que qualquer subconjunto de um grande conjunto necessariamente também é grande. Desta forma, um conjunto candidato terá sua poda antecipada caso seja verificado que pelo menos um de seus subconjuntos não é grande. Assim, sempre que os candidatos forem gerados, para cada um deles, todos os seus subconjuntos serão pesquisados. A função retorna um valor indicando se o subconjunto foi ou não encontrado. O fato de pelo menos um subconjunto não ser encontrado causa o descarte do candidato em análise.

Os passos da rotina são brevemente descritos na figura 20:

```
PesquisaConj(Tabela varchar2,CONJUNTO CONJUNTO_t) RETURN boolean
   Início
      Recupere o nó raiz
      Contador = 0;
      Para i = 1 até tamanho de Conjunto faça
         Para cada apontador no nó atual faça
            Busque o filho pelo apontador;
            Se Filho.Produto = Conjunto(i) então
               Achou  := true;
               NoAtual  := NoFilho;
                Contador := Contador + 1;
            Fim do se;
         Fim do para;
      Fim do para;
      Se Contador = Tamanho do Conjunto então
         Achou := true;
      Senão
         Achou := false;
      Fim do se;
      Retorne Achou;
   Fim;
```

Figura 20 – Rotina de busca de um Conjunto na Árvore de Conjuntos

4.2.1.5. Pesquisa de Suporte

Esta função é bastante simples e, dado um conjunto como parâmetro, seu suporte é retornado. É semelhante àquela descrita para busca de um conjunto (seção 4.2.1.4). A diferença é que, aqui, o valor do suporte é retornado. A partir da raiz, a árvore é percorrida até chegar ao último elemento do conjunto, cujo nó contém o respectivo suporte.

4.2.1.6. Atualização de Suporte

Este procedimento recebe o conjunto como parâmetro e seu novo suporte. A árvore é então percorrida até chegar no nó correspondente ao último elemento do conjunto, o qual terá seu suporte atualizado.

4.3. Árvores de Intervalos

Esta seção visa explicar como as árvores de intervalos, similares a *kd-trees* [8], foram implementadas. Inicialmente são feitas algumas considerações sobre as diferenças entre árvores de intervalos e *kd-trees* propriamente ditas, de maneira a justificar alguns detalhes da implementação das primeiras no *Oracle*.

4.3.1. Algumas Considerações sobre *kd-trees*

As árvores utilizadas pelo algoritmo *Apriori* Quantitativo diferem fundamentalmente, em alguns aspectos, de uma *kd-tree* clássica. Embora adotássemos *kd-trees* como base para a nossa implementação, as diferenças exigiram a adaptação das rotinas de manipulação de *kd-trees* às reais necessidades do algoritmo.

Em uma *kd-tree*, cada nó da árvore contém um registro com *k* valores, onde um deles serve como discriminante, ou seja, chave de pesquisa indicando se o caminhamento continuará pela subárvore da esquerda ou da direita. Se o valor procurado for menor que o discriminante o caminhamento deve ser feito pelo filho esquerdo. Caso contrário, pelo direito. A escolha de qual dos valores de atributo servirá como chave é feita de acordo com o nível corrente e levando-se em consideração a dimensão da árvore. Em uma *2d-tree* (árvore de duas dimensões, com dois valores de atributo em cada nó), no nível 0, o primeiro valor do registro é o discriminante, no nível 1, o segundo, já no terceiro nível a chave volta a ser o primeiro atributo e assim sucessivamente.

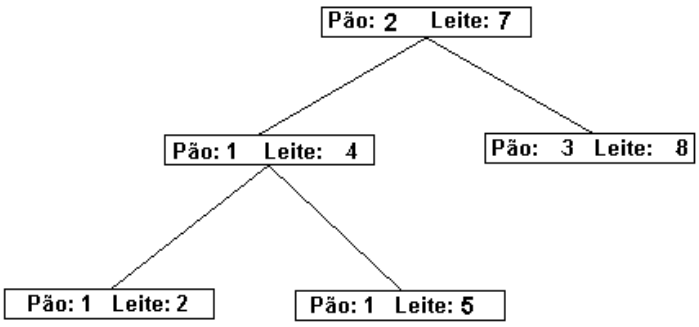


Figura 21 – Exemplo de uma *kd-tree*

A figura 21 representa uma *2d-tree*. Assim, ao se tentar inserir um novo nó, se o valor relativo a "Pão" for menor que 2, o caminhamento é feito pela esquerda. Caso contrário, pela direita. No segundo nível o discriminante é o segundo atributo, qual seja, "Leite". No terceiro volta a ser o atributo "Pão" e assim sucessivamente. O discriminante é o próprio valor do atributo no nó.

Em contrapartida, a estrutura utilizada no *Apriori* Quantitativo armazena os atributos com as faixas de valores adquiridos (informações quantitativas) e é justamente aí que começam as diferenças. Seja a figura 22, que simula uma árvore de intervalos como as que são utilizadas pelo algoritmo.

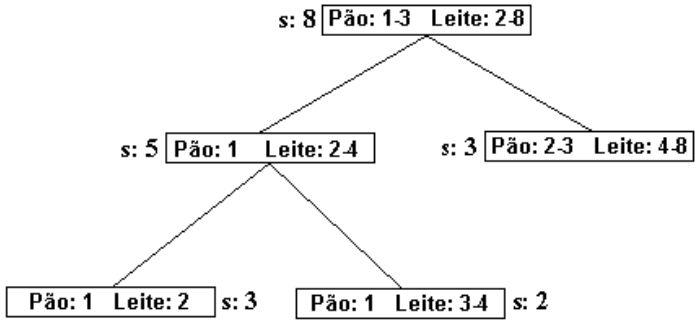


Figura 22 - Árvore de Intervalos utilizada pelo *Apriori* Quantitativo

A partir da árvore mostrada na figura 22 percebe-se que são guardados intervalos de valores adquiridos, ao invés de apenas um valor como na *kd-tree*. Além disso, é necessário manter um contador para cada nó que indica a quantidade de transações efetuadas que correspondem àquelas faixas de valores para todos os atributos, o que também não existe na estrutura original. Ante o exposto, algumas considerações foram feitas para a construção das

rotinas. Primeiro, o discriminante de um nó passa a ser um item *a* de seu *rangeset* e um valor $d_a \in V_a$ (seção 2.3.2), isto é, a quantidade adquirida do item e deve ser escolhido baseado nos comprimentos dos intervalos no *rangeset*. Para esta escolha, foi definida uma função que calcula um valor intermediário do intervalo. Exemplificando, para o nó raiz da figura 22, cuja faixa do atributo chave "Pão" é igual a {1-3}, o discriminante é o item Pão e valor 2.

4.3.2. Simulação de uma Árvore de Intervalos em Tabelas *Oracle*

Assim como a árvore de conjuntos, as árvores de intervalos também são simuladas através de tabelas *Oracle*. O raciocínio é o mesmo: cada linha da tabela representa um nó da árvore e os ponteiros para os nós filhos são feitos através do uso de *ROWID*'s. O que vai diferir agora é a estrutura de um nó, que será de acordo com o que está sendo exibido na figura 22. Assim, cada linha da tabela armazena um contador de ocorrência e o *rangeset* (faixa de aquisição). Além disso, como cada nó passa a ter, no máximo, dois filhos (esquerdo e direito), a tabela suportará apenas dois apontadores por linha. Cada árvore de intervalos é uma tabela *Oracle*.

A tabela 5 traz uma descrição dos tipos implementados para serem utilizados na construção das árvores de intervalos. Cada tipo é seguido da sua lista de atributos:

NOME DO TIPO	**ATRIBUTOS**	**DESCRIÇÃO**
ITEMRANGE_T	Minimo number, Maximo number,	Representa a faixa de quantidades adquiridas para um determinado atributo, com seu valor mínimo e máximo.
RANGESET_T	VARRAY DE ITEMRANGE_T	Cada nó contém um array de faixas de quantidades adquiridas. O tamanho desse array vai depender da dimensão da árvore. Por isso, foi utilizado VARRAY.

KDTREE_T	Contador number, Rangeset RANGESET_T, FilhoEsquerdo char(18), FilhoDireito char(18)	Representa um nó da árvore. Contém um contador que guarda o total de ocorrências para o *rangeset;* e dois apontadores: FilhoEsquerdo e FilhoDireito para os nós filhos.
AQUISIÇÃO_T	VARRAY DE NUMBER.	Serve para passar para a rotina de inserção as quantidades adquiridas a serem inseridas na árvore de intervalos já definida. Também é do tipo *varray* porque o número de elementos vai depender da dimensão da árvore.

Tabela 5 – Tipos utilizados pelas Árvores de Intervalos

A figura 23 ilustra o esquema básico da tabela árvore de intervalos:

Contador	Coleção de {Min-Max}	FilhoEsquerdo: ROWID	FilhoDireito: ROWID

Figura 23 – A estrutura da tabela que simula uma árvore de intervalos

Para uma determinada árvore, a coleção que contém as faixas de aquisição para os atributos (indicada na figura 23 como "Coleção de {Min-Max}") será sempre do mesmo tamanho, independente do nó. Assim, numa árvore de intervalos do conjunto {Leite, Manteiga}, cada nó conterá duas faixas de aquisição, uma para cada atributo.

Na figura 24 está sendo simulada uma árvore de intervalos de duas dimensões, onde a quantidade adquirida dos dois itens juntos é 4, e as faixas de valores adquiridos são 1-2 e 1-4. Os dois nós filhos mostram subconjuntos do nó raiz e suas respectivas freqüências.

	Suporte	**Rangeset**	**FilhoEsquerdo**	**FilhoDireito**
Linha1	4	{1-2, 1-4}	ROWID da linha2	ROWID da linha3
Linha2	2	{1-1, 1-4}	NULL	NULL
Linha3	2	{2-2, 3-4}	NULL	NULL

Figura 24 – Preenchimento de uma tabela árvore de intervalos

4.3.3. Rotinas de Construção e Manipulação de uma Árvore de Intervalos

As rotinas para manipulação das árvores de intervalos são significativamente mais complexas que as da árvore de conjuntos. Isto é justificado pela própria definição do primeiro tipo de estrutura. Várias questões devem ser levadas em consideração, como: atualização de faixas de valores adquiridos, limite do contador de ocorrências e possíveis divisões de um nó. Além disso, o algoritmo manipula várias árvores desse tipo e não apenas uma, como no caso da árvore de grandes conjuntos.

4.3.3.1. Criação de uma Árvore de Intervalos

A tabela *kd-tree* é criada dinamicamente no procedimento *CriaArvore*. O nó raiz tem o intervalo representando todos os valores possíveis para cada atributo, ou seja, tem que ter os valores mínimo e máximo adquiridos, levando em consideração todas as transações. Desta forma, tais valores precisam ser conhecidos de antemão.

É importante ressaltar que não dá para ir atualizando o intervalo dinamicamente, pois isso modificaria toda a estrutura da árvore. Por exemplo, seja um nó raiz com a faixa para o atributo chave {1-5}. Aqui, o discriminante será 3, considerando que a forma de encontrá-lo é calculando a média do intervalo. Os nós filhos contêm sempre subintervalos da raiz, como {1-2} para o filho esquerdo e {3-5}, para o direito, com seus respectivos contadores. Caso apareça agora uma transação com o valor 7 para o primeiro atributo, a faixa deve ser mudada para {1-7} e o discriminante passa a ser 4. Fica claro que isto altera toda a estrutura da árvore, visto que os subintervalos dos nós filhos também seriam alterados. Desta forma, o primeiro passo da rotina de criação da árvore é buscar estes valores, o que pode ser feito com uma simples consulta na tabela de vendas.

Como existe uma árvore para cada conjunto, o número de *itemranges*, ou seja, a dimensão do *rangeset*, em um determinado nó vai variar. Para a árvore de intervalos do item de tamanho dois {Leite, Pão}, cada nó conterá duas faixas de aquisição, uma para cada item. Aqui, esse tamanho será tratado como dimensão da árvore.

O procedimento de criação de uma nova árvore recebe como parâmetros: o nome da Tabela, o tamanho do conjunto ao qual ela se refere e um *rangeset* com todas as faixas de quantidades adquiridas para cada atributo. Quanto ao limite de um nó, não há necessidade de

passá-lo como parâmetro, visto que tal dado não precisa ser guardado na tabela. O valor pode ser passado e checado no momento da inserção de um nó.

O procedimento é explicado brevemente na figura 25:

```
CriaArvore(Tabela varchar2, Dimensão number, Rangeset Rangeset_t)
INÍCIO
    Crie nova object table de kdtree_t;
    Insira um novo nó com as faixas de quantidades adquiridas;
    Crie tabela temporária com número de atributos = Dimensão;
FIM
```

Figura 25 – Rotina de Criação de uma árvore de intervalos

4.3.3.2. Inserção de Valores

O procedimento de inserção recebe como parâmetros o nome da tabela onde os valores serão inseridos, a dimensão da tabela (informação que será utilizada para calcular o discriminante de um nó), o limite aceitável para o contador de um nó e os valores adquiridos para os atributos.

Quando é feita a divisão de um nó, é necessário saber o intervalo de valores adquiridos para os demais atributos que não compõem o discriminante. Exemplificando, para a divisão de um nó que contém os valores Pão{1-2} e Leite{1-4}, onde a chave é Pão{2}, é necessário saber qual a faixa de valores adquiridos para o atributo “Leite”, quando o valor adquirido de “Pão” foi menor do que 2, bem como a faixa de valores de “Leite” para transações com “Pão” maior ou igual a dois. A idéia inicial seria conseguir elaborar uma maneira de guardar esses dados na própria tabela que simula a *kd-tree.* Entretanto, seria necessário muito espaço extra para fazer esse armazenamento, mesmo porque o controle deve ser feito por nó, já que qualquer um deles poderá ser dividido. Esta estratégia, além do espaço extra necessário, deixaria a rotina de inserção bastante complexa. Além disso, mesmo que tais faixas fossem guardadas, no momento da criação de um novo nó, as novas faixas também precisariam ser conhecidas e armazenadas nesse novo nó, já que a chave seria outra e, conseqüentemente as faixas mudariam. Para descobrir tais informações seria necessário pesquisar dentre todas as transações, apenas no conjunto de transações já passadas pelo algoritmo de inserção, o que

exige uma forma de “marcar” as transações já analisadas na tabela de dados utilizada na mineração. Entretanto, esta tabela pode ter qualquer formato e provavelmente será muito grande, o que fez com que fosse necessário pensar em uma nova alternativa. A decisão tomada então foi a utilização de tabelas temporárias para ir guardando os valores já inseridos na árvore. Essas tabelas são extremamente simples e têm o mesmo número de atributos da dimensão da árvore. Ao se montar uma árvore de dois atributos, a tabela temporária também terá dois atributos, quais sejam, *valor1* e *valor2*. À medida que as transações vão sendo analisadas, os valores de aquisição também vão sendo inseridos na tabela temporária. Se já foram analisadas as transações com Pão{3}, Leite{2} e Pão{5}, Leite{1}, a tabela temporária irá conter duas linhas: {3,2} e {5,1}. A descoberta das faixas de valores será feita então a partir dessa tabela.

Também deve ser observado que na hora de dividir um nó é necessário saber o *total* do contador dos nós filhos. Se um nó chegou ao limite do contador, aqui considerado como 6, é necessário saber quantos dos valores adquiridos eram menores do que o discriminante e quantos eram maiores ou iguais. Esses totais serão os contadores dos novos nós criados e não necessariamente serão 3 para o filho da esquerda e 3 para o da direita. Esta informação também é consultada a partir da tabela temporária.

Uma última consideração é que não é necessário armazenar os atributos nos nós. Quando uma nova tabela é criada para um conjunto {Leite, Pão} com seus valores adquiridos, não é necessário armazenar em todos os nós os atributos “Leite” e “Pão”. Isso exigiria muito espaço extra e não traria benefício algum. Assim, a árvore armazena apenas as informações quantitativas. Para saber a que conjunto de atributos uma árvore está relacionada, foi criado um esquema padronizado de nomes. Supondo que o código do atributo “Leite” seja 2 e de “Pão” seja 5, o nome da tabela será *kdtree25* e isto é passado como parâmetro para a rotina de criação. As tabelas temporárias deverão seguir o mesmo esquema, acrescentando “_temp” ao nome da tabela. As tabelas temporárias são descartadas após a criação e atualização das *kd-trees*.

A figura 26 traz a especificação da rotina:

```
InsereNo(Tabela varchar2,Dimensao number,Limite number,Aquisicao Aquisicao_T)
INÍCIO
    NóAtual := NóRaiz;
    Nível := 1;
    Insira valores na tabela Temporária;
    Enquanto Inseriu = false faça
        Incremente contador nó atual;
        Se Nivel <> Dimensao  então
             Disc := MOD(Nivel, Dimensao);
        senão
             Disc := Dimensao;
        Fim do se;
        Chave := TRUNC(Discriminante.Minimo + Discriminante.Maximo)/2
        Se Contador NóAtual > Limite então
            Se Valor < Chave então
                 NoAtual := NoAtual.FilhoEsquerdo
            Senão
                 NoAtual := NoAtual.FilhoDireito
            Fim do se;
            Nível := Nível + 1;
            Se Nivel <> dimensão então
                Disc := MOD(Nivel, Dimensao);
            senão
                Disc := Dimensão;
            Fim do se;
           Chave := TRUNC(Discriminante.Minimo + Discriminante.Maximo)/2
        Senão
           Insira novos valores na kd-tree;
           Inseriu := true;
          Se contador NóAtual = Limite então
              Crie novo nó filho Esquerdo;
              Busque na tabela temporária o contador do Filho Esquerdo;
              Busque na tabela temporária as faixas de aquisições dos atributos;
              Atualize apontador no nó pai para o Filho Esquerdo;
              Crie novo nó filho Direito;
              Busque contador do Filho Direito;
              Busque na tabela temporária as faixas de aquisições dos atributos;
              Atualize apontador no nó pai para o Filho Direito;
          Fim do se;
        Fim do se;
    Fim do enquanto;
FIM
```

Figura 26 – Rotina de inserção de valores em uma árvore de intervalos

4.4. Implementação do Algoritmo

Esta seção visa detalhar a implementação do *Apriori* Quantitativo, que foi feita levando em consideração toda a descrição do algoritmo trazida em [4].

É importante frisar que a grande proposta de diferencial do *Apriori* Quantitativo é em relação à fase de poda dos conjuntos candidatos, a qual se propõe a eliminar mais cedo, antes mesmo da contagem do suporte, aqueles que mais tarde serão tidos como infreqüentes. A poda baseia-se agora em dois aspectos: primeiro, para cada conjunto, se pelo menos um de seus subconjuntos não for grande, o candidato é imediatamente descartado, exatamente como é feito no *Apriori* clássico; segundo, através da análise das árvores de intervalos já devidamente montadas, o candidato poderá também ser eliminado. Assim, dois critérios passam agora a ser considerados, para que seja feita uma possível poda antecipada de um candidato.

A figura 13, capítulo 2, detalha apenas a poda do *Apriori* Quantitativo.

Na figura 27 está sendo mostrado o pseudocódigo completo do algoritmo. Para a sua implementação, muito já foi explicado nas seções 4.2 e 4.3, tratando, respectivamente, das rotinas de criação e manipulação das árvores de conjuntos e das árvores de intervalos. Com exceção da poda e da manutenção das árvores de intervalos, as demais etapas do algoritmo são similares às correspondentes do algoritmo *Apriori*, que foram explicadas no Capítulo 2, seção 2.2. A implementação completa do algoritmo *Apriori* Quantitativo pode ser vista no Anexo 4.

```
APRIORIQuant(SupMin, ConfMin)
BEGIN
    SE (0 < Suporte <= 1) e (0 < Confiança <= 1) ENTÃO
        GereL1;
        Crie Arvore de Conjuntos;
        PARA (k=2; Lk-1 <> 0; k++) FAÇA
            Insira Lk-1 na Arvore de Grandes Conjuntos;
            GereCk(Lk-1);
            Pode Apriori Quantitativo;
            Crie Árvore de Intervalos para os Ck;
            PARA cada Transação T FAÇA
                PARA cada Ck FAÇA
                    SE Existe Ck em T ENTÃO
                        Incremente Suporte de Ck;
                        Atualize Árvore de Intervalos de Ck;
                    FIM DO SE;
                FIM DO PARA
            FIM DO PARA;
            Apague candidatos que não obedecem SupMin;
        FIM DO PARA
        Gere Regras;
    FIM DO SE;
END;
```

Figura 27 – O algoritmo *Apriori* Quantitativo

O algoritmo é iniciado testando os valores de suporte e confiança passados como parâmetros, os quais devem ser sempre maiores que zero 0 e menores ou iguais a 1. Depois disto, a árvore de conjuntos é criada, fazendo uso da rotina de criação. A próxima fase é a geração dos grandes conjuntos de tamanho 1, explicada na próxima seção. Em seguida, o algoritmo entra em um laço até que nenhum grande conjunto seja encontrado. Dentro deste laço, os candidatos são gerados a partir dos grandes conjuntos anteriores. Depois disso, vem a fase de poda, coração do *Apriori* Quantitativo que terá, portanto, sua explicação na seção

4.4.2. Para os candidatos restantes, as árvores de intervalos são criadas e mantidas e o suporte é contado. O último passo do algoritmo é a geração das regras.

4.4.1. Geração e Armazenamento dos Conjuntos Candidatos

A geração dos conjuntos candidatos é feita através de um "*self join*" na tabela de grandes conjuntos encontrados na passagem anterior. Para gerar C_k, conjuntos candidatos de tamanho k, a junção é feita na tabela que armazena os L_{k-1} com ela mesma, exatamente como foi explicado no capítulo 2, seção 2.2.1, para o *Apriori* simples.

Os candidatos independentemente de seus tamanhos serão sempre armazenados em tabelas, com cada linha para um conjunto com o respectivo suporte. As tabelas são criadas dinamicamente, com tantos atributos quantos forem necessários. Para guardar candidatos de tamanho 2, será criada uma tabela com dois atributos reservados aos códigos dos itens e um terceiro que guardará o suporte.

Na busca pelos candidatos de tamanho 1, C_1, é feita a primeira passagem pelos dados. Para cada transação, cada um de seus itens é analisado. Caso ele já tenha sido encontrado antes, seu suporte é incrementado. Caso contrário, um novo candidato acaba de ser descoberto e seu suporte já passa a ser 1. Depois disto, aqueles que respeitarem o suporte irão compor os L_1, utilizados para a geração dos C_2. Por questões de desempenho, os primeiros candidatos foram armazenados em um vetor na memória, enquanto tiveram seus suportes sendo contados. Só depois foi criada a tabela de grandes conjuntos de tamanho 1, com um atributo para o item e outro para o suporte. A partir da segunda fase, os candidatos já são inseridos na tabela depois do passo de *join*.

Para os candidatos de tamanho 2, o processo é feito normalmente através do *self join*, repetido para as fases posteriores, até que nenhum grande conjunto seja encontrado. A tabela 6 exemplifica uma tabela *Oracle* com candidatos de tamanho 2:

Suporte	**Produto1**	**Produto2**
7	Leite	Manteiga
8	Leite	Pão
7	Manteiga	Pão

Tabela 6 – Representação dos candidatos de tamanho 2

A inserção dos grandes conjuntos na árvore é sempre feita ao final de cada fase. Por exemplo, ao serem encontrados os L_2, estes são usados para gerar C_3 e depois os primeiros são inseridos na árvore de grandes conjuntos. A partir daí, a tabela referente a L_2 pode ser descartada, visto que, mais adiante, a geração das regras será feita através da própria árvore de grandes conjuntos.

4.4.2. Fase de Poda

Esta fase já foi detalhada na seção 2.3.3, capítulo 2. O importante que deve ser frisado aqui é que a poda baseia-se em dois pontos: primeiro, assim como no *Apriori* original, para cada candidato, cada um de seus subconjuntos é pesquisado, exceto os dois a partir dos quais o candidato atual foi gerado. Caso algum deles não seja grande, o candidato é descartado. Para o candidato {Açúcar, Leite, Pão} gerado a partir de {Açúcar, Leite} e {Açúcar, Pão}, somente precisa ser verificado se o subconjunto {Leite, Pão} é grande. Isto foi implementado exatamente desta forma sugerida pelos algoritmos. A verificação de subconjuntos é feita a partir da árvore de grandes conjuntos.

4.4.3. Geração das Regras

A geração de regras é feita a partir da árvore de grandes conjuntos que já contém todos eles com respectivos suportes, assim como explicado na seção 2.2.4, capítulo2. Então, para cada conjunto, a confiança é verificada. Um grande conjunto pode dar origem a várias regras. O *itemset* {Açúcar, Leite, Pão} pode originar as regras como: [Açúcar => Leite, Pão], [Açúcar, Leite => Pão], dentre várias outras. Assim, as combinações de itens devem ser feitas e, para cada uma, a confiança é calculada baseada nos já conhecidos suportes.

4.5. Bases de Dados Utilizadas

Esta seção visa expor e discutir os dois modelos de dados utilizados na nossa implementação do algoritmo *Apriori* Quantitativo: *relacional* e *objeto-relacional*.

Para ilustrar a aplicação dos modelos, as bases de dados apresentadas modelam as vendas de uma mercearia. O objetivo é encontrar correlação entre itens vendidos, através das versões relacional e objeto-relacional do algoritmo, quantificando a influência das estruturas de dados utilizadas sobre o desempenho global do algoritmo.

4.5.1. Base de Dados Relacional

A base de dados relacional utiliza tipos de dados convencionais e tabelas normalizadas. A figura 28 traz o script de criação da base de dados relacional, composta por três tabelas: *Item_r*, contendo informações gerais dos itens disponíveis, como código e descrição; *Venda_r*, com dados sobre as vendas dos itens; e *ItemVenda_r*, que faz o relacionamento entre vendas e itens vendidos.

```
CREATE TABLE Item_r (
  CodItem         NUMBER,
  Descrição       VARCHAR2(30),
  CONSTRAINT pk_item PRIMARY KEY (CodItem) )
  TABLESPACE BASES
/
CREATE TABLE Venda_r (
  CodVenda   NUMBER,
  Data           DATE,
  CONSTRAINT pk_venda PRIMARY KEY (CodVenda) )
  TABLESPACE BASES
/
CREATE TABLE ItemVenda_r (
  CodVenda NUMBER,
  CodItem   NUMBER,
  Qte          NUMBER,
  CONSTRAINT pk_itemvenda PRIMARY KEY (CodVenda,CodItem) )
  TABLESPACE BASES
```

Figura 28 – *Script* de criação da base relacional

A figura 29 é uma instância da base relacional da figura 28.

Cod	Item
001	Açúcar
002	Leite
003	Pão

Venda	Data
001	28/10/2001
002	29/10/2001

Venda	Item	Qte
001	002	3
002	001	1
002	003	2
002	002	6

Figura 29 – Instância da base relacional

4.5.2. Base Objeto-Relacional

A segunda base faz uso de extensões objeto-relacionais, com novos tipos nativos de dados como o tipo coleção, bem como a criação de novos tipos de dados.

Inicialmente um "*Object Type*" (um "molde", um tipo), chamado *Item_t*, é definido com atributos para armazenar as informações sobre os itens da mercearia. Neste caso, os atributos escolhidos foram os mesmos armazenados na tabela relacional *Item_r*: código do item e descrição. Em seguida, um tipo *Venda_t* é criado e contém os dados de uma determinada venda, assim como em *Venda_r*. Esses tipos foram criados através da sintaxe *CREATE TYPE .. AS OBJECT* do *Oracle*. Os dados de um tipo são armazenados em uma *Object Table* do tipo. Restrições de integridade como *chave primária* e *chave externa* dizem respeito a *object tables*, e não a tipos. No nosso exemplo de vendas, a tabela *Item_OR* armazena objetos do tipo *Item_t*. Cada linha de uma "*Object Table*" representa de fato um objeto do tipo associado à tabela, com seu OID (*Object Identifier).* Um OID pode ser a própria chave primária de uma *object table*, ou então ele pode ser gerado automaticamente pelo sistema. No exemplo, como código de item é sempre único, ele é definido como chave primária e é também utilizado como OID. Outro importante recurso do modelo objeto-relacional foi utilizado: trata-se do tipo *coleção*. No *Oracle*, os tipos coleção chamam-se *nested table* e *variable array (varray).*

Recorde-se que o tipo *varray* foi utilizado para a implementação da árvore de conjuntos, seção 4.3. Já o tipo coleção n*ested table* define uma coleção unidimensional de elementos homogêneos, sem limite em seu número, ao contrário de *varray*, em que o número máximo de elementos da coleção é previamente fixado. Também, com *nested tables*, a ordem dos elementos da coleção não é importante, ao contrário das coleções do tipo *varray*. O conceito de *nested table* encaixa-se muito bem nos requisitos da aplicação de vendas de uma mercearia. Por exemplo, uma determinada venda tem um número ilimitado e não previsto de elementos, cuja ordem não é interessante.

Assim, a tabela de vendas foi modelada como uma *nested table*, *LinhasItens_t*, guardando informações dos itens vendidos para cada uma das vendas. Cada linha da coleção é do tipo *LinhaItem_t*, definido para guardar as informações de um item em uma determinada venda como, por exemplo, a quantidade adquirida. A sintaxe da criação da coleção é *CREATE TYPE LinhasItens_t AS TABLE OF LinhaItem_t*.

Por fim, uma *object table*, *Venda_OR*, foi criada. Ela contém objetos do tipo *Venda_t*, cujos atributos são o código da venda, data e a coleção *LinhasItens_t*.

Outros detalhes são muito específicos do *Oracle*, e não serão explicados.

A figura 30 resume os tipos e *object tables* criados.

```
 CREATE TYPE Item_t AS OBJECT (
   CodItem   NUMBER,
   Descricao VARCHAR2(30) )
/
CREATE TYPE LinhaItem_t AS OBJECT (
   NumLinha number,
   CodItem  number,
   Qte      NUMBER );
/
CREATE TYPE LinhasItens_t AS TABLE OF LinhaItem_t;
/
CREATE TYPE Venda_t AS OBJECT (
   NumVenda  NUMBER,
   LinhasItens LinhasItens_t );
/
CREATE TABLE Item_OR OF Item_t ( CodItem Primary Key )
OBJECT ID PRIMARY KEY
TABLESPACE BASES
/
CREATE TABLE Venda_OR OF Venda_t (
  PRIMARY KEY (NumVenda) )
  OBJECT ID PRIMARY KEY
  NESTED TABLE LinhasItens STORE AS LinhasItens_tab (
  ( PRIMARY KEY(NESTED_TABLE_ID,NUMLINHA) )
   ORGANIZATION INDEX COMPRESS )
   RETURN AS LOCATOR
   TABLESPACE BASES
/
```

Figura 30 – *Script* de Criação da base objeto-relacional

A figura 31 é uma instância da base da figura 30.

Venda	Data	Linhas Itens	
		Coditem	Qte
001	28/10/2001	002	3
002	29/10/2001	001	1
		002	6
		003	2

CodItem	Descrição
001	Açúcar
002	Leite
003	Pão

Figura 31 – Exemplo da base objeto-relacional preenchida

4.6. O Algoritmo *Apriori* Quantitativo Através de um Exemplo

Esta seção mostra o funcionamento passo a passo da nossa implementação do algoritmo *Apriori* Quantitativo, detalhando como as principais estruturas vão sendo dinamicamente criadas e atualizadas. Para apoiar a descrição, é usado o exemplo das vendas de uma mercearia, ligeiramente ampliado com a inclusão de mais um item de venda, cujas estruturas de dados relacional e objeto-relacional foram explicadas na seção anterior.

Para uma dada transação de venda, se um item foi adquirido a quantidade vendida é associada ao item. Caso contrário, o valor associado é 0 (tabela 7).

Transação	Açúcar	Leite	Manteiga	Pão
1	1	1	3	4
2	2	1	2	0
3	3	2	0	4
4	2	3	3	0
5	2	1	0	0
6	3	2	3	0
7	6	0	0	4
8	0	2	1	3
9	0	4	3	0
10	0	1	0	1

Tabela 7 – Base de transações para ilustrar o *Apriori* Quantitativo

A base da tabela 7 é um arquivo puramente normalizado, especialmente preparado para mineração e que, portanto, limita os itens adquiridos àqueles cujas colunas estão presentes na tabela. Uma base como esta na prática seria inviável para armazenar transações de uma mercearia, visto que todos os itens teriam que ser listados na tabela e muito espaço seria desperdiçado para armazenar informações de itens que nem mesmo foram comprados.

Na versão relacional do *Apriori* Quantitativo a base de dados, com os mesmos valores da tabela 7, fica conforme figura 32.

CodItem	Descrição
1	Açúcar
2	Leite
3	Manteiga
4	Pão

CodVenda	Data
1	28/10/2001
2	28/10/2001
3	28/10/2001
4	28/10/2001
5	28/10/2001
6	29/10/2001
7	29/10/2001
8	29/10/2001
9	29/10/2001
10	29/10/2001

CodVenda	CodItem	Qte
1	1	1
1	2	1
1	3	3
1	4	4
2	1	2
2	2	1
2	3	2
3	1	3
3	2	2
3	4	4
4	1	2
4	2	3
4	3	3
5	1	2
5	2	1
6	1	3
6	2	2
6	3	3
7	1	6
7	4	4
8	2	2
8	3	1
8	4	3
9	2	4
9	3	3
10	2	1
10	4	1

Figura 32 – Base de transações relacional

Já a base objeto-relacional, a base é similar à figura 33.

CodVenda	Data	Linhas Itens	
		Coditem	Qte
1	28/10/2001	1	1
		2	1
		3	3
		4	4
2	28/10/2001	1	2
		2	1
		3	2
3	28/10/2001	1	3
		2	2
		4	4
4	28/10/2001	1	2
		2	3
		3	3
5	28/10/2001	1	2
		2	1
6	29/10/2001	1	3
		2	2
		3	3
7	29/10/2001	1	6
		4	4
8	29/10/2001	2	2
		3	1
		4	3
9	29/10/2001	2	4
		3	3
10	29/10/2001	2	1
		4	1

CodItem	Descrição
1	Açúcar
2	Leite
3	Manteiga
4	Pão

Figura 33 – Base de transações objeto-relacional

Para o mesmo algoritmo, a versão relacional difere da objeto-relacional apenas no que diz respeito à consulta que busca os itens de uma transação. O resultado é então passado ao algoritmo que prosseguirá com a execução da mesma forma, independente da base utilizada. Desta forma, o exemplo aqui mostrado terá sempre os mesmos resultados em cada passo da execução independente de está sendo utilizada uma base relacional ou objeto-relacional.

A explicação da execução do algoritmo, descrito na figura 27, será dividida em fases, cada uma envolvendo um ou mais passos. Aqui será utilizado um suporte baixo para que grandes conjuntos possam servir para geração de candidatos na próxima fase. O valor do suporte será definido em 30%.

Importante ressaltar que a construção das árvores de intervalos será mostrada passo a passo. Entretanto, isto será feito através de figuras e não pelos resultados colhidos do *Oracle*.

Essa é só uma questão de comodidade, visto que a visualização da árvore a partir de uma consulta ao banco é de difícil compreensão já que a estrutura é simulada em linhas de tabelas e os apontadores são *ROWID's*. O mesmo será feito para a árvore de conjuntos.

Por fim, algumas telas mais importantes serão capturadas diretamente do SGBD com o objetivo de mostrar os resultados das fases e podem ser analisadas no Anexo 6.

FASE 1: O objetivo principal é encontrar grandes conjuntos de tamanho 1 (L_1)

- ***Gerar L_1: primeira passagem pelos dados***

Nesta parte, haverá a primeira passagem pelos dados, a qual irá pegar cada item individualmente e contar seu suporte. A tabela 8 ilustra os candidatos de tamanho 1:

Produto	Suporte
Açúcar	7
Leite	9
Manteiga	6
Pão	5

Tabela 8 – Candidatos de Tamanho 1

Como todos eles obedecem ao suporte mínimo, nenhum será descartado, formando os grandes conjuntos de tamanho 1.

- ***Criar Árvore de Grandes Conjuntos e Inserir L_1***

Nesta etapa, a árvore de grandes conjuntos é criada e terá inicialmente apenas o nó raiz, sem nenhum apontador para os nós filhos. Os grandes conjuntos encontrados na primeira passagem são então inseridos na estrutura, com seus respectivos suportes.

Em todas as fases o algoritmo trabalha com os códigos dos itens ao invés de suas descrições. Entretanto, por questão de comodidade, as estruturas serão demonstradas com este último atributo .

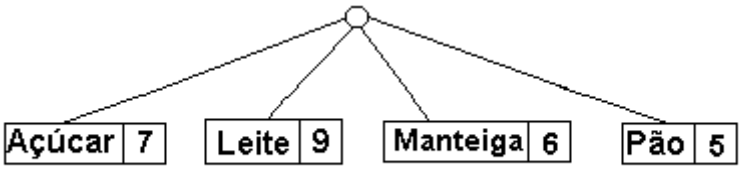


Figura 34 – Árvore com Grandes Conjuntos de Tamanho 1

FASE 2: O objetivo principal é encontrar L_2

- ***Gerar C_2***

A partir de L_1, os candidatos de tamanho 2 serão gerados. A tabela 9 mostra os candidatos de tamanho 2. O suporte ainda não foi contado, tendo, portanto, valor inicial 0:

Produto1	Produto2	Suporte
Açúcar	Leite	0
Açúcar	Manteiga	0
Açúcar	Pão	0
Leite	Manteiga	0
Leite	Pão	0
Manteiga	Pão	0

Tabela 9 – Candidatos de Tamanho 2

- ***Efetuar a poda de candidatos conforme o Apriori Quantitativo***

Aqui teoricamente a primeira poda seria efetuada. Entretanto, como todos os candidatos têm tamanho 2, necessariamente foram formados a partir de dois conjuntos de tamanho 1 que são, por sua vez, obrigatoriamente grandes. Desta forma, não existe subconjunto algum a ser analisado. Por outro lado, não existem árvores de intervalos para conjuntos de tamanho 1. Assim, não há nada a ser feito na poda. O algoritmo passa então para a próxima fase.

- ***Criar Árvore de Intervalos para os C_2 restantes***

As árvores de intervalos são montadas para cada candidato e serão atualizadas durante a passagem pelos dados para contagem do suporte. A montagem da árvore de intervalos do conjunto {Leite, Manteiga} será mostrada agora passo a passo. Esse conjunto foi escolhido porque é um dos que tem maior suporte, permitindo a criação de uma árvore mais profunda. Aqui o valor utilizado para o limite de um nó será 3.

De acordo com a base da tabela 7 e observando as transações na quais os itens Leite e Manteiga foram adquiridos juntos, percebe-se que os intervalos de quantidades adquiridas são respectivamente [1-4] e [1-3] para cada item, faixas estas mostradas no nó raiz, figura 35 (a).

A primeira transação que contém os itens juntos é a 1, cujas quantidades adquiridas foram 1 e 3, respectivamente. Aqui, as aquisições serão representadas por [1,3]. Sendo assim, o contador do nó raiz recebe o valor 1, conforme figura 35 (a).

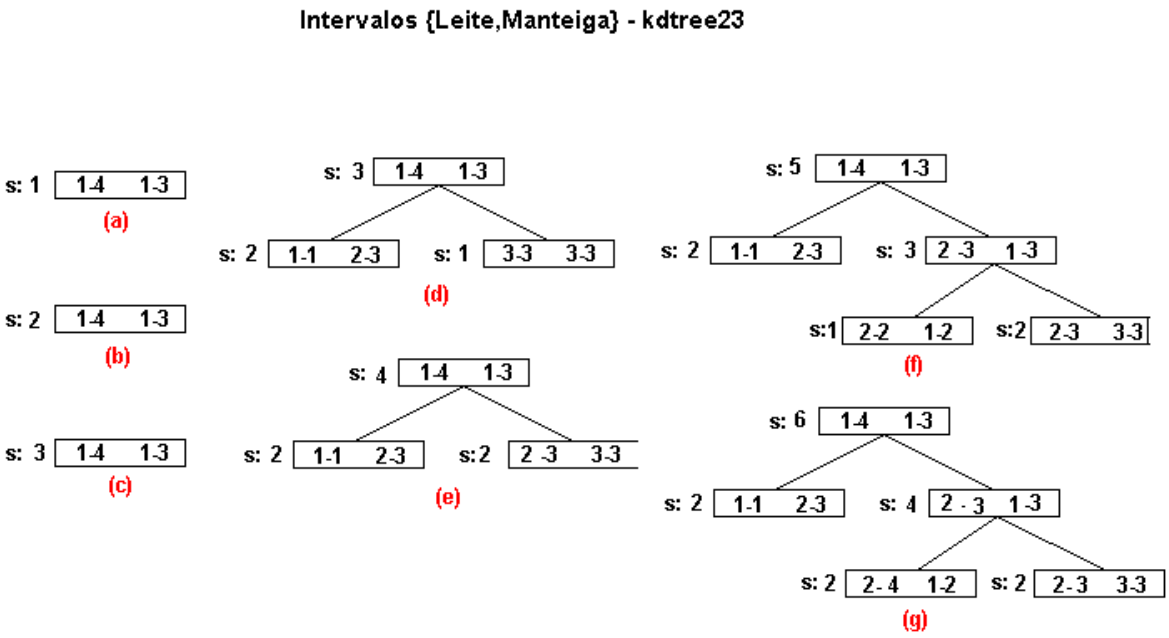


Figura 35 – Montagem da Árvore Intervalos do conjunto {Leite, Manteiga}

Para a transação 2, as quantidades de aquisição são 1 e 2. O contador da raiz é mais uma vez incrementado (figura 35(b)).

A próxima transação que inclui Leite e Manteiga juntos é a de número 4, com quantidade adquirida de 3 para os dois itens. A árvore fica conforme apresentado no item (c). Neste ponto, o limite da raiz foi atingido e o nó precisará ser dividido. Neste momento, é

necessário verificar o discriminante (chave de pesquisa) para definir os intervalos dos nós filhos. Neste caso, como se está no primeiro nível da árvore, o atributo chave é o primeiro (Leite) e a chave de pesquisa é um valor intermediário no intervalo de aquisições deste atributo. Tirando a media deste intervalo o valor seria 2,5. A rotina foi padronizada para sempre arredondar para menos o valor a chave. Assim, esta será igual a 2. No nó esquerdo devem ficar as transações cuja quantidade do atributo Leite foi menor que 2. Voltando à base a analisando as transações já inseridas na árvore ([1,3], [1,2] e [3,3]), percebe-se que o nó esquerdo receberá o valor 2 em seu contador. Além disso, a faixa para o atributo Leite terá o valor 1 tanto no atributo referente ao mínimo adquirido, quanto no máximo, já que a quantidade foi 1 nas duas transações que serão colocadas no nó esquerdo. Já para o atributo não chave, no caso Manteiga, a faixa deve ser definida em [2-3], valores mínimo e máximo destas duas transações. No nó direito, o contador deve receber o valor 1 (em apenas uma transação a quantidade de Leite foi maior do que 2) e as faixas de ambos os atributos terão 3 nos seus dois valores. Isto está representado na figura 35 (d).

A transação de código 6 tem valores 2 e 3 de aquisições. Começando pela raiz, o contador é incrementado e passa a valer 4. A chave é 2, como calculado anteriormente. Como o valor de Leite também vale dois, a árvore deve ser percorrida pelo filho direito da raiz. Assim, o contador desse nó e incrementado e passa para o valor 2. Neste ponto, é necessário verificar se as faixas devem ser alteradas. Para o atributo Leite, sua faixa até então continha o valor 3 nos dois extremos. Entretanto, na transação analisada, a quantidade adquirida foi igual a 2. Assim, o limite inferior passa a ser 2 e nada acontece com o superior. No caso do item Manteiga, nada precisa ser feito, visto que a quantidade adquirida foi 3, valor que já está dentro da faixa existente no nó. A nova estrutura da árvore está sendo exibida na figura 35 (e).

A próxima transação é a 8, com valores de aquisição de 2 e 1 para Leite e Manteiga, respectivamente. Mais uma vez, o contador do nó raiz é incrementado e vale agora 5. A chave no primeiro nível é sempre 2. Portanto, a árvore deve ser percorrida pela direita, visto que a quantidade adquirida na transação corrente é igual a dois. Chegando no filho direito, o contador é incrementado e passa a ser 3. É hora então de dividir esse nó. Entretanto, no segundo nível o discriminante é um valor intermediário na faixa do segundo atributo. Assim, a chave passa a ser dois e o atributo chave é Manteiga. No momento da divisão então, é necessário buscar a quantidade de transações cujo valor de Manteiga foi menor que dois e Leite estava entre 2 e 3, dentre as 3 transações que formaram este nó ([3,3],[2,3] e [2,1]). Neste caso, aparece uma transação e o contador do nó esquerdo (que acabou de ser criado)

para a ser 1. O direito recebe então o valor 2. As faixas são atualizadas de acordo com essas três transações. A estrutura esta sendo mostrada agora na figura 35 (f).

Por fim, a última transação 9 traz os valores de aquisição 4 e 3. Seguindo o mesmo raciocínio, o contador do nó raiz é incrementado e o caminhamento continua pelo nó direito, cujo contador também recebe mais um unidade em seu valor. A partir do nó corrente (filho direito da raiz) deve-se seguir pelo seu filho esquerdo, visto que a quantidade Manteiga adquirida na transação foi 1 e a chave no segundo nível é 2 ((1+3)/2). O contador da folha esquerda passa então a ser 2. A faixa do atributo Leite precisa ser atualizada, pois 4, quantidade adquirida na transação corrente, não está dentro do intervalo. Este passa a ser então [2-4]. A árvore terá a aparência mostrada na figura 35(g).

- ***Contar suporte de C_2: segunda passagem pelos dados***

Depois de contado o suporte, os candidatos ficam conforme a tabela 10. A essa altura as arvores de intervalos de todos eles já foram montadas.

Produto1	**Produto2**	**Suporte**
Açúcar	Leite	6
Açúcar	Manteiga	4
Açúcar	Pão	3
Leite	Manteiga	6
Leite	Pão	4
Manteiga	Pão	2

Tabela 10 – Candidatos de Tamanho 2 depois de contado o suporte

- ***Eliminar candidatos que não atingem o suporte mínimo***

Aqui os candidatos que não atingiram o suporte mínimo são podados. Considerando o suporte mínimo em 30%, a tabela de grandes conjuntos de tamanho 2 ficará com 5 elementos. O item {Manteiga, Pão} é descartado. A árvore de intervalos do referido conjunto pode então ser descartada também.

- ***Inserir L_2 na Árvore de Conjuntos***

Os grandes conjuntos de tamanho 2 são inseridos na árvore, como mostrado na rotina de inserção. A estrutura fica então da seguinte forma:

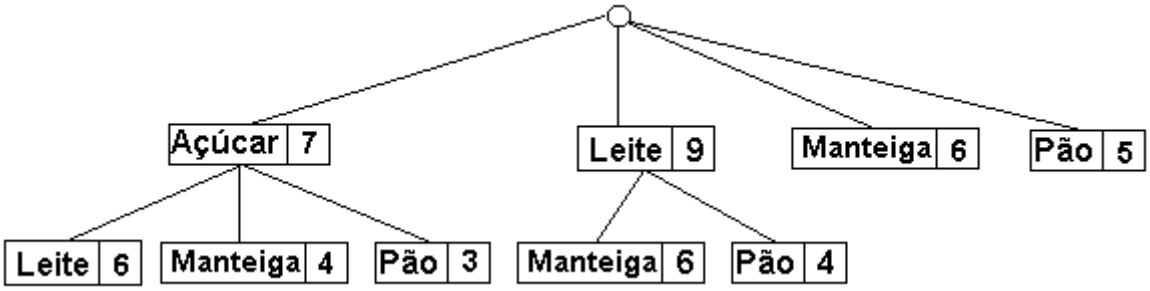


Figura 36 – Árvore com Grandes Conjuntos de Tamanho 2

FASE 3: O objetivo principal é encontrar L_3

- ***Gerar C_3***

A partir de L_2, os candidatos de tamanho 3 serão gerados, conforme tabela 11:

Produto1	Produto2	Produto3	Suporte
Açúcar	Leite	Manteiga	0
Açúcar	Leite	Pão	0
Açúcar	Manteiga	Pão	0
Leite	Manteiga	Pão	0

Tabela 11 – Candidatos de Tamanho 3

- ***Efetuar a poda dos candidatos conforme o Apriori Quantitativo***

Aqui a poda é realizada de fato. Inicialmente os candidatos que possuem algum subconjunto que não é grande são eliminados, todos sombreados na tabela 12:

Produto1	Produto2	Produto3	Suporte
Açúcar	Leite	Manteiga	0
Açúcar	Leite	Pão	0
Açúcar	Manteiga	Pão	0
Leite	Manteiga	Pão	0

Tabela 12 – Candidatos de Tamanho 3 após a primeira poda

O conjunto {Açúcar, Manteiga, Pão} é descartado porque {Manteiga, Pão} não é grande. O mesmo ocorre para {Leite, Manteiga, Pão}.

A seguir é efetuada a poda dos candidatos restantes com base nas árvores de intervalos.

O candidato sombreado é descartado. Isto vai acontecer por causa do critério de poda proposto pelo algoritmo, conforme explicação abaixo:

Produto1	Produto2	Produto3	Suporte
Açúcar	Leite	Manteiga	0
Açúcar	Leite	Pão	0

Tabela 13 – Candidatos de Tamanho 3 após a segunda poda

Para analisar se o candidato {Açúcar, Leite, Pão} pode ser podado antes mesmo de ter seu suporte contado, o *Apriori* Quantitativo irá verificar as árvores de intervalos de todos os seus subconjuntos para testar a sobreposição dos intervalos. A figura 37 traz então as árvores de intervalos dos conjuntos {Açúcar, Leite}, {Açúcar, Pão} e {Leite, Pão}.

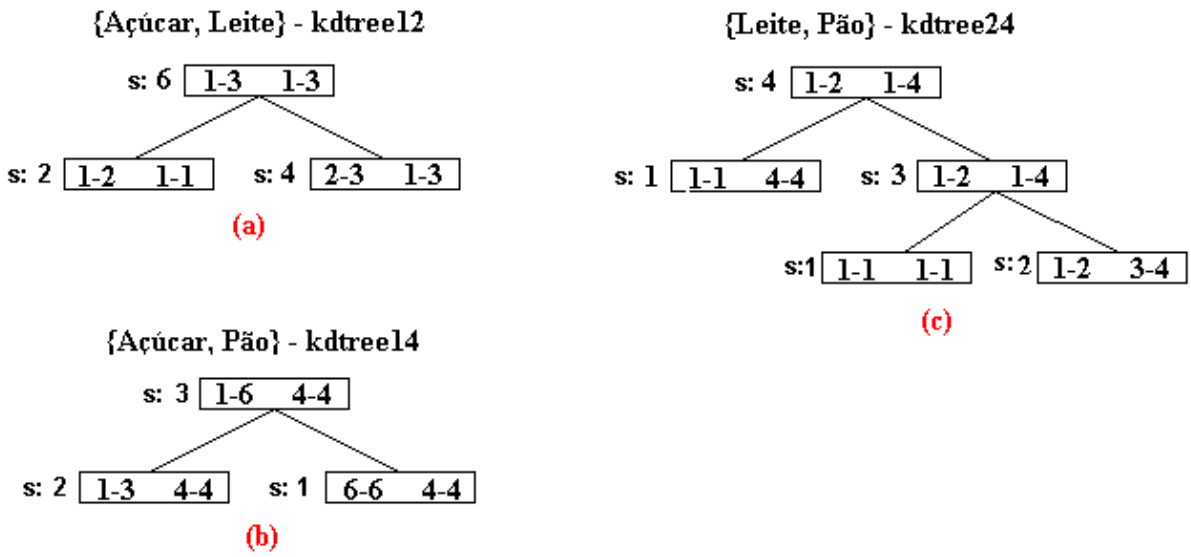


Figura 37 – Árvores de Intervalos dos conjuntos de tamanho 2

A análise começa a partir do subconjunto com menor suporte (chamado anteriormente de *pmin*). Neste caso, *pmin* = {Açúcar, Pão}. Então, para cada folha de *pmin*, começando mais uma vez pela de menor suporte, testa-se a sobreposição de intervalos. É fácil perceber que ao analisar a folha referente às faixas {6-6 e 4-4} na árvore de {Açúcar, Pão} e testar a sobreposição desta folha com aquela com as faixas {2-3 e 1-3} na árvore de {Açúcar, Leite}, estas folhas não se sobrepõem. Isto porque o atributo de teste é Açúcar (já que é o que existe nas duas estruturas). Assim [6-6] não se sobrepõe a [2-3]. Além disso, [6-6] também não se sobrepõe à outra folha esquerda da figura 34 (a). Ë fácil perceber pela base de dados que a transação cujo valor de Açúcar = 6, contém o atributo Pão, mas não contém Leite. Desta forma, o suporte global de {Açúcar, Pão} é diminuído do suporte da folha que acabou de ser testada, cujo valor é 1. {Açúcar, Pão} passa a ter suporte = 2 e o superconjunto {Açúcar, Leite, Pão} é descartado.

Aqui foi mostrado apenas um exemplo de teste de sobreposição. Entretanto, cabe ressaltar que isto deve ser feito para todas as folhas de *pmin* em relação a todas as outras folhas dos demais subconjuntos, que demonstra a complexidade do processo.

- ***Criar Árvore de Intervalos para os C_3 restantes***

As árvores de intervalos são montadas para cada candidato e serão atualizadas durante a passagem pelos dados para contagem do suporte.

- ***Contar suporte de C_3: terceira passagem pelos dados***

Depois de contado o suporte, os candidatos ficam como mostra a tabela 14:

Produto1	Produto2	Produto3	Suporte
Açúcar	Leite	Manteiga	4

Tabela 14 – Candidatos de Tamanho 3 depois de contado o suporte

- ***Eliminar candidatos que não atingem o suporte mínimo***

Aqui os candidatos que não atingiram o suporte mínimo devem ser podados. Entretanto, é fácil perceber que o único candidato existente não será descartado.

- ***Inserir L_3 na Árvore de Conjunto***

Os grandes conjuntos de tamanho 3 são inseridos na árvore:

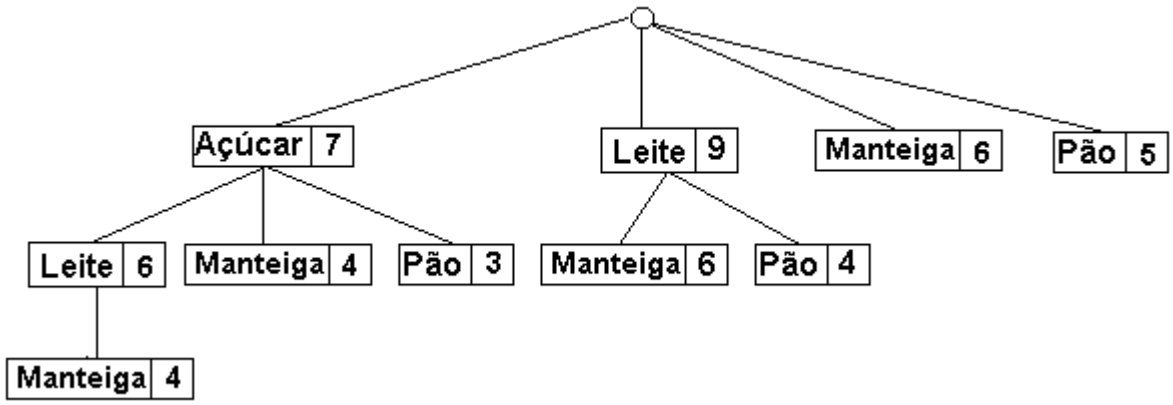


Figura 38 – Árvore com Grandes Conjuntos de Tamanho 3

FASE 4: Gerar as Regras

Como só restou um conjunto de tamanho 3, não existe mais possibilidade de estender este grande conjunto de maneira a gerar um candidato com 4 elementos. Sendo assim, o algoritmo entra na sua última fase: a geração das regras. Neste ponto, a confiança é levada em consideração. Usando uma confiança de 70%, o algoritmo irá retornar as sombreadas na tabela 15:

Regra	Fator de Confiança
{Açúcar, Leite}	0.86
{Leite, Açúcar}	0.67
{Açúcar, Manteiga}	0.57
{Manteiga, Açúcar}	0.67
{Açúcar, Pão}	0.43
{Pão, Açúcar}	0.60
{Leite} ➔ {Manteiga}	0.67
{Manteiga} ➔ {Leite}	1
{Leite} ➔ {Pão}	0.44
{Pão} ➔ {Leite}	0.80
{Açúcar, Leite} ➔ {Manteiga}	0.67
{Manteiga} ➔ {Açúcar, Leite}	0.67
{Açúcar} ➔ {Leite, Manteiga}	0.57
{Leite, Manteiga}➔ {Açúcar}	0.67
{Açúcar, Manteiga}➔ {Leite}	1
{Leite} ➔ {Açúcar, Manteiga}	0.44

Tabela 15 – Regras geradas pelo *Apriori* Quantitativo

4.7. Conclusões do Capítulo

Os SGBD's objeto-relacionais aceitam a modelagem de coleções acabando com as restrições relacionais. Assim, *nested tables* e *varrays* podem ser usados, como mostrado neste capítulo. Além disso, uma outra grande vantagem é a possibilidade de criação de "pacotes" para encapsular as extensões de tipos e serviços. No caso do *Oracle*, os novos tipos podem ser encapsulados com a ajuda dos Cartuchos de Dados ("*Data Cartridges*"), disponibilizados a partir da versão 8i [10]. Onde algum aspecto de um serviço nativo não é adequado para o processamento especializado desejado, pode-se prover implementação própria desses serviços. Exemplificando, se é necessário construir um cartucho para manipulação de imagens, devem-se implementar as rotinas que criam e mantêm índices para esse tipo de estrutura. O servidor irá então invocar essa implementação toda vez que for necessária alguma operação nesses dados. Neste caso, o serviço de indexação do servidor foi estendido.

As árvores de intervalos utilizadas pelo *Apriori* Quantitativo de certa forma podem ser encaradas como um tipo de índice não nativo que é utilizado pelo algoritmo de mineração. Se tudo for seguido à risca, as rotinas para criar, inserir, atualizar e executar qualquer operação no índice devem ser escritas e encapsuladas em um cartucho. Além disso, a implementação deve ser registrada no *Oracle* usando a interface de extensão de índice (ODCIIndexInterface), que já indica quais rotinas devem ser escritas (ODCIIndexCreate, ODCIIndexInsert, etc) inclusive com parâmetros e tipos pré-definidos. Depois de definidas e registradas as rotinas, um novo "*indextype*" foi criado e pode ser aplicado a um atributo qualquer de uma determinada tabela. Se tudo for feito desta forma, o servidor irá invocar automaticamente esta implementação toda vez que forem necessárias operações no novo tipo de índice. Por exemplo, quando da execução de um comando de inserção de dados em uma tabela, o próprio servidor irá invocar a rotina ODCIIndexInsert, que indicará as regras da inserção no índice. Isso é extensibilidade na sua forma mais pura.

O fato da criação das rotinas de manipulação de *kdtrees* no *Oracle*, desde que elas seguissem as *regras* impostas pela interface de extensão de índices, permitiria que um novo "*indextype*", qual seja, *kd-tree* pudesse ser utilizado. Entretanto, ao se utilizar esse novo tipo, um índice do tipo *kdtree* seria gerado para um determinado atributo de tabela. Portanto, uma única *kd-tree* seria gerada e atualizada à medida que as inserções na referida tabela fossem feitas. Porém, o que se faz necessário para o *Apriori* Quantitativo é a criação de várias árvores. Por tudo isso, percebe-se que as árvores de intervalos foram implementadas no

Oracle, porém sem o uso das rotinas padrão para criação de índices. A solução encontrada neste trabalho para que isto pudesse ser feito foi justamente a de simular a estrutura através de tabelas. Dessa forma, várias *kd-trees* são geradas, montadas e consultadas explicitamente pelo algoritmo de mineração.

Quanto à construção do cartucho, cabe lembrar que o principal objetivo do *Data Cartridge* é encapsular algum serviço do banco que foi estendido, de forma a disponibilizar tal extensão em um componente que possa ser instalado facilmente mais tarde. Entretanto, nada impede que serviços sejam estendidos e não necessariamente encapsulados em um cartucho.

Por fim, vale enfatizar que a simulação das árvores de conjuntos e de intervalo em tabelas dinâmicas exige muito esforço para manter e atualizar essas estruturas. Apesar do otimizador de consultas trabalhar de forma a deixar a manipulação das tabelas o mais rápida possível, esse custo de manutenção pode contrabalançar as vantagens intrínsecas da integração do algoritmo com o *Oracle*.

Capítulo 5

Avaliação Experimental do Algoritmo *Apriori* Quantitativo Integrado ao SGBDOR *Oracle9i*

O objetivo deste capítulo é descrever e avaliar os testes feitos com o algoritmo *Apriori* Quantitativo integrado ao *Oracle* 9i. Para fins de comparação de desempenho, desenvolvemos também uma versão integrada ao *Oracle9i* do algoritmo *Apriori* básico, diferindo apenas no fato de que nesta última não existem as árvores de intervalos, que são utilizadas somente pelo *Apriori* Quantitativo na sua fase de poda.

Os experimentos foram planejados visando dois objetivos: (1) avaliar em que medida a noção de árvores de intervalos pode melhorar o desempenho do *Apriori* Quantitativo em relação ao *Apriori* básico, ambos integrados ao *Oracle9i* — isto não é de modo algum claro, haja visto o grande overhead ocasionado pela construção dinâmica das árvores de intervalo (seção 4.3, capítulo 4); e (2) comparar as versões relacional e objeto-relacional do *Apriori* Quantitativo — uma contribuição à avaliação da nascente tecnologia objeto-relacional, assunto ainda muito escasso ou quase inexistente na literatura científica especializada.

Inicialmente serão feitas considerações sobre as bases de dados utilizadas na avaliação experimental. Questões relativas à otimização de consultas no *Oracle* são também levantadas. O capítulo termina com os resultados e análise crítica dos experimentos realizados para o cumprimento dos objetivos fixados.

5.1. Ambiente de Testes

Os testes dos algoritmos foram realizados em um *desktop* Pentium II, 256 Mb de memória RAM, com o sistema operacional *Windows 2000 Server* e o SGBDOR *Personal Oracle9i.*

Como os resultados analisados são tempos de execução dos algoritmos, foi importante garantir que todos os testes fossem feitos nas mesmas condições ambientais, para permitir a

comparação dos resultados. Desta forma, ao fim de cada teste e antes do próximo, o sistema era completamente reiniciado, reproduzindo as mesmas condições iniciais.

5.2. Geração das Bases de Dados Utilizadas

Para perceber as diferenças entre os algoritmos, principalmente em se tratando de tempo de execução, foi necessária uma quantidade substancial de dados. Também foi de suma importância a utilização, para a mineração, de bases de dados realistas.

A técnica de geração de regras de associação tem sido muito utilizada em aplicações de comércio varejista como supermercados.

Diante da dificuldade de encontrar uma base real para o tema de comércio varejista escolhido, criamos um gerador automático de transações de vendas em um supermercado hipotético. Para isto, um procedimento PL/SQL foi desenvolvido, recebendo três parâmetros de entrada: número máximo de transações (*MaxTrans*), número máximo de itens por transação (*MaxItens*) e valor máximo para a quantidade adquirida de cada item (*MaxQte*).

O procedimento gera então *MaxTrans* transações na base de dados. Cada transação tem um número aleatório de itens (de 1 a *MaxItens*) e, para cada item, a quantidade supostamente adquirida também é gerada aleatoriamente (de 1 a *MaxQte*). Uma tabela auxiliar (*TabItem*), valendo para as duas versões do *Apriori* Quantitativo, contém apenas o código e a descrição dos 'produtos'. A partir das transações e da tabela auxiliar, são instanciadas as tabelas *Item_r* e *Item_OR*, descritas na seção 4.5 do capítulo 4, correspondendo respectivamente às versões relacional e objeto-relacional do *Apriori* Quantitativo.

A quantidade média de itens por transação foi propositalmente pequena para evitar uma explosão de regras, um problema comum aos algoritmos de regras de associação.

O código completo da rotina de geração dos dados pode ser encontrado no Anexo 5 deste trabalho.

5.3. Recursos de Otimização do *Oracle9i*

O primeiro tópico sobre otimização aqui discutido é como os objetos, sejam eles tabelas com os dados de entrada ou aquelas utilizadas para simulação das estruturas usadas pelos algoritmos, foram armazenados.

Também analisamos nesta seção o otimizador de consultas do *Oracle*, abordando suas possíveis formas de trabalhar e o que fizemos para que as consultas fossem realizadas de forma eficiente.

5.3.1. Armazenamento dos Objetos

O *Oracle* armazena os objetos logicamente em *tablespaces* e fisicamente em arquivos de dados ("*datafiles*") associados com o correspondente *tablespace*. Sendo assim, um banco de dados *Oracle* é uma coleção de *tablespaces*, consistindo de um ou mais arquivos de dados. Um *tablespace* especial denominado *system tablespace* guarda o dicionário de dados do banco de dados e os índices de acesso ao dicionário. Um ou mais *tablespaces de usuários* completam o banco.

Os objetos relativos às bases de transações foram todos criados em um *tablespace* especialmente definido para este propósito, inclusive em outra unidade de disco distinta daquela onde está o *system tablespace*. O uso de dois discos é útil pois, no momento em que o SGBD precisar consultar o *tablespace* do sistema e o *tablespace* com os dados do usuário, dois canais de entrada e saída poderão ser aproveitados.

Com relação aos objetos manipulados dinamicamente pelos algoritmos (árvore de conjuntos, tabelas de candidatos, árvores de intervalos, etc), a idéia inicial era que todos eles fossem criados dentro de um *tablespace* temporário, um tipo especial de *tablespace* que só sobrevive durante a execução do problema. Entretanto, há várias restrições ao uso de *tablespaces* temporários. A primeira delas é que um *tablespace* temporário só pode armazenar tabelas temporárias. Também, uma tabela temporária não pode conter colunas do tipo *varray*, que, como vimos no capítulo 4, é muito usado na definição das estruturas de suporte ao algoritmo. A solução que encontramos foi armazenar os objetos criados dinamicamente em um *tablespace* não temporário, sendo programaticamente removido no término da execução.

Muito pouco é comentado nos manuais do *Oracle* a respeito de possíveis formas de armazenamento de *nested tables*, com o objetivo de melhorar o desempenho das consultas a coleções. Sobressai-se apenas a forma denominada *Index Organized Table* (IOT). O argumento para o uso de IOT é que esta forma permite agrupar fisicamente os elementos de uma coleção, como, por exemplo, agrupar fisicamente os itens de uma transação de venda.

Testamos na prática a organização IOT, e constatamos que ela realmente causa grande melhoria no desempenho das consultas.

Um segundo ponto é a questão da economia de espaço para o armazenamento das *nested tables* (no nosso caso, uma para cada transação de venda). A especificação *Organization Index Compress* para a organização IOT permitiu-nos uma grande economia de espaço de armazenamento das transações de venda.

Por fim, cabe ressaltar que as bases de transações são apenas consultadas no decorrer da execução dos algoritmos, dispensando, portanto, cuidados relativos ao desempenho de operações de atualização.

5.3.2. O Otimizador de Consultas do *Oracle*

Vários fatores podem influenciar o desempenho de um SGBD. Isto vai desde ajustes nas aplicações, nos próprios bancos de dados e, até mesmo, no sistema operacional e no hardware. A forma de ajustar um SGBD é denominada *sintonia* ("*tuning*") e, se feita de maneira apropriada, propicia um melhor desempenho para as aplicações.

Nesta seção, discutimos como sintonizamos nossa aplicação (implementações do algoritmo *Apriori* Quantitativo) com o *Oracle*.

Dada uma consulta a uma ou várias tabelas, existem inúmeras maneiras de resolvê-la, ou vários *planos de execução*. A razão para isto é que há diversos *métodos de acesso* tanto diretamente a tabelas, por exemplo, varrendo a tabela inteira, ou indiretamente através de índices. A combinação dos passos escolhidos para executar um comando SQL é então chamada um *plano de execução.* Isto inclui um *método de acesso* escolhido para cada tabela acessada.

A seleção dos métodos de acesso, ou do 'melhor' plano de execução, fica a cargo do módulo *Otimizador de Consultas* do *Oracle*, como de resto de qualquer SGBD.

Infelizmente, o Otimizador de Consultas não é perfeito, podendo não gerar sozinho um plano que satisfaça os requisitos da aplicação. A solução então é permitir ao programador, que afinal de contas detém muita informação sobre os requisitos da aplicação e sobre a própria aplicação, orientar o Otimizador na escolha do plano, através do mecanismo chamado de *hint* (instruções explícitas do programador que direcionam o Otimizador na escolha dos métodos de acesso).

O tipo de otimização pode ser definido de maneira geral em: baseado em custos e baseado em regras. No primeiro, o *Oracle* irá estimar, para cada consulta, o seu custo de execução, fazendo uso de possíveis estatísticas geradas para os objetos. Para o segundo tipo, o SGBD já tem um plano de execução pré-definido que será executado de imediato. O critério de otimização pode ser definido através de parâmetros ou diretamente na consulta, através do uso de *hints*.

O parâmetro OPTIMIZER_MODE pode assumir os seguintes valores [23]:

– CHOOSE: este é o valor padrão do parâmetro. Neste caso, o otimizador baseia-se em estatísticas (tamanho, número de linhas, etc), caso existam para ao menos uma das tabelas. A meta é processar o maior número de serviços em um determinado intervalo de tempo. Quando não houver estatísticas para a otimização baseada em custos, o Otimizador tenta estimar algumas estatísticas.

– ALL_ROWS: abordagem baseada em custos, com o objetivo de obter o maior numero de serviços processados por unidade de tempo, independentemente da existência de estatísticas;

– FIRST_ROWS: também baseada em custos, procura diminuir o tempo de resposta para todos os comandos de uma seção e independe da existência de estatísticas;

– RULE: abordagem baseada em regras independentemente da existência ou não de estatísticas;

De maneira geral, a *Oracle* recomenda que se utilize otimização baseada em custos. Segundo [23], este tipo de otimização está em processo constante de melhoria e a otimização baseada em regras continua existindo ainda por questões de compatibilidade com sistemas legados. Ao mesmo tempo, o uso de *IOT's*, feito neste trabalho, requer a otimização baseada em custos. Desta forma, ficou claro que este é o tipo de otimização usado. Como ele leva em consideração as estatísticas e, caso não existam, o otimizador baseia-se apenas em estimativas, também foram geradas estatísticas para todos os objetos utilizados pelos algoritmos.

Mesmo levando em consideração que o tipo de otimização usado seria o baseado em custos, por causa do uso de IOT's, vários testes foram realizados em relação às possíveis formas de otimização das consultas.

Inicialmente foram analisadas, através do comando "*EXPLAIN PLAN*", que tem como objetivo mostrar todo o plano de execução usado pelo otimizador, as formas de execução das principais consultas realizadas pelos algoritmos. Percebeu-se claramente que estava sendo

utilizado principalmente acesso através do *ROWID* e através de índices. Isso é facilmente justificado: ora, as consultas dos algoritmos geralmente recuperam apenas uma linha de uma determinada transação para análise e os '*apontadores*' nas árvores foram sempre simulados pelo identificador da linha. Desta forma, a impressão inicial foi a de que o otimizador já estava trabalhando de forma eficiente.

O segundo passo foi forçar o otimizador a usar determinado método de otimização, o que pode ser através da alteração do OPTIMIZER_MODE ou com o uso dos *hints*. A primeira idéia era ir trocando os valores do parâmetro para analisar o comportamento do *Oracle*. Entretanto, após várias pesquisas, foi constatado que a alteração desse parâmetro através do comando "ALTER SESSION SET OPTIMIZER_MODE = valor" só afeta as consultas efetuadas diretamente em comandos SQL, ou seja, as que não estão embutidas em trechos de programas. Assim, tudo o que estava embutido no código dos algoritmos de mineração permanecia com o critério de otimização inalterado. Desta forma, a solução foi o uso de "*hints*" diretamente nos comandos dentro dos algoritmos para forçar a execução nos moldes desejados, fazendo uso do tipo de otimização escolhido.

Neste ponto, duas coisas ficaram realmente claras: a implementação que utiliza base relacional sofreu poucas variações em termos de desempenho. Isso já era esperado porque as consultas utilizadas são bem simples e geralmente visam recuperar apenas uma linha, fazendo uso de índices. Já a implementação objeto-relacional diminuiu um pouco seu tempo de execução logo após a criação das estatísticas. Isso também é plenamente justificado: antes das estatísticas o otimizador estava sendo executado no modo baseado em custos, já que isto é exigência quando se trata de *Nested Tables* e, ao mesmo tempo, baseava-se apenas em estimativas dos objetos.

Desta forma, a conclusão relativa à questão do otimizador de consultas foi que a criação de estatísticas dos objetos é de fundamental importância para "guiar" o otimizador em suas escolhas, mesmo porque a *Oracle* confirma que cada vez mais se está investindo no otimizador baseado em custos.

5.4. Análise dos Resultados

Esta seção visa analisar os resultados do trabalho através dos tempos de execução dos algoritmos. Convém frisar que os testes visam avaliar dois pontos distintos: primeiro, em que medida a proposta da fase de poda do *Apriori* Quantitativo, através das árvores de intervalos

pode melhorar o desempenho deste algoritmo em relação ao *Apriori* básico, ambos integrados ao *Oracle9i*; segundo, avaliar o comportamento de cada um dos algoritmos sobre bases relacional e objeto-relacional.

A tabela 16 é uma síntese de todas as tomadas de tempo:

Incremento (Vendas)	**Produtos Adquiridos***	**Suporte**	**Desempenho Relacional (s)**		**Desempenho ObjetoRelacional (s)**	
			Apriori	*Apriori* Quant	*Apriori*	*Apriori* Quant
500	1519	20%	61.79	217.89	63.43	243.93
		30%	53.32	198.22	55.48	219.34
		40%	40.56	155.80	43.83	162.12
1000	3099	20%	111.78	387.37	122.57	452.36
		30%	99.88	347.03	113.43	404.73
		40%	77.85	276.15	88.59	316.44
2000	6060	20%	227.59	726.85	263.29	886.66
		30%	206.84	665.70	233.88	769.40
		40%	156.64	526.57	172.91	603.45
4000	12115	20%	467.59	1465.67	566.00	1740.85
		30%	409.10	1316.79	466.12	1548.79
		40%	307.17	1042.53	355.52	1191.25
8000	24108	20%	932.81	2869.44	1118.86	3496.00
		30%	802.84	2606.97	919.63	3075.30
		40%	621.36	2069.39	703.53	2374.65

* - medida referente a itens vendidos, independente das quantidades de aquisição.

Tabela 16 – Tempos de Execução dos Algoritmos

A primeira coluna denominada *Incremento* indica a quantidade de transações da base utilizada. Desta forma, a primeira linha representa uma base com 500 transações diferentes, sendo que cada uma tem itens adquiridos e respectivas quantidades. Já a segunda coluna traz o número total de itens adquiridos para todas as transações. Este valor não tem nada a ver com o atributo quantidade, utilizado pelo *Apriori* Quantitativo. Exemplificando, seja uma base com apenas duas transações. Neste caso, o valor da primeira coluna da tabela 16 seria 2. Considere ainda que na primeira transação o cliente comprou 1 litro de leite, 6 pães e 2

pacotes de café, ou seja, 3 produtos distintos. Na segunda, 2 litros de leite e 10 pães. O número de produtos diferentes adquiridos foi 5: 3 da primeira transação e 2 da segunda. Este seria o valor armazenado na segunda coluna. Assim, analisando a primeira linha da tabela 16, percebe-se que para esta base existem 500 transações de venda cuja soma total de produtos foi 1519.

Vale lembrar que o gerador de transações recebe como parâmetros a quantidade de transações e o número máximo de itens por transações. Este último valor foi 5 em todos os casos. Voltando à tabela 16, percebe-se que a quantidade *média* de itens por transação em cada uma das bases ficou em torno de 3. Por exemplo, na base com 500 transações cerca de 1500 produtos foram comprados, o que dá uma média de 3 por transação. Da mesma forma, para 1000 vendas, o total de produtos foi 3099. O fato de essa média ser bem próxima é de extrema importância, pois se garante que as bases foram crescendo de maneira uniforme, ou seja, apenas o número de transações aumenta, mantendo-se fixa a média de produtos por transação.

Para cada valor de suporte, mostrado na terceira coluna da tabela 16, quatro medidas em segundos foram tomadas: duas para o *Apriori* simples (base relacional e objeto-relacional) e outras duas para o *Apriori* Quantitativo, sobre as mesmas duas bases.

A utilização de três valores de suporte distintos, como aparece na tabela 16, foi importante porque o valor do suporte mínimo aceitável tem extrema influência no desempenho dos algoritmos, visto que quanto menor o suporte, maior o número de candidatos de uma fase para outra. A confiança foi mantida constante, de maneira a evitar que muitas regras fossem filtradas apenas por conta deste valor. Assim, em todos os casos, a confiança utilizada foi de 30%, permitindo a geração de várias regras. Os valores de suporte escolhidos foram 20%, 30% e 40% porque, em todos os casos, um suporte mínimo maior que 40% permitiu que pouquíssimas regras fossem geradas.

Em todos os casos, os valores de suporte de 20%, 30% e 40% geraram 152, 80 e 20 regras, respectivamente. Isto parece uma coincidência estranha, mas pode ser explicado: as bases foram geradas nos mesmos moldes, aumentando somente o parâmetro número de transações. Além disso, o número de itens escolhido foi pequeno. Desta forma, a tendência é que a freqüência de itens também cresça de forma uniforme, causando poucas alterações nas medidas de suporte. Cabe ressaltar que as regras geradas não foram idênticas, mas apenas com suporte e confiança aproximados. Era comum o caso de, em uma determinada base, uma regra do tipo {Pão, Leite} tivesse suporte igual a 42% e, em outra, a mesma regra apresentava

uma freqüência de 43%. Essa regularidade na geração das regras foi muito importante, visto que oscilações apenas do volume de transações (excluindo quantidade de regras geradas) permitiram comparações eficientes acerca deste parâmetro.

À primeira vista, os resultados da tabela 16 são totalmente surpreendentes. Percebe-se claramente que o *Apriori* básico é sempre melhor que o *Apriori* Quantitativo. Além disso, que a tecnologia relacional comportou-se melhor que a objeto-relacional. Cada um dos aspectos de comparação será melhor abordado a seguir.

Um dos pontos analisados na tomada de tempo dos algoritmos foi a quantidade de conjuntos candidatos que passavam de uma fase para outra. Era importante perceber se o *Apriori* Quantitativo estava realmente cumprindo seu papel de podar candidatos antes mesmo de terem seus suportes contados. Aqui, chegou-se a um ponto crucial deste trabalho de pesquisa: apenas na pequena base de dados trazida em [4] e inicialmente utilizada como teste, o algoritmo conseguiu podar candidatos através das informações quantitativas. A partir do momento em que foram geradas bases relativamente grandes e que cresciam cada vez mais como entrada dos algoritmos, nenhum candidato foi descartado pelo critério de otimização do *Apriori* Quantitativo. Como se pode perceber, visto que nenhum corte de candidato foi feito, o algoritmo montou toda uma infraestrutura de otimização que acabou sendo inútil e acabou onerando os custos de execução.

5.4.1. Análise Crítica *Apriori* (Ap) x *Apriori* Quantitativo (ApQ)

A primeira análise a ser feita é um comparativo do desempenho do *Apriori* Quantitativo em relação ao *Apriori* original sobre a mesma base de dados, independente do tamanho desta base e de detalhes de modelagem, conforme discutido nos tópicos a seguir. Antes serão feitas considerações sobre a freqüência de poda antecipada de um candidato com auxílio das árvores de intervalos.

5.4.1.1. Percentual de Tempo Consumido com a Contagem de Suporte

A montagem e manutenção das árvores de intervalos é uma tarefa que consome, além de muito espaço, posto que cada uma delas acaba transformando-se em uma tabela do banco

de dados, muito tempo, já que as atualizações das referidas estruturas devem considerar várias questões como ajustes de faixas de valores, divisões de nós, entre outros fatores, conforme descrito no capítulo 4, seção 4.3. Por outro lado, as pesquisas indicam que a parte mais onerosa da família de algoritmos de geração de regras de associação é realmente a contagem do suporte para os vários candidatos. A tabela 17 ilustra o tempo gasto para a contagem do suporte dos candidatos. Daqui para frente, a terminação "_R" estará sempre relacionada a base relacional e o "_OR" refere-se a um exemplo com dados com extensões objeto-relacionais. Assim, Ap_R indica que foi utilizado o *Apriori* clássico com base relacional. Já ApQ_OR ilustra o uso do *Apriori* Quantitativo e base Objeto-Relacional.

	Tempo Total da Rotina(s)	**C_1(s)**	**C_2(s)**	**C_3(s)**	**C_4(s)**	**Tempo Total Contagem(s)**	**% Tempo Contagem**
Ap_R	53,74	3,61	16,26	18,00	11,38	49,25	92,0
ApQ_R	205,47	3,68	76,10	74,69	42,81	197,28	96,0

Tabela 17 – Tempo Gasto para Contagem do Suporte

A primeira coluna da tabela 17 mostra o algoritmo utilizado (*Apriori* ou *Apriori* Quantitativo). A segunda coluna mostra o tempo total de execução da rotina. As próximas 4 colunas mostram o tempo gasto para a contagem do suporte dos candidatos de tamanho 1 a 4, respectivamente. A penúltima coluna ilustra todo o tempo consumido com contagem do suporte. Por fim, a tabela 17 traz o percentual do tempo total da rotina consumido com esta tarefa.

Tomando como exemplo uma base qualquer de 500 transações e 1548 itens vendidos, o custo de manutenção dos candidatos para cada um dos algoritmos foi analisado, conforme tabela 17. Os parâmetros utilizados foram Suporte = 30% e Confiança = 80%. Com esses valores, cada um dos algoritmos encontrou 8 regras e o número de candidatos foi 5, 10, 10, 5, para os tamanhos de 1 a 4, respectivamente. Todos os tempos estão mostrados em segundos.

A análise da tabela 17 permite chegar a várias conclusões importantes: inicialmente, como já era de se esperar, mais de 90% de todo o tempo gasto pelos algoritmos de mineração diz respeito à contagem do suporte dos conjuntos candidatos. Para os candidatos de tamanho 1, a geração e a contagem do suporte são tarefas feitas em paralelo e não há diferença entre os algoritmos, visto que, neste momento, as árvores de intervalos ainda não existem. Isto ficou

claro nas medidas da tabela 17, já que o tempo gasto com esta tarefa foi praticamente o mesmo em ambos os casos. Partindo para os conjuntos de tamanho 2, cuja freqüência foi de 10 candidatos, verifica-se que, no caso do *Apriori*, o custo médio de contagem por candidato foi de aproximadamente 1,63 segundo. Já para o Quantitativo, esse valor subiu para 7,61 segundos. Para os candidatos de tamanho 3, que também apareceram com uma freqüência de 10, os valores foram bem próximos dos que acabaram de ser citados. Um total de 5 candidatos foi criado com tamanho 4. Neste ponto, a média de contagem do suporte para o *Apriori* foi de 2,28 segundos, enquanto para o Quantitativo foi de 8,56. Este aumento do valor médio era de se esperar, visto que, quanto maior o tamanho do candidato, maior o número de subconjuntos a serem checados, tarefa feita pelas duas versões do algoritmo e, conseqüentemente, cresce também o número de árvores de intervalos analisadas.

O restante do tempo foi gasto com tarefas menores como geração dos candidatos através de *joins* e geração das regras, tarefas comuns às duas versões do algoritmo.

Fica então comprovado que o custo de manutenção (geração e contagem do suporte) dos candidatos na versão quantitativa é muito maior que na original do *Apriori*. Desta forma, para que o algoritmo na sua proposta de otimização através das informações quantitativas pudesse realmente apresentar melhores resultados, o número de candidatos podados teria que ser extremamente grande, fato que não ocorreu em todos os casos testados.

5.4.1.2. Apriori Relacional x Apriori Quantitativo Relacional

Depois de feitas comparações sobre a manutenção dos candidatos para cada um dos algoritmos, o próximo passo foi analisar as diferenças em termos de desempenho entre as duas versões do *Apriori* sobre as bases de dados puramente relacionais.

Analisando os dados da tabela 16, percebe-se que em todos os exemplos a execução do *Apriori* Quantitativo é cerca de 3 vezes mais lenta em relação ao *Apriori* Original. A tabela 18 resume os resultados. Cada coluna representa uma base de dados utilizada, com o número de transações variando de 500 a 8000, assim como na tabela 16. Para cada uma das bases e para cada suporte é mostrado o fator de aumento de tempo do Quantitativo em relação ao *Apriori* clássico. Exemplificando, para a base relacional com 500 transações e suporte de 20%, o tempo de execução do *Apriori* Quantitativo relacional foi 3,53 vezes maior que o do *Apriori* clássico.

Incremento	500			1000			2000			4000			8000		
Suporte(%)	**20**	**30**	**40**	**20**	**30**	**40**	**20**	**30**	**40**	**20**	**30**	**40**	**20**	**30**	**40**
Fator	3,53	3,72	3,84	3,47	3,47	3,55	3,19	3,22	3,36	3,13	3,22	3,39	3,08	3,25	3,33

Tabela 18 – Fator de Aumento do tempo para o Quantitativo Relacional

5.4.1.3. Apriori Objeto-Relacional x Apriori Quantitativo Objeto-Relacional

Continuando a análise das diferenças em termos de desempenho entre as duas versões do *Apriori* sobre as mesmas bases de dados, e seguindo os moldes da análise anterior, a tabela 19 mostra que, também na versão Objeto-Relacional, o *Apriori* Quantitativo foi cerca de três vezes mais lento que o *Apriori* clássico.

Incremento	500			1000			2000			4000			8000		
Suporte (%)	**20**	**30**	**40**	**20**	**30**	**40**	**20**	**30**	**40**	**20**	**30**	**40**	**20**	**30**	**40**
Fator	3,85	3,95	3,70	3,69	3,57	3,57	3,37	3,29	3,49	3,08	3,32	3,35	3,12	3,34	3,38

Tabela 19 – Fator de Aumento do tempo para o Quantitativo OR

Os resultados das tabelas 18 e 19 permitem concluir que, independente do tipo e do tamanho da base utilizada, o *Apriori* Quantitativo é sempre cerca de 3 vezes mais lento que o *Apriori* Clássico. Isto é justificado pela inútil infraestrutura de otimização que é montada pela versão quantitativa.

5.4.2. Análise Crítica Base Relacional x Objeto-Relacional Independente do Algoritmo

Após comparações entre algoritmos, o próximo passo é analisar o desempenho do mesmo algoritmo sobre bases puramente relacionais ou com extensões objeto-relacionais. A análise será feita inicialmente para o *Apriori* simples e depois se estenderá ao Quantitativo.

Cabe ressaltar que as implementações relacional e objeto-relacional dos dois algoritmos são idênticas, diferindo apenas na leitura da base de dados utilizada como entrada.

As tabelas de candidatos, árvores de conjuntos e de intervalos independem do tipo de base e são sempre conforme descrito nas seções 4.2 e 4.3.

Conforme comprovado anteriormente, o que realmente influencia no desempenho global dos algoritmos são as várias passagens pelos dados, recuperando as transações uma a uma, para a contagem do suporte. Um fato que justifica a rapidez das diversas funções menores executadas pelos algoritmos é que, de uma maneira geral, as consultas ao banco de dados são muito simples. Aqui está se falando das consultas que são realizadas pelos dois algoritmos e independem da base. São excluídas as varreduras das bases transação a transação, o que obviamente depende da modelagem das bases e a manutenção das árvores de intervalos, operação particular do *Apriori* Quantitativo.

Como os 'ponteiros' das árvores são sempre simulados a partir do *ROWID*, várias consultas utilizam este valor para recuperação de candidatos, suportes e códigos de produtos. Este é um método ultra-eficiente de acesso e, para comprovar o seu uso, a tabela 20 traz algumas consultas realizadas por ambos os algoritmos, independente do modelo da base de dados.

Consulta	**Função**
select count(*) from venda_r	Contar o total de transações
select count(*) from c2_t;	Contar candidatos de tamanho 2
Insert into C_3 Select p.item$_1$, p.item$_2$,q.item$_2$ From L_2 p, L_2 q Where p.item$_1$ = q.item$_1$,p.item$_2$ < q.item$_{2;}$	Geração de candidatos de tamanho 3
select * from ArvConj where rowid = Rowid_v;	Recupera um nó da árvore
select Produto2 from C2_t where rowid = Rowid_v;	Recupera o segundo produto de um candidato de tamanho 2
select Suporte from C3_t where rowid = Rowid_v;	Recupera o suporte de um candidato de tamanho 3
select * from C3_t where rowid = Rowid_v;	Recupera um candidato de tamanho 3, com seu suporte.
select ROWID from C1_T where Produto1= CODITEM;	Recupera o identificador da linha de um determinado candidato

Tabela 20 – Algumas consultas freqüentes realizadas pelos algoritmos

As próximas duas seções versam sobre as possíveis formas de varredura da base de dados, analisando em que medida isto influencia no tempo para contagem do suporte dos conjuntos candidatos. No caso do *Apriori* Quantitativo, a montagem e manutenção das estruturas árvores de intervalos também causarão um impacto considerável no tempo total de execução.

5.4.2.1. Apriori Relacional x Apriori Objeto-Relacional

A única diferença da implementação relacional para a objeto-relacional é com relação à busca das transações a partir da base modelada de forma diferente. Para cada venda, os códigos dos itens vendidos precisam ser recuperados. Neste ponto, para cada modelo de base de dados, haverá uma consulta diferente.

Na base relacional, a consulta será algo do tipo, realizada através de um cursor:

```
SELECT coditem FROM itemvenda_r WHERE codvenda = codvenda_v;
```

Já no caso objeto-relacional, a consulta será um pouco diferente. Uma *Nested Table* não pode ser consultada diretamente. Desta forma, seus valores só podem ser recuperados a partir da tabela mãe. A consulta pode ser feita usando a expressão "*TABLE*" (ou "*THE"*). Isto faz com que a *Nested Table* seja vista como uma tabela que pode ser usada normalmente na cláusula *FROM* de forma que os dados possam ser recuperados. Um "alias" é dado à porção da *Nested Table* recuperada que pode ser usada como se fosse uma tabela normal. A consulta fica como segue:

```
SELECT L.coditem
FROM venda_or v, TABLE(v.Linhasitens) L
WHERE v.numvenda =  numvenda_v;
```

Toda a manipulação de dados dos algoritmos utiliza apenas os códigos dos itens adquiridos. É fácil perceber que, neste caso, na base relacional, tais códigos são recuperados diretamente da tabela ItemVenda_r. Já no caso OR, os códigos são recuperados da *Nested Table*. Caso a recuperação de outras informações relativas a um item, fosse constante, poderia

ter sido incluído um apontador (REF) para o item diretamente na *Nested Table*. Desta forma, o acesso aos dados seria feito apenas percorrendo tal apontador, ao invés de ser necessário ir numa terceira tabela, através de junções, como no caso puramente relacional.

A tabela 21 ilustra o ganho, em termos de desempenho, da versão com simples tabelas normalizadas. Para cada base é mostrado o percentual de ganho da versão relacional sobre a objeto-relacional. Exemplificando, para a base com 1000 transações e suporte de 20%, a versão relacional do *Apriori* Quantitativo levou aproximadamente o tempo do *Apriori* simples mais 9,70% desse mesmo tempo.

Incremento	**500**			**1000**			**2000**			**4000**			**8000**		
Suporte(%)	**20**	**30**	**40**	**20**	**30**	**40**	**20**	**30**	**40**	**20**	**30**	**40**	**20**	**30**	**40**
Fator	2,70	4,10	8,10	9,70	13,60	13,79	15,68	13,07	10,39	21,04	13,94	15,74	19,95	14,55	13,22

Tabela 21 – Percentual de ganho do Ap_R sobre o Ap_OR

Pela observação da tabela 21, percebe-se que a versão relacional sempre ganha da objeto-relacional e esse ganho tende a variar com o crescimento da base e com a mudança do suporte.

5.4.2.2. Apriori Quantitativo Relacional x Apriori Quantitativo Objeto-Relacional

Para o caso do *Apriori* Quantitativo, tudo o que foi comentado na seção anterior em relação às diferenças das bases de dados é válido. Além disso, as estruturas árvores de intervalos são criadas em tempo de execução. Entretanto, toda essa manipulação das árvores é feita tanto na versão relacional quanto na objeto-relacional. Assim, a diferença continua sendo apenas em relação à passagem pelos dados para contagem do suporte.

Incremento	**500**			**1000**			**2000**			**4000**			**8000**		
Suporte(%)	**20**	**30**	**40**	**20**	**30**	**40**	**20**	**30**	**40**	**20**	**30**	**40**	**20**	**30**	**40**
Fator	11,95	10,66	4,06	16,78	16,63	14,59	21,98	15,58	14,60	18,78	17,60	14,27	21,84	17,96	14,75

Tabela 22 – Percentual de Ganho do ApQ_R sobre o ApQ_OR

A tabela 22 comprova mais uma vez que a base objeto-relacional causa perda no desempenho do algoritmo.

5.4.3. Análise do Desempenho em Função do Tamanho da Base

Esta seção tem como objetivo mostrar como a diferença evolui em função da variação do volume de dados, aqui denominado *ΔV*.

Como mostrado na tabela 16, o volume de dados foi crescendo de maneira uniforme. Partindo de uma base com 500 transações até chegar na última com 8000, o total de vendas foi sempre dobrado. Cabe lembrar que se tentou utilizar bases bem parecidas, todas com uma média de 3 itens por transação, justamente para que esta análise pudesse ser feita.

A tabela 23 ilustra o fator de aumento do tempo de execução dos algoritmos em função do *ΔV*. É fácil perceber que este fator foi em todos os casos bem próximo de 2, significando que o tempo de execução dobrou.

ΔV	**Suporte**	**Ap_R**	**Ap_OR**	**ApQ_R**	**ApQ_OR**
500-1000	20	1,81	1,93	1,78	1,85
	30	1,87	2,04	1,75	1,84
	40	1,91	2,02	1,77	1,95
1000-2000	20	2,3	2,14	1,87	1,96
	30	2,07	2,06	1,91	1,90
	40	2,01	1,95	1,90	1,90
2000-4000	20	2,05	2,14	2,01	1,96
	30	1,97	1,99	1,97	2,01
	40	1,96	2,05	1,97	1,97
4000-8000	20	1,99	1,97	1,95	2,00
	30	1,96	1,97	1,97	1,98
	40	2,02	1,97	1,98	1,99

Tabela 23 – Fator de variação do tempo de execução em função do *ΔV*

Utilizando os resultados da tabela 23, percebe-se que para a variação no tamanho da base de dados que foi sempre dobrada, causou uma variação também uniforme nos tempos de execução que, independente da base e do algoritmo, praticamente também dobrou. Exemplificando, para um suporte de 20%, quando a base passou de 500 para 1000 transações o *Apriori* relacional (Ap_R) aumentou em 1,8 seu tempo de execução. Assim, dobrar o volume dos dados significa dobrar o tempo de execução do algoritmo seja ele em qualquer uma das duas versões e utilizando qualquer tipo de base.

5.4.4. Variação do Limite de um Nó

O limite do nó de uma árvore de intervalos é um parâmetro de extrema importância, visto que influencia diretamente na profundidade da estrutura. Quanto menor seu valor, mais divisões serão efetuadas e, conseqüentemente, mais profunda a árvore. Por outro lado, teoricamente, quanto mais profunda, menores os intervalos de quantidades adquiridas e, conseqüentemente mais difícil a sobreposição de tais intervalos. Entretanto, quanto mais profunda mais onerosa sua manipulação, posto que a tabela referente a essa estrutura terá mais linhas.

O artigo [4] de definição do algoritmo *Apriori* Quantitativo não faz ressalva alguma a respeito deste valor. Neste trabalho, em todas as bases utilizadas como teste, a quantidade adquirida de um item era, no máximo, igual a 10, valor este definido randomicamente. Com isto, os intervalos já são por natureza pequenos, não fazendo sentido muitas divisões.

Durante os testes, o valor do limite foi variado várias vezes e percebeu-se claramente o quanto ele influencia no desempenho das rotinas. No entanto, em nenhuma dessas variações o *Apriori* Quantitativo conseguiu podar candidatos.

Ficou claro que, mantendo o valor do limite constante e aumentando o tamanho da base, o *Apriori* quantitativo perdia cada vez mais em termos de desempenho em relação ao *Apriori*. Ora, se o limite é o mesmo, mas o número de transações cresce, evidentemente que as árvores ficarão mais profundas. Após várias análises desse tipo, a solução foi variar o valor do limite de acordo com o tamanho da base. O parâmetro foi então estabelecido em 30% do número de transações. Assim, a tomada dos tempos dos algoritmos começou a mostrar comportamento uniforme. Como mostrado, tendo o número de transações dobrado, os tempos

de execução das rotinas também apresentam um comportamento mais ou menos uniforme, dobrando as medidas de tempo.

5.4.5. Crítica dos Resultados Obtidos

Dos vários testes executados para análise dos algoritmos, ficou claro que a poda dos candidatos pela otimização proposta para o algoritmo *Apriori* Quantitativo em [4] não é freqüente, pelo menos em se tratando do caso analisado (vendas de uma mercearia) e levando em consideração as modelagens das bases de dados aqui propostas. Ao mesmo tempo, o gasto com a montagem e manutenção da infraestrutura de poda é altíssimo.

Um outro fator a ser analisado é que a estrutura das árvores de intervalos pode variar de acordo com vários fatores. Por exemplo, as quantidades adquiridas dos itens e o valor limite de um nó exercem um papel crucial. O primeiro parâmetro depende realmente das bases de dados, enquanto que, para o segundo, não existe até então uma forma ótima de definir seu valor. De um modo geral, a simulação das árvores de intervalos em tabelas *Oracle* mostrou que a manutenção dessas estruturas ficou bastante onerosa, causando séria queda de desempenho do *Apriori* Quantitativo em relação ao *Apriori* original.

Por outro lado, ficou claro também, e isto foi comprovado através da amostragem de alguns tempos, que a parte mais onerosa do algoritmo, em qualquer uma das suas versões, é a passagem pelos dados para contagem do suporte dos candidatos. Como é justamente aí que residem as diferenças entre as modelagens relacional e objeto-relacional (busca pelos itens de uma venda modelada de forma diferente), várias comparações puderam ser feitas. Em todos os casos, a base puramente relacional mostrou-se mais rápida, mesmo levando em consideração que as bases foram modeladas seguindo as sugestões dos manuais do *Oracle* e que o funcionamento do otimizador de consultas foi amplamente avaliado e testado. Aparentemente, a tendência é que isto seja agravado de acordo com o crescimento da base de dados. Por exemplo, houve um aumento sensível na perda de desempenho das versões objeto-relacionais quando o incremento da base passou de 500 para 1000 transações.

Analisando o crescimento da base individualmente, percebe-se que também em todos os casos, o comportamento foi linear: dobrar a base significa dobrar o tempo de execução dos algoritmos.

Por fim, uma conclusão geral é que, em todos os casos, sejam eles com bases relacionais ou objeto-relacionais, o *Apriori* Quantitativo mostrou-se cerca de 3 vezes mais lento que o original.

Capítulo 6

Conclusões e Sugestões de Trabalhos Futuros

A utilização do processo de mineração de dados é de extrema importância, visto que, através dela, muitos dados armazenados em poderosos SGBD's podem ser transformados em informações relevantes e que podem ser utilizadas de várias formas no processo de tomada de decisão.

Especificamente falando da geração de regras de associação, o poder e, ao mesmo tempo, a flexibilidade desta técnica já foram devidamente ressaltados em aplicações de comércio varejista, pelo menos. A capacidade de descobrir associação entre produtos vendidos é extraordinariamente importante ao processo de tomada de decisão nas empresas de comércio varejista. Entretanto, os algoritmos até agora propostos com esta finalidade ainda apresentam muitos problemas, que são agravados com o crescimento das bases de dados a serem mineradas.

A busca por algoritmos de regras de associação mais eficientes ainda é claramente uma questão aberta de pesquisa. O algoritmo *Apriori* Quantitativo, que pode ser considerado o estado-da-arte em otimização dos algoritmos da família *Apriori*, não passa afinal de contas de uma idéia para a descoberta eficiente de grandes conjuntos para gerar regras de associação válidas (isto é, com suporte e confiança iguais ou maiores que os valores mínimos especificados), mas que carece de uma investigação mais aprofundada do *overhead* necessário justamente para realizar a otimização proposta. A explicação para esta 'falha' é que o algoritmo nunca tinha sido efetivamente implementado e testado.

A primeira contribuição do nosso trabalho é que, até onde vai o nosso conhecimento, fomos os primeiros a efetivamente por à prova o algoritmo *Apriori* Quantitativo. Ao ser completamente integrado ao *Oracle* e comparado com o *Apriori* em sua forma original e submetido a condições severas de teste, apresentou desempenho claramente insatisfatório relativamente ao *Apriori* básico, clássico. A explicação é que a versão quantitativa do algoritmo, explorando justamente as informações quantitativas para acelerar a descoberta de grandes conjuntos válidos de itens, exige por outro lado a montagem de uma infra-estrutura de otimização complexa e onerosa, ou pelo menos muito mais complexa e onerosa que a exigida pelo *Apriori* básico. Assim, mesmo montando tal infra-estrutura o desempenho foi

insatisfatório. No final, as vantagens do *Apriori* Quantitativo foram em larga margem obscurecidas pelas desvantagens que elas engendram (*side effects*). Assim, recomendamos ainda o algoritmo *Apriori* básico como o melhor algoritmo existente para gerar regras de associação em aplicações práticas críticas. A proposta de otimização do *Apriori* Quantitativo pode até ser benéfica quando se está trabalhando de forma independente dos SGBD's, conforme proposto em [4], mas isto vai de encontro ao segundo objetivo deste trabalho que é justamente integração do algoritmo ao *Oracle*.

Outra grande vertente das pesquisas em mineração de dados se preocupa com a integração, em princípio natural e necessária, entre algoritmos de mineração de dados e SGBD's, somando as vantagens de cada uma das tecnologias. Até o presente momento, as bases de dados utilizadas pelos algoritmos de mineração de dados têm sido sob a forma de arquivos completamente desintegrados de SGBD's (arquivos *stand alone*) e preparados especialmente para mineração. Tais arquivos são de preferência não normalizados, conseqüentemente contendo muitos dados redundantes. A redundância é uma característica importante em mineração de dados, visto que potencialmente permite encontrar mais rapidamente associações entre dados.

A segunda grande conclusão é que muito ainda precisa ser feito para que as técnicas de mineração através da geração de regras de associação possam ser utilizadas de forma eficiente e integrada aos SGBD's, como forma de aproveitar, ainda mais, o poder destes dois mundos.

Pelos resultados que obtivemos, e os nossos experimentos são uma evidência insofismável disso, integrar mineração de dados com SGBD's não é de modo algum uma panacéia. Esperava-se que os desempenhos das versões objeto-relacionais dos algoritmos *Apriori* quantitativo e básico fossem bem superiores aos desempenhos das respectivas versões relacionais dos algoritmos. A razão para isto é que os tipos coleção suportados pelo modelo objeto-relacional permitem a construção de tabelas não normalizadas, o que em princípio é bom para os algoritmos de mineração. Não foi, porém o que se viu, pelo menos em nossos experimentos com uma aplicação de comércio varejista. O que aconteceu foi que mesmo sendo modelada seguindo as sugestões dos manuais da *Oracle*, qualquer consulta à base objeto-relacional foi mais lenta. Desta forma, o ponto em que a implementação relacional foi superior à objeto-relacional foi exatamente na contagem do suporte dos conjuntos candidatos, operação que exige várias passagens pelos dados.

A experiência que adquirimos nos leva a insistir que as pesquisas em integração de mineração de dados com SGBD's ainda são muito incipientes e precisam ser intensificadas, e que as promessas da tecnologia objeto-relacional ainda não se realizaram em sua plenitude, exigindo que muito esforço ainda seja gasto para o aprimoramento desta tecnologia.

6.1. Sugestões de Trabalhos Futuros

Como trabalhos futuros, várias sugestões podem ser deixadas. Inicialmente, seria interessante investigar em que medida as conclusões relativas ao caso de vendas valem para outras aplicações de mineração de dados. Este foi só um estudo de caso, no qual os dados foram modelados de forma bem simples. É interessante testar o comportamento do *Apriori* Quantitativo, bem como o desempenho das bases objeto-relacionais em outros casos.

Especificamente em relação ao *Apriori* Quantitativo, muita coisa ainda pode ser pesquisada em relação à montagem e manutenção das tabelas árvores de intervalos. Não ficou claramente definido, por exemplo, qual o limite 'ótimo' – aquele que indica quando um nó precisa ser dividido - do contador de ocorrência, para ser usado em um nó de uma árvore de intervalos.

Por fim, seria extremamente interessante ir mais a fundo nas diferenças de implementação relacional e objeto-relacional. Neste trabalho, apenas as estruturas dos dados de entrada para os algoritmos diferiam. Entretanto, as estruturas dinâmicas (tabelas de candidatos, árvores de conjuntos e de intervalos) eram idênticas em todas as versões do algoritmo. Sendo assim, a aplicação de estruturas dinâmicas diferentes, para cada implementação utilizada, poderia acarretar sensíveis alterações nos resultados das rotinas.

Anexo 1

Operações sobre Árvore de Conjuntos

```
CREATE OR REPLACE TYPE APONT_T AS VARRAY(100) OF CHAR(18);

CREATE OR REPLACE TYPE CONJUNTO_T AS VARRAY(500) OF NUMBER;

CREATE OR REPLACE TYPE NO_T AS OBJECT (
  PRODUTO       number,
  SUPORTE       number,
  APONT         APONT_t,
  APONTPAI      CHAR(18),
  STATIC PROCEDURE CriaArvore(Tabela varchar2),
  STATIC PROCEDURE InsereNo(Tabela varchar2,CONJUNTO CONJUNTO_t),
  STATIC FUNCTION    PesquisaConj(Tabela varchar2,CONJUNTO CONJUNTO_t)
                     RETURN boolean,
  STATIC PROCEDURE AtualizaSuporte
                    (Tabela varchar2,CONJUNTO CONJUNTO_t, Suporte number),
  STATIC FUNCTION   BuscarSuporte(Tabela varchar2,CONJUNTO CONJUNTO_t)
                    RETURN number);
/
CREATE OR REPLACE TYPE CONJUNTOROWID_T AS VARRAY(1500) OF char(18)
/
CREATE OR REPLACE TYPE BODY NO_T AS

STATIC PROCEDURE CriaArvore(Tabela varchar2) IS
 v_Comando varchar2(500);
 AUX         varchar2(500) := '';
 BEGIN
   V_Comando := 'Create table ' || Tabela || ' of NO_T tablespace Apriori';
   EXECUTE IMMEDIATE V_Comando;
   v_comando :='insert into '||Tabela||' values (000,0,APONT_t(),NULL)';
   EXECUTE IMMEDIATE V_Comando;
 END;

 STATIC PROCEDURE InsereNo(Tabela varchar2,CONJUNTO CONJUNTO_T) IS
 NORAIZ   NO_T;
 NOVONO           NO_T;
 NOATUAL          NO_T;
 NOFILHO          NO_T;
 ROWID_V          ROWID;
 ROWID_ATUAL   ROWID;
 v_Comando        varchar2(800);
 AUX              varchar2(800) := '';
 ACHOU            BOOLEAN     := FALSE;
```

```
NIVEL               NUMBER;
Cont                number;
BEGIN
  v_comando := 'select value(T) from ' || Tabela || ' T where PRODUTO = 000';
  EXECUTE IMMEDIATE V_Comando INTO NoRaiz;
  v_comando :='select ROWID from '||Tabela||' T where PRODUTO = 000';
  EXECUTE IMMEDIATE V_Comando INTO ROWID_ATUAL;
  NoAtual := NoRaiz;
  IF CONJUNTO.COUNT <> 1 THEN
     Nivel   := 1;
     NoAtual := NoRaiz;
     FOR i IN 1..CONJUNTO.COUNT-1 LOOP
       ACHOU   := FALSE;
       FOR i IN 1..NoAtual.Apont.Count LOOP
            ROWID_V := NoAtual.Apont(i);
            v_comando := 'select value(T) from '||Tabela || ' T where ROWID= ''' || Rowid_v || '''';
            EXECUTE IMMEDIATE V_Comando INTO NoFilho;
            ROWID_ATUAL := ROWID_V;
         IF NoFilho.Produto = Conjunto(Nivel) THEN
           ACHOU := TRUE;
           EXIT;
         END IF;
          END LOOP;
          NoAtual := NoFilho;
      NIVEL := NIVEL + 1;
     END LOOP;
  END IF; /*IF CONJUNTO.COUNT <> 1 THEN */
  v_comando :='insert into '||Tabela||' values (999,0,APONT_t(),NULL)';
  EXECUTE IMMEDIATE V_Comando;
  v_comando :='select ROWID from '||Tabela||' T where PRODUTO = 999';
  EXECUTE IMMEDIATE V_Comando INTO ROWID_V;
  v_comando:='UPDATE '||Tabela||' SET PRODUTO = ' || Conjunto(CONJUNTO.COUNT)||
                    ', APONTPAI = ''' || ROWID_ATUAL || ''' where ROWID= ''' || Rowid_v || '''';
  EXECUTE IMMEDIATE V_Comando;
  NOATUAL.APONT.EXTEND;
  NOATUAL.APONT(NOATUAL.APONT.COUNT) := ROWID_V;
  AUX := '';
  FOR i IN 1..NOATUAL.APONT.COUNT LOOP
     AUX := AUX || '''' || NoATUAL.APONT(i) || '''';
     IF i <> (NOATUAL.APONT.count) THEN
        AUX := AUX || ',' ;
     END IF;
  END LOOP;
  v_comando :='update ' || Tabela ||' set APONT = APONT_t('||AUX||')'
            ||' where ROWID ='' || ROWID_ATUAL || '''';
  EXECUTE IMMEDIATE V_Comando;
END;
```

STATIC FUNCTION

PesquisaConj(Tabela varchar2,CONJUNTO CONJUNTO_t) RETURN boolean IS

```
ACHOU            BOOLEAN := FALSE;
v_Comando        varchar2(200);
ROWID_V          ROWID;
ROWID_ATUAL ROWID;
NIVEL             NUMBER;
NORAIZ           NO_T;
NOATUAL          NO_T;
NOFILHO          NO_T;
BEGIN
v_comando := 'select value(T) from ' || Tabela || ' T where PRODUTO = 000';
EXECUTE IMMEDIATE V_Comando INTO NoRaiz;
v_comando :='select ROWID from '||Tabela||' T where PRODUTO = 000';
EXECUTE IMMEDIATE V_Comando INTO ROWID_ATUAL;
Nivel   := 1;
NoAtual := NoRaiz;
FOR i IN 1..CONJUNTO.COUNT LOOP
  ACHOU   := FALSE;
  FOR J IN 1..NoAtual.Apont.Count LOOP
    ROWID_V := NoAtual.Apont(J);
    v_comando := 'select value(T) from ' || Tabela || ' T where ROWID= ''' || Rowid_v || '''';
    EXECUTE IMMEDIATE V_Comando INTO NoFilho;
    IF NoFilho.Produto = Conjunto(Nivel) THEN
      ACHOU := TRUE;
      NoAtual := NoFilho;
      EXIT;
    END IF;
  END LOOP;
  if Achou then
   NIVEL := NIVEL + 1;
  else
   EXIT;
  end if;
END LOOP;
RETURN Achou;
END;
```

STATIC PROCEDURE
AtualizaSuporte(Tabela varchar2,CONJUNTO CONJUNTO_t, Suporte number) IS

```
v_Comando        varchar2(200);
Achou            boolean;
ROWID_V          ROWID;
ROWID_ATUAL ROWID;
NIVEL             NUMBER;
CONT              NUMBER;
NORAIZ           NO_T;
NOATUAL          NO_T;
NOFILHO          NO_T;
BEGIN
```

```
v_comando := 'select value(T) from ' || Tabela || ' T where PRODUTO = 000';
EXECUTE IMMEDIATE V_Comando INTO NoRaiz;
 v_comando :='select ROWID from '||Tabela||' T where PRODUTO = 000';
EXECUTE IMMEDIATE V_Comando INTO ROWID_ATUAL;
Nivel   := 1;
NoAtual := NoRaiz;
FOR i IN 1..CONJUNTO.COUNT LOOP
  ACHOU   := FALSE;
  FOR i IN 1..NoAtual.Apont.Count LOOP
    ROWID_V := NoAtual.Apont(i);
    if ROWID_V is not null then
      v_comando := 'select value(T) from ' || Tabela || ' T where ROWID= '' || Rowid_v || '';
      EXECUTE IMMEDIATE V_Comando INTO NoFilho;
      IF NoFilho.Produto = Conjunto(Nivel) THEN
        ACHOU := TRUE;
        NoAtual := NoFilho;
        rowid_atual := rowid_v;
        EXIT;
      END IF;
    END IF;
  END LOOP;
  if Achou then
    NIVEL := NIVEL + 1;
  else
    EXIT;
  end if;
END LOOP;
 v_comando :='update '|| Tabela ||
                  ' set Suporte = '||Suporte||' where ROWID ='''|| ROWID_ATUAL || '';
EXECUTE IMMEDIATE V_Comando;
END;
```

```
STATIC FUNCTION BuscarSuporte(Tabela varchar2,CONJUNTO CONJUNTO_t)
                    Return number IS
  v_Comando       varchar2(200);
  Achou           boolean;
  ROWID_V         ROWID;
  ROWID_ATUAL ROWID;
  NIVEL           NUMBER;
  CONT            NUMBER;
  Suporte         NUMBER;
  NORAIZ          NO_T;
  NOATUAL         NO_T;
  NOFILHO         NO_T;
  BEGIN
  v_comando := 'select value(T) from ' || Tabela || ' T where PRODUTO = 000';
  EXECUTE IMMEDIATE V_Comando INTO NoRaiz;
  v_comando :='select ROWID from '||Tabela||' T where PRODUTO = 000';
  EXECUTE IMMEDIATE V_Comando INTO ROWID_ATUAL;
```

```
   Nivel   := 1;
   NoAtual := NoRaiz;
   FOR i IN 1..CONJUNTO.COUNT LOOP
     ACHOU   := FALSE;
     FOR i IN 1..NoAtual.Apont.Count LOOP
       ROWID_V := NoAtual.Apont(i);
       if ROWID_V is not null then
         v_comando := 'select value(T) from ' || Tabela || ' T where ROWID= ''' || Rowid_v || '''';
         EXECUTE IMMEDIATE V_Comando INTO NoFilho;
         IF NoFilho.Produto = Conjunto(Nivel) THEN
           ACHOU := TRUE;
           NoAtual := NoFilho;
           rowid_atual := rowid_v;
           EXIT;
         END IF;
       END IF;
     END LOOP;
     if Achou then
      NIVEL := NIVEL + 1;
     else
      EXIT;
     end if;
   END LOOP;
   if Achou then
     v_comando :='select Suporte from ' ||Tabela||' where ROWID =''' || ROWID_ATUAL || '''';
     EXECUTE IMMEDIATE V_Comando into Suporte;
   else
     Suporte := 0;
   end if;
   RETURN Suporte;
  END;
END;

CREATE OR REPLACE PROCEDURE BuscarConjunto
                          (Origem varchar2, ROWID_v ROWID, ROWIDRAIZ ROWID,
                           Conjunto IN OUT Conjunto_t, Suporte OUT number) IS
Cont           number;
v_comando   varchar2(500);
Produto        number;
APONTPAI  ROWID;
BEGIN
 Cont := 1;
 v_comando:='select Produto from '||Origem ||' where rowid= ''' || Rowid_v || '''';
 execute immediate v_comando into Produto;
 v_comando:='select APONTPAI from '||Origem ||' where rowid= ''' || Rowid_v || '''';
 execute immediate v_comando into APONTPAI;
 Conjunto(Cont) := Produto;
 Cont := Cont + 1;
 WHILE APONTPAI <> ROWIDRAIZ LOOP
```

```
   v_comando:='select Produto from '|| Origem ||' where rowid= '' || APONTPAI || '';
   execute immediate v_comando into Produto;
   v_comando:='select APONTPAI from '|| Origem ||' where rowid= '' || APONTPAI || '';
   execute immediate v_comando into APONTPAI;
   if Conjunto.Count < Cont then
     Conjunto.EXTEND;
   end if;
   Conjunto(Cont) := Produto;
   Cont := Cont + 1;
 END LOOP;
 v_comando:='select Suporte from ' || Origem || ' where rowid= '' || Rowid_v || '';
 execute immediate v_comando into Suporte;
END;
```

CREATE OR REPLACE PROCEDURE InserirGrandesConjuntos
(Destino varchar2, Origem varchar2, Tamanho number) IS

```
 valor               number;
 i                    NUMBER := 0;
 Cont                number;
 Numlinhas          number;
 Suporte             number;
 k                    number;
 j                    number;
 v_Comando         varchar2(800);
 TYPE Cursor_t    IS REF CURSOR;
 CursorROWID     Cursor_t;
 RowId_v            ROWID;
 Conjunto           Conjunto_t           := Conjunto_t();
 ConjuntoROWID ConjuntoRowid_t := ConjuntoRowid_t();
BEGIN
   v_comando := 'select ROWID from '|| Origem;
   Cont := 0;
   OPEN CursorRowid FOR V_comando;
     FETCH CursorRowId INTO rowid_v;
     WHILE CursorRowid%FOUND LOOP
       Cont := Cont + 1;
       if Cont > ConjuntoRowid.Count then
            ConjuntoRowId.EXTEND;
       end if;
       ConjuntoRowid(Cont) := rowid_v;
       FETCH CursorRowid INTO rowid_v;
     END LOOP;
   CLOSE CursorRowid;
   v_comando := 'select count(*) from '|| Origem;
   execute immediate V_comando into NumLinhas;
   for k in 1..Tamanho loop
    Conjunto.EXTEND;
    Conjunto(k) := 0;
   end loop;
```

```
   for k in 1..NumLinhas loop
     for j in 1..Tamanho loop
       v_comando:='select Produto'||j||
                        ' from '||Origem||' where rowid= ''' ||ConjuntoRowid(k) || '''';
       execute immediate v_comando into valor;
       Conjunto(j) := Valor;
     end loop;
     v_comando:='select Suporte from ' || Origem ||
                      ' where rowid= ''' ||ConjuntoRowid(k) || '''';
     execute immediate v_comando into Suporte;
     NO_T.InsereNo(Destino,Conjunto);
     NO_T.AtualizaSuporte(Destino,CONJUNTO, Suporte);
   end loop;
   V_COMANDO := 'drop table ' || Origem;
   execute immediate v_comando;
END;
```

Anexo 2
Operações sobre Árvores de Intervalos

```
CREATE OR REPLACE TYPE ITEMRANGE_T AS OBJECT (
  Minimo        number,
  Maximo        number);
/
CREATE OR REPLACE TYPE RANGESET_T AS VARRAY(20) OF ITEMRANGE_T;
/
CREATE OR REPLACE TYPE KDTREE_T AS OBJECT (
  Contador        number,
  Rangeset        rangeset_t,
  FilhoEsquerdo   char(18),
  FilhoDireito    char(18),
  STATIC PROCEDURE
          CriaArvore(Tabela varchar2, Dimensao number, Rangeset Rangeset_t),
  STATIC PROCEDURE
      InsereNo(Tabela varchar2,dimensão number,Limite number,Aquisicao Conjunto_t));
  /
CREATE OR REPLACE TYPE BODY KDTREE_T AS

 STATIC PROCEDURE
          CriaArvore(Tabela varchar2, Dimensao number, Rangeset Rangeset_t) IS
 v_Comando varchar2(200);
 Rangesets    varchar2(200) := '';
 BEGIN
  V_Comando := 'Create table ' ||
                    Tabela || ' of KDTREE_T TABLESPACE APRIORIQUANT';
  EXECUTE IMMEDIATE V_Comando;
  FOR i IN 1..Rangeset.COUNT LOOP
   Rangesets:=Rangesets||'Itemrange_t('||Rangeset(i).Minimo||','||Rangeset(i).Maximo||')';
   IF i <> (Rangeset.count) THEN
    Rangesets := Rangesets || ',';
   END IF;
  END LOOP;
  v_comando :='insert into '||Tabela||' values (KDTREE_T(0,Rangeset_t('||
                  Rangesets||'),null,null))';
  EXECUTE IMMEDIATE V_Comando;
  v_comando := 'create index ' || Tabela || '_idx on ' || Tabela ||
                  ' (Contador)TABLESPACE APRIORIQUANT';
  EXECUTE IMMEDIATE V_Comando;
  V_Comando := 'Create table ' || Tabela || '_t (';
  FOR i IN 1..dimensão LOOP
```

```
    v_comando := v_comando || 'Valor' || i || ' number';
    IF i <> (Dimensao) THEN
      v_comando := v_comando || ',';
    END IF;
  END LOOP;
  v_comando := v_comando || ')TABLESPACE APRIORIQUANT';
  EXECUTE IMMEDIATE V_Comando;
 END;

 STATIC PROCEDURE
        InsereNo(Tabela varchar2,dimensão number,Limite number,Aquisicao
Conjunto_T) IS
  NoAtual          kdtree_t;
  NoEsquerdo       kdtree_t;
  NoDireito        kdtree_t;
  ROWID_V          ROWID;
  FilhoEsquerdo_v ROWID;
  FilhoDireito_v   ROWID;
  Cont             number  := 0;
  Nivel            number  := 0;
  Disc             number  := 0;
  DiscAnterior     number  := 0;
  Chave            number  := 0;
  Inseriu          boolean := false;
  Alterou          boolean := false;
  v_Comando        varchar2(200) := '';
  Rangesets        varchar2(200) := '';
  NaoDivide        boolean;
  BEGIN
   v_comando := 'select max(Contador) from '|| Tabela;
   EXECUTE IMMEDIATE V_Comando INTO Cont;
   v_comando := 'select value(T) from ' || Tabela || ' T where Contador = ' || Cont;
   EXECUTE IMMEDIATE V_Comando INTO NoAtual;
   v_comando := 'select ROWID from ' || Tabela || ' T where Contador = ' || Cont;
   EXECUTE IMMEDIATE V_Comando INTO ROWID_V;
   Nivel := 1;
   v_comando := 'insert into '|| Tabela || '_t values(';
   for i in 1..aquisicao.Count LOOP
    v_comando := v_comando || Aquisicao(i);
    if i <> Aquisicao.Count THEN
      v_comando := v_comando ||',';
    END IF;
   END LOOP;
   v_comando := v_comando ||')';
   EXECUTE IMMEDIATE v_Comando;
   commit;
   LOOP
    NoAtual.Contador := NoAtual.Contador + 1;
    IF Nivel <> Dimensao  THEN
```

```
        Disc := MOD(Nivel, Dimensao);
        if Disc = 0 then
             Disc:= Dimensao;
        end if;
      ELSE
        Disc := Dimensao;
      END
IF;
Chave:=TRUNC((NoAtual.Rangeset(Disc).Minimo+NoAtual.Rangeset(Disc).Maximo)/2);
v_comando := 'update '|| Tabela || ' set Contador = ' || NoAtual.Contador ||
                    ' where ROWID= '' || Rowid_v || '';
      EXECUTE IMMEDIATE V_Comando;
      IF (NoAtual.Contador > Limite) and (NoAtual.FilhoEsquerdo is not NULL)
         and (NoAtual.FilhoDireito is not NULL) THEN
        IF AQUISICAO(Disc) < Chave THEN
          ROWID_V := NoAtual.FilhoEsquerdo;
          v_comando := 'select value(T) from ' || Tabela ||
                    ' T where ROWID = ''|| NoAtual.FilhoEsquerdo || '';
          EXECUTE IMMEDIATE V_Comando INTO NoAtual;
        ELSE
          ROWID_V := NoAtual.FilhoDireito;
          v_comando := 'select value(T) from ' || Tabela ||
                    ' T where ROWID = '' || NoAtual.FilhoDireito || '';
          EXECUTE IMMEDIATE V_Comando INTO NoAtual ;
        END IF;
        Nivel := Nivel + 1;
        IF (Nivel <> Dimensao)THEN
          Disc := MOD(Nivel, Dimensao);
          if Disc = 0 then
            Disc:= Dimensao;
          end if;
        ELSE
          Disc := Dimensao;
        END IF;
      Chave:=TRUNC
              ((NoAtual.Rangeset(Disc).Minimo+NoAtual.Rangeset(Disc).Maximo)/2);
      ELSE
      Alterou := false;
      FOR i IN 1..Aquisicao.Count LOOP
         IF Aquisicao(i) < NoAtual.Rangeset(i).Minimo THEN
           NoAtual.Rangeset(i).Minimo := Aquisicao(i);
           Alterou := true;
         END IF;
         IF Aquisicao(i) > NoAtual.Rangeset(i).Maximo THEN
           NoAtual.Rangeset(i).Maximo := Aquisicao(i);
           Alterou := true;
         END IF;
        END LOOP;
        IF Alterou THEN
```

```
  FOR i IN 1..Aquisicao.COUNT LOOP
    Rangesets := Rangesets||'Itemrange_t(' || NoAtual.Rangeset(i).Minimo ||','||
                 NoAtual.Rangeset(i).Maximo || ')';
    IF i <> (Aquisicao.count) THEN
      Rangesets := Rangesets || ',';
    END IF;
  END LOOP;
  v_comando :='update ' || Tabela ||' set Rangeset = Rangeset_t('||Rangesets||')'
          ||' where ROWID= '' || Rowid_v || '';
  EXECUTE IMMEDIATE V_Comando;
  Rangesets:= '';
  Chave:=TRUNC((NoAtual.Rangeset(Disc).Mínimo
          +NoAtual.Rangeset(Disc).Maximo)/2);
END IF; /* IF Alterou...*/
IF (NoAtual.Contador >= Limite) and (NoAtual.FilhoEsquerdo is NULL)
                                    and (NoAtual.FilhoDireito is NULL)THEN
   FOR j IN 1..Aquisicao.Count LOOP
  Rangesets := Rangesets||'Itemrange_t(0,0)';
  IF j <> (Aquisicao.count) THEN
    Rangesets := Rangesets || ',';
  END IF;
  END LOOP;
  v_comando :='insert into ' ||Tabela||' values (KDTREE_T(0,Rangeset_t('||
              Rangesets||'),null,null))';
  EXECUTE IMMEDIATE V_Comando;
  v_comando := 'select value(T) from ' || Tabela || ' T where Contador = 0';
  EXECUTE IMMEDIATE V_Comando INTO NoEsquerdo;
  IF Disc = 1 THEN
    DiscAnterior := Dimensao;
  ELSE
    DiscAnterior := Disc - 1;
  END IF;
  v_comando:='select count(*) from '||Tabela||'_t where valor'||Disc||' <= '
          ||Chave||' and Valor'||DiscAnterior||' >= '
          ||NoAtual.Rangeset(DiscAnterior).Minimo||' and Valor'||DiscAnterior
          ||' <= ' ||NoAtual.Rangeset(DiscAnterior).Maximo;
  EXECUTE IMMEDIATE V_Comando INTO NoEsquerdo.Contador;
  If (NoEsquerdo.Contador = 0) or (Noesquerdo.Contador >= Limite) then
  v_comando := 'select max(valor' ||Disc|| ') from '||Tabela||'_t where valor'
          || Disc || ' <= ' ||chave ||' and Valor'|| DiscAnterior ||' >= '
          ||NoAtual.Rangeset(DiscAnterior).Minimo ||' and Valor'
          ||DiscAnterior ||' <= ' ||NoAtual.Rangeset(DiscAnterior).Maximo;
  EXECUTE IMMEDIATE V_Comando INTO NoESquerdo.Rangeset(Disc).Maximo;
  v_comando := 'select min(valor' || Disc || ') from '
          ||Tabela ||'_t where valor' || Disc || ' <= ' ||chave ||' and Valor'
          || DiscAnterior ||' >= '||NoAtual.Rangeset(DiscAnterior).Minimo
          ||' and Valor'||DiscAnterior ||' <= '
          ||NoAtual.Rangeset(DiscAnterior).Maximo;
  EXECUTE IMMEDIATE V_Comando INTO NoESquerdo.Rangeset(Disc).Minimo;
```

```
FOR i IN 1..Aquisicao.Count LOOP
  IF i <> Disc THEN
    v_comando:= 'select max(valor' ||i|| ') from ' ||Tabela||'_t where valor'
            || Disc || ' <= ' ||chave||' and Valor'|| DiscAnterior
            ||' >= '|| NoAtual.Rangeset(DiscAnterior).Minimo||' and Valor'
            ||DiscAnterior||' <= '||NoAtual.Rangeset(DiscAnterior).Maximo;
    EXECUTE IMMEDIATE V_Comando INTO NoESquerdo.Rangeset(i).Maximo;
    v_comando:= 'select min(valor' ||i|| ') from '||Tabela||'_t where valor'
            || Disc || ' <= ' ||chave||' and Valor'|| DiscAnterior
            ||' >= '||NoAtual.Rangeset(DiscAnterior).Minimo||' and Valor'
            ||DiscAnterior||' <= '||NoAtual.Rangeset(DiscAnterior).Maximo;
    EXECUTE IMMEDIATE V_Comando INTO NoESquerdo.Rangeset(i).Minimo;
  END IF;
END LOOP;
v_comando := 'select ROWID from ' || Tabela || ' T where Contador = 0';
EXECUTE IMMEDIATE V_Comando INTO FilhoEsquerdo_V;
v_comando := 'update '|| Tabela || ' set Contador = ' || NoEsquerdo.Contador ||
             ' where ROWID= '' || FilhoEsquerdo_v || '';
EXECUTE IMMEDIATE V_Comando;
Rangesets:= '';
FOR i IN 1..Aquisicao.COUNT LOOP
  Rangesets := Rangesets||'Itemrange_t(' || NoEsquerdo.Rangeset(i).Minimo ||','||
               NoEsquerdo.Rangeset(i).Maximo || ')';
  IF i <> (Aquisicao.count) THEN
     Rangesets := Rangesets || ',';
  END IF;
END LOOP;
v_comando :='update ' || Tabela ||' set Rangeset = Rangeset_t('||Rangesets||')'
             ||' where ROWID= '' || FilhoEsquerdo_v || '';
EXECUTE IMMEDIATE V_Comando;
Rangesets:= '';
v_comando := 'update '||Tabela||' set FilhoEsquerdo = '' || FilhoEsquerdo_v || '' ||
             ' where ROWID = '' || RoWID_v || '';
EXECUTE IMMEDIATE V_Comando;
Rangesets:= '';
FOR j IN 1..Aquisicao.Count LOOP
   Rangesets := Rangesets||'Itemrange_t(0,0)';
   IF j <> (Aquisicao.count) THEN
     Rangesets := Rangesets || ',';
   END IF;
END LOOP;
v_comando :='insert into ' ||Tabela||' values (KDTREE_T(0,Rangeset_t('||
             Rangesets||'),null,null))';
EXECUTE IMMEDIATE V_Comando;
v_comando := 'select value(T) from ' || Tabela || ' T where Contador = 0';
EXECUTE IMMEDIATE V_Comando INTO NoDireito;
NoDireito.Contador := NoAtual.Contador - NoEsquerdo.Contador;
v_comando := 'select max(valor' || Disc || ') from '
        ||Tabela ||'_t where valor' || Disc || ' > ' ||chave||' and Valor'
```

```
                ||DiscAnterior ||' >= '||NoAtual.Rangeset(DiscAnterior).Minimo
                ||' and Valor'||DiscAnterior ||' <= ' ||NoAtual.Rangeset(DiscAnterior).Maximo;
        EXECUTE IMMEDIATE V_Comando INTO NoDireito.Rangeset(Disc).Maximo;
        v_comando := 'select min(valor' || Disc || ') from '
                    ||Tabela ||'_t where valor' || Disc || ' > ' ||chave||' and Valor'
                    || DiscAnterior ||' >= '||NoAtual.Rangeset(DiscAnterior).Minimo
                ||' and Valor'||DiscAnterior ||' <= ' ||NoAtual.Rangeset(DiscAnterior).Maximo;
        EXECUTE IMMEDIATE V_Comando INTO NoDireito.Rangeset(Disc).Minimo;
        FOR i IN 1..Aquisicao.Count LOOP
          IF i <> Disc THEN
            v_comando := 'select max(valor' || i || ') from '
                    ||Tabela ||'_t where valor' || Disc || ' > ' ||chave
                    ||' and Valor'|| DiscAnterior
                    ||' >= '||NoAtual.Rangeset(DiscAnterior).Minimo
                    ||' and Valor'||DiscAnterior ||' <= '
||NoAtual.Rangeset(DiscAnterior).Maximo;
            EXECUTE IMMEDIATE V_Comando INTO NoDireito.Rangeset(i).Maximo;
            v_comando := 'select min(valor' || i || ') from '
                    ||Tabela ||'_t where valor' || Disc || ' > ' ||chave
                    ||' and Valor'|| DiscAnterior
                    ||' >= '||NoAtual.Rangeset(DiscAnterior).Minimo
                    ||' and Valor'||DiscAnterior ||
                    ' <= ' ||NoAtual.Rangeset(DiscAnterior).Maximo;
            EXECUTE IMMEDIATE V_Comando INTO NoDireito.Rangeset(i).Minimo;
          END IF;
        END LOOP;
        v_comando := 'select ROWID from ' || Tabela || ' T where Contador = 0';
        EXECUTE IMMEDIATE V_Comando INTO FilhoDireito_V;
        v_comando := 'update '|| Tabela || ' set Contador = ' || NoDireito.Contador ||
                     ' where ROWID= '' || FilhoDireito_v || '';
        EXECUTE IMMEDIATE V_Comando;
        Rangesets:= '';
        FOR i IN 1..Aquisicao.COUNT LOOP
          Rangesets := Rangesets||'Itemrange_t(' || NoDireito.Rangeset(i).Minimo ||','||
                       NoDireito.Rangeset(i).Maximo || ')';
          IF i <> (Aquisicao.count) THEN
             Rangesets := Rangesets || ',';
          END IF;
        END LOOP;
        v_comando :='update ' || Tabela ||' set Rangeset = Rangeset_t('||Rangesets||')'
                    ||' where ROWID= '' || FilhoDireito_v || '';
        EXECUTE IMMEDIATE V_Comando;
        Rangesets:= '';
        v_comando := 'update '||Tabela||' set FilhoDireito = '' || FilhoDireito_v || '' ||
                ' where ROWID = '' || RoWID_v || '';
        EXECUTE IMMEDIATE V_Comando;
        Rangesets:= '';
      else
       v_comando := 'delete from ' || Tabela || ' where Contador = 0';
```

```
        execute immediate v_comando;
       end if; /* if not naodivide */
        END IF; /* IF NoAtual.Contador = Limite */
        Inseriu := true;
     END IF; /* IF NoAtual.Contador > Limite THEN */
     EXIT WHEN Inseriu;
   END LOOP;
END;
END;
/

CREATE OR REPLACE PROCEDURE
                CriarArvoresIntervalos(Origem varchar2, DIMENSAO number) IS
v_Comando       varchar2(500);
v_comandoaux    varchar2(500);
Tabela          varchar2(50);
Cont            number;
NumLinhas       number;
Valor           number;
TYPE Cursor_t   IS REF CURSOR;
CursorROWID     Cursor_t;
RowId_v         ROWID;
ConjuntoROWID ConjuntoRowid_t := ConjuntoRowid_t();
Conjunto        Conjunto_t      := Conjunto_t();
Faixas          Rangeset_t      := Rangeset_t();
Item            Itemrange_t     := Itemrange_t(0,0);
BEGIN
 v_comando := 'select ROWID from '|| Origem;
 Cont := 0;
 OPEN CursorRowid FOR V_comando;
 FETCH CursorRowId INTO rowid_v;
 WHILE CursorRowid%FOUND LOOP
   Cont := Cont + 1;
   if Cont > ConjuntoRowid.Count then
      ConjuntoRowId.EXTEND;
   end if;
   ConjuntoRowid(Cont) := rowid_v;
   FETCH CursorRowid INTO rowid_v;
 END LOOP;
 CLOSE CursorRowid;
 v_comando := 'select count(*) from '|| Origem;
 execute immediate V_comando into NumLinhas;
 for k in 1..Dimensao loop
   Conjunto.EXTEND;
   Conjunto(k) := 0;
   Faixas.EXTEND;
   Faixas(k) := Item;
 end loop;
 for k in 1..NumLinhas loop
```

```
  Tabela := 'kdtree';
  for j in 1..Dimensao loop
    v_comando:='select Produto'||j||' from '||Origem||' where rowid= '' || ConjuntoRowid(k) || '';
    execute immediate v_comando into valor;
    Conjunto(j) := Valor;
    Tabela := Tabela || Valor;
  end loop;
  Cont := 1;
  for j in 1..Dimensao loop
     v_comandoaux := 'select min(k';
     v_comando := j || '.qte) from ';
     for i in 1..Dimensao loop
        v_comando := v_comando || 'itemvenda_r k' || i;
        if I <> Dimensao then
           v_comando := v_comando || ', ';
        end if;
     end loop;
     v_comando := v_comando || ' where ';
     for i in 1..Dimensao loop
        v_comando := v_comando || 'k' || i || '.coditem = ' || Conjunto(i) || ' and ';
     end loop;
     for i in 2..Dimensao loop
        v_comando := v_comando || 'k1.codvenda = ' || 'k' || i || '.codvenda';
        if I <> Dimensao then
             v_comando := v_comando || ' and ';
        end if;
     end loop;
     v_comandoaux := v_comandoaux || v_comando;
     execute immediate v_comandoaux into Item.Minimo;
     v_comandoaux := 'select max(k';
     v_comandoaux := v_comandoaux || v_comando;
     execute immediate v_comandoaux into Item.Maximo;
     Faixas(Cont) := Item;
     Cont := Cont + 1;
  end loop; /* for j in 1..Dimensao loop */
  kdtree_t.CriaArvore(Tabela, Dimensao,faixas);
  v_comando := 'insert into temp values (' || Dimensao || ','' || Tabela || '')';
  execute immediate v_comando;
 end loop;/* for k in 1..NumLinhas loop */
END;
```

Anexo 3

Implementação do *Apriori*

```
CREATE OR REPLACE PROCEDURE
        APRIORI_R(SupMin NUMBER, ConfMin NUMBER) IS
v_comando        varchar2(200);
C1               varchar2(200);
C2               varchar2(200);
Origem           varchar2(20);
i                number;
Cont             number;
NumLinhas        number;
codvenda         number;
coditem          number;
Valor            number;
TotalTransacoes  number;
TYPE Cursor_t    IS REF CURSOR;
Cursor1          Cursor_t;
Cursor2          Cursor_t;
CursorROWID      Cursor_t;
ConjuntoTransacao Conjunto_t     := Conjunto_t();
ConjuntoCandidato Conjunto_t;
ConjuntoROWID    ConjuntoRowid_t := ConjuntoRowid_t();
ExisteConjunto   boolean;
Rowid_v          rowid;
BEGIN
if (SupMin > 1 or Supmin <=0) or (ConfMin > 1 or Confmin <=0) then
  dbms_output.put_line (' Suporte e Confianca devem estar na seguinte faixa: 0 < Valor <= 1');
else
NO_T.CriaArvore('ARVCONJ');
C1 := 'SELECT codvenda FROM venda_r';
C2 := 'SELECT coditem FROM itemvenda_r where codvenda = ';
GerarC1(C1,C2,1); i := 1;
v_comando := 'select count(*) from venda_r';
execute immediate v_comando into TotalTransacoes;
v_comando := 'select count(*) from c'|| i || '_t';
execute immediate V_comando into NumLinhas;
dbms_output.put_line('Candidados de tamanho 1 : ' || NumLinhas);
v_comando := 'delete from c' ||i||'_t where ((Suporte/'||TotalTransacoes||')*100) <'||
(SupMin*100);
EXECUTE IMMEDIATE v_comando;
v_comando := 'select count(*) from c' || i || '_t';
execute immediate V_comando into NumLinhas;
Cont := 1;
WHILE (Cont <>0) and (NumLinhas <> 0) LOOP
```

```
 I := i + 1;
 Origem := 'C'||(i-1)||'_t';
 GerarCandidatos(Origem,i);
 Origem := 'C' || (i-1) || '_t';
 InserirGrandesConjuntos('ArvConj',Origem, i-1);
 Origem := 'C'|| i ||'_t';
 if I > 2 then   Poda Apriori ('ArvConj',Origem,i); end if;
v_comando := 'select count(*) from '|| Origem;
execute immediate V_comando into NumLinhas;
dbms_output.put_line('Candidados de tamanho ' || i || ' : ' || NumLinhas);
 OPEN Cursor1 FOR C1;
 FETCH Cursor1 INTO codVenda;
 WHILE Cursor1%FOUND LOOP
   v_comando := C2 || codVenda;
   ConjuntoTransacao := Conjunto_t();
   Cont := 0;
   OPEN Cursor2 FOR v_comando;
   FETCH Cursor2 INTO coditem;
   WHILE Cursor2%FOUND LOOP
     Cont := Cont + 1;
     ConjuntoTransacao.EXTEND;
     ConjuntoTransacao(Cont) := 0;
     ConjuntoTransacao(Cont) := codItem;
     FETCH Cursor2 INTO coditem;
   END LOOP;
   CLOSE Cursor2;
   v_comando := 'select count(*) from '|| Origem;
   execute immediate V_comando into NumLinhas;
   ConjuntoCandidato := Conjunto_t();
   for k in 1..i loop
      ConjuntoCandidato.EXTEND;
      ConjuntoCandidato(k) := 0;
   end loop;
   v_comando := 'select ROWID from '|| Origem;
   Cont := 0;
   OPEN CursorRowid FOR V_comando;
     FETCH CursorRowId INTO rowid_v;
     WHILE CursorRowid%FOUND LOOP
       Cont := Cont + 1;
       if Cont > ConjuntoRowid.Count then
         ConjuntoRowId.EXTEND;
       end if;
       ConjuntoRowid(Cont) := rowid_v;
       FETCH CursorRowid INTO rowid_v;
     END LOOP;
   CLOSE CursorRowid;
   for k in 1..NumLinhas loop
      for j in 1..i loop
       if Numlinhas <> 0 then
```

```
          v_comando:='select Produto'||j||' from '||Origem||' where rowid= '''||ConjuntoRowid(k)||
''';
            execute immediate v_comando into valor;
            ConjuntoCandidato(j) := Valor;
          end if;
         end loop;
         Cont := 0;
            FOR l in 1..ConjuntoCandidato.Count LOOP
               FOR J IN 1..ConjuntoTransacao.Count LOOP
                  if ConjuntoCandidato(l) = ConjuntoTransacao(J) then
                    Cont := Cont + 1;
                  end if;
               END LOOP;
            END LOOP;
            if Cont = ConjuntoCandidato.Count then
               v_comando:='select Suporte from '||Origem||' where rowid= '''||ConjuntoRowid(k)
|| '';
               execute immediate v_comando into Valor;
               Valor := Valor + 1;
                v_comando:='update C'||i||'_t set Suporte ='||
                            Valor||' where rowid= ''' ||ConjuntoRowid(k)|| '';
               execute immediate v_comando;
             end if;
     end loop;
     FETCH Cursor1 INTO codVenda;
   END LOOP;
   CLOSE Cursor1;
   v_comando:='delete from c'||i||'_t where ((Suporte/'||TotalTransacoes||')*100) <'||
(SupMin*100);
   EXECUTE IMMEDIATE v_comando;
   v_comando := 'select count(*) from C' || i || '_t';
   execute immediate v_comando into Cont;
END LOOP;
  v_comando := 'drop table C' || i || '_t';
  execute immediate v_comando;
GerarRegras('ArvConj', 'Item_r', ConfMin, TotalTransacoes);
end if;
END;

CREATE OR REPLACE PROCEDURE PodaApriori
                            (Arvore varchar2, Li varchar2, Tamanho number)IS
  valor               number;
  i                    NUMBER := 0;
  k                    number;
  j                    number;
  Numlinhas          number;
  Cont                 number;
  v_Comando         varchar2(800);
  TYPE Cursor_t    IS REF CURSOR;
```

```
  CursorROWID    Cursor_t;
  Achou          boolean;
  RowId_v        ROWID;
  Conjunto       Conjunto_t          := Conjunto_t();
  ConjuntoROWID ConjuntoRowid_t := ConjuntoRowid_t();
BEGIN
  v_comando := 'select count(*) from '|| Li;
  execute immediate V_comando into NumLinhas;
  for k in 1..(Tamanho-1) loop
     Conjunto.EXTEND;
     Conjunto(k) := 0;
  end loop;
  v_comando := 'select ROWID from '|| Li;
  Cont := 0;
  OPEN CursorRowid FOR V_comando;
     FETCH CursorRowId INTO rowid_v;
     WHILE CursorRowid%FOUND LOOP
        Cont := Cont + 1;
        if Cont > ConjuntoRowid.Count then
             ConjuntoRowId.EXTEND;
        end if;
        ConjuntoRowid(Cont) := rowid_v;
        FETCH CursorRowid INTO rowid_v;
     END LOOP;
  CLOSE CursorRowid;
   v_comando := 'select count(*) from '|| Li;
  execute immediate V_comando into NumLinhas;
  if NumLinhas <> 0 then
  for i in 1..NumLinhas loop
     for k in 1..(Tamanho-2) loop
       Cont := 1;
       for j IN 1..Tamanho LOOP
         if j <> k then
           v_comando:='select Produto'|| j ||' from ' || Li|| ' where rowid= '' ||ConjuntoRowid(i)|| '';
           execute immediate v_comando into valor;
           Conjunto(Cont) := Valor;
           Cont := Cont + 1;
         end if;
       END LOOP;
       Cont := 0;
       Achou := NO_T.PesquisaConj(Arvore,Conjunto);
       if not achou then
          v_comando:='delete from ' || Li ||' where rowid= '' || ConjuntoRowid(i) || '';
          execute immediate v_comando;
       end if;
     end loop;
  end loop;
 end if;
 END;
```

Anexo 4

Implementação do *Apriori* Quantitativo

```
CREATE OR REPLACE PROCEDURE
        APRIORIQuant_R(SupMin NUMBER, ConfMin NUMBER, LimiteKdTree number)
IS

v_comando        varchar2(200);
C1               varchar2(200);
C2               varchar2(200);
Origem           varchar2(20);
Arvore           varchar2(20);
i                number;
Cont             number;
NumLinhas        number;
codvenda         number;
coditem          number;
Valor            number;
TotalTransacoes  number;
TYPE Cursor_t    IS REF CURSOR;
Cursor1          Cursor_t;
Cursor2          Cursor_t;
CursorROWID      Cursor_t;
ConjuntoTransacao  Conjunto_t      := Conjunto_t();
ConjuntoQte        Conjunto_t      := Conjunto_t();
ConjuntoCandidato  Conjunto_t;
ConjuntoROWID    ConjuntoRowid_t := ConjuntoRowid_t();
Rowid_v          rowid;
BEGIN
if (SupMin > 1 or Supmin <=0) or (ConfMin > 1 or Confmin <=0) then
  dbms_output.put_line
    (' Os valores de Suporte e Confianca devem estar na seguinte faixa: 0 < Valor <= 1');
else
NO_T.CriaArvore('ARVCONJ');
C1 := 'SELECT codvenda FROM venda_r';
C2 := 'SELECT coditem FROM itemvenda_r where codvenda = ';
GerarC1(C1,C2,1);
i := 1;
v_comando := 'select count(*) from venda_r';
execute immediate v_comando into TotalTransacoes;
v_comando := 'select count(*) from c'|| i || '_t';
execute immediate V_comando into NumLinhas;
dbms_output.put_line('Candidados de tamanho 1 : ' || NumLinhas);
v_comando := 'create table temp (Dimensao number, Tabela varchar2(50))';
execute immediate v_comando;
```

```
v_comando:= 'delete from c'||i||'_t where ((Suporte/' || TotalTransacoes||')*100) <'||
(SupMin*100);
EXECUTE IMMEDIATE v_comando;
v_comando := 'select count(*) from c' || i || '_t';
execute immediate V_comando into NumLinhas;
Cont := 1;
WHILE (Cont <>0) and (NumLinhas <> 0) LOOP
  I := i + 1;
  Origem := 'C'||(i-1)||'_t';
  GerarCandidatos(Origem,i);
  Origem := 'C' || (i-1) || '_t';
  InserirGrandesConjuntos('ArvConj',Origem, i-1);
  Origem := 'C'|| i ||'_t';
  if I > 2 then
    Poda Apriori Quant('ArvConj',Origem,i, SupMin, TotalTransacoes);
    v_comando := 'select ROWID from temp where Dimensao = ' || (i-2);
    OPEN CursorRowid FOR V_comando;
    FETCH CursorRowId INTO rowid_v;
    WHILE CursorRowid%FOUND LOOP
      v_comando := 'select tabela from temp where rowid = '' || rowid_v ||'';
      execute immediate v_comando into Arvore;
      v_comando := 'drop table ' || Arvore;
      execute immediate v_comando;
      v_comando := 'drop table ' || Arvore || '_t';
      execute immediate v_comando;*/
      v_comando := 'delete from temp where rowid = '' || rowid_v ||'';
      execute immediate v_comando;
      FETCH CursorRowid INTO rowid_v;
      END LOOP;
    CLOSE CursorRowid;
  end if;
  CriarArvoresIntervalos(Origem, i);
  v_comando := 'select count(*) from '|| Origem;
  execute immediate V_comando into NumLinhas;
  dbms_output.put_line('Candidados de tamanho ' || i || ' : ' || NumLinhas);
  OPEN Cursor1 FOR C1;
  FETCH Cursor1 INTO codVenda;
  WHILE Cursor1%FOUND LOOP
    v_comando := C2 || codVenda;
    ConjuntoTransacao := Conjunto_t();
    Cont := 0;
    OPEN Cursor2 FOR v_comando;
    FETCH Cursor2 INTO coditem;
    WHILE Cursor2%FOUND LOOP
      Cont := Cont + 1;
      ConjuntoTransacao.EXTEND;
      ConjuntoTransacao(Cont) := 0;
      ConjuntoTransacao(Cont) := codItem;
      FETCH Cursor2 INTO coditem;
```

```
      END LOOP;
      CLOSE Cursor2;
       v_comando := 'select count(*) from '|| Origem;
      execute immediate V_comando into NumLinhas;
      ConjuntoCandidato := Conjunto_t();
      ConjuntoQte := Conjunto_t();
      for k in 1..i loop
         ConjuntoCandidato.EXTEND;
         ConjuntoCandidato(k) := 0;
         ConjuntoQte.EXTEND;
         ConjuntoQte(k) := 0;
      end loop;
      v_comando := 'select ROWID from '|| Origem;
      Cont := 0;
      OPEN CursorRowid FOR V_comando;
        FETCH CursorRowId INTO rowid_v;
        WHILE CursorRowid%FOUND LOOP
          Cont := Cont + 1;
          if Cont > ConjuntoRowid.Count then
            ConjuntoRowId.EXTEND;
          end if;
          ConjuntoRowid(Cont) := rowid_v;
          FETCH CursorRowid INTO rowid_v;
        END LOOP;
      CLOSE CursorRowid;
      for k in 1..NumLinhas loop
         Arvore := 'kdtree';
         for j in 1..i loop
          if Numlinhas <> 0 then
           v_comando:='select Produto'||j||' from '||Origem||' where rowid= '''||ConjuntoRowid(k)||
'''';
            execute immediate v_comando into valor;
            ConjuntoCandidato(j) := Valor;
            Arvore := Arvore || Valor;
          end if;
         end loop;
         Cont := 0;
            FOR l in 1..ConjuntoCandidato.Count LOOP
               FOR J IN 1..ConjuntoTransacao.Count LOOP
                  if ConjuntoCandidato(l) = ConjuntoTransacao(J) then
                    Cont := Cont + 1;
                    v_comando:=' select qte from itemvenda_r where coditem ='
                                ||ConjuntoTransacao(J) || ' and codvenda = ' || codvenda;
                    execute immediate v_comando into ConjuntoQte(l);
                  end if;
               END LOOP;
            END LOOP;
           if Cont = ConjuntoCandidato.Count then
```

```
                v_comando:='select Suporte from '|| Origem ||' where rowid=
'"||ConjuntoRowid(k)|| "";
                execute immediate v_comando into Valor;
                Valor := Valor + 1;
                v_comando:='update C'||i||'_t set Suporte ='
                                ||Valor||' where rowid= '" ||ConjuntoRowid(k)|| "";
             execute immediate v_comando;
             kdtree_t.InsereNo(Arvore,i,LimiteKdtree,ConjuntoQte);
              end if;
      end loop;
      FETCH Cursor1 INTO codVenda;
    END LOOP;
    CLOSE Cursor1;
    v_comando := 'select ROWID from C' || i || '_t where((Suporte/' || TotalTransacoes
              || ')*100) <'  || (SupMin*100);
    OPEN CursorRowid FOR V_comando;
    FETCH CursorRowId INTO rowid_v;
    WHILE CursorRowid%FOUND LOOP
      Arvore := 'kdtree';
      for j in 1..i loop
        v_comando := 'select Produto'||j||' from C'||i||'_t where rowid='"||rowid_v||"";
        execute immediate v_comando into Valor;
        Arvore := Arvore || Valor;
      end loop;
      v_comando := ' drop table ' || Arvore;
      execute immediate v_comando;
      v_comando := ' delete from temp where tabela = '" || Arvore || "";
      execute immediate v_comando;
      Arvore := Arvore || '_t';
      v_comando := ' drop table ' || Arvore;
      execute immediate v_comando;
      FETCH CursorRowid INTO rowid_v;
    END LOOP;
    CLOSE CursorRowid;
    v_comando:='delete from c'||i||'_t where ((Suporte/'||TotalTransacoes||')*100) <'||
(SupMin*100);
    EXECUTE IMMEDIATE v_comando;
    v_comando := 'select count(*) from C' || i || '_t';
    execute immediate v_comando into Cont;
END LOOP;
   execute immediate v_comando;
   v_comando := 'select ROWID from temp';
   OPEN CursorRowid FOR V_comando;
   FETCH CursorRowId INTO rowid_v;
   WHILE CursorRowid%FOUND LOOP
      v_comando := 'select tabela from temp where rowid = '" || rowid_v ||"";
      execute immediate v_comando into Arvore;
      v_comando := 'drop table ' || Arvore;
      execute immediate v_comando;
```

```
    v_comando := 'drop table ' || Arvore || '_t';
    execute immediate v_comando;
    FETCH CursorRowid INTO rowid_v;
  END LOOP;
  CLOSE CursorRowid;
  v_comando := 'drop table temp';
  execute immediate v_comando;
GerarRegras('ArvConj', 'Item_r', ConfMin, TotalTransacoes);
end if;
END;
```

CREATE OR REPLACE PROCEDURE PodaAprioriQuant
(Origem varchar2, Li varchar2, Tamanho number, MinSup number, TotalTrans number)IS

```
  valor       number;
  i           NUMBER := 0;
  k           number;
  j           number;
  w           number;
  Cont        number;
  OveSup      number;
  NumLinhas number;
  SupPMin    number := 0;
  SomaSuporte       number := 0;
  v_Comando         varchar2(800);
  v_Comando2        varchar2(300);
  Arvore            varchar2(50);
  Pmin              varchar2(50);
  FolhaL            kdtree_t;
  FolhaSub          kdtree_t;
  TYPE Cursor_t     IS REF CURSOR;
  CursorROWID       Cursor_t;
  CursorROWIDSub Cursor_t;
  Achou             boolean;
  RowId_v           ROWID;
  Conjunto          Conjunto_t          := Conjunto_t();
  SubConjuntos      Conjunto_t          := Conjunto_t();
  ConjuntoROWID ConjuntoRowid_t := ConjuntoRowid_t();
  ConjPMin          Conjunto_t          := Conjunto_t();
  ConjSub           Conjunto_t          := Conjunto_t();
BEGIN
  or k in 1..(Tamanho-1) loop
    Conjunto.EXTEND;
    Conjunto(k) := 0;
  end loop;
  v_comando := 'select ROWID from '|| Li;
  Cont := 0;
  OPEN CursorRowid FOR V_comando;
    FETCH CursorRowId INTO rowid_v;
    WHILE CursorRowid%FOUND LOOP
```

```
      Cont := Cont + 1;
      if Cont > ConjuntoRowid.Count then
           ConjuntoRowId.EXTEND;
      end if;
      ConjuntoRowid(Cont) := rowid_v;
      FETCH CursorRowid INTO rowid_v;
   END LOOP;
CLOSE CursorRowid;
v_comando := 'select count(*) from '|| Li;
execute immediate V_comando into NumLinhas;
for i in 1..NumLinhas loop
   for k in 1..(Tamanho-2) loop
     Cont := 1;
     for j IN 1..Tamanho LOOP
       if j <> k then
         v_comando:='select Produto'||j||' from '||Li|| ' where rowid= ''' || ConjuntoRowid(i) || '''';
         execute immediate v_comando into valor;
         Conjunto(Cont) := Valor;
         Cont := Cont + 1;
       end if;
     END LOOP;
     Cont := 0;
     Achou := NO_T.PesquisaConj(Origem,Conjunto);
     if not achou then
        v_comando:='delete from ' || Li ||' where rowid= ''' || ConjuntoRowid(i) || '''';
        execute immediate v_comando;
     end if;
   end loop;/* for k in 1..(Tamanho-2) loop */
end loop;   /* for i in 1..ConjuntoRowid.Count loop */
v_comando := 'select ROWID from '|| Li;
ConjuntoRowId := ConjuntoRowid_t();
OPEN CursorRowid FOR V_comando;
FETCH CursorRowId INTO rowid_v;
WHILE CursorRowid%FOUND LOOP
  ConjuntoRowId.EXTEND;
  ConjuntoRowid(Conjuntorowid.Count) := rowid_v;
  FETCH CursorRowid INTO rowid_v;
END LOOP;
CLOSE CursorRowid;
for i in 1..ConjuntoRowid.Count loop
     Arvore := 'kdtree';
     SubConjuntos := Conjunto_t();
     SupPMin := 1000000;
     for k in 1..Tamanho loop
       for j IN 1..Tamanho LOOP
         if j <> k then
           v_comando:='select Produto'||j||' from '||Li||' where rowid= ''' || ConjuntoRowid(i) || '''';
           execute immediate v_comando into valor;
           SubConjuntos.EXTEND;
```

```
      SubConjuntos(SubConjuntos.Count) := Valor;
      Arvore := Arvore || Valor;
    end if;
  END LOOP;
      v_comando:='select max(Contador) from ' || Arvore;
      execute immediate v_comando into valor;
      if Valor < SupPMin then
        Pmin := Arvore;
        SupPMin := Valor;
        ConjPMIN := Conjunto_t();
        Cont := Subconjuntos.Count;
        for j in 1..(Tamanho-1) loop
          ConjPMIN.EXTEND;
          ConjPMin(ConjPMin.COunt) := Subconjuntos(Cont);
          Cont := Cont - 1;
        end loop;
        Cont := ConjPMin.Count;
        for j in 1..ConjPMin.count loop
          Conjunto.EXTEND;
          Conjunto(Conjunto.COunt) := ConjPMin(Cont);
          Cont := Cont - 1;
        end loop;
        ConjPMin := Conjunto;
        Conjunto := COnjunto_t();
  end if;
  Arvore := ('kdtree');
end loop;/* for k in 1..(Tamanho) loop */
v_comando:='select ROWID from '|| Pmin ||
                  ' where FilhoDireito is null and FilhoEsquerdo is null';
OPEN CursorRowid FOR V_comando;
FETCH CursorRowId INTO rowid_v;
WHILE CursorRowid%FOUND LOOP
  v_comando := 'select value(T) from ' || PMin || ' T where rowid = '' ||rowid_v|| '';
  EXECUTE IMMEDIATE V_Comando INTO FolhaL;
  OveSup := FolhaL.Contador;
  k:= 1;
  Arvore := 'kdtree';
  ConjSub := Conjunto_t();
  for j in 1..SubConjuntos.Count loop
    Arvore := Arvore || Subconjuntos(j);
    ConjSub.EXTEND;
    ConjSub(ConjSub.COunt) := Subconjuntos(j);
    if (Arvore <> Pmin) then
      if (k = Tamanho-1) then
        v_comando := 'select ROWID from '|| Arvore ||
                  ' where FilhoDireito is null and FilhoEsquerdo is null';
        OPEN CursorRowidSub FOR V_comando;
          FETCH CursorRowIdSub INTO rowid_v;
        WHILE CursorRowidSub%FOUND LOOP
```

```
              v_comando := 'select value(T) from ' ||Arvore||  ' T where rowid = '" ||rowid_v|| ""';
              EXECUTE IMMEDIATE V_Comando INTO FolhaSub;
              for w in 1..(Tamanho-1) loop
               for z in 1..(Tamanho-1) loop
                 if ConjSub(w) = ConjPMin(z) then
                     if (FolhaL.Rangeset(z).Minimo <= FolhaSub.Rangeset(w).Maximo) and
                      (FolhaSub.Rangeset(w).Minimo <= FolhaL.Rangeset(z).MAximo) then
                      SomaSuporte := SomaSuporte + FolhaSub.Contador;
                    end if;
                 end if;
               end loop;
              end loop;
              FETCH CursorRowIdSub INTO rowid_v;
            END LOOP;
            k := 1;
               Arvore := 'kdtree';
               ConjSub := Conjunto_t();
                  if OveSup > SomaSuporte then
                    OveSup := SomaSuporte;
                  end if;
                  SomaSuporte := 0;
             else
               k:= K + 1;
             end if;
           else
             if k = (Tamanho-1) then
               Arvore := 'kdtree';
               ConjSub := Conjunto_t();
               k := 1;
               SomaSuporte := 0;
             end if;
        end if; /*  if Arvore <> Pmin then */
      end loop; /* for j in 1..SubConjuntos.Count loop */
      if OveSup = 0 then
        SupPMin := SupPMin - FolhaL.Contador;
      end if;
      FETCH CursorRowid INTO rowid_v;
    END LOOP;
    CLOSE CursorRowid;
    if ((SupPMin/TotalTrans)*100) < (MinSup * 100) then
      for k in 1..tamanho loop
        V_comando:='select Produto'||k||' from ' ||Li|| ' where rowid= '"||ConjuntoRowid(i)|| ""';
        execute immediate v_comando into numlinhas;
      end loop;
      v_comando:='delete from ' || Li || ' where rowid= '" || ConjuntoRowid(i) || ""';
      execute immediate v_comando;
    end if;
  end loop;   /* for i in 1..ConjuntoRowid.Count loop */
END;
```

Anexo 5

Rotinas Comuns *Apriori* e *Apriori* Quantitativo

```
CREATE SEQUENCE SQ_Linha
INCREMENT BY 1
START WITH 1
MAXVALUE 99999999999999999
NOCACHE
NOCYCLE;

CREATE OR REPLACE PROCEDURE GERABASE
            (NumTrans number, MaxItens number, MaxQte number) IS
  DescItem    varchar2(30);
  v_comando varchar2(500);
  ItensTransAtual number;
  ItemAtual       number;
  Qte               number;
  ItensAdquiridos Conjunto_T := Conjunto_t();
  Achou     boolean;
  numlinha number;
BEGIN
  v_comando := 'delete from itemvenda_r';
  execute immediate v_comando;
  v_comando := 'delete from item_r';
  execute immediate v_comando;
  v_comando := 'delete from venda_r';
  execute immediate v_comando;
  v_comando := 'delete from item_or';
  execute immediate v_comando;
  v_comando := 'delete from venda_or';
  execute immediate v_comando;
  for i in 1..NumTrans loop
    IF I = 1 THEN
      for j in 1..MaxItens loop
        v_comando := 'select descricao from TabItem where codItem = ' || j;
        execute immediate v_comando into DescItem;
        v_comando := 'insert into item_r values( ' || j || ',' ||""|| DescItem ||"" || ')';
        execute immediate v_comando;
        v_comando := 'insert into item_or values( ' || j || ',' ||""|| DescItem ||""|| ')';
        execute immediate v_comando;
      end loop;
    END IF;
    v_comando := 'insert into venda_r values( ' || i || ')';
    execute immediate v_comando;
    v_comando := 'insert into venda_or values( ' || i || ',linhasitens_t())';
```

```
    execute immediate v_comando;
    v_comando := 'select random.rand_max(' || MaxItens || ') from dual';
    execute immediate v_comando into ItensTransAtual;
    Achou := false;
    for J in 1..ItensTransAtual loop
      v_comando := 'select random.rand_max(' || MaxItens || ') from dual';
      execute immediate v_comando into ItemAtual;
      for k in 1..ItensAdquiridos.Count loop
        if ItensAdquiridos(k) = ItemAtual then
          Achou := true;
          while Achou loop
            Achou := false;
            v_comando := 'select random.rand_max(' || MaxItens || ') from dual';
            execute immediate v_comando into ItemAtual;
            for l in 1..ItensAdquiridos.COunt loop
              if ItensAdquiridos(l) = ItemAtual then
                 Achou := true;
              end if;
            end loop;
          end loop;
        end if;
      end loop;
      ItensAdquiridos.EXTENd;
      ItensAdquiridos(ItensAdquiridos.COunt) := ItemAtual;
    end loop; /* J in 1..ItensTransAtual */
    for j in 1..ItensAdquiridos.Count loop
       v_comando := 'select random.rand_max(' || MaxQte || ') from dual';
       execute immediate v_comando into Qte;
       v_comando := 'insert into itemvenda_r values (' || i ||','|| ItensAdquiridos(j)|| ',' || Qte || ')';
       execute immediate v_comando;
        SELECT SQ_Linha.NEXTVAL INTO NumLinha FROM DUAL;
       v_comando:='INSERT INTO TABLE
                        (SELECT V.LINHASITENS FROM VENDA_OR V WHERE
                          V.NUMVENDA =' || i ||')
                          SELECT '||numlinha||','||ItensAdquiridos(j)||','||Qte||' FROM dual';
       execute immediate v_comando;
    end loop;
    ItensAdquiridos := Conjunto_t();
  end loop; /* i in 1..NumTrans */
END;

CREATE OR REPLACE PROCEDURE GerarC1
          (CursorVenda varchar2, CursorItens varchar2, Tipo number) IS
TYPE Cursor_t IS REF CURSOR;
Cursor1      Cursor_t;
Cursor2      Cursor_t;
codvenda     number;
coditem      number;
Temp         number;
```

```
Cont         number;
v_comando    varchar2(200);
ROWID_ATUAL   ROWID;
ConjItem     Conjunto_t := Conjunto_t();
ConjSup      Conjunto_t := Conjunto_t();
Posicao      number;
Achou        boolean;
BEGIN
V_Comando := 'Create table C1_t
                    (Produto1 number, Suporte number)TABLESPACE APRIORI';
  EXECUTE IMMEDIATE V_Comando;
  OPEN Cursor1 FOR CursorVenda;
    FETCH Cursor1 INTO codVenda;
    WHILE Cursor1%FOUND LOOP
      v_comando := CursorItens || codVenda;
      if Tipo = 2 then
        v_comando := v_comando || ')c';
      end if;
      OPEN Cursor2 FOR v_comando;
      FETCH Cursor2 INTO coditem;
      WHILE Cursor2%FOUND LOOP
        Achou := false;
        if ConjItem.COunt >= 1 then
        for i in 1..ConjItem.Count loop
           if ConjItem(i) = codItem then
             Achou := true;
             Posicao := i;
             exit;
           end if;
        end loop;
        end if;
        if Achou then
          ConjSup(Posicao) := ConjSup(Posicao) + 1;
        else
          ConjItem.Extend;
          ConjSup.Extend;
          ConjItem(ConjItem.Count) := codItem;
          ConjSup(ConjSup.Count)   := 1;
        end if;
        FETCH Cursor2 INTO coditem;
      END LOOP;
      CLOSE Cursor2;
      FETCH Cursor1 INTO codVenda;
    END LOOP;
    CLOSE Cursor1;
    for i in 1..ConjSup.Count loop
      v_comando := 'insert into C1_t values(' || ConjItem(i) || ',' || ConjSup(i) || ')';
      execute immediate v_comando;
    end loop;
```

```
END;

CREATE OR REPLACE PROCEDURE GerarCandidatos
                    (GrandeConjunto varchar2, Tamanho number) IS
v_Comando varchar2(500);
j         number;
BEGIN
v_comando := 'create table C';
 v_comando := v_comando || Tamanho || '_t(';
 for J in 1..Tamanho loop
    v_comando := v_comando || ' Produto' || j || ' number,';
 end loop;
 v_comando := v_comando || 'Suporte number) TABLESPACE APRIORI';
 execute immediate v_comando;
 v_comando := 'insert into C' || Tamanho ||'_t select ';
 for J in 1..(Tamanho-1) loop
   v_comando := v_comando || 'P.Produto' || j;
   v_comando := v_comando || ', ';
 end loop;
 v_comando := v_comando || 'Q.Produto'   || (Tamanho-1) ||',0 from ';
 v_comando := v_comando || GrandeConjunto|| ' P, ' ||GrandeConjunto||' Q ';
 v_comando := v_comando || ' where';
 for J in 1..(Tamanho-2) loop
    v_comando := v_comando ||' P.Produto' || J || '= Q.Produto' || J || ' and ';
 end loop;
 v_comando := v_comando ||' P.Produto' || (Tamanho-1) || ' < Q.Produto' || (Tamanho-1);
 execute immediate v_comando;
END;

CREATE OR REPLACE PROCEDURE GerarRegras
            (Origem varchar2, TabProd varchar2, ConfMin number, TotalTrans number) IS
  ContAntecedente number;
  ContConsequente number;
  Cont            number;
  Confianca       number;
  Indice          number;
  i               number;
  v_Comando       varchar2(500);
  ConjAntecedente Conjunto_t := Conjunto_t(null);
  ConjuntoRegra   Conjunto_t := Conjunto_t(null);
  Temp            Conjunto_t;
  ConjConsequente Conjunto_t;
  Numlinhas       number;
  Suporte         number;
  SupAntecedente  number;
  ROWIDRAIZ       ROWID;
  TYPE Cursor_t   IS REF CURSOR;
  CursorROWID     Cursor_t;
  RowId_v         ROWID;
```

```
 ConjuntoROWID   ConjuntoRowid_t := ConjuntoRowid_t();
 Descprod        varchar2(50);
 TotalRegras     number := 0;
BEGIN
  v_comando := 'select ROWID from '|| Origem || ' where Produto <> 0';
  Cont := 0;
  OPEN CursorRowid FOR V_comando;
    FETCH CursorRowId INTO rowid_v;
    WHILE CursorRowid%FOUND LOOP
       Cont := Cont + 1;
       if Cont > ConjuntoRowid.Count then
            ConjuntoRowId.EXTEND;
       end if;
       ConjuntoRowid(Cont) := rowid_v;
       FETCH CursorRowid INTO rowid_v;
    END LOOP;
  CLOSE CursorRowid;
  v_comando := 'select ROWID from '|| Origem || ' where produto = 0';
  execute immediate V_comando into ROWIDRAIZ;
  v_comando := 'select count(*) from '|| Origem;
  execute immediate V_comando into NumLinhas;
  dbms_output.put_line('-----------------------------------------------------  ');
  for k in 1..(NumLinhas-1) loop
     BuscarConjunto (Origem, ConjuntoRowid(k), ROWIDRAIZ, ConjuntoRegra, Suporte);
     Temp := Conjunto_t();
     Cont := ConjuntoRegra.Count;
     for i in 1..ConjuntoRegra.Count LOOP
        Temp.EXTEND;
        Temp(i) := ConjuntoRegra(Cont);
        Cont := Cont - 1;
     END LOOP;
     ConjuntoRegra := Temp;
     if ConjuntoRegra.Count > 1 then
       Indice := 1;
       WHILE Indice < ConjuntoRegra.Count LOOP
         ContAntecedente := 1;
         ConjAntecedente := Conjunto_t();
         FOR i IN 1..Indice LOOP
           ConjAntecedente.EXTEND;
           ConjAntecedente(ContAntecedente) := ConjuntoRegra(i);
           ContAntecedente := ContAntecedente + 1;
         END LOOP;
            SupAntecedente := NO_T.BuscarSuporte(Origem, ConjAntecedente);
            if SupAntecedente > 0 then
           Confianca := (Suporte/SupAntecedente);
         else
           Confianca := 0;
         end if;
            if Confianca >= ConfMin then
```

```
      FOR I IN 1..(CONTAntecedente-1) LOOP
       v_comando:='select Descricao from '||
                        TabProd||' where codItem = ' || ConjuntoRegra(i);
       execute immediate v_comando into DescProd;
       DBMS_OUTPUT.PUT(DescProd || ' ');
      END LOOP;
      DBMS_OUTPUT.PUT( ' ==> ');
      FOR I IN (ContAntecedente)..(ConjuntoRegra.Count) LOOP
       v_comando:='select Descricao from '||
                        TabProd||' where codItem = ' || ConjuntoRegra(i);
       execute immediate v_comando into DescProd;
       DBMS_OUTPUT.PUT(DescProd || '  ');
      END LOOP;
       DBMS_OUTPUT.NEW_LINE;
       DBMS_OUTPUT.PUT
                    ('SUPORTE    : '||ROUND((SUPORTE/TotalTrans),2) * 100 || ' %     ');
       DBMS_OUTPUT.PUT_LINE
                     ('CONFIANCA  : '||ROUND(CONFIANCA,2) * 100 || ' % ');
       TotalRegras := TotalRegras + 1;
       dbms_output.put_line('------------------------------------------------------------  ');
    end if;
    ContAntecedente := 1;
    ConjAntecedente := Conjunto_t();
    FOR i IN (Indice+1)..ConjuntoRegra.Count LOOP
      if ConjAntecedente.COunt < ContAntecedente then
        ConjAntecedente.EXTEND;
      end if;
      ConjAntecedente(ContAntecedente) := ConjuntoRegra(i);
      ContAntecedente := ContAntecedente + 1;
    END LOOP;
    SupAntecedente := NO_T.BuscarSuporte(Origem, ConjAntecedente);
    if SupAntecedente > 0 then
      Confianca := (Suporte/SupAntecedente);
    else
      Confianca := 0;
end if;
    if Confianca >= ConfMin then
      FOR I IN 1..(CONjAntecedente.Count) LOOP
       v_comando:='select Descricao from '||
                        TabProd||' where codItem = ' || ConjAntecedente(i);
       execute immediate v_comando into DescProd;
       DBMS_OUTPUT.PUT(DescProd || ' ');
      END LOOP;
      DBMS_OUTPUT.PUT( ' ==> ');
      FOR I IN 1..(Indice) LOOP
       v_comando:='select Descricao from '||
                         TabProd||' where codItem = ' || ConjuntoRegra(i);
       execute immediate v_comando into DescProd;
       DBMS_OUTPUT.PUT(DescProd || '  ');
```

```
  END LOOP;
  DBMS_OUTPUT.NEW_LINE;
  DBMS_OUTPUT.PUT
     ('SUPORTE    : '||ROUND((SUPORTE/TotalTrans),2) * 100 || ' %     ');
  DBMS_OUTPUT.PUT_LINE
     ('CONFIANCA  : '||ROUND(CONFIANCA,2) * 100 || ' % ');*/
  TotalRegras := TotalRegras + 1;
  dbms_output.put_line('-----------------------------------------------------  ');
end if;
if ConjuntoRegra.Count >= 2 then
  ContAntecedente := 2;
  while ContAntecedente <= ConjAntecedente.Count LOOP
   Temp := Conjunto_t();
   Cont := 1;
   FOR i IN 1..(ConjAntecedente.Count) LOOP
    if i <> ContAntecedente then
      Temp.EXTEND;
      Temp(Cont) := ConjAntecedente(i);
      Cont := Cont + 1;
    end if;
   END LOOP;
   SupAntecedente := NO_T.BuscarSuporte(Origem, Temp);
   if (SupAntecedente > 0) then
        Confianca := (Suporte/SupAntecedente);
      else
        Confianca := 0;
      end if;
      if Confianca >= ConfMin then
        FOR i IN 1..(Temp.Count) LOOP
       v_comando:='select Descricao from '||TabProd||' where codItem = ' || Temp(i);
          execute immediate v_comando into DescProd;
          DBMS_OUTPUT.PUT(DescProd || ' ');
        END LOOP;
        DBMS_OUTPUT.PUT( ' ==> ');
        ConjConsequente := Conjunto_t();
        ConjConsequente.EXTEND;
        ContConsequente := 1;
        ConjConsequente(1) := ConjAntecedente(ContAntecedente);
        v_comando:='select Descricao from '||TabProd||
                    ' where codItem = ' || ConjAntecedente(ContAntecedente);
        execute immediate v_comando into DescProd;
        DBMS_OUTPUT.PUT(DescProd || ' ');
        FOR i IN 1..Indice LOOP
          v_comando:='select Descricao from '||
                      TabProd||' where codItem = ' || ConjuntoRegra(i);
          execute immediate v_comando into DescProd;
          if ConjuntoRegra(i) <> ConjAntecedente(ContAntecedente) then
            DBMS_OUTPUT.PUT(DescProd || '  ');
            ConjConsequente.EXTEND;
```

```
          ContConsequente := ContConsequente + 1;
          ConjConsequente(ContConsequente) := ConjuntoRegra(i);
        end if;
    END LOOP;
    DBMS_OUTPUT.NEW_LINE;
    DBMS_OUTPUT.PUT
('SUPORTE    : '||ROUND((SUPORTE/TotalTrans),2) * 100 || ' %     ');
    DBMS_OUTPUT.PUT_LINE
('CONFIANCA  : '||ROUND(CONFIANCA,2) * 100 || ' % ');
    TotalRegras := TotalRegras + 1;
    dbms_output.put_line('-----------------------------------------------------  ');
 else
    ConjConsequente := Conjunto_t();
    ConjConsequente.EXTEND;
    ContConsequente := 1;
    ConjConsequente(1) := ConjAntecedente(ContAntecedente);
    FOR i IN 1..Indice LOOP
      if ConjuntoRegra(i) <> ConjAntecedente(ContAntecedente) then
        ConjConsequente.EXTEND;
        ContConsequente := ContConsequente + 1;
        ConjConsequente(ContConsequente) := ConjuntoRegra(i);
      end if;
    END LOOP;
 end if;
Temp := Conjunto_t();
 ContConsequente := ConjConsequente.Count;
 for i in 1..ConjConsequente.Count loop
     Temp.EXTEND;
     Temp(i) := ConjConsequente(ContConsequente);
     ContConsequente := ContConsequente-1;
 end loop;
 ConjConsequente := temp;
 SupAntecedente := NO_T.BuscarSuporte(Origem, ConjConsequente);
           if SupAntecedente > 0 then
    Confianca := (Suporte/SupAntecedente);
 else
    Confianca := 0;
 end if;
 /* Se Confianca >= ConfMin imprime a regra */
 if Confianca >= ConfMin then
    FOR I IN 1..(CONjConsequente.Count) LOOP
      v_comando:='select Descricao from '||
          TabProd||' where codItem = ' || ConjConsequente(i);
      execute immediate v_comando into DescProd;
      DBMS_OUTPUT.PUT(DescProd || ' ');
    END LOOP;
    DBMS_OUTPUT.PUT( ' ==> ');
    FOR I IN 1..(ConjAntecedente.Count) LOOP
      if i <> ContAntecedente then
```

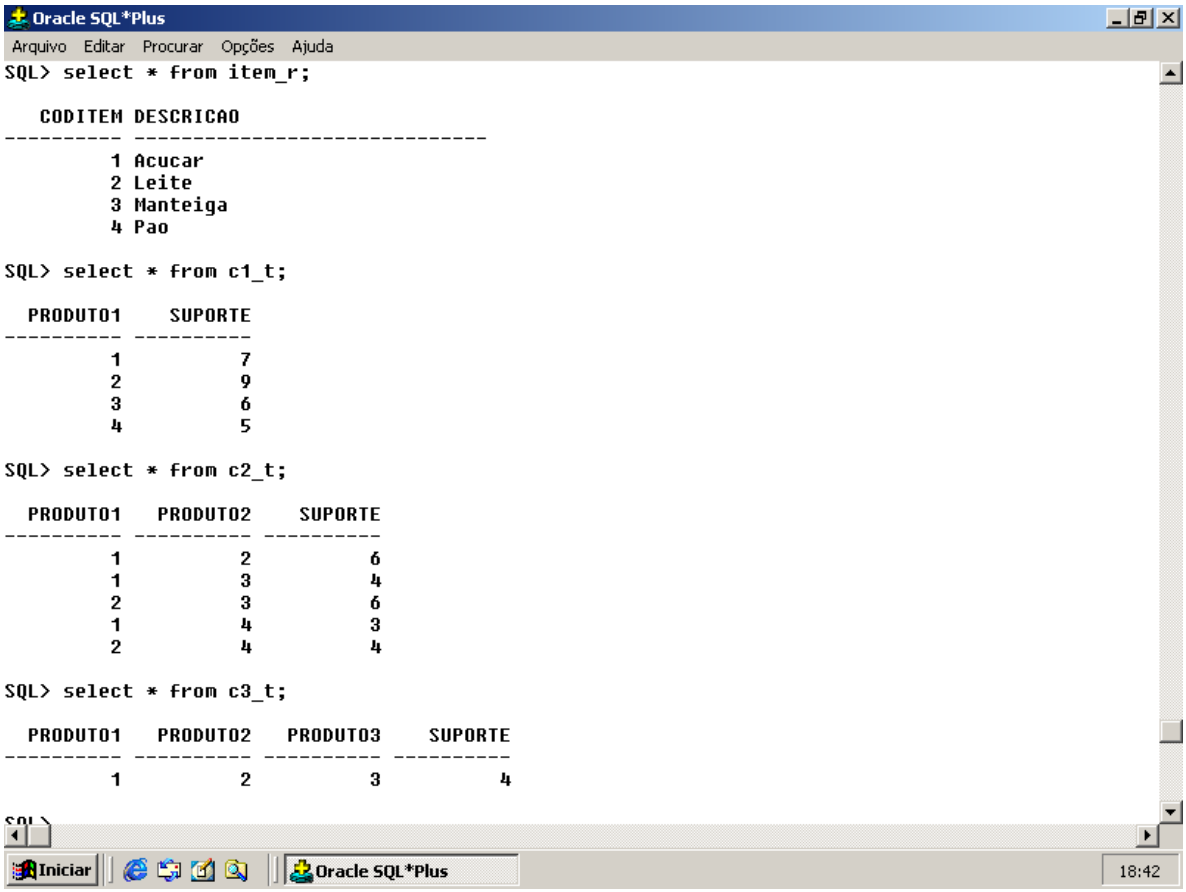
```
                        v_comando:='select Descricao from '||
                          TabProd||' where codItem = ' || ConjAntecedente(i);
                        execute immediate v_comando into DescProd;
                        DBMS_OUTPUT.PUT(DescProd || '  ');
                      end if;
                    END LOOP;
                    DBMS_OUTPUT.NEW_LINE;
                    DBMS_OUTPUT.PUT
                  ('SUPORTE    : '||ROUND((SUPORTE/TotalTrans),2) * 100 || ' %     ');
                    DBMS_OUTPUT.PUT_LINE
                  ('CONFIANCA  : '||ROUND(CONFIANCA,2) * 100 || ' % ');
                    TotalRegras := TotalRegras + 1;
                   dbms_output.put_line('-------------------------------------------------------  ');
                end if;
                  ContAntecedente := ContAntecedente + 1;
              end loop;
          end if;
            Indice := Indice + 1;
          END LOOP;
     end if;
  end loop;
  v_comando := 'drop table ' || Origem;
 execute immediate v_comando;
 dbms_output.put_line('Total de Regras Geradas: ' || TotalRegras);
END;
```

Anexo 6

Algumas Telas da Execução dos Algoritmos

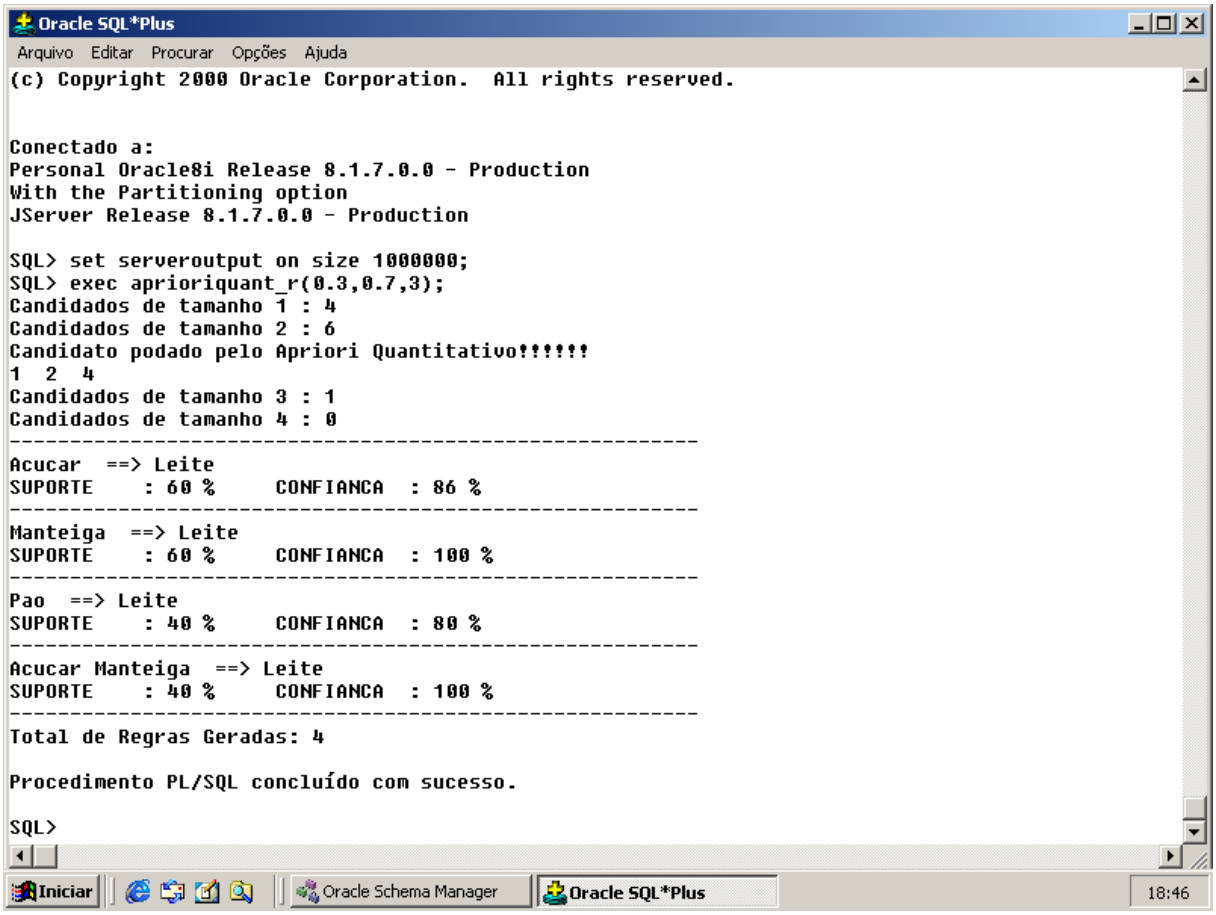
Algumas Telas do Exemplo Passo a Passo do *Apriori* Quantitativo

- Candidatos de Tamanhos 1, 2 e 3:



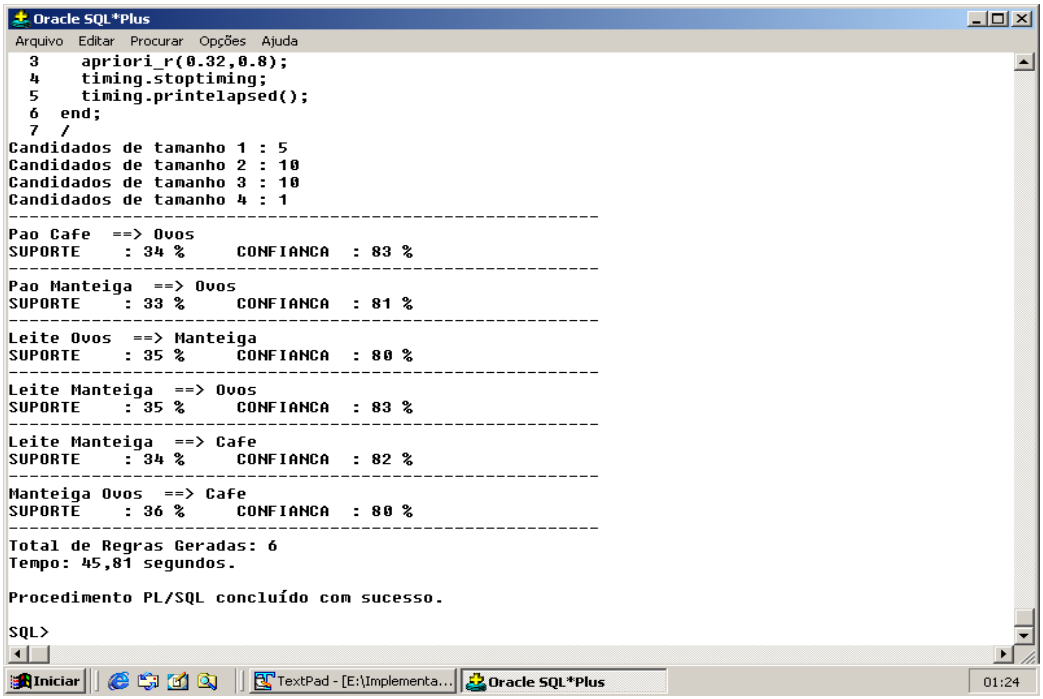
- *Regras para Suporte = 0.3 e Confiança = 0.7.*

  Notar poda do candidato pelo *Apriori* Quantitativo

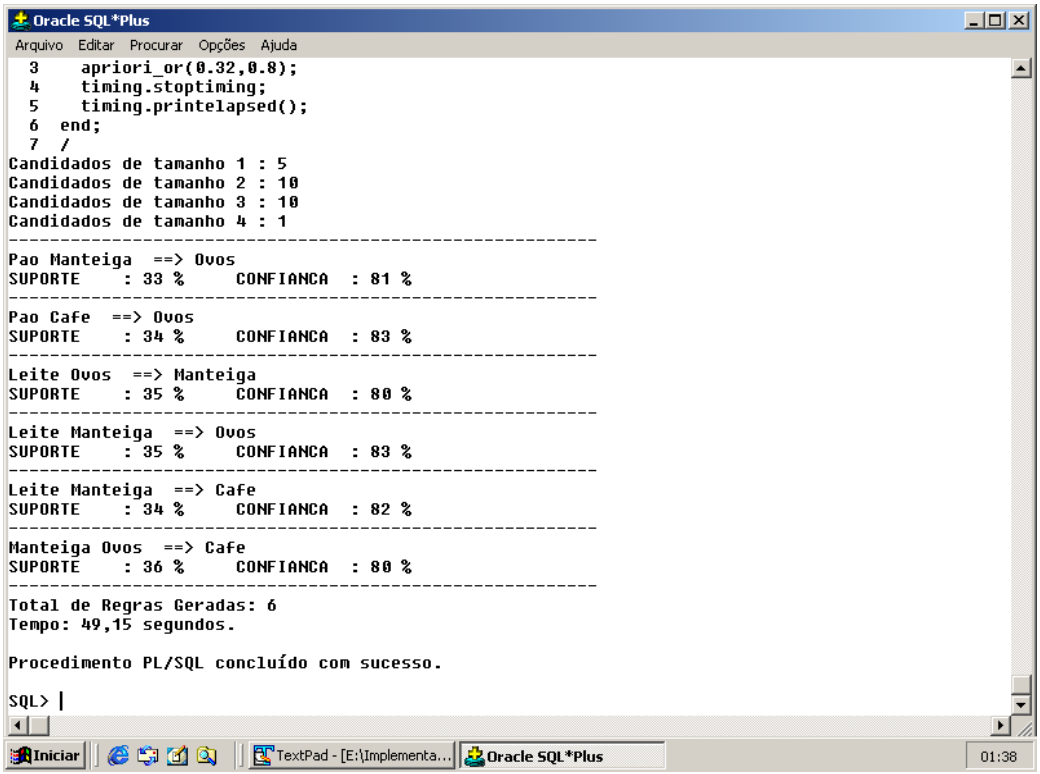


Resultados com base de 500 transações:

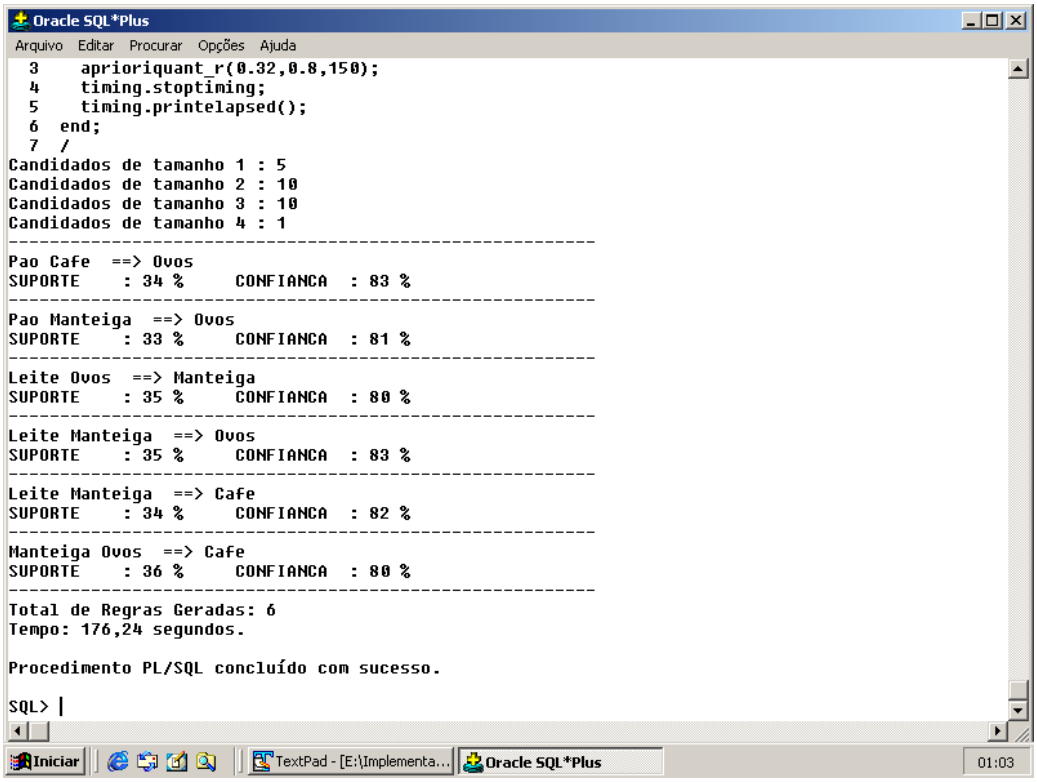
- *Apriori Relacional: Suporte = 0.32 e Confiança = 0.8*



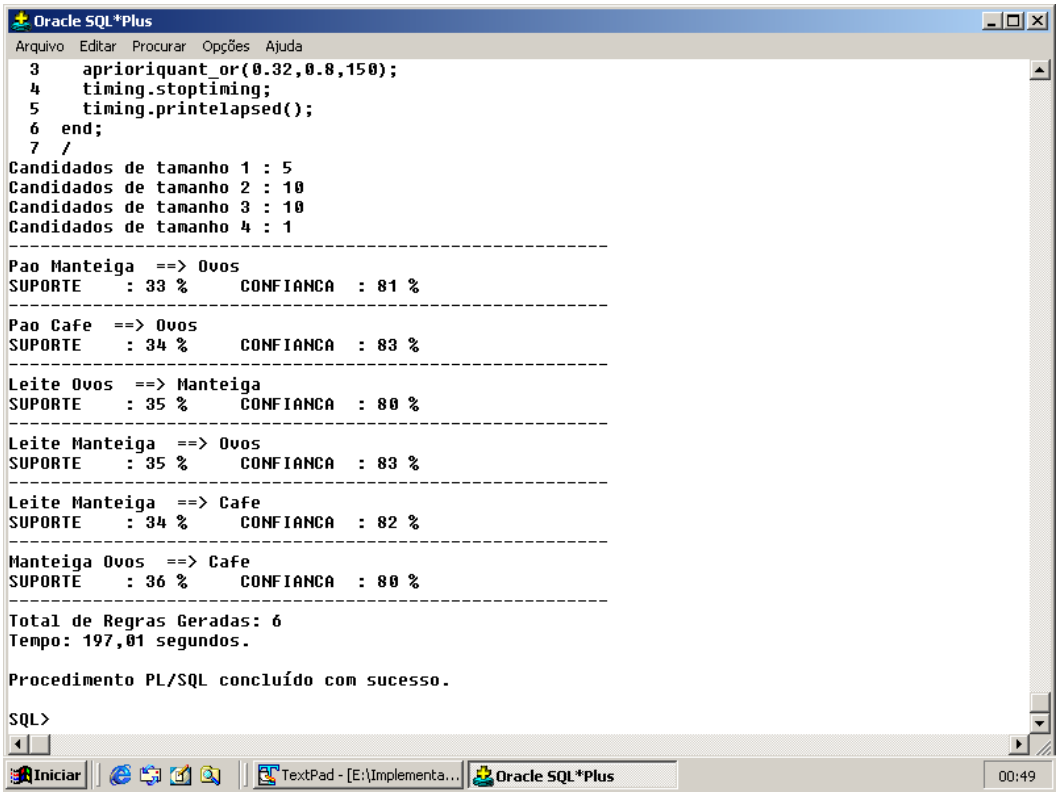
- *Apriori Objeto-Relacional: Suporte = 0.32 e Confiança = 0.8*



- *Apriori Quantitativo Relacional: Suporte = 0.32 e Confiança = 0.8*



- *Apriori Quantitativo Objeto-Relacional: Suporte = 0.32 e Confiança = 0.8*



Referências Bibliográficas

[1] AGRAWAL, Rakesh; IMIELINSKI, T.; SWAMI, A. *Mining Association Rules between Sets of Items in Large Databases*, SIGMOD 5/93, 207-216, Washington, USA, 1993.

[2] AGRAWAL, Rakesh; SRIKANT, Ramakrishnan, *Fast Algorithms for Mining Association Rules*, In: 20th VLDB Conference, 487-498, Santiago, Chile, 1994

[3] CARVALHO, Juliano Varella de; SAMPAIO, Marcus Costa; MONGIOVI, Giuseppe, *Utilização de Técnicas de Data Mining para o Reconhecimento de Caracteres Manuscritos*, In: XIV Simpósio Brasileiro de Banco de Dados, 235-249, Florianópolis, Santa Catarina, Brasil, 1999.

[4] POSSAS, B; MEIRA W.; CARVALHO, M.; RESENDE, R.*Using Quantitative Information for Efficient Association Rule Generation*. XV SBBD, João Pessoa-PB, 2000, 361-374.

[5] AGRAWAL, Rakesh; SHIN, K.. *Developing Tightly-coupled Data Mining Applications on a Relational Database System*. In Proc. Of the 2nd Int'l Conference on Knowledge Discovery in databases and data Mining, Portland, Oregon, August, 1996.

[6] SARAWAGI S.; THOMAS S.; AGRAWAL R. *Integrating association rule mining with relational database systems: Alternatives and implications*. Proc. ACM-SIGMOD, 1998.

[7] GARCIA-MOLINA H; ULLMAN J.; WIDOM J. *Database System Implementation*. Prentice-Hall, 2000

[8] BENTLEY J.. *Multidimensional Binary Search Trees Used for Associative Searching*. In Communications of ACM, September, 1975.

[9] WITTEN Ian H; EIBE Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*, Morgan Kaufmann Publishers, 2000.

[10] *ORACLE* CORPORATION. *Oracle 8i Data Cartridge Developer's Guide*. Release 8.1.5.

[11] SARAWAGI S.; AGRAWAL R. *Integrating Association Rule Mining with Relational Database Systems: Alternatives and implications*. Research Report RJ 10107 (91923), IBM Almaden Research Center, San Jose, CA 95120, March 1998.

Available from http://www.almaden.ibm.com/cs/quest.

[12] POSSAS B; RUAS F; MEIRA W; RESENDE R. *Geração de regras de Associação Quantitativas*. XIV Simpósio Brasileiro de Banco de Dados, 1999.

[13] RAJAMANI K; IYER B.; COX A; CHADHA A.. *Efficient Mining for Association Rules with Relational database Systems.* Proceedings of the International Database Engineering and Applications Symposium (IDEAS'99), August 1999.

[14] CARVALHO, Juliano Varella de; SAMPAIO, Marcus Costa; MONGIOVI, Giuseppe, *Utilizando Técnicas de Data Mining para o Reconhecimento de Caracteres Manuscritos*, Dissertação de Mestrado, Universidade Federal da Paraíba, Brasil, Fevereiro, 2000.

[15] ONODA, Maurício; EBECKEN, Nelson F. F., *Implementação em JAVA de um Algoritmo de Árvore de Decisão Acoplado a um SGBD Relacional*, In: XV Simpósio Brasileiro de Banco de Dados, Rio de Janeiro, RJ, Brasil, 2001.

[16] *ORACLE* CORPORATION, *Oracle 9i Data Mining. Oracle* Data Sheet. Fevereiro, 2002.

[17] *ORACLE* CORPORATION, *Oracle 9i Data Mining*, *Oracle* Technical White Paper, December, 2001.

[18] URMAN, Scott, *Oracle 8i Pl/SQL Programming, The Essential Guide for Every Oracle Programmer*, Osborne/McGraw-Hill, 1997.

[19] URMAN, Scott, *Oracle 8i Advanced Pl/SQL Programming, Build Powerful, Web-Enabled PL/SQL Applications*, Osborne/McGraw-Hill, 2000.

[20] AURÉLIO, Marco; VELLASCO, Marlley; LOPES, Carlos Henrique, *Descoberta de Conhecimento e Mineração de Dados*, ICA – Laboratório de Inteligência Computacional Aplicada, Departamento de Engenharia Elétrica, PUC–Rio

[21] *ORACLE* CORPORATION, *Oracle8i Designing and Tuning for Performance*, Release 2 (8.1.6), December, 1999.

[22] *ORACLE* CORPORATION, *Oracle9i Application Developer's Guide - Object-Relational Features*, Release 1 (9.0.1)

[23] *ORACLE* CORPORATION, *Oracle9i Database Performance Guide and Reference*, Release 1 (9.0.1).

[24] *ORACLE* CORPORATION, *Oracle8i SQL Reference*, Release 3 (8.1.7).

[25] BEZERRA, E. et al. *An Analysis of the Integration between Data Mining Applications and Database Systems*. In: INTERNATIONAL CONFERENCE ON DATA MINING, 2nd, 2000, Cambridge-UK. *Proceedings.* Cambridge: WIT Press, 2000. p. 151-160.

[26] *ORACLE* CORPORATION, *Oracle9i Data Mining Concepts Release 9.0.1*, June 2001 Part No. A90389-01.

[27] AGRAWAL, Rakesh; IMIELINSKI, T.; SWAMI, A. *Database Mining: A performance perspective.* In IEEE Transactions on Knowledge and Data Engineering, December 1993.

[28] AGRAWAL, R, ; IMIELINSKI, T.; SWAMI, A. "*Mining Associations between Sets of Items in Massive Databases*", *Proc. of the ACM-SIGMOD 1993 Int'l Conference on Management of Data*, Washington D.C., May 1993, 207-216

[29] AGRAWAL, Rakesh; SRIKANT, R. *Mining quantitative Association Rules in Large Relational Tables.* In Proceedings of the ACM SIGMOD, June 1996