

OPERADORES VISANDO À OTIMIZAÇÃO DE PRECISÃO NO CÁLCULO DE
SÉRIES TEMPORAIS

Bruno Portella de Aguiar Grieco

Tese de Doutorado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Doutor em Engenharia de Sistemas e Computação.

Orientadores: Nelson Maculan Filho
Marta Lima de Queirós Mattoso

Rio de Janeiro
Março de 2012

OPERADORES VISANDO À OTIMIZAÇÃO DE PRECISÃO NO CÁLCULO DE
SÉRIES TEMPORAIS

Bruno Portella de Aguiar Grieco

TESE SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ
COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA
UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS
NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM
ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

Prof. Nelson Maculan Filho, H.D.R.

Profa. Marta Lima de Queirós Mattoso, D.Sc.

Prof. José Manoel de Seixas, D.Sc.

Prof. Alexandre de Assis Bento Lima, D.Sc.

Prof. Luidi Gelabert Simonetti, D.Sc.

Prof. Luiz Satoru Ochi, D.Sc.

RIO DE JANEIRO, RJ - BRASIL

MARÇO DE 2012

Grieco, Bruno Portella de Aguiar

Operadores Visando à Otimização de Precisão no
Cálculo de Séries Temporais / Bruno Portella de Aguiar

Grieco. – Rio de Janeiro: UFRJ/COPPE, 2012.

XIV, 190 p.: il.; 29,7 cm.

Orientadores: Nelson Maculan Filho,

Marta Lima de Queirós Mattoso.

Tese (doutorado) – UFRJ/ COPPE/ Programa de
Engenharia de Sistemas e Computação, 2012.

Referências Bibliográficas: p. 169-174.

1. Séries Temporais. 2. Banco de Dados. 3.
Operadores. 4. Álgebra Relacional. I. Universidade
Federal do Rio de Janeiro, Maculan, et al. II.
Universidade Federal do Rio de Janeiro, COPPE,
Programa de Engenharia de Sistemas e Computação. III.
Título.

Dedicatória

Às minhas filhas, Laura e Helena;

À minha esposa, Bettina;

Agradecimentos

Ao Prof. Maculan, pelo voto de confiança em mim, por tudo que aprendi.

À Profa. Marta, pela objetividade e paciência.

À minha família, por todos os momentos difíceis nos quais a sacrifiquei.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

OPERADORES VISANDO À OTIMIZAÇÃO DE PRECISÃO NO CÁLCULO DE SÉRIES TEMPORAIS

Bruno Portella de Aguiar Grieco

Março/2012

Orientadores: Nelson Maculan Filho
Marta Lima de Queirós Mattoso.

Programa: Engenharia de Sistemas e Computação

Esta tese apresenta uma série de operadores que podem ser adicionados, como extensão, integrada a uma álgebra relacional de modo a aprimorar a gerência e manipulação de dados correspondentes a séries temporais oriundas de experimentos científicos. A principal vantagem de uma solução no nível algébrico é permitir que consultas sejam especificadas em alto nível e otimizadas algebricamente pelo sistema de banco dados. Os operadores foram definidos formalmente, implementados em um protótipo de sistema de banco de dados e explorados em um estudo de caso com dados reais. A completude e correteza destes operadores é validada por meio do Secondo, um sistema voltado para permitir extensões à álgebra relacional. Secondo possui um ferramental que verifica a compatibilidade entre os operadores da álgebra relacional e os operadores sendo propostos. Além da verificação teórica, o ferramental do Secondo permite a efetivação dos novos operadores em um protótipo de sistema de banco de dados relacionais visando à validação prática do novo conjunto integrado de operadores. Experimentos mostraram, de fato, que o comportamento dos mesmos corresponde às expectativas em se trabalhar com dados de séries temporais ao mesmo tempo em que se opera com dados textuais operados com a álgebra relacional tradicional. Por fim, um caso real é apresentado e tratado com estes novos recursos, mostrando que os mesmos possuem uma aplicabilidade prática.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

OPERATORS FOR PRECISION OPTIMIZATION ON THE CALCULUS OF TIME
SERIES

Bruno Portella de Aguiar Grieco

March/2012

Advisors: Nelson Maculan Filho
Marta Lima de Queirós Mattoso.

Department: Systems and Computing Engineering

This work presents a set of operators that may be used as an extension of a Relational Algebra as a method for optimizing the management of data derived from scientific experiments in the form of Time Series. The soundness and correctness of those operators are proved by a practical implementation (prototype) that shows that the behavior of those operators actually correspond to the expectations. Finally, a real case study is presented and treated with those new tools, showing, at last, that they do present a practical benefit.

SUMÁRIO

1	INTRODUÇÃO	1
1.1	CONTEXTO.....	2
1.2	PROBLEMAS.....	5
1.3	PROBLEMAS ESPECÍFICOS.....	7
1.3.1	<i>Questões de Modelagem e Gerência</i>	7
1.3.2	<i>Arquivos Binários</i>	8
1.4	SOLUÇÕES EXISTENTES	9
1.4.1	<i>Análise das Soluções Existentes</i>	10
1.5	OBJETIVO.....	12
1.5.1	<i>Consistência e Completude da Álgebra</i>	13
1.6	CONSIDERAÇÕES ADICIONAIS.....	14
1.6.1	<i>Operações Matemáticas</i>	14
1.7	ORGANIZAÇÃO DOS CAPÍTULOS.....	16
2	BANCO DE DADOS	18
2.1	MODELAGEM.....	18
2.2	DETALHAMENTO DO PROBLEMA	19
2.2.1	<i>Armazenamento e Consulta</i>	19
2.2.2	<i>Procedimentos de Análise</i>	21
2.2.3	<i>Precisão</i>	23
2.2.4	<i>Operações com incertezas diferentes</i>	26
3	SOLUÇÕES EXISTENTES E PROPOSTAS	28
3.1	MOTIVAÇÃO	28
3.2	SOLUÇÕES EXISTENTES	30
3.2.1	<i>COUGAR</i>	30
3.2.2	<i>Bancos de Dados Ativos</i>	31
3.2.3	<i>Proficy Historian</i>	31
3.2.4	<i>DIAdem</i>	33
3.2.5	<i>Historis</i>	34
3.2.6	<i>kdb+</i>	34
3.2.7	<i>SciDB</i>	35
3.3	ALTERNATIVAS.....	37
3.3.1	<i>Workflows</i>	38
3.3.2	<i>SECONDO</i>	39
3.3.3	<i>Metadados</i>	40
4	SECONDO: AMBIENTE DE EXTENSÃO DA ÁLGEBRA RELACIONAL PARA OPERADORES CIENTÍFICOS.....	43
4.1	CRITÉRIOS DE ESCOLHA DA FERRAMENTA DE INTEGRAÇÃO.....	44
4.1.1	<i>Escolha do SECONDO como plataforma</i>	45
4.2	ASSINATURAS DE SEGUNDA-ORDEM.....	46
4.2.1	<i>Definição de Naturezas e Tipos</i>	47
4.2.2	<i>Definição de Operadores</i>	49
4.2.3	<i>Polimorfismo e Sobrecarga de Operadores</i>	50
4.3	ESTRUTURA DO SISTEMA	51
4.4	UTILIZANDO O SECONDO.....	54
4.4.1	<i>Conceito de Stream</i>	58
5	OPERADORES PROPOSTOS PARA SÉRIES TEMPORAIS	63
5.1	SÉRIES TEMPORAIS	64
5.1.1	<i>Sinais, Períodos de Tempo e Amostragem</i>	64
5.1.2	<i>Definições de séries temporais</i>	66
5.1.3	<i>Operadores para Séries Temporais</i>	67
5.1.4	<i>Consistência dos Operadores</i>	69
5.2	ADEQUAÇÃO AO MODELO RELACIONAL.....	70
5.2.1	<i>Tipo TimeSpan</i>	70

5.2.2	<i>Limitações Práticas</i>	71
5.2.3	<i>Tipo RealArray e operadores Trans-Relacionais</i>	72
5.2.4	<i>Valores Indefinidos</i>	73
5.2.5	<i>Operadores Trans-Relacionais Semânticos</i>	73
6	IMPLEMENTAÇÃO DA PROPOSTA	74
6.1	TIPO REALARRAY	74
6.1.1	<i>Operadores Aritméticos</i>	74
6.1.2	<i>Operadores Booleanos</i>	76
6.1.3	<i>Operadores Aritméticos Unários Horizontais</i>	78
6.1.4	<i>Utilização em Tuplas</i>	79
6.1.5	<i>Operadores Aritméticos Unários Verticais</i>	80
6.2	OPERADORES TRANS-RELACIONAIS	85
6.2.1	<i>Operador column</i>	85
6.2.2	<i>Operador uncolumn</i>	87
6.2.3	<i>Operador transpose</i>	89
6.2.4	<i>Operador untranspose</i>	91
6.2.5	<i>Operador window</i>	95
6.3	CARACTERÍSTICAS ALGÉBRICAS DOS OPERADORES TRANS-RELACIONAIS	101
6.3.1	<i>Proposta de Validação Algébrica</i>	101
6.3.2	<i>Conceito de Para-Extensionalidade</i>	101
6.4	SEMÂNTICA TEMPORAL	106
6.4.1	<i>Adequação à Semântica Temporal</i>	106
6.4.2	<i>Implementação do tipo TimeSpan</i>	106
6.4.3	<i>Operadores do tipo TimeSpan</i>	109
6.4.4	<i>Indexação de TimeSpan com R-Trees</i>	110
6.5	APLICAÇÃO DA SEMÂNTICA TEMPORAL	114
6.5.1	<i>Sincronização</i>	114
6.5.2	<i>Operadores de Pré-Sincronização</i>	114
6.5.3	<i>Operadores de Extração de Intervalos</i>	117
6.5.4	<i>Operadores Trans-Relacionais Semânticos</i>	119
6.5.5	<i>Operador Iuncolumn</i>	123
6.5.6	<i>Operador Icolumn</i>	125
6.5.7	<i>Operadores Itranspose e Iuntranspose</i>	126
6.5.8	<i>Operador Iwindow</i>	128
7	ESTUDO DE CASO	135
7.1	PROPOSTA.....	135
7.2	PRÓLOGO.....	135
7.3	EXPLORAÇÃO OFFSHORE.....	138
7.3.1	<i>Engenharia Subsea</i>	139
7.4	RISERS FLEXÍVEIS.....	140
7.4.1	<i>Engenharia de Risers Flexíveis</i>	140
7.4.2	<i>Qualificações de Linhas</i>	142
7.4.3	<i>Pesquisa e Desenvolvimento</i>	143
7.5	TESTE DE UMA AMOSTRA	144
7.5.1	<i>Workflow de Análise de Dados</i>	147
7.5.2	<i>Informações Relevantes</i>	149
7.5.3	<i>Análise Holística dos Dados</i>	150
7.5.4	<i>Hiato entre Engenharia e TI</i>	151
7.6	APLICAÇÃO DA PROPOSTA AO PROBLEMA	152
7.6.1	<i>Elementos do Estudo de Caso</i>	152
7.6.2	<i>Objetivo</i>	152
7.6.3	<i>Escopo e Limitações</i>	153
7.6.4	<i>Importação dos Dados</i>	154
7.6.5	<i>Manipulação dos Dados</i>	154
7.6.6	<i>Exportação dos Dados</i>	157
7.6.7	<i>Reanálise dos Dados</i>	157
7.6.8	<i>Operações com o modelo relacional</i>	158
7.6.9	<i>Conclusão</i>	160

8	CONCLUSÃO.....	161
8.1	CONTRIBUIÇÃO DESTE TRABALHO.....	161
8.2	RECAPITULAMENTO.....	161
8.3	ESCOPO FINAL.....	162
8.4	TEMAS FUTUROS.....	164
8.4.1	<i>Otimização da Precisão dos Resultados Algébricos.....</i>	<i>164</i>
8.4.2	<i>Permitir a Configuração de Campos com Incertezas Diferentes.....</i>	<i>165</i>
8.4.3	<i>Modelagem Hierárquica do Experimento.....</i>	<i>165</i>
8.4.4	<i>Extensão da Linguagem SQL.....</i>	<i>166</i>
8.5	INCORPORAÇÃO DA ÁLGEBRA RELACIONAL ESTENDIDA EM OUTROS SGBDS.....	169
	REFERÊNCIAS.....	171
	APÊNDICE A. AUTOMAÇÃO INDUSTRIAL.....	177
A.1	HISTÓRICO.....	177
A.1.1	<i>Supervisão, Controle e Monitoramento.....</i>	<i>180</i>
A.1.2	<i>Monitoramento.....</i>	<i>183</i>
A.1.3	<i>Componentes da Automação Industrial.....</i>	<i>185</i>
A.2	PROCESSAMENTO DA INFORMAÇÃO.....	188
A.2.1	<i>Interligação de Tecnologias e Protocolos de Comunicação.....</i>	<i>188</i>
A.2.2	<i>Precisão e Incerteza.....</i>	<i>191</i>
A.2.3	<i>Valores de Data e Hora.....</i>	<i>193</i>

LISTA DE FIGURAS

Figura 1 - Exemplo de filtro com janela móvel.....	22
Figura 2 - Exemplo do conceito de Banda-Morta.....	32
Figura 3 - Exemplo de "Ragged Array" do SciDB. Extraído de [20].....	36
Figura 4 – Diagrama básico de construtores de tipos	48
Figura 5 – Diagrama para o operador +	49
Figura 6 - Arquitetura do Sistema SECONDO.....	52
Figura 7 – Interface gráfica JavaGUI do sistema Secondo	55
Figura 8 – Interface com o sistema através de Terminal	56
Figura 9 – Exemplo de consulta à tabelas (relações).....	58
Figura 10 - Exemplo de transação utilizando <i>Streams</i>	59
Figura 11 - Exemplo de utilização dos operadores feed, count e consume	60
Figura 12 - Operadores Aritméticos para RealArray.....	75
Figura 13 - Exemplo de operadores booleanos.....	77
Figura 14 - Operadores Aritméticos Unários.....	79
Figura 15 - Exemplo de Relação com atributos RealArray.....	80
Figura 16 - Exemplo de utilização de operadores verticais	81
Figura 17 - Exemplo de comutatividade de operadores horizontais e verticais	82
Figura 18 - Comutatividade da média em relação à transposição	83
Figura 19 - Funcionamento do operador column	86
Figura 20 – Resultado da utilização do operador column.....	86
Figura 21 - Funcionamento do operador uncolumn.....	87
Figura 22 – Uso do operador uncolumn	88
Figura 23 - Funcionamento do operador transpose.....	90
Figura 24 – Uso do operador transpose	91
Figura 25 - Funcionamento do operador untranspose	92
Figura 26 – Uso do operador untranspose.....	93
Figura 27 – Uso do operador untranspose com maior número de colunas.....	94
Figura 28 – Uso do operador untranspose em tabelas não-transpostas	94
Figura 29 – Criação da tabela RelAT com comando untranspose.....	95
Figura 30 - Exemplo de uma média móvel sobre uma senóide.....	96
Figura 31 - Conceito de Janela Móvel	98
Figura 32 - Funcionamento do operador window.....	98
Figura 33 – Uso do operador window com janela Fechada/Fechada.....	99
Figura 34 – Uso do operador window com janela Fechada/Aberta	99
Figura 35 – Uso do operador window com janela Aberta/Fechada	100
Figura 36 – Uso do operador window com janela Aberta/Aberta	100
Figura 37 - Criação de tipos TimeSpan	108
Figura 38 - Operadores de intervalo para o tipo TimeSpan	110
Figura 39 - Tabela relTS	111
Figura 40 - Exemplos de utilização de uma RTree.....	112
Figura 41 - Exemplo de re-amostragem com patamares.....	116
Figura 42 - Exemplo de re-amostragem com interpolação linear	116
Figura 43 - Extração de elementos de uma série temporal.....	117
Figura 44 - Utilização dos operadores lextract e lextractspan.....	118
Figura 45 - Tabela sensorDATA	122
Figura 46 - Exemplo de uso do operador luncolumn.....	124
Figura 47 - Tabela sensor2DATA	124
Figura 48 - Exemplo de uso do operador lcolumn	126
Figura 49 - Exemplo de uso do operador ltranspose	127
Figura 50 - Exemplo de uso do operador luntranspose	128
Figura 51 - Resultado do operador IWindow com janela fechada em ambas as extremidades.....	131

Figura 52 - Resultado do operador Iwindow com janela aberta em ambas as extremidades	132
Figura 53 - Exemplo do operador Iwindow com janela Aberta/Fechada	133
Figura 54 - Exemplo do operador Iwindow com janela Fechada/Aberta	134
Figura 55 - Evolução da Profundidade de Exploração de Petróleo no Brasil (até 2003)	137
Figura 56 – Exemplos de tipos de plataformas	138
Figura 57- Configuração submarina de exploração (<i>Field Layout</i>)	139
Figura 58 - Estrutura comum de um <i>Riser</i> Flexível.....	140
Figura 59 - Linha Flexível Enrolada.....	142
Figura 60 - Plataforma de Produção	145
Figura 61 - Tela do Programa AqDados - (Extraída do sítio do fabricante).....	147
Figura 62 - Tela do software AqDAnalysis	148
Figura 63 – Visualização das Tensões na Fase de Dano no Matlab.....	158
Figura 64 - Tabela de Eventos	159
Figura 65 - Exemplo de Processo de Fabricação de Cerveja.....	178
Figura 66 - Exemplos de marcadores de pena (gráficos de tendência)	178
Figura 67 - Operação através de painel sinótico.....	179
Figura 68 - Operação por painel Sinótico, Controle por PLC.....	181
Figura 69 - Operação através de sistema SCADA	182
Figura 70 - Teste de Avião em Túnel de Vento	184
Figura 71 - Processo de Conversão Analógico/Digital	186
Figura 72 - Diferentes Taxas de Aquisição	187
Figura 73 - Exemplo de Formatos de Transmissão de Dados.....	190

LISTA DE TABELAS

Tabela 1 - Exemplo de Implementação de Séries Temporais	18
Tabela 2 - Tabela resultante da consulta de dois sinais temporais	20
Tabela 3 - Exemplo de instância de valores A, B e C	24
Tabela 4 - Resultado da consulta anterior.....	25
Tabela 5 - Propriedades algébricas inválidas em aritmética de ponto-flutuante.....	26
Tabela 6 – Polimorfismo do operador Adição.....	51
Tabela 7 – Polimorfismo do operador Positivo	51
Tabela 8 - Tabela com RealArray	72
Tabela 9 – Tabela Trans-Relacionada.....	72
Tabela 10 - Elemento referenciado pelo operador lwindow	130

LISTA DE LISTAGENS

Listagem 1 - Possível consulta a uma base de dados distribuída.....	24
Listagem 2 - Consulta à base de dados de teste	25
Listagem 3 - Comando utilizado para calcular a média do período de cada ciclo	156
Listagem 4 - Comando alternativo para calcular a média do período de cada ciclo ...	168

LISTA DE SIGLAS

ACID	– (do inglês) Atomicidade, Consistência, Isolamento e Durabilidade	
ADC	– Analog to Digital Converter	– Conversor Analógico Digital
AI	– Analog Input	– Entrada Analógica
API	– Application Programming Interface	– Interface de Programação de Aplicações
AO	– Analog Output	– Saída Analógica
BD	– Banco de Dados	
CLP	– Controlador Lógico Programável	
CSV	– Comma Separated Values	– Valores Separados por Vírgulas
DAC	– Digital to Analog Converter	– Conversor Digital Analógico
DBMS	– Database Management System	– SGBD
DI	– Digital Input	– Entrada Digital
DO	– Digital Output	– Saída Digital
GE	– General Electric	
GUI	– Graphical User Interface	– Interface Gráfica com o Usuário
IEEE	–	
JCL	– Job Control Language	– Linguagem de Controle de Tarefas
PID	– Controle Proporcional, Integral, Derivativo	
PIMS	– Plant Information Management System	– Sistema de Informações Gerenciais de Plantas
PLC	– Programmable Logic Controller	– CLP
RTU	– Remote Terminal Unit	– Unidade Remota
SCADA	– Supervisory, Control and Data Acquisition	– Sistema Supervisório e de controle
SDK	– System Development Kit	– Kit de desenvolvimento do Sistema
SGBD	– Sistema de Gerência de Banco de Dados	
SQL	– Structured Query Language	– Linguagem Estruturada de Consultas
TCP/IP	– Transport Control Protocol / Internet Protocol	– Protocolo de acesso à internet
XML	– Extensible Markup Language	– Linguagem de Marcadores Estensível
GIS	– Geographic Information Systems	– Sistema de Informações Geográficas

1 Introdução

Por algum tempo, a pesquisa na área de Banco de Dados (BD) pareceu deixar de lado as aplicações científicas. Este enfoque, de fato, fazia sentido. Aplicações científicas não eram tão sujeitas à manipulação e gerência dos dados. Dados científicos eram obtidos e armazenados em arquivos de forma convencional. Uma das questões científicas que existiam há trinta anos atrás, conforme nos mostra Daini [46], era como armazenar matrizes em memória. Apesar de este trabalho ter sido publicado na SIGMOD, a melhor conferência de BD, ele se diferencia bastante da linha de trabalho de Graefe [7,10,16], cuja preocupação se refletia, basicamente, em como armazenar dados em dispositivos de memória secundária de forma eficiente.

Em contra-partida, as aplicações comerciais foram a força motora da área de BD. Com a disseminação do computador pessoal em empresas e a quantidade de dados gerados aumentando exponencialmente, soluções que permitissem combinar estes dados com as bases já instaladas de sistemas de grande-porte e, por fim, sistemas Web, foram motivos mais do que suficientes para alavancar o desenvolvimento dos Sistemas Gerenciadores de Banco de Dados.

Finalmente, parece ter havido uma revolução semelhante à dos micro-computadores no campo de aplicações científicas. Juntamente com o barateamento do custo dos instrumentos sensores, o micro-computador passou a ser capaz de dar conta da aquisição e coleta de dados, antes realizado apenas por sistemas fechados, também de grande porte. Surgem, então, problemas semelhantes aos das aplicações convencionais, isto é, como armazenar e gerenciar as grandes quantidades de dados gerados pelas ferramentas científicas. A natureza destes dados, a princípio, parece semelhante à das aplicações convencionais, mas uma análise um pouco mais

profunda pode constatar que dados científicos possuem estruturas de representação voltadas ao cálculo científico, como matrizes, que precisam ser manipulados de forma específica. São exemplos, dados de natureza geográfica, astronômica e séries temporais.

A motivação por trás desta tese, foi a inexistência de um sistema de gerência de bases de dados convencionais, que, ao mesmo tempo, contemplasse dados de sinais temporais (SGBST). Para tal, faz-se necessário entender as características próprias deste tipo específico de dado e como ele se diferencia, em forma de representação e operação, dos tipos de dados tratados por bancos de dados ditos convencionais [3].

Como veremos a seguir, surge um problema chamado de “*Impedance Mismatch*” (desalinhamento de impedância), um termo que significa que a estrutura de representação dos dados científicos não é compatível com a estrutura de dados dos sistemas de banco de dados existentes. De fato, ao analisarmos as séries temporais, vemos que elas não se encaixam naturalmente no modelo relacional de banco de dados.

1.1 Contexto

Como se falou anteriormente, percebe-se um hiato entre a evolução dos SGBDs convencionais e as necessidades para fins científicos. Essa seção discute algumas abordagens existentes para o hiato no contexto dos dados de sinais temporais.

Uma abordagem para tratar o de “*Impedance Mismatch*” é adaptar a representação de sinais temporais às estruturas relacionais. Por exemplo, modelar o sinal temporal simplesmente como uma série de valores sequenciais, representado por meio de uma

relação de um tipo numérico qualquer. Este tipo de modelagem, apesar de seu pragmatismo, não prevê as operações típicas de séries temporais que precisam ser feitas com estes dados no futuro. Isto não quer dizer que esta representação impossibilita o tratamento dos dados, mas sim que o trabalho para realizar a operação é deixado *a posteriori*, fora do contexto das operações relacionais do banco de dados. Tais operações precisariam ser programadas em linguagens tradicionais de programação ou em linguagens de rotinas ("*stored procedures*") do banco de dados. Isso implica que cada aplicação terá que realizar o desenvolvimento desta programação. Como consequência, este trabalho poderá ser mais complicado que o usuário final (do sinal temporal em si) possa executar. Um dos motivos é porque o profissional responsável por gerenciar o dado não é o mesmo profissional responsável por analisar o dado, em outras palavras possuem habilidades diferentes.

Outra abordagem encontrada em soluções atuais é decorrente da evolução dos sistemas de Automação Industrial, isto é, sistemas de controles de plantas industriais (fábricas). Esta evolução adicionou aos sistemas já instalados, o uso de bancos de dados e, posteriormente, com a evolução do hardware de controle e o aumento do número de dados coletados, surgiu a necessidade de uma análise automática dos mesmos. Porém, a recuperação dos dados para análise não ocorre de forma tão simples devido à grande quantidade de dados existentes, principalmente por questões de disponibilidade de memória principal, onde os dados são trabalhados. Portanto, esta é mais uma abordagem que integra dados científicos e convencionais, porém mantém o problema do "*impedance mismatch*".

Assim, é este o problema que deseja-se melhorar, propondo-se uma forma de se trabalhar os dados de forma holística, isto é, como um todo, sem que seja necessária sua recuperação por partes.

Para o entendimento desta tese, será necessário discorrer sobre tópicos de diversas áreas : Automação Industrial (supervisão, controle e monitoramento), Banco de Dados (modelagem e manipulação), Processamento de Sinais (filtros e transformadas), Otimização e, por último, Aritmética Computacional, onde serão mostrados alguns problemas decorrentes da armazenagem e manipulação dos dados.

Serão analisadas algumas formas de modelagem deste dado, desde a mais básica, como um vetor numérico, até o modelo normalizado que poderia ser implementado em um SGBD relacional. Veremos que esta modelagem é muito aberta para ser otimizada pelo processador da álgebra relacional e, na prática, procuram-se outras implementações que visem a um aumento de desempenho nas consultas e operações. Esta escolha acaba por excluir os SGBDs convencionais, pois os mesmos não apresentam uma boa relação custo/benefício no que tange aos seus custos de aquisição, manutenção e gerência.

Por fim, é sugerida uma extensão da álgebra relacional por meio de operadores voltados para o armazenamento e manipulação de sinais temporais. Uma das vantagens dessa abordagem é que podem, posteriormente, ter sua implementação otimizada para competir com as soluções existentes, mantendo ainda um certo grau de abstração. Ou seja, uma implementação completa da proposta pode trazer de volta os SGBDs relacionais como uma opção eficiente e viável.

1.2 Problemas

Conforme foi descrito anteriormente, o problema em si é como prover uma forma eficiente para se armazenar, operar e consultar dados de sinais temporais oriundos de aplicações científicas e, posteriormente, analisá-los, como um todo, junto aos dados convencionais. As operações a serem realizadas podem ser vistas como a alavanca que motiva a pesquisa de uma nova forma de gerência de dados para as séries temporais.

É consenso na literatura consultada [9, 17, 20, 21] que o atual estado-da-arte de SGBDs em pouco ajuda na representação e armazenamento de dados científicos que requerem um formato específico, pois a estrutura de representação da maioria dos dados científicos em si, não se adequa aos modelos existentes.

Consideremos um problema simples como ilustração: Uma tabela é composta por colunas representando diversos sensores; e linhas contendo as diversas leituras destes sensores. Como seria a melhor forma de armazenar esta tabela ? Ora, se observarmos que todas as leituras são simultâneas, armazenar a tabela linha por linha faz muito sentido. Porém, se considerarmos que o conjunto de linhas de cada sensor será recuperado individualmente, o armazenamento por colunas torna-se mais eficiente.

Existem, porém várias propostas para a solução deste problema através de diversas tecnologias. Alguns propõem a utilização de um modelo de *Workflows* aliado à um sistema convencional de bancos de dados para representar a descrição dos dados científicos [42,43,44]. Outros, uma quebra total do paradigma relacional com a proposta de um modelo de banco de dados específico para o problema, como é o

caso do SciDB [20, 21]. Enquanto isto, algumas soluções proprietárias de representação de dados bem específicas, já começam a tomar parte do mercado, como é o caso do Proficy Historian, da GE Fanuc [29,39], e o DiaDEM [38], da National Instruments.

Apesar de as duas últimas soluções apresentadas não utilizarem SGBDs, parece natural que uma solução baseada em um SGBD seja apresentada e bem recebida pela comunidade científica. Isso porque, se por um lado os SGBDs não trabalham bem com estruturas de dados do tipo vetor numérico ou representações binárias, por outro lado, SGBDs trabalham muito bem com dados textuais que complementam os dados científicos. Qualquer solução que isole os dois mundos, ou seja, dados científicos separados dos dados textuais recai no problema do mantém o problema do "impedance mismatch". O uso da solução baseada em extensões de SGBDs traz a vantagem de aproveitar a consolidação da área no que diz respeito à eficiência e generalidade de representação armazenamento e consulta de dados e metadados. Assim, pode se beneficiar de toda a perícia da própria comunidade para sua evolução.

Como não se sabe, *a priori*, o que se deseja fazer com os dados armazenados em um SGBD, este trabalho sugere e implementa uma série de operações para que a gerência de uma base de sinais temporais tenha sentido prático. Não só operações matemáticas, como também operações que trabalhem a estrutura da base em si, através de projeções e filtros (de consulta e matemáticos) de modo a adequar o sinal para consulta.

1.3 Problemas Específicos

Dado que esta tese visa a administrar séries temporais, é natural que comecemos tentando conceituar o que é o Tempo. Infelizmente, a Física moderna ainda não possui uma conceituação definitiva do que é o Tempo, se é uma grandeza contínua ou discreta, uma dimensão ou um fenômeno [82].

Maiores detalhes sobre este assunto são apresentados em capítulos posteriores, porém, para efeito desta tese, consideraremos o Tempo como uma dimensão contínua, mensurável em segundos, e seus múltiplos (horas, dias....) e divisores (milissegundos, nanosegundos...).

Como não podemos tratar computacionalmente uma dimensão contínua¹, é necessário trabalhar apenas com valores amostrados. Portanto, podemos assumir uma série temporal como sendo o valor de uma variável tomados em determinados momentos. Estas amostras podem ser tomadas em intervalos de tempo constantes ou não. Dependendo deste último quesito, a modelagem da série temporal pode apresentar formas diferentes.

1.3.1 Questões de Modelagem e Gerência

Como veremos a seguir, o modelo de séries temporais não se encaixa nas estruturas de representação inerentes ao paradigma relacional. Algumas perguntas que podemos fazer seriam: Como seria um modelo de dados que atendesse a estes requisitos específicos ? Como recuperar estes dados de forma simples ? Como consultar estes dados combinados com dados convencionais de forma simples ? Séries temporais paralelas (adquiridas simultaneamente) devem ser vistas como uma matriz ou como um conjunto de vetores separados ?

¹ Apenas grandezas finitas e enumeráveis são computáveis.

Quanto à interface com o usuário e outras aplicações, seria a linguagem SQL suficiente para suprir esta demanda ? Que extensões podem ser propostas ?

Quanto à forma de representação do tempo, qual o formato mais adequado ?

1.3.2 Arquivos Binários

É verdade que pode-se modelar uma Série Temporal de modo a armazená-la em um SGBD. Na prática, porém, esta solução não é viável, uma vez que não basta armazenar. É preciso manipular e consultar. Porém, mesmo o próprio armazenamento não se mostra viável. Os programas aquisitores de dados podem trabalhar com uma taxa de aquisição muito alta, o que costuma inviabilizar o uso de um SGBD genérico pelo tempo computacional de processar uma transação. O percebido é que os mesmos utilizam arquivos binários como forma de armazenamento, por ser uma opção mais rápida de armazenamento e, portanto, mais segura, pois não implica a perda de dados devido ao tempo de processamento da transação. A manipulação desses arquivos é incompatível com a manipulação de dados por meio da álgebra relacional. Além disso, o uso de arquivos binários produz uma miríade de arquivos, em formatos proprietários, que com frequência só podem ser manipulados pelas ferramentas de análise disponibilizadas pelos fabricantes. Ou seja, a gerência e recuperação de dados destes torna-se mais um problema para o usuário final.

1.4 Soluções Existentes

O armazenamento de séries temporais é um problema que já foi abordado na literatura e até mesmo pelo mercado. Das soluções existentes, difundidas no mercado, podemos ressaltar o Proficy Historian da GE [29] e o DiaDEM da National Instruments [38]. Ambas são voltadas para o mercado de automação industrial e podem ser consideradas como sistemas específicos que modelam as séries temporais através de uma ótica própria. Também foram encontradas duas outras soluções mais abrangentes: o Historis da LIM [58] e o kdb+ da Kx [59]; estas duas últimas soluções ainda não têm a mesma penetração das duas primeiras, porém, todas serão analisadas mais profundamente no decorrer deste trabalho.

Conforme já discutido, apesar de todos os problemas do hiato de representações causando o "Impedance Mismatch", o uso de um SGBD clássico para o armazenamento de séries temporais é a solução corrente disponível. A adequação desta solução às necessidades do problema é feito através de linguagem de programação convencionais, seja como *Storage Procedures*, ou programas que preenchem o caminho entre o dado bruto, aquisitado pelos sensores e seu repositório.

Soluções revolucionárias, recentes, como SciDB [20, 21] ainda não puderam mostrar sua efetividade. O SciDB é um sistema criado especificamente para armazenamento de dados científicos que utiliza o Postgres como base de dados. Assim, tem a vantagem de combinar a representação e o armazenamento de tipos de dados científicos, como matrizes, com dados relacionais em um único ambiente. Entretanto, de acordo com Ailamaki [18], a solução genérica do SciDB, como um banco de dados abarcando todas as aplicações científicas, pode se tornar inviável devido à inexistência de uma estrutura de representação que atenda à diversidade das

aplicações científicas. Assim, ele pode resolver parte do problema, mas ainda vai depender

Uma visão intermediária para o problema seria o uso de um SGBD expansível, voltado para acrescentar novas operações visando a atender uma classe específica de aplicações científicas. Uma solução bem sucedida nesta visão é o SECONDO [23, 24, 25, 26], um sistema que permite expandir um SGBD convencional, no caso o Berkeley DB [52], com uma álgebra relacional de segunda ordem.

1.4.1 Análise das Soluções Existentes

Alguns dos problemas destes três tipos de solução podem ser inferidos de imediato. Soluções proprietárias envolvem altos custos. A forma de acesso aos dados torna-se rígida, o modelo proposto não pode ser alterado, ou seja, torna-se difícil evoluir se o problema não se encaixar na solução, principalmente se a base já estiver instalada e em uso, neste caso corre-se o risco de perder-se o montante já investido na solução. Além disso, a solução acaba por gerar uma dependência em relação ao seu fabricante. Estes também inviabilizam a vantagem de se executar consultas genéricas, utilizando a linguagem SQL, requisitando, também, treinamento adicional de seus usuários.

A adoção de uma solução voltada para modelos convencionais, como os de negócio, implica a criação de toda uma estrutura humana que desenvolva a interface deste sistema com o problema em si e os usuários finais. Todas as estruturas e operações devem ser criadas a partir do zero, e põe-se em dúvida se o resultado não seria o mesmo de se utilizar um sistema proprietário, pois a flexibilidade do sistema final depende exclusivamente de uma boa modelagem do problema aliado a uma visão de expansibilidade. Além disso, reutilizar essas estruturas e operações em diferentes aplicações do mesmo domínio científico não é trivial, o que acarreta em esforço de programação repetitivo e custoso.

Das duas soluções apresentadas como intermediárias, (SGBDs expansíveis) SECONDO e SciDB, este último parece ser, a primeira vista, genérico demais. Pequenos experimentos realizados com a versão beta, obtida por meio de contatos com a equipe do SciDB, mostraram que seus recursos não atendem por exemplo às necessidades de representação e manipulação de sinais temporais. Observou-se ainda que não está disponível documentação suficiente para que possa ser adotado como uma solução viável. Portanto, o SECONDO foi escolhido como base para o desenvolvimento e validação desta tese.

1.5 Objetivo

O objetivo desta tese é apresentar uma estrutura de representação e operações para dados de sinais temporais. Esta abordagem visa a ser instanciada em soluções existentes de forma holística gerando um SGBST. A contribuição é permitir ao gerente realizar as operações em alto nível sem se preocupar com a implementação.

Para tal, foram analisadas as necessidades do usuário final deste sistema, as quais direcionaram as propostas de estruturas de representação para sinais temporais e operações com séries temporais de forma integrada ao modelo relacional.

A solução encontrada foi uma extensão no nível da álgebra relacional, com tipos e operadores que permitem trabalhar com séries temporais de forma integrada e consistente, de modo a prover uma representação mais intuitiva dessas séries, bem como a sua gerência e interface como outros sistemas. A principal vantagem de uma solução no nível algébrico é permitir que consultas sejam especificadas pelo usuário final em alto nível e otimizadas pelo sistema de banco dados. Essa otimização é realizada sobre a consulta expressa de forma algébrica, ao se gerar, automaticamente, por meio de regras, expressões algébricas equivalentes. Essas expressões representam a mesma consulta, mas possuem custos bem diferentes, cabendo ao otimizador escolher dentre as equivalentes, a expressão mais eficiente (ótima). Desta forma, o usuário final se preocupa apenas em especificar o quê e não o como e obtém a resposta da forma mais eficiente possível.

A extensão da álgebra relacional foi feita criando-se novos tipos e operadores. Estes tipos foram chamados de Trans-Relacionais, pois são suscetíveis a operações que “transcendem” o modelo relacional, que não admite relações entre as linhas de uma

tabela. Isto é, os tipos Trans-Relacionais admitem que linhas de uma tabela possuam um relacionamento entre si.

Quanto aos operadores, foram criados operadores lógicos e matemáticos que permitem manipular os tipos; operadores Trans-Relacionais que permitem aplicar este novo conceito aos tipos e, por fim, operadores Trans-Relacionais Semânticos. Este últimos aplicam uma Semântica Temporal ao realizar as operações Trans-Relacionais. Por Semântica Temporal, entende-se que os dados armazenados representam uma série Temporal e, portanto devem atender a alguns quesitos específicos em sua manipulação.

Como operadores matemáticos, podemos supor o de interpolação, que altera uma tabela incluindo valores calculados que não existiam previamente; bem como diversos filtros que incluam janelas móveis, como por exemplo a média móvel. Alguns destes operadores foram implementados de forma direta. Outras operações são obtidas a partir de combinações de operadores.

1.5.1 Consistência e Completude da Álgebra

Dado o objetivo da tese, faz-se necessário uma análise sobre a consistência e completude da nova álgebra apresentada. Algumas abordagens são possíveis para esta análise, por exemplo, uma exclusivamente teórica, envolvendo o formalismo da prova matemática ou uma realizando uma prova de conceito, por meio da análise de compatibilidade com tipos e operações da álgebra relacional, mostrando, de fato, que os operadores comportam-se como deveriam, para as necessidades de manipulação de séries temporais analisados. Optou-se pela segunda abordagem de validação, que é a mais utilizada em extensões da álgebra relacional.

Os operadores foram definidos formalmente, implementados em um protótipo de sistema de banco de dados e explorados em um estudo de caso com dados reais. A completude e correteza destes operadores é validada por meio do Secondo, uma plataforma construída sobre um arcabouço formal de Assinaturas de Segunda Ordem, voltada para propiciar extensões à álgebra relacional. Secondo possui um ferramental que verifica a compatibilidade entre as estruturas e os operadores da álgebra relacional e a extensão sendo propostos. Além da verificação teórica, o ferramental do Secondo permite a efetivação dos novos operadores em um protótipo de sistema de banco de dados relacionais visando à validação prática do novo conjunto integrado de estruturas e operadores. Experimentos mostraram, de fato, que o comportamento dos mesmos corresponde às expectativas em se trabalhar com dados de séries temporais, ao mesmo tempo em que se opera com dados textuais operados com a álgebra relacional tradicional. Por fim, um caso real é apresentado e tratado com estes novos recursos, mostrando que os mesmos possuem uma aplicabilidade prática.

1.6 Considerações Adicionais

Apesar de propostos no exame de qualificação, alguns problemas foram excluídos do escopo do trabalho de forma a manter-se um foco e torná-lo mais conciso. No entanto, estes problemas foram levantados e, portanto, podem ser mencionados como características desejadas em um futuro Sistema de Gerência de Bases Temporais.

1.6.1 Operações Matemáticas

É desejável que sistemas de gerência de bases temporais (SGBSTs) possuam ferramentas que possibilitem análises matemáticas sobre os dados ou, como Cudre-Mauroux et Al. [20], bem coloca, “o cozimento dos dados”.

Suponhamos o seguinte exemplo. Um cliente possui uma conta em um banco com um saldo de R\$ 100,00 no dia 20 de dezembro. No dia 26 de dezembro o cliente realiza um depósito de R\$ 100,00. Portanto seu saldo passa a ser R\$ 200,00. Qual seria o seu saldo em 24 de dezembro ? Esta pergunta tem a resposta óbvia de R\$ 100,00 referente ao último valor depositado. Esta série temporal que representa o saldo é alterada apenas por eventos discretos. Mas suponhamos outro exemplo, uma chaleira cheia d'água está sobre um fogão. Às 15:00 sua temperatura é de 25° Celsius e a chama do fogão é ligada. Às 15:10 ouvimos o sibilar do vapor e sabemos que a sua temperatura é de 100°. Celsius. Qual seria a temperatura às 15:05 ? Esta resposta não é trivial como a primeira. Temos também dois eventos, mas é necessário interpolar o valor entre as duas medições para se estimar o valor correto. À princípio poderíamos estimar 62,5°, através de uma simples interpolação linear, mas uma fórmula mais adequada, levando-se em conta o volume, o calor específico da água e outros dados relevantes poderia, também, ser utilizada.

Isto nos leva a imaginar que a utilização de um sistema de gerência de bases de séries temporais necessita, não somente, apresentar os dados lá armazenados, como também, extrapolar resultados a partir destes dados. Esta extrapolação é necessária, principalmente, se armazenarmos séries temporais com diferentes taxas de amostragem. Em uma consulta, faz-se necessário apresentar um número consistente de valores ao operarmos séries temporais com essas características.

1.7 Organização dos Capítulos

Entende-se que o problema da análise de dados científicos é bastante similar aos problemas existentes na área de Automação Industrial, de modo que o Apêndice A é destinado à apresentação dos problemas desta área e a extensão destes problemas para o problema de armazenamento de sinais no tempo. Esta organização também permite entender melhor o enfoque da escolha de séries temporais como base para o trabalho.

No capítulo 2 são vistas as questões da área de Banco de Dados e sua utilização para análises. Os problemas quanto à modelagem e armazenamento de séries temporais, os requisitos para análises de dados e o problema da precisão que se mencionou anteriormente.

O capítulo 3 analisa as alternativas existentes e apresenta as soluções utilizadas no desenvolvimento da prova de conceito proposta.

O capítulo **Error! Reference source not found.** explica como foi definida a ferramenta utilizada, no caso o SECONDO, quais foram os critérios de escolha e como a ferramenta funciona.

O capítulo 5 apresenta os tipos e os operadores propostos ainda de forma conceitual. É mostrada uma maior conceituação do que é uma Série Temporal e de como ela seria modelada.

O capítulo 6 apresenta a implementação em si. O funcionamento do protótipo mostrando como se trabalha com os diversos operadores propostos no capítulo anterior e como foram adequados a um exemplo prático.

No capítulo 7, é mostrado um estudo de caso prático. Com dados reais, oriundos de um teste de um tubo flexível de petróleo instrumentado. Neste capítulo é apresentado um pequeno resumo sobre o que é o caso estudado e apresentado um resultado que não seria viável sem o uso da solução proposta e implementada.

A conclusão geral do trabalho é mostrada no capítulo 8.

2 Banco de Dados

Neste capítulo serão apresentados os problemas inerentes da utilização de um banco de dados tradicional, isto é, que utiliza o modelo relacional, para o gerenciamento de séries temporais.

Estes problemas não são observados, a priori, pois a modelagem de um sinal temporal é aparentemente trivial, porém, eles podem ser observados quando o sistema implementado não atinge a eficácia esperada.

2.1 Modelagem

Um dos problemas básicos é a conversão do modelo teórico normalizado para um modelo prático que possa ser implementado de forma eficiente. Do ponto de vista teórico, o sinal, ou o valor que se deseja representar, possui dados relativos a identificação do sinal (nome, origem, escala, unidade de engenharia ou moeda) e dados relativos aos valores no tempo. O valor deste sinal em um determinado instante é um atributo desta relação. Ou seja, uma tabela que armazene estes dados terá como atributos, uma forma de relacionamento para a identificação do sinal, a informação de data e hora e o valor do sinal naquele instante. Todos estes atributos são necessários para identificar o sinal.

Tabela 1 - Exemplo de Implementação de Séries Temporais

Planta	Sensor	Time Stamp	Valor
1	1	22/05/2009 14:29:32.00	23.5
1	1	22/05/2009 14:29:33.00	23.5
1	1	22/05/2009 14:29:34.00	23.492
1	2	22/05/2009 14:29:34.10	122.221233
1	1	22/05/2009 14:29:35.00	23.492

Mesmo utilizando-se o modelo teórico de sinais no tempo, verificamos que cada linha da tabela representa um valor do sinal no tempo e não tem relação com a linha anterior ou posterior. Isto por que no modelo relacional, as linhas de uma tabela não têm relação entre si. Porém, em séries temporais isto raramente é verdade. Costuma-se assumir que o valor de um sinal em momento entre duas medições pode ser calculado como uma interpolação entre os dois valores existentes – excluindo-se casos em que se suspeite da taxa de amostragem ou em que se possua alguma informação que refute esta hipótese. Este tipo de heurística, não é contemplado pelo modelo relacional.

Outro problema deriva do fato de sinais diferentes possuírem frequências de aquisição diferentes. Fazer uma consulta que produza um resultado consistente de sinais sincronizados não é tão simples de se escrever.

2.2 Detalhamento do problema

A gerência de sinais temporais torna-se um problema complexo quando observamos as questões de modelagem usando-se como base de armazenamento o sistema relacional e como saída o resultado de análises sobre os dados armazenados. Como foi apresentado anteriormente o modelo relacional, podemos ver que este apresentará graves problemas de implementação quanto ao espaço a ser alocado. Da mesma forma, veremos que a falta de relação entre as linhas de uma tabela será incompatível com possíveis transformadas ou filtragens que seriam úteis aos usuários.

2.2.1 Armazenamento e Consulta

Quanto à questão do armazenamento, como a frequência em que o sinal é adquirido costuma não variar e a identificação do sinal repete-se a cada linha, uma solução

prática comum é armazenar um vetor de valores, juntamente com a hora inicial, a frequência de amostragem e a identificação do sinal. Esta solução pode realmente diminuir o espaço utilizado no armazenamento, mas o tamanho do vetor de sinais será um problema dependendo de qual análise espera-se fazer posteriormente.

Esta solução também fere a normalização da modelo. Caso queira-se recuperar a informação de dois ou mais sinais diferentes, a tabela resultante pode não ser simples de montar através de consultas em SQL.

Tabela 2 - Tabela resultante da consulta de dois sinais temporais

Time Stamp	Value 1	Value 2
22/05/2009 14:29:32.00	23.5	
22/05/2009 14:29:33.00	23.5	
22/05/2009 14:29:34.00	23.492	
22/05/2009 14:29:34.10		122.221233
22/05/2009 14:29:35.00	23.492	

Mesmo supondo-se que a tabela acima consiga ser montada a partir do modelo relacional², podemos notar a grande quantidade de células em branco. Isto dá-se devido ao problema de sincronização dos sinais. No exemplo, eles foram captados com taxas de aquisições diferentes, e, portanto, em um determinado momento não possuímos o valor dos dois sinais concomitantemente.

² Para que esta operação seja realizada de forma implícita, é necessário a utilização de recursos externos à linguagem SQL (*scripts*) pois o *schema* da tabela resultante não é conhecido *a priori*. Porém, a operação pode ser feita de forma explícita dentro da linguagem SQL se para cada sensor for criada (explicitamente) uma tabela que será combinada com as tabelas geradas para os outros sensores.

Ainda utilizando-se a figura acima, pode-se imaginar, que no momento 14:29:34.10, o valor do sinal 1 pudesse ser o mesmo da linha anterior. Porém, o modelo relacional não prevê uma correlação entre linhas, portanto, este valor não pode ser calculado.

2.2.2 Procedimentos de Análise

O objetivo da análise dos dados é identificar algum tipo de correlação entre eles. Vários artigos já cobrem o conceito de *Data-Mining*, ou de como identificar relações entre linhas de tabelas. Outro tipo de análise que pode ser realizada são as transformadas espectrais, como por exemplo, a Transformada de Fourier, que analisa as frequências presentes em uma série histórica.

Implementar uma Transformada de Fourier não é uma tarefa complexa, basta que se recupere os dados em blocos de tamanho fixo (janela) e que se passe um algoritmo conhecido sobre eles. O resultado será a transformada daquele bloco específico.

Uma utilização deste algoritmo é na criação de filtros, como por exemplo, os de passa-baixa, passa-alta e passa-banda, que deixam passar apenas frequências especificadas, atenuando as demais.

Para a execução de um filtro, é necessário que a janela seja móvel, ou seja, cada novo valor é computado em relação a um grupo de outros. Um filtro relativamente simples de se escrever é um filtro de média móvel. Neste caso, considerando-se n valores e uma janela de tamanho k , o novo valor de um elemento i será a média dos valores i até $i+k$ (Figura 1).

Podemos perceber que o do conjunto resultante possui k elementos a menos que o conjunto original. Isto é, o tamanho da janela³.

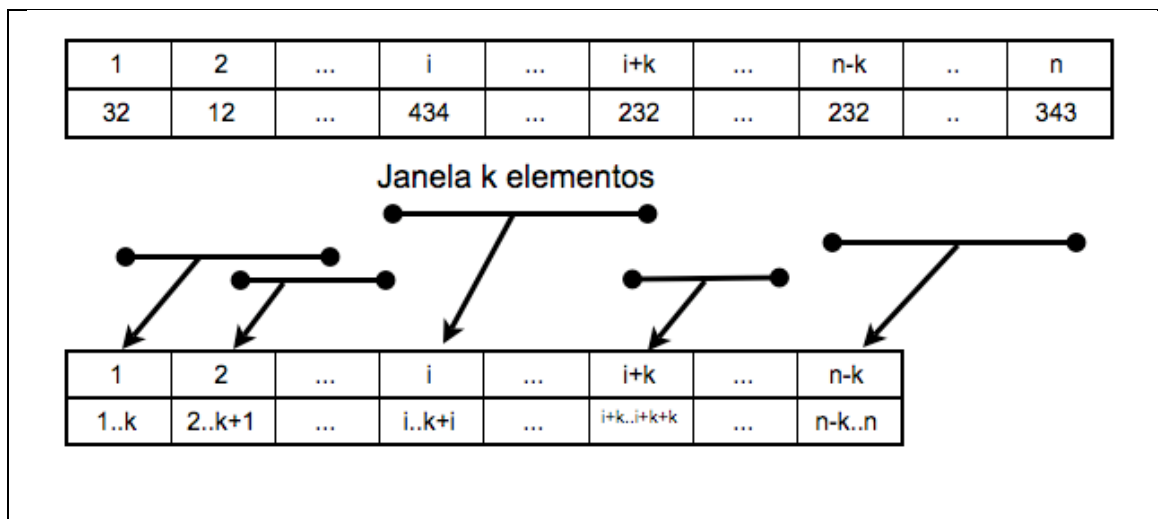


Figura 1 - Exemplo de filtro com janela móvel

Sistemas de Bancos de Dados não possuem as ferramentas necessárias para uma análise complexa sobre um conjunto de sinais. Apesar de conseguirem calcular a média e talvez o desvio padrão de uma tabela, não estão aptos a calcular uma média móvel, por exemplo. Para realizar este tipo de tarefa é necessário que se construa uma interface entre o aplicativo que realizará a análise, por exemplo, o software Matlab, e o SGBD.

A construção desta interface é uma tarefa onerosa para o analista que utilizará a ferramenta, pois a formação necessária ao processamento de sinais não prevê formação necessária para a modelagem e otimização de bancos de dados. São dois mundos a partes, cada qual observando o problema de um ponto de vista diferente e o resultado final pode ser ruim para ambos.

³ A correção desta discrepância é feita com algum tipo de regra arbitrária. Ou repetem-se valores, ou varia-se o tamanho da janela para $i > n - k$.

Uma proposta para solução deste problema é a criação de uma extensão da álgebra relacional, através da criação de novos operadores, para que o modelo de sinais no tempo possa ser manipulado mais facilmente em um SGBD.

2.2.3 Precisão

Ao se proceder com uma análise qualquer em sinais temporais é esperado que o sistema nos forneça um grau de precisão. Esperamos que o gerenciamento do dado não implique em alteração do resultado. Mas será isto possível ?

A resposta trivial para esta pergunta é que a precisão do resultado depende da forma como o dado é armazenado. Como os números reais não podem ser representados em um conjunto finito de bits, é natural que erros de precisão apareçam em sua representação. Este problema pode ser minimizado, dentro do possível, aumentando-se o espaço reservado ao armazenamento de modo a garantir a maior precisão possível. O Banco de Dados MySQL, por exemplo, armazena números de ponto flutuantes como sequências de caracteres (*strings*) de modo a garantir esta precisão. Para operá-los, os mesmos são convertidos para o formato de ponto-flutuante de precisão dupla (padrão *double* do IEEE) e após a operação, apresentados (ou re-armazenados) novamente como *strings* [5].

Apesar dos problemas inerentes ao armazenamento de dados numéricos e sua operação já terem sido estudados e estarem listados nos manuais de operação dos SGBDs [4], existem diversos problemas quanto à operação dos dados que não estão listados nos manuais e são ignorados devido às diversas camadas de abstração com as quais trabalhamos. A forma de manipulação dos dados influi no resultado final da computação. Estes erros de precisão não são devidos a forma de armazenamento e sim a forma de operação.

Consideremos o seguinte cenário:

Um banco de dados possui duas tabelas (T1 e T2) distribuídas em dois locais distintos, T1 localizada em S1 e T2 localizada em S2. T1 possui as colunas (numéricas) A e C; T2 possui a coluna numérica B. Ambas as tabelas possuem também um campo ID (de formato qualquer) que possa ser referenciado por uma operação de JOIN. Existe um terceiro sítio S3 que executará o JOIN propriamente dito. Neste cenário executaremos a seguinte consulta:

```
SELECT T1.A+T2.B+T1.C
FROM T1, T2
WHERE T1.ID = T2.ID
```

Listagem 1 - Possível consulta a uma base de dados distribuída

Se considerarmos as propostas de otimização apresentadas em Kossman [6], elucubraríamos que S1 enviaria para S3 as duas colunas de T1 (A e C) já somadas, o que diminuiria o custo de transmissão de dados para S3, onde, finalmente, a coluna restante (B) de T2 seria adicionada. Este raciocínio é correto dadas as propriedades comutativa e associativa da adição :

$$\begin{aligned}
 a + b &= b + a \\
 (a + b) + c &= a + (b + c) \\
 \text{Portanto :} \\
 a + b + c &= a + c + b
 \end{aligned}$$

Consideremos ainda uma instância para as colunas A, B e C de T1 e T2:

Tabela 3 - Exemplo de instância de valores A, B e C

T1 em S1		T2 em S2
A	C	B
555555555512341	0.4321	-555555555512340

Podemos notar que :

$$\begin{aligned}
 A + B &= 555555555512341 + (-555555555512340) = 1 \\
 A + B + C &= 1 + 0.4321 = 1.4321
 \end{aligned}$$

Para efeitos práticos, esta situação foi simulada em um Banco de Dados MySQL (v. 5.0.77 em ambiente Linux), foi criada uma tabela apenas com os três campos acima, todos utilizando precisão dupla (*double*). Foi executada a seguinte consulta :

```
SELECT a+(b+c), (a+b)+c, a+c+b, a+b+c
FROM T1
```

Listagem 2 - Consulta à base de dados de teste

produzindo o seguinte resultado :

Tabela 4 - Resultado da consulta anterior

a+(b+c)	(a+b)+c	a+c+b	a+b+c
1.4320983886719	1.4321	1.4375	1.4321

Através do qual podemos notar que uma mesma operação aritmética, simples e básica, conseguiu produzir três resultados diferentes !

A diferença entre os dois primeiros resultados é da ordem 10E-6, o que, em certos casos, poderia ser considerado como um erro de precisão aceitável, porém, a diferença de 0,0054 entre o terceiro caso e os demais não pode passar despercebida. Principalmente, levando-se em conta que é fruto da mesma operação.

Este erro não está relacionado à forma na qual o dado está armazenado. No caso mostrado, todas as operações foram realizadas com os mesmos valores, armazenados da mesma forma.

O problema ocorrido é apontado por Goldberg [1] e adaptado do mesmo proposto em Parhami [2] (p.316). Parhami demonstra que a lei da associatividade da aritmética não se aplica à operações com números de ponto flutuante "Muitas leis da álgebra não se aplicam à aritmética de ponto-flutuante (algumas não se aplicam nem

aproximadamente)". A tabela, a seguir, apresenta, além desta, outras leis que não se aplicam :

Tabela 5 - Propriedades algébricas inválidas em aritmética de ponto-flutuante

Associatividade na adição	$a + (b + c) = (a + b) + c$
Associatividade na multiplicação	$a \times (b \times c) = (a \times b) \times c$
Cancelamento (para $a > 0$)	$a \times b = a \times c \implies b = c$
Distributividade	$a \times (b + c) = (a \times b) + (a \times c)$
Multiplicação cancela divisão	$a \times (b / a) = b$

2.2.4 Operações com incertezas⁴ diferentes

Existe, por parte da comunidade científica, uma expectativa de erro de precisão nos cálculos pois os sensores já possuem um grau de incerteza incorporado em suas medições. Isto é caracterizado por Cudre-Mauroux em [20] como uma necessidade inerente aos SGBDs científicos, que eles operem de forma diferenciada em sinais com graus de incerteza diferenciados. Ou seja, sinais que possuem uma incerteza de 5% não devem ser operados diretamente com sinais com 1% de incerteza.

Para isto, existem regras de como propagar estas incertezas. Mas como elas não são modeladas, a priori, no SGBD, estas não podem ser aplicadas de forma trivial.

Juntando esta questão com a anterior, podemos notar que falta, também, a indicação que mesmo estes sensores de maior precisão podem por vezes falhar e produzirem resultados astronomicamente fora da escala esperada. Por exemplo, um sensor que nominalmente produziria um resultado entre 0,0 e 1,0; no caso de falha poderia produzir um sinal na faixa de 10 elevado a 60. Esta falha pode ser apenas

⁴ Aqui utiliza-se o termo incerteza ao invés do termo precisão de modo a diferenciar conceitos distintos. Precisão é utilizado no sentido de representação computacional do dado em si, como por exemplo, formatos de precisão dupla ou simples. Incerteza é utilizado para denominar a precisão física do sensor: um termômetro com 1% de incerteza, ao reportar 30° C, se refere a uma faixa de 29,7° a 30,3° C. Embora a representação do valor com 30° tenha precisão suficiente, a medida é incerta.

momentânea, até mesmo imperceptível, mas geraria um valor que ofuscaria a precisão do restante dos dados coletados.

3 Soluções Existentes e Propostas

3.1 Motivação

Apesar do crescente uso de SGBDs, podemos ver que ainda existem lacunas em sua utilização no meio científico. Alguns desses problemas são devido à grande quantidade de dados que devem ser armazenado (Gannon [8], Larus [9]); outros apresentam a necessidade de se alterar o paradigma de armazenamento para uma forma que melhor atenda à estrutura dos dados científicos (Cudre-Mauroux [20], Stonebraker [21]), alegando que “não existe um sistema de tamanho único que possa atender a todos e cientistas precisarão utilizar uma combinação de SGBDs especializados” [21].

Ailamaki [18] atesta em seu recente artigo que :

Processos científicos orientados por dados dependem em análises rápidas e acuradas dos dados experimentais gerados através de observações empíricas e simulações. Entretanto, os cientistas estão cada vez mais atordoados pelo volume de dados produzidos por seus próprios experimentos. Com o aumento da precisão dos instrumentos e a complexidade dos modelos simulados, a quantidade de dados gerados promete tornar-se ainda mais preocupante.

Szalay et al. em [3], concorda com essas premissas:

Com os (seus) conjuntos de dados crescendo para além de dezenas de terabytes, cientistas ainda não possuem uma ferramenta disponível que possa ser utilizada prontamente para gerenciar e analisar estes dados (...). O que é preciso é uma visão geral e sistemática para este problema com uma arquitetura que possa ser escalável no futuro.

tudo indica que a solução para este problema esteja sendo estudada sob a ótica de Banco de Dados. Afinal, faz sentido pensar na solução do problema de gerência de um grande conjunto de dados como uma questão de como, onde e quando armazená-los. Kossman [6] apresenta diversas formas de lidar com este problema através de bancos de dados distribuídos, levando-se em conta questões como o custo de transmissão de dados através de redes, onde estes dados devem ser processados, quando transmití-los e como. Graefe [7] revisita a “Lei dos cinco minutos” de Gray (apud Graefe) e discursa sobre os sistemas de armazenamento em memória Flash, conseqüentemente, como, onde e quando devem ser armazenados os dados de modo a diminuir o custo de processamento.

Porém, devido aos problemas existentes em adaptar-se os SGBDs ao uso científico, sendo que um deles é o próprio potencial financeiro deste mercado, “uma indústria de zero bilhões de dólares” [21], já existem algumas soluções proprietárias que tentam resolver o problema sem o uso dos SGBDs. Como já foi mencionado, Proficy Historian [29], utilizado para PIMS, fabricado pela GE e DIAdem, da National Instruments [38], são exemplos destas iniciativas privadas.

3.2 Soluções Existentes

3.2.1 COUGAR

Do ponto de vista da área de Banco de Dados, o armazenamento de séries temporais não é um problema novo. Podemos citar o projeto COUGAR [11], criado em 1987 pela universidade de Cornell, nos EUA. Séries temporais têm aplicações não somente na área de automação e análise de dados, mas também na área financeira, principalmente para análise do comportamento do mercado acionário.

O sistema COUGAR separa os dados do sensor (localização física, tipo do sensor, nome, etc.), que são codificados como Tipos Abstratos de Dados, das séries temporais geradas pelos mesmos. Estas séries são armazenadas separadamente e os dados de cada instante são recuperados através de janelas de tempo, através dos métodos associados aos sensores. Ou seja, um sensor de temperatura deve ter um método `getTemp()`, que retorna o seu valor de temperatura.

O COUGAR dá suporte às chamadas consultas de longa duração, ou seja, processamento dos dados on-line, no momento em que são incluídos na base de dados. A sua interface é feita através de uma pequena extensão da linguagem SQL. Na versão atual, existem ainda problemas com a união de janelas de tempo diferentes, o que impacta na utilização de filtros de processamento de sinais.

Atualmente, o desenvolvimento do sistema como um todo apresenta-se mais direcionado para problemas como redes de sensores remotos [48], a última publicação listada em seu site [40], data de 2005.

3.2.2 Bancos de Dados Ativos

Active Databases (Bancos de Dados Ativos), como o HiPAC [47], são SGBDs que executam procedimentos (*stored procedures*) baseados nos valores que certos campos possam apresentar. Este tipo de sistema pode ter aplicações tanto na área comercial, por exemplo executando a venda de ações quando o valor de uma ação atingir algum patamar, quanto no contexto científico, gerando um alarme quando algum valor atingir um limite.

O problema da utilização deste tipo de arquitetura recai nos problemas que estão sendo estudados neste trabalho. O tipo de dado que indicará (*trigger*) a execução de uma ação é ainda uma informação local, ou seja, baseado em apenas um valor e não em conjunto de valores com uma análise histórica ou espectral. Este tipo de sistema não está preparado para realizar ações a partir destas análises. Seria muito interessante realizar uma ação caso a tendência de uma variável observada se alterasse. O conceito de Banco de Dados ativos, porém, é muito interessante e pode ser utilizado pela comunidade científica caso exista um banco de dados científico que possa se tornar “ativo”.

3.2.3 Proficy Historian

O sistema Historian não utiliza um SGBD, para armazenamento dos dados. Isto é feito através de um repositório próprio, baseado em um sistema de arquivos. Uma discussão sobre os ganhos deste tipo de arquitetura é apresentada em sua brochura [39]. Os principais itens listados são o tempo de processamento de uma consulta e a facilidade de operação.

O Historian armazena os dados utilizando-se do conceito de Banda-Morta (*Dead-Band*) para otimização do armazenamento. Neste conceito, estabelece-se uma largura

de banda na qual o valor excursiona. Enquanto o valor permanecer dentro desta banda ele é tratado como se não houvesse sido alterado. O valor só é efetivamente registrado se sair da banda-morta. Um exemplo pode ser visto na Figura 2.

A utilização de banda-morta diminui o número de registros a serem gravados no banco de dados, porém, dificulta uma análise precisa a posteriori, pois simplifica o comportamento da variável. Outro problema do Historian é o limite na taxa de aquisição (10 Hz). Variáveis que se alteram rapidamente, acabam por perder resolução temporal.

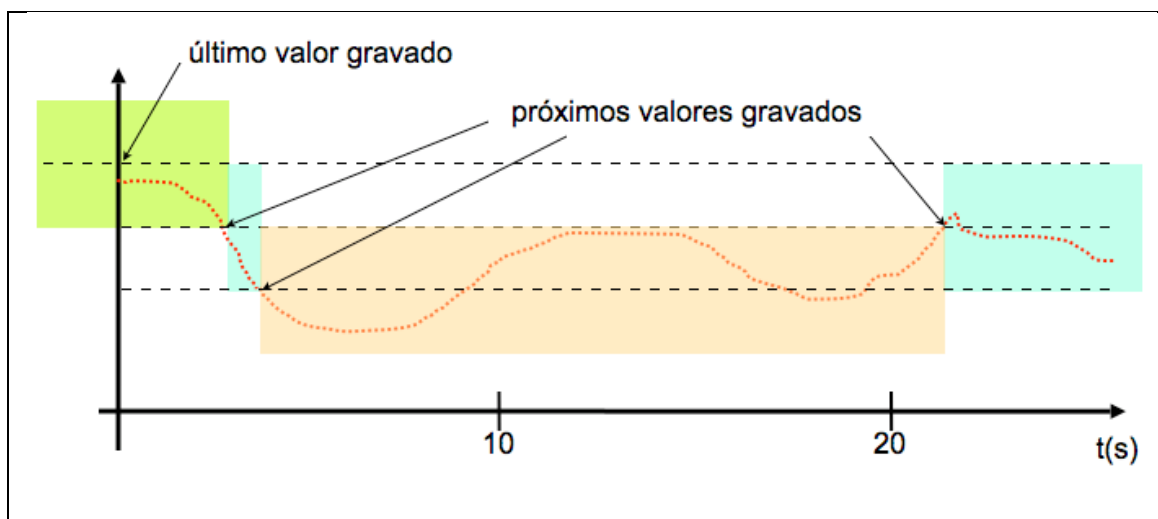


Figura 2 - Exemplo do conceito de Banda-Morta

3.2.4 DIAdem

O Sistema DIAdem da National Instruments [38], é um sistema de indexação e consulta de séries temporais armazenadas em arquivos previamente coletados. Partindo-se do fato que os fabricantes dos sistemas de monitoramento, produzem sistemas de aquisição que gravam seus dados em arquivos – basicamente por não existirem SGBDs onde possam ser armazenados com facilidade – o sistema DIAdem supre a necessidade de consulta a partir da gerência destes arquivos diversos.

O sistema é modular e configurado através de plug-ins que leem os tipos de arquivos diversos, por exemplo: arquivos Excel (.XLS) ou .CSV (*Comma Separated Values* – Valores separados por vírgulas) e recupera os dados dos sinais representados em linhas ou colunas destes arquivos.

Uma vez recuperados, estes dados podem ser processados pelo seu sistema LabView [41], uma ferramenta de grande utilização industrial e pela comunidade científica justamente pela sua facilidade de uso e integração com os sensores para aquisição e processamento de sinais.

Por basear-se nos arquivos gerados pelas ferramentas de aquisição dos dados, o DIAdem não está sujeito aos problemas do Proficy Historian; porém, o tempo necessário à consulta depende exclusivamente da forma como os arquivos estão organizados. O DIAdem é capaz de fazer um índice baseado nas informações temporais dos arquivos, mas, fora isto, nenhuma estratégia de otimização de consultas é produzida. Outro problema é a necessidade do usuário ser treinado na utilização do LabView, para onde os dados serão exportados.

3.2.5 Historis

Comercializado pela LIM – Logical Information Machines, o Historis [58], propõe-se a ser um banco de dados orientado ao armazenamento de séries temporais. É apresentado um estudo de caso [60] a partir de uma empresa distribuidora de energia elétrica americana, cujo nome não foi revelado e é referenciada pelo nome fictício de PowCo, que demonstra a eficiência do produto na gerência de suas séries temporais oriundas das várias turbinas elétricas distribuídas nos EUA.

Infelizmente não foi possível, até este momento, adquirir-se maiores informações sobre o produto. Não foi encontrada nenhuma versão de demonstração do produto que pudesse ser instalada sem um contato comercial direto com o fabricante.

3.2.6 kdb+

Trata-se de um produto para armazenamento de séries temporais produzido pela empresa Kx [59], utilizando-se como arcabouço o ambiente de desenvolvimento da linguagem K [54]. Este banco de dados utiliza uma extensão da linguagem SQL chamada KSQL. Segundo Shasha [57], esta extensão permite trabalhar-se com uma nova estrutura de dados chamada “*Arrable*”, uma combinação dos termos “*Array*” e “*Table*”, de forma a permitir que os resultados relacionais de uma tabela possuam uma ordem específica da qual possa-se tirar proveito.

Sasha et Zhu [53, 61], fazem uso do kdb em seu *paper* para a detecção de “*Bursts*” (picos), em janelas elásticas, isto é que podem ser de tamanhos diferentes, em oposição à janela fixa (vide seção 2.2.2). Estes algoritmos utilizam uma transformada de *wavelet*, [53] e Kaplan [55], para otimizar a procura destes picos. É provável que esta ferramenta tenha sido utilizado também por Neylon [56] e nos trabalhos de Cole et al [54] e Shasha [57].

Pela documentação estudada, acredita-se tratar de uma ferramenta bem promissora. Existem versões de demonstração disponíveis para serem instaladas. Porém, o foco principal da ferramenta parece ser o armazenamento de séries temporais financeiras (mercado de ações). Não foi encontrada nenhuma referência quanto à manipulação de séries com graus de incerteza distintos (vide seção 2.2.4).

3.2.7 SciDB

Fruto do *Workshop XLDB-1*, ocorrido em 2007, e realizado com a presença de cientistas e usuários de aplicações comerciais, o SciDB responde à série de requisitos apresentados pelos seus participantes, como uma proposta de um Banco de Dados voltado para a área científica.

Sua principal característica é romper com o modelo relacional e propor uma estrutura na forma de um “*Ragged Array*” ou (*Array Retalhado*), conforme pode ser visto na Figura 3. Este *array* pode conter diversas dimensões, de modo a, segundo os criadores do projeto [20,21], melhor atender as necessidades das aplicações científicas, em relação ao sistema relacional.

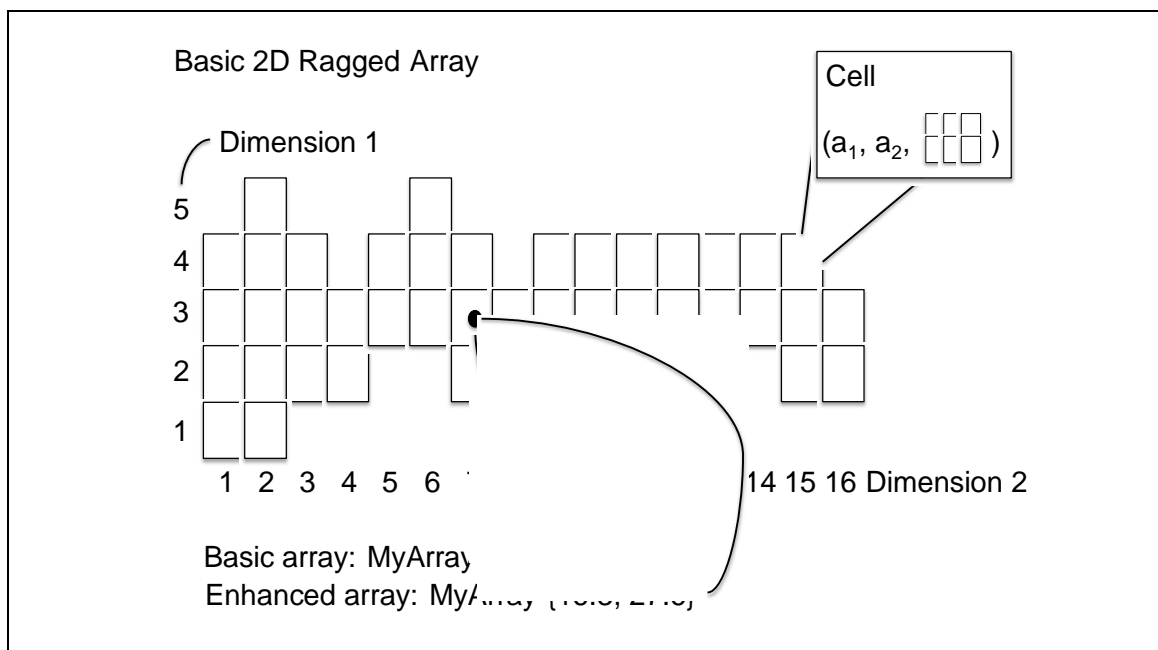


Figura 3 - Exemplo de "Ragged Array" do SciDB. Extraído de [20]

Esta implementação alcançaria performance superior do que extensão de um SGBD relacional através de uma álgebra específica, para uso científico. Mas os próprios autores do projeto também argumentam que não existe uma resposta única para todas as necessidades e que a solução apresentada visa atender apenas a um grande conjunto de usuários.

O SciDB opera com um tipo de dado denominado célula, que pode conter outros tipos de dados convencionais, como números reais, ou ainda outros *arrays*. Cada célula do SciDB é indexada através de coordenadas inteiras, mas o sistema pode ser estendido de modo a permitir a indexação de outras formas. Um exemplo apresentado é um índice em que se utilizam as coordenadas de latitude e longitude do sistema de Mercator, muito utilizada em aplicações que utilizam mapas terrestres. A utilização deste tipo de indexação, à princípio, atende à aplicação do *Large Synoptic Survey Telescope* (LSST), um sistema de mapeamento astronômico que se beneficia de consultas a partir de coordenadas geográficas.

Uma das funções mais poderosas do Banco de Dados é a possibilidade de re-agrupar (“*Regrid*”) suas células arbitrariamente, fornecendo novos índices para consulta. As etapas do processo de reagrupamento são gravadas de modo a poderem ser reconstituídas novamente. Funções de versionamento permitem que uma série de células seja modificada sem que seu conteúdo original seja perdido.

O SciDB permite que sejam feitas operações de adequação prévia dos dados, ou seja pré-processamento, chamado de “*cooking*” (cozimento), dentro ou fora do Banco de Dados. O “*cooking*” prévio seria feito nas máquinas as quais estão ligados os sensores, ou no próprio SGBD. Também é prevista a manipulação interna dos dados (*In Situ*), de modo que os mesmos não precisariam ser exportados para um programa à parte, como o Matlab, para este fim.

3.3 Alternativas

Apesar do SciDB propor um caminho heterodoxo para a solução de armazenamento de dados científicos, nas referências examinadas não foram encontrados modelos formais que possam representar os dados que serão armazenados. Levando-nos a fazer a pergunta: “Se o SciDB é a resposta, qual é o problema?”. Um exemplo de alternativa, análoga, e bem sucedida que utiliza banco de dado relacional é o *Sloan Digital Sky Survey* (SDSS, apud [18, 20, 21]), que utiliza um banco de dados Postgres como base e possui extensões para tratar os tipos de dados específicos.

3.3.1 Workflows

Uma alternativa proposta pela literatura disponível para o problema de gerência de dados científicos é a utilização de Workflows, ou seja, através do controle do fluxo dos processos pelos quais os dados são tratados e armazenados [42, 43, 44, 45].

A ideia de workflows não é recente. Conforme Gannon et al.[42], podemos identificar ferramentas que gerenciam o fluxo de dados entre programas desde os antigos sistemas de grande-porte com a linguagem JCL (*Job Control Language*), que tinha por objetivo orquestrar a carga de programas de modo a repassar a massa de dados gerados entre eles. Mas sua aplicação nos meios científicos começou apenas a ser usada recentemente.

Ferramentas de controle de workflows são utilizadas com frequência no âmbito comercial. São utilizadas, por exemplo, para integração de sistemas Web com os outros sistemas corporativos, segundo Barga e Gannon [43]. Porém, estes autores também listam os problemas de adoção deste paradigma para dados científicos.

Um dos problemas mencionados é o paradigma ACID, que representa Atomicidade, Consistência, Isolamento e Durabilidade. Um exemplo de uso deste paradigma é que uma transação convencional em um SGBD pode ser desfeita (*rollback*), caso haja um problema em sua execução, de tal forma que a consistência do SGBD, ao final, seja preservada. A transação, então poderia ser executada novamente, uma vez que os problemas fossem sanados. O mesmo não pode ser esperado de um sistema de aquisição de dados de um experimento real. Podemos imaginar o seguinte exemplo para ilustrar esta situação: um workflow espera adicionar em um SGBD um conjunto de 100 leituras oriundas de um experimento a cada hora. Em um determinado instante, ocorre um problema e o sistema de aquisição gera um arquivo com apenas

30 leituras, por conta de inúmeros problemas como falta de luz, etc. Após o problema resolvido, o sistema volta a coletar os dados e a gerar os arquivos corretamente. Dado este cenário hipotético, a ferramenta de controle de workflows não será capaz de incluir os dados coletados pela metade no SGBD, por questões de consistência (tamanho < 100). Estes dados deverão ser incluídos posteriormente, manualmente. Problemas como o narrado são comuns no contexto analisado e afetam a eficiência de uma ferramenta automática.

Shields [44] propõe a necessidade de se desenvolver uma nova linguagem comum de *workflows* que compatibilize as ferramentas comerciais ao uso científico; Gannon [45] apresenta modelos baseados em componentes como um caminho para esta solução. Porém, uma crítica que podemos fazer ao uso de ambas as referências é quanto à complexidade da solução proposta. Estas propostas englobam uma série de tecnologias, CORBA, XML, SOAP, Java, *Web-services*, etc. que fazem parte do currículo de TI, mas não fazem parte da expertise dos cientistas usuários. Faz-se, então, necessário envolver uma equipe de TI para solucionar o problema. Podemos considerar que os laboratórios em questão já tenham algum tipo de suporte de TI, porém, neste caso o problema complica-se pois a escolha da ferramenta apropriada começa a ser influenciada por questões de arquitetura corporativa, como por exemplo a filosofia de sistema operacional (Windows vs. Linux), segurança, acesso, etc.

3.3.2 SECONDO

Outra alternativa possível para o problema de armazenamento de dados científicos é a utilização do sistema SECONDO [23, 24, 25 e 26], criado pela universidade Fern na cidade alemã de Hagen.

SECONDO é um sistema de extensão de bancos de dados através de álgebras modulares. Ele possui um sistema de álgebra padrão, que lida com os operadores aritméticos e de cadeias para tipos de dados atômicos (numéricos, cadeias de caracteres, booleanos); um sistema para álgebra relacional, típica de SGBDs convencionais; um sistema de álgebra para valores de datas e assim sucessivamente, de modo que outras álgebras possam ser adicionadas, estendendo o sistema básico.

Especificamente, o SECONDO trabalha com o Berkeley DB como base, mas trata-se de um sistema aberto (*open-source*, assim como o SciDB). Extensões podem ser feitas em C++ (álgebras) e/ou Prolog (*parsing* gramatical). Também está disponível uma interface gráfica (GUI) criada na linguagem Java para acesso ao Banco de Dados.

3.3.3 Metadados

Um ponto pacífico em vários artigos examinados [3, 8, 9, 17, 18, 20, 21, 22] é a necessidade de armazenamento de metadados sobre o experimento. O que é exatamente um metadado, porém, é um tanto controverso. A estrutura proposta pelo SciDB [20, 21] é tão ligada à implementação que qualquer coisa fora do escopo é considerado como metadado. Isto inclui informações como o nome do sensor, unidade de engenharia (bar, N, mm, etc.) e qualquer outra informação diferente do dado gerado, propriamente dito. Algo como dizer que o nome de um cliente é um metadado de sua conta corrente bancária. Já o COUGAR [17] considera que estas informações são parte de uma estrutura diferente, relacional, mas são ainda dados consideráveis do processo. Ailamaki [18] levanta, ainda, a necessidade de serem armazenadas anotações sobre eventos que ocorreram no processo, bem como a realimentação de resultados de simulações para uso em conjunto com os resultados de sensoriamento empíricos, e vice-versa, de modo a promover análises futuras.

3.3.3.1 Faixa de Trabalho do Sinal

Pode-se dizer que as otimizações utilizadas em SGBDs foram criadas visando-se o caso genérico. Preocupando-se com o armazenamento e processamento do dado e não com a sua “natureza”. A eliminação de sub-expressões constantes, que pode estar presente no processo de otimização de uma consulta, não leva em conta a faixa numérica em que o dado excursiona. Ela é desconhecida do projetista do otimizador, mas é conhecida pelo usuário.

Do ponto de vista da acurácia do resultado, o uso de metadados é de importância fundamental se analisarmos o problema da questão da precisão do resultado, proposto anteriormente em A.2.2.

Como exemplo de faixa de trabalho utilizemos a temperatura de um equipamento industrial, um forno, por exemplo. Pode-se supor que a temperatura do mesmo varie entre um mínimo que seria temperatura ambiente e um máximo que seria especificado pelo fabricante do equipamento. Ao instrumentar-se este equipamento, será escolhido um sensor que tenha maior sensibilidade (acurácia) na faixa de altas temperaturas. Ou seja, um sensor que seria utilizado em um forno não é necessariamente o mesmo utilizado em uma geladeira. Se este sensor reportar valores muito altos (acima da especificação do equipamento) ou muito baixos, é razoável imaginar que o mesmo está defeituoso.

É preciso que estas informações sejam tratadas pela álgebra de modo a aumentar a precisão dos resultados, evitando-se “sujar” o processamento operando-se dados que estão dentro da escala do sensor, porém fora da escala de utilização, ou por exemplo, operá-los à posteriori diminuindo seu impacto no todo. Por outro lado, também é necessário distinguir-se entre um valor proveniente de um sensor defeituoso e um

valor desconhecido. Este último, se comprovado que o sensor não estava danificado, é relevante para o processo de análise.

O sistema SciDB prevê uma forma de operação em que o grau de incerteza de um sensor altera a forma em que a operação é realizada. Pois, de uma forma geral, pode-se minimizar os erros de precisão operando-se valores de faixas similares primeiro. Faz sentido que o processo de otimização de consultas para bases de dados numéricas de aplicações científicas usem estes metadados como parâmetros na confecção de estratégias de busca.

3.3.3.2 Anotações

Outra sugestão seria utilizar a informação de metadados através de anotações para facilitar uma consulta. Uma anotação seria uma referência a um instante ou um determinado valor, de modo que o usuário possa recuperar os dados a partir da mesma. Por exemplo, um usuário pode criar uma anotação de um determinado evento chamando-a de “evento 1”, e realizar consultas a partir desta denominação.

Considerando-se que os recursos de gerência de dados em SGBDs convencionais não são suficientes para a operação de valores de sinais no tempo, este trabalho procura encontrar uma forma de armazenamento, representação, processamento e consulta de sinais temporais de tal forma a aproveitar o que já foi desenvolvido para a área comercial pois acredita-se na importância de uso de um SGBD como ferramenta de consulta para usuários da comunidade científica.

4 Secondo:

Ambiente de extensão da álgebra relacional para operadores científicos

Como já foi dito anteriormente, serão apresentados diversos operadores que facilitarão o gerenciamento e manipulação de séries temporais em conjunto com dados relacionais. Uma abordagem para o problema de integração da álgebra relacional com operadores de séries temporais, passa pelo uso de um SGBD expansível, voltado para acrescentar novas operações visando a atender uma classe específica de aplicações científicas. Essa extensão precisa passar por uma prova da consistência destes operadores.

Para o desenvolvimento da solução proposta foi necessário não apenas definir os novos operadores algébricos, como também utilizar-se de uma ferramenta já existente que permitisse criar uma extensão de um Banco de Dados para realizar uma validação funcional e prática. Este capítulo é dedicado à análise de ferramentais de extensões a SGBDs e à descrição do SECONDO como plataforma escolhida, tanto para extensão da álgebra, pois adota uma álgebra relacional de segunda ordem, como ambiente de validação de consistência algébrica. Além disso, permite a validação prática um sistema que permite expandir um SGBD convencional, no caso o Berkeley DB, com a nova álgebra relacional estendida e disponibilizada de forma integrada via SQL.

4.1 Critérios de escolha da ferramenta de integração

Para a escolha da ferramenta utilizada, foram levantados vários critérios de modo a guiar esta decisão. O primeiro critério que podemos citar foi o de originalidade para abordar o problema de "impedance mismatch" entre dos dados textuais e os dados de séries temporais. A ferramenta escolhida deve permitir a criação de um produto final que integre os dois ambientes, visto que, várias ferramentas para a manipulação e armazenamento de séries temporais são específicas e não provêm a integração de consultas genéricas tanto a dados textuais quanto de séries temporais. Este critério, por si só, já serviu como fator de corte de diversas ferramentas, entre elas o DIADem da National Instruments, o Proficy Historian da GE e o SciDB. O interesse não é realizar mais um sistema de armazenamento de séries temporais, e sim, definir bases de integração dos dois mundos que possam servir para o avanço da tecnologia de solução do problema de gerência, representação, armazenamento e consulta a dados textuais e de séries temporais em um único ambiente.

O segundo critério utilizado foi a possibilidade de conceituação teórica. Após a análise das diversas referências apresentadas, apenas algumas ([57] e [61], por exemplo) propõem um modelo para séries temporais e, nestes casos, são apresentados modelos de armazenamento para as séries e não modelos conceituais. A ferramenta escolhida deve permitir que seja implementada uma representação do modelo conceitual proposto.

Após a utilização deste critério, três ferramentas estavam disponíveis para o uso: kdb+, SciDB e SECONDO. A possibilidade de utilização de um modelo de *workflows* foi eliminada por não atender a este critério, uma vez que não envolveria um modelo conceitual de séries temporais.

Por se tratarem de ferramentas comerciais, o kdb+ e o Historis, foram eliminados da escolha. Não se conseguiu ter acesso às versões de demonstração, seja por questões de comunicação com o fabricante ou por questões de disponibilidade de hardware para a versão demo existente (arquitetura SUN Sparc). Acredita-se, também, que por serem comerciais, quaisquer extensões à esses sistemas seriam por demais complicadas e academicamente infrutíferas.

Por fim, apresentaram-se apenas duas soluções: o SciDB e o SECONDO. A primeira solução é baseada no Postgres [68], um SGBD “extensível”, enquanto a segunda utiliza o Berkeley DB [52] para armazenamento de suas estruturas. O SciDB, utiliza o caráter extensível do Postgres para implementar um modelo de dados chamado *ragged-array* e uma linguagem de acesso chamada AQL – *Array Query Language*.

Porém, uma das críticas que Güting [62], um dos criadores do SECONDO, faz ao Postgres é quanto à clareza do significado de “extensibilidade”. O sistema SECONDO é apresentado, então, como uma forma de conceituar esta “extensibilidade” através do uso de “assinaturas de segunda-ordem”. Estas assinaturas possibilitam a criação de álgebras que podem ser adicionadas ao sistema proporcionando novas funcionalidades. Uma vez acopladas, o sistema SECONDO possui um validador de compatibilidade entre a álgebra relacional clássica e os operadores propostos para estendê-la, assim como um otimizador (desenvolvido em PROLOG) que pode acelerar as consultas ao sistema e implementa um interpretador da linguagem SQL. Esse novo SQL provê a integração da nova álgebra relacional estendida sob a forma da linguagem clássica de consultas genéricas de SGBDs.

4.1.1 Escolha do SECONDO como plataforma

As duas últimas características do SECONDO apresentadas (a criação de álgebras e a posterior otimização das mesmas) foram os quesitos que mais pesaram na seleção do SECONDO como plataforma de implementação. Esses dois critérios não são contemplados nas soluções relacionadas. A criação de uma álgebra permite que se faça um modelo conceitual do que é uma série temporal; como ela se caracterizaria e contextualizaria em um ambiente estruturado relacional; cuja implementação também validaria a compatibilidade com a Álgebra Relacional existente. A possibilidade de otimização a posteriori permite que esta solução seja acessada através de uma extensão da linguagem SQL, que escolheria o melhor plano de execução da consulta possível com estes operadores⁵.

Uma vez provada a consistência e praticidade da solução apresentada, ela poderia ser aproveitada por outros SGBDs, que utilizem conceitos diferentes, como, por exemplo, armazenamento por colunas, como é o caso do Monet DB [65]. Ou seja, a escolha do sistema SECONDO como plataforma de desenvolvimento, por atuar na base da questão, isto é, na parte da fundamentação algébrica, se concilia com os dois quesitos apresentados anteriormente, de originalidade e generalidade para posterior contribuição tecnológica e de contribuição conceitual para definição e modelagem de séries Temporais em produtos.

4.2 Assinaturas de Segunda-Ordem

O SECONDO deve seu nome à proposta de Güting [62], na qual é apresentado um sistema que utiliza assinaturas de segunda-ordem, assim como uma álgebra que possibilite operá-las, de modo a aplicar o conceito de Tipo Abstrato de Dados à SGBDs.

⁵ Apesar de proposta no exame de qualificação, a extensão da linguagem SQL foi retirada do escopo do trabalho.

O sistema possui dois níveis. O nível superior define um sistema de tipos, na qual vários tipos podem ser agrupados em categorias (*sorts*), segundo sua natureza (*kind*)^{NT}. Enquanto o nível inferior provê as operações em si. Estas são polimórficas, isto é, preparadas para receber os diversos tipos que constituem uma natureza (*kind*), dos objetos definidos o primeiro nível.

4.2.1 Definição de Naturezas e Tipos

Um sistema implementado com o SECONDO possui várias categorias, cada qual de uma ou mais naturezas. Assim sendo, *int* e *real* seriam tipos de natureza (*kind*) NUMERIC e DATA (ambas), enquanto *string* e *bool* seriam apenas tipos de natureza DATA. o conjunto formado pelas diversas naturezas (DATA e NUMERIC, por exemplo) constituiriam as categorias (*sorts*) com as quais o sistema opera⁶.

Podem ser criados tipos que são construídos com base em *kinds*. O tipo *tuple*, por exemplo pode conter uma lista de um ou mais atributos, de natureza DATA, identificados pelos seus nomes, que por sua vez serão do tipo *ident*. Este tipo terá a natureza TUPLE. Ou seja, uma tupla não poderá ser formada por outras tuplas, pois o *kind* TUPLE é diferente do *kind* DATA. O tipo *rel* (relação) deve ser declarado a partir de um tipo TUPLE, porém não a partir de um tipo DATA.

É importante ressaltar que a relação é definida a partir de um tipo *tuple*, e não por uma lista de tuplas (como mandaria a intuição), pois o que se está definindo é o tipo da relação e não a relação em si, esta última conterá uma lista de tuplas como se espera. Porém, o sistema é aberto o suficiente para que possa definir algum tipo de relação que, por ventura, permita armazenar tuplas de diferentes tipos, desde que o

^{NT} A tradução de “*sorts*” em categorias e “*kind*” por natureza é livre, dado que os termos são praticamente sinônimos e propostos por Güting [62] em seu trabalho.

⁶ A diferenciação entre *sorts* e *kinds* é utilizada nos conceitos teóricos apresentados por Güting [62]. Na prática, apenas o conceito de *kind* se faz necessário para a utilização do sistema.

usuário implemente os operadores necessários à esta álgebra. A figura abaixo ilustra o sistema de tipos apresentado.

A figura deve ser entendida da seguinte forma: objetos definidos pelos construtores `int`, `real`, `bool` ou `string` serão considerados como sendo da natureza `DATA`; objetos definidos pelo construtor `tuple`, serão da natureza `TUPLE`, desde que se apresentem na forma de uma lista de um ou mais pares de objetos do tipo `ident` e objetos de natureza `DATA`.

	→ IDENT	ident
	→ NUMERIC	int , real
	→ DATA	int, real, bool, string
(ident × DATA) ⁺	→ TUPLE	tuple
TUPLE	→ REL	rel

Figura 4 – Diagrama básico de construtores de tipos

É importante ressaltar que o diagrama acima não apresenta, ainda, uma sintaxe específica de definição de tipos, e sim uma representação de como os tipos e naturezas devem ser formados. Na linguagem em si, alguns destes construtores podem se apresentar explicitamente, como em um comando:

```
create i : int;
```

ou implicitamente, como:

```
create t1 : tuple( a : int, b: real);
```

No primeiro caso, cria-se um objeto do tipo `int`, de nome `i` (porém sem valor atribuído). No segundo caso, cria-se um objeto de nome `t1`, do tipo `tuple` (explicitamente) e os nomes de seus atributos `a` e `b` são (implicitamente) entendidos como sendo do tipo `ident`; como `int` e `real` são de natureza `DATA`, a linguagem

aceita o comando. Maiores detalhes da linguagem serão apresentados posteriormente e podem ser consultados no Manual do Usuário do SECONDO [25].

4.2.2 Definição de Operadores

Após a definição dos tipos e naturezas faz-se necessário definir os operadores que trabalharão com os objetos definidos. Para tal, como exemplo, será usado o operador “+”. Podemos imaginar três sentidos (semânticos) para este operador (levando-se em conta o símbolo “+”): adição (operador binário), positivo (operador unário que especifica números positivos, o número +5, por exemplo) e concatenação (de *strings*).

O primeiro nível da assinatura de segunda-ordem para este operador será responsável por identificar qual operador (adição, positivo ou concatenação) está sendo referenciado e produzirá o tipo ou natureza do resultado, como mostra a figura abaixo

NUMERIC	→ NUMERIC	+, -	#_
NUMERIC × NUMERIC	→ NUMERIC	+, -, *, /	_#_
string × string	→ string	+	_#_

Figura 5 – Diagrama para o operador +

Este diagrama apresenta uma coluna a mais que o anterior na qual a posição do operador deve ser inserida. O operador é representado pelo símbolo “#” e os parâmetros representados pelo símbolo “_”.

São apresentados os operadores:

- positivo e negativo, unários, operam com natureza NUMERIC, são prefixados (operador antecede operando), o resultado é NUMERIC;

- adição, subtração, multiplicação e divisão, binários, operam com natureza `NUMERIC`, são de sintaxe *infix* (operador intercala operandos), o resultado é `NUMERIC`;
- concatenação, binário, opera com tipo `string` e é também de sintaxe *infix*, o resultado é `string`;

4.2.3 Polimorfismo e Sobrecarga de Operadores

Pode-se dizer que a primeira ordem da assinatura do operador lida com a questão da sobrecarga (*overloading*) dos mesmos, porém, apesar de lidar com o conceito de *kinds*, no momento de execução, o sistema precisa determinar qual o tipo (*strictu-sensu*) resultante da expressão, para que possa ser feita a alocação de memória para o objeto resultante.

Para o operador concatenação, este tipo já é especificado na própria definição do operador (embora na implementação tenha que seguir o mesmo processo dos outros), mas os operadores positivo e adição apresentam *kinds* como resultado, e os tipos resultantes precisarão ser definidos.

Esta definição será feita no segundo nível da assinatura, que lida com o polimorfismo do operador. Este polimorfismo pode ser representado através de uma tabela que cruza os tipos dos parâmetros e apresenta o tipo do resultado. As tabelas para o exemplo anterior podem ser vistas abaixo:

Tabela 6 – Polimorfismo do operador Adição

Operando 1	Operando 2	Resultado
real	real	real
real	int	real
int	real	real
int	int	int

Tabela 7 – Polimorfismo do operador Positivo

Operando 1	Resultado
int	int
real	real

A implementação de todo operador no SECONDO requer uma definição do tipo resultante. Para operadores mais simples podem ser utilizadas tabelas como a do exemplo acima, porém, o resultado do operador projeção de uma relação, por exemplo, produzirá tuplas de tipos diferentes dependendo-se de quais atributos deseja-se que sejam projetados. Nestes casos é necessário que um algoritmo seja usado para determinar o tipo resultante.

4.3 Estrutura do Sistema

O Sistema SECONDO é composto de quatro níveis:

- Gerenciador (interpretador) de Comandos;
- Processador de Consultas, Otimizador e Catálogo;
- Álgebras;
- Gerenciador de Armazenamento e Ferramentas;

Estes níveis estão dispostos conforme a figura abaixo:

Gerenciador de Comandos (interpretador)			
Processador de Consultas		Catálogo	Otimizador
Álgebra 1	Álgebra 2	...	Álgebra N
Gerenciador de Armazenamento		Ferramentas Diversas de Manutenção	

Figura 6 - Arquitetura do Sistema SECONDO

O interpretador de comandos faz o *parsing* sintático dos comandos e repassa o resultado ao processador de consultas ou ao otimizador. O interpretador também pode ser acessado remotamente, através de um protocolo cliente-servidor, o que possibilita também a conexão de uma interface gráfica (GUI) feita na linguagem Java e de um otimizador de consultas, implementado na linguagem Prolog. Este último módulo é de grande valia, pois permite também a consulta ao sistema utilizando-se a sintaxe da linguagem SQL, que pode ser estendida para a inclusão de novos comandos, além do fato de programar estratégias de acesso aos dados armazenados. Apesar de úteis, ambos os módulos não fazem parte deste estudo. Serão apresentados resultados que permitirão a otimização dos operadores que serão apresentados, mas a otimização dos mesmos está fora do escopo deste trabalho.

O processador de consultas é chamado pelo interpretador de comandos e fará uso das diversas álgebras acopladas no nível inferior. É mantido por ele um Catálogo com todos os objetos criados e armazenados.

Para que as álgebras sejam acopladas, uma interface (API) foi especificada de modo que o processador de consultas consiga identificar os tipos envolvidos e operados (sobrecarga e polimorfismo) por cada uma delas. As álgebras são desenvolvidas separadamente e compiladas e ligadas ao sistema como um todo.

Por fim o Gerenciador de Armazenamento implementa uma camada de abstração (*wrapper*) que faz a interface com o DBMS que armazenará, efetivamente, os objetos. No caso o Berkely DB [52] foi utilizado, mas alterando-se este nível, outros sistemas (Oracle, Postgres) poderiam, a princípio, ser utilizados.

De uma forma geral, pode-se dizer que basta-se programar uma álgebra, seguindo-se a API, em C++, para acoplá-la ao sistema. Um conceito chamado de “*Plug & Play*” por Dieker e Güting [63]. Na prática, é necessário se alterar um pouco mais do que isso. Tipos precisam de pequenas alterações no Interpretador de Comandos e GUI (em Java) para que possam ser apresentados (*displayed*) de forma mais “agradável”, principalmente se contiverem informação gráfica.

As álgebras também requerem uma pequena linguagem de definição, que represente os diagramas das Figura 4 e Figura 5, esta linguagem é processada pelas ferramentas Flex [69] e Bison [71], disponíveis nas distribuições GNU [70] e MinGW [72] (este último para o sistema Windows). O uso destas ferramentas requer que o sistema seja recompilado para cada nova álgebra adicionada ou modificada. Porém, uma vez que se entenda o funcionamento do sistema, o desenvolvimento desses módulos é realizado sem grandes problemas.

Por fim, o sistema está disponível para várias plataformas, entre elas Windows, Linux e Mac OS X (versão 10.4 e posterior). O sistema escolhido para desenvolvimento foi a distribuição Ubuntu 10.10 [73] do Linux, em um computador DELL Vostro 230 com processador Intel Celeron E450 de 2.2 GHz, que foi bem sucedida. Foi feita uma tentativa de instalação em um sistema Mac OS X com processador Motorola G3, porém uma das bibliotecas fornecidas com o interpretador SWI-Prolog [74], parte do SDK do sistema, não foi atendida pelo hardware, requisitando uma processador Motorola G4. Foi feita uma nova tentativa de se compilar o sistema com uma versão

mais antiga da ferramenta que não surtiu resultados devido à mudanças na API. A tentativa de gerar esta outra versão da plataforma foi então abandonada.

4.4 Utilizando o *SECONDO*

Como já foi dito anteriormente, o sistema *SECONDO* é composto por um conjunto de álgebras que podem ser operadas através de uma linguagem própria. Cada álgebra acoplada estende esta linguagem com seus próprios tipos e operadores.

As álgebras (incluídas no sistema) utilizadas neste trabalho são :

- **Standard** – provê os tipos básicos: `int`, `real`, `bool`, `string`; de natureza `DATA`, bem como os operadores lógicos e aritméticos para os mesmos;
- **Relational** – implementa os tipos e *kinds* `rel` (`REL`) e `tuple` (`TUPLE`)⁷ bem como algumas operações de tabelas;
- **ExtRelational** – implementa os operadores restantes da Álgebra Relacional, por exemplo, `product`, `join`, `filter`, etc.;
- **Stream** – implementa o tipo `stream` (`STREAM`) e os operadores `feed` e `consume`, utilizados com este tipo. O conceito de *stream* será apresentado a posteriori;
- **DateTime** – implementa tipos que lidam com tempo, `instant` e `duration` (`DATA`), junto com operadores e funções de suporte como, por exemplo, `now()`, que retorna a data e hora corrente;
- **RTree** – Implementa a estrutura de índices `rtree`, capaz de armazenar tipos de natureza `SPATIAL2D` e operadores para criação e consulta da mesma;

⁷ Como forma de diferenciar um tipo de uma natureza, os tipos são grafados em fonte courier (monoespaçada) em minúsculas, enquanto as naturezas (*kinds*), são grafadas em maiúsculas;

Uma interface GUI está disponível e é útil para mostrar objetos gráficos e que se movam no tempo, uma das motivações que levaram ao desenvolvimento do sistema. A Figura 7 apresenta esta interface.

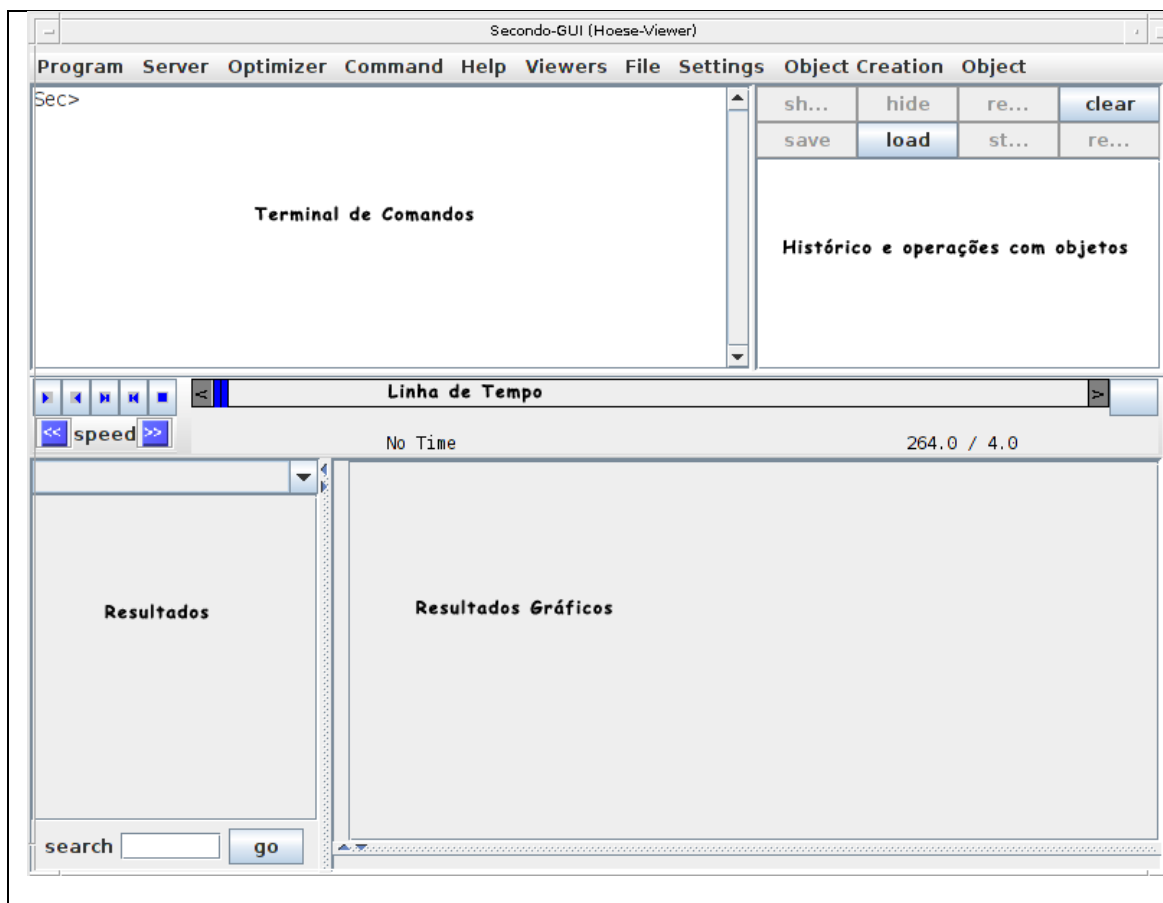


Figura 7 – Interface gráfica JavaGUI do sistema Secondo

No campo superior esquerdo existe o terminal utilizado para se digitar os comandos, ao seu lado é apresentado um histórico dos comandos passados. Na parte central é apresentada uma Linha de Tempo, para se trabalhar com objetos que se movem no espaço (não utilizada neste trabalho). Abaixo desta linha, no lado esquerdo, os resultados são mostrados, formatados, como texto, e ao lado direito, são apresentados resultados gráficos.

Outra forma de se operar o sistema é através de uma janela de terminal texto, de linha de comando. Neste trabalho, as duas formas serão usadas, pois o terminal mostra um histórico dos comandos utilizados junto com seus resultados. Um exemplo de acesso através de terminal com alguns comandos já digitados pode ser visto na Figura 8.

```

File Edit View Search Terminal Help

Secondo => query 5;
Total runtime ... Times (elapsed / cpu): 0.002287sec / 0sec = inf

5
Secondo => let A = 5;
Total runtime ... Times (elapsed / cpu): 0.106645sec / 0sec = inf

=> []
Secondo => query A;
Total runtime ... Times (elapsed / cpu): 0.002281sec / 0sec = inf

5
Secondo => query MyRel;
Total runtime ... Times (elapsed / cpu): 0.018463sec / 0sec = inf

cod:1
nome:Ana

cod:2
nome:Beatriz

cod:3
nome:Carlos

Secondo => query MyRel count;
Total runtime ... Times (elapsed / cpu): 0.017367sec / 0sec = inf

3
Secondo =>

```

Figura 8 – Interface com o sistema através de Terminal

No exemplo acima, podemos ver uma série de interações com o sistema. Após cada transação é mostrada a duração da mesma. As cinco transações são explicadas abaixo:

query 5;

Consulta o valor de um objeto constante (o número inteiro 5);

```
let A = 5;
```

Cria um objeto (A) que contenha o valor do objeto ao qual lhe foi atribuído. O tipo do objeto resultante é inferido e definido pelo operador `let`. Neste caso, `int`, pois 5 é uma constante inteira.

Uma operação similar poderia ser feita através da sequência de transações:

```
create A : int;
update A := 5;
```

Esta última forma seria mais explícita, criando primeiro o objeto com um tipo especificado e depois atribuindo-lhe seu valor.

```
query A;
```

Consulta o valor do objeto A, no caso o número inteiro 5;

O sistema SECONDO apesar de ter um Banco de Dados para armazenamento, trabalha com o conceito de objetos. Uma relação (tabela) não é um objeto “nativo”, como seria em um SGBD convencional que só trabalha com este tipo de dados, mas um objeto que é acessado por uma álgebra específica. Desta forma, o sistema permite armazenar objetos das mais variadas naturezas: números, relações, índices,...

O comando:

```
query MyRel;
```

Consulta a relação (tipo `rel`) `MyRel` (préviamente definida). Cada linha da tabela é mostrada como um grupo de valores separados por uma linha em branco; Esta consulta pode ser melhor visualizada pela GUI, conforme pode ser vista na Figura 9. Neste caso, o sistema optou por um tipo de visualizador próprio para tabelas.

Por último, a consulta:

```
query MyRel count;
```

Aplica o operador `count` ao objeto `MyRel`, resultando no número de linhas armazenadas na tabela: 3.

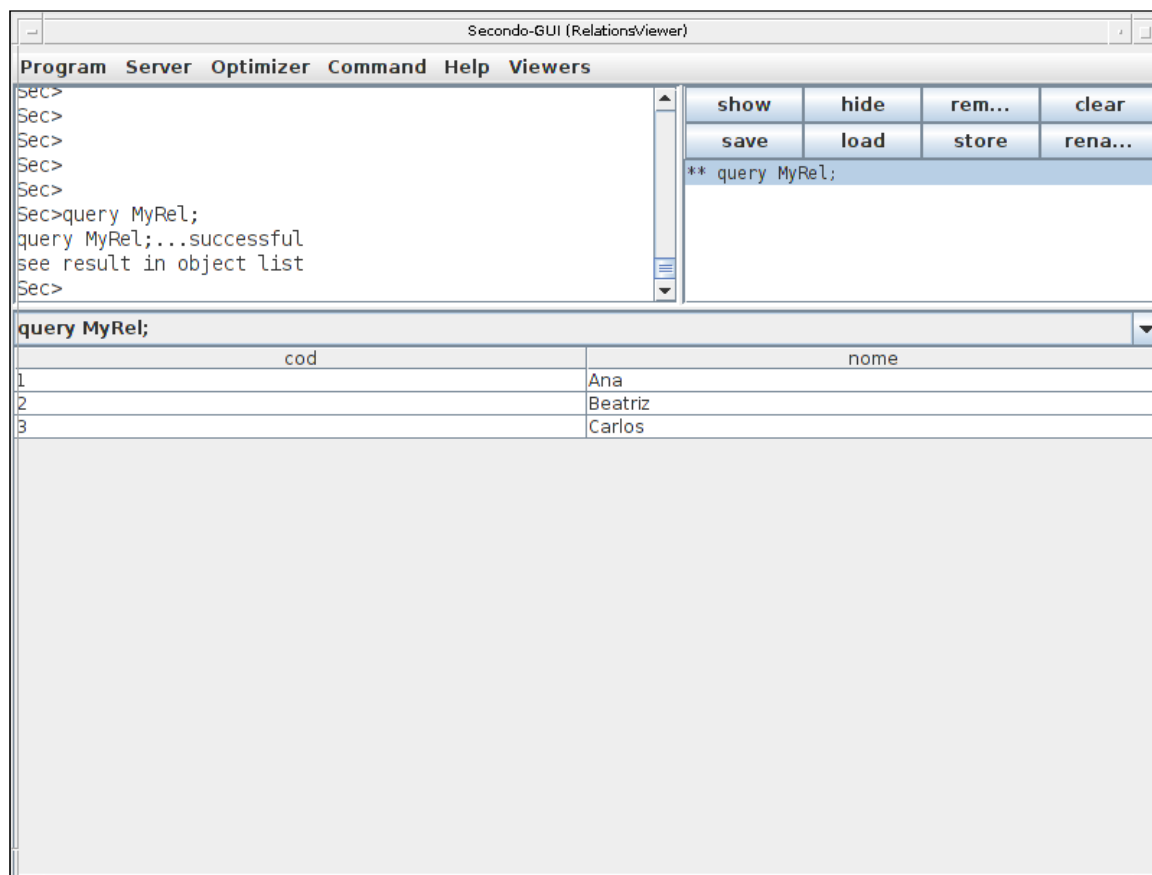


Figura 9 – Exemplo de consulta à tabelas (relações)

4.4.1 Conceito de *Stream*

No paradigma do sistema SECONDO, o conceito de *Stream* (fluxo de dados^{NT}) é bastante utilizado. Pragmaticamente, cada transação é composta por diversos módulos desacoplados que obedecem a um sistema de *pipelining* em que cada instância do problema de um é passado como instância de entrada do próximo.

O termo “instância do problema” é utilizado, pois pode-se imaginar uma tabela como sendo composta por várias linhas, cada linha seria uma “instância” do “problema” tabela.

No caso de *Streaming*, imaginando-se uma cadeia de operadores acoplados, existem: um operador que inicia o processo, produzindo cada instância; operadores que

^{NT} Tradução livre

processam cada instância e passam o resultado para o próximo operador; e um operador que consome a instância “pronta” e as acumula para produzir o resultado final.

O operador que inicia o processo chama-se *feed*. Os operadores que consomem as instancias, ou seja, atuam como escoadouros (*sinks*), são o *consume* e o *count*. No momento de execução, o operador escoadouro é avaliado e este solicita (através de um comando `REQUEST`⁸) uma instância ao operador anterior da cadeia, que por sua vez faz o mesmo, e assim sucessivamente, até ser avaliado o operador que iniciou o processo (*feed*). Este responde com um comando `YIELD` para o operador posterior, que então processa a instância passada e gera, também, um comando `YIELD` para o próximo. O processo termina quando não existem mais instâncias a serem processadas e o operador *feed* responde com `CANCEL`. Este comando é então replicado pelos demais operadores até chegar ao operador escoadouro (*sink*).

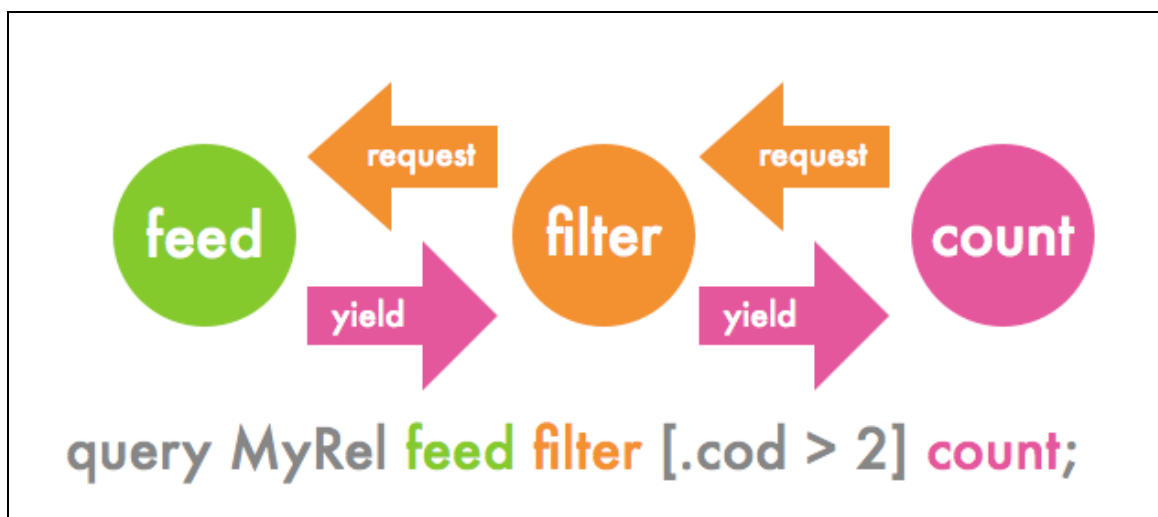


Figura 10 - Exemplo de transação utilizando Streams

A Figura 10 mostra este processo graficamente, e a Figura 11 demonstra, na prática, como são feitas consultas utilizando estes operadores;

⁸ `REQUEST`, `YIELD` e `CANCEL` são, efetivamente, mensagens internas da API de programação do sistema.

```

File Edit View Search Terminal Help
Secondo => query MyRel feed filter [.cod >= 2] count
Secondo ->
Total runtime ... Times (elapsed / cpu): 0.067278sec / 0sec = inf
2
Secondo => query MyRel feed filter [.cod >= 2] project [nome] consume;
Total runtime ... Times (elapsed / cpu): 0.21288sec / 0.01sec = 21.288

nome:Beatriz
nome:Carlos

Secondo => query MyRel feed project [nome] filter [.cod >= 2] consume;
Total runtime ... Times (elapsed / cpu): 0.032551sec / 0sec = inf

Type map error for operator attr!
-----
Input: ((tuple ((nome string))) cod)
-----
Error Message(s):
-----
RelationAlgebra: Attribute name 'cod' is not known in the tuple.
  Known Attribute(s): ((nome string))
-----

Secondo: Expression not evaluable. (Operator not recognized or stream?)
=> []
Secondo =>

```

Figura 11 - Exemplo de utilização dos operadores feed, count e consume

Na primeira transação:

```
query MyRel feed filter [.cod >= 2] count;
```

é utilizado o comando `filter`, que recebe linhas (instâncias) de uma tupla, produzindo um resultado booleano (verdadeiro ou falso), a partir da avaliação da expressão entre colchetes. A expressão `[.cod >= 2]` identifica tuplas, cujo atributo `cod`, possui valores maiores ou iguais a 2; caso o valor avaliado seja verdadeiro, a tupla é passada a diante (`YIELD`) para o próximo operador da cadeia (`count`), caso contrário, a tupla é ignorada e o operador `filter` faz um novo `REQUEST` para o operador `feed` até receber um `CANCEL`. O operador `count` “consume” todas as tuplas gerando, a contagem das tuplas recebidas.

A segunda transação:

```
query MyRel feed filter [.cod >= 2] project [nome] consume;
```

encadeia o operador `project` à transação, após o operador `filter`. Este operador implementa a operação de projeção da álgebra relacional [27] à cada tupla recebida, antes de repassá-la ao operador `consume`, que transforma um `stream(tuple(...))` em uma `rel(tuple(...))`, ou seja, cria uma tabela com as instâncias recebidas.

A terceira transação:

```
query MyRel feed project [nome] filter [.cod >= 2] consume;
```

apresenta um erro^{9 10}. Desta vez a operação de filtragem é realizada após a operação de projeção. O objeto fornecido pelo comando `feed` é do tipo `stream(tuple(cod:int,nome:string))`; O operador `project` extrai apenas o atributo `nome`, fornecendo ao próximo operador (no caso `filter`) `stream(tuple(nome:string))`. O parâmetro de avaliação do comando `filter` requer a presença do atributo `cod`, que não está presente na tupla. Daí o erro e a impossibilidade de se executar a consulta.

A comutatividade dos operadores depende, também, do contexto da operação. Em alguns casos essa comutação é permitida. Supondo-se que as tuplas do exemplo acima tivessem outros atributos que fossem incluídos na projeção e avaliados pela filtragem, talvez a comutação fosse possível.

⁹ O erro refere-se ao operador `attr` (atributo). Este operador é representado sintaticamente através do caracter “.” e recebe como parâmetro uma tupla e um identificador.

¹⁰ O `SECONDO` utiliza internamente um formato de listas similar à linguagem `LISP`. Existe um tratamento sintático dos comandos para facilitar a interface com o sistema, porém, os erros, frequentemente, são mostrados em utilizando-se esta sintaxe menos agradável.

O otimizador do SECONDO permite a consulta à base utilizando a linguagem SQL. Neste caso ele avalia a possibilidade de comutação dos operadores como forma de se otimizar a consulta. Se no exemplo anterior existisse um comando `sort`, o mesmo deveria ser realizado antes da projeção e filtragem, porém a comutação dos dois últimos operadores seria avaliada.

É importante salientar que não existe nenhuma garantia de cardinalidade em uma consulta com *streams*. Como o próprio exemplo mostra, o comando `filter` altera a cardinalidade do resultado. Esta propriedade será de grande uso para os operadores propostos neste trabalho.

5 Operadores Propostos para séries temporais

Dado o problema apresentado no capítulo 2, as ferramentas disponíveis apresentadas no capítulo 3 e a escolha do ambiente de desenvolvimento e validação mostrado no capítulo **Error! Reference source not found.**, serão apresentados, neste capítulo, os operandos e operadores sugeridos como forma de solução para o problema.

À princípio, será apresentada uma definição de Série Temporal, de um ponto de vista conceitual, junto com possíveis operadores para esta proposta. Esta primeira abordagem servirá de base para um segundo passo, no qual esta Série Temporal será adequada aos modelos existentes.

Para isto, propõe-se a definição de outros dois tipos de operandos, um que representa o período de tempo de uma série temporal e outro que representa os valores da série em si. Neste capítulo, eles serão apresentados respectivamente, nesta ordem, pois facilitará o entendimento do trabalho. Porém, no próximo capítulo, esta ordem será invertida de modo a facilitar o entendimento da sua utilização.

5.1 Séries Temporais

5.1.1 Sinais, Períodos de Tempo e Amostragem

Dissertar sobre a definição teórica de sinais, amostragem, e, principalmente, a natureza do tempo em si, é uma tarefa hercúlea e, já abordada por outras áreas (Física, Eletrônica, Lógica e até Filosofia). Cabe a este trabalho escolher uma definição consistente que possa ser levada a cabo pela implementação.

Dado um sinal s , pertencente ao domínio temporal, e um instante t , dizemos que uma **amostra** deste sinal corresponde a um valor v tal que $v = s(t)$. Apesar de correta, esta definição, sozinha, é de pouco uso, pois supõe-se que se uma série temporal será composta por um conjunto de amostras.

Portanto, existem outros dados que são também de interesse, como a frequência de aquisição do dado ou seu período e o número de amostras, para que se crie uma linha de tendência consistente.

Alguns destes dados são dependentes uns dos outros. O período e a frequência são inversos entre si. Logo, tendo-se o período, tem-se a frequência. O número de amostras é obviamente obtido através da cardinalidade dos dados. Desta forma, escolheu-se utilizar uma representação para uma **amostra (v)** como sendo o valor do sinal em um intervalo de tempo fechado à esquerda e aberto à direita tal que:

$$v[t_0, t_1[= s(t_0).$$

Vale notar que esta definição é arbitrária. O intervalo poderia ser aberto à esquerda e fechado à direita, representando o valor final, ou ainda, aberto em ambas extremidades e representar o valor no meio do intervalo. Na prática, as três representações são válidas e, caso levem a conclusões diferentes, pode ser facilmente conjecturado que um aumento da taxa de amostragem levem-nas, por fim, a convergirem.

É necessário, porém, que o intervalo deva ser aberto, ao menos, em uma de suas extremidades, pois, caso contrário, pode ser suposta a existência de um sinal que tenha dois valores distintos em um determinado instante. Esta definição, infelizmente, inviabiliza o uso do sistema em experimentos que observem paradoxos da Física Quântica. Esta restrição, porém, não chega a ser impeditiva quanto ao universo de seu uso potencial.

Ao analisarmos a definição acima, notamos que o período pode ser obtido subtraindo-se t_0 de t_1 . Por conseguinte, a frequência pode ser obtida invertendo-se este valor.

5.1.2 Definições de séries temporais

A partir da definição do que constitui uma amostra, podemos definir uma **série temporal V** como sendo um conjunto de amostras $\{v_1, v_2, \dots, v_n\}$. A princípio, não existe nenhuma restrição à frequência e ordem destas amostras, portanto definiremos uma **série temporal ordenada VO** como sendo uma série temporal em que:

$$t_0(v_{k+1}) \geq t_1(v_k), \forall k, 1 \leq k < n.$$

por fim definiremos uma série **temporal ordenada e sincronizada VOS** como sendo uma série temporal ordenada (VO) na qual:

$$t_0(v_{k+1}) = t_1(v_k), \forall k, 1 \leq k < n \text{ e,}$$

$$\forall j, k, 1 \leq j, k \leq n \mid t_1(v_j) - t_0(v_j) = t_1(v_k) - t_0(v_k)$$

Ou seja, uma **série temporal ordenada e sincronizada** contém valores em série, ordenados, adjacentes e adquiridos na mesma taxa. Doravante, no contexto deste trabalho, o termo série temporal se referirá à séries temporais ordenadas e sincronizadas.

Dada esta definição, podemos então definir o **período** de uma série temporal como sendo a união de todos os períodos de suas amostras. Pela propriedade que cada período tem de ser fechado à esquerda e aberto à direita, é fácil concluir que o período da série também apresentará esta propriedade.

A **cardinalidade** de uma série temporal será definida como o **número de amostras desta série**. Duas séries temporais serão consideradas síncronas, quando tiverem o mesmo período e a mesma cardinalidade.

Por definição, não serão consideradas séries temporais vazias, ou seja, de cardinalidade igual a zero. Também não fazem sentido séries temporais com cardinalidade negativa.

5.1.3 Operadores para Séries Temporais

Podemos dividir os operadores de séries temporais em dois grupos: os que geram séries temporais como resultados e os que produzem valores em outros domínios. No primeiro grupo, estão operadores como união, interseção, interpolação e, no segundo, soma, média, cardinalidade e transformadas, como a de Fourier, por exemplo.

Os operadores do segundo grupo não necessitam de um formalismo teórico extenso. Como a imagem^{11 12} do resultado não pertence ao mesmo domínio dos termos, eles não influem na consistência de uma possível álgebra das séries temporais apresentadas até este ponto. Obter-se uma média a partir de uma série temporal vazia, o que resultaria em uma inconsistência (valor indefinido), não é permitido por definição. Posteriormente, quando for apresentado o modelo que será implementado, serão necessárias maiores considerações.

Por outro lado, os operadores que produzem séries temporais como resultados requerem maior formalismo.

5.1.3.1 Operador união

A união de séries temporais só é possível quando as mesmas forem adjacentes, isto é: $t_1(V_0) = t_0(V_1)$ e o período de ambas as séries for o mesmo. É trivial demonstrar que este operador não é comutativo, porém é associativo.

¹¹ Termo utilizado no sentido de conjunto imagem de uma função.

¹² Até então não foi definido nenhum domínio para cada amostra, pode-se supor tanto o domínio de números reais, como qualquer outro.

5.1.3.2 Operador Interseção

Este operador apresenta um problema quando a interseção de duas séries é vazia, pois não é definida uma série temporal vazia. A solução para este problema é a utilização de um operador `teste_de_interseção`, que produz um resultado booleano indicando se existe uma interseção entre as séries. Este operador também deve considerar o período das mesmas, pois só pode ser calculada uma interseção entre duas séries se os períodos forem iguais. Também é trivial a demonstração de que este operador é comutativo e associativo.

5.1.3.3 Operador Interpolação

Trata-se de um operador unário¹³ que recebe uma série temporal e um número inteiro k , positivo, e produz uma série temporal de mesmo período e cardinalidade igual a k , contendo os valores interpolados da série original. Como a função de interpolação não é definida, a consistência das amostras (com seu domínio¹²) deve ser provada à parte.

5.1.3.4 Operador Sub-Série

Operador unário, como o anterior, que retorna uma série temporal com um período contido ou igual ao período do operando e com o número de amostras correspondentes. A forma de indexação da série original não é definida, podendo ser tanto por um período de tempo ou por um subconjunto de amostras. Requer um operador adicional de teste, assim como o operador interseção, para certificar-se que o resultado pode ser definido (isto é, séries não-vazias). Este operador é bastante similar ao interseção.

¹³ O termo unário é utilizado aqui, pois apenas um operando do domínio de séries temporais é processado, o outro parâmetro pode ser visto como uma constante, notadamente: $op_k V$

5.1.4 Consistência dos Operadores

Como pode ser notado, apesar desta álgebra tentar ser correta e completa, algumas inconsistências são insolúveis. A possibilidade de gerar séries temporais vazias com o operador interseção, nos leva a perguntar se poderíamos relaxar esta premissa. Caso o fizéssemos, teríamos problemas com o operador interpolação. O que seria a interpolação de uma série vazia ? Outra questão é, como seria a representação temporal de uma série vazia ? Qual o sentido de um período de tempo nulo ? Dada a representação do período como dois instantes, como pode existir um evento cujo momento de início é o mesmo de seu fim ? Estes são alguns dos paradoxos criados por este relaxamento.

Uma solução parcial para este problema é a criação do valor **indefinido**. Neste caso temos um evento com duração definida, que não foi adquirido. Esta solução nos permite fechar o operador união quanto à sua completeza. A união de duas séries temporais não adjacentes, passa a conter uma série de elementos indefinidos correspondentes ao período entre as séries.

É importante salientar, que o ideal seria a utilização de uma nomenclatura diferente para o valor indefinido no sentido de não observado, faltante, e do valor indefinido por outros motivos, como, por exemplo, fruto de uma divisão por zero ou infinito. Por questões práticas, ambos os sentidos serão tratados da mesma forma.

5.2 Adequação ao Modelo Relacional

A partir de uma proposta conceitual de Séries Temporais, podemos imaginar como esta deve ser adequada a um modelo relacional para obtermos resultados práticos. À princípio, podemos imaginar que um experimento, ao ser observado, gerará não uma, mas várias séries simultaneamente. Estas séries podem estar sincronizadas (na mesma base de tempo) ou não. Também, podem conter o mesmo número de amostras ou não. Uma decisão sensata seria a de separar a base de tempo das amostras em si.

5.2.1 Tipo TimeSpan

Poderíamos imaginar uma tabela em que a base de tempo (período de coleta) seria um dos atributos, enquanto cada conjunto de amostras seria um atributo a parte. A correlação de um conjunto de amostras com sua base de tempo passa a formar uma relação no sentido semântico.

Esta abordagem permite também a normalização dos dados e uma certa economia, uma vez que só é necessário armazenar uma informação temporal para vários conjuntos de dados.

Levando-se em conta também a questão de indexação, foi criado no Sistema SECONDO, um tipo chamado `TimeSpan`. Apesar do sistema já ter tipos implementados para instantes e durações, o tipo `TimeSpan`, devido às suas características de implementação, permite o uso da álgebra de *R-Trees*, permitindo sua indexação, como se fosse uma região geométrica. Afinal, pode-se imaginar um intervalo de tempo como sendo um segmento de reta.

Este tipo implementa os operadores união, interseção e sub-série, propostos anteriormente. Uma visão mais completa de toda a implementação será mostrada no próximo capítulo.

5.2.2 Limitações Práticas

Um dos problemas no armazenamento de séries temporais é a grande quantidade de dados gerados. Um experimento pode levar apenas alguns minutos, como também alguns meses e anos. Testes de desgaste e corrosão são, por exemplo, experimentos de longa duração. Não há uma correlação direta entre tempo do experimento e quantidade de dados gerados, um minuto de um teste em um acelerador atômico pode gerar mais dados do que um ano de um teste de desgaste, por que temos frequências de aquisição altíssimas no primeiro e baixíssimas no segundo.

Para se trabalhar com a série de forma consistente, seria interessante poder tratá-la como um todo e não em partes. Porém, surge outro problema, que é o da alocação do espaço de armazenamento. Para esta tarefa, é interessante que a série seja tratada em partes e não como um todo. Por fim, podemos citar as diferenças na forma de gravação e na forma de recuperação dos dados. Durante o momento de aquisição, todos os sensores são consultados, simultaneamente, e cada um deles responde com uma amostra. Ou seja, uma linha de uma tabela é gerada a cada consulta e é conveniente que seja gravada assim. No entanto, no momento da consulta aos dados gravados, normalmente se recuperará toda, ou grande parte, da série de um sensor por vez. Ou seja, será recuperada uma coluna. Apesar desses problemas não serem ligados ao modelo relacional e sim a implementação, seria interessante tentar abordá-los na solução.

5.2.3 Tipo RealArray e operadores Trans-Relacionais

Como proposta de solução aos limites do modelo relacional, foi criado o tipo `RealArray`. A princípio, este tipo implementa apenas um arranjo (*array*) de tipos `real` de cardinalidade (número de elementos) variável. A diferença se encontra na possibilidade de reorganizar este *array* em colunas de números reais e vice-versa.

Esta operação de reorganização do `RealArray` é chamada de **Trans-Relacional**, por que transpõe os limites do modelo relacional. Tomemos, por exemplo, as duas tabelas mostradas abaixo. Na Tabela 8, temos dois atributos do tipo `RealArray`, o primeiro, `S1`, com 5 elementos e o segundo, `S2`, com 2 elementos.

Tabela 8 - Tabela com RealArray

S1: RealArray	S2: RealArray
[1, 2, 3, 4, 5]	[11, 12]

Tabela 9 – Tabela Trans-Relacionada

S1: real	S2: real
1	11
2	12
3	Undef
4	Undef
5	Undef

Após aplicarmos o operador trans-relacional “`column`”, a Tabela 8 será transformada na Tabela 9. Transformando o `RealArray` em uma coluna de números reais. Existe um operador inverso a este, chamado “`uncolumn`” que surtirá o efeito contrário, agrupando as colunas de números reais em `RealArrays`.

Estes operadores são muito similares às operações de *Map / Reduce* utilizadas por *workflows* [67]. Porém, as operações de *Map / Reduce* criam outras tabelas e utilizam

tipos convencionais em seus resultados, enquanto os operadores **trans-relacionais** atuam diretamente no *schema* dos resultados, criando um mapeamento direto entre os tipos `RealArray` \leftrightarrow `real`.

5.2.4 Valores Indefinidos

Podemos ver, na Tabela 9, que algumas colunas apresentam atributos com valores indefinidos. Conforme discutido na seção anterior, a álgebra foi relaxada para permitir o uso destes valores. Para números reais, estes valores já vêm implementados pelo sistema. Uma divisão por zero, por exemplo, resulta em um valor indefinido. Eles também são necessários, pois a criação de um objeto pode ser feita independentemente da atribuição de seu valor. Neste caso, o objeto também terá um valor indefinido.

5.2.5 Operadores Trans-Relacionais Semânticos

Devida à separação realizada entre os tipos `TimeSpan` e `RealArray`, em que o primeiro trata da base temporal e o segundo dos valores armazenados, foi necessário também separar os operadores **trans-relacionais** em dois grupos: Os que atuam apenas sobre o tipo `RealArray`, alterando sua representação; e os que atuam sobre este tipo, tomando por referência um atributo tipo `TimeSpan`; estes últimos são chamados de **trans-relacionais semânticos**, pois aplicam às operações uma **semântica de série temporal**. Isto quer dizer que questões como adjacência e taxa de aquisição passam a ser levadas em conta para o resultado da operação.

No capítulo seguinte serão mostradas as implementações para os operadores que lidam com `RealArray`, os simples (soma, média,...), os trans-relacionais e os trans-relacionais semânticos

6 Implementação da Proposta

Neste capítulo, será apresentado o que foi implementado. Optou-se por demonstrar os tipos e operadores de uma forma “construtivista”, de modo a facilitar o entendimento. Embora a programação tenha sido realizada, não serão abordadas questões sobre a programação dos mesmos. Estas estão de acordo com o Guia de Programação do SECONDO [24]. Discussões sobre a estrutura do sistema estão fora do escopo deste trabalho.

6.1 Tipo *RealArray*

Este tipo é o cerne da implementação da proposta. Apesar de armazenar apenas um arranjo de números reais, ele possui a propriedade **Trans-relacional** de se transformar em um atributo `real` e vice-versa.

6.1.1 Operadores Aritméticos

Para a manipulação algébrica deste tipo, foi criada uma série de atributos que permitem operar os seus elementos de forma simples. Soma, Subtração, Multiplicação e Divisão, por exemplo.

Estes atributos foram construídos tanto para operações binárias entre dois `RealArrays`, quanto para operações entre `RealArrays` e números escalares (inteiros ou reais).


```

File Edit View Search Terminal Help
Secondo =>
Secondo => query ra1;
Total runtime ... Times (elapsed / cpu): 0.002386sec / 0sec = inf

[ 1 ; 2 ; 3 ; 4 ; 5 ]
Secondo => query ra2;
Total runtime ... Times (elapsed / cpu): 0.002332sec / 0sec = inf

[ 2 ; 3 ; 4 ]
Secondo => let ra3 = ra1 + ra2;
Total runtime ... Times (elapsed / cpu): 0.06922sec / 0.01sec = 6.922

=> []
Secondo => query ra3;
Total runtime ... Times (elapsed / cpu): 0.002365sec / 0sec = inf

[ 3 ; 5 ; 7 ; UNDEFINED ; UNDEFINED ]
Secondo => query ra4;
Total runtime ... Times (elapsed / cpu): 0.002471sec / 0sec = inf

[ 0 ; 1 ; 1 ; 1 ; 0 ]
Secondo => let ra5 = ra1 / ra4;
Total runtime ... Times (elapsed / cpu): 0.121959sec / 0sec = inf

=> []
Secondo => query ra5;
Total runtime ... Times (elapsed / cpu): 0.002444sec / 0sec = inf

[ UNDEFINED ; 2 ; 3 ; 4 ; UNDEFINED ]
Secondo =>

```

Figura 12 - Operadores Aritméticos para RealArray

Acompanhando-se a Figura 12¹⁴, podemos ver as duas primeiras transações:

```
query ra1;
```

```
query ra2;
```

Estas mostram o conteúdo dos dois objetos `ra1` e `ra2` do tipo `RealArray`. Podemos notar que estes dois objetos são de tamanho (cardinalidade) diferentes. `ra1 = [1;2;3;4;5]` (possui 5 elementos) enquanto que `ra2 = [2;3;4]` (possui 3 elementos).

¹⁴ Apesar do sistema `SECONDO` apresentar o tempo decorrido (*elapsed time*) da consulta, o mesmo não deve ser levado em conta pois não serão feitas comparações da performance do sistema. Trata-se apenas de uma configuração *default* do sistema que os apresenta após cada consulta.

As próximas linhas:

```
let ra3 = ra1 + ra2;  
query ra3;
```

criam um novo `RealArray` chamado `ra3`, através do operador `let`, com o resultado da expressão `ra1 + ra2` e consulta o resultado criado. Este novo objeto (`ra3`) possui 5 elementos, sendo os dois últimos indefinidos e criados a partir da operação de soma dos dois últimos elementos de `ra1` com elementos indefinidos, pois `ra2` não possuía elementos suficientes para a operação. A cardinalidade de `ra3` será discutida posteriormente.

Na sequência, é apresentado o objeto `ra4`, através do comando:

```
query ra4;
```

E o objeto `ra5` é criado através de uma operação de divisão com `ra1`. Como `ra4` possui elementos de valor zero, e a divisão por zero não é definida, elementos indefinidos são criados por este motivo, como pode ser visto nas transações:

```
let ra5 = ra1 / ra4;  
query ra5;
```

6.1.2 Operadores Booleanos

Foram criados uma série de operadores booleanos: maior que, menor que, maior ou igual, menor ou igual e diferente, para a comparação de `RealArrays`. Nestes casos, apenas tipos `RealArray` são permitidos. A comparação é feita lexicograficamente. O primeiro elemento de um objeto é comparado com o primeiro elemento do outro objeto e assim sucessivamente. Elementos indefinidos são considerados como menores do que quaisquer outros, a não ser eles mesmos. A operação termina quando todos os elementos são comparados, uns com os outros, ou quando um resultado falso é atingido.

```

File Edit View Search Terminal Help
Secondo => query ra1;
Total runtime ... Times (elapsed / cpu): 0.002377sec / 0sec = inf

[ 1 ; 2 ; 3 ; 4 ; 5 ]
Secondo => query ra2;
Total runtime ... Times (elapsed / cpu): 0.002346sec / 0sec = inf

[ 2 ; 3 ; 4 ]
Secondo => let ra6 = ra2 - 1;
Total runtime ... Times (elapsed / cpu): 0.076809sec / 0sec = inf

=> []
Secondo => query ra6;
Total runtime ... Times (elapsed / cpu): 0.002482sec / 0sec = inf

[ 1 ; 2 ; 3 ]
Secondo => query ra1 = ra6;
Total runtime ... Times (elapsed / cpu): 0.002942sec / 0sec = inf

FALSE
Secondo => query ra1 > ra6;
Total runtime ... Times (elapsed / cpu): 0.002818sec / 0sec = inf

TRUE
Secondo => query ra1 > ra2;
Total runtime ... Times (elapsed / cpu): 0.002906sec / 0sec = inf

FALSE
Secondo =>

```

Figura 13 - Exemplo de operadores booleanos

Na Figura 13, são mostrados novamente os objetos `ra1` e `ra2`. O objeto `ra6` é criado através do operador subtração utilizando-se um escalar. Neste caso o escalar é operado com todos os elemento do `RealArray` através do comando:

```
Let ra6 = ra2 - 1;
```

Em seguida, operações de comparação são realizadas. Podemos ver que `ra1` é diferente de `ra6` e é também maior. Porém `ra1` não é maior que `ra2` pois o primeiro elemento de `ra1` é o número 1 e o primeiro elemento de `ra2` é o número 2;

6.1.3 Operadores Aritméticos Unários Horizontais

Por se tratarem de tipos “agregados”, foram projetados alguns operadores aritméticos unários de modo a extraírem resultados escalares a partir de seu conteúdo. Eles são chamados de operadores “horizontais” por tratarem o arranjo como um vetor linha. São eles: `Hsum` (soma), `Havg` (média) e `count` (cardinalidade). Este último não recebeu o prefixo “H”.

Como demonstrado na Figura 14, a primeira linha utiliza o comando:

```
query ra1 Hsum;
```

para totalizar todos os elementos do objeto `ra1`. O objeto em questão é mostrado na linha inferior. Quando se consulta a soma dos elementos do objeto `ra3`, que contém elementos indefinidos, através do comando:

```
query ra3 Hsum;
```

podemos ver que os elementos indefinidos não são computados. Este comportamento é diferente do que se esperaria se somássemos os elementos 1 a 1, pois neste caso o resultado todo seria indefinido.

O operador `count` também não computa elementos indefinidos em sua contagem. Isto é necessário para se manter a consistência com o operador `Havg`.

Podemos notar que o comando:

```
query ra3 Havg;
```

também ignora os indefinidos, isto é não os contabiliza. Portanto o comando:

```
query ra3 Hsum / ra3 count;
```

produz o mesmo resultado do comando anterior.

```

File Edit View Search Terminal Help
Secondo =>
Secondo => query ra1 Hsum;
Total runtime ... Times (elapsed / cpu): 0.002578sec / 0sec = inf

15
Secondo => query ra1;
Total runtime ... Times (elapsed / cpu): 0.002367sec / 0sec = inf

[ 1 ; 2 ; 3 ; 4 ; 5 ]
Secondo => query ra3;
Total runtime ... Times (elapsed / cpu): 0.002512sec / 0sec = inf

[ 3 ; 5 ; 7 ; UNDEFINED ; UNDEFINED ]
Secondo => query ra3 Hsum;
Total runtime ... Times (elapsed / cpu): 0.002619sec / 0sec = inf

15
Secondo => query ra1 count
Secondo ->
Total runtime ... Times (elapsed / cpu): 0.002626sec / 0sec = inf

5
Secondo => query ra3 count;
Total runtime ... Times (elapsed / cpu): 0.002558sec / 0sec = inf

3
Secondo => query ra1 Havg;
Total runtime ... Times (elapsed / cpu): 0.0025sec / 0sec = inf

3
Secondo => query ra3 Havg;
Total runtime ... Times (elapsed / cpu): 0.002511sec / 0sec = inf

5
Secondo => query ra3 Hsum / ra3 count;
Total runtime ... Times (elapsed / cpu): 0.003062sec / 0sec = inf

5
Secondo =>

```

Figura 14 - Operadores Aritméticos Unários

6.1.4 Utilização em Tuplas

Como já foi mencionado antes, o tipo `RealArray` pode ser utilizado em tuplas e relações no sistema `SECONDO` pois representa um *kind DATA*. Um exemplo de uma relação chamada `RelA`, foi criado e pode ser vista na Figura 15.

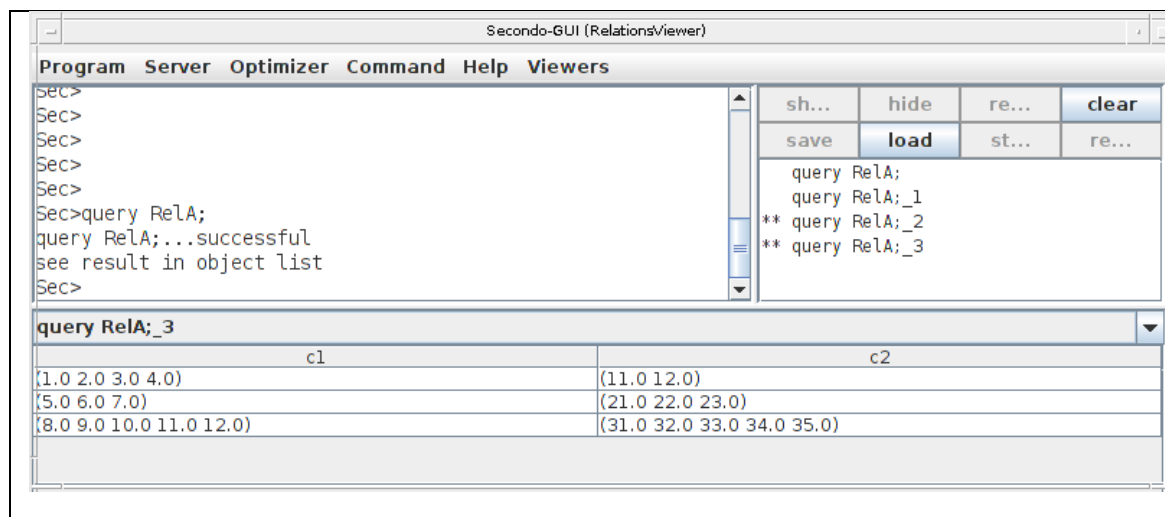


Figura 15 - Exemplo de Relação com atributos RealArray

Esta relação é composta de dois atributos (*c1* e *c2*), ambos do tipo *RealArray*. Em cada linha pode ser visto o valor de cada atributo. Não é necessário que cada atributo tenha a mesma cardinalidade.

6.1.5 Operadores Aritméticos Unários Verticais

Operadores “verticais” são aqueles que operam entre linhas de tabelas. Apesar de já serem conhecidos e implementados em vários SGBDs e Álgebras como operadores de agregação, o termo “vertical” é utilizado em contraponto com o termo “horizontal” apresentado anteriormente. São eles: *sum* (soma) e *avg* (média). O prefixo “V” foi omitido de modo a manter a consistência com a sintaxe já existente do SECONDO.

```

File Edit View Search Terminal Help
Secondo => query ReLA feed sum[c1];
[ 14 ; 17 ; 20 ; 15 ; 12 ]
Secondo => query ReLA feed sum[c2];
[ 63 ; 66 ; 56 ; 34 ; 35 ]
Secondo => query ReLA feed avg[c1];
[ 4.6666666667 ; 5.6666666667 ; 6.6666666667 ; 7.5 ; 12 ]
Secondo => query ReLA feed avg[c2];
[ 21 ; 22 ; 28 ; 34 ; 35 ]
Secondo =>

```

Figura 16 - Exemplo de utilização de operadores verticais

Na Figura 16, podem ser vistos exemplos de utilização destes dois operadores com as duas colunas da tabela. Uma consulta utilizando-se estes operadores é de sintaxe bastante simples:

```
query ReLA feed sum[c1];
```

Que indica que desejamos somar a coluna `c1` da relação `ReLA`.

Pode-se notar a presença do comando `feed`, indicando que o operador trabalha sobre *streams* (ver seção 4.4.1) de tuplas e não sobre a relação em si. Como trata-se de um operador do tipo *sink*, o objeto final não é uma relação mas sim um objeto do tipo `Realarray`.

Dada a diferença de cardinalidade das diversas linhas da tabela, estes operadores tratam elemento indefinidos tais quais seus equivalentes “horizontais”, desconsiderando-os do cômputo para somas e médias.

Esta consistência faz sentido, embora no caso do operador `avg`, os resultados possam parecer contraditórios.

```

File Edit View Search Terminal Help
Secondo => query ReLA feed extend[soma: .c1 Hsum] sum[soma];

78
Secondo => query ReLA feed sum[c1] Hsum;

78
Secondo => query ReLA feed extend[media: .c1 Havg] avg[media];

6.1666666667
Secondo => query ReLA feed avg[c1] Havg;

7.3
Secondo => query ReLA feed avg[c1];

[ 4.6666666667 ; 5.6666666667 ; 6.6666666667 ; 7.5 ; 12 ]
Secondo => query ReLA feed project [c1] extend [media: .c1 Havg] consume;

    c1:[ 1 ; 2 ; 3 ; 4 ]
media:2.5

    c1:[ 5 ; 6 ; 7 ]
media:6

    c1:[ 8 ; 9 ; 10 ; 11 ; 12 ]
media:10

Secondo =>

```

Figura 17 - Exemplo de comutatividade de operadores horizontais e verticais

Na Figura 17, são apresentados exemplos que mostram a comutatividade destes operadores. O primeiro exemplo:

```
query ReLA feed extend [soma: .c1 Hsum] sum [soma];
```

cria um novo atributo em cada linha (uma extensão da tabela) chamado `soma`, a partir da soma horizontal do atributo `c1`. Em seguida, estes novos atributos são totalizados.

O segundo exemplo:

```
query ReLA sum[c1] Hsum;
```

é o exemplo comutativo da operação. Em ambos os casos podemos conferir que o mesmo resultado, `78`, foi calculado. No terceiro e quarto exemplo, uma operação similar é executada utilizando-se o operador `avg`. Neste caso é claramente visível a incompatibilidade desta operação. Um exemplo calculou a média como sendo `6.16667` e o outro como sendo `7.3`.

Os dois exemplos subsequentes:

```
query RelA feed avg[c1];
```

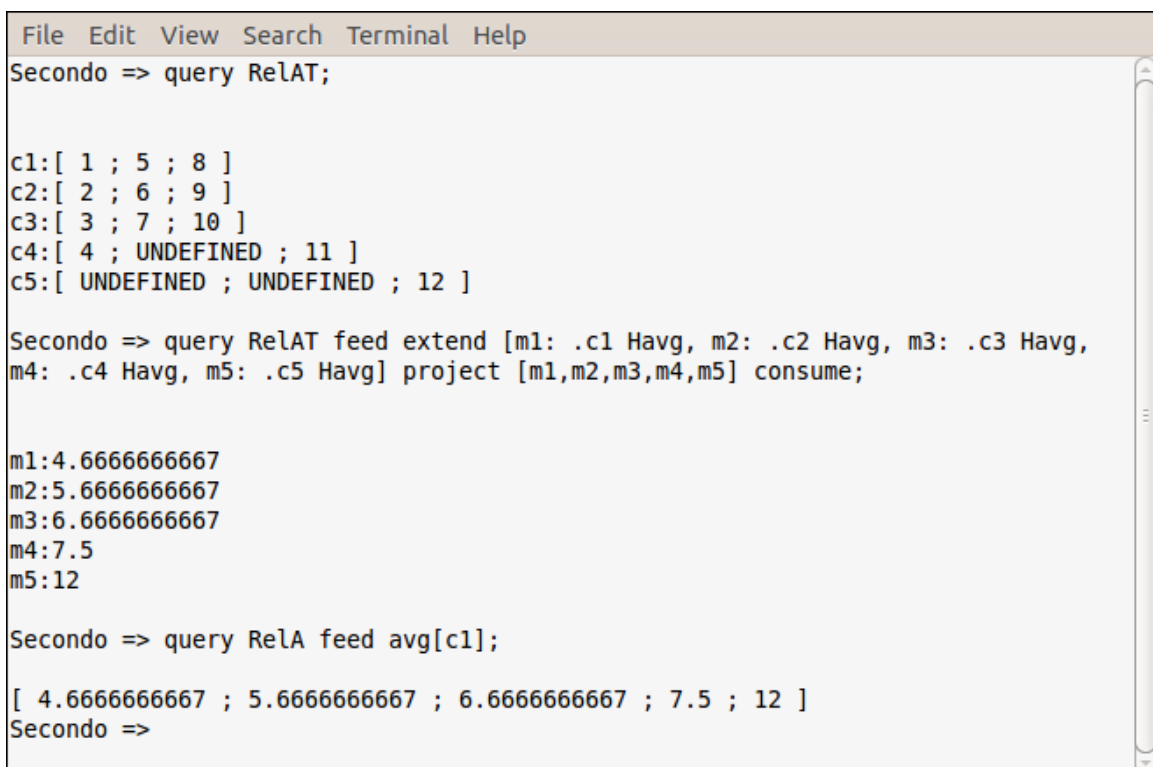
e

```
query RelA feed project [c1] extend [media: .c1 Havg] consume;
```

se destinam a mostrar a base de cálculo destas operações. Podemos ver que as mesmas foram executadas corretamente.

A comutatividade dos operadores `avg` / `Havg`, no entanto, é visível ao se transpor a relação. A possibilidade e o conceito de transposição de relações são pontos fundamentais explorados por este trabalho.

A Figura 18, apresenta a relação `RelA` após ter sua coluna `c1` projetada e transposta¹⁵. Esta nova relação chama-se `RelAT` e possui apenas uma linha com 5 atributos `RealArray`, cada qual de cardinalidade 3, o mesmo número de linhas da tabela anterior.



```
File Edit View Search Terminal Help
Secondo => query RelAT;

c1:[ 1 ; 5 ; 8 ]
c2:[ 2 ; 6 ; 9 ]
c3:[ 3 ; 7 ; 10 ]
c4:[ 4 ; UNDEFINED ; 11 ]
c5:[ UNDEFINED ; UNDEFINED ; 12 ]

Secondo => query RelAT feed extend [m1: .c1 Havg, m2: .c2 Havg, m3: .c3 Havg,
m4: .c4 Havg, m5: .c5 Havg] project [m1,m2,m3,m4,m5] consume;

m1:4.6666666667
m2:5.6666666667
m3:6.6666666667
m4:7.5
m5:12

Secondo => query RelA feed avg[c1];

[ 4.6666666667 ; 5.6666666667 ; 6.6666666667 ; 7.5 ; 12 ]
Secondo =>
```

Figura 18 - Comutatividade da média em relação à transposição

¹⁵ A sintaxe que permite esta operação será apresentada posteriormente.

Seguindo o exemplo acima, uma consulta é realizada calculando-se as médias horizontais de cada atributo (operador `extend`) e projetando-as. Pode ser visto que os atributos calculados (`m1` a `m5`) correspondem aos elementos da média vertical calculada pela terceira consulta.

6.2 Operadores Trans-Relacionais

Nesta seção, serão descritos o primeiro grupo de operadores trans-relacionais implementados. Esta primeira série de operadores lida com tabelas compostas apenas de tipos trans-relacionais, ou seja o tipo `RealArray` e o tipo `real`, que podem ser transformados um no outro. É válido lembrar que sempre pode-se operar sobre uma projeção de uma tabela, de forma que uma tabela que contenha outros atributos, de outros tipos, pode ser projetada para que se utilizem estes operadores. Posteriormente, serão apresentados outros operadores que permitam a operação em tabelas mais diversas. A separação desta seção da posterior foi feita por caráter didático.

Todos os operadores trans-relacionais foram implementados utilizam o conceito de *streaming*, já comentado neste trabalho. Os operadores `feed` e `consume`, que, respectivamente, inicializa um *stream* e o consome (transforma-o em relação), poderão ser notados em todos os exemplos.

Na sequência de exemplos posteriores, serão apresentadas as transformações que podem ser realizadas na tabela `RelA`, apresentada na Figura 15 da seção anterior.

6.2.1 Operador `column`

Este é o operador que transforma um atributo `RealArray` em várias linhas de atributos do tipo `real`. Como a tabela é composta apenas por atributos do primeiro tipo, e os mesmos podem ter cardinalidade diferente, elementos indefinidos são criados de modo manter o mesmo número de linhas criadas por linha processada.

A Figura 19, apresenta um diagrama esquemático do funcionamento do operador.

Podemos notar na assinatura do operador que apenas atributos do tipo `RealArray`

(RA) são permitidos. O “S” representa um *stream* e o “T” uma tupla. O nome dos atributos de saída é copiado a partir dos atributos de entrada.

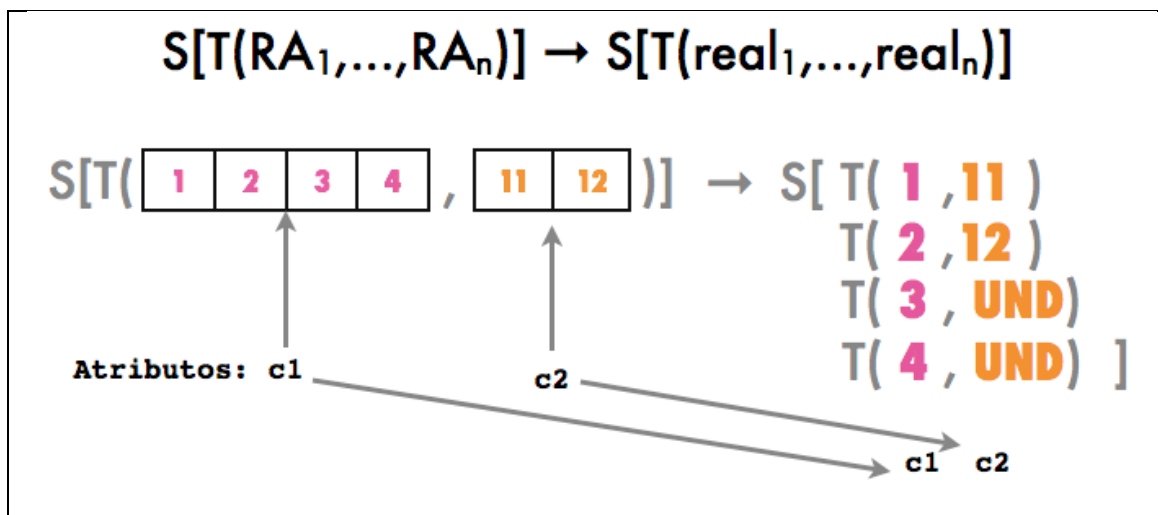


Figura 19 - Funcionamento do operador column

	c1	c2
1.0		11.0
2.0		12.0
3.0		undef
4.0		undef
5.0		21.0
6.0		22.0
7.0		23.0
8.0		31.0
9.0		32.0
10.0		33.0
11.0		34.0
12.0		35.0

Figura 20 – Resultado da utilização do operador column

Na Figura 20, é apresentado um exemplo prático do uso do operador `column` através do comando:

```
query RelA feed column consume;
```

A tabela `RelA` é alimentada para um *stream*, tem seus atributos transformados em colunas e o resultado é reconstituído em uma tabela pelo operador `consume`.

Podemos notar a presença de dois elementos indefinidos, nas linhas 3 e 4, devido à diferença de cardinalidade existente na primeira linha, `c1` com 4 elementos e `c2` com apenas 2.

6.2.2 Operador `uncolumn`

Este operador executa o efeito inverso do operador `column`. Agregando várias linhas com tipos `real` em uma linha contendo `RealArrays`. Sua sintaxe demanda um número inteiro que indicará o tamanho máximo do `RealArray` construído. Caso não existam linhas suficientes para completá-lo, serão incluídos elementos indefinidos ao seu final. É importante lembrar que mesmo que um `RealArray` receba elementos indefinidos, sua cardinalidade é contabilizada pelo comando `count`, que exclui de sua conta estes elementos.

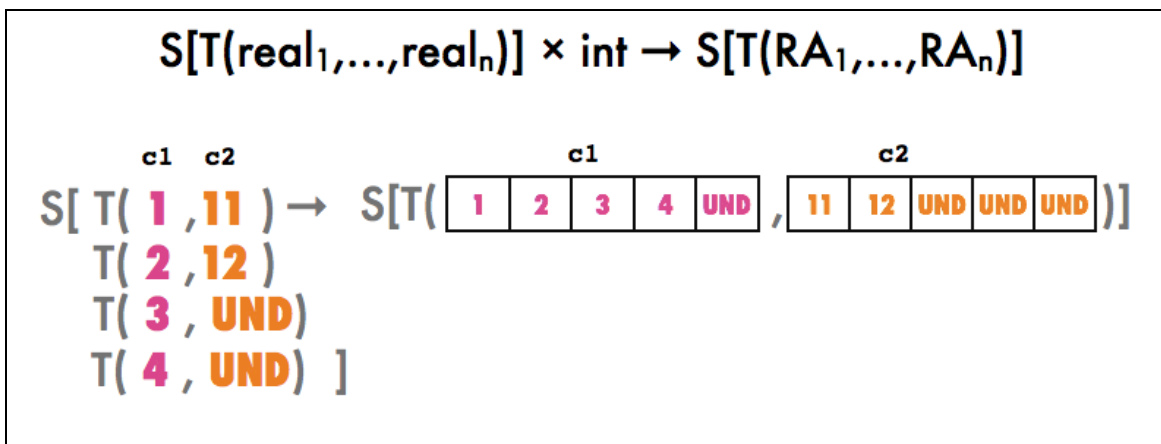


Figura 21 - Funcionamento do operador `uncolumn`

A Figura 21 demonstra o funcionamento deste operador. Exemplos de utilização podem ser vistos na Figura 22, abaixo. Devido ao uso do conceito de *streams*, bastamos acrescentar o operador `uncolumn` após o `column`. Podemos visualizar diferentes resultados alterando-se apenas o parâmetro de tamanho.

```

File Edit View Search Terminal Help
Secondo => query ReIA feed column uncolumn[3] consume;

c1:[ 1 ; 2 ; 3 ]
c2:[ 11 ; 12 ; UNDEFINED ]

c1:[ 4 ; 5 ; 6 ]
c2:[ UNDEFINED ; 21 ; 22 ]

c1:[ 7 ; 8 ; 9 ]
c2:[ 23 ; 31 ; 32 ]

c1:[ 10 ; 11 ; 12 ]
c2:[ 33 ; 34 ; 35 ]

Secondo => query ReIA feed column uncolumn[5] consume;

c1:[ 1 ; 2 ; 3 ; 4 ; 5 ]
c2:[ 11 ; 12 ; UNDEFINED ; UNDEFINED ; 21 ]

c1:[ 6 ; 7 ; 8 ; 9 ; 10 ]
c2:[ 22 ; 23 ; 31 ; 32 ; 33 ]

c1:[ 11 ; 12 ; UNDEFINED ; UNDEFINED ; UNDEFINED ]
c2:[ 34 ; 35 ; UNDEFINED ; UNDEFINED ; UNDEFINED ]

Secondo => query ReIA feed column uncolumn[6] consume;

c1:[ 1 ; 2 ; ..(+2).. ; 5 ; 6 ]
c2:[ 11 ; 12 ; ..(+2).. ; 21 ; 22 ]

c1:[ 7 ; 8 ; ..(+2).. ; 11 ; 12 ]
c2:[ 23 ; 31 ; ..(+2).. ; 34 ; 35 ]

Secondo =>

```

Figura 22 – Uso do operador uncolumn

Neste exemplo, apresentamos operações de uncolumn com tamanhos de 3, 5 e 6 elementos. No primeiro e último exemplo, podemos ver que não foi necessária a inclusão de elementos indefinidos. Os elementos indefinidos existentes são frutos da operação anterior (column). Já no segundo exemplo, podemos ver a inclusão de três elementos indefinidos ao fim de cada RealArray.

Este operador também mantém o nome dos atributos recebidos nas tuplas anteriores.

Os operadores apresentados anteriormente, `column` e `uncolumn`, demonstram um pouco da prática do processo de aquisição de dados. Na prática, um conjunto de sinais é considerado como uma tabela em que cada coluna representa um sensor e cada linha representa uma amostra. Cada coluna de `RealArray`, pode ser vista como uma coluna de números reais. Estes operadores permitem facilitar a consulta e armazenamento destas tabelas, guardando os valores em “blocos”, separando-os e concatenando-os da forma que se quiser.

6.2.3 Operador transpose

Em experimentos com muitos sensores, serão criadas várias colunas, uma para cada sensor. Uma operação que calcule a média destes sensores deverá ser escrita, explicitamente, na forma de uma expressão como :

$$\text{media} = (s1 + s2 + \dots + sn-1 + sn) / n;$$

Este tipo de expressão deverá ser reescrita para cada experimento, o quê é uma tarefa cansativa e que diminui a praticidade de uso de SGBDs no processamento de sinais. Uma das soluções propostas para este problema é o uso do operador `transpose`. Este operador recebe este nome pois simula “transpor” a tabela. A cardinalidade da tabela não é alterada, porém todas as colunas são agregadas, “horizontalmente”, formando apenas um atributo do tipo `RealArray` – que recebe o nome fixo de “`elem`”.

Este atributo pode ser trabalhado com muito mais facilidade através do uso dos operadores horizontais. Por exemplo, podemos reescrever a operação acima como:

$$\text{media} = \text{elem Havg};$$

De mais fácil leitura e reaproveitamento.

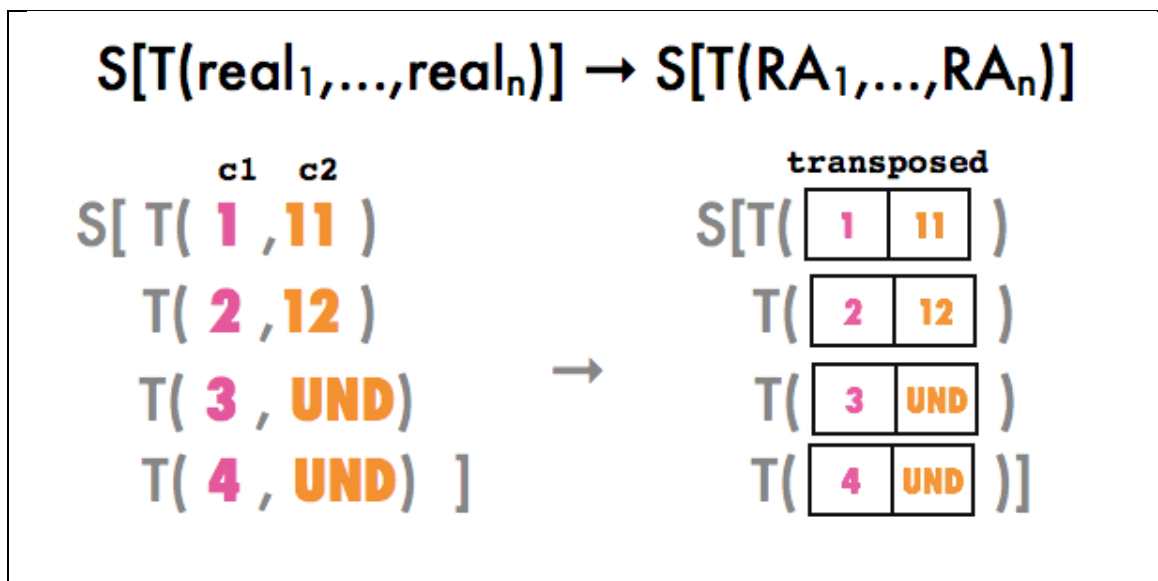
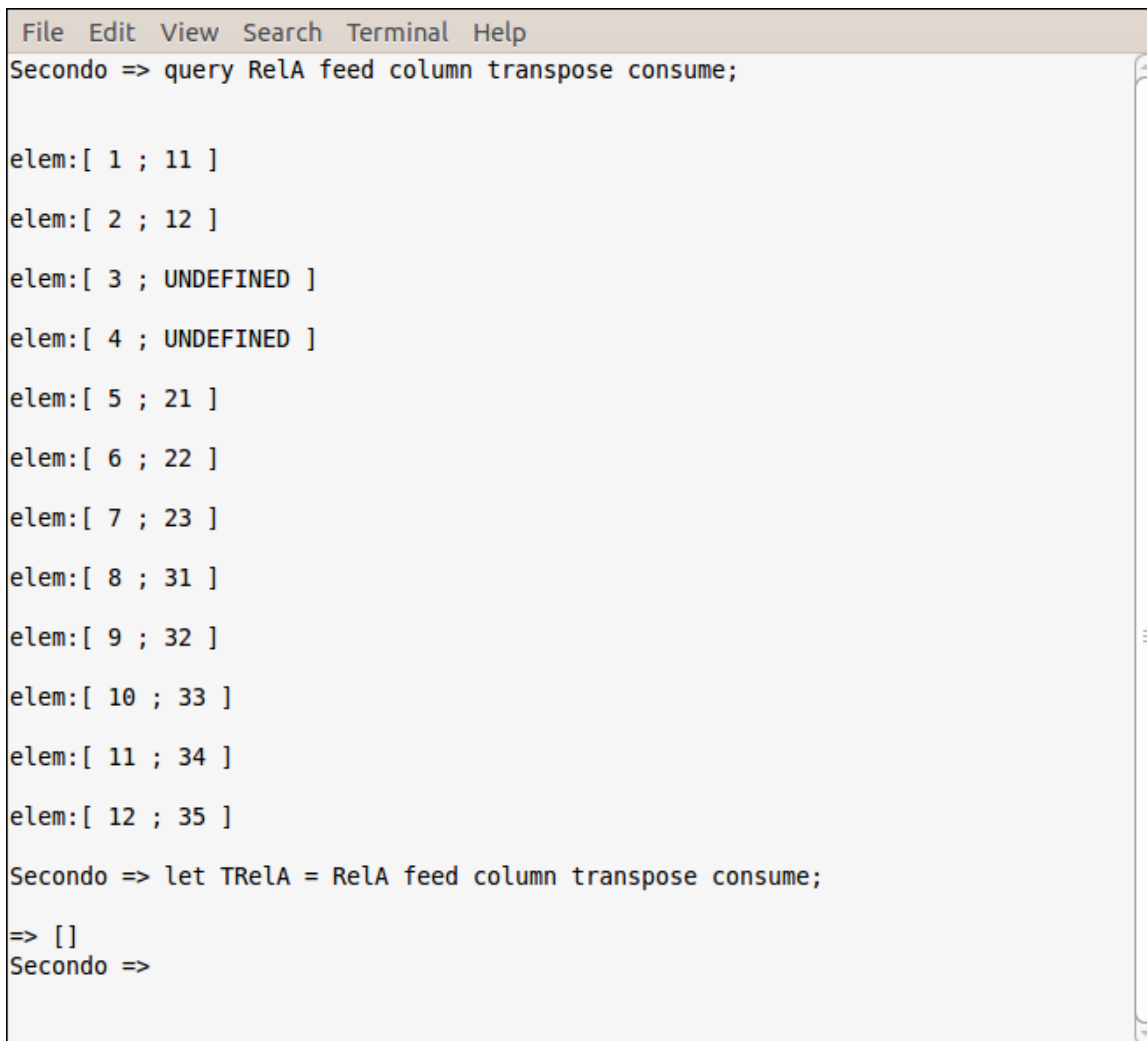


Figura 23 - Funcionamento do operador `transpose`

O funcionamento do operador `transpose` pode ser visto na Figura 23 e um exemplo prático sobre a tabela `RelA` (ver Figura 15), pode ser visto na Figura 24. Observando-se a última figura, vemos que ele também se utiliza do conceito de *stream*. No segundo comando é criada uma tabela `TRelA`, como uma forma de se armazenar o resultado da transposição para uso em exemplos posteriores.



```

File Edit View Search Terminal Help
Secondo => query ReIA feed column transpose consume;

elem:[ 1 ; 11 ]
elem:[ 2 ; 12 ]
elem:[ 3 ; UNDEFINED ]
elem:[ 4 ; UNDEFINED ]
elem:[ 5 ; 21 ]
elem:[ 6 ; 22 ]
elem:[ 7 ; 23 ]
elem:[ 8 ; 31 ]
elem:[ 9 ; 32 ]
elem:[ 10 ; 33 ]
elem:[ 11 ; 34 ]
elem:[ 12 ; 35 ]

Secondo => let TReIA = ReIA feed column transpose consume;

=> []
Secondo =>

```

Figura 24 – Uso do operador transpose

6.2.4 Operador untranspose

De modo a funcionar de forma inversa ao comando `transpose`, foi criado também o comando `untranspose`. Assim como o operador `uncolumn`, inverso do `column`, o operador `untranspose` requer parâmetros extras para a realização da operação. No caso do operador `uncolumn`, este parâmetro era um número inteiro, que especificava como agrupar as diversas linhas de números reais. Porém, o operador `untranspose` recebe como entrada um `RealArray`¹⁶ que deve ser transformado em várias colunas.

¹⁶ Existindo mais de um atributo `RealArray`, estes são concatenados para a operação.

O parâmetro requerido por este operador é, portanto, uma lista com os nomes das colunas que serão criadas¹⁷. O funcionamento deste operador pode ser visto na Figura 25.

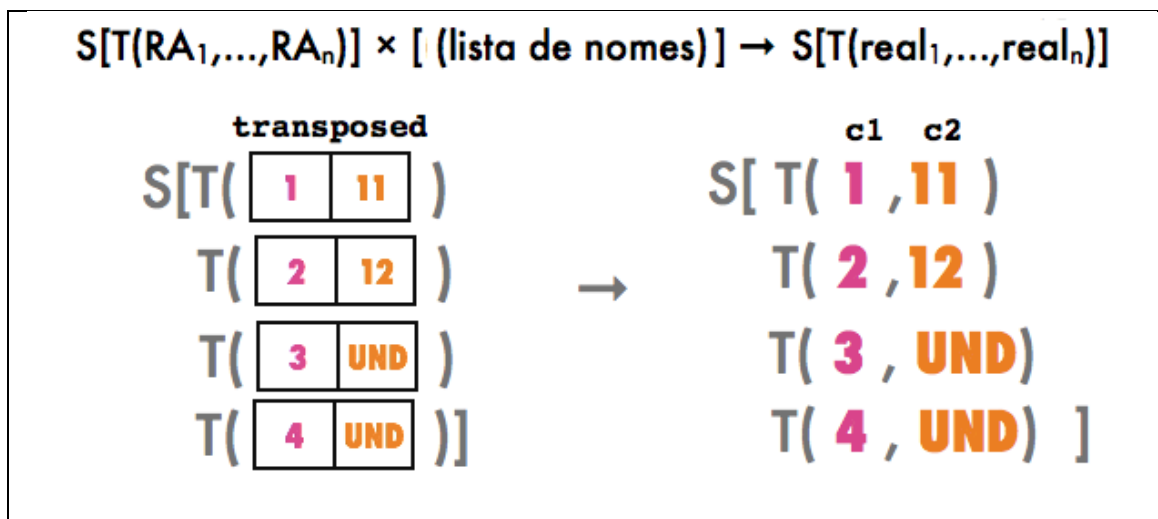


Figura 25 - Funcionamento do operador untranspose

Para os exemplos de utilização do operador, a tabela `TReLA`, criada nos exemplos anteriores foi usada de modo a “limpar” a sintaxe dos comandos.

Na Figura 26, é mostrado uma utilização “tradicional” do operador. A tabela `TReLA` foi criada a partir da transposição de dois atributos, `c1` e `c2`, do tipo `real`. O comando:

```
query TReLA feed untranspose[c1,c2] consume;
```

Retorna esta tabela a seu formato de colunas original.

¹⁷ Foi imaginada uma sintaxe mais simples que possibilitaria ao operador receber apenas um parâmetro, um número inteiro, representando o número de colunas a serem re-transpostas. Porém, devido a questões de nível sintático do `SECONDO` (segunda-órden), isto não foi possível. Pois o parâmetro recebido indicava somente que era um tipo `int`, e não seu valor. Esta informação é necessária para se determinar o schema da tabela resultante.

The screenshot shows the Secondo-GUI (RelationsViewer) interface. The command window on the left contains the following text:

```

Sec>
Sec>
Sec>
Sec>
Sec>
Sec>query TReLA feed untranspose[c1,c2] consume;
query TReLA feed untranspose[c1,c2]
consume;...successful
see result in object list
Sec>

```

The right pane shows a list of queries with buttons for 'show', 'hide', 'rem...', 'clear', 'save', 'load', 'store', and 'ren...'. The 'load' button is highlighted. Below the list, a table displays the result of the query:

query TReLA feed untranspose[c1,c2] consume;	
c1	c2
1.0	11.0
2.0	12.0
3.0	undef
4.0	undef
5.0	21.0
6.0	22.0
7.0	23.0
8.0	31.0
9.0	32.0
10.0	33.0
11.0	34.0
12.0	35.0

Figura 26 – Uso do operador untranspose

Outros usos mais “criativos” também podem ser imaginados. Na Figura 27, é mostrado um exemplo cujo número de colunas a serem criadas é maior do que o número de colunas original. Pode-se notar que a coluna nova (c3) possui apenas elementos indefinidos.

Na Figura 28, pode ser visto um exemplo, cuja tabela é composta por `RealArrays` que não foram transpostos. O comando separou os elementos em colunas, concatenando os diversos `RealArrays` como se fossem apenas um.

Secondo-GUI (RelationsViewer)

Program Server Optimizer Command Help Viewers

```

Sec>
Sec>query TReLA feed untranspose[c1,c2] consume;
query TReLA feed untranspose[c1,c2]
consume;...successful
see result in object list
Sec>query TReLA feed untranspose[c1,c2,c3] consume;
query TReLA feed untranspose[c1,c2,c3]
consume;...successful
see result in object list
Sec>

```

query TReLA feed untranspose[c1,c2,c3] consume;

c1	c2	c3
1.0	11.0	undef
2.0	12.0	undef
3.0	undef	undef
4.0	undef	undef
5.0	21.0	undef
6.0	22.0	undef
7.0	23.0	undef
8.0	31.0	undef
9.0	32.0	undef
10.0	33.0	undef
11.0	34.0	undef
12.0	35.0	undef

Figura 27 – Uso do operador untranspose com maior número de colunas

Secondo-GUI (RelationsViewer)

Program Server Optimizer Command Help Viewers

```

query ReLA feed column uncolumn[2] untranspose [c1,c2,c3,c4] consume;

```

c1	c2	c3	c4
1.0	2.0	11.0	12.0
3.0	4.0	undef	undef
5.0	6.0	21.0	22.0
7.0	8.0	23.0	31.0
9.0	10.0	32.0	33.0
11.0	12.0	34.0	35.0

Figura 28 – Uso do operador untranspose em tabelas não-transpostas

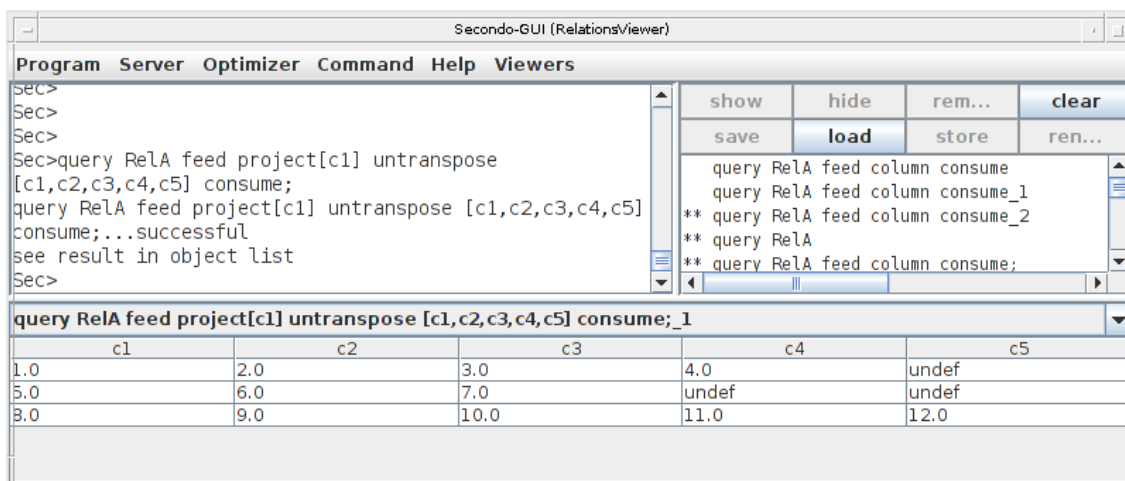


Figura 29 – Criação da tabela RelAT com comando untranspose

Por fim, na Figura 29, é mostrado o comando que criou a tabela RelAT, utilizada nos exemplos da seção 6.1.5 :

```
query RelA feed project[c1] untranspose [c1,c2,c3,c4,c5];
```

O que nos permite notar a flexibilidade e o potencial de manipulação de dados que estes conjuntos de operadores (column, uncolumn, transpose e untranspose) proporcionam.

6.2.5 Operador window

Como já foi apresentado na seção 2.2.2, o conceito de “janela móvel” é um dos conceitos fundamentais para o processamento de sinais no tempo. Ele é necessário para se executar filtragens, transformadas e outras operações com o intuito de “limpar” o sinal.

A álgebra proposta apresenta um operador trans-relacional para lidar com este conceito. A falta deste operador em SGBDs convencionais implica no uso de consultas recursivas (quando possíveis) para realizar esta operação. Estima-se que a complexidade desta operação, se realizada nesses sistemas, seria de $O(nk)$, sendo n o número de elementos e k o tamanho da janela móvel (vide Figura 1). Enquanto que

a implementação deste operador em um SGBST possibilitaria uma redução na complexidade para $O(n)$, pois a janela seria persistida, internamente, durante a consulta.

Esta janela móvel pode ser definida como sendo **aberta** ou **fechada** em suas extremidades. Ter uma extremidade aberta significa incluir porções do sinal que não existem, isto é, dados faltantes, mas que servirão como base de cálculo para uma operação subsequente. Isto pode ser necessário para garantir que todo valor existente possua uma janela correspondente. De uma forma geral, se considerarmos apenas janelas fechadas, para n valores e uma janela de tamanho k , apenas $n-k$ valores terão uma janela correspondente.

Considerando como exemplo uma operação de média móvel, em que o valor de um elemento é calculado como a média sobre uma janela móvel, podemos inferir dois tipos de cálculo: um primeiro, no qual os valores são calculados a partir do histórico de um valor atual e um segundo, em que os valores são calculados a partir da progressão do valor atual. Ou seja, uma média móvel é calculada sobre o passado e a outra sobre o futuro. Dada uma série temporal, a qual pretendemos submeter a esta operação, precisamos escolher qual tipo de operação será utilizada.

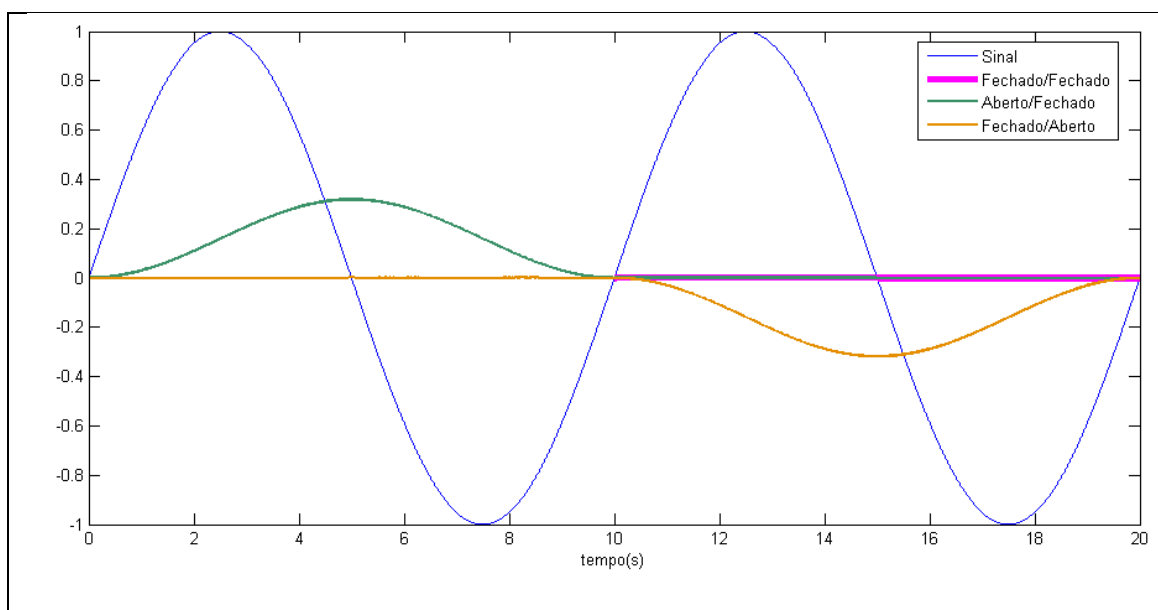


Figura 30 - Exemplo de uma média móvel sobre uma senóide

A Figura 30 mostra um exemplo de uma operação de média móvel sobre um sinal no tempo. Trata-se de uma senóide, da qual foram amostrados 20 segundos. Podemos identificar quatro tipos diferentes de média sobre este sinal:

- em azul, o sinal original, que varia de -1 a +1. A média de todos os valores é zero;
- em magenta, uma média móvel feita sobre os 10 segundos anteriores do sinal. A média começa no segundo 10, momento em que a janela começa a existir;
- em verde, uma média móvel feita sobre os 10 segundos anteriores “Abertos” do sinal. A média começa no momento 0, mas o seu valor é o próprio valor do sinal, pois não existem mais elementos para serem calculados. Ao se aproximar do segundo 10, a média passa então a convergir para seu valor correto;
- em laranja, uma média móvel feita sobre os 10 segundos posteriores “Abertos” do sinal. A partir do momento 10, seu valor passa a divergir do esperado.

Não é usual se fazer operações utilizando ambas as extremidades abertas. O resultado disto seria uma massa de resultados superior ao número de amostras existentes. A álgebra, porém permite este tipo de operação.

A Figura 31, mostra os conceitos de janelas abertas e fechadas do ponto de vista do resultado produzido. Existe um período “completo” no qual todos os resultados são “coerentes” e dois períodos correspondentes ao início e fim da série, cujos resultados apresentam valores indefinidos.

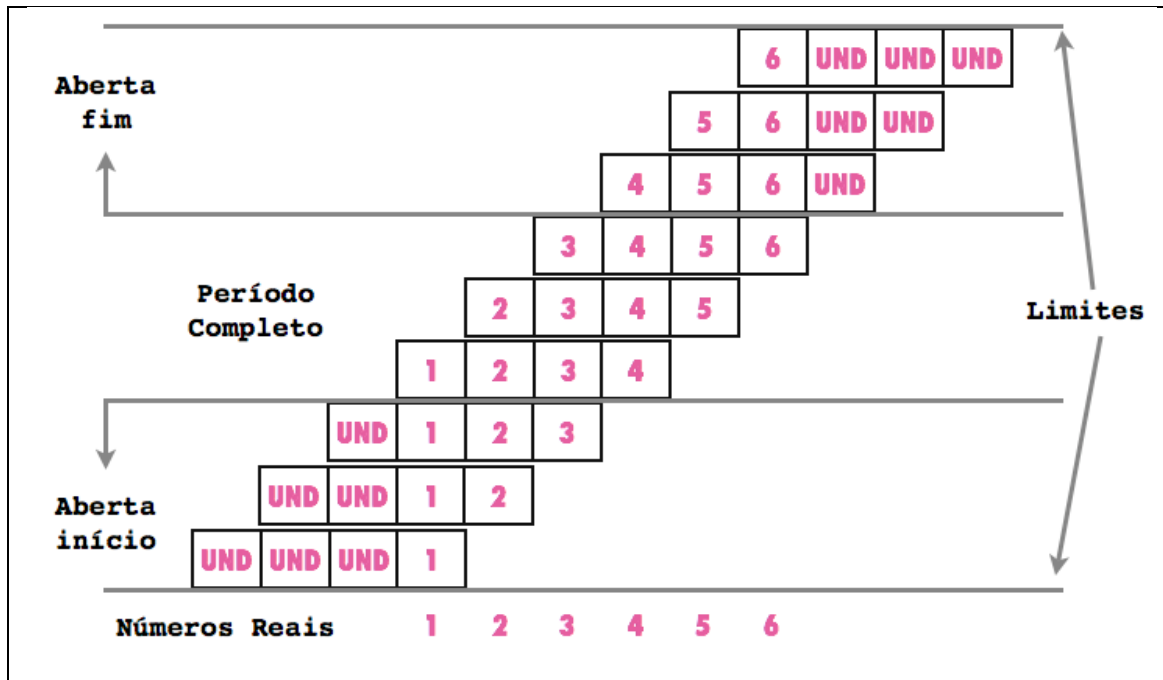


Figura 31 - Conceito de Janela Móvel

A sintaxe do operador demanda, portanto, 3 parâmetros: o tamanho da janela desejada, um número inteiro; um valor booleano indicando a abertura (falso) ou fechamento (verdadeiro) do início do período e um valor booleano, similar, indicando a abertura/fechamento do fim do período. O funcionamento deste operador e sua sintaxe podem ser vistos na Figura 32.

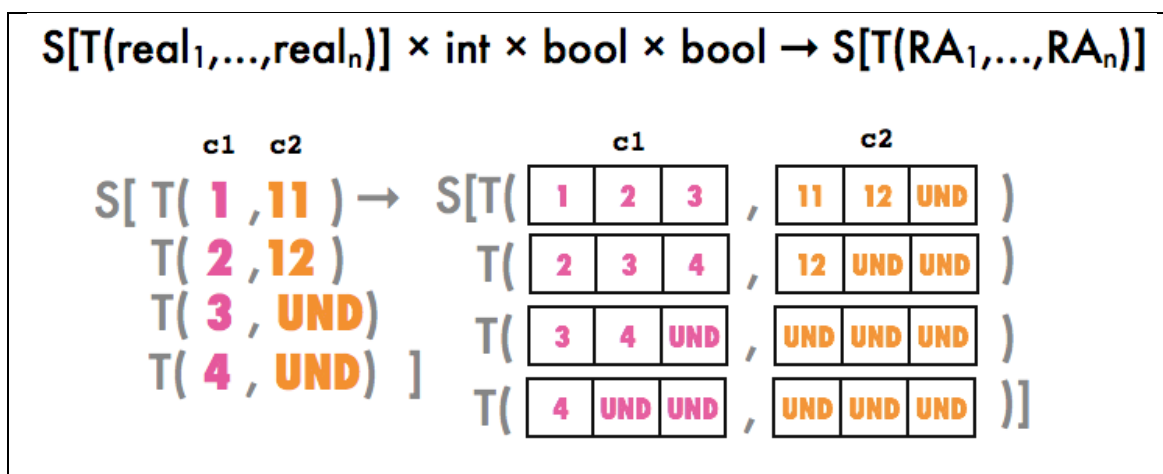


Figura 32 - Funcionamento do operador window

As figuras Figura 33 a Figura 36 exemplificam consultas feitas a tabela RelA, com janelas de mesmo tamanho, alterando-se apenas a abertura/fechamento das mesmas.

A sintaxe basica para todas as consultas é :

```
query RelA feed column window [5, TRUE, TRUE] consume;
```

Onde a sequencia TRUE, TRUE é substituída por TRUE, FALSE ; FALSE, TRUE ou FALSE, FALSE, respectivamente.

The screenshot shows the 'Secondo-GUI (RelationsViewer)' interface. The command window contains the following text:

```
Sec>
Sec>
Sec>
Sec>query RelA feed column window [5,TRUE,TRUE] consume;
query RelA feed column window [5,TRUE,TRUE]
consume;...successful
see result in object list
Sec>
```

The output window displays the following table:

c1	c2
(1.0 2.0 3.0 4.0 5.0)	(11.0 12.0 undef undef 21.0)
(2.0 3.0 4.0 5.0 6.0)	(12.0 undef undef 21.0 22.0)
(3.0 4.0 5.0 6.0 7.0)	(undef undef 21.0 22.0 23.0)
(4.0 5.0 6.0 7.0 8.0)	(undef 21.0 22.0 23.0 31.0)
(5.0 6.0 7.0 8.0 9.0)	(21.0 22.0 23.0 31.0 32.0)
(6.0 7.0 8.0 9.0 10.0)	(22.0 23.0 31.0 32.0 33.0)
(7.0 8.0 9.0 10.0 11.0)	(23.0 31.0 32.0 33.0 34.0)
(8.0 9.0 10.0 11.0 12.0)	(31.0 32.0 33.0 34.0 35.0)

Figura 33 – Uso do operador window com janela Fechada/Fechada

The screenshot shows the 'Secondo-GUI (RelationsViewer)' interface. The command window contains the following text:

```
query RelA feed column window [5,TRUE,FALSE] consume;
```

The output window displays the following table:

c1	c2
(1.0 2.0 3.0 4.0 5.0)	(11.0 12.0 undef undef 21.0)
(2.0 3.0 4.0 5.0 6.0)	(12.0 undef undef 21.0 22.0)
(3.0 4.0 5.0 6.0 7.0)	(undef undef 21.0 22.0 23.0)
(4.0 5.0 6.0 7.0 8.0)	(undef 21.0 22.0 23.0 31.0)
(5.0 6.0 7.0 8.0 9.0)	(21.0 22.0 23.0 31.0 32.0)
(6.0 7.0 8.0 9.0 10.0)	(22.0 23.0 31.0 32.0 33.0)
(7.0 8.0 9.0 10.0 11.0)	(23.0 31.0 32.0 33.0 34.0)
(8.0 9.0 10.0 11.0 12.0)	(31.0 32.0 33.0 34.0 35.0)
(9.0 10.0 11.0 12.0 undef)	(32.0 33.0 34.0 35.0 undef)
(10.0 11.0 12.0 undef undef)	(33.0 34.0 35.0 undef undef)
(11.0 12.0 undef undef undef)	(34.0 35.0 undef undef undef)
(12.0 undef undef undef undef)	(35.0 undef undef undef undef)

Figura 34 – Uso do operador window com janela Fechada/Aberta

query RelA feed column window [5,FALSE,TRUE] consume;	
c1	c2
(undef undef undef undef 1.0)	(undef undef undef undef 11.0)
(undef undef undef 1.0 2.0)	(undef undef undef 11.0 12.0)
(undef undef 1.0 2.0 3.0)	(undef undef 11.0 12.0 undef)
(undef 1.0 2.0 3.0 4.0)	(undef 11.0 12.0 undef undef)
(1.0 2.0 3.0 4.0 5.0)	(11.0 12.0 undef undef 21.0)
(2.0 3.0 4.0 5.0 6.0)	(12.0 undef undef 21.0 22.0)
(3.0 4.0 5.0 6.0 7.0)	(undef undef 21.0 22.0 23.0)
(4.0 5.0 6.0 7.0 8.0)	(undef 21.0 22.0 23.0 31.0)
(5.0 6.0 7.0 8.0 9.0)	(21.0 22.0 23.0 31.0 32.0)
(6.0 7.0 8.0 9.0 10.0)	(22.0 23.0 31.0 32.0 33.0)
(7.0 8.0 9.0 10.0 11.0)	(23.0 31.0 32.0 33.0 34.0)
(8.0 9.0 10.0 11.0 12.0)	(31.0 32.0 33.0 34.0 35.0)

Figura 35 – Uso do operador window com janela Aberta/Fechada

query RelA feed column window [5,FALSE,FALSE] consume;	
c1	c2
(undef undef undef undef 1.0)	(undef undef undef undef 11.0)
(undef undef undef 1.0 2.0)	(undef undef undef 11.0 12.0)
(undef undef 1.0 2.0 3.0)	(undef undef 11.0 12.0 undef)
(undef 1.0 2.0 3.0 4.0)	(undef 11.0 12.0 undef undef)
(1.0 2.0 3.0 4.0 5.0)	(11.0 12.0 undef undef 21.0)
(2.0 3.0 4.0 5.0 6.0)	(12.0 undef undef 21.0 22.0)
(3.0 4.0 5.0 6.0 7.0)	(undef undef 21.0 22.0 23.0)
(4.0 5.0 6.0 7.0 8.0)	(undef 21.0 22.0 23.0 31.0)
(5.0 6.0 7.0 8.0 9.0)	(21.0 22.0 23.0 31.0 32.0)
(6.0 7.0 8.0 9.0 10.0)	(22.0 23.0 31.0 32.0 33.0)
(7.0 8.0 9.0 10.0 11.0)	(23.0 31.0 32.0 33.0 34.0)
(8.0 9.0 10.0 11.0 12.0)	(31.0 32.0 33.0 34.0 35.0)
(9.0 10.0 11.0 12.0 undef)	(32.0 33.0 34.0 35.0 undef)
(10.0 11.0 12.0 undef undef)	(33.0 34.0 35.0 undef undef)
(11.0 12.0 undef undef undef)	(34.0 35.0 undef undef undef)
(12.0 undef undef undef undef)	(35.0 undef undef undef undef)

Figura 36 – Uso do operador window com janela Aberta/Aberta

6.3 Características Algébricas dos Operadores Trans-Relacionais

Uma das possibilidades permitidas pelo uso de operadores Trans-Relacionais em uma extensão de um SGBD é a otimização da consulta através de planos devido a suas características algébricas. Isto não seria possível se eles fossem apenas implementados como *Storage Procedures* em um SGBD convencional, pois estas características não poderiam ser exploradas pelo otimizador de consultas.

6.3.1 Proposta de Validação Algébrica

Ao mencionarmos uma validação algébrica de um conjunto de operadores, pressupomos que características básicas desta álgebra serão demonstradas, isto é, questões relativas à Teoria dos Conjuntos e demonstrações que a álgebra com estes operadores é um corpo ou um anel. Uma vez que o trabalho foi especificado, desenvolvido e implementado através de um protótipo na plataforma SECONDO, que é reconhecida pela comunidade acadêmica como uma forma de apresentação e validação de extensão da álgebra relacional, o formalismo da demonstração matemática foi de certa forma “relaxado” de modo a concentrar-se, unicamente, nas questões que podem ser de valia para as propostas de otimização posteriores a este trabalho.

6.3.2 Conceito de Para-Extensionalidade

Para seguirmos com as validações algébricas necessárias, é necessário definir o conceito de **para-extensionalidade** que será utilizado nas definições.

Definição 1: Uma tabela é k -para-extensa, quando todos seus atributos são ou do tipo `RealArray`, ou do tipo `real`, e possuem o mesmo tamanho, igual a k (k pertencente aos inteiros positivos).

No caso de uma tabela que possua apenas atributos do tipo `real`, a mesma pode ser dita 1-para-extensa. As tabelas mostrada nos exemplos do operador `window`, Figura 33 a Figura 36, são todas 5-para-extensas.

De modo que podemos propor o seguinte comportamento operacional:

Comportamento 1: um operador Trans-Relacional pertencente ao conjunto de operadores { `column`, `uncolumn`, `transpose`, `untranspose` } será notado como TR_k e transformará uma tabela A em uma tabela k -para-extensa.

Detalhamento: Pelo funcionamento dos operadores, podemos ver que todos os resultados produzidos por eles são k -para-extensos, por força bruta:

- `column` : produz colunas de tipo `real` – 1-para-extenso por definição;
- `uncolumn` : produz colunas de tipo `RealArray` de tamanho k , a partir de uma tabela 1-para-extensa – São incluídos elementos indefinidos ao final da última linha, caso n não seja divisível por k - k -para-extenso por definição;
- `transpose` : produz 1 coluna de tipo `RealArray` a partir de uma tabela 1-para-extensa, k é dado pelo número de colunas - k -para-extenso por definição;
- `untranspose` : produz k colunas do tipo `real`, k é parametrizado, elementos indefinidos são incluídos caso a linha não possua elementos suficientes; elementos adicionais são passadas para linha seguinte - 1-para-extenso por definição;

Notação: nota-se um operador trans-relacional que produz uma tabela k -para-extensa por TR_k . Pode-se também utilizar a notação TR_{jk} , quando o operador recebe uma tabela j -para-extensa e a converte em uma tabela k -para-extensa.

Comportamento 2: Seja A uma tabela k -para-extensa, com n linhas, ao se convertê-la para uma tabela A' j -para-extensa, através de um operador `column` TR_1 , ou `uncolumn` TR_j seu número de linhas será $\lceil kn / j \rceil$.

Detalhamento:

- Para $j = 1, k \geq 1$; cada linha de A se transformará em k linhas de A' , ou seja: $\lceil kn / 1 \rceil = kn$;
- Para $j \geq 1, k = 1$; cada linha L de A será agrupada em uma linha de A' , obedecendo o tamanho de j , para L menor ou igual a $\lfloor kn / j \rfloor$. Quando L for maior do que este limite, uma nova linha de A' será criada com as linhas restantes e elementos indefinidos serão acrescentados, logo o valor $\lceil kn / j \rceil$ será atingido.

Comportamento 3: Seja A uma tabela k -para-extensa, com n linhas, e c colunas, ao se convertê-la para uma tabela A' j -para-extensa, através de um operador `transpose` TR_j , ou `untranspose` TR_1 seu número de linhas será $\lceil kcn / \min(j, kc) \rceil$.

Detalhamento:

- Para o operador `transpose`, temos $j = c$ e $k = 1$, por definição, portanto, $\min(j, kc) = \min(c, c)$, $\lceil kcn / j \rceil = \lceil n \rceil = n$;
- Para o operador `untranspose`, para $c = 1$, se j for menor que k , linhas extras serão geradas, seguindo-se a mesma lógica do Comportamento 2, e a última linha receberá valores indefinidos; Caso contrário (j maior do que k): $\min(j, k) = k$, $\lceil kn / k \rceil = n$;

- Ainda no caso do operador *untranspose*, para $c > 1$, as colunas são concatenadas para a operação (kc) e podemos seguir o raciocínio anterior considerando-se:

$$k' = kc.$$

Os comportamentos 2 e 3 reforçam o comportamento 1 que os operadores {*column*, *uncolumn*, *transpose* e *untranspose* } produzem de fato resultados *k*-para-extensos e que o número de linhas das tabelas resultantes podem ser calculadas.

Notação: Nota-se um operador TR_k^{-1} , quando este é a representação da operação inversa de um operador TR_k . São considerados *column* / *uncolumn* e *transpose* / *untranspose* como sendo inversos, por definição.

Comportamento 4:

Seja *A* uma tabela *k*-para-extensa, com *n* linhas. Seja *A'* uma tabela resultante de uma operação $A' = TR_{jk}^{-1}(TR_{kj}(A))$. *A'* será igual a *A*.

Detalhamento:

Se *A* é *k*-para-extensa, com *n* linhas após a aplicação de uma operação TR_{kj} , obteremos uma tabela *A''* com número de linhas *n'*, que será calculado ou por $\lceil kn / j \rceil$ conforme o Comportamento 2, ou por $\lceil kcn / \min(j, kc) \rceil$ conforme o Comportamento 3. Ao fazermos a operação inversa TR_{jk}^{-1} , sobre *A''*, obteremos um número de linhas $\lceil jn' / k \rceil$ ou $\lceil jc'n' / \min(jc', kc) \rceil$.

No primeiro caso (*column* / *uncolumn*), ao multiplicarmos *n'* por *j* obteremos *kn* que posteriormente será dividido por *k* resultando em *n*.

No segundo caso teremos, para um operador *transpose* como TR_{kj} , $c'=1$, $n'=n$, para $k=1$ e $j>k$ e $c=j$ (o operador *transpose* trabalha sobre tabelas 1-para-extendidas), ao se fazer o *untranspose* (TR_{jk}^{-1}) teremos:

$$\lceil jc' \lceil kcn / \min(j, kc) \rceil / \min(jc', kc) \rceil = \lceil j \lceil jn / j \rceil / j \rceil = n.$$

Alterando-se a ordem dos operadores *transpose* e *untranspose* obteremos o mesmo resultado.

Ou seja, como o número de linhas de A' continua igual ao de A , e A era, por definição k -para-extendida, não foram incluídos elementos indefinidos nestas operações e logo, $A' = A$.

Lema:

Se A for uma tabela não-para-extendida, e forem aplicadas as mesmas operações inversas do Comportamento 4, verificaremos que A' será k -para-extendida, apesar de não ser necessariamente igual a A . Porém, se aplicarmos as operações inversas a A' , pelo fato dela ser k -para-extendida, a tabela A'' resultante destas operações será igual a A' .

Estas definições e comportamentos de equivalência mostram que podem ser feitas transformações algébricas gerando expressões equivalentes, porém que podem ter um plano físico de execução da consulta mais eficiente. Estas transformações formam a base de otimizações automáticas de consultas SQL. Heurísticas podem ser definidas e aplicadas, por exemplo, se forem encontradas situações em que se gere um resultado a partir de operações inversíveis. Estes resultados intermediários podem também ser armazenados para otimizar novas transações.

6.4 Semântica Temporal

6.4.1 Adequação à Semântica Temporal

De modo a adequar a álgebra para o modelo temporal, faz-se necessário implementar um tipo que atenda às necessidades apresentadas na seção 5.1. O sistema SECONDO já apresenta dois tipos destinados a representar tempo, são eles : `instant` e `duration`. O primeiro serve para armazenar um valor de tempo, apresentado no formato “AAAA-MM-DD hh:mm:ss.ms”¹⁸, o segundo armazena uma duração (período de tempo), apresentado no formato “dias, milissegundos”. Internamente, ambos são armazenados na forma de números de ponto flutuante.

6.4.2 Implementação do tipo `TimeSpan`

Como também foi discutido na seção 5.1, uma amostra de um sinal temporal não tem sentido sem que sejam especificados tanto o seu instante, como também seu período. De forma que foi necessário implementar um novo tipo `TimeSpan` que atendesse a esta demanda.

A apresentação de valores deste tipo segue o formato do tipo `instant`, sendo um para o início e outro para o fim do período, e, analogamente, sua representação interna é a de números reais.

Para permitir que este tipo fosse também indexável por uma estrutura *R-Tree*, já implementada pelo SECONDO, foi necessário transformá-lo em um objeto com características bidimensionais, como um retângulo. Portanto, apesar de armazenar apenas a informação de um segmento de reta, o tipo possui dois atributos internos,

¹⁸ Ano, Mês, dia, hora, minuto, segundo e frações

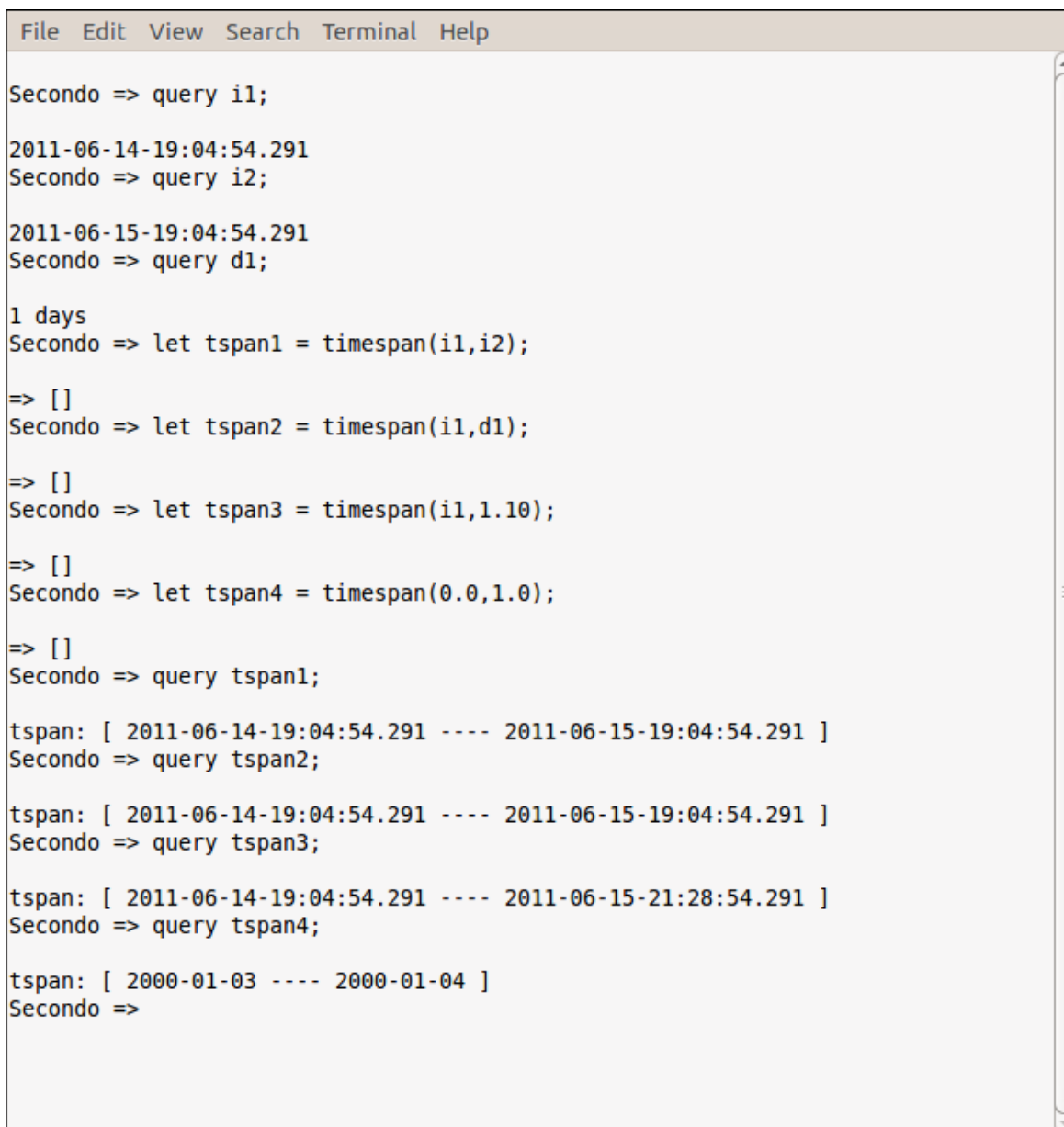
constantes, representando uma altura, para que o segmento seja tratado como um retângulo pela estrutura *R-Tree*.

A Figura 37 apresenta um exemplo de criação de objetos do tipo `TimeSpan`. São utilizados três objetos, `i1`, `i2` e `d1`, previamente definidos, do tipo `instant` e `duration`. Os três primeiros comandos mostram que `i2` representa a mesma hora do instante `i1` com um dia de diferença e o objeto `d1` representa, também, a duração de um dia.

A função de construção utilizada para os objetos do tipo `TimeSpan` foi sobrecarregada (*overloading*) para permitir quatro diferentes formas de utilização:

- utilizando-se dois objetos do tipo `instant`;
- um objeto `instant` e um `duration`;
- um objeto `instant` e um `real`, indicando a duração em dias e frações;
- dois objetos `real` indicando, o primeiro, um instante em sua representação interna e, o segundo, uma duração em dias e frações;

Como era esperado, os objetos `tspan1` e `tspan2` são iguais. Ambos representam o período de um dia a partir do instante `i1`. O objeto `tspan3` foi criado com uma duração de 1.10 dias, isto é 1 dia, 2 horas e 24 minutos (1 décimo de dia). Já o objeto `tspan4`, que também possui a duração de um dia, foi criado a partir do instante zero do qual o `SECONDO` calcula suas datas, 0h de 3 de janeiro de 2000.



```
File Edit View Search Terminal Help

Secondo => query i1;

2011-06-14-19:04:54.291
Secondo => query i2;

2011-06-15-19:04:54.291
Secondo => query d1;

1 days
Secondo => let tspan1 = timespan(i1,i2);

=> []
Secondo => let tspan2 = timespan(i1,d1);

=> []
Secondo => let tspan3 = timespan(i1,1.10);

=> []
Secondo => let tspan4 = timespan(0.0,1.0);

=> []
Secondo => query tspan1;

tspan: [ 2011-06-14-19:04:54.291 ---- 2011-06-15-19:04:54.291 ]
Secondo => query tspan2;

tspan: [ 2011-06-14-19:04:54.291 ---- 2011-06-15-19:04:54.291 ]
Secondo => query tspan3;

tspan: [ 2011-06-14-19:04:54.291 ---- 2011-06-15-21:28:54.291 ]
Secondo => query tspan4;

tspan: [ 2000-01-03 ---- 2000-01-04 ]
Secondo =>
```

Figura 37 - Criação de tipos TimeSpan

6.4.3 Operadores do tipo TimeSpan

A seção 5.1.3 apresenta uma lista de operadores para serem utilizados com séries temporais. O tipo `TimeSpan` não corresponde à série temporal em si mas dará o sentido semântico a mesma. De forma que apenas alguns dentre os listados se aplicam. Os mais importantes são : `intersects`, `inside`, `intersection` e `union`. Os dois primeiros da lista comparam objetos do tipo `TimeSpan` e apresentam um resultado booleano caso os intervalos possuam uma interseção ou se um está dentro do outro. Os dois últimos criam objetos novos contendo a interseção e a união dos operandos, respectivamente. No caso da união, é criado um intervalo que vai do “menor” instante ao “maior” instante dos dois operandos. A adjacência dos intervalos não é examinada. No caso da interseção, é possível a obtenção de um intervalo vazio (indefinido) e caso um intervalo esteja dentro do outro, o resultado será ele próprio.

A Figura 38, apresenta exemplos diversos exemplos destes operadores. Como pode ser visto, a sintaxe é muito simples e não requer explicações detalhadas sobre seu uso.

```

File Edit View Search Terminal Help
Secondo => let tspan5 = timespan(1.0,2.0);
=> []
Secondo => let tspan6 = timespan(1.5,2.5);
=> []
Secondo => query tspan6 intersects tspan5;
TRUE
Secondo => query intersection(tspan5,tspan6);
tspan: [ 2000-01-04-12:00 ---- 2000-01-05 ]
Secondo => query tspan6 inside tspan5
Secondo ->
FALSE
Secondo => query tspan1 inside tspan3;
TRUE
Secondo => query tspan5 union tspan6;
tspan: [ 2000-01-04 ---- 2000-01-05-12:00 ]
Secondo => query tspan4 union tspan6;
tspan: [ 2000-01-03 ---- 2000-01-05-12:00 ]
Secondo => query tspan1 = tspan2;
TRUE
Secondo => query tspan2 = tspan3;
FALSE

```

Figura 38 - Operadores de intervalo para o tipo TimeSpan

6.4.4 Indexação de TimeSpan com R-Trees

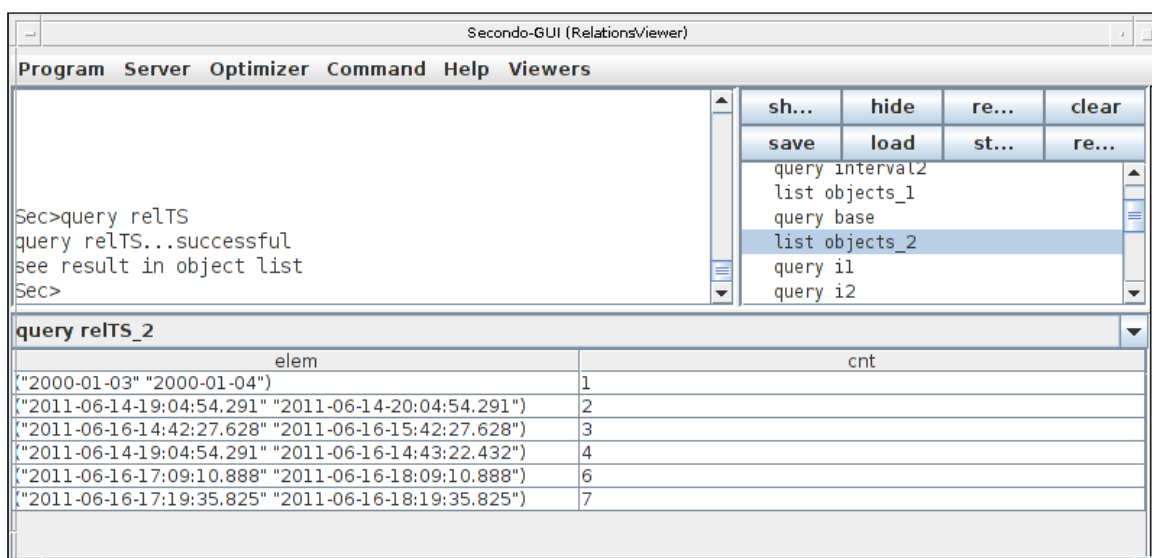
Segundo Ramakrishnan [27] (p. 688), *R-Trees* são utilizadas em Sistemas de Informação Geográficas (GIS) para a criação de índices multidimensionais. E, neste sentido, faz parte do pacote do sistema SECONDO. Porém, elas também podem ser utilizadas no caso unidimensional.

A utilização de índices para busca de elementos em relações contendo séries temporais não faz parte do foco principal deste trabalho. Entende-se que a existência de uma estrutura que facilite a busca de informações em uma base de séries temporais é uma benesse, mas não o objetivo principal. Caso existam outras

estruturas que se provem mais eficientes que a *R-Tree*, elas poderão vir a ser implementadas posteriormente.

A disponibilidade desta estrutura no sistema SECONDO foi um convite a sua utilização. O seu código não foi modificado e o seu funcionamento interno não será explicado.

A adequação do tipo `TimeSpan` para utilização com a *R-Tree*, foi feito, conforme mencionado anteriormente, transformando o tipo em um retângulo com altura constante, que represente um *kind* `SPATIAL2D`. Nos exemplos posteriores será mostrada a sintaxe de sua utilização. Para tal, foi criada uma tabela chamada `relTS`, que contém um atributo do tipo `TimeSpan` (`elem`) e um atributo inteiro (`cnt`).



The screenshot shows the Secondo-GUI interface. The main window title is "Secondo-GUI (RelationsViewer)". The menu bar includes "Program", "Server", "Optimizer", "Command", "Help", and "Viewers". The command window shows the following interaction:

```
Sec>query relTS
query relTS...successful
see result in object list
Sec>
```

The result is displayed in a table titled "query relTS_2". The table has two columns: "elem" and "cnt". The data rows are as follows:

elem	cnt
("2000-01-03" "2000-01-04")	1
("2011-06-14-19:04:54.291" "2011-06-14-20:04:54.291")	2
("2011-06-16-14:42:27.628" "2011-06-16-15:42:27.628")	3
("2011-06-14-19:04:54.291" "2011-06-16-14:43:22.432")	4
("2011-06-16-17:09:10.888" "2011-06-16-18:09:10.888")	6
("2011-06-16-17:19:35.825" "2011-06-16-18:19:35.825")	7

Figura 39 - Tabela relTS

A Figura 39 mostra esta tabela. Pode-se notar que os elementos não estão necessariamente em ordem. A Figura 40 apresenta a forma de criação e utilização da estrutura *R-Tree*.

```

File Edit View Search Terminal Help
Secondo => let rtree = relTS creatertree[elem];

=> []
Secondo => query rtree;

No specific display function defined. Generic function used.
Type:
(rtree
  (tuple
    (
      (elem tspan)
      (cnt int)))
  tspan
  FALSE)
Value:
("R-Tree statistics"
  ("Height" 0)
  ("# of (leaf) entries" 6)
  ("# of nodes" 1)
  ("Bounding Box"
    (0.0 4182.763609085649 0.0 1.0)))
Secondo =>
Secondo => query rtree relTS windowintersects[tspan1] consume;

elem:tspan: [ 2011-06-14-19:04:54.291 ---- 2011-06-14-20:04:54.291 ]
cnt:2

elem:tspan: [ 2011-06-14-19:04:54.291 ---- 2011-06-16-14:43:22.432 ]
cnt:4

Secondo => query rtree relTS windowintersects[tspan4] consume
Secondo ->

elem:tspan: [ 2000-01-03 ---- 2000-01-04 ]
cnt:1

Secondo =>

```

Figura 40 - Exemplos de utilização de uma RTree.

Na primeira linha:

```
let rtree = relTS creatertree[elem];
```

O objeto `rtree` é criado a partir da relação `relTS`, utilizando-se o atributo `elem`. Como o índice não possui uma forma de apresentação específica, ao se acessá-lo, por um comando `query`, o resultado é apenas uma informação genérica com alguns parâmetros, como por exemplo: altura, número de nós, *bounding box* etc.

A consulta através do índice é realizada com o operador `windowintersects` (da álgebra *R-Tree*) , que recebe como operandos, o índice, a relação e um objeto que conterá a chave a ser consultada.

O comando:

```
query rtree relTS windowintersects[tspan1] consume;
```

Procura todos os elementos de `relTS` cuja janela (atributo `elem`) possui interseção com o operando `tspan1`. O resultado é mostrado logo após o comando. Utilizando-se o objeto `tspan4` (criados nos exemplos anteriores) é mostrado um resultado diferente. O uso do comando `consume` é necessário pois o resultado do operador `windowintersects` é do tipo *stream* de tuplas, para que possa ser encadeado com outros operadores, e não uma relação.

6.5 Aplicação da Semântica Temporal

6.5.1 Sincronização

Como já foi mencionado anteriormente, a Semântica Temporal é obtida através da associação de um tipo `TimeSpan` com um tipo `RealArray` (ou `real`). De forma a não quebrar o paradigma relacional, esta associação será feita explicitamente¹⁹ através do uso de operadores que manipularão os atributos destes tipo. Uma vez associados, estes dois tipos podem ser vistos como uma Série Temporal em que a cada elemento do `RealArray`²⁰ corresponde uma amostra da série com o seu instante de medição e período.

Podemos então definir o termo “**sincronização**”. Duas séries temporais estarão “sincronizadas”, quando a sua **base de tempo** (tipo `TimeSpan`) e número de amostras (**cardinalidade** do `RealArray`²⁰) forem as mesmas. Como pode-se notar, esta definição é consistente com o apresentado na seção 5.1.2.

6.5.2 Operadores de Pré-Sincronização

De forma a sincronizar atributos que possuam cardinalidade distinta, foram desenvolvidos quatro operadores que alteram a cardinalidade de um atributo do tipo `RealArray` de acordo com o paradigmas da Semântica Temporal, ou seja a alteração do número de elementos se dá levando-se em conta que os valores foram aquisitados a partir da observação de um fenômeno realizada em períodos constantes de tempo.

¹⁹ Esta associação poderia ser feita implicitamente, isto é poderiam ser criados objetos a partir destes dois tipos, cujo acesso usaria esta associação de forma implícita.

²⁰ No caso de atributos do tipo `real`, entende-se que são uma série temporal com uma amostra apenas.

Este grupo é composto de quatro operadores: `resample`, `resampleexact`, `interpolatelin` e `columnsync`. Os três primeiros operam com objetos do tipo `RealArray` e o último opera com *streams* de tuplas que contém atributos do tipo `RealArray`, de forma análogo ao operador `column`. Nos três primeiros casos, um novo `RealArray` de tamanho k (parâmetro) é criado a partir de um operando de tamanho j . O sinal é reconstruído e depois re-amostrado em k instâncias equidistantes.

- `resample` – O sinal é reconstruído em um **domínio contínuo**, em patamares. Valores não existentes são calculados a partir do valor existente anterior (Figura 41);
- `resampleexact` – O sinal é reconstruído em um **domínio discreto**. Valores não existentes, são tratados como indefinidos;
- `interpolatelin` – O sinal é reconstruído em um **domínio contínuo**. Elementos novos são calculados a partir de uma interpolação linear²¹ realizada no *array* inicial (Figura 42);
- `columnsync` – Considerando-se que o operador `column` não aplica operações de pré-sincronização, gerando sempre valores indefinidos ao final, pode-se que este operador faz o mesmo, porém aplicando a operação `resample`, tendo como parâmetro a cardinalidade do maior `RealArray` dentre os atributos da linha processada.

²¹ Outros tipos de interpolação (quadrática, cúbica, ...) podem ser imaginadas e, futuramente, incluídas como operadores similares.

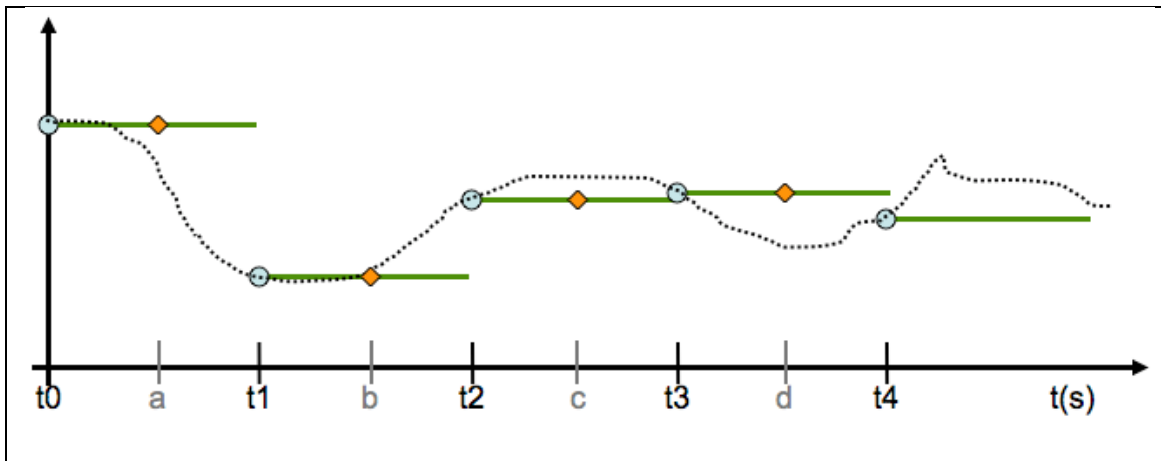


Figura 41 - Exemplo de re-amostragem com patamares

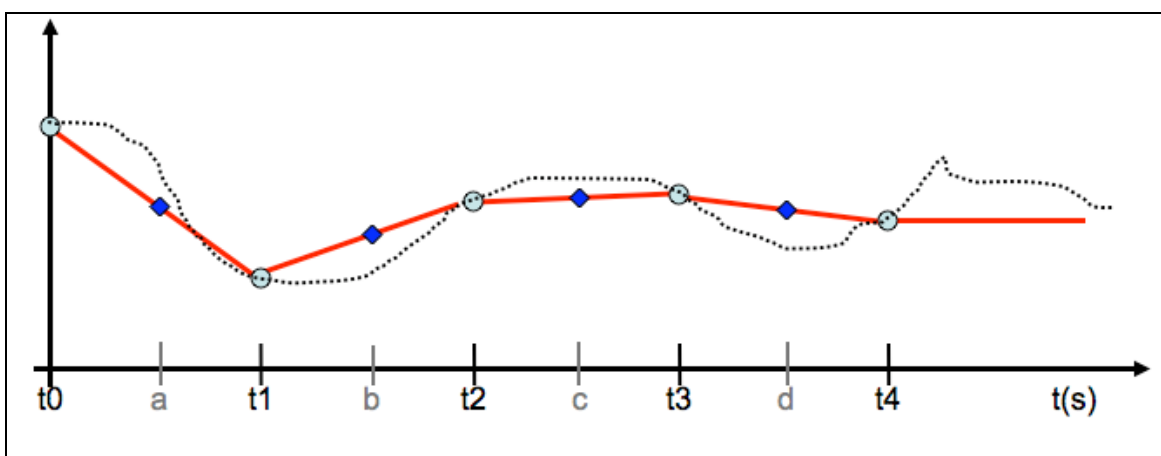


Figura 42 - Exemplo de re-amostragem com interpolação linear

Em ambas as figuras acima, temos os valores coletados nos instantes t_0 a t_4 , representando a curva traçada pela linha pontilhada. Nestes casos desejamos re-amostrar o resultado acrescentando-se mais 4 pontos, respectivos às amostras a, b, c e d. Para tal, é preciso reconstruir a curva original em um domínio contínuo de tempo. Na Figura 41, pode-se ver que a reconstrução deste sinal foi feita com patamares; na Figura 42, a reconstrução do sinal foi feita através de uma interpolação linear.²²

²² Pelos exemplos apresentados, pode-se supor, à princípio, que a interpolação linear é uma forma de re-amostragem mais fidedigna ao sinal original e, portanto, intuir-se da necessidade de outros operadores. Isto, porém, não é verdade. É relativamente simples apresentar exemplos em que uma representação por patamares será mais fiel ao sinal original do que uma interpolação linear.

6.5.3 Operadores de Extração de Intervalos

Ainda mantendo-se a semântica temporal, uma das operações que se deseja efetuar é a da extração (consulta) de elementos de uma série temporal a partir de um índice. Neste caso este índice será um intervalo de tempo e o resultado da consulta será a série temporal que possui intercessão com o índice.

A Figura 43, apresenta uma consulta desta natureza. Uma série temporal, representada por um `RealArray` relacionado a um intervalo de tempo `TimeSpan`, possui seis elementos e a duração de uma hora (a partir de um t_0 qualquer). Deseja-se encontrar a série que abrange o intervalo de tempo dos 25 aos 45 minutos (a partir do mesmo t_0). Como pode ser visto, a série resultante possui os elementos de valor 3, 4 e 5 que intercedem com o intervalo solicitado.

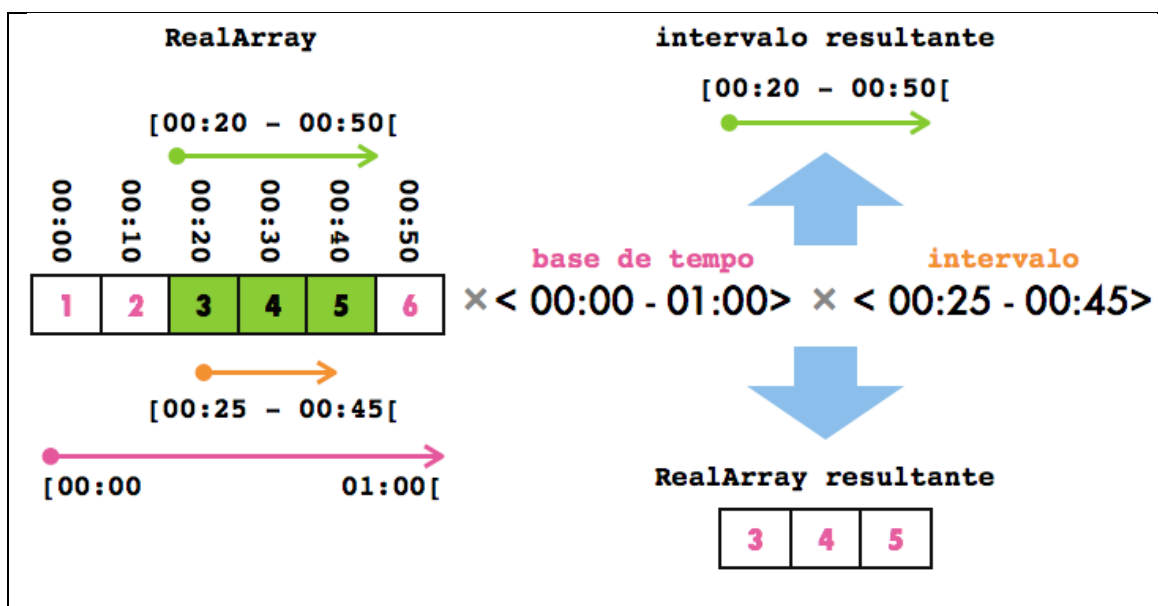


Figura 43 - Extração de elementos de uma série temporal

É importante ressaltar, que o intervalo resultante não é igual ao intervalo da consulta. Ele representa um intervalo que vai dos 20 aos 50 minutos do intervalo original, pois deve manter a consistência com a série original.

Dada esta característica, é necessário separar os operadores de extração em dois. O primeiro, `Iextract` recupera o `RealArray` correspondente, enquanto o segundo `Iextractspan`, recupera o intervalo de tempo (`TimeSpan`) ao qual o primeiro operador se refere.²³

```
File Edit View Search Terminal Help

Secondo => query ra7;

[ 1 ; 2 ; ..(+2).. ; 5 ; 6 ]
Secondo => query H1;

tspan: [ 2000-01-03 ---- 2000-01-03-01:00 ]
Secondo => query H2;

tspan: [ 2000-01-03-00:25 ---- 2000-01-03-00:45 ]
Secondo => query Iextract(ra7,H1,H2);

[ 3 ; 4 ; 5 ]
Secondo => query Iextractspan(ra7,H1,H2);

tspan: [ 2000-01-03-00:20 ---- 2000-01-03-00:50 ]
Secondo =>
```

Figura 44 - Utilização dos operadores `Iextract` e `Iextractspan`

A Figura 44, mostra a utilização destes operadores, assim como exemplificado na Figura 43. Foram criados um `RealArray` (`ra7`) e dois intervalos `TimeSpan` (`H1` e `H2`) de modo a refletir, precisamente, o exemplo anterior, como pode ser visto através das três primeiras consultas.

A consulta:

```
query Iextract(ra7,H1,H2);
```

Produz como resultado um `RealArray` de três elementos, contendo os valores 3, 4 e 5; enquanto que a consulta:

²³ Operadores que operam com semântica temporal utilizando `RealArrays` e `TimeSpan`, sempre que possível, receberão o prefixo “I”, significando que são “indexados”.

```
query Iextractspan(ra7,H1,H2);
```

Apresenta o intervalo de tempo que foi extraído da série temporal <ra7,H1>.

6.5.4 Operadores Trans-Relacionais Semânticos

Neste momento, em que os aspectos da aplicação de uma Semântica Temporal a uma relação de `RealArray` e `TimeSpan` foram devidamente explicados e exemplificados, pode-se seguir com a extensão destes conceitos aos operadores trans-relacionais mostrados anteriormente (vide seção 6.2).

Mantendo-se a coerência de nomenclatura²³, estes novos operadores são : `Icolumn`, `Iuncolumn`, `Itranspose`, `Iuntranspose` e `Iwindow`;

Os exemplos anteriores trabalhavam com tabelas que continham apenas atributos trans-relacionais (do tipo `RealArray` e `real`). Embora o modelo Relacional possua operadores que possam adequar as tabelas a este formato (projeção e filtragem, por exemplo), a inclusão de um atributo do tipo `TimeSpan`, necessário para a aplicação da semântica temporal aos atributos trans-relacionais, quebraria este paradigma. De modo que novos operadores precisaram ser criados para a adequação à esta semântica fosse feita de forma consistente e coerente.

Para tal, os atributos de uma tabela foram divididos em três classes²⁴ conforme a sua função como operandos. São eles: chave, índice (base) temporal e atributos trans-relacionais.

- O índice temporal (ou base temporal) representa um atributo do tipo `TimeSpan` que regulará a correta aplicação da semântica temporal na tabela;
- Os atributos trans-relacionais são todos aqueles que serão operados diretamente pelo operador, ou seja, são os operandos propriamente ditos;

²⁴ No sentido simples do termo, sem conotações ao paradigma de Orientação a Objetos.

- A chave corresponde a todos os outros atributos que não sofrerão a ação do operador. Eles serão “copiados” para a tabela final;

A denominação “**chave**”, foi escolhida por que estes atributos serão responsáveis por uma ação de “**quebra**” na função do operador trans-relacional semântico. Esta “quebra” é similar à ação dos operadores de agregação do Modelo Relacional, como por exemplo o `GROUP BY` da linguagem SQL.

Podemos também comparar alguns destes operadores com os operadores *Map / Reduce*, que são comentados em Osegawara et al [67]. Em operadores que se comportam como *Map*:

- A cardinalidade da relação é aumentada (número de linhas da tabela);
- A chave é copiada para cada linha;
- O `RealArray` é distribuído, criando um atributo do tipo `real` para cada linha;
- O índice de tempo é dividido, mantendo a semântica temporal, para cada linha;
- `Icolumn`;

Operadores que se comportam como *Reduce* têm:

- A cardinalidade da relação diminuída;
- O `RealArray` é consolidado a partir de atributos do tipo `real`;
- O índice de tempo é consolidado, concatenando-se todos os índices menores;
- `Iuncolumn`;

A operação de “quebra”, mencionada anteriormente, faz sentido quando utilizada em operadores que executam um *Reduce*, pois como uma tupla é criada a partir de várias, é necessário saber quando deve-se “quebrar” esta linha e começar uma nova.

Existem três eventos que forçam a finalização (“quebra”) da tupla corrente e fazem o sistema criar uma nova, são eles:

- Quebra da chave, quando os atributos que compõem a chave diferem daqueles da linha anterior;
- Adjacência, quando o índice temporal de uma tupla não é adjacente ao do grupo anterior;
- Intervalo de amostragem, quando o intervalo da nova tupla (frequência de aquisição) não é o mesmo das tuplas do grupo anterior.

Estes eventos estão ligados diretamente ao funcionamento do operador união de séries temporais (seção 5.1.3.1) de modo a preservar a consistência da série temporal resultante.

Os operadores `Itranspose` e `Iuntranspose` não alteram o número de linhas da tabela, e portanto, não utilizam estes conceitos. Já o operador `Iwindow` possui um comportamento peculiar em relação aos demais. A cardinalidade do resultado não pode ser definida à priori; são criados elementos tanto do tipo `real` quanto `RealArray` e o índice de tempo resultante é calculado a partir original.

Para exemplificar o uso destes novos operadores, foram criadas duas tabelas, `sensorDATA` e `sensor2DATA`. Ambas simulam dados coletados de sensores e contém uma informação de data e hora de coleta (de nome “`ts`”, já no formato `TimeSpan`), um atributo que corresponderia a uma fase do teste (“`phase`”, do tipo inteiro); e os dados coletados. Na primeira tabela (`sensorDATA`, Figura 45), estes dados estão no formato `real`, cada dado em uma coluna. Na outra (`sensor2DATA`, Figura 47) os valores estão já no formato `RealArray`. Esta última tabela visa simular a captura de dados com diferentes taxas de aquisição, e podemos ver linhas com diferentes cardinalidades nos `RealArrays`.

The screenshot shows a window titled "Secundo-GUI (Relations/viewer)". The main area contains a command prompt with the following text:

```

Sec>
Sec>
Sec>
Sec>
Sec>
Sec>query sensorDATA
query sensorDATA...successful
see result in object list
Sec>

```

To the right of the command prompt is a menu with buttons: sh..., hide, re..., clear, save, load, st..., re... Below the menu is a list of queries: query ra4; query ra5; query ra6; list algebra RealArrayAlgebra; query ts1; query ts2;

Below the command prompt is a table titled "query sensorDATA_3". The table has the following columns: ts, phase, s1, and s2.

ts	phase	s1	s2
("2000-01-03" "2000-01-03-00:00:01")	1	1.0	11.0
("2000-01-03-00:00:01" "2000-01-03-00:00:02")	1	2.0	12.0
("2000-01-03-00:00:02" "2000-01-03-00:00:03")	1	3.0	13.0
("2000-01-03-00:00:03" "2000-01-03-00:00:04")	1	4.0	14.0
("2000-01-03-00:00:04" "2000-01-03-00:00:05")	1	5.0	15.0
("2000-01-04" "2000-01-04-00:00:01")	1	1.1	11.1
("2000-01-04-00:00:01" "2000-01-04-00:00:02")	1	2.1	12.1
("2000-01-04-00:00:02" "2000-01-04-00:00:03")	1	3.1	13.1
("2000-01-04-00:00:03" "2000-01-04-00:00:06")	1	4.2	14.2
("2000-01-04-00:00:06" "2000-01-04-00:00:09")	1	5.2	15.2
("2000-01-04-00:00:09" "2000-01-04-00:00:12")	1	6.2	16.2
("2000-01-04-00:00:12" "2000-01-04-00:00:15")	2	7.3	17.3
("2000-01-04-00:00:15" "2000-01-04-00:00:18")	2	8.3	18.3
("2000-01-04-00:00:18" "2000-01-04-00:00:20")	2	9.4	19.4
("2000-01-04-00:00:20" "2000-01-04-00:00:22")	2	10.4	20.4

Figura 45 - Tabela sensorDATA

Analisando-se a tabela `sensorDATA`, podemos notar que as colunas `s1` e `s2` estão sincronizadas entre si. Isto é, para cada medida de `s1` existe uma medida feita no mesmo momento para `s2`.

Os valores de `s1` e `s2` foram criados, arbitrariamente, de modo a apontar para as peculiaridades da tabela, por exemplo, que a tabela apresenta descontinuidades na medição. Podemos notar que:

- Na quinta linha, a medida foi tomada aos 5 segundos do dia 3, e na linha subsequente a medida foi tomada já no dia 4. A mudança no valor de `s1` de 5.0 para 1.1 auxilia esta percepção;
- Na nona linha (valor de `s1` igual a 4.2) existe uma mudança na taxa de aquisição. As medições anteriores foram tomadas à cada segundo, enquanto que a partir desta linha às medições foram feitas à cada três segundos;

- Na décima-segunda linha (s_1 igual a 7.3) ocorre uma mudança na fase do projeto (por algum motivo alheio);
- Na décima-quarta linha (s_1 igual a 9.4) ocorre novamente uma mudança na taxa de aquisição, com as próximas amostras tomadas a um intervalo de aquisição de dois segundos.

6.5.5 Operador `Iuncolumn`

Com esta tabela será realizada uma operação `Iuncolumn`. O objetivo desta operação é transformar as colunas do tipo `real` em colunas do tipo `RealArray`. O índice de tempo utilizado será o atributo `ts`. O atributo `phase` será considerado como “chave” nesta operação por não ser nem um índice de tempo, nem de um tipo trans-relacional. Conforme explicado anteriormente, a operação de `Iuncolumn` é do tipo *Reduce*, isto é, produzirá como resultado uma tabela com número de linhas menor do que a tabela original. Dado os conceitos de “quebra” que são levados em consideração no momento de gerar estas novas linhas, podemos notar que as peculiaridades apresentadas na tabela `sensorDATA` são todas candidatas à geração destas quebras. Neste exemplo será mostrado o agrupamento da tabela em arranjos de cardinalidade igual a cinco. A sintaxe do comando `Iuncolumn` para esta operação é:

```
query sensorDATA feed Iuncolumn[ts,5,s1,s2] consume;
```

Como podemos ver na Figura 46, o comando foi bem-sucedido, gerando uma tabela com 5 linhas. Se examinarmos cada um destas linhas poderemos ver os pontos em que ocorreram as “quebras”, conforme as peculiaridades listadas. Na maioria dos casos não foi possível a criação de *arrays* com tamanho 5 devido às “quebras”. Pode-se notar, também, que o campo `ts` foi consolidado com a união de todos os períodos de aquisição que formaram cada linha.

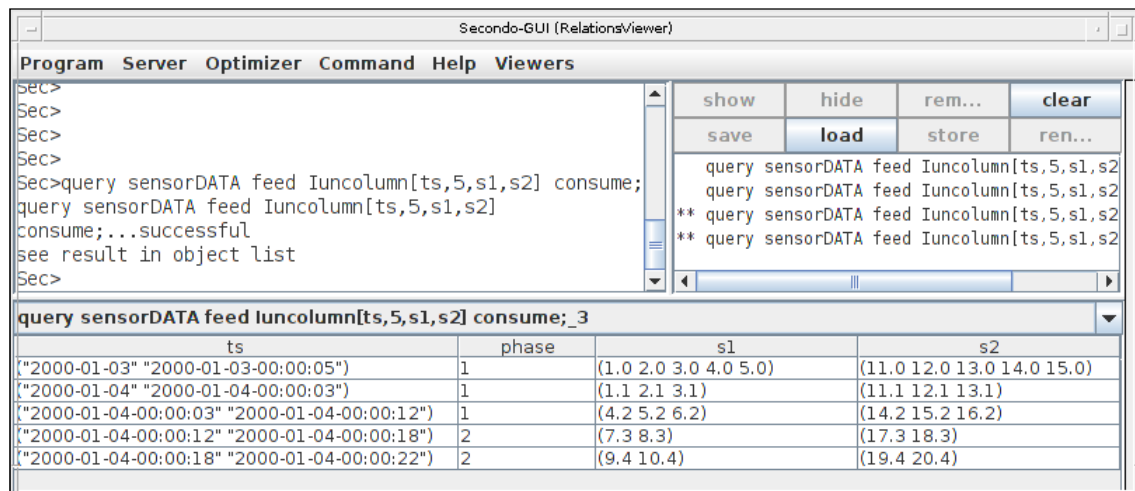


Figura 46 - Exemplo de uso do operador luncolumn

O resultado do comando `Iuncolumn` possui grande semelhança com a tabela `sensor2DATA`, mostrada na figura seguinte.

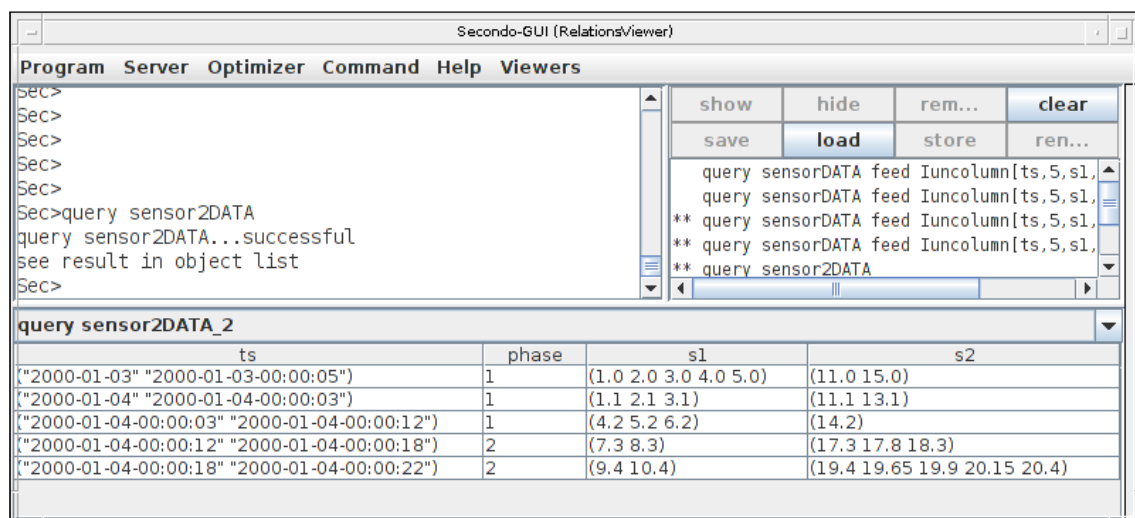


Figura 47 - Tabela sensor2DATA

De fato, este operador foi utilizado para gerar a segunda tabela, que foi posteriormente modificada na coluna `s2` para servir de exemplo para o operador `Icolumn`.

Este último visa fazer a operação inversa do primeiro, ou seja, transformar linhas com atributos do tipo `RealArray` em atributos `real`, comportando-se como um operador do tipo `Map`.

6.5.6 Operador `Icolumn`

Neste exemplo, podemos simular uma situação em que os sensores não estão sincronizados entre si. Este é um caso comum em testes cujas variáveis adquiridas apresentam comportamentos diferentes, como por exemplo pressão e temperatura. A pressão pode apresentar mudanças extremamente bruscas – um balão, ao ser furado com um alfinete, por exemplo – enquanto a temperatura apresenta mudanças muito lentas em um processo. Nestes casos, a taxa de aquisição destas duas variáveis é diferente para não se coletar dados desnecessários.

Entretanto, para se correlacionar estas duas variáveis, é necessário sincronizá-las. E para tal faremos uso do operador `Icolumn`.

Em operações do tipo `Map` não é necessário utilizar o conceito de “quebra”, visto como o número de linhas a serem geradas deverá ser maior do que o número de linhas iniciais, o operador irá transformar todas as colunas cujos atributos forem do tipo `RealArray`, observando um atributo índice como base de tempo (que será dividido) e as demais colunas serão copiadas. Quando uma linha tiver mais de um atributo `RealArray`, eles serão sincronizados, utilizando-se a forma `resample`, apresentada na seção 6.5.2.

The screenshot shows the Secundo-GUI interface. The command prompt area contains the following text:

```

sec>
Sec>
Sec>
Sec>
Sec>query sensor2DATA feed Icolumn[ts] consume;
query sensor2DATA feed Icolumn[ts]
consume;...successful
see result in object list
Sec>

```

Below the command prompt, a table is displayed with the title "query sensor2DATA feed Icolumn[ts] consume;_1". The table has four columns: "ts", "phase", "s1", and "s2". The data rows are as follows:

ts	phase	s1	s2
("2000-01-03" "2000-01-03-00:00:01")	1	1.0	11.0
("2000-01-03-00:00:01" "2000-01-03-00:00:02")	1	2.0	11.0
("2000-01-03-00:00:02" "2000-01-03-00:00:03")	1	3.0	11.0
("2000-01-03-00:00:03" "2000-01-03-00:00:04")	1	4.0	15.0
("2000-01-03-00:00:04" "2000-01-03-00:00:05")	1	5.0	15.0
("2000-01-04" "2000-01-04-00:00:01")	1	1.1	11.1
("2000-01-04-00:00:01" "2000-01-04-00:00:02")	1	2.1	11.1
("2000-01-04-00:00:02" "2000-01-04-00:00:03")	1	3.1	13.1
("2000-01-04-00:00:03" "2000-01-04-00:00:06")	1	4.2	14.2
("2000-01-04-00:00:06" "2000-01-04-00:00:09")	1	5.2	14.2
("2000-01-04-00:00:09" "2000-01-04-00:00:12")	1	6.2	14.2
("2000-01-04-00:00:12" "2000-01-04-00:00:14")	2	7.3	17.3
("2000-01-04-00:00:14" "2000-01-04-00:00:16")	2	7.3	17.8
("2000-01-04-00:00:16" "2000-01-04-00:00:18")	2	8.3	18.3
("2000-01-04-00:00:18" "2000-01-04-00:00:18.800")	2	9.4	19.4
("2000-01-04-00:00:18.800" "2000-01-04-00:00:19.600")	2	9.4	19.65
("2000-01-04-00:00:19.600" "2000-01-04-00:00:20.400")	2	9.4	19.9
("2000-01-04-00:00:20.400" "2000-01-04-00:00:21.200")	2	10.4	20.15
("2000-01-04-00:00:21.200" "2000-01-04-00:00:22")	2	10.4	20.4

Figura 48 - Exemplo de uso do operador Icolumn

A sintaxe de utilização do operador é bastante simples e o resultado pode ser visto na Figura 48:

```
query sensor2DATA feed Icolumn[ts] consume;
```

Podemos notar nas últimas linhas da tabela, que o atributo *s1* com o valor 9.4 foi repetido várias vezes de modo apresentar uma sincronia com o atributo *s2*. Valores de tempo correspondentes aos instantes destas “novas” medições também foram incluídos (atributo *ts*).

6.5.7 Operadores Itranspose e Iuntranspose

Como os operadores *transpose* e *untranspose* trabalham apenas com tabelas que contenham atributos trans-relacionais (*real* e *RealArray*) foi necessário estendê-los

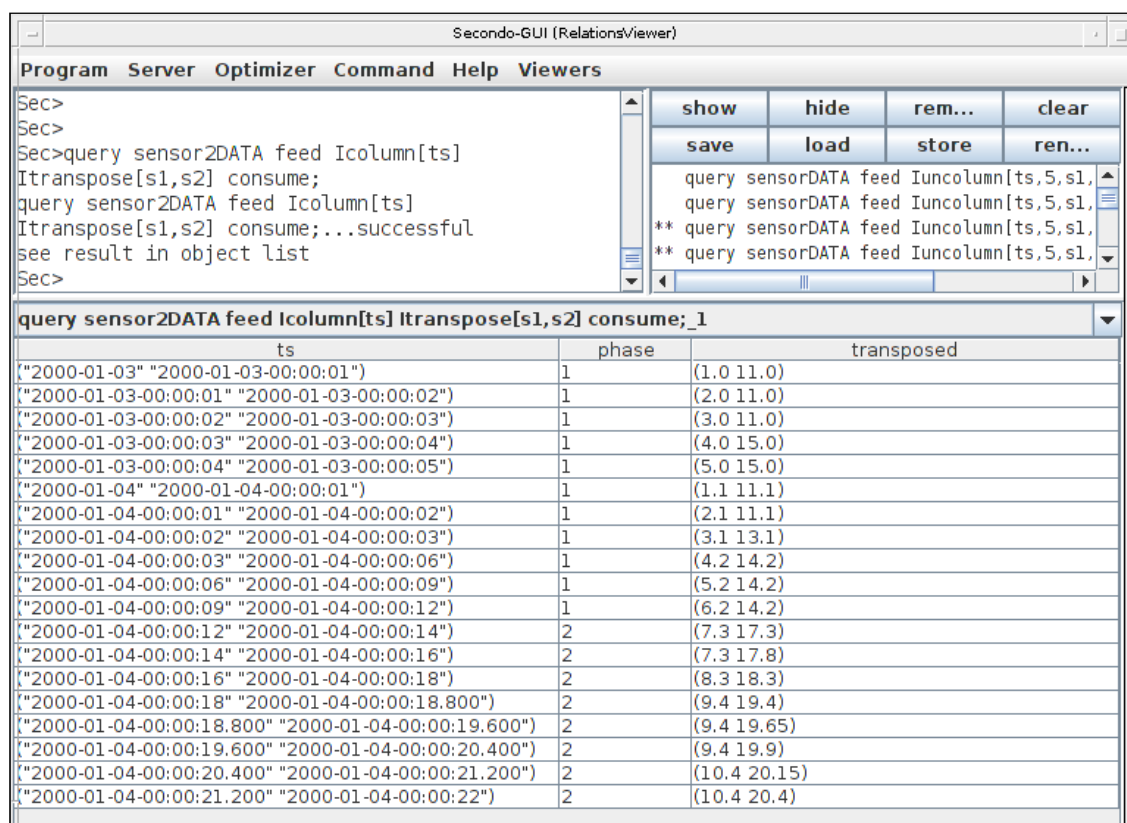
para trabalhar com tabelas mais diversas. Como esta é a única diferença entre eles, esta seção apresentará apenas seus exemplos e resultados.

Como base, foi utilizada a tabela `sensor2DATA` em conjunto com um operador `Icolumn` que projeta os atributos da tabela em colunas.

A diferença sintática no uso dos operadores `Itranspose` e `Iuntranspose` está na especificação das colunas que se deseja operar. No caso do operador `Itranspose`, a sintaxe do exemplo foi:

```
query sensor2DATA feed Icolumn[ts] Itranspose[s1,s2] consume;
```

Como pode ser notado, as colunas `s1` e `s2` da tabela foram escolhidas para serem transpostas, criando uma nova coluna chamada `transposed`. O resultado pode ser visto na Figura 49.



The screenshot shows the Secondo-GUI interface with a command window and a results table. The command window shows the following interaction:

```
Sec>
Sec>
Sec>query sensor2DATA feed Icolumn[ts]
Itranspose[s1,s2] consume;
query sensor2DATA feed Icolumn[ts]
Itranspose[s1,s2] consume;...successful
see result in object list
Sec>
```

The results table is titled `query sensor2DATA feed Icolumn[ts] Itranspose[s1,s2] consume; 1` and contains the following data:

ts	phase	transposed
("2000-01-03" "2000-01-03-00:00:01")	1	(1.0 11.0)
("2000-01-03-00:00:01" "2000-01-03-00:00:02")	1	(2.0 11.0)
("2000-01-03-00:00:02" "2000-01-03-00:00:03")	1	(3.0 11.0)
("2000-01-03-00:00:03" "2000-01-03-00:00:04")	1	(4.0 15.0)
("2000-01-03-00:00:04" "2000-01-03-00:00:05")	1	(5.0 15.0)
("2000-01-04" "2000-01-04-00:00:01")	1	(1.1 11.1)
("2000-01-04-00:00:01" "2000-01-04-00:00:02")	1	(2.1 11.1)
("2000-01-04-00:00:02" "2000-01-04-00:00:03")	1	(3.1 13.1)
("2000-01-04-00:00:03" "2000-01-04-00:00:06")	1	(4.2 14.2)
("2000-01-04-00:00:06" "2000-01-04-00:00:09")	1	(5.2 14.2)
("2000-01-04-00:00:09" "2000-01-04-00:00:12")	1	(6.2 14.2)
("2000-01-04-00:00:12" "2000-01-04-00:00:14")	2	(7.3 17.3)
("2000-01-04-00:00:14" "2000-01-04-00:00:16")	2	(7.3 17.8)
("2000-01-04-00:00:16" "2000-01-04-00:00:18")	2	(8.3 18.3)
("2000-01-04-00:00:18" "2000-01-04-00:00:18.800")	2	(9.4 19.4)
("2000-01-04-00:00:18.800" "2000-01-04-00:00:19.600")	2	(9.4 19.65)
("2000-01-04-00:00:19.600" "2000-01-04-00:00:20.400")	2	(9.4 19.9)
("2000-01-04-00:00:20.400" "2000-01-04-00:00:21.200")	2	(10.4 20.15)
("2000-01-04-00:00:21.200" "2000-01-04-00:00:22")	2	(10.4 20.4)

Figura 49 - Exemplo de uso do operador `Itranspose`

Já no comando seguinte:

```
query sensor2DATA feed Icolumn[ts] Itranspose[s1,s2]
    Iuntranspose[transposed,c1,c2] consume;
```

A coluna `transposed` é escolhida para ser transformada nas colunas `c1` e `c2`. O operador `Iuntranspose` foi encadeado no final do comando do exemplo anterior de forma a mostrar como estes operadores são inversos. O resultado pode ser visto na Figura 50.

The screenshot shows the Secundo-GUI interface. The command window contains the following text:

```
Sec>
Sec>
Sec>query sensor2DATA feed Icolumn[ts] Itranspose[s1,s2]
Iuntranspose[transposed,c1,c2] consume;
query sensor2DATA feed Icolumn[ts] Itranspose[s1,s2]
Iuntranspose[transposed,c1,c2] consume;...successful
see result in object list
Sec>
```

The data table below the command window has the following structure:

ts	phase	c1	c2
("2000-01-03" "2000-01-03-00:00:01")	1	1.0	11.0
("2000-01-03-00:00:01" "2000-01-03-00:00:02")	1	2.0	11.0
("2000-01-03-00:00:02" "2000-01-03-00:00:03")	1	3.0	11.0
("2000-01-03-00:00:03" "2000-01-03-00:00:04")	1	4.0	15.0
("2000-01-03-00:00:04" "2000-01-03-00:00:05")	1	5.0	15.0
("2000-01-04" "2000-01-04-00:00:01")	1	1.1	11.1
("2000-01-04-00:00:01" "2000-01-04-00:00:02")	1	2.1	11.1
("2000-01-04-00:00:02" "2000-01-04-00:00:03")	1	3.1	13.1
("2000-01-04-00:00:03" "2000-01-04-00:00:06")	1	4.2	14.2
("2000-01-04-00:00:06" "2000-01-04-00:00:09")	1	5.2	14.2
("2000-01-04-00:00:09" "2000-01-04-00:00:12")	1	6.2	14.2
("2000-01-04-00:00:12" "2000-01-04-00:00:14")	2	7.3	17.3
("2000-01-04-00:00:14" "2000-01-04-00:00:16")	2	7.3	17.8
("2000-01-04-00:00:16" "2000-01-04-00:00:18")	2	8.3	18.3
("2000-01-04-00:00:18" "2000-01-04-00:00:18.800")	2	9.4	19.4
("2000-01-04-00:00:18.800" "2000-01-04-00:00:19.600")	2	9.4	19.65
("2000-01-04-00:00:19.600" "2000-01-04-00:00:20.400")	2	9.4	19.9
("2000-01-04-00:00:20.400" "2000-01-04-00:00:21.200")	2	10.4	20.15
("2000-01-04-00:00:21.200" "2000-01-04-00:00:22")	2	10.4	20.4

Figura 50 - Exemplo de uso do operador `Iuntranspose`

6.5.8 Operador `Iwindow`

Por fim, apresentamos o operador `Iwindow`. Este operador pode ser considerado como o mais complexo de todos. Análogo ao operador `window` (seção 6.2.5), que apresenta “janelas” de amostras, o `Iwindow` estende o anterior de modo a manter

uma consistência destas janelas com a semântica temporal. Para tal, o `Iwindow` se utiliza dos mesmos conceitos de “quebra” do operador `Iuncolumn`. Isto é, linhas consecutivas são testadas de modo a garantir que fazem parte do mesmo contínuo temporal, e que realmente formam uma “janela” consistente.

Conforme explicado na seção 6.2.5, e ilustrado na Figura 31, a consistência da “janela” de dados, por si só, já exige alguns cuidados. Uma “janela”, pode ter uma extremidade **aberta**, isto é conter elementos indefinidos, ou **fechada**, contendo apenas elementos que fazem parte da série. Estas extremidades podem ser abertas ou fechadas no início e/ou no fim.

A configuração destes parâmetros impactará diretamente o número de linhas da tabela resultante. Por exemplo, consideremos uma consulta a uma tabela com n linhas, e uma janela de tamanho k fechada em ambas as extremidades - isto é, sem a inclusão de elementos indefinidos :

- Para $k = n$, obteremos apenas 1 linha, ou seja, todos os elementos de n caberão em apenas 1 janela;
- Para $k < n$, obteremos $n-k+1$ janelas, o que corresponde ao número de “deslizamentos” da janela sobre o conjunto;
- Já para $k > n$, nenhuma janela com aqueles elementos poderá ser produzida.

Abrindo-se uma extremidade da janela, seja no início ou ao final ou ambas, obteremos um número maior de linhas pois se incluirão tantos elementos indefinidos quantos forem necessários de forma a completar a janela.

Consultas com janelas abertas não se constituíam em um problema para o operador `window`. Porém, o `Iwindow` produz não só a janela em si, como também referências a um elemento da janela específico (seu valor e período) e ao período de tempo ao qual a janela se refere. O elemento referenciado depende do fecho utilizado na janela, ou seja, quais extremidades estão abertas ou fechadas. A Tabela 10, apresenta a relação que especifica qual elemento será referenciado pelo operador.

A ideia por trás desta referência é apontar sempre para um elemento que faz parte da série original, isto é, não é indefinido. Na maioria dos casos considera-se o último elemento, excetuando-se o caso em que a janela é fechada no início, porém a consistência é impossível se a janela tiver ambas as extremidades em aberto.

Tabela 10 - Elemento referenciado pelo operador Iwindow

Início	Fim	Elemento Referenciado	Exemplo
Fechado	Fechado	Final	
Aberto	Fechado	Final	
Fechado	Aberto	Inicial	
Aberto	Aberto	Final	

Utilizando-se a tabela `sensorDATA` (Figura 45), um tamanho de janela de 5 elementos, fechada em ambas as extremidades, e o atributo `ts` como índice de tempo, obteremos o resultado mostrado na Figura 51, ao utilizarmos o comando:

```
query sensorDATA feed Iwindow[5,TRUE,TRUE,ts,s1,s2] consume;
```

Neste comando, 5 representa o tamanho da janela; `TRUE, TRUE`, indica que a janela é fechada tanto no início quanto no final, `ts` aponta para o índice de tempo e `s1, s2` são os atributos, propriamente ditos. Os atributos não mencionados, são utilizados como elementos para a “quebra”, no caso, o campo `phase`.

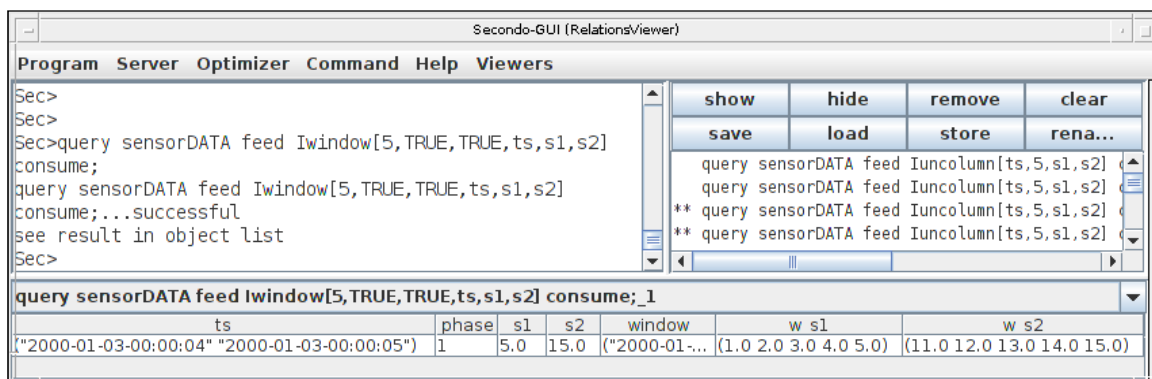


Figura 51 - Resultado do operador IWindow com janela fechada em ambas as extremidades

Podemos ver que a consulta só obteve uma linha de resultado. Isto deve-se ao fato de que apenas as cinco primeiras linhas da tabela `sensorDATA` apresentam a consistência necessária para serem operadas.

Foram incluídos três novos atributos no resultado:

- `window`: do tipo `TimeSpan`, que indica o período de tempo da janela;
- `w_s1` e `w_s2`: do tipo `RealArray` que apresentam os elementos da janela. A adição do prefixo "w_" é automática.

O atributo `ts` referencia o último elemento da janela, no caso, os valores 5.0 e 15.0 para `s1` e `s2`, respectivamente.

Podemos comparar este resultado com o que obteríamos ao usarmos uma janela aberta em ambas as extremidades. Isto pode ser feito mudando-se apenas os parâmetros de fecho, com o comando:

```
query sensorDATA feed Iwindow[5,FALSE,FALSE,ts,s1,s2] consume;
```

resultando na tabela mostrada na Figura 52.

ts	phase	s1	s2	window	w_s1	w_s2
"2000-01-03" "20...	1	1.0	11.0	("2000-01-02-23:59:56.001"...	(undef undef undef 1.0)	(undef undef undef 11.0)
"2000-01-03-00:0...	1	2.0	12.0	("2000-01-02-23:59:57.001"...	(undef undef undef 1.0 2.0)	(undef undef undef 11.0 12.0)
"2000-01-03-00:0...	1	3.0	13.0	("2000-01-02-23:59:58.001"...	(undef undef 1.0 2.0 3.0)	(undef undef 11.0 12.0 13.0)
"2000-01-03-00:0...	1	4.0	14.0	("2000-01-02-23:59:59.001"...	(undef 1.0 2.0 3.0 4.0)	(undef 11.0 12.0 13.0 14.0)
"2000-01-03-00:0...	1	5.0	15.0	("2000-01-03" "2000-01-03-...	(1.0 2.0 3.0 4.0 5.0)	(11.0 12.0 13.0 14.0 15.0)
undef	1	undef	undef	("2000-01-03-00:00:01" "20...	(2.0 3.0 4.0 5.0 undef)	(12.0 13.0 14.0 15.0 undef)
undef	1	undef	undef	("2000-01-03-00:00:02" "20...	(3.0 4.0 5.0 undef undef)	(13.0 14.0 15.0 undef undef)
undef	1	undef	undef	("2000-01-03-00:00:03" "20...	(4.0 5.0 undef undef undef)	(14.0 15.0 undef undef undef)
undef	1	undef	undef	("2000-01-03-00:00:04" "20...	(5.0 undef undef undef undef)	(15.0 undef undef undef undef)
"2000-01-04" "20...	1	1.1	11.1	("2000-01-03-23:59:56" "20...	(undef undef undef undef 1.1)	(undef undef undef undef 11.1)
"2000-01-04-00:0...	1	2.1	12.1	("2000-01-03-23:59:57" "20...	(undef undef undef 1.1 2.1)	(undef undef undef 11.1 12.1)
"2000-01-04-00:0...	1	3.1	13.1	("2000-01-03-23:59:58" "20...	(undef undef 1.1 2.1 3.1)	(undef undef 11.1 12.1 13.1)
undef	1	undef	undef	undef	(undef 1.1 2.1 3.1 undef)	(undef 11.1 12.1 13.1 undef)
undef	1	undef	undef	("2000-01-04" "2000-01-04-...	(1.1 2.1 3.1 undef undef)	(11.1 12.1 13.1 undef undef)
undef	1	undef	undef	("2000-01-04-00:00:01" "20...	(2.1 3.1 undef undef undef)	(12.1 13.1 undef undef undef)
undef	1	undef	undef	("2000-01-04-00:00:02" "20...	(3.1 undef undef undef undef)	(13.1 undef undef undef undef)
"2000-01-04-00:0...	1	4.2	14.2	("2000-01-03-23:59:51" "20...	(undef undef undef undef 4.2)	(undef undef undef undef 14.2)
"2000-01-04-00:0...	1	5.2	15.2	("2000-01-03-23:59:54" "20...	(undef undef undef 4.2 5.2)	(undef undef undef 14.2 15.2)
"2000-01-04-00:0...	1	6.2	16.2	("2000-01-03-23:59:57" "20...	(undef undef 4.2 5.2 6.2)	(undef undef 14.2 15.2 16.2)
undef	1	undef	undef	undef	(undef 4.2 5.2 6.2 undef)	(undef 14.2 15.2 16.2 undef)
undef	1	undef	undef	("2000-01-04-00:00:03" "20...	(4.2 5.2 6.2 undef undef)	(14.2 15.2 16.2 undef undef)
undef	1	undef	undef	("2000-01-04-00:00:06" "20...	(5.2 6.2 undef undef undef)	(15.2 16.2 undef undef undef)
undef	1	undef	undef	("2000-01-04-00:00:09" "20...	(6.2 undef undef undef undef)	(16.2 undef undef undef undef)
"2000-01-04-00:0...	2	7.3	17.3	("2000-01-04" "2000-01-04-...	(undef undef undef undef 7.3)	(undef undef undef undef 17.3)
"2000-01-04-00:0...	2	8.3	18.3	("2000-01-04-00:00:03" "20...	(undef undef undef 7.3 8.3)	(undef undef undef 17.3 18.3)
undef	2	undef	undef	undef	(undef undef 7.3 8.3 undef)	(undef undef 17.3 18.3 undef)
undef	2	undef	undef	undef	(undef 7.3 8.3 undef undef)	(undef 17.3 18.3 undef undef)
undef	2	undef	undef	("2000-01-04-00:00:12" "20...	(7.3 8.3 undef undef undef)	(17.3 18.3 undef undef undef)
undef	2	undef	undef	("2000-01-04-00:00:15" "20...	(8.3 undef undef undef undef)	(18.3 undef undef undef undef)
"2000-01-04-00:0...	2	9.4	19.4	("2000-01-04-00:00:10" "20...	(undef undef undef undef 9.4)	(undef undef undef undef 19.4)
"2000-01-04-00:0...	2	10.4	20.4	("2000-01-04-00:00:12" "20...	(undef undef undef 9.4 10.4)	(undef undef undef 19.4 20.4)
undef	2	undef	undef	undef	(undef undef 9.4 10.4 undef)	(undef undef 19.4 20.4 undef)
undef	2	undef	undef	undef	(undef 9.4 10.4 undef undef)	(undef 19.4 20.4 undef undef)
undef	2	undef	undef	("2000-01-04-00:00:18" "20...	(9.4 10.4 undef undef undef)	(19.4 20.4 undef undef undef)
undef	2	undef	undef	("2000-01-04-00:00:20" "20...	(10.4 undef undef undef undef)	(20.4 undef undef undef undef)

Figura 52 - Resultado do operador lwindow com janela aberta em ambas as extremidades

Nesta figura, podemos observar as situações estranhas que surgem ao se utilizar a janela aberta em ambas as extremidades. Logo na primeira linha, notamos que o atributo `ts` referencia ao primeiro elemento de `s1` (1.0), que por sua vez é o último elemento da janela `w_s1`. O atributo `window`, porém indica que a janela começa antes do instante da primeira amostra (2000-01-02-23:59:56!), isto por que, se a janela acaba com o valor da primeira amostra, ela deve, portanto, começar antes!

De forma análoga, após a quinta linha, a janela prossegue com a inclusão de elementos indefinidos. Porém, nestes casos, como se está referenciando sempre ao último elemento, tanto o índice de tempo como o elemento em si, são indefinidos!

No terceiro grupo de valores (4.2), ocorre outra peculiaridade. Desta vez a “quebra” foi devida a uma mudança na taxa de aquisição em comparação com o grupo anterior (3.1). Neste caso, o que pode ser observado é que a referência da janela (2000-01-03-23:59:51) possui uma interseção com a do grupo anterior. Elementos indefinidos foram incluídos nesta janela, quando haviam valores reais que poderiam ser usados. Como estes valores faziam parte de outro grupo, optou-se por utilizar os elementos indefinidos de forma a se manter uma consistência na taxa de aquisição.

Este último problema é consequência do uso de janelas abertas e pode vir a surgir também com as outras opções.

The screenshot shows a window titled "Secondo-GUI (RelationsViewer)". The main area contains a command prompt with the following text:

```

query sensorDATA feed lwindow[5,FALSE,TRUE,ts,s1,s2]
consume;...successful
see result in object list
Sec>query sensorDATA feed lwindow[5,FALSE,TRUE,ts,s1,s2]
consume;
query sensorDATA feed lwindow[5,FALSE,TRUE,ts,s1,s2]
consume;...successful
see result in object list
Sec>

```

Below the command prompt is a table with the following columns: ts, phase, s1, s2, window, w s1, and w s2. The table contains 16 rows of data, showing a sequence of time intervals and their corresponding window parameters.

ts	phase	s1	s2	window	w s1	w s2
"2000-01-03" "2...	1	1.0	11.0	("2000-01-02-23:59:56.00..."	(undef undef undef undef 1.0)	(undef undef undef undef 1...
"2000-01-03-00:...	1	2.0	12.0	("2000-01-02-23:59:57.00..."	(undef undef undef 1.0 2.0)	(undef undef undef 11.0 12.0)
"2000-01-03-00:...	1	3.0	13.0	("2000-01-02-23:59:58.00..."	(undef undef 1.0 2.0 3.0)	(undef undef 11.0 12.0 13.0)
"2000-01-03-00:...	1	4.0	14.0	("2000-01-02-23:59:59.00..."	(undef 1.0 2.0 3.0 4.0)	(undef 11.0 12.0 13.0 14.0)
"2000-01-03-00:...	1	5.0	15.0	("2000-01-03" "2000-01-0...	(1.0 2.0 3.0 4.0 5.0)	(11.0 12.0 13.0 14.0 15.0)
"2000-01-04" "2...	1	1.1	11.1	("2000-01-03-23:59:56" "...	(undef undef undef undef 1.1)	(undef undef undef undef 1...
"2000-01-04-00:...	1	2.1	12.1	("2000-01-03-23:59:57" "...	(undef undef undef 1.1 2.1)	(undef undef undef 11.1 12.1)
"2000-01-04-00:...	1	3.1	13.1	("2000-01-03-23:59:58" "...	(undef undef 1.1 2.1 3.1)	(undef undef 11.1 12.1 13.1)
"2000-01-04-00:...	1	4.2	14.2	("2000-01-03-23:59:51" "...	(undef undef undef undef 4.2)	(undef undef undef undef 1...
"2000-01-04-00:...	1	5.2	15.2	("2000-01-03-23:59:54" "...	(undef undef undef 4.2 5.2)	(undef undef undef 14.2 15.2)
"2000-01-04-00:...	1	6.2	16.2	("2000-01-03-23:59:57" "...	(undef undef 4.2 5.2 6.2)	(undef undef 14.2 15.2 16.2)
"2000-01-04-00:...	2	7.3	17.3	("2000-01-04" "2000-01-0...	(undef undef undef undef 7.3)	(undef undef undef undef 1...
"2000-01-04-00:...	2	8.3	18.3	("2000-01-04-00:00:03" "...	(undef undef undef 7.3 8.3)	(undef undef undef 17.3 18.3)
"2000-01-04-00:...	2	9.4	19.4	("2000-01-04-00:00:10" "...	(undef undef undef undef 9.4)	(undef undef undef undef 1...
"2000-01-04-00:...	2	10.4	20.4	("2000-01-04-00:00:12" "...	(undef undef undef 9.4 10.4)	(undef undef undef 19.4 20.4)

Figura 53 - Exemplo do operador lwindow com janela Aberta/Fechada

Utilizando-se o operador com janelas abertas em apenas uma extremidade podemos ver que os resultados são diferentes dependendo-se de qual lado queremos deixar aberto. No caso de janelas abertas no início (Figura 53), a tabela resultante é basicamente a mesma do exemplo anterior, retirando-se as linhas com referências indefinidas.

Já em janelas com o início fechado, optou-se por excluir grupos cuja primeira janela não fosse completa, como pode ser visto na Figura 54. Neste caso, como apenas o primeiro grupo apresentava esta característica, ele foi o único apresentado.

ts	phase	s1	s2	window	w s1	w s2
("2000-01-03" "2...	1	1.0	11.0	("2000-01-03" "2000-01-0...	(1.0 2.0 3.0 4.0 5.0)	(11.0 12.0 13.0 14.0 15.0)
("2000-01-03-00...	1	2.0	12.0	("2000-01-03-00:00:01" "...	(2.0 3.0 4.0 5.0 undef)	(12.0 13.0 14.0 15.0 undef)
("2000-01-03-00...	1	3.0	13.0	("2000-01-03-00:00:02" "...	(3.0 4.0 5.0 undef undef)	(13.0 14.0 15.0 undef undef)
("2000-01-03-00...	1	4.0	14.0	("2000-01-03-00:00:03" "...	(4.0 5.0 undef undef undef)	(14.0 15.0 undef undef undef)
("2000-01-03-00...	1	5.0	15.0	("2000-01-03-00:00:04" "...	(5.0 undef undef undef undef)	(15.0 undef undef undef un...

Figura 54 - Exemplo do operador Iwindow com janela Fechada/Aberta

Esta decisão foi proposital de modo a permitir ao usuário escolher um grau de consistência do resultado sem que fosse incluído mais um parâmetro na sintaxe do operador.

Neste último caso, também pode ser notado que a referência dada é a do primeiro elemento da janela.

Apesar de complexo, este operador é bastante poderoso e permite ao usuário aplicar vários tipos de filtros e transformadas aos dados de seu experimento. A referência ao elemento em questão (atributos *ts*, *s1* e *s2*) permitem que o usuário venha a substituí-los por um novo elemento, calculado pelo filtro, mantendo-se o índice de tempo e evitando o “*delay*” característico de filtros digitais. Este efeito pode ser visto na Figura 30, quando se aplica uma média móvel, a variável observada “leva tempo” para adequar-se a média.

7 Estudo de Caso

7.1 Proposta

Uma vez que foram apresentados tipos e operadores visando à integração de consultas de séries temporais a consultas clássicas relacionais e posterior otimização dessas consultas, bem como validações sobre a consistência dos mesmos e sua utilização, resta uma apresentação sobre o pragmatismo de sua utilização.

Para tal foi desenvolvida uma aplicação prática representativa da necessidade de consultas de séries temporais. A aplicação se baseou em dados reais, utilizando dados oriundos de testes em tubos flexíveis. Uma vez disponibilizados os dados, os operadores foram utilizados de forma a gerar resultados que não eram obtíveis sem os mesmos, ou que necessitariam de um trabalho extenso para tal, devido ao problema de "*impedance mismatch*".

Os resultados apresentados se assemelharam aos resultados obtidos no caso real, pela empresa que cedeu os dados, porém o estudo de caso foi desenvolvido à parte, utilizando a nova álgebra via SECONDO. O intuito deste estudo de caso é mostrar que as operações propostas beneficiam, de fato, um aplicação que possa fazer uso de uma implementação completa dos operadores desenvolvidos no protótipo.

Dada a especificidade do estudo de caso, faz-se necessária uma explanação do que é o problema que está sendo estudado, principalmente a sua importância na área de exploração de óleo e gás. Uma área estratégica para o desenvolvimento nacional.

7.2 Prólogo

O petróleo é atualmente, um dos recursos mais estratégicos do planeta. Além de ser transformado em energia, a indústria petroquímica o utiliza na manufatura de diversos polímeros que são utilizados em uma gama de produtos que vai de brinquedos,

embalagens para alimentos, plásticos nas indústrias automotivas, naval, civil, colas, solventes e até utilizado na indústria farmacêutica.

A importância estratégica do petróleo está explicitamente ligada à estrutura geopolítica mundial. Ou se tem o recurso ou é necessário comprá-lo no mercado. Esta situação não é mutuamente exclusiva, a composição química do petróleo pode fornecer um produto que pode ser interessante à petroquímica, porém indesejável para o refino de combustíveis. Portanto um país pode explorar, comprar e vender petróleo concomitantemente.

O componente geográfico influi no preço do produto com o custo e produção e transporte, do poço ao refino ao consumidor. Porém, é o componente político, hoje em dia, que mais afeta o preço. Pode-se dizer que a instabilidade política na região do oriente-médio é um dos principais fatores que atualmente encarecem o produto.

Dado este cenário, a prospecção de novas reservas é ampliada à localizações previamente consideradas inviáveis do ponto de vista financeiro. Um exemplo do fruto destas explorações é o chamado “pré-sal”, recentemente descoberto pela Petrobras, na costa brasileira. Localizada sob uma lâmina-d’ água de 2500 metros, encontra-se a camada de sal, que deve também ser perfurada até a localização das reservas, totalizando cerca de 5000 metros da superfície ao óleo. O custo de exploração e produção destes campos é extremamente alto, mas viabilizado pelo atual preço do produto e do potencial da reserva.

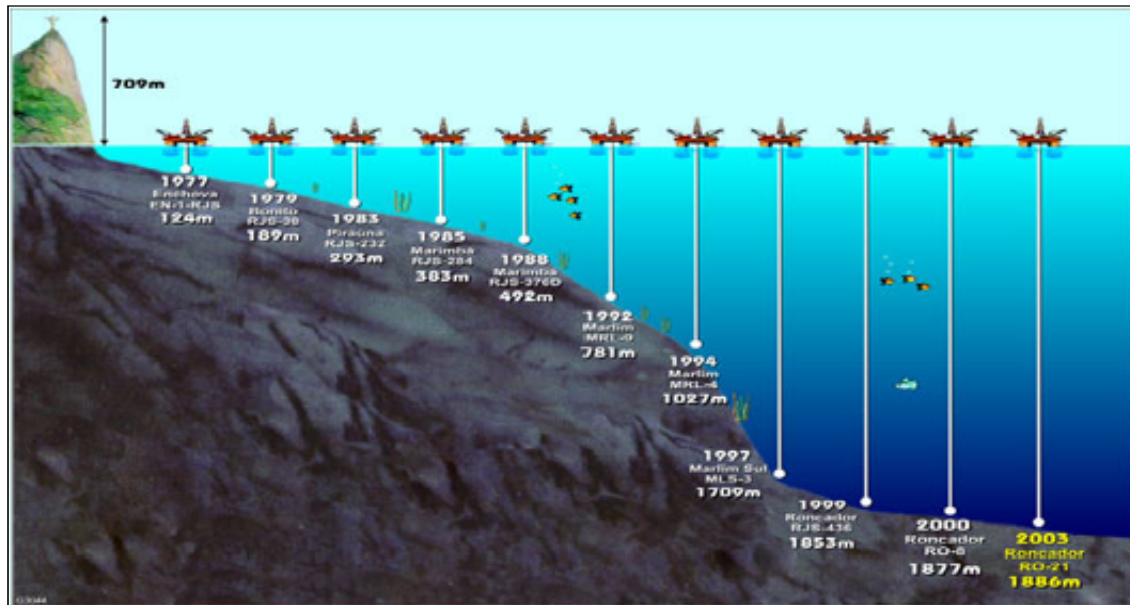


Figura 55 - Evolução da Profundidade de Exploração de Petróleo no Brasil (até 2003)

7.3 Exploração Offshore

O termo *Offshore* é utilizado para indicar a exploração de petróleo²⁵ sobre o mar, em estruturas chamadas de plataformas. Estas podem ser de diversos tipos: fixas, semissubmersíveis, SPAR e FPSO, por exemplo.

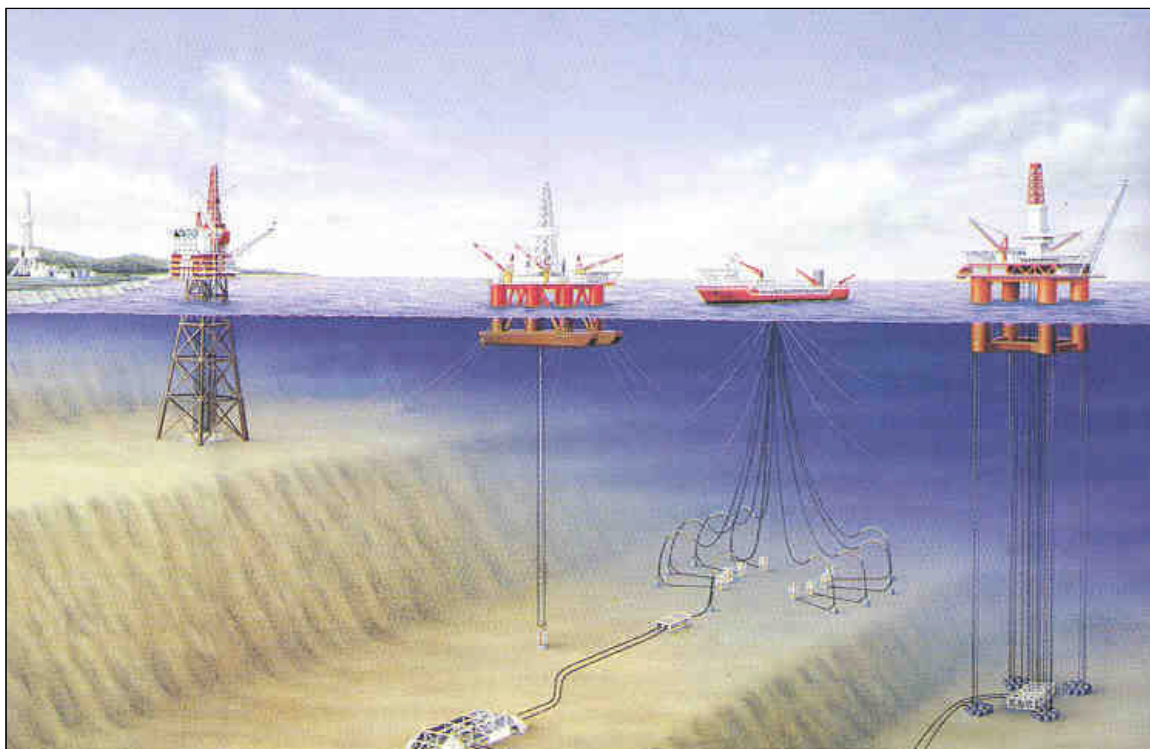


Figura 56 – Exemplos de tipos de plataformas

Plataformas são construídas em estaleiros navais e rebocadas para seus campos, onde são ancoradas ao fundo do mar. A escolha do tipo de plataforma a ser utilizada é determinada tanto pelas características do poço a ser explorado quanto pelas condições do mar ao qual será exposta. A construção, instalação e manutenção das estruturas submarinas as quais ela será conectada é feita por um segmento chamado de engenharia *Subsea*.

²⁵ O termo não é restritivo à exploração de petróleo. Pode ser utilizado, por exemplo, para uma estação geradora de energia eólica flutuante.

7.3.1 Engenharia Subsea

Sob o mar, várias estruturas estão presentes na exploração do petróleo. Não necessariamente, a plataforma situa-se verticalmente sobre a cabeça-de-poço. Muitas vezes esta unidade está ligada a um tubo rígido até um *manifold* de distribuição, de onde sai o tubo que leva o fluido até a plataforma.

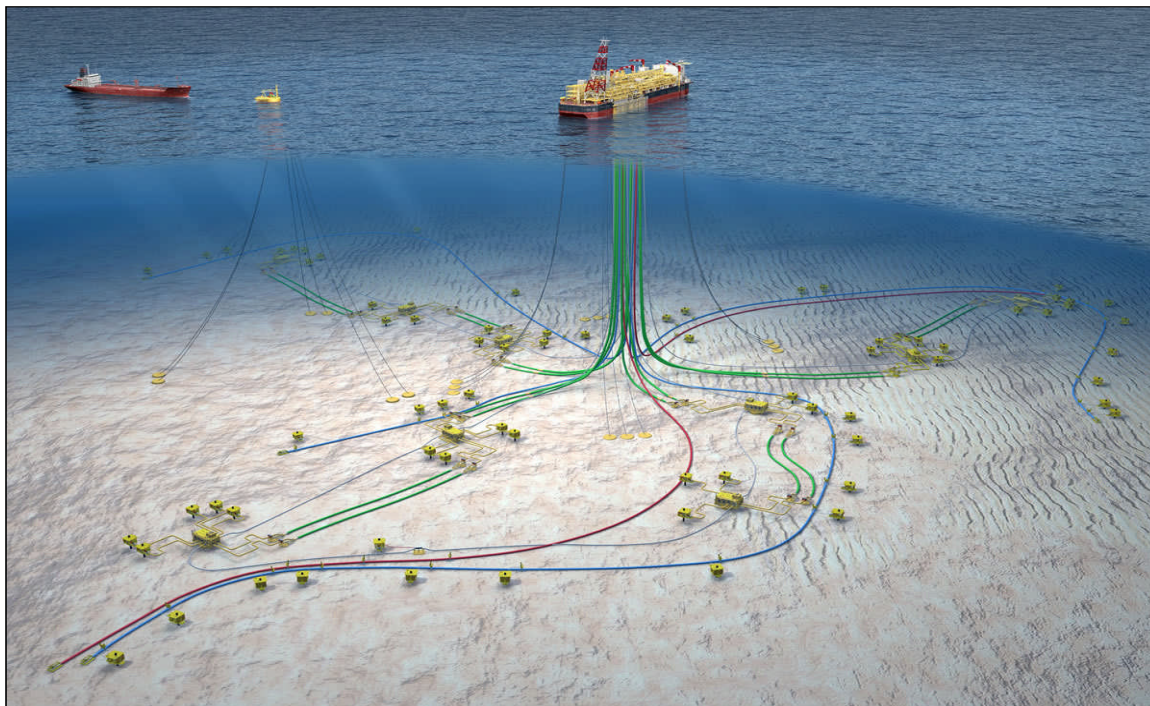


Figura 57- Configuração submarina de exploração (*Field Layout*)

7.4 Risers Flexíveis

Os tubos flexíveis, são muitas vezes chamados de *risers*, por “levantarem” o petróleo do fundo do mar à plataforma, são alvo de uma engenharia complexa. A ação de correntes submarinas, ondas e ventos, faz também com que o tubo apresente um comportamento dinâmico que sujeita sua estrutura a fadiga estrutural.

O custo de aquisição de um *riser* flexível não é, comparativamente, o maior dentre os custos fixos de exploração de um campo. Porém, a falha de um destes dutos certamente interromperá a produção, cujo custo diário é alto, ou, na pior das hipóteses, criará um desastre ambiental de grandes proporções. Ou seja, é uma peça chave do quebra-cabeças da exploração de petróleo *offshore*.

7.4.1 Engenharia de Risers Flexíveis

Um tubo flexível é composto de diversas camadas estruturais, cada uma com sua função, podendo ser estruturadas de acordo com as condições da localização em que o *riser* será instalado, comprimento da linha, e fluido que transportará (óleo, gás ou até mesmo água – para ser injetada no poço) .

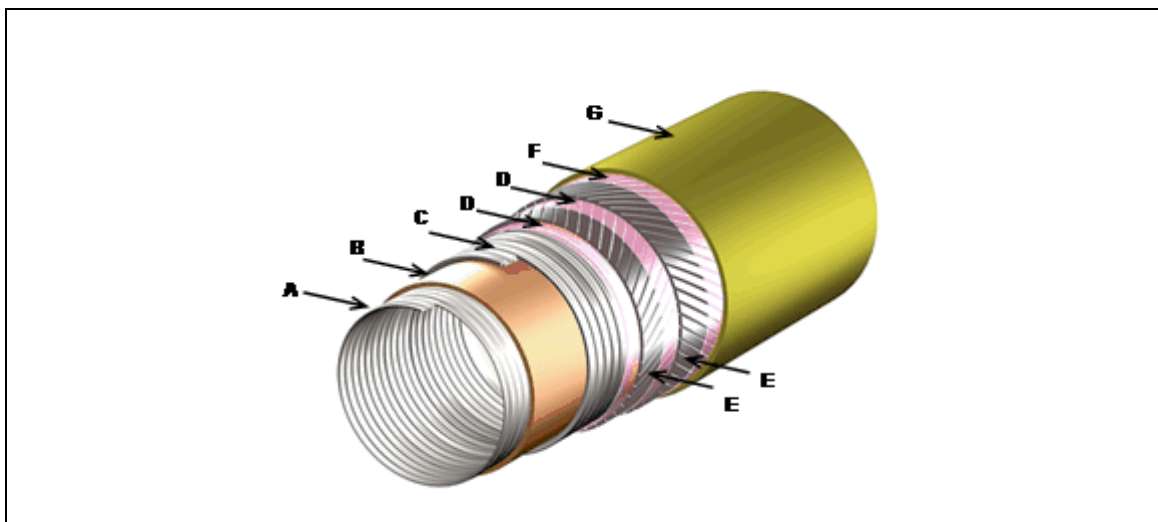


Figura 58 - Estrutura comum de um *Riser* Flexível

A Figura 58 apresenta um exemplo comum de uma destas estruturas. Nela podem ser vistas as diversas camadas que formam o tubo:

- *Bore*: Denominação da parte interna da linha, por onde o fluido escoar.

- Carcaça (A): Camada feita de “anéis” de aço , responsável por resistir às pressões externas;
- Camada de Pressão (B): Feita de material polimérico, ela envolve a carcaça e é responsável por manter a estanqueidade do tubo;
- Armadura de Pressão (C): Trata-se de uma espiral de aço, cujo perfil em forma de “s” ou “z”, trava a própria espiral na posição e protege a camada anterior da pressão interna;
- Camadas *Anti-Wear* (D e F): Protegem as armaduras de pressão e tração do atrito com a próxima camada;
- Armaduras de Tração (E) : São encontradas em pares. Cada camada é composta de diversos fios de aços (arames) enrolados helicoidalmente sobre as camadas anteriores. Para cada camada, sempre existe uma camada equivalente (seu par) enrolada no sentido oposto. Isto é se a primeira camada é enrolada no sentido horário, a segunda será enrolada no sentido anti-horário e vice-versa. Isto faz com que os esforços sejam contrabalançados. A função desta camada é resistir aos esforços de tração.
- Capa Externa (G): Por último, uma capa externa, feita também de material polimérico, é aplicada de forma a garantir a estanqueidade e proteger o sistema como um todo da ação corrosiva da água do mar.

Algumas destas camadas podem aparecer ou não no produto final. Tubulações de injeção de água não requerem a carcaça; a armadura de pressão pode ser retirada se a profundidade de exploração for pequena. Neste último caso, o ângulo da trama das armaduras de tração pode ser aumentado, isto fará com que a estrutura suporte menos tração vertical (peso), mas terá mais resistência às forças de colapso.

As diversas combinações dessas variáveis podem produzir tubos (cada tipo de tubo é chamado de uma estrutura) com comprimentos da ordem de centenas de metros com diâmetros que podem chegar a um metro e com um peso total de centenas de toneladas. Apesar disso, as linhas (de tubo) são enroladas em carretéis para serem transportadas em navios e lançadas ao mar.



Figura 59 - Linha Flexível Enrolada

7.4.2 Qualificações de Linhas

Antes da instalação de uma linha flexível no campo, vários testes *full-scale*, com amostras da linha, podem ser solicitados de forma a garantir que as especificações fornecidas pelo fabricante foram realmente alcançadas. Estes testes são, na maioria das vezes, destrutivos e visam, também, identificar se a estrutura falha no ponto especificado, de forma a analisar se o conservadorismo empregado no projeto foi excessivo ou não.

7.4.2.1 Testes Dinâmicos

Testes Dinâmicos são utilizados para reproduzir, em larga escala, o comportamento de uma linha flexível no campo. São analisados, a priori, os dados oceanográficos

correspondentes ao local de instalação da linha flexível que são traduzidos em tabelas de “classes de ondas”. A amostra deverá ser então submetida a blocos dessas ondas por um período que representa a sua vida útil. Cada “onda” é representada como um ciclo de tração, que varia de uma carga mínima a uma carga máxima em uma senóide de determinado período. Como a vida útil de uma linha é em torno de 20 anos, testes dinâmicos costumam aplicar cerca de milhões de ciclos às amostras e ter um prazo de duração (contínuo) de três meses a um ano.

As bancadas nas quais os testes dinâmicos são realizados são estruturas pesadas, desenhadas para aplicarem cargas na faixa de 500 toneladas, através de atuadores hidráulicos e necessitam de um sistema de controle complexo de forma a garantir que os parâmetros aplicados durante a ciclagem se mantenham constantes e que todos os ciclos sejam registrados. Também são necessários sistemas de alarme e parada automática da bancada no caso de acidentes. Ou seja, o custo de realização de um teste é na faixa de milhões de reais, excluindo-se o custo do corpo-de-prova.

7.4.3 Pesquisa e Desenvolvimento

Por representar a vida útil de uma linha, e devido ao seu alto custo, um teste dinâmico é uma oportunidade única para o grupo de pesquisa e desenvolvimento observar o comportamento do produto, analisá-lo, corrigir falhas e/ou introduzir melhorias nas próximas gerações.

Uma das linhas de pesquisa é a detecção automática de falhas, que podem ser, entre outras, uma ruptura na capa externa, condensação de água nas camadas internas e a ruptura de arames das armaduras de tração. Esta última, é uma das falhas mais críticas, ao romper-se um arame de tração, o peso que este sustentava é dividido pelos demais, o que pode causar uma sobrecarga em outro arame, e assim por diante. A identificação automática destas rupturas em campo é de extrema importância para determinar o tempo de vida restante do *riser* e efetuar-se uma parada de manutenção

programada para a substituição do mesmo. Algumas das falhas, podem ser reparadas em campo, esta porém, requer a substituição do tubo inteiro, isto é, uma operação complexa realizada com o auxílio de navios especializado para tal, portanto, de alto custo.

Para este tipo de pesquisa, uma grande quantidade de sensores é aplicada no corpo-de-prova de forma a monitorar todos os aspectos possíveis do teste: sensores de temperatura, pressão, extensômetros (*strain-gauges*), inclinômetros, acelerômetros, células de carga (*load cells*), além de outros sistemas fornecidos por terceiros (emissão acústica, por exemplo) que são testados e cujos resultados são comparados uns com os outros para o aferimento dos resultados. A especificação dos sensores a serem utilizados pode ser consultada na Norma Técnica Corporativa N-2409 da Petrobras [83].

Por conseguinte, a quantidade de dados gerado em cada teste é da ordem de 300 a 500 Gigabytes de informação. Estes dados devem ser armazenados e meticulosamente analisados, durante e depois do teste. Durante o teste, um dos critérios de parada usualmente utilizado é a quebra de arames das armaduras de tração. A análise dos dados em tempo real é importante para a geração de alarmes que detectem uma situação de parada. Após o teste, os dados coletados são correlacionados de forma a se conhecer melhor o comportamento do produto e propor novas formas de sensoriamento.

7.5 Teste de uma Amostra

Para este estudo de caso, os dados coletados no teste de bancada de um *riser*, foi utilizado, tomando-se alguns cuidados para se preservar a confidencialidade da informação, um dos quais foi o adimensionamento dos dados, isto é, eles foram arbitrariamente alterados de forma a impedir que conclusões, que não as deste trabalho, sejam tiradas de sua apresentação.



Figura 60 - Plataforma de Produção

Em 2009 foi realizado um teste em uma amostra de tubo flexível. Este teste consistia em uma parte chamada “fase de vida”, na qual foram aplicados 1 milhão de ciclos de tração em cinco blocos de carregamentos, visando simular o tempo de vida projetado para a amostra; e uma parte chamada “fase de dano”, na qual foram aplicados mais 290 mil ciclos com o objetivo de efetivamente danificar a amostra, através de fadiga²⁶, nas armaduras de tração.

Esta estrutura – conjunto de parâmetros que constitui a linha – possui 52 arames na armadura de tração externa e 50 arames na armadura de tração interna. A amostra desta estrutura utilizada (corpo-de-prova) possui aproximadamente 10 metros, com os conectores que a ligam na bancada de teste e pesa em torno de 2 toneladas. São

²⁶ Pode-se entender “fadiga” como um efeito mecânico ao qual os diversos materiais estão sujeitos. Para maiores informações sugere-se consultar Callister [76],

aplicados diversos sensores no comprimento da amostra: inclinômetros, acelerômetros, pressão e temperatura; e diversos sensores em volta da amostra, extensômetros, um em cada fio de armadura externa. Cada um desses sensores deve ser calibrado e recebe um certificado de calibração correspondente. As unidades aquisitoras, também precisam ser certificadas e seu número de certificado aparece nas figuras. Toda esta documentação é anexada em um relatório e verificado por uma Autoridade Independente Verificadora (IVA) que testemunha o teste e verifica que os todos procedimentos foram seguidos corretamente.

Neste caso de teste, foram utilizados equipamentos de aquisição ADS 2000 fornecidos pela Lynx Tecnologia [77], uma empresa nacional. A escolha do equipamento a ser utilizado para aquisição é do laboratório onde o teste é realizado. O único requisito para a escolha é que ele atenda às necessidades especificadas pelo procedimento de teste. Podem existir casos em que equipamentos de mais de um fabricante são utilizados.

7.5.1 Workflow de Análise de Dados

Durante o teste, os dados foram capturados através do programa AqDados [78] (Figura 62), que roda em sistema operacional Windows, a uma taxa de 200Hz (200 amostras por segundo), e gravados em arquivos contendo uma hora de coleta. O formato de gravação destes arquivos é proprietário da Lynx, utilizando uma extensão “.LTD” para discriminá-los. Cada arquivo armazena 63 canais de dados, em uma hora, são coletadas 720 mil amostras. O tamanho de cada arquivo é de aproximadamente 90 Mbytes. O teste teve uma duração aproximada de 1000 horas, gerando uma massa de dados em torno de 300 Gbytes.

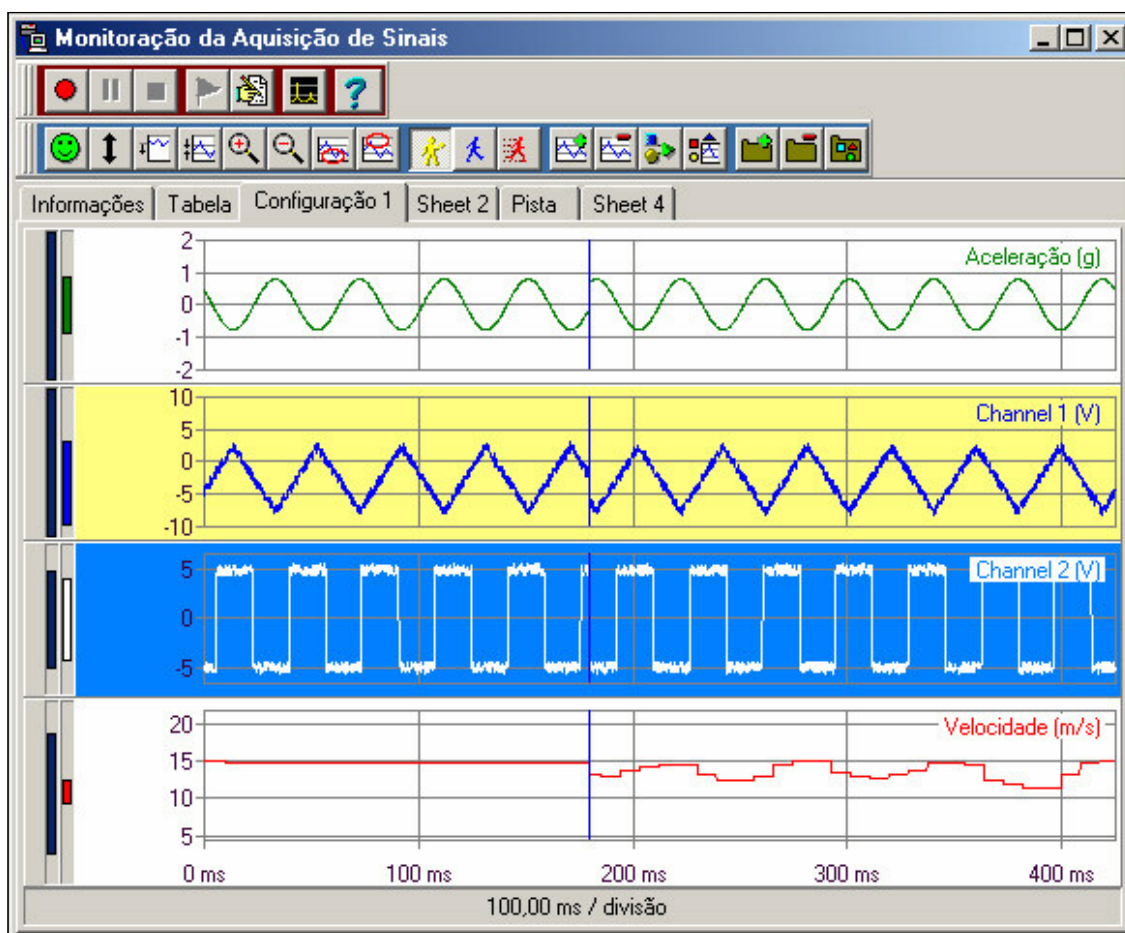


Figura 61 - Tela do Programa AqDados - (Extraída do sítio do fabricante)

Os dados eram analisados com o software AqDAnalysis [78], que permite a visualização das séries temporais. A Figura 62, apresenta um exemplo da utilização deste software. Nela é mostrado o comportamento do sinal de um extensômetro durante a ciclagem e após o rompimento do fio de armadura ao qual estava colado.

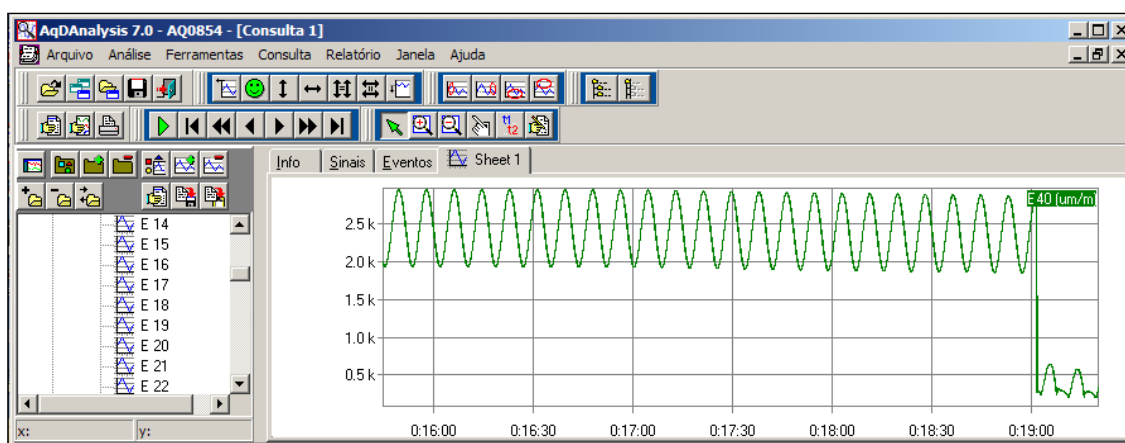


Figura 62 - Tela do software AqDAnalysis

7.5.1.1 Ferramentas de Trabalho

O programa AqDAnalysis permite que se realizem algumas análises numéricas sobre os dados coletados. Algumas destas funcionalidades, como de análise de *Rainflow*²⁷, são vendidas separadamente e habilitadas através de uma chave USB²⁸. O programa permite que se concatene séries temporais, porém, o tamanho máximo do arquivo analisado depende da memória principal da estação de trabalho, portanto, a concatenação de arquivos é apenas viável para 3 ou 4 horas de coleta de dados.

A Lynx Tecnologia dispõe de um pequeno programa que permite apenas a visualização destas séries temporais e a sua exportação para arquivos do tipo CSV, em formato texto. Vale lembrar que um arquivo texto gerado a partir de um desses

²⁷ Um tipo de análise numérica muito utilizada em cálculos de fadiga

²⁸ Dispositivo de proteção de software, fornecido pelo fabricante com cada cópia, que se conecta na entrada USB do computador.

arquivos de 90Mb possui em torno de 300Mb. Também estão disponíveis para *download* algumas rotinas para MatLab que importam estes arquivos diretamente, sem a necessidade de conversão para texto.

7.5.2 Informações Relevantes

Dentre as informações relevantes que estes arquivos contém, duas podem ser ressaltadas: A **contagem de ciclos** e a **ruptura de arames das armaduras**. A contagem de ciclos é necessária para se confirmar que o procedimento de teste foi seguido corretamente, um fator importante no cálculo de fadiga da estrutura.

7.5.2.1 Identificação de Rupturas de Arames

Durante o teste, podem ocorrer rupturas dos arames das duas armaduras do tubo flexível: armadura interna e armadura externa. Na maioria dos testes, só é possível a instrumentação da armadura externa do flexível. Isto porque ela é a camada logo abaixo da capa externa, e portanto acessível, abrindo-se a capa. Para acessar a armadura interna é necessário retirar a externa ! E em alguns casos em que se consegue acesso a esta armadura (pelas pontas do tubo), porém, os cabos que conectam os sensores interferem no teste que está sendo realizado. Esta interferência deve ser minimizada sempre que possível.

A identificação de uma ruptura é uma condição de parada de um teste dinâmico, portanto devem ser reportadas o mais rápido o possível.

Com o *workflow* existente, a identificação das rupturas externas é um trabalho repetitivo e lento, porém possível de ser levado a cabo. Consiste em abrir cada arquivo gerado (cada hora) e examinar cada canal de extensômetro por situações semelhantes à Figura 62.

Por outro lado, a ruptura de um arame na camada interna, pela falta de instrumentação direta, não apresenta uma “assinatura” visível de sua ocorrência.

7.5.3 Análise Holística dos Dados

Devido a falta de uma “assinatura” que pode ser identificada localmente, o problema da ruptura na(s) camada(s) de armaduras interna(s) requer uma análise holística (como um todo) dos dados. As séries temporais de cada canal precisam ser analisadas como um todo e por um grande período de tempo.

Como já foi dito anteriormente, as ferramentas existentes permitem a concatenação das séries temporais, porém, é necessário também fazer uma re-amostragem dos dados de modo a diminuir a taxa de coleta para que a série final caiba na memória principal. Nestes casos, é imprescindível utilizar uma ferramenta que permita a manipulação de grande quantidade de dados de forma “transparente” (*seamlessly*). O Matlab, foi a ferramenta escolhida para tal.

A vantagem do uso do Matlab está na sua linguagem de programação/utilização. Esta linguagem permite a manipulação de matrizes como um todo, postergando a necessidade de criação de rotinas procedurais para esta manipulação. Apesar disto, estas análises ainda requisitaram a criação de um conjunto de rotinas procedurais que auxiliaram na manipulação dos dados.

Consideremos, como exemplo, uma análise de 10 horas consecutivas de teste. Para tal, 10 arquivos devem ser concatenados e exportados para o Matlab para pós-processamento. Caso se deseje adicionar a próxima hora a análise, é necessário concatenar-se novamente os 9 últimos arquivos, concatenar o arquivo novo e repetir-se a operação.

Consideremos, também, que pode ser necessário submeter os dados a algum tipo de pré-processamento com um filtro digital que utilize uma janela de tamanho k . Supondo que cada arquivo possui n amostras, o resultado da aplicação do filtro em um arquivo

seria de $n-k$ amostras significativas (ver seção 2.2.2), se executarmos esta operação localmente, um arquivo por vez, para j arquivos, e depois concatenássemos cada resultado intermediário, o produto final teria $jn - jk$ amostras significativas. Se executarmos a mesma operação após a concatenação, o produto final teria $jn - k$ amostras significativas. Ou seja, perde-se menos.

A partir deste exemplo, percebemos a falta de um banco de dados de onde possa-se extrair estes dados de forma transparente, através de uma linguagem não procedural.

7.5.4 Hiato entre Engenharia e TI

Em visita ao Brasil, o Prof. C. Mohan [79], pesquisador da IBM, proferiu uma palestra na Coppe, no dia 07/10/2011. O objetivo da palestra foi apresentar a visão de futuro da computação por parte da IBM e nela, foi comentada a existência de um hiato (*gap*) entre a Engenharia e a Tecnologia de Informação. Esse hiato pode ser visto como o "impedance mismatch" que foi descrito na definição do problema desta tese.

Este caso pode ser entendido como um exemplo típico deste hiato. De certa forma, os fabricantes de equipamentos e software de aquisição e análise de dados não são capazes de fornecer ferramentas que ultrapassem o escopo de seu próprio negócio (sensoriamento e instrumentação). Por outro lado, o profissional de TI não está acostumado a lidar com os problemas de engenharia, até mesmo porque o foco de sua formação está em análise de sistemas e de modelos de negócio. Por analogia, pode-se dizer que o sensoriamento e instrumentação são realizados com softwares específicos que "só tratam" das séries temporais e o ferramental de TI "só trata" dos dados clássicos de SGBDs relacionais. A integração desses dois mundos é um primeiro passo para diminuir esse hiato.

7.6 Aplicação da Proposta ao Problema

De forma a mostrar a funcionalidade prática dos operadores propostos neste trabalho, parte dos dados coletados no teste da estrutura utilizada foram carregados na ferramenta desenvolvida com o sistema SECONDO.

7.6.1 Elementos do Estudo de Caso

O desenvolvimento do protótipo foi realizado em um computador DELL Vostro 230, equipado com um processador Intel Celeron 450 de 2,20 Ghz, com 1Gb de memória RAM. Um segundo disco rígido interno de 400Gb, foi adicionado ao sistema para comportar o espaço do Banco de Dados. O sistema operacional utilizado foi o Linux, distribuição Ubuntu 10.10 de 32 bits.

Como pode-se notar, esta configuração está longe de ser a mais adequada para um Servidor de Banco de Dados, mas mostrou-se suficiente para atender à proposta de desenvolvimento, prototipação e testes.

A massa de dados escolhida para o teste corresponde a um bloco de carregamento aplicado na estrutura durante o teste que simula o comportamento do tubo flexível após o seu tempo de vida projetado. Nesta fase do teste, é necessário que ocorram rupturas nos arames de modo a provar que a estrutura não foi superdimensionada para sua aplicação.

Esta massa corresponde a 83 horas de captura de dados, a uma taxa de 200Hz, ou seja, aproximadamente 60 Milhões de linhas (amostras) para 63 colunas (canais).

7.6.2 Objetivo

O objetivo deste teste é mostrar que a utilização dos operadores propostos permite que se extraiam informações a partir dos dados brutos, que não poderiam ser extraídas através das ferramentas disponíveis, tanto na parte de análise de dados, como também na parte de Banco de Dados.

7.6.3 Escopo e Limitações

Devido à configuração do sistema, os tempos de processamento desta amostra tão grande foi na faixa de horas. Como o objetivo do teste é mostrar a simplicidade de uso dos operadores, não foram feitos *benchmarks* para a análise da performance do sistema. Mesmo porque os operadores foram implementados utilizando o sistema SECONDO, cuja função é ser uma plataforma acadêmica de pesquisa, em conjunto com o Berkeley DB, um SGBD que, apesar de eficaz, é mantido pela Oracle (que comprou o projeto) como uma solução acadêmica de código aberto e não como um produto de linha.

Para a carga dos dados no sistema, foram necessárias algumas alterações no código fonte dos operadores, pois a própria versão do SECONDO foi atualizada de 2.6 para a 3.2, pois a versão anterior apresentava falhas que não permitiam a importação de uma massa de dados tão grande.

A versão atual do SECONDO ainda possui algumas limitações em relação à alocação e liberação dos FLOBs, um conceito de armazenamento utilizado pelo sistema no qual o tipo `RealArray` se baseia. Isto impactou diretamente a utilização do operador `Iwindow` com uma grande escala de dados, devido ao desempenho.

O escopo do experimento foi demonstrar a possibilidade de extrair novas informações sobre o comportamento do tubo flexível que não eram acessíveis anteriormente pela falta de ferramentas adequadas. Neste escopo, a identificação de rupturas passa a ser o principal foco.

7.6.4 Importação dos Dados

A conversão dos dados adquiridos pelo sistema da Lynx (arquivos “.LTD”) para um formato texto mostrou-se inviável. Cada arquivo gerado possuía em torno de 300Mb de tamanho e o tempo de importação destes dados também era grande.

A solução foi a criação de um novo operador na álgebra do SECONDO que lesse os arquivos diretamente de seu formato nativo (binário) e os convertesse em uma *stream* de tuplas do tipo real, de modo a inserí-los em uma relação na base de dados. A sintaxe deste operador (`LTDStream`²⁹) pode ser vista abaixo:

```
query LTDStream("/media/Data/BL/XXX-YY-BL-0001.LTD", B7, 0, -1) B7
        insert count;
```

Este comando lê o arquivo “XXX-YY-BL-0001.LTD”, que apresenta o formato da tabela B7, a partir da linha 0, até o final do arquivo (-1). Isto gera uma *stream*, que é inserida na tabela B7 pelo comando `insert` e, ao final, o número de linhas inseridas é contado.

Ao final da importação dos dados brutos, a base de dados resultante ficou com aproximadamente 50Gb. Esta discrepância de tamanho em comparação com o espaço tomado pelos arquivos no formato original (9Gb) é devido à forma de como o tipo de ponto flutuante é armazenado (vide seções 2.2.3 e A.2.1)

7.6.5 Manipulação dos Dados

O primeiro teste realizado foi a execução de uma média móvel utilizando-se o operador `Iwindow`. A ideia por trás deste teste seria de filtrar os dados brutos. A janela escolhida foi de 1666 amostras. Este número é correspondente ao tempo do período de um ciclo de teste (8.33 segundos), amostrado a 200 Hz.

²⁹ Este operador foi criado exclusivamente em função deste teste. Vários quesitos de robustez de código foram relaxados em função do pragmatismo.

O comando consistia em se extrair as janelas móveis (tipo `RealArray`), processar cada janela tirando-se a média (tipo `real`), e, por fim, gravar as médias. A cardinalidade da tabela final seria de $n - k$, onde n seria o tamanho inicial e k o tamanho da janela. A estrutura `RealArray` criada à cada iteração do processo seria liberada ao fim da mesma, logo o espaço em disco da tabela final seria semelhante ao da tabela inicial.

Essa operação foi bem sucedida no que diz respeito à especificação da consulta em alto nível permite a escolha de amostras, janelas móveis e cálculo da média em uma única expressão. Infelizmente, do ponto de vista de desempenho, a fragilidade do protótipo não permitiu a conclusão da execução do teste. O espaço em disco requerido pelo `SECONDO` esgotava-se no meio do processamento, devido às estruturas temporárias em memória secundária que não eram liberadas. Não foi possível contornar este problema.

Porém, trata-se de um problema da plataforma `SECONDO`, e não um problema da álgebra em si. Muito menos de sistemas reais, produtos que estão preparados para maiores volumes de dados. Foi possível entretanto, validar a execução dessa operação, ao verificar que funciona corretamente com um subconjunto dos dados (aproximadamente 100.000 linhas).

Partiu-se, então para um teste mais simples. Desta vez a média móvel foi descartada e foi tirada a média de cada ciclo. O comando desta operação pode ser visto na Listagem 3, abaixo:

```

let RB7 = B7 feed

Iuncolumn[ts,1666]

projectextend[

ts;

me1  : .e1  Havg, me2  : .e2  Havg, me3  : .e3  Havg,
me4  : .e4  Havg, me5  : .e5  Havg, me6  : .e6  Havg,
me7  : .e7  Havg, me8  : .e8  Havg, me9  : .e9  Havg,
me10 : .e10 Havg, me11 : .e11 Havg, me12 : .e12 Havg,
me13 : .e13 Havg, me14 : .e14 Havg, me15 : .e15 Havg,
me16 : .e16 Havg, me17 : .e17 Havg, me18 : .e18 Havg,
me19 : .e19 Havg, me20 : .e20 Havg, me21 : .e21 Havg,
me22 : .e22 Havg, me23 : .e23 Havg, me24 : .e24 Havg,
me25 : .e25 Havg, me26 : .e26 Havg, me27 : .e27 Havg,
me28 : .e28 Havg, me29 : .e29 Havg, me30 : .e30 Havg,
me31 : .e31 Havg, me32 : .e32 Havg, me33 : .e33 Havg,
me34 : .e34 Havg, me35 : .e35 Havg, me36 : .e36 Havg,
me37 : .e37 Havg, me38 : .e38 Havg, me39 : .e39 Havg,
me40 : .e40 Havg, me41 : .e41 Havg, me42 : .e42 Havg,
me43 : .e43 Havg, me44 : .e44 Havg, me45 : .e45 Havg,
me46 : .e46 Havg, me47 : .e47 Havg, me48 : .e48 Havg,
me49 : .e49 Havg, me50 : .e50 Havg, me51 : .e51 Havg,
me52 : .e52 Havg]

consume;

```

Listagem 3 - Comando utilizado para calcular a média do período de cada ciclo

O funcionamento do trecho :

```
Let RB7 = B7 feed Iuncolumn[ts,1666] ... consume;
```

Produz uma tabela (RB7) contendo `RealArrays` a partir das colunas de tipo real da tabela B7. O que toma grande parte do comando é a utilização do operador `projectextend`, padrão da álgebra relacional do SECONDO, este operador realiza a projeção de uma tabela e ao mesmo tempo a estende com atributos novos. No caso a coluna `ts`, foi projetada e, para cada coluna que representava um extensômetro (`e1`, `e2`, ..., `e52`) foi calculada a **média horizontal** do `RealArray` com o operador `Havg` (seção 6.1.3), cada novo atributo recebeu um nome correspondente: `me1`, `me2`, ..., `me52`. Ou seja, para cada novo atributo existe uma expressão “`mei : .ei Havg`” na lista de operandos do operador `projectextend`³⁰.

7.6.6 Exportação dos Dados

Ao final da manipulação, a tabela RB7 continha 36 mil linhas, um tamanho razoável para se operar com o Matlab. Ela foi exportada para formato texto (.CSV) e importada pelo Matlab para a geração de um gráfico, que pode ser visto na Figura 63

7.6.7 Reanálise dos Dados

Neste gráfico (Figura 63), podemos visualizar dois eventos claros, que indicam a ruptura dos fios 40 e 15, mas o mais interessante é que podem ser notados eventos que seriam “invisíveis” a partir de análises locais com os métodos clássicos. O primeiro, um possível evento de ruptura na Armadura de Tração Interna, pode ser notado por um “desbalanceamento” das tensões nos fios, em que um lado torna-se um pouco mais “verde”; o segundo, o possível início de trinca que culminou na ruptura do fio 15, pode ser notado por um sulco contínuo que culmina na quebra do fio.

³⁰ O projeto de um operador que sirva como *Syntactic Sugar* fica como proposta para trabalhos futuros.

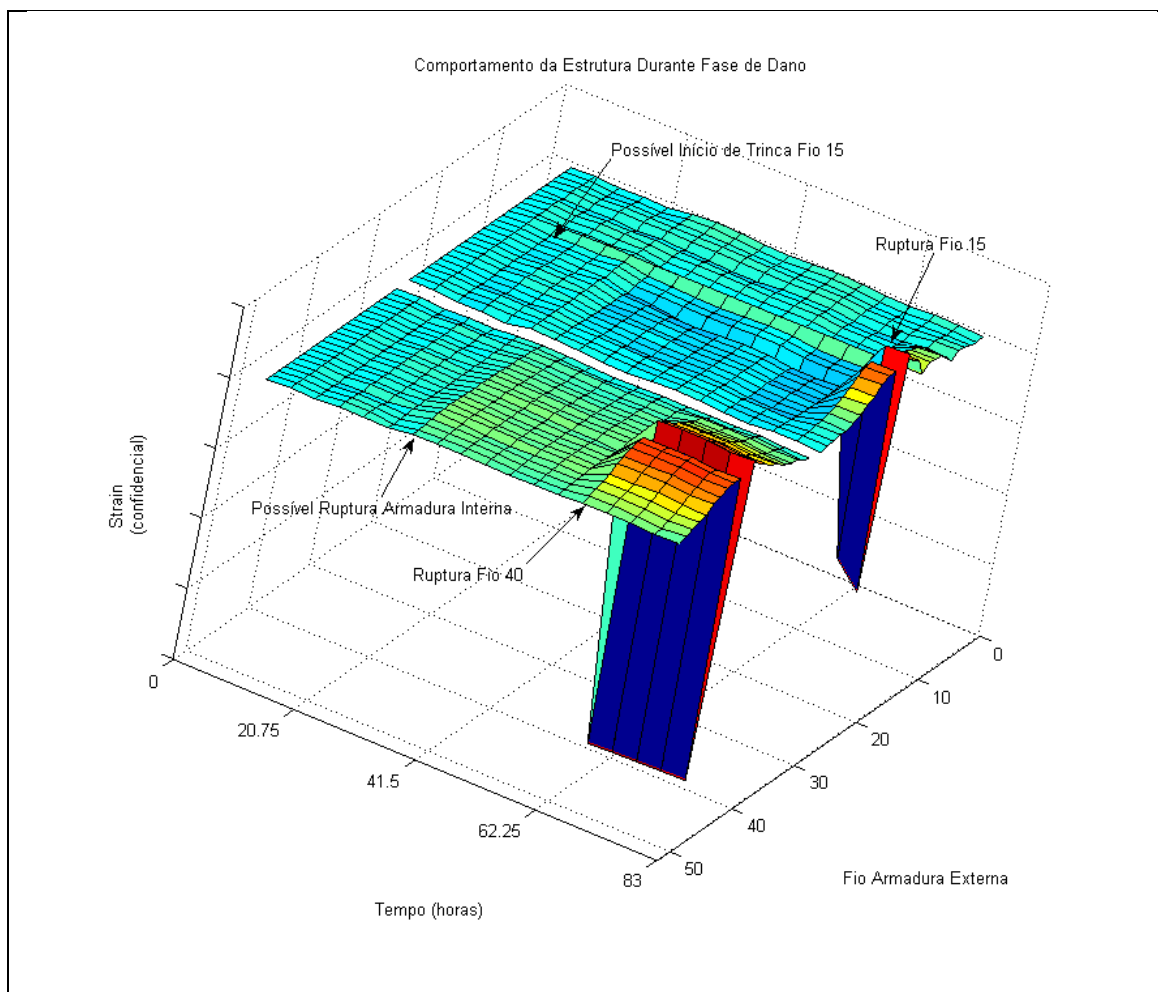


Figura 63 – Visualização das Tensões na Fase de Dano no Matlab

7.6.8 Operações com o modelo relacional

Uma das facilidades apresentadas pela ferramenta é a possibilidade de se mesclar consultas típicas de tecnologia da informação com os operadores trans-relacionais. Esta característica permite que o acesso aos dados de séries temporais seja feito de forma homogênea, sem a necessidade de utilização de sistemas em separado, como um SGBD e o Matlab.

Consideremos como exemplo, as anotações da figura 63. Elas poderiam fazer parte de uma tabela conforme a mostrada na Figura 64.

Program Server Optimizer Command Help Viewers

Sec>query EVENTS
 query EVENTS...successful
 see result in object list
 Sec>

query EVENTS_2

evnt	desc	period	wbreak
1	Inicio do Teste	("2009-05-25-13:28:31" "2009-05-25-13:28:32")	FALSE
2	Evento Fio 15	("2009-05-26-13:48" "2009-05-26-13:49")	FALSE
3	Evento Fio Interno	("2009-05-27-02:32" "2009-05-27-02:33")	TRUE
4	Quebra Fio 40	("2009-06-02-09:13" "2009-06-02-09:14")	TRUE
5	Quebra Fio 15	("2009-06-05-23:46" "2009-06-05-23:47")	TRUE
6	Fim do Teste	("2009-06-07-12:00" "2009-06-07-12:01")	FALSE

export

Figura 64 - Tabela de Eventos

Esta tabela está normalizada conforme o Modelo Relacional. Os campos são:

Evnt: número sequencial do evento;

Desc: descrição do evento;

Period: O momento em que o evento ocorreu, utilizando-se o tipo Tspan, proposto neste trabalho;

Wbreak: um campo booleano que especifica se o evento refere-se a uma quebra de fio.

Uma vez que a tabela que contém os dados do experimento esteja indexada pelo campo *ts* (*time stamp*) em uma estrutura RTree através do comando:

```
let RTree = B7 creatertree[ts];
```

a consulta a seguir poderia ser realizada.:

```
query EVENTS feed filter[.wbreak = TRUE]
loopjoin [ RTree B7 windowintersects[.period] ]
project [ts,e1, e2]
Iuncolumn[ts,200]
consume;
```

A compreensão imediata da sintaxe desta consulta exige uma certa proficiência com o sistema `SECONDO`, porém os diversos operadores podem ser analisados separadamente:

A primeira operação a ser realizada é a consulta à tabela `EVENTS` (comando `query`), transformando-a em um *stream* (operador `feed`) e filtrando-a de modo a selecionar apenas as linhas que contenham eventos com quebra de arames (`filter[w.break = TRUE]`).

O *stream* resultante é então repassado ao operador `loopjoin`, que realiza a operação relacional de *Join*, entre dois operandos do tipo `stream(tuples)`. O primeiro operando segue diretamente da primeira consulta, o segundo operando é recuperado através da operação `RTree B7 windowintersects[.period]`, que recupera linhas de uma tabela através de um índice (ver seção 6.4.4).

A próxima operação a ser efetuada é uma projeção do resultado. No caso apenas os campos `ts`, `e1` e `e2`, foram escolhidos de modo a simplificar o exemplo. Por fim uma operação Trans-Relacional `Icolumn` é realizada com o resultado e toda a consulta é finalizada pelo operador `consume`.

7.6.9 Conclusão

A utilização da ferramenta proposta permitiu a visualização de novos resultados a partir dos dados brutos usando o ferramental integrado. O resultado mostrado na Figura 63 não poderia ser criado, de forma sistêmica, com os trabalhos relacionados existentes no atual estado do arte. A partir da álgebra integrada, interfaces de mais alto nível de interação podem ser construídas para facilitar o acesso. A possibilidade de se manipular dados de forma simples e sistêmica provou-se fundamental para que se possa entender um fenômeno físico como um todo e não só através de pequenas parcelas não integradas naturalmente.

8 Conclusão

8.1 *Contribuição deste Trabalho*

Espera-se que esta tese tenha contribuído, não só para melhorar o entendimento dos problemas enfrentados ao se trabalhar com séries temporais, como também validado que as propostas de operadores apresentadas realmente atendem tanto às necessidades de um possível usuário final como também podem se acoplar em soluções (SGBDs) existentes, produzindo resultados corretos e completos.

8.2 *Recapitulamento*

Podemos dizer que este trabalho pode ser dividido em três partes: Uma primeira parte, composta pelos capítulos 2 e 3 que explanam a utilização de séries temporais, de sua origem, geração e aquisição à forma de armazenamento, seja em SGBDs ou não, no formato de arquivos binários ou texto. Na segunda parte, apresentada no capítulo 4, faz-se uma revisão do estado-da-arte de desenvolvimento ligado ao problema e do ferramental disponível que poderia resolver o problema, tanto através de ferramentas comerciais, fechadas, como também de ferramentas mais genéricas, abertas ou acadêmicas nas quais podem ser desenvolvidas propostas de forma a minimizar o problema estudado. Por fim, nos capítulos 5,6,7 e 8, é exposta uma proposta de solução através da criação de novos operadores que se agregam à Álgebra Relacional em uma plataforma de desenvolvimento acadêmico aberta (SECONDO). Estes capítulos são ordenados de forma construtivista, começando pela apresentação da plataforma de desenvolvimento escolhida; definição formal teórica dos operadores e tipos; implementação prática dos operadores propostos e, por fim, a apresentação de

um estudo de caso utilizando estes operadores e mostrando que na prática os mesmos podem ser efetivamente utilizados.

8.3 Escopo Final

Trata-se de um trabalho de fundamentação sobre a análise e modelagem do que vem a ser uma série-temporal, bem como uma proposta de uma série de operadores que podem ser agregados de forma consistente à Álgebra Relacional. Estes operadores são chamados de Trans-Relacionais, quando operam quebrando o paradigma relacional de que não existem relações entre as diversas linhas de uma tabela; e de Trans-Relacionais Semânticos, quando os mesmos aplicam uma Semântica Temporal a estas relações.

A completude e correteza destes operadores foi validada por meio do Secondo, uma plataforma construída sobre um arcabouço formal de Assinaturas de Segunda Ordem, voltada para propiciar extensões à álgebra relacional. O ferramental do Secondo verificou a compatibilidade entre as estruturas e os operadores da álgebra relacional e a extensão realizada com os operadores propostos nesta tese. Além da verificação teórica, o ferramental do Secondo permitiu a efetivação dos novos operadores, por meio de um protótipo de sistema de banco de dados relacionais. A implementação realizada no protótipo permitiu a validação prática do novo conjunto integrado de estruturas e operadores. Experimentos, realizados no estudo de caso real, mostraram, de fato, que o comportamento dos mesmos corresponde às expectativas em se trabalhar com dados de séries temporais, ao mesmo tempo em que se opera com dados textuais operados com a álgebra relacional clássica.

Como o sistema foi montado sobre uma plataforma desenvolvida com intuito acadêmico, e por tratar-se de um protótipo, não foi realizado um estudo de

desempenho pois isto esbarraria nas limitações da plataforma SECONDO, sobre as quais não podem ser feitas garantias a este respeito.

Com os resultados obtidos, pode-se dizer que os operadores propostos conseguem inserir as ferramentas necessárias para análises de dados de Séries Temporais em um contexto único, de forma consistente e sistêmica, sem que o usuário precise alternar entre sistemas e linguagens diferentes para que atinja os seus objetivos. Em resumo, o usuário está finalmente apto a fazer o “*cooking*” de seus dados em uma só plataforma.

8.4 Temas Futuros

Uma vez terminado o desenvolvimento deste trabalho, podemos citar alguns temas que merecem atenção especial e poderiam ser parte de futuros trabalhos, são eles: Otimizar (no sentido de maximizar) a precisão do resultado dos operadores algébricos; Permitir a configuração de campos com incertezas diferentes (ver seção 2.2.4); Estender a linguagem SQL de modo a incluir os operadores propostos e a inclusão de uma modelagem hierárquica para representar o experimento.

8.4.1 Otimização da Precisão dos Resultados Algébricos

Este assunto, apesar de tema da tese, não foi abordado na implementação. Isto porque não é um problema a ser resolvido. Existe literatura suficiente (Parhami [2], por exemplo) na área de Aritmética Computacional, juntamente com exemplos disponíveis na Web ([81]), já codificados, que permitem a aplicação imediata da teoria em SGBDs. O que parece faltar é uma conscientização por parte dos desenvolvedores, para a existência do problema em si. Os operadores aritméticos implementados neste trabalho (`sum`, `Hsum`, `avg`, `Havg`), poderiam ter respeitado estas premissas. Isto não foi feito pois como o protótipo se tratava de uma aplicação em uma plataforma na qual não havia experiência prévia, houve um certo receio em se complicar o algoritmo caso surgissem necessidades de depuração.

O desenvolvimento deste tema se adequa a um estudo de Iniciação Científica e, permitiria a um aluno, um contato inicial com a plataforma SECONDO de modo a estender o que foi desenvolvido.

8.4.2 Permitir a Configuração de Campos com Incertezas Diferentes

Este caso é mais complexo do que se poderia esperar. O primeiro aspecto a ser pesquisado seria também o de Aritmética Computacional ligado a questões de Metrologia. Ou seja, como operandos com incertezas diferentes devem, efetivamente, ser operados. Porém, o aspecto mais complicado, do ponto de vista de Banco de Dados, é como o dado deve ser modelado. Deve ser criado um novo tipo que permita ser associado a uma incerteza ? Isto não deveria ser caracterizado como um metadado (ver seção 3.3.3) ? Neste último caso, como seria a modelagem, operação e gerência destes metadados ?

Pode-se ver que este tema poderia gerar um trabalho exclusivo para tal. Considerando-se a plataforma SECONDO, escolhida para a implementação. Pode-se dizer que seria possível a criação de um outro tipo de dado (algo como `uncertain real`) que permitisse a operação com incertezas diferentes, porém, não é possível dizer o mesmo da criação de uma estrutura de metadados para abordar este problema, sem que se redefina toda a álgebra já existente para relações e tuplas.

8.4.3 Modelagem Hierárquica do Experimento

Através dos exemplos práticos, podemos perceber a falta que uma modelagem hierárquica do experimento, proposta no exame de qualificação, faz no momento de consulta aos dados. Podemos observar, na listagem 3 e na listagem 4, como uma mesma operação precisa ser explicitamente declarada para vários operandos, gerando consultas de sintaxe extensa.

Um dos problemas que este enfoque traria a implementação é a transposição do Modelo Hierárquico para a estrutura de Assinaturas de Segunda Ordem utilizada pelo SECONDO. Esta transposição não seria trivial, requerendo a alteração da álgebra

relacional previamente implementada na plataforma ou programando-se uma álgebra hierárquica a partir do zero.

Uma solução mais simples seria a possibilidade de utilização de um sistema de macro-substituição parametrizada, como o disponível no pré-processador da linguagem C. Apesar de não ser uma solução direta para o problema, isto proveria ao usuário o chamado *syntactic sugar*³¹, que facilita o processo de consultas. Este sistema, porém, também não está disponível na plataforma SECONDO e precisaria também ser desenvolvido.

8.4.4 Extensão da Linguagem SQL

O sistema SECONDO já permite o acesso às suas bases através de um subconjunto da linguagem SQL. Este subconjunto é implementado em seu módulo de otimização de consultas. Este módulo foi desenvolvido em PROLOG e se conecta ao interpretador de comandos. Ao se executar uma consulta no otimizador utilizando-se a linguagem SQL, a princípio é feito o *parsing* do comando de consulta em si, e em seguida são criados os diversos planos para acesso aos dados. A geração destes planos toma como base o Catálogo do sistema, que contém informações das tabelas como número de linhas, tamanho de cada bloco, tempo de acesso, entre outros. Em alguns casos são realizados testes com um número pequeno de linhas para se determinar qual estratégia apresentaria melhores resultados.

Uma extensão da linguagem SQL para se contemplar os novos operadores passaria por três etapas: a primeira, de modelagem, especificaria uma nova sintaxe para os operadores compatível com SQL; a segunda etapa, programação, incluiria no

³¹ uma tradução seria “açúcar sintático”, trata-se de jargão de programação que indica que uma linguagem de programação possui uma forma mais simples de se representar processos repetitivos, algo como $i++ \Leftrightarrow i = i + 1$

processo de *parsing* o reconhecimento destes novos operadores; e, por fim, em uma terceira etapa, seria implementado o processo de otimização de consultas com estes novos operadores.

Destas três etapas, pode-se dizer que a segunda (*parsing*) é a mais simples. Trata-se de traduzir partes de um comando em cláusulas PROLOG que produzirão comandos a serem interpretados pelo sistema SECONDO. Esta tarefa é um trabalho prático de Compiladores que pode ser resolvido por alunos de graduação, desde que devidamente orientados.

A etapa de modelagem dos operadores ainda requer maior pesquisa sobre a linguagem SQL em si, sua evolução e características, ou seja, procurar qual seria a melhor forma de encaixá-los em um contexto. Em contraste, outra opção seria uma abordagem *ad-hoc*, no qual cada operador seria diretamente modelado de forma a espelhar sua implementação na álgebra do SECONDO.

A escolha da abordagem *ad-hoc*, no entanto, dificultaria a questão de otimização. Levando-se em consideração o exemplo prático do estudo de caso, mostrado na listagem 3, podemos fazer uma outra consulta, que produz os mesmos resultados, utilizando-se o comando apresentado na listagem 4, abaixo.

```

let TRB7 = B7 feed Itranspose [
e1,  e2,  e3,  e4,  e5,  e6,  e7,  e8,  e9, e10,
e11, e12, e13, e14, e15, e16, e17, e18, e19, e20,
e21, e22, e23, e24, e25, e26, e27, e28, e29, e30,
e31, e32, e33, e34, e35, e36, e37, e38, e39, e40,
e41, e42, e43, e44, e45, e46, e47, e48, e49, e50,
e51, e52]

addcounter [Cnt,1]

projectextend [ transposed; Grp: .Cnt div 1666]

groupby [Grp; MTransposed: group feed avg[transposed]]

project [MTransposed]

Iuntranspose[MTransposed,
me1,  me2,  me3,  me4,  me5,  me6,  me7,  me8,  me9, me10,
me11, me12, me13, me14, me15, me16, me17, me18, me19, me20,
me21, me22, me23, me24, me25, me26, me27, me28, me29, me30,
me31, me32, me33, me34, me35, me36, me37, me38, me39, me40,
me41, me42, me43, me44, me45, me46, me47, me48, me49, me50,
me51, me52]

consume;

```

Listagem 4 - Comando alternativo para calcular a média do período de cada ciclo

Neste comando, foi utilizado o operador `transpose` para transformar cada linha em um único `RealArray`, foi incluído um contador de linhas:

```
addcounter[Cnt, 1]
```

Este contador foi transformado em um grupo, baseado no resultado de sua divisão por 1666:

```
projectextend [ transposed; Grp: .Cnt div 1666]
```

Em seguida foi utilizado o comando `groupby` da álgebra relacional do SECONDO para produzir uma média vertical destes `RealArrays` (`avg`), quebrando-se pelo grupo (`Grp`) criado:

```
groupby [Grp; MTransposed: group feed avg[transposed]]
```

Por último, os resultados foram re-transpostos para a configuração inicial através do comando `untranspose`.³²

Dada esta situação, em que duas estratégias de consulta podem produzir o mesmo resultado, seria interessante deixar esta escolha para o Banco de Dados ao invés do usuário, pois o SGBD possui as informações necessárias para uma melhor decisão de qual estratégia utilizar. Portanto, faz sentido uma pesquisa maior sobre a utilização final dos dados de modo a produzir uma transcrição mais genérica dos operadores algébricos para a linguagem de consulta de modo a permitir uma melhor otimização.

8.5 Incorporação da álgebra relacional estendida em outros SGBDs

A plataforma SECONDO foi fundamental para a validação dos operandos e operadores propostos. Entretanto, trata-se de um ferramental de validação e análise da solução e não um ambiente de produção. Assim, com o a álgebra validada, torna-se possível pensar em uma proposta para desenvolvimento futuro, como por exemplo, agregar os operadores desenvolvidos a um outro SGBD, excluindo-se a interface do sistema SECONDO. Por exemplo: criar uma extensão com estes operadores diretamente no sistema PostgreSQL [68], que permite extensões. Outra opção seria a de se utilizar um dos novos SGBDs, por exemplo, orientado a colunas. Neste caso,

³² Esta operação, não obteve sucesso com a tabela contendo os dados completos do estudo de caso, provavelmente pelos mesmos motivos do problema do operador `Iwindow`, porém a consulta funcionou com um subconjunto dos dados.

uma sugestão seria uma maior análise do Monet DB [65], que começa a se voltar para a área científica.

Referências

- [1] Goldberg, D., "What Every Computer Scientist Should Know About Floating-Point Arithmetic", March, 1991 Issue of Computing Surveys, Association for Computing Machinery, 1991;
- [2] Parhami, B. "Computer Arithmetic : Algorithms And Hardware Designs", Oxford University Press, New York, 2000;
- [3] Szalay, A. S. , Blakely J. A. "Gray's Laws: Database-centric Computing in Science" em "The Fourth Paradigm: Data-Intensive Scientific Discovery", pp.5-10, Hey, T., Tansley, S., Tolle, K. eds., Microsoft Research, Redmond, 2009;
- [4] "B.5.5.8 Problems With Floating Point Comparisons", MySQL 5.0 Reference Manual, disponível em <<http://dev.mysql.com/doc/refman/5.0/en/problems-with-float.html>>, último acesso 31/10/2009;
- [5] "10.1.1 Overview of Numeric Types", MySQL 5.0 Reference Manual, disponível em <<http://dev.mysql.com/doc/refman/5.0/en/numeric-type-overview.html>>, último acesso 31/10/2009;
- [6] Kossman, D. "The State of the Art in Distributed Query Processing", ACM Computing Surveys (CSUR), Vol. 32, issue 4, pp 422-469, ACM, New York, 2000;
- [7] Graefe, G., "The Five-Minute Rule 20 Years Later (and How Flash Memories Changes the Rules)", Communications of the ACM, Vol. 52, No. 7, p.48, ACM, New York, July 2009;
- [8] Gannon, D., Reed, D., "Parallelism and the Cloud" em "The Fourth Paradigm: Data-Intensive Scientific Discovery", pp. 131-135, Hey, T., Tansley, S., Tolle, K. eds., Microsoft Research, Redmond, 2009;
- [9] Larus, J., Gannon, D., "Multicore Computing and Scientific Discovery" em "The Fourth Paradigm: Data-Intensive Scientific Discovery", pp. 125-129, Hey, T., Tansley, S., Tolle, K. eds., Microsoft Research, Redmond, 2009;
- [10] Grafe, G., "Query Evaluation Techniques for Large Databases", ACM Computing Surveys (CSUR), Vol. 25, issue 2 (June 1993), pp 73-169, ACM, New York, 1993;
- [11] Srividya, L., Nematollah, S., "A Compact Multi-resolution Index for Variable Length Queries in Time Series Databases", Knowledge and Information Systems, Volume 15, issue 2 (May 2008), pp 131-147, Springer-Verlag, New York, NY, 2008;
- [12] Yang-Sae, M., Jinho, K., "Efficient Moving Average Transform-based Subsequence Matching Algorithms in Time-Series Databases", Information Sciences: an International Journal, Volume 177, Issue 23 (December 2007), pp 5415-5431, Elsevier Science Inc. New York, NY, 2007;
- [13] Woong-Kee S., Kyu-Young, W., "A Subsequence Matching Algorithm that Supports Normalization Transform in Time-Series Databases", Data Mining and Knowledge Discovery, Volume 9, Issue 1 (July 2004), pp 5-28, Kluwer Academic Publishers, Hingham, MA, 2004;

- [14] Eltabakh, M. Y., Hon, W., Shah, R., Aref, W. G., Vitter, J. S., "The SBC-tree: An Index for Run-length Compressed Sequences", ACM International Conference Proceeding Series; Vol. 261, Proceedings of the 11th international conference on Extending database technology: Advances in database technology, pp 523-534, ACM, New York, NY, 2008;
- [15] Püschel, M., Moura, J. M. F., Singer, B., Xiong, J., Johnson, J., Padua, D., Veloso, M., Johnson, R. W., "Spiral: A Generator for Platform-Adapted Libraries of Signal Processing Algorithms", The International Journal of High Performance Computing Applications, Volume 18, No. 1, Spring 2004, pp 21-45, Sage Publications, 2004;
- [16] Graefe, G., "Implementing Sorting in Database Systems", ACM Computing Surveys, Vol. 38, No. 3, Article 10, September 2006, ACM, New York, NY, 2006.
- [17] Bonnet, P., Gehrke J., Seshadri P., "Towards Sensor Database Systems", Lecture Notes in Computer Science, Mobile Data Management, Vol. 1987/2001, pp 3014, Springer Berlin/ Heidelberg, Berlin, 2001;
- [18] Ailamaki, A., Kantere, V., Dash, D., "Managing Scientific Data", Communications of the ACM, June 2010, Vol. 53, No. 6, pp 68-78, ACM, New York, NY, 2010;
- [19] Leonhardi, B., Mitschang, B., Pulido, R., Sieb, C., Wurst, M., "Augmenting OLAP Exploration with Dynamic Advanced Analytics", ACM International Conference Proceeding Series; Vol. 426., Proceedings of the 13th International Conference on Extending Database Technology", pp 687-692, ACM, New York, NY, 2010;
- [20] Cudre-Mauroux, P., Kimura, H., Lim, K.-T., Rogers, J., Simakov, R., Soroush, E., Velikhov, P., Wang, D. L., Balazinska, M., Becla, J., DeWitt, D., Heath, B., Maier, D., Madden, S., Patel, J., Stonebraker, M., Zdonik, S., "A Demonstration of SciDB: A Science-Oriented DBMS", VLDB 2009, August 24-28, Lyon, France, ACM, New York, NY, 2009;
- [21] Stonebraker, M., Becla, J., DeWitt, D., Lim, K.-T., Maier, D., Ratzesberger, O., Zdonik, S., "Requirements for Science Data Bases and SciDB", In Conference on Innovative Data Systems Research (CIDR), CIDR Perspectives 2009, 2009; disponível em <http://www-db.cs.wisc.edu/cidr/cidr2009/Paper_26.pdf>, último acesso 21/08/2010;
- [22] Ahmad, Y., Papaemmanouil, O., Çetintemel, U., Rogers, J., "Simultaneous Equation Systems for Query Processing on Continuous-Time Data Streams", Proceedings of the 24th International Conference on Data Engineering, ICDE 2008, April 7-12, 2008, Cancún, México 2008;
- [23] Lema, J. A. C., Behr, T., Duentgen, C. , "External Representation of Spatial and Spatio-Temporal Values", Abril 2004, disponível em <<http://dna.fernuni-hagen.de/Secondo.html/files/SpatialListFormat.pdf>>, último acesso em 21/08/2010;
- [24] "Secondo – Version 2.9.0 – Programmer's Guide", disponível em <<http://dna.fernuni-hagen.de/Secondo.html/files/ProgrammersGuide.pdf>>, último acesso em 21/08/2010;
- [25] "Secondo 2.9.0 – User Manual", disponível em <<http://dna.fernuni-hagen.de/Secondo.html/files/SecondoManual.pdf>>, último acesso em 21/08/2010;

- [26] “Syntax of the SECONDO Optimizer Query Language”, disponível em <<http://dna.fernuni-hagen.de/Secondo.html/files/Optimizer-Syntax.pdf>>, último acesso em 21/08/2010;
- [27] Ramakrishnan, R., “Database Management Systems”, WCB/McGraw-Hill, New York, NY, 1997;
- [28] Exemplo de Chart Recorder, disponível em, <http://en.wikipedia.org/wiki/Chart_recorder>, último acesso em 21/08/2010;
- [29] Sistema Historian da GE FANUC, disponível em, <<http://www.ge-ip.com/account/download/3912/2420>>, último acesso em 21/08/2010;
- [30] Gomes, J., Velho, L., “Computação Gráfica: Imagem”, Série de Computação e Matemática, IMPA/SBM Rio de Janeiro, 1994;
- [31] “Communications protocol”, Wikipedia, disponível em : <http://en.wikipedia.org/wiki/Communications_protocol>, último acesso em 21/08/2010;
- [32] “Ethernet” , Wikipedia, disponível em : <<http://en.wikipedia.org/wiki/Ethernet>>, último acesso em 21/08/2010;
- [33] “Transmission Control Protocol” , Wikipedia, disponível em : <http://en.wikipedia.org/wiki/Transmission_Control_Protocol>, último acesso em 21/08/2010;
- [34] “EIA-485”, Wikipedia, disponível em : <<http://en.wikipedia.org/wiki/EIA-485>>, último acesso em 21/08/2010;
- [35] “IEEE 754-2008”, Wikipedia, disponível em : <http://en.wikipedia.org/wiki/IEEE_floating-point_standard>, último acesso em 21/08/2010;
- [36] Gabrijelcic, P., “Time is The Simplest Thing”, The Delphi Magazine, issue 65, Jan 2001, pp 55-62, disponível em : <<http://www.17slon.com/blogs/gabr/TDM/tdm65-gp.pdf>>, último acesso em 21/08/2010;
- [37] “Class Timestamp”, disponível em : <<http://download.oracle.com/javase/1.4.2/docs/api/java/sql/Timestamp.html>>, último acesso em 21/08/2010;
- [38] “What is DIAdem”, National Instruments, disponível em : <http://www.ni.com/diadem/what_is.htm>, último acesso em 21/08/2010;
- [39] “The Advantages of Plant-wide Historians vs. Relational Databases (GFT-740A)”, disponível em, <<http://www.ge-ip.com/account/download/10707/2420>>, último acesso em 21/08/2010;
- [40] “COUGAR: The Network Is The Database”, disponível em, <<http://www.cs.cornell.edu/bigreddata/cougar/index.php#overview>>, último acesso em 21/08/2010;
- [41] “NI Labview”, National Instruments, disponível em : <<http://www.ni.com/labview/>>, último acesso em 21/08/2010;

- [42] Gannon, D., Deelman, E., Shield, M., Taylor, I., "Introduction", em "Workflows for e-Science, Scientific Workflows for Grids", pp. 1-8, Taylor, I. J., Deelman, E. Gannon, D., Shields, M. S., ed, Springer, London, 2007;
- [43] Barga, R., Gannon, D. "Scientific versus Business Workflows", em "Workflows for e-Science, Scientific Workflows for Grids", pp. 9-16, Taylor, I. J., Deelman, E. Gannon, D., Shields, M. S., ed, Springer, London, 2007;
- [44] Shields, M., "Control- versus Data-Driven Workflows", em "Workflows for e-Science, Scientific Workflows for Grids", pp. 167-173, Taylor, I. J., Deelman, E. Gannon, D., Shields, M. S., ed, Springer, London, 2007;
- [45] Gannon, D., "Component Architectures and Services: From Application Construction to Scientific Workflows", em "Workflows for e-Science, Scientific Workflows for Grids", pp. 174-189, Taylor, I. J., Deelman, E. Gannon, D., Shields, M. S., ed, Springer, London, 2007;
- [46] Daini, O. A., "Numerical Database Management System: A Model.", ACM SIGMOD Conference 1982. (engineering and scientific applications), pp. 192-199., ACM, New York, 1982;
- [47] McCarthy, D., Dayal, U., "The Architecture of an Active Database Management System", ACM SIGMOD, Volume 18, June 1989, pp. 215-224, ACM, New York, 1989;
- [48] Yao, Y., Gehrke, J. "The Cougar Approach to In-Network Query Processing in Sensor Networks", ACM SIGMOD, Volume 31, Setembro 2002, pp. 9-18, ACM, New York, 2002;
- [49] Singhal, M., "Issues and Approaches to Design Real-Time Database Systems", ACM SIGMOD, Volume 17, Março 1988, pp. 19-33, ACM, New York, 1988;
- [50] Aref, W. G., Elfeky, M. G., Elmagarmid, A. K. "Incremental, Online, and Merge Mining of Partial Periodic Patterns in Time-Series", IEEE Transactions on Knowledge and Data Engineering, Volume 16, Março 2004, pp. 332-342, IEEE Educational Activities Department, Piscataway, NJ, 2004;
- [51] "JCGM 100:2008 Evaluation of measurement data – Guide to the expression of uncertainty in measurement", JCGM – Joint Committee for Guides in Metrology, BIPM – Bureau Internationale de Poids e Mesures, Setembro 2008, disponível em <<http://www.bipm.org/en/publications/guides/gum.html>>, último acesso em 12/10/2010;
- [52] Berkely DB, disponível em: <<http://www.oracle.com/technetwork/database/berkeleydb/overview/index.html>>, último acesso em 29/01/2011;
- [53] Zhu, Y., Shasha, D. "Efficient Elastic Burst Detection in Data Streams", ACM SIGKDD '03, August 24-27, Washington, DC, USA; 2003;
- [54] Cole, R., Shasha, D., Zhao, .X. "Fast Window Correlations Over Uncooperative Time Series", ACM SIGKDD '05, August 21-24, Chicago, Il, USA, 2005;
- [55] Kaplan, I., "Applying the Haar Wavelet Transform to Time Series Information", disponível em: <http://www.bearcave.com/misl/misl_tech/wavelets/haar.html>, último acesso em 29/01/2011;

[56] Neylon, T., “Sparse solutions for linear prediction problems”, Tese de doutorado, Sasha, D., orientador, Department of Mathematics, NYU, NY, maio 2006; disponível em : <<http://cs.nyu.edu/cs/faculty/shasha/papers/neylonthesis.pdf>>, último acesso em 29/01/2011;

[57] Sasha, D., “Time Series in Finance: the array database approach”, Courant Institute of Mathematical Sciences, Department of Computer Science, New York University, disponível em : <<http://cs.nyu.edu/shasha/papers/jagtalk.html>>, último acesso em 29/01/2011;

[58] Historis, “Relational vs. Time Series”, disponível em : <<http://historisdatabase.com/time-series-data/relational-time-series.php>>, último acesso em 29/01/2011;

[59] KX, “[kx] white paper”, disponível em : <<http://kx.com/papers/Kdb+ Whitepaper-2010-1005.pdf>>, último acesso em 29/01/2011;

[60] Historis, “Time-Series Database Implementation”, disponível em: <http://www.historisdatabase.com/pdfs/Historis_Case_Study.pdf> , último acesso em 29/01/2011;

[61] Zhu, Y., Shasha, D. “Elastic Burst detection”, disponível em: <<http://cs.nyu.edu/cs/faculty/shasha/papers/burst.d/burst.html>>, último acesso em 29/01/2011;

[62] Güting, R.H., “Second-Order Signature: A Tool for Specifying Data Models, Query Processing and Optimization”. ACM SIGMOD Conference, pp. 277-286, Washington, EUA, 1993;

[63] Dieker, S., Güting, R.H., “Plug and Play with Query Algebras: SECONDO. A Generic DBMS Development Environment”. Proc. Int. Database Engineering and Applications Symposium (IDEAS), pp. 380-392, Yokohama, Japão, 2000;

[64] Güting, R.H., Behr, T., Almeida, V.T., Ding, Z., Hoffmann, F., Spiekermann, M., “SECONDO: An Extensible DBMS Architecture and Prototype”, Informatik-Report 313, Fernuniversität, Hagen, Alemanha, 2004;

[65] Manegold, S., Kersten, M.L., Boncz, P., “Database Architecture Evolution: Mammals Flourished long before Dinosaurs became Extinct”, ACM VLDB '09, Lyon, França, 2009;

[66] Figueiredo, G., Braganholo, V., Mattoso, M. “Um Mediador para o Processamento de Consultas sobre Bases XML Distribuídas”, Seção de Demos do Simpósio Brasileiro de Banco de Dados, 2007, João Pessoa, p. 21-26, 2007;

[67] Ogasawara, E., Oliveira, D., Valduriez, P., Dias, J., Porto, F., Mattoso, M., “An Algebraic Approach for Data-Centric Scientific Workflows”, ACM VLDB '11, Seattle, EUA, 2011;

[68] PostgreSQL , disponível em: <<http://www.postgresql.org/>>, último acesso em 11/07/2011;

[68] Flex, “Flex – The Fast Lexical Analyzer”, disponível em : <<http://flex.sourceforge.net/>>, último acesso em 21/07/2011;

- [69] Flex, “Flex – The Fast Lexical Analyzer”, disponível em : <<http://flex.sourceforge.net/>>, último acesso em 21/07/2011;
- [70] GNU, “The GNU Operating System”, disponível em : <<http://www.gnu.org/>>, último acesso em 21/07/2011;
- [71] Bison, “Bison – GNU parser generator”, disponível em : <<http://www.gnu.org/software/bison/>>, último acesso em 21/07/2011;
- [72] MinGW. “MinGW | Minimalist GNU for Windows”, disponível em : <<http://www.mingw.org/>>, último acesso em 21/07/2011;
- [73] Ubuntu, “Homepage | Ubuntu”, disponível em : <<http://www.ubuntu.com/>>, último acesso em 21/07/2011;
- [74] SWI-Prolog, “SWI-Prolog’s home”, disponível em : <<http://www.swi-prolog.org/>>, último acesso em 21/07/2011;
- [75] Technip, “Technip”, disponível em : <<http://www.technip.com/en>>, último acesso em 23/11/2011;
- [76] Callister, W. D., “Ciência e Engenharia de Materiais: Uma Introdução”, 5ª edição, LTC, Rio de Janeiro, 2002;
- [77] Lynx Tecnologia, “Sobre a Lynx”, <<http://www.lynxtec.com.br/lynx.htm>>, último acesso em 17/11/2011;
- [78] AqDados e AqDAnalysis, “Lynx Produtos”, <http://www.lynxtec.com.br/prod_prog_1.htm>, último acesso em 17/11/2011;
- [79] HBM, “HBM Test and Measurement: Transducer, Load Cell, DAQ System”, <http://www.hbm.com/en/?geoip_cn=none>, último acesso em 17/11/2011;
- [80] Mohan, C. – “IBM Global Technology Outlook (GTO) 2011”, palestra proferida na Coppe Sistemas, no dia 07/10/2011, sala H-324b, Rio de Janeiro, 2011;
- [81] Stackoverflow, “How best to sum up lots of floating point numbers?”, disponível em: <<http://stackoverflow.com/questions/394174/how-best-to-sum-up-lots-of-floating-point-numbers>>, último acesso em 07/04/2012;
- [82] Hawking, Stephen, “Uma Breve História do Tempo: do Big Bang aos Buracos Negros”, ed. Rocco, Rio de Janeiro, 1988;
- [83] Petrobras N-2409, “2ndAmendment - ENGLISH - FLEXIBLE PIPE”, Referenciada em : “Catálogo de Normas Técnicas PETROBRAS – Ordem Numérica – Mai/2012”, disponível em : <<http://www.petrobras.com.br/CanalFornecedor/portugues/pdf/Catalogo-Normas-Tecnicas-Petrobras.pdf>>, último acesso em 13/06/2012;

Apêndice A. Automação Industrial

A.1 Histórico

Para o entendimento deste trabalho se faz necessário apresentar um breve histórico sobre a evolução dos sistemas de controle industriais. Apesar de resumido, espera-se apresentar a nomenclatura necessária para o entendimento do problema.

Uma fábrica é composta de várias máquinas e equipamentos que, ligadas umas as outras, formam uma linha de produção. O controle do processo é realizado a partir do modelo das variáveis que formam o produto. Por exemplo, uma fábrica de cerveja possui vários vasos em que são depositadas diferentes quantidades de diversos produtos, de modo a seguir a receita de um tipo de cerveja específica. Em cada vaso pode ser feito um tipo de cerveja diferente, portanto, quantidade de malte e lúpulo (em kg), quantidade de água (em litros), temperatura do vaso (em graus Celsius) e tempo de fermentação (em horas ou dias) seriam algumas das variáveis que modelam este processo.

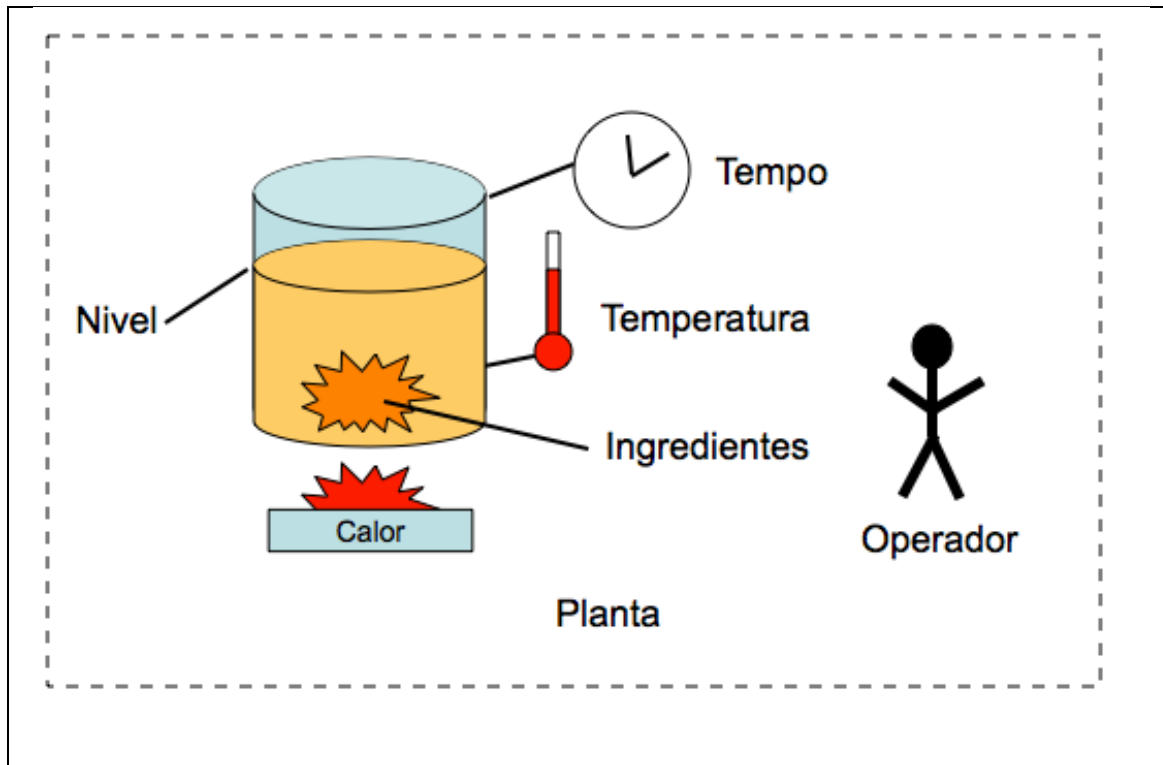


Figura 65 - Exemplo de Processo de Fabricação de Cerveja

Inicialmente, todo o controle destes processos era feita manualmente, por exemplo, medindo-se a temperatura a cada hora e anotando-se este valor em uma folha de papel. Aos poucos foram surgindo equipamentos que auxiliavam esta rotina, como os marcadores de pena [28], que desenhavam gráficos em bobinas de papel que rolavam de acordo com o tempo. Até hoje, o termo “pena” é utilizado quando se fala em gráficos de tendência.

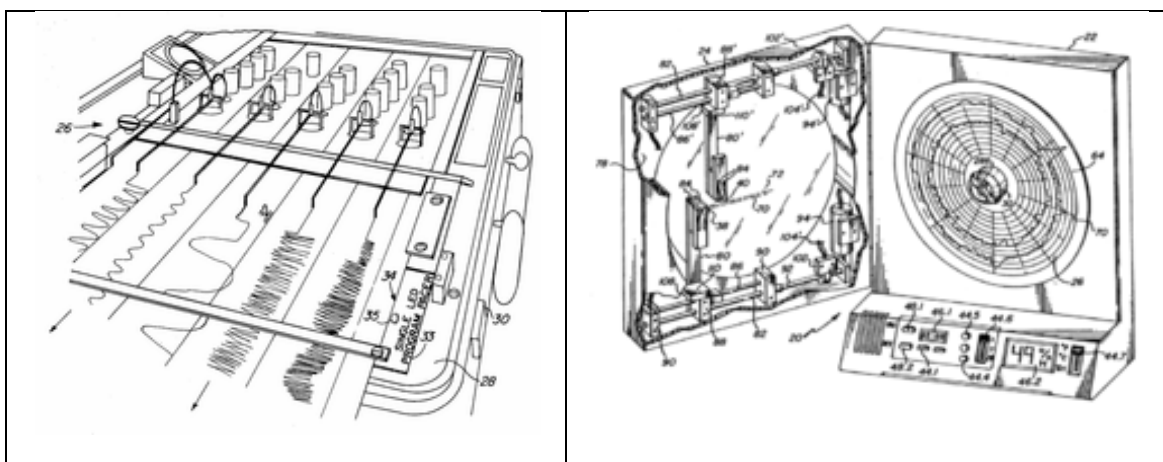


Figura 66 - Exemplos de marcadores de pena (gráficos de tendência)

Estes sensores não atuavam no processo. Era necessário que o responsável avaliasse e os dados e mandasse abrir ou fechar uma válvula ou aquecer o resfriar algum vaso. Estes sensores também estavam localizados fisicamente ao lado dos equipamentos, isto é para se efetuar o controle de uma planta era necessário percorrê-la toda.

Surgem então os sensores que conseguem transmitir seus dados à distância. As leituras são transformadas em sinais elétricos e transmitidas para uma unidade de controle central, de modo que todas as informações sobre a planta podem ser obtidas instantaneamente. Surgem também os atuadores. Equipamentos que são capazes de abrir ou fechar válvulas e que também podem ser operados à distância. A figura do Operador passa a ficar apenas na sala de controle, lendo as variáveis do campo (planta) e atuando através de um painel de controle (painel sinótico).

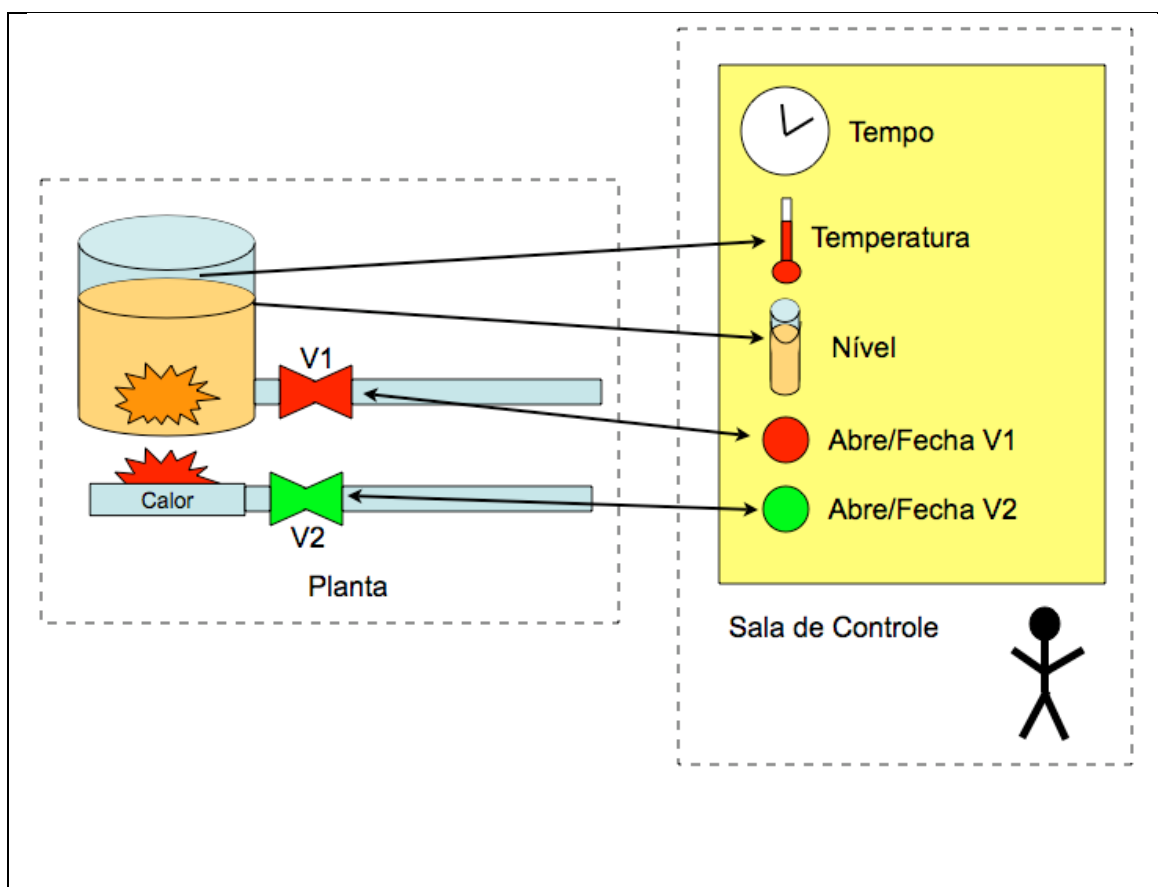


Figura 67 - Operação através de painel sinótico

Por fim, surgem os equipamentos digitais de controle. Estes equipamentos visam ajudar o operador controlando o processo ou gerando alarmes quando algo está fora do normal. O equipamento mais utilizado é o PLC (*Programmable Logical Controller*) ou CLP em português. Este equipamento pode ser programado com a lógica (receita) do controle e atua na planta conforme a sua programação. Outro equipamento utilizado é a chamada RTU (*Remote Terminal Unit*), chamada em português apenas de unidade remota. Este equipamento concentra vários sinais vindos do campo e os transmite para outros equipamentos (PLCs).

Hoje em dia existem diversos tipos de PLCs para cada tipo de controle desejado. Alguns que suportam várias linguagens de programação, alguns que possuem linguagens próprias para controlarem problemas específicos, como por exemplo, os Controladores de PID, que atuam especificamente conforme o algoritmo de controle PID (Proporcional, Integral e Derivativo), estes últimos, por serem tão específicos e pouco versáteis, acabam por não serem considerados como PLCs.

A.1.1 Supervisão, Controle e Monitoramento

A interface de um PLC com um operador é feita através de indicadores e botões, sendo desenhada especificamente para o processo controlado. Mudando-se o processo, é necessário redesenhar todo um painel. Com a entrada do micro-computador no cenário de automação, surgiu a possibilidade de se criar um sistema de interface mais flexível. O micro-computador passou a ser ligado ao PLC e os painéis sinóticos compostos por luzes, indicadores e botões foram convertidos em interfaces gráficas, facilitando o processo de criação e modificação no caso de alterações no processo. Exemplos podem ser vistos nas Figura 68 e Figura 69.

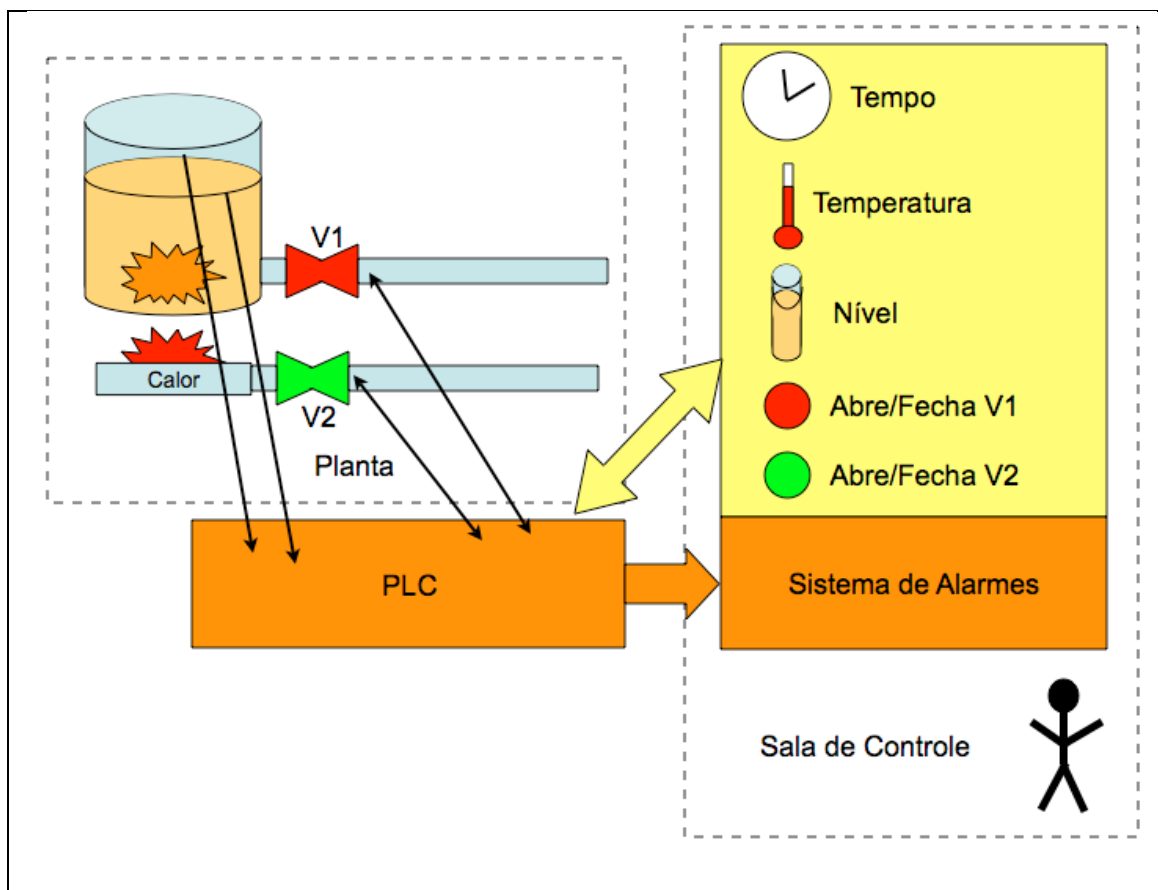


Figura 68 - Operação por painel Sinótico, Controle por PLC

Uma das siglas comuns na área de Automação Industrial é SCADA, significando, *Supervisory Control and Data Acquisition* (Controle, Supervisão e Aquisição de Dados). Quando falamos em controle, deixamos claro que o sistema atua diretamente sobre o processo. Se existe uma situação inesperada, um alarme é gerado e o sistema atua abrindo ou fechando uma válvula. Por se tratar de uma tarefa crítica o controle é delegado apenas para máquinas com dedicação exclusiva (PLCs). Quando o sistema não atua diretamente no processo dizemos que é utilizado somente para Supervisão.

O computador ligado ao PLC passou a ter o nome de sistema supervisório. O controle do processo é ainda de responsabilidade do PLC, mas a interface passou a ser feita pelo computador.

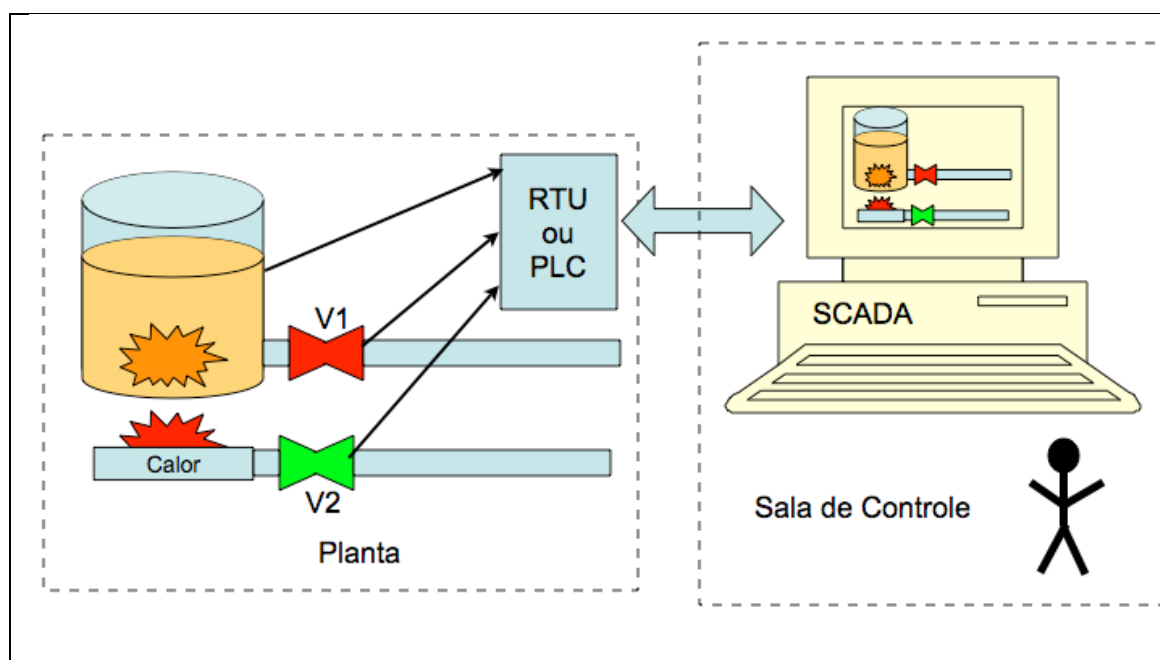


Figura 69 - Operação através de sistema SCADA

Outra função que também foi acumulada é a de armazenamento de dados históricos do processo. Os marcadores de pena foram substituídos por algum tipo de banco de dados e a recuperação da informação passou a ser através de telas gráficas ou relatórios gerados pelo sistema supervisor.

Um termo comum utilizado atualmente é PIMS, *Process Information Management System*, ou Sistema Gerencial de Informações do Processo. Sistemas PIMS são capazes de armazenar grandes quantidades de informação e extrair informações gerenciais a partir dos dados coletados. Uma informação possível seria, por exemplo, qual foi a temperatura média de um vaso no último mês. Considerando que este valor pode ter sido adquirido a cada segundo, falamos de tirar a média de uma série histórica com mais de 2 milhões de valores. Um exemplo de sistema PIMS é o Proficy Historian [29] da GE Fanuc.

A.1.2 Monitoramento

Por fim podemos falar da ideia de monitoramento. Monitoramento pode ser entendido como uma evolução do controle, porém, a função do monitoramento não é o controle do processo em si. Isto já é feito pelos sistemas descritos anteriormente. A função do monitoramento é um melhor entendimento do processo através da análise dos dados coletados.

Monitoramento é uma característica muito comum de laboratórios de análises. Nestes casos, o processo a ser controlado é apenas uma simulação do processo real com características de controle muito simples. O número de dados aquisitados, por outro lado, é muito superior ao do controle propriamente dito. Tomemos por exemplo um túnel de vento onde queremos testar uma maquete de um avião (Figura 70). O controle do túnel de vento consiste apenas em fazer girar um ventilador e garantir através de um anemômetro que a velocidade de vento desejada foi atingida. Por outro lado, o modelo do avião contém diversos sensores que visam indicar seu comportamento. Esta grande massa de dados deve ser aquisitada a uma velocidade muito mais alta que o necessário apenas pelo controle. Cada informação coletada fará diferença no momento da análise.

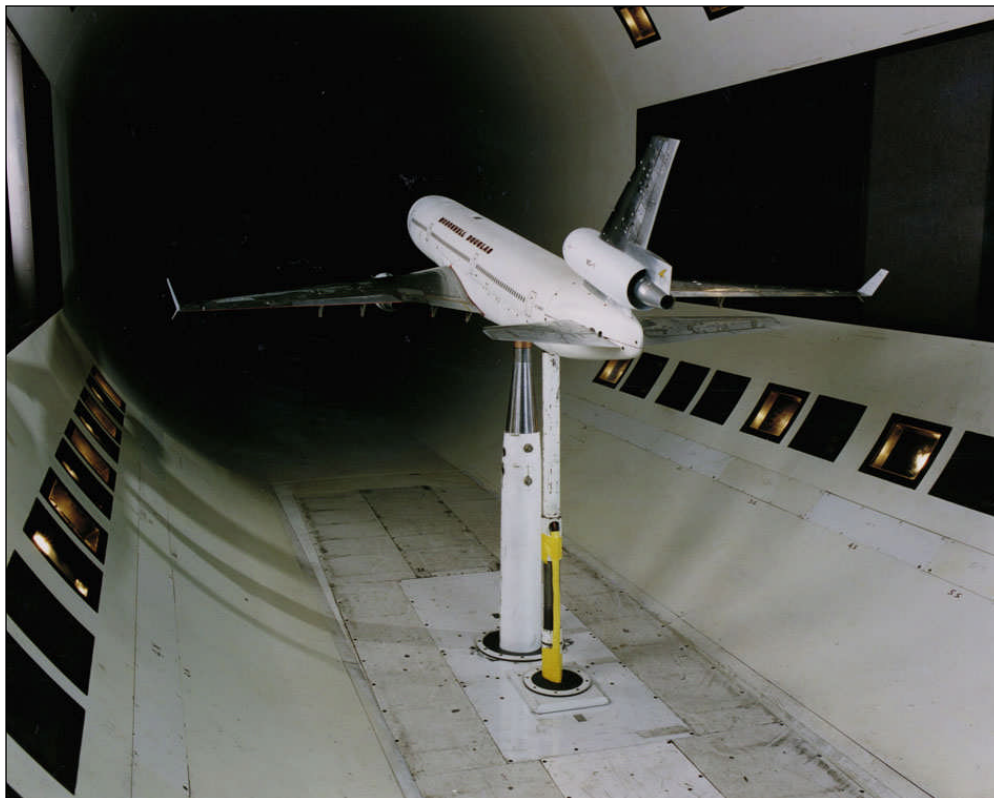


Figura 70 - Teste de Avião em Túnel de Vento

Um dos problemas estudados neste trabalho é justamente como esta grande quantidade de informações deve ser armazenada. Existem no mercado diversos SGBDs e sistemas proprietários que armazenam grande quantidade de informação, mas, como veremos adiante, podem existir problemas quanto a forma de efetuarem cálculos sobre este largo espectro de dados.

A.1.3 Componentes da Automação Industrial

Para efetuar o controle de uma planta é necessário processar o dado vindo do sensor. Este, por sua vez, pode ser uma unidade analógica, por exemplo, um sensor de temperatura, ou uma unidade digital³³, indicando se uma bomba está ligada ou desligada. Em casos de unidades digitais é extremamente simples converter este dado como entrada de um sistema digital. No caso de unidades analógicas, é necessário um conversor (*ADC – Analog to Digital converter*) que interprete o sinal e converta-o para a grandeza desejada, em forma digital, de modo a ser tratado pelo PLC.

Os sinais podem ser classificados também como de entrada (leituras do campo) e de saída (comandos de atuação), de modo que existem quatro tipos de sinais que devem ser modelados: Entradas Analógicas e Digitais (AI e DI – *Analog Inputs* e *Digital Inputs*) e Saídas Analógicas e Digitais (AOs e DOs – *Analog Outputs* e *Digital Outputs*). Assim como as grandezas de entrada analógicas possuem um ADC, as grandezas de saída analógicas requerem um DAC (*Digital to Analog Converter*).

³³ Todas as variáveis da planta serão eventualmente digitalizadas para o controle do processo, porém, utiliza-se a nomenclatura Digital/Analógica para grandezas que apresentam este tipo de comportamento no processo.

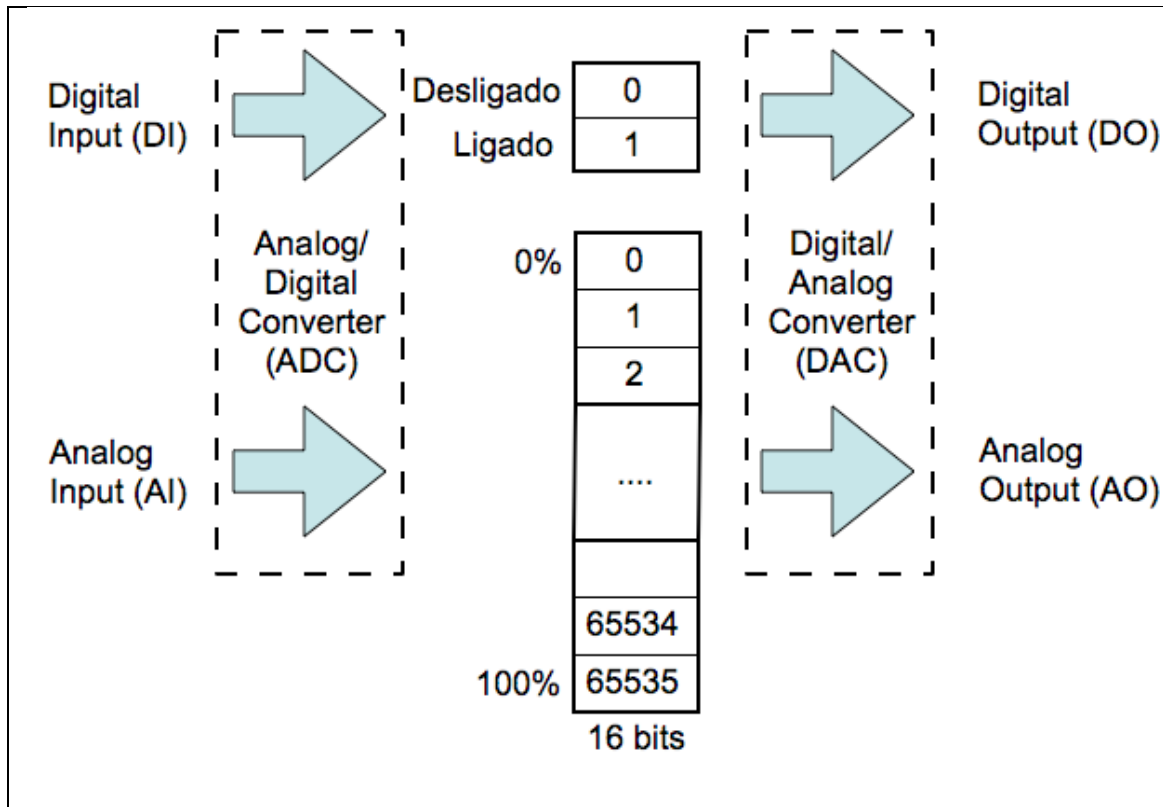


Figura 71 - Processo de Conversão Analógico/Digital

O processo de conversão de sinais analógicos para sinais digitais chama-se quantização (Gomes [30]) e consiste em dividir a escala em que a variável pode percorrer em 2^b unidades, sendo b o número de bits de precisão que se deseja utilizar (Figura 71). Na prática, quem determina este valor de precisão é o equipamento ADC/DAC. Uma vez digitalizado, este valor pode ser armazenado pelo sistema digital como um número inteiro, de ponto fixo ou de ponto flutuante. As vantagens e desvantagens de cada método serão discutidas posteriormente.

Outro ponto necessário à modelagem do processo é o tempo (ou frequência) de aquisição de cada variável. Alguns pontos são mais sujeitos a variações do que outros. Valores de temperatura costumam apresentar uma variação muito lenta. Valores de pressão, por outro lado, apresentam variação muito rápida. Basta tomarmos como exemplo uma chaleira ao fogo e um balão cheio de ar. A chaleira levará em torno de minutos para que a água aqueça e ferva. Já o balão, se o furarmos

com um alfinete, alterará sua pressão interna instantaneamente. A Figura 72 apresenta um exemplo com um medidor de temperatura, um medidor de pressão e um extensômetro (*Strain Gauge*) que mede a deformação de um corpo.

Equipamentos de automação industrial costumam medir o tempo em segundos. Quando são utilizados para medir grandezas rápidas, a unidade passa a ser milissegundos. Por outro lado, equipamentos de análise costumam utilizar frequência de aquisição, medida em Hz. Estas duas nomenclaturas são inversamente proporcionais e converter de uma para outra é trivial.

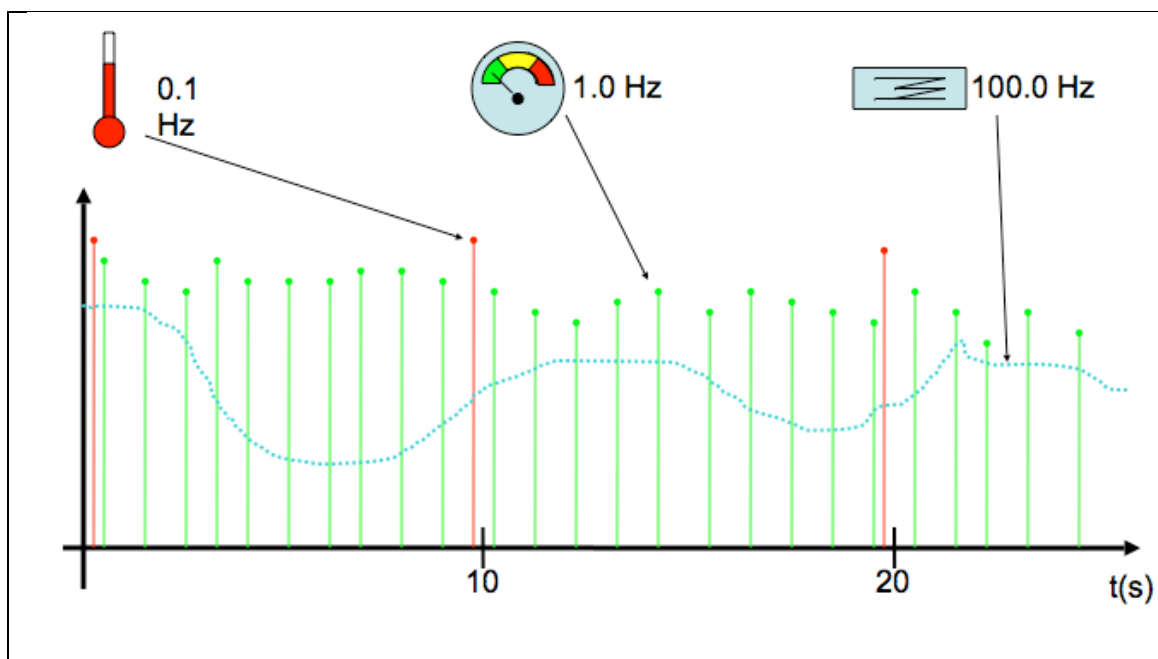


Figura 72 - Diferentes Taxas de Aquisição

A.2 Processamento da Informação

A.2.1 Interligação de Tecnologias e Protocolos de Comunicação

De modo a interligar todas as tecnologias existentes, e aqui utilizamos o termo tecnologias por se tratarem tanto de equipamentos físicos (hardware), como também diversas camadas de programas (software), existem vários protocolos de comunicação utilizados. Não é nosso objetivo discutir qual protocolo é melhor do que o outro, mas exemplificar o tipo de problema que surge nas interfaces.

Entendem-se por protocolos, o conjunto de regras que estabelecem a comunicação de dados em uma rede [31]. Estas regras podem estabelecer padrões tanto de hardware como de software. O protocolo Ethernet [32], por exemplo, estabelece padrões para os conectores e para os dados. Alguns estabelecem apenas para os dados (TCP/IP [33]), outros apenas para as conexões físicas (RS-485 [34]).

Sensores normalmente são ligados às unidades remotas e/ou PLCs através de protocolos elétricos analógicos. São os chamados “*loops*” de corrente, tensão ou impedância. Um sensor de *loop* de corrente varia uma medida em ampéres, por exemplo, de 4 a 20 miliampéres (mA) conforme uma escala pré-estabelecida do sensor que é reconhecida pela unidade remota. A escolha do tipo de comunicação do sensor está ligada, entre outros fatores, a distância do mesmo à remota. Sensores que se comunicam por tensão são sensíveis a resistência causada pelos fios e a leitura do dado pode ser comprometida à longas distâncias. Outro fator prejudicial também é o ruído que pode afetar o sinal.

Uma vez digitalizado pela unidade remota o sinal pode ser enviado para um sistema SCADA por algum protocolo digital mais robusto. Existem no mercado vários protocolos de comunicação comumente utilizados para tal. Exemplos são ModBus,

Profibus, GPIB, DNP etc. Estes protocolos podem ser utilizados sobre uma série de meios físicos, portas seriais RS-232, redes ethernet e cabos USB. O aspecto que nos é importante analisar é como o dado é transmitido.

As unidades remotas costumam enviar os dados de três maneiras diferentes: em formato binário inteiro, em formato de ponto-flutuante e em formato texto. O dado em formato binário representa a conversão do sinal lido (4 a 20 mA, por exemplo) para um número inteiro de b bits. Normalmente são utilizados números de 16 bits, mas números de 12 ou 14 bits de precisão também podem ser encontrados. No caso de 16 bits, 0 representaria um valor de 4 mA e 65535 um valor de 20mA. Nestes casos a remota não faz a conversão do sinal para a unidade de engenharia (escala) da variável.

Remotas que enviam dados em números de ponto-flutuante (padrão IEEE-754), já o fazem com o número convertido para a unidade de engenharia desejadas. Ou seja, os valores em mA, Volts ou Ohms, são transformados em valores de graus Celsius, Bar, PSI ou qualquer outra unidade desejada. Números de ponto flutuante possuem 23 bits na mantissa em precisão simples e 53 bits na mantissa de precisão dupla, ou seja, a princípio suficientes para armazenar a precisão dos conversores analógico-digitais.

Alguns protocolos enviam a informação em formato texto. Os valores são convertidos para unidade de engenharia e enviados em decimal. Por exemplo, o valor 100, seria enviado como 100.00. O número de casas decimais é pré-estabelecido. Neste caso poderíamos chamar este formato de ponto-fixo decimal.

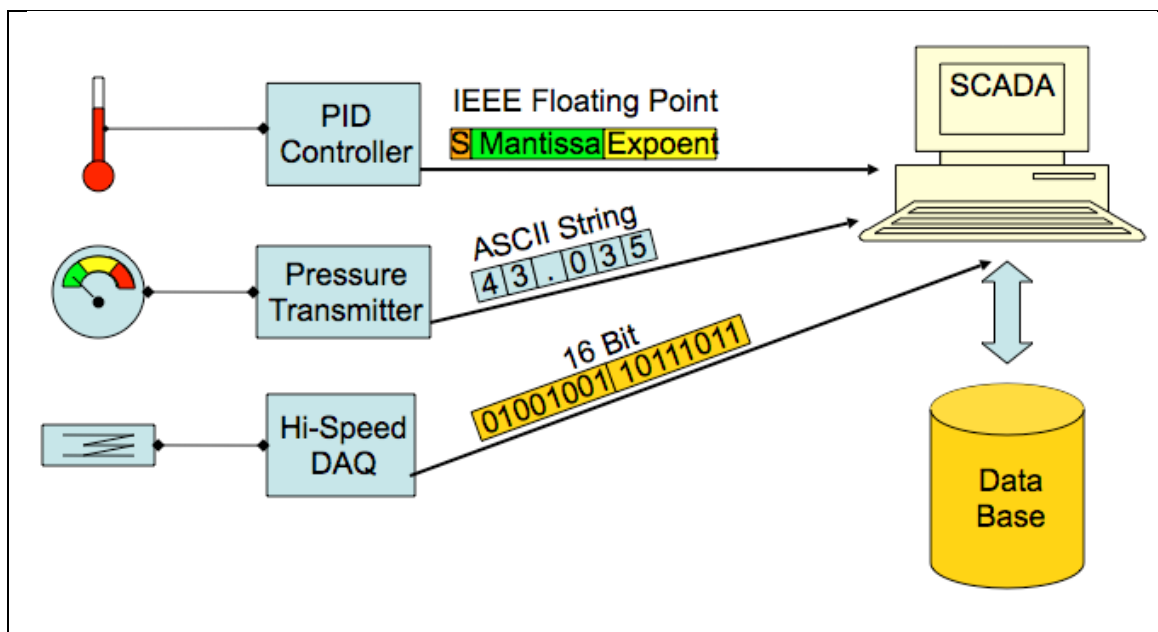


Figura 73 - Exemplo de Formatos de Transmissão de Dados

Após receber o valor, o sistema SCADA precisará convertê-lo para a sua representação interna ou ainda disponibilizá-lo para um banco de dados para armazenamento histórico (Figura 73). Este tipo de conversão é normalmente para um formato de ponto-flutuante de precisão dupla.

A princípio, assume-se que como está-se convertendo um valor sempre para formatos com maior precisão, a mesma será mantida. Mas esta afirmação é falsa. O valor 0.1, por exemplo, não possui representação exata em formatos de ponto flutuante, isto é, gera uma dízima periódica em seu formato binário. A representação em formato inteiro oriunda do sensor pode ser mais acurada do que a representação pelos formatos IEEE.

A.2.2 Precisão e Incerteza

De um ponto de vista prático, a questão da precisão em SGBDs convencionais é uma matéria já ultrapassada. Ao se modelar o tipo de dado, define-se a precisão em números de casas inteiras e decimais. Quando armazenamos cifras, normalmente duas casas decimais, relativas aos centavos, são empregadas, quando armazenamos quotas de participação, por exemplo, uma precisão maior é empregada. Programas de contabilidade se beneficiam de tipos de dados com ponto fixo, pois a aritmética inteira evita problemas contábeis de arredondamento, mas de uma forma geral, o formato de ponto flutuante é o mais utilizado.

Um ponto pouco abordado pela área de Banco de Dados, é quanto à precisão do resultado de uma operação aritmética. Isto é, não só, uma questão da linha de Aritmética Computacional, conforme mostra o livro de Parhami [2], como também um problema de como usar as incertezas oriundas das medidas, conforme nos indica o Guia para expressão de incertezas em medidas [51].

O problema de precisão quanto ao uso do ponto flutuante é que premissas aritméticas básicas, como por exemplo a propriedade associativa da adição, não se mantêm como verdades nesta representação. Um exemplo simples é que se somarmos uma lista de valores do primeiro ao último e do último ao primeiro, obteremos dois valores distintos, se esta lista possuir números que sejam notavelmente grandes em relação aos demais.

Este primeiro problema, em si, não possui solução, pois trata-se de uma limitação da representação numérica com precisão finita de números que possuem um número extremamente grande, e possivelmente infinito, de casas decimais.

O segundo problema, é quanto à incerteza da medida em si. Imaginemos um termômetro que possui uma precisão de $0,5^\circ$, em sua medição. Ao ser lido um valor de 25° , na verdade está apresentando-se uma faixa de $24,5^\circ$ a $25,5^\circ$. Ao se inserir o valor no banco de dados, este valor torna-se uma medida precisa ($0,2500000 \times 10^2$ e não $0,25 \times 10^2$)³⁴ uma média de valores calculadas SGBD apresentará um valor preciso, que não representará a realidade.

Quanto ao segundo, é preciso que os SGBDs apresentem uma forma para modelar a incerteza, naturalmente inerente à medida, que seja utilizada pelos operadores na geração do resultado. Porém podem ser propostas formas mais precisas de se trabalhar com estes números.

³⁴ Ao utilizar-se a notação científica, entende-se que o último algarismo da mantissa é considerado o “algarismo duvidoso” e representa uma incerteza na medida.

A.2.3 Valores de Data e Hora

Um dos problemas inerentes ao armazenamento de séries temporais é a sincronização dos diversos sinais. Este problema não se trata somente da sincronização dos relógios internos dos equipamentos envolvidos, mas também da forma que os equipamentos representam estes valores.

Falando-se apenas de software, existem algumas formas diferentes de se armazenar este valor de tempo. A Microsoft possui um formato próprio, a comunidade Unix possui outro diferente e as linguagens Delphi e Java, cada qual possui seu formato próprio.

A linguagem Delphi [36], por exemplo, armazena valores de data como números de ponto-flutuante cuja parte inteira representa o número de dias entre uma data base e a parte decimal representa a hora como uma fração do dia. Já o formato Java [37] armazena como um número inteiro de 64 bits representando os milissegundos a partir de uma outra data base. Conforme já foi comentado, trata-se do mesmo problema de conversão entre números inteiros e números de ponto-flutuante.