UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

HENRY GOMES DE CARVALHO

Linguagem de Consulta Temporal: definição e implementação

Dissertação apresentada como requisito parcial
para a obtenção do grau de
Mestre em Ciência da Computação

Prof. Dr. Carlos Alberto Heuser
Orientador

Porto Alegre, outubro de 2002

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Carvalho, Henry Gomes de

Linguagem de Consulta Temporal: definição e implementação / Henry Gomes de Carvalho. – Porto Alegre: Programa de Pós-Graduação em Computação, 2002.

95 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2002. Orientador: Carlos Alberto Heuser.

1. ATSQL2. 2. OASIS. 3. Modelo Temporal. I. Heuser, Carlos Alberto. II. Título.

Dedico esta dissertação
Às minhas filhas Jéssica, Fernanda e Bibiana,
que são o motivo de todo o meu esforço.

Às minhas tias Aracy e Flora,
que sempre me ajudaram nos momentos mais difíceis.

AGRADECIMENTOS

Ao professor Heuser, agradeço pela orientação atenciosa e incentivadora, bem como, por aceitar a minha proposta e acreditar em meu trabalho.

À professora Nina, agradeço pela co-orientação e revisão técnica do trabalho.

Ao colega Prof. Cristiano, por sua amizade, atenção e espírito de colaboração, durante o acerto dos detalhes finais da dissertação.

Ao colega Roberto Luiz, por suas caronas de ida e volta a Porto Alegre e sua infalível amizade.

Agradeço à amiga Lorena, pela revisão final do texto.

Agradeço especialmente ao caríssimo professor e amigo Luis H. William Lagos T. Guedes, pelas contribuições que me permitiram encontrar as melhores traduções para termos técnicos ainda inexplorados por autores de trabalhos sobre banco de dados temporais brasileiros.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

ATSQL2	*Applied Temporal Structured Query Language 2*
DBA	*Data Base Administrator*
SGBD	Sistema Gerenciador de Banco de Dados
OASIS	*Open and Active Specification of Information Systems*
SQL	*Structured Query Language*
TSQL	*Temporal Structured Query Language*
TSQL2	*Temporal Structured Query Language 2*

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Até hoje, não existem implementações de SGBDs Temporais disponíveis no mercado de *software*. A tradução de linguagens de consulta temporais para o padrão SQL é uma alternativa para implementação de sistemas temporais com base em SGBDs comerciais, os quais não possuem linguagem e estrutura de dados temporais.

OASIS (*Open and Active Specification of Information Systems*) é uma linguagem que serve como repositório de alto nível para especificação formal orientada a objetos e geração automática de *software*, em diversas linguagens, através da ferramenta CASE *OO-Method*. As aplicações geradas desta forma utilizam, como meio de persistência de objetos, SGBDs comerciais baseados na abordagem relacional. A linguagem OASIS foi estendida com aspectos temporais. A extensão de OASIS com aspectos temporais requer a especificação de um modelo de dados e de uma linguagem de consulta temporais que possam ser utilizados em SGBDs convencionais. Há duas abordagens para resolver o problema. A primeira baseia-se em extensões da linguagem e/ou do modelo de dados de modo que o modelo não-temporal é preservado. A segunda, abordagem de generalização temporal, é mais radical e não preserva o modelo não-temporal. A linguagem ATSQL2 fornece recursos adequados aos conceitos encontrados na abordagem de generalização temporal. Neste trabalho utiliza-se os conceitos de generalização temporal preservando o modelo não-temporal.

A presente dissertação tem por finalidade propor um modelo de dados para suporte à extensão temporal da linguagem OASIS, bem como estender a linguagem ATSQL2 para facilitar as consultas a eventos temporais. O sistema de tradução da linguagem de consulta temporal para SQL é também adaptado ao modelo de dados proposto.

Palavras-chave: ATSQL2, OASIS, Modelo Temporal.

Temporal Query Language: definition and implementation

ABSTRACT

Until today, there are no Temporal DBMSs implementations available at the software market. The translation from temporal query languages to SQL is an option for the implementation of temporal systems based on commercial DBMS that do not have temporal language and data structures.

OASIS (Open and Active Specification of Information Systems) is a language that acts as a high-level repository to object-oriented formal specification and automatic software generation, in many languages, through OO-Method CASE tools. The applications generated through this approach use commercial relational DBMSs to achieve object persistency. The OASIS language was extended with temporal aspects. The extension of OASIS with temporal aspects requires the specification of a temporal data model and query language that might be used along with conventional DBMSs. There are two approaches to solve the problem. The first is based on language extensions and/or data model, so the non-temporal model is preserved. The second, the temporal generalization approach is more radical and does not preserve the non-temporal model. The ATSQL2 language offers adequate resources to the concepts found on the temporal generalization approach. In this work, the concepts of temporal generalization preserving the non-temporal model are used.

This thesis aims to propose a data model to support the temporal extension of the OASIS language and to extend the ATSQL2 language, in order to help the temporal event queries. The temporal query language to SQL translation system is also adapted to the proposed data model.

Keywords: ATSQL2, OASIS, Temporal Model.

1 INTRODUÇÃO

Linguagens de consulta temporais têm sido alvo de inúmeras pesquisas nas últimas duas décadas. O poder de expressão de tais linguagens torna as consultas, que envolvem aspectos temporais, fáceis de serem escritas em comparação com as linguagens de consulta convencionais, tal como a SQL (MELTON; SIMON, 1993). Mesmo com todo o esforço da comunidade de pesquisa de banco de dados temporais, ainda permanecem os Sistemas Gerenciadores de Banco de Dados (SGBD), os quais, não possuem linguagens de consulta e estruturas de dados temporais.

Tendo em vista essa realidade, a alternativa seria continuar utilizando os SGBDs comerciais, com a diferença que ao acrescentar um *software* intermediário, o mesmo faria a tradução de uma linguagem de consulta temporal (escrita pelo usuário) para o padrão SQL (utilizado pelo SGBD). O *software* intermediário, então, receberia como entrada uma consulta em linguagem temporal e faria a conversão para a linguagem convencional (não-temporal) correspondente àquela consulta, colocando-a como saída final do processo.

A Universidade Federal do Rio Grande do Sul e a Universidade Politécnica de Valência (Espanha), desenvolvem um projeto em conjunto com o objetivo de enriquecer a expressividade da produção automática de um *software* chamado *OO-Method* (PASTOR, 1992a) com características temporais. A linguagem OASIS (PASTOR; SALVERT, 1995; PASTOR et al., 1997, 1998; TORRES, 1998) constitui uma abordagem formal para a especificação de modelos conceituais, seguindo o paradigma orientado a objetos, enriquecida através de regras semânticas. Ela é a base da ferramenta CASE *OO-Method,* cuja especificação das aplicações é feita através de três modelos gráficos obtidos na fase de análise (modelo de objetos, modelo dinâmico e modelo funcional). A notação destes modelos baseia-se em representações gráficas conhecidas na literatura (LARMAN, 1997).

Os modelos gráficos são transformados automaticamente em uma especificação formal OO, OASIS. A partir de OASIS é possível gerar completamente aplicações em várias linguagens de programação. A linguagem OASIS foi estendida com aspectos temporais (ZANATTA, 2000) e as aplicações produzidas através da ferramenta CASE *OO-Method* utilizam SGBDs relacionais como meio de persistência de objetos. Esses dois fatos constituem o problema a ser resolvido.

O objetivo deste trabalho é desenvolver um modelo de dados para suporte à extensão temporal (ZANATTA, 2000) feita na linguagem OASIS que seja implementável em SGBDs baseados na abordagem relacional, estender a linguagem ATSQL2 (BÖHLEN; JENSEN; SNODGRASS, 1995; SNODGRASS et al., 1996a,b) para possibilitar consultas a eventos temporais e propor uma maneira de traduzir as consultas escritas em linguagem temporal (ATSQL2) para a linguagem SQL, de

acordo com o modelo proposto.

Para atingir o objetivo, a presente dissertação vale-se, em parte, da abordagem de generalização temporal (STEINER, 1995) que é uma proposta radical para converter um modelo de dados não-temporal em um temporal. Usar generalização temporal é incrementar todos os construtores de um modelo de dados não-temporal, tais como, estruturas de dados, operações e restrições de integridade para suportar o gerenciamento de dados temporais. Isto significa que a partir dessa conversão, mesmo os comandos SQL convencionais deverão ser modificados pelo *software* intermediário para obter a última situação da tabela consultada. A abordagem de generalização temporal é mais adequada e também mais implementável do que as abordagens que baseiam-se em extensões do modelo de dados e/ou da linguagem de consulta. No entanto, nesta dissertação são utilizados partes das duas abordagens, pois os conceitos introduzidos na abordagem de generalização temporal são aplicados em conformidade com um modelo de dados estendido.

A dissertação está organizada como segue.

No segundo capítulo são apresentados conceitos gerais associados a banco de dados temporais. No terceiro capítulo é apresentado um panorama da linguagem TSQL2, a qual é considerada uma base comum para o desenvolvimento de pesquisas sobre banco de dados temporais. Também é apresentada a linguagem ATSQL2, que oferece suporte aos conceitos de generalização temporal e constitui o principal foco de estudo do presente trabalho. No mesmo capítulo é mostrado como é feita a tradução de comandos ATSQL2 para comandos SQL. No quarto capítulo é feita uma revisão da linguagem OASIS e a sua extensão com aspectos temporais. No quinto capítulo é proposto um modelo de dados estendido para suporte a atributos variáveis temporais e eventos temporais de OASIS. No sexto capítulo é apresentada uma extensão da linguagem ATSQL2 para tratamento de consultas a eventos temporais e uma maneira de traduzir os comandos ATSQL2 para SQL com base no modelo de dados proposto.

Na conclusão da dissertação, apresentada no sétimo capítulo, estão expostas as contribuições e também as limitações do presente trabalho, assim como são apontados os possíveis estudos futuros que poderão ser realizados a partir do mesmo.

2 FORMAS DE REPRESENTAÇÃO TEMPORAL

Neste capítulo são apresentados conceitos gerais para a representação de aspectos temporais. Os conceitos deste capítulo foram baseados em Edelweiss e Oliveira (1994).

2.1 Formas de representação de informações temporais em bancos de dados

Em bancos de dados convencionais, a manipulação dos dados temporais só é possível através da definição de um complexo modelo lógico de entidades e relacionamentos, sendo manipulado por um conjunto de programas de aplicação onde toda a semântica associada ao tempo está contida. De outra forma, quando o mundo real muda, os novos valores são incorporados no banco de dados substituindo os valores antigos. Em banco de dados temporais, quando o mundo real muda é inserido um novo fato na base de dados, continuando armazenados também os valores antigos. A implementação do conceito de tempo pode ser realizada de três formas de acordo com o grau de integração crescente do conceito de tempo no SGBD. Estas formas são:

- a manipulação dos dados temporais é realizada explicitamente pelo usuário, O SGBD só pode armazenar dados dos tipos tradicionais como strings, inteiros, reais, etc. Toda a semântica associada ao tempo está contida na lógica dos programas de aplicação. Neste nível, o usuário deve conhecer a semântica associada ao tempo e assegurar a validade das operações sobre os dados temporais;

- a manipulação dos dados temporais é realizada por meio de ações associadas a propriedades definidas como temporais, isto corresponde à extensões semânticas de tipos de dados normais. Esta solução pode ser aplicada em SGBDs extensíveis pela definição de ações semânticas associadas a tipos de dados temporais. Neste caso, todas as aplicações compartilham o código associado aos novos tipos de dados. A grande fraqueza é o isolamento entre as operações e o esquema conceitual. É impossível representar as propriedades temporais no esquema conceitual pois a semântica temporal é definida como modificação na manipulação de dados tradicionais (reais, strings);

- as propriedades temporais são tratadas por uma extensão do modelo de dados e da linguagem de manipulação. Neste caso, a semântica temporal se torna

estrutural, isto é, ela pertence ao modelo de dados e, portanto, não pode ser alterada pelas aplicações. A definição de um esquema conceitual inclui as propriedades temporais. O principal inconveniente consiste na necessidade de ser desenvolvida uma nova versão do SGBD, incluindo as extensões.

2.2 Ordem no tempo

A definição de uma ordem a ser seguida no tempo é fundamental quando utilizada alguma representação temporal (EDELWEISS; OLIVEIRA, 1994).

Tempo linear

É a forma mais comum de representação da ordem no tempo. Assume-se que o tempo flui linearmente, fazendo com que exista uma ordenação entre dois pontos no tempo, ou seja, definidos dois pontos diferentes no tempo t e t', representando a ordem de precedência temporal através do operador "$<$", uma e somente uma das seguintes expressões é verdadeira: $t < t'$ ou $t' < t$.

Tempo ramificado

Nesta forma a restrição linear é abandonada permitindo a possibilidade de dois pontos diferentes serem sucessores (ramificação no futuro) ou antecessores (ramificação no passado) imediatos de um mesmo ponto. A combinação "passado linear, futuro ramificado" trabalha com uma só história passada e admite múltiplas histórias futuras, representando desta maneira a realidade atual de uma forma bastante fiel.

Tempo circular

Esta forma pode ser utilizada para modelar eventos e processos recorrentes (MAIOCCHI; PERNICI, 1991a).

2.3 Conceitos associados ao domínio de tempo

Existem diversas formas para representar o tempo. Pesquisas tais como Arapis (1991), Gabbay (1991) e Maiocchi (1991a) são exemplos onde podem ser encontrados inúmeros conceitos e formas de representação de aspectos temporais. Devido a isto, viu-se necessário unificar os principais conceitos relativos a bancos de dados temporais. As denominações utilizadas foram apresentadas em Jensen (1994), resultado de um esforço conjunto da comunidade de banco de dados temporais no intuito de estabelecer uma nomenclatura de consenso para a área.

Banco de dados temporal

Um banco de dados temporal é aquele que apresenta alguma forma de representação de informações temporais.

Variação temporal

Duas formas basicamente diferentes podem ser consideradas: tempo contínuo e tempo discreto. O tempo é contínuo por natureza. Entretanto, sem grande perda de generalidade, o tempo pode ser considerado como discreto. Esta segunda forma de representação simplifica grandemente a implementação de modelos de dados.

Chronon

Chronon é um intervalo temporal que não pode ser decomposto. Cada *chronon* de uma seqüência de *chronons* consecutivos possui duração idêntica aos demais.

Instante no tempo

O instante no tempo representa um determinado ponto no tempo. Quando é considerado tempo contínuo, um instante é um ponto no tempo de duração infinitesimal. No entanto, se for considerado tempo discreto, um instante é representado por um dos chronons da linha de tempo.

Intervalo temporal

Intervalo temporal é o tempo decorrido entre dois instantes. O intervalo depende da forma de representação temporal definida no modelo. Quando é considerado tempo contínuo, o intervalo consiste de infinitos instantes de tempo. Na variação discreta um intervalo é representado por um conjunto finito de chronons consecutivos. Um intervalo é representado pelos dois instantes que o delimitam.

Elemento temporal

Elemento temporal é a união finita de todos os intervalos temporais disjuntos, sendo mais uma forma de representação da informação temporal, como o instante no tempo e o intervalo temporal.

Tempo de transação, tempo de validade e tempo definido pelo usuário

Três diferentes conceitos temporais podem ser identificados em aplicações de banco de dados: (i) tempo de transação, o tempo no qual é feita a atualização do banco de dados; (ii) tempo de validade, representando o período de validade da informação armazenada; e (iii) tempo definido pelo usuário, consistindo de propriedades temporais definidas explicitamente pelos usuários em um domínio temporal e manipuladas pelos programas de aplicação. É hoje consenso na comunidade de modelagem temporal, de que tanto a representação do tempo de validade de uma informação como o do seu tempo de transação devem ser consideradas. Estes dois tempos são ortogonais, podendo ser tratados separadamente ou em conjunto.

O tempo de validade pode ser representado de formas distintas, dependendo do elemento temporal básico utilizado no modelo. Quando for utilizado o elemento temporal ponto no tempo, o tempo pode ser representado: (i) através de um ponto no tempo indicando o início da validade, permanecendo o valor até que inicie o tempo de validade de outro valor; ou (ii) através de dois pontos no tempo, o primeiro indicando o início da validade e o segundo, o final da validade. Nos modelos que utilizam o intervalo temporal como elemento temporal básico, o tempo de validade é definido através do intervalo de validade do valor.

Granularidade temporal

A granularidade temporal de um sistema consiste na duração de um *chronon*. Entretanto, dependendo da aplicação considerada, às vezes é necessário considerar simultaneamente diferentes granularidades (minutos, dias, anos) para permitir uma melhor representação da realidade. Embora o *chronon* do sistema seja único, é possível manipular estas diferentes granularidades através de funções e operações

disponíveis nos gerenciadores do banco de dados que implementam o modelo.

Duração temporal

Durações temporais podem ser basicamente de dois tipos, dependendo do contexto em que são definidas: fixas e variáveis. Uma duração fixa independe do contexto de sua definição. Um exemplo típico de uma duração fixa é uma hora que tem sempre, independentemente do contexto de sua utilização, a duração de 60 minutos. Já a duração variável depende do contexto, sendo um exemplo típico a duração de um mês, que pode ser de 28, 29, 30 ou 31 dias.

Representação temporal explícita e implícita

A definição de tempo pode ser feita de forma explícita, através por exemplo da associação de um instante de tempo a uma informação, isto é, a informação recebe um rótulo temporal (*timestamping*), ou de forma implícita através da utilização de uma linguagem de lógica temporal. Esta segunda forma de representação é feita através da manipulação de conhecimentos sobre a ocorrência de eventos ou do relacionamento de intervalos de tempo como, por exemplo: a aula de lógica temporal ocorreu ontem. Para a representação explícita de tempo é necessária a definição de um elemento temporal primitivo, como instante ou intervalo.

2.4 Classificação dos bancos de dados quanto ao suporte temporal

Um banco de dados temporal armazena vários estados dos dados, assim como os instantes em que estes diferentes estados são válidos (SNODGRASS, 1995). Segundo AL (1994), conforme a forma utilizada para armazenar valores temporais, os bancos de dados podem ser classificados em quatro tipos diferentes:

- banco de dados instantâneos;
- banco de dados de tempo de transação;
- banco de dados de tempo de validade;
- banco de dados bitemporais.

2.4.1 Banco de dados instantâneos

Este tipo de banco de dados é simplesmente um banco de dados convencional, tal como um banco de dados relacional que não apresenta suporte para aspectos temporais (Figura 2.1). Os únicos valores perceptíveis são os valores presentes. Cada modificação no valor de uma propriedade pode ser percebida como uma transição do banco de dados. Em uma transição o valor anteriormente armazenado é destruído e somente o último valor está disponível. O estado atual do banco de dados, composto pelos valores atuais de todas as propriedades, é o único existente. Os estados anteriores são perdidos, com exceção do registro das transações anteriores feito para permitir a recuperação de erros de processamento ("log"). Entretanto esta informação é utilizada apenas para procedimentos operacionais não sendo acessível à linguagem de manipulação de dados.

Em bancos de dados instantâneos, a manipulação dos dados temporais só é possível através da definição de um complexo modelo lógico de tabelas e atributos sendo manipulado por um conjunto de programas de aplicação onde toda a semântica associada ao tempo deverá estar contida.

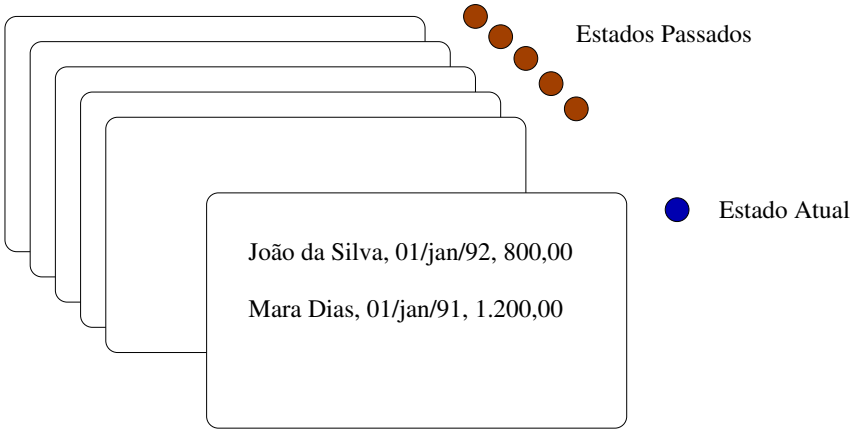

Figura 2.1: Banco de dados instantâneo (adaptado de Edelweiss e Oliveira (1994)).

2.4.2 Banco de dados de tempo de transação

Neste tipo de banco de dados, o valor definido é associado ao instante temporal em que foi realizada a transação, sob forma de um rótulo temporal (*timestamp*). Uma relação que utilize esta abordagem pode ser vista como tendo três dimensões (tuplas, atributos, tempo de transação).

Os bancos de dados de tempo de transação permitem a recuperação de informações definidas em algum instante do passado, pois todos os dados passados estão armazenados associados ao seu instante de definição (tempo de transação). Os bancos de dados de tempo de transação possuem uma limitação, pois armazenam a história das transações e não história do mundo real. Sendo assim, um tempo de transação futuro é impossível de ser representado. Uma tupla passa a ser válida no momento em que é inserida na base de dados. O tempo de transação é contínuo. Alterações retroativas não podem ser executadas, bem como erros em tuplas passadas não poderão ser corrigidos (Figura 2.2).

2.4.3 Banco de dados de tempo de validade

Bancos de dados de tempo de validade trabalham com o tempo que uma informação é verdadeira no mundo real, ou seja, a cada informação é associado o seu tempo de validade. Este tempo deve ser fornecido pelo usuário ou, se não informado, o tempo corrente é associado. Este tipo de banco de dados registra a história relativa aos dados e não às transações. Permite a correção de erros através da modificação de dados. Portanto, dados errados não são registrados. Este tipo de banco de dados admite intervalos de tempo não válidos. Os bancos de dados de tempo de validade também permitem a recuperação de informações passadas, porém possuem a vantagem de representar melhor a semântica da realidade em relação aos bancos de dados de tempo de transação. Permitem a recuperação de informações válidas em momentos passados, presente e futuros (Figura 2.3).

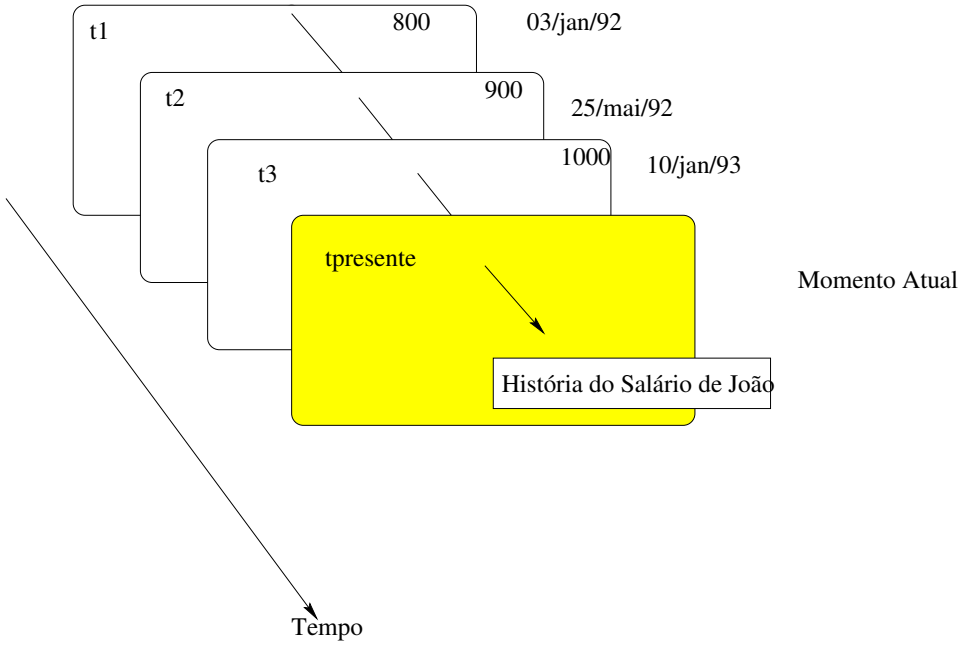

Figura 2.2: Bando de dados de tempo de transação (adaptado de Edelweiss e Oliveira (1994)).

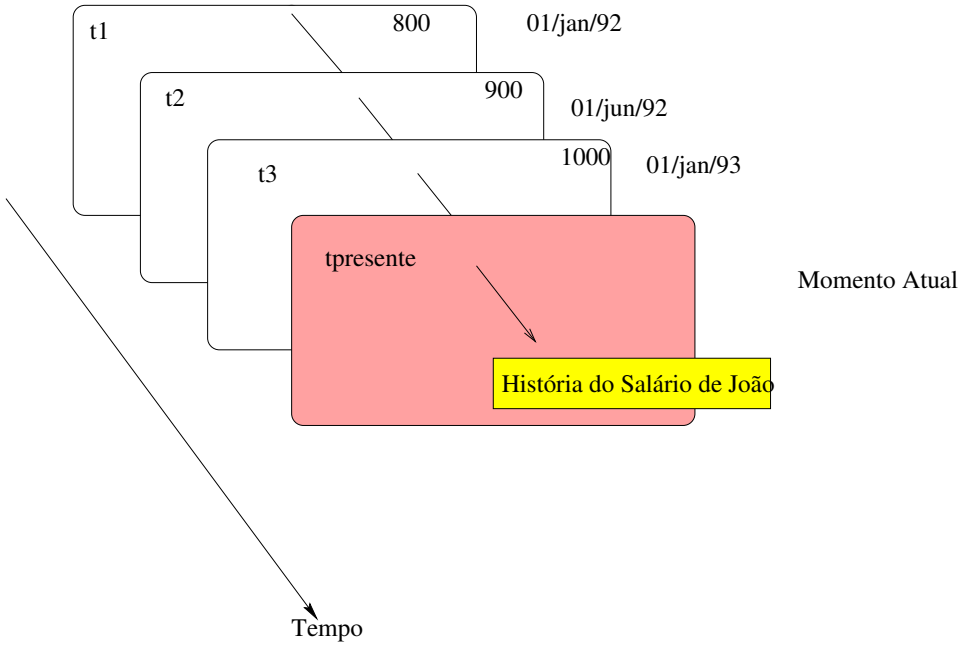

Figura 2.3: Banco de dados de tempo de validade (adaptado de Edelweiss e Oliveira (1994)).

2.4.4 Banco de dados bitemporais

Neste tipo, o banco de dados combina as propriedades do banco de dados de tempo de transação e de tempo de validade, ou seja, ele trata as duas dimensões de tempo. Toda a história do banco de dados é armazenada. É possível ter acesso a todos os estados passados do banco de dados, tanto à história das transações realizadas como à história da validade dos dados (Figura 2.4). O estado atual do banco de dados é constituído pelos valores atualmente válidos. Valores futuros podem ser definidos através do tempo de validade, sendo possível recuperar o momento em que estes valores foram definidos para eventuais alterações. Pela classificação de AL (1994), os bancos de dados de tempo de transação fornecem uma solução automatizada da parcela temporal do banco de dados, pois não necessitam da intervenção do usuário para fornecer o tempo. Entretanto, os bancos de dados bitemporais são a forma mais completa de armazenar informações temporais. A linguagem TSQL2, por exemplo, apresenta suporte para ambos tempos de transação e validade.

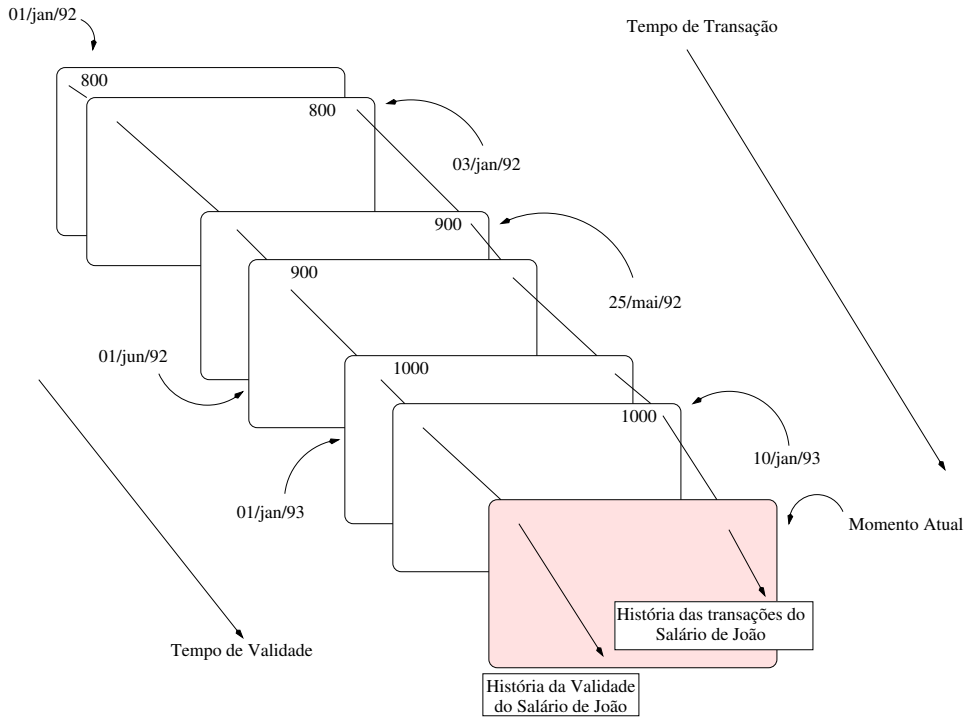

Figura 2.4: Banco de dados bitemporal (adaptado de Edelweiss e Oliveira (1994)).

3 EXTENSÕES TEMPORAIS DA SQL

Este capítulo apresenta um panorama da linguagem TSQL2 juntamente com alguns exemplos de uso da mesma. Descreve como foi o surgimento da linguagem ATSQL2, bem como o algoritmo de tradução de comandos ATSQL2 para SQL. Finalmente, conclui explicando os motivos pelos quais a linguagem ATSQL2 foi escolhida para utilização como linguagem de consulta na extensão temporal de OASIS.

3.1 TSQL2

A definição da linguagem TSQL2 (SNODGRASS, 1995) se deu através do esforço da comunidade de banco de dados temporais em definir uma linguagem padrão para tratar os aspectos temporais das aplicações em banco de dados. Primeiramente, foram definidas características que esta linguagem deveria suportar e, a partir destas, conduziu-se o projeto de construção até a especificação da linguagem.

A TSQL2, é antes de tudo, uma linguagem de consulta baseada na abordagem relacional. Em geral, ela é consistente com o padrão SQL-92 e tanto quanto possível com o SQL3. Não pretendendo ser um novo padrão, a TSQL2 está focalizada em proporcionar uma base comum para futuras pesquisas em bancos de dados temporais.

A linguagem TSQL2 é um *design* de linguagem. Por esta razão, ela não define aspectos, tais como, estruturas de armazenamento, índices, métodos de acesso, interfaces de quarta geração, suporte para sistemas distribuídos ou bases de dados heterogêneas, bem como técnicas de otimização. Algumas das principais características da TSQL2 são:

- suporte para tempo de transação e para tempo de validade;
- para simplicidade, rótulos temporais (*timestamps*) devem ser aplicados nas tuplas;
- deve ser baseada em tuplas homogêneas, isto é, cada atributo da tupla deve estar associado diretamente às propriedades temporais desta tupla (EDELWEISS; OLIVEIRA, 1994);
- o tempo de validade deve suportar tempos no passado e no futuro;
- deve ser totalmente compatível com o SQL-92;
- deve permitir a reestruturação de tabelas em qualquer conjunto de atributos;

- deve permitir projeção temporal com flexibilidade;

- o suporte temporal deve ser opcional em cada tabela, ou seja, tabelas que não são especificadas como temporais devem ser consideradas como tabelas instantâneas, mantendo, desta forma, grande compatibilidade com o SQL-92;

- o suporte ao tempo definido pelo usuário deve incluir instantes, períodos e intervalos;

- suporte a múltiplos calendários e múltiplas linguagens deve estar presente em rótulos temporais de entrada e saída, bem como operações com rótulos temporais;

- deve ser possível derivar uma tabela temporal ou não-temporal a partir de uma tabela temporal ou não-temporal.

3.1.1 Ontologia do tempo

O modelo de tempo da TSQL2 é que decide se o tempo é contínuo, denso ou discreto, não permitindo ao usuário diferenciá-los. Ao invés disso, o modelo acomoda as três alternativas assumindo que um instante em uma linha de tempo é muito menor que um *chronon*, o qual é a menor entidade que um rótulo temporal pode representar, o tamanho do *chronon* é dependente de implementação. Um instante pode apenas ser aproximadamente representado. Uma imagem discreta de tempos representados surge em tempo de execução como rótulos temporais com granularidades especificadas pelo usuário e as operações nesses rótulos temporais são realizadas em uma escala determinada. Um instante é modelado por um rótulo temporal associado a uma escala, tal como dias, meses e anos. Um período é modelado pela composição de dois instantes e com a restrição de que o instante que inicia o período é igual ou precede o instante que termina o período.

3.1.2 Tipos de dados

Os tipos de dados *datetime* e *interval* do SQL-92 são substituídos com maior precisão pelos tipos *instants*, *intervals*, e *spans* de tamanho e precisão especificáveis. O tipo de dado *surrogate* é utilizado na TSQL2 como identificador único para objetos que possuem atributos temporais, mas não substitui a chave da tabela. Os tipos de dados *surrogate* podem ser comparados por igualdade, os seus valores não podem ser vistos pelos usuários.

3.1.3 Linhas de tempo

Segundo Simoneto (1998), a TSQL2 suporta três linhas de tempo: tempo definido pelo usuário, tempo de validade e tempo de transação. O tempo de transação é limitado por *inception*, quando o banco de dados é criado, e *until changed*, quando ocorre uma alteração na base de dados. O tempo de validade possui os atributos *beginning* e *forever*, que são o início e o fim do tempo de validade. O tempo de transação possui os atributos *inception* e *until changed*, que são o início e o fim do tempo de transação.

3.1.4 Tabelas de tempo de validade

Em cada tabela, cada tupla é rotulada com um elemento temporal, o qual é composto por todos os intervalos temporais (SIMONETO, 1998). Como exemplo, consideremos a tabela empregados com os atributos nome, salário e gerente contendo a seguinte tupla (Antônio, 10000, Ana). O elemento temporal rotulado neste registro será o conjunto de intervalos, nos quais, Antônio irá receber R$10.000, tendo Ana como sua gerente. Informações sobre outros salários e outros gerentes de Antônio serão armazenadas em outras tuplas. O rótulo temporal é implicitamente associado com cada tupla (Tabela 3.1).

Tabela 3.1: Representação do tempo de validade na TSQL2 (Adaptado de Simoneto (1998)).

Nome	Salario	Gerente	Beggining	Forever
Ana	20000		01/01/1991	-
Antônio	10000	Ana	01/01/1992	31/12/1999
Antônio	15000	Ana	31/12/1999	-

3.1.5 Tabelas de tempo de transação e bitemporais

O tempo de transação pode ser associado em tabelas ortogonalmente em relação ao tempo de validade. O tempo de transação de uma tupla, o qual é um elemento temporal, especifica quando a tupla é logicamente armazenada no banco de dados (SIMONETO, 1998). Se uma determinada tupla foi armazenada no banco de dados em 20 de janeiro de 1992 (com uma sentença *APPEND*) e removida do banco de dados no dia 15 de janeiro de 2000 (com uma sentença *DELETE*), então o tempo de transação da tupla é o intervalo compreendido entre os dias 20 de janeiro de 1992 e 15 de janeiro de 2000. O rótulo temporal do tempo de transação tem tamanho e precisão dependentes de implementação (Tabela 3.2). Quando as linhas de tempo validade e tempo de transação são representadas em uma mesma tabela, esta tabela é classificada como bitemporal (Tabela 3.3).

Tabela 3.2: Representação do tempo de transação na TSQL2 (Adaptado de Simoneto (1998)).

Nome	Salario	Gerente	Inception	Until Changed
Ana	20000		15/01/1991	-
Antônio	10000	Ana	20/01/1992	15/01/2000
Antônio	15000	Ana	15/01/2000	-

Tabela 3.3: Representação do tempo de transação e de validade (bitemporal) na TSQL2 (Adaptado de Simoneto (1998)).

Nome	Salario	Gerente	Beggining	Forever	Inception	Until Changed
Ana	20000		01/01/1991	-	15/01/1991	-
Antônio	10000	Ana	01/01/1992	31/12/1999	20/01/1992	15/01/2000
Antônio	15000	Ana	31/12/1999	-	15/01/2000	-

Muitos aspectos da TSQL2 são puras extensões da SQL-92 para tratar o aspecto tempo de determinadas aplicações, segundo Snodgrass (1995), "estas extensões constituem um estrito superconjunto da SQL-92".

3.1.6 Tabelas evento

A TSQL2 também permite a especificação de tabelas evento. Em tais tabelas, cada tupla é rotulada com um instante fixo. Como exemplo, uma tabela que armazena as contratações dos empregados com a seguinte tupla (Ana, Gerente) possui o rótulo temporal, implicitamente associado, do instante em que Ana foi contratada como gerente. A informação sobre outras contratações de Ana devem ser registradas em outras tuplas. As tabelas evento também podem ser bitemporais, armazenando tanto o tempo de transação quanto o tempo de validade.

3.1.7 Considerações sobre tabelas

Em síntese, existem seis tipos de tabelas em TSQL2: tabelas instantâneas (sem suporte temporal), tabelas de tempo de validade (consistindo de conjuntos de tuplas rotuladas temporalmente com tempos de validade), tabelas evento de tempo validade (rotuladas temporalmente com instantes de tempos de validade), tabelas de tempo de transação (rotuladas temporalmente com elementos de tempos de transação), tabelas bitemporais (rotuladas temporalmente com elementos bitemporais), e tabelas evento bitemporais (rotuladas temporalmente com instantes bitemporais).

3.1.8 Indeterminância temporal

Muitas vezes, não se conhece com precisão quando um evento acontece. Alguns exemplos de indeterminância temporal são os seguintes: ¨ ... entre duas e quatro horas da tarde ...¨ , ¨ ... primeira semana de junho ...¨ ¨e ¨... meados do mês de agosto ...¨ .

A seguir, serão apresentadas as definições de instante, período, intervalo e elementos temporais indeterminados.

Instante indeterminado

Um instante é *determinado* quando sua localização (ponto no tempo) é conhecida. Se a localização do instante é desconhecida, então o instante é *indeterminado*. A indeterminância está relacionada com a localização do instante e não com a existência do mesmo. Em outras palavras, a existência do instante deve ser conhecida (o instante deve existir) e a sua localização deve ser desconhecida.

Período indeterminado

Um período delimitado por instantes indeterminados é chamado de período indeterminado. Um período indeterminado pode iniciar durante qualquer membro do conjunto de possíveis instantes do instante inicial. Da mesma forma, o período indeterminado pode terminar durante qualquer membro do conjunto de possíveis instantes do instante final. Uma vez que o instante inicial e o instante final são conhecidos imprecisamente, é desconhecido quando um período indeterminado inicia ou termina.

Um período indeterminado representa um conjunto de possíveis períodos, um

dos quais é o período atual, porém desconhecido. O conjunto de possíveis períodos consiste de muitas combinações de possíveis instantes de início e de término. Um único período possível é obtido pela escolha de um membro de um conjunto possível de instantes para cada instante delimitador.

Intervalo indeterminado

Um intervalo determinado é uma duração conhecida de tempo, que é uma soma de instantes. Um intervalo indeterminado, por sua vez, é uma duração imprecisa que descreve o conjunto de durações possíveis. Um intervalo indeterminado é representado por um número impreciso de instantes.

Elemento indeterminado

Assim como um elemento temporal determinado é um conjunto de períodos determinados, um elemento temporal indeterminado é um conjunto de períodos indeterminados.

3.1.9 Exemplo de uso da TSQL2

Para uma boa compreensão do poder de expressão da TSQL2 é necessário a apresentação de uma série de exemplos de comandos de definição e manipulação de dados da linguagem juntamente com comentários explicativos. A seguir, será apresentado um pequeno esquema de banco de dados utilizando a TSQL2. O exemplo foi adaptado de Snodgrass (1995). No esquema existem três entidades chamadas *Emp* (com informações sobre os empregados, *Hab* (com informações sobre as habilidades dos empregados) e *Dep* (com informações sobre os departamentos. A definição da tabela *Emp* é a seguinte:

```
SET SCALE AS INTERVAL "1" DAY

CREATE TABLE Emp (
    Id          SURROGATE NOT NULL,
    Nome        CHARACTER (30) NOT NULL,
        PRIMARY KEY(Name),
    Salario     DECIMAL(8,2),
    Sexo        CHARACTER(1),
    Nascimento  DATE,
    NomeDepto   CHARACTER (30)
        FOREIGN KEY (NomeDepto) REFERENCES Dep(Nome)
)
AS VALID STATE
```

No código DDL acima, o comando *SET SCALE* determina que a granularidade será de um dia, ou seja, somente a última atualização da mesma tupla no mesmo dia será considerada no registro da informação temporal.

No comando *CREATE TABLE*, o atributo *Id* é definido com o tipo de dado *SURROGATE* para identificar univocamente cada tupla da tabela histórica *Emp*. Tal como o atributo *Nome*, o campo *Salário* também pode variar no tempo. Os atributos *Sexo* e *Nascimento*, no entanto, não podem sofrer variação no tempo, pois armazenam os valores "M" ou "F" e a data de nascimento respectivamente. Essa restrição é dada pelo mundo real, pois nada impede que os atributos mencionados sofram variação no tempo caso tenham seus valores alterados devido a um erro de digitação que está sendo corrigido.

A cláusula "*AS VALID STATE*" significa que a tabela é de tempo de validade e se aplica a todos os atributos da tabela.

O atributo *NomeDepto* é a implementação da chave estrangeira que referencia a tabela de Departamentos (*Dep*), cuja definição é apresentada a seguir:

```
CREATE TABLE Dep (
   Nome     CHARACTER (30) NOT NULL,
        PRIMARY KEY (Name),
   Orcam    DECIMAL (9,0),
   IdGer    SURROGATE NOT NULL,
        FOREIGN KEY (IdGer) REFERENCES Emp (Id)
)
AS VALID STATE
```

Na definição da tabela *Dep* acima, a coluna *Nome* é definida como chave primária e não sofrerá variação temporal. O atributo *Orcam*, por sua vez, armazena o valor do orçamento para um determinado setor e pode sofrer variação temporal. Finalmente, a coluna *IdGer* é definida como *SURROGATE,* pois referencia a coluna *Id* da tabela *Emp*, cujo tipo de dado é *SURROGATE.* O relacionamento através de colunas do tipo *SURROGATE* é necessário para preservar a variação temporal do ponto de vista do relacionamento entre as tabelas, pois desse modo a referência poderá efetuar a junção da linha correta da tabela referenciada considerando a variação temporal da mesma. Este mesmo aspecto da modelagem é encontrado na tabela *Hab* cuja definição é apresentada a seguir:

```
CREATE TABLE Hab
    EmpId   SURROGATE NOT NULL,
        FOREIGN~KEY (EmpId) REFERENCES Emp (Id),
    Nome    CHARACTER (30) NOT NULL
)
AS VALID STATE
```

A coluna *EmpId* é definida como *SURROGATE* devido aos mesmos motivos descritos anteriormente com relação à tabela *Dep.* O atributo *Nome* armazena o nome da habilidade possuída pelos empregados durante o tempo, ou seja, para cada nova habilidade desenvolvida por cada empregado, uma nova entrada será acrescentada nesta tabela.

Nos exemplos que se seguirão, existem dois empregados, o primeiro identificado por *ED* e o segundo identificado por *DI.* Os acontecimentos do mundo real relacionados com o empregado *ED*, para efeito de exemplificação, são os seguintes:

1. o empregado *ED* trabalhou no departamento de Brinquedos de 1/2/82 até 31/1/87, e no departamento de Livros de 1/4/87 até o dia de hoje;

2. de 1/4/87 até hoje, ele gerenciou o departamento de Livros e o orçamento do departamento é de R$ 50.000 desde que *ED* tornou-se o gerente;

3. seu nome foi Ed de 1/2/82 até 21/12/87, e Eduardo de 1/1/88 até o presente momento;

4. seu salário foi de R$ 20.000 de 1/2/82 até 31/5/82, depois R$ 30.000 de 1/6/82 até 31/1/85, depois R$ 40.000 de 1/2/85 até 31/1/87 e 1/4/87 até o presente;

5. *ED* é do sexo masculino e nascido em 1/7/55;

6. muitas habilidades são registradas por *ED*. Ele qualificou-se para digitador desde 1/4/82 e qualificado para arquivista desde 1/1/85;

7. ele esteve qualificado para dirigir de 1/1/82 até 1/5/82 e de 1/6/84 até 31/5/88.

Para registrar todas as informações descritas acima são apresentadas as seguintes operações de modificação do banco de dados:

```
INSERT INTO Emp
VALUES (NEW,'Ed', 20000, 'M', DATE '1/7/1955', 'Brinquedos')
VALID~PERIOD '[1/2/1982 - 31/1/1987]'

INSERT INTO Emp
    SELECT (ID, 'Ed', 40000, 'M', DATE '1/7/1955', 'Livros')
    VALID PERIOD '[1/4/1987 - now]'
    FROM Emp
    WHERE Nome = 'Ed'

INSERT INTO Dep
    SELECT ('Livros', 50000, Emp.Id)
    VALID PERIOD '[1/4/1987 - now]'
    FROM Emp
    WHERE Emp.Nome = 'Ed'

UPDATE Emp
    SET Nome TO 'Eduardo' VALID PERIOD '[1/1/1988 - now]'
    WHERE Id = (SELECT DISTINCT Id FROM Emp WHERE Nome = 'Ed')

UPDATE Emp
    SET Salario TO 30000 VALID PERIOD '[1/6/1982 - 31/1/1985]'
    WHERE Id = (SELECT DISTINCT Id FROM Emp WHERE Nome = 'Ed')

UPDATE Emp
    SET Salario TO 40000 VALID PERIOD '[1/2/1985 - 31/1/1987]'
    WHERE Id = (SELECT DISTINCT Id FROM Emp WHERE Nome = 'Ed')

INSERT INTO Hab
    SELECT (Id, 'Digitador')
    VALID PERIOD '[1/4/1982 - now]'
    FROM Emp
    WHERE Nome = 'Ed'

INSERT INTO Hab
    SELECT (Id, 'Arquivista')
    VALID PERIOD '[1/1/1985 - now]'
    FROM Emp
    WHERE Nome = 'Ed'

INSERT INTO Hab
    SELECT (Id, 'Motorista')
        VALID (PERIOD '[1/1/1982 - 5/1/1982]'
            + PERIOD '[1/6/1984 - 31/5/1988]')
    FROM Emp
    WHERE Nome = 'Ed'
```

Na seqüencia de operações acima, pode-se observar, no primeiro comando, a atribuição da função *NEW* para a coluna *Id*, cujo tipo de dado é *SURROGATE*. A palavra *NEW* retorna um novo valor para identificar univocamente a linha que está

sendo inserida. Na segunda operação, no entanto, é aproveitado o mesmo valor da coluna *Id* inserida no primeiro comando.

Em todos os comandos existe a cláusula *VALID PERIOD* que é a maneira pela qual são informados os períodos de validade. A palavra *now* que aparece no valor final dos períodos de validade indica que o mesmo ainda não está encerrado, isto é, representa sempre o momento presente.

No último exemplo da seqüencia de operações observa-se a atribuição de dois períodos de validade disjuntos. Neste caso, tem-se o chamado *elemento temporal,* o qual foi apresentado no capítulo 2.

Dando continuação aos exemplos, será apresentada uma seqüência de acontecimentos do mundo real, relacionados com o empregado *DI,* para posterior atualização do banco de dados:

1. *DI* trabalhou e gerenciou o departamento de brinquedos de 1/1/82 até o presente;

2. o orçamento do departamento de brinquedos foi de R$ 150.000 de 1/1/82 até 31/7/84, R$ 200.000 de 1/8/84 até 31/12/86, e R$ 100.000 de 1/1/87 até hoje;

3. o salário de *DI* foi R$ 30.000 de 1/1/82 até 31/7/84, R$ 40.000 de 1/8/84 até 31/8/86, e R$ 50.000 de 1/9/86 até hoje;

4. *DI* é do sexo feminino nascida em 1/10/60;

5. finalmente, *DI* foi qualificada para o cargo de diretora de 1/1/82 até o presente momento.

Para registrar todas as informações descritas sobre *DI*, as seguintes operações de modificação do banco de dados são necessárias:

```
 INSERT INTO Emp
 VALUES (NEW,  'Di', 30000, 'F', DATE '10/1/1960', 'Brinquedos')
 VALID PERIOD  '[1/1/1982 - now]'

INSERT INTO Dep
    SELECT ( 'Brinquedos', 150000, EmpId)
        VALID PERIOD  '[1/1/1982 - 31/7/1984]'
        FROM Emp
        WHERE Emp.Nome = 'Di'

UPDATE Dep
    SET Orcam TO 200000 VALID PERIOD  '[1/8/1984 - 31/12/1986]'
    WHERE Nome = 'Brinquedos'

UPDATE Dep
    SET Orcam TO 100000 VALID PERIOD  '[1/1/1987 - now]'
    WHERE Nome = 'Brinquedos'

UPDATE Emp
    SET Salario TO 40000 VALID PERIOD  '[1/8/1984 - 31/8/1986]'
    WHERE Id = (SELECT DISTINCT Id FROM Emp WHERE Nome = 'Di')

UPDATE Emp
    SET Salario TO 50000 VALID PERIOD  '[1/9/1986 - now]'
    WHERE Id = (SELECT DISTINCT Id FROM Emp WHERE Nome = 'Di')
```

```
INSERT INTO Hab
   SELECT (Id, 'Direcao')
       VALID PERIOD  '1/1/1982 - now]'
       FROM Emp
       WHERE Id = SELECT Id FROM Emp WHERE Nome = 'Di')
```

Todos os comandos de modificação exemplificados até aqui produzem as tabelas apresentadas a seguir.

A tabela *Emp:*

Id	Nome	Salario	Sexo	Nascimento	NomeDepto	Validade
ED	Ed	20000	M	1/7/1955	Brinquedos	{[1/2/1982 - 31/5/1982]}
ED	Ed	30000	M	1/7/1955	Brinquedos	{[1/6/1982 - 31/1/1985]}
ED	Ed	40000	M	1/7/1955	Brinquedos	{[1/2/1985 - 31/1/1987]}
ED	Ed	40000	M	1/7/1955	Livros	{[1/4/1987 - 31/12/1987]}
ED	Eduardo	40000	M	1/7/1955	Livros	{[1/1/1988 - *now*]}
DI	Di	30000	F	1/10/1960	Brinquedos	{[1/1/1982 - 31/7/1984]}
DI	Di	40000	F	1/10/1960	Brinquedos	{[1/8/1984 - 31/8/1986]}
DI	Di	50000	F	1/10/1960	Brinquedos	{[1/9/1986 - *now*]}

A tabela *Hab*:

EmpId	Nome	Validade
ED	Digitador	{[1/4/1982 - *now*]}
ED	Arquivista	{[1/1/1985 - *now*]}
ED	Motorista	{[1/1/1982 - 1/5/1982]}
		{[1/6/1984 - 31/5/1988]}
DI	Direcao	{[1/1/1982 - *now*]}

A tabela *Dep*:

Nome	Orcam	IdGer	Validade
Brinquedos	150000	DI	{[1/1/1982 - 31/7/1984]}
Brinquedos	200000	DI	{[1/8/1984 - 31/12/1986]}
Brinquedos	100000	DI	{[1/1/1987 - *now*]}
Livros	50000	ED	{[1/4/1987 - *now*]}

A seguir são apresentados alguns exemplos de consultas em TSQL2. Alguns construtores da linguagem utilizados nos exemplos não existem na SQL:

- *SNAPSHOT*: determina que a projeção dos rótulos temporais não deve ser efetuada;
- *CAST*: retorna a granularidade desejada a partir de um rótulo temporal ou a quantidade de *chronons* se especificado como *INTERVAL*;
- (*PERIOD*): efetua a operação de coalescência das linhas com valores de atributos idênticos reunindo-as em uma única linha com os tempos de início e fim maximais dos rótulos temporais.

Consulta 1: Quais departamentos tiveram os gerentes que trabalharam pelo menor período contínuo de tempo?

```
SELECT SNAPSHOT Nome
FROM Dep(Nome, IdGer)(PERIOD) AS D
WHERE CAST(VALID(D) AS INTERVAL DAY)
     <= ALL (SELECT CAST(VALID(D2) AS INTERVAL DAY)
             FROM Dep(Nome, IdGer)(PERIOD) AS D2
             WHERE D2.Nome = D.Nome

Resposta: {('Livros')}
```

Consulta 2: Quem trabalhou no departamento de Brinquedos por um período contínuo pelo mesmo tempo que o empregado *Di*, no mínimo?

```
SELECT SNAPSHOT E4.Nome
FROM Emp(Id, NomeDepto)(PERIOD) AS E1 E2, E1(Nome) AS E3,
     E2(Nome) AS E4
WHERE E3.Nome = 'Di' AND E1.NomeDepto = 'Brinquedos'
     AND E2.NomeDepto = 'Brinquedos'
     AND CAST(VALID(E2) AS INTERVAL DAY)
           > CAST(VALID(E1) AS INTERVAL DAY)

Resposta: {('Ed'),('Edward')}
```

Consulta 3: Quem teve o mesmo salário por um período contínuo de tempo mais longo?

```
SELECT SNAPSHOT Nome
FROM Emp(Id, Salario)(PERIOD) AS E, E(Nome) AS E2
WHERE CAST(VALID(E) AS INTERVAL DAY)
      > ALL (SELECT CAST(VALID(E3) AS INTERVAL DAY)
             FROM Emp(Id, Salario)(PERIOD) AS E3
             WHERE E3.Id <> E.Id)

Resposta: {('Di')}
```

Consulta 4: Quem trabalhou por um período tão longo quanto o gerente do departamento?

```
SELECT SNAPSHOT E2.Nome
FROM Emp(Id, NomeDepto) AS E, Dep(Nome, IdGer)(PERIOD) AS D,
       E(Nome) AS E2
WHERE E.NomeDepto = D.Nome AND VALID(E) CONTAINS VALID(D)

Resposta: {('Ed'),('Edward'),('Di')}
```

Consulta 5: Quem recebeu o mesmo salário pelo maior tempo total?

```
SELECT SNAPSHOT E2.Nome
FROM Emp(Id, Salario) AS E1, Emp(Id, Nome) AS E2
     (SELECT MAX(CAST(VALID(E) AS INTERVAL DAY)
      FROM Emp(Id, Salario)(PERIOD) AS E) AS E3
WHERE E2.Id = E1.Id

Resposta: {('Ed')}
```

Consulta 6: Encontrar os salários de *ED* quando ele trabalhou no mesmo departamento de *D*I.

```
SELECT SNAPSHOT E3.Salario
FROM Emp(Id, NomeDepto) AS E1 E2, E1(Salario) AS E3
WHERE E1.Id = 'Ed' AND E2.Id = 'Di'
   AND E1.NomeDepto = E2.NomeDepto

Resposta: {(20000),(30000),(40000)}
```

Consulta 7: Encontrar as habilidades que *ED* adquiriu quando ele trabalhou no departamento de Brinquedos.

```
SELECT SNAPSHOT S1.Nome
FROM Emp(Id, NomeDepto) AS E1, Hab(EmpId, Nome) S1
WHERE E1.Id = 'Ed' AND E1.NomeDepto = 'Brinquedos' AND
    E1.Id = S1.EmpId

Resposta: {('Motorista'),('Arquivista'),('Digitador')}
```

Consulta 8: Qual foi o orçamento do departamento de Brinquedos quando *DI* era gerente?

```
SELECT SNAPSHOT D1.Orcam
FROM Dept(Nome, IdGer, Orcam) AS D1, Emp(Id, Nome) AS G
WHERE D1.IdGer = G.Id AND G.Nome = 'Di' AND
    D1.Nome = 'Brinquedos'

Resposta: {(150000),(200000),(100000)}
```

Consulta 9: Encontrar os tempos de início do tempo de validade de gerentes os quais gerenciaram um departamento por mais que 5 anos.

```
SELECT BEGIN(VALID(D))
FROM Dept(IdGer, Nome)(PERIOD) AS D
WHERE CAST(VALID(D) AS INTERVAL YEAR)
            > INTERVAL '5' YEAR

Resposta: {(1/1/1982)}
```

3.2 ATSQL2

A linguagem ATSQL2 (BÖHLEN; JENSEN; SNODGRASS, 1995; SNODGRASS et al., 1996a,b) é baseada no padrão SQL (MELTON; SIMON, 1993). Ela permite a escrita de comandos de consulta e modificação bitemporais, bem como a definição de dados e a especificação de restrições de integridade.

No seção anterior a linguagem TSQL2 foi apresentada. A linguagem TSQL2 foi a primeira a especificar um padrão para uma linguagem de consulta temporal. Muitos pesquisadores da área de banco de dados temporais contribuíram no desenvolvimento dessa linguagem, que é uma extensão da SQL. O foco da TSQL2 é consolidar diferentes abordagens de modelos de dados temporais com o objetivo de

criar uma linguagem de consulta temporal, que associada a um modelo de dados, venha a ser um consenso entre os pesquisadores, para que futuros trabalhos possam ser desenvolvidos. A linguagem ATSQL2, por sua vez, é um dos frutos dessa idéia. No ano de 1996, a ATSQL2 foi proposta para o padrão SQL/Temporal nos comitês ANSI (*American National Standards Institute*) e ISO (*International Organization for Standardization*) que tratam da padronização da SQL3. As duas propostas (SNODGRASS et al., 1996a,b), foram aceitas por unanimidade pelo comitê ANSI e repassadas para o comitê ISO.

3.2.1 A linguagem ATSQL2

Assim como a linguagem TSQL2, a linguagem ATSQL2 é uma extensão da SQL e possui os seguintes focos principais:

- oferecer o máximo suporte para migrar bases de dados não-temporais para bases de dados temporais. A noção de *compatibilidade ascendente* (BÖHLEN; JENSEN; SNODGRASS, 1995) e *compatibilidade ascendente temporal* (SNODGRASS et al., 1996b) descrevem os requisitos para a migração de uma base de dados. A *compatibilidade ascendente* demanda que a SQL é um subconjunto da ATSQL2. Assim, muitos comandos SQL devem ser usados com a mesma semântica em ATSQL2. A *compatibilidade ascendente temporal*, por sua vez, especifica os requisitos para a linguagem de manipulação e consulta da ATSQL2 após a migração da base de dados ter sido efetuada. Cada comando SQL, executado em uma base de dados temporal, deverá apresentar o mesmo resultado, tal como em uma base de dados não-temporal;

- tornar a transição da SQL para a ATSQL2 fácil para os programadores. Os comandos de manipulação e consulta temporais ou não-temporais devem ser sintaticamente similares. De acordo com a definição de *integralidade temporal* (BÖHLEN; MARTI, 1994), duas consultas q e q^t são sintaticamente similares se existirem duas *strings* possivelmente vazias S_1 e S_2 tal que $q^t = S_1\ q\ S_2$. Isto permite ao programador escrever um comando SQL não-temporal e transformá-lo em temporal simplesmente acrescentando uma palavra na frente da consulta não-temporal.

Os exemplos desta subseção foram adaptados de Steiner (1998).

Compatibilidade ascendente

Como apresentado na seção anterior, *compatibilidade ascendente* demanda que qualquer comando SQL e tabela não-temporais possam ser usados em ATSQL2. A vantagem dessa abordagem é que dados e código legados podem continuar sendo utilizados durante a migração de uma base de dados não-temporal para uma base de dados temporal.

Como exemplo, consideremos a criação da seguinte tabela não-temporal cuja finalidade é armazenar informações sobre os empregados de uma companhia:

```
CREATE TABLE Empregados (
EmpCod  INTEGER,
Nome    CHAR(30),
Salario INTEGER,
Gerente INTEGER);
```

```
INSERT INTO Empregados VALUES (111, 'João', 8700, 111);
INSERT INTO Empregados VALUES (112, 'Paulo', 9100, 111);
INSERT INTO Empregados VALUES (113, 'Jorge', 8300, 111);
```

Os comandos acima produzem a seguinte tabela:

EmpCod	Nome	Salario	Gerente
111	João	8700	111
112	Paulo	9100	111
113	Jorge	8300	111

Pode-se agora consultar os dados dos empregados que ganham mais de 9000 juntamente com os nomes de seus gerentes:

```
SELECT e1.EmpCod, e1.Nome, e1.Salario, e2.Nome Gerente
FROM Empregados e1, Empregados e2
WHERE e1.Gerente = e2.EmpCod
AND   e1.Salario > 9000;
```

Esta consulta retorna a seguinte tabela:

EmpCod	Nome	Salario	Gerente
112	Paulo	9100	João

Compatibilidade ascendente temporal

Quando os dados de uma tabela não-temporal são migrados para uma tabela temporal é desejável que os comandos SQL convencionais possam continuar sendo executados. Assim, comandos SQL são convertidos automaticamente em comandos ATSQL2. O comando SQL, uma vez reescrito em ATSQL2, recupera somente o último estado da base de dados.

Sendo assim, *compatibilidade ascendente temporal* demanda que qualquer consulta não-temporal q executada em uma base de dados temporal db^t fornece o mesmo resultado como se fosse executada em uma base de dados instantânea $db = \tau_{now}(db^t)$. Uma base de dados não-temporal armazena somente um estado da base de dados, normalmente aquele que representa o mais recente estado do mundo real. A base de dados temporal db^t, por sua vez, corresponde à seleção do estado da base de dados no instante *now*.

Como exemplo, em 21 de outubro de 1996, migrou-se a tabela *Empregados* apresentada no exemplo anterior para uma tabela de tempo de validade. Isto significa manter a história dos empregados com respeito ao tempo de validade desde 21 de outubro de 1996.

```
ALTER TABLE Empregados ADD VALID;
```

O comando acima altera o esquema e o conteúdo da tabela *Empregados* de modo que o tempo de validade é adicionado. O tempo de validade das tuplas da tabela *Empregados* é ajustado para [1996/10/21 - ∞). É possível utilizar os comandos de consulta e atualização de uma aplicação não-temporal em tabelas migradas. Consideremos que o salário de João foi alterado no dia 29 de outubro de 1996 com o seguinte comando SQL:

```
UPDATE Empregados SET Salario = 9300 WHERE Nome = 'João';
```

Após a modificação, a tabela de tempo de validade *Empregados* está assim:

VALID	EmpCod	Nome	Salario	Gerente
[1996/10/21 - ∞)	112	Paulo	9100	111
[1996/10/21 - ∞)	113	Jorge	8300	111
[1996/10/21 - 1996/10/29)	111	João	8700	111
[1996/10/29 - ∞)	111	João	9300	111

A mesma consulta não-temporal do exemplo anterior é executada no dia 29 de outubro de 1996:

```
SELECT e1.EmpCod, e1.Nome, e1.Salario, e2.Nome~Gerente
FROM Empregados e1, Empregados e2
WHERE e1.Gerente = e2.EmpCod
AND   e1.Salario > 9000;
```

A consulta acima retorna o seguinte resultado:

EmpCod	Nome	Salario	Gerente
112	Paulo	9100	João
111	João	9300	João

Para que os comandos SQL possam continuar apresentando resultados não-temporais quando aplicados a tabelas temporais, os mesmos também devem ser traduzidos. O comando que efetua a consulta não-temporal apresentado acima deve sofrer a seguinte modificação:

```
SELECT e1.EmpCod, e1.Nome, e1.Salario, e2.Nome Gerente
FROM Empregados e1, Empregados e2
WHERE e1.Gerente = e2.EmpCod
AND   e1.Salario > 9000
AND   e1.VTS <= SYSDATE AND e1.VTE > SYSDATE
AND   e2.VTS <= SYSDATE AND e2.VTE > SYSDATE;
```

Verifica-se que no comando SQL modificado foi acrescentado à cláusula *WHERE* mais dois critérios de seleção, um para cada tabela da cláusula *FROM* e que efetuam o teste dos tempos de início de fim do tempo de validade de cada tupla de cada tabela em relação ao momento presente (instante *NOW*):

- o instante inicial do tempo de validade (VTS) nunca deve ser maior que o instante *NOW*;
- o instante final do tempo de validade (VTE) sempre deve ser maior que o instante *NOW*.

Comandos temporais seqüenciados

Consultas seqüenciadas usam operações de álgebra temporal tendo a semântica redutível a um instantâneo. Isso significa que uma operação de álgebra temporal usa os rótulos temporais das tuplas que estão sendo selecionadas e faz a projeção automática dos mesmos na relação resultante.

O conceito de similaridade sintática descrito anteriormente é alcançado pela introdução das palavras *VALID* e *TRANSACTION,* as quais são escritas na frente de qualquer comando de consulta SQL. Essas palavras expressam qual dimensão temporal deve ser usada na consulta, ou seja, tempo de validade ou tempo de transação.

Caso as duas dimensões temporais devam ser utilizadas, podemos obter uma consulta bitemporal combinando as duas palavras: *VALID AND TRANSACTION.*

Como exemplo, consideremos a mesma consulta dos exemplos anteriores formulado agora como uma consulta seqüenciada pela simples adição da palavra *VALID* na frente do consulta não-temporal:

```
VALID
SELECT e1.EmpCod,e1.Nome, e1.Salario, e2.Nome Gerente}
FROM Empregados e1, Empregados e2
WHERE e1.Gerente = e2.EmpCod
AND   e1.Salario > 9000;
```

A consulta acima retorna a seguinte tabela:

VALID	EmpCod	Nome	Salario	Gerente
[1996/10/21 - 1996/10/29)	112	Paulo	9100	João
[1996/10/29 - ∞)	112	Paulo	9100	João
[1996/10/29 - ∞)	111	João	9300	João

A consulta seqüenciada acima retorna duas tuplas com períodos de tempo de validade adjacentes para o empregado Paulo. Isto é causado pelo produto cartesiano temporal da tabela *Empregados* com ela mesma o qual é necessário para encontrar o nome do gerente de cada empregado. João é gerente de Paulo e a alteração do salário de João em 29 de outubro de 1996 provocou a separação da informação sobre Paulo em duas tuplas.

A operação de produto cartesiano temporal calcula os períodos de tempo comuns de ambas tuplas envolvidas e retorna um único rótulo temporal para cada tupla resultante na junção.

Comandos temporais não-seqüenciados

Existe uma segunda classe de comandos temporais, os quais necessitam examinar diferentes estados da base de dados e obter um único estado como resultado. Tais comandos são chamados de não-seqüenciados. As consultas não-seqüenciadas tratam a informação sobre o tempo como qualquer outro atributo e não efetuam a projeção automática dos rótulos temporais. Entretanto, a informação sobre o tempo pode ser obtida através da projeção explícita dos tempos de início e/ou fim dos rótulos temporais nos comandos ATSQL2.

Para comparar diferentes estados da base de dados é necessário calcular o produto cartesiano das relações temporais envolvidas com uma condição de junção. Na consulta seqüenciada não é possível comparar os diferentes períodos de tempo das tuplas envolvidas, visto que um produto cartesiano com semântica seqüenciada resulta em uma tabela que contém um único rótulo temporal para cada linha de tempo envolvida, o qual corresponde ao período de tempo comum entre as tuplas que compõem a nova tupla. Por sua vez, a operação de produto cartesiano com semântica não-seqüenciada, trata os rótulos temporais como atributos definidos pelo usuário. Sendo assim, para cada relação envolvida no produto cartesiano, um rótulo temporal separado aparece na tabela resultante.

Como exemplo, deseja-se saber quando os dados dos empregados foram alterados. Isto significa que deve-se encontrar as tuplas contendo informações sobre os mesmos empregados tendo intervalos de tempo de validade adjacentes (*meeting*). Em AT-

SQL2, os intervalos de tempo de validade de uma relação R podem ser referenciados usando $VALID(R)$.

```
NONSEQUENCED VALID
    SELECT BEGIN(VALID(a2)), a2.Nome
    FROM~Empregados~a1, Empregados~a2
    WHERE a1.EmpCod = a2.EmpCod
    AND   VALID(a1) meets VALID(a2);
```

BEGIN(VALID(a2))	a2.Nome
1996/10/29	João

No exemplo acima, o produto cartesiano das tabelas a_1 e a_2 é calculado usando semântica não-temporal e o tempo de validade é tratado como um atributo definido pelo usuário, acessível usando as expressões $VALID(a_1)$ e $VALID(a_2)$.

3.2.2 Algoritmo de tradução da ATSQL2 para SQL

O principal problema na utilização do padrão SQL para avaliar consultas temporais é que os intervalos de tempo devem ser calculados durante a avaliação da consulta (STEINER, 1998). Logo, deve-se encontrar uma forma de integrar o cálculo destes em algum lugar nos comandos SQL. O protótipo *TimeDB* (STEINER, 1995) utiliza o seguinte algoritmo de tradução de consultas temporais para o padrão SQL: uma consulta temporal é traduzida em uma expressão de álgebra usando as operações de conjunto *união* ($\cup^t$), *intersecção* ($\cap^t$), e *diferença* ($-^t$). Os argumentos dessas operações de conjunto são simples expressões de álgebra usando uma operação de *seleção temporal* (σ^t), *projeção temporal* (π^t), *produto cartesiano temporal* ($\times^t$) ou o resultado de outra operação de conjunto temporal. Cada expressão de álgebra usando uma combinação das operações de seleção, projeção e produto cartesiano pode ser avaliada separadamente usando um comando padrão do tipo *SELECT-FROM-WHERE*. Os resultados intermediários de cada comando são armazenados em tabelas temporárias e depois utilizados para calcular outras partes da expressão. Um intervalo de tempo de validade é mapeado internamente para dois atributos *VTS* (*valid-time start*) e *VTE* (*valid-time end*). Um intervalo de tempo de transação, do mesmo modo, possui dois atributos *TTS* (*transaction-time start*) e *TTE* (*transaction-time end*). Tabelas bitemporais possuem atributos para o tempo de validade e para o tempo de transação. Isto irá permitir o tratamento de operações unitemporais (somente o tempo de validade ou somente o tempo de transação) e bitemporais (ambos tempo de validade e tempo de transação).

A seguir é apresentado um exemplo (Figura 3.1) mostrando como uma operação unitemporal pode ser implementada usando o padrão SQL, todavia, sem focalizar a eficiência. É possível desenvolver muitas implementações diferentes da mesma operação de álgebra, e até mesmo criar um otimizador de consultas que escolha a melhor implementação para uma determinada consulta.

3.2.3 Álgebra temporal

O protótipo *TimeDB* (STEINER, 1995) implementa as operações de álgebra temporal união ($\cup^t$), diferença ($-^t$), intersecção ($\cap^t$), seleção (σ^t), projeção (π^t) e produto cartesiano ($\times^t$) usando comandos do padrão SQL. Sem focalizar a eficiência, as subseções que se seguem descrevem como operações unitemporais podem ser

Considerando a tabela de tempo de validade chamada Empregado abaixo:

VALID	CodEmp	Nome	Salário	Gerente
[1999/10/21 - ∞)	112	Paulo	910,00	111
[1999/10/21 - 1999/10/29)	111	João	870,00	111
[1999/10/29 - ∞)	111	João	930,00	111
[1999/10/21 - 1999/11/1)	113	Jorge	830,00	111
[1999/11/1 - ∞)	113	Jorge	830,00	112

Para saber o empregado que foi gerente de Jorge antes de Paulo, escrevemos o seguinte comando em ATSQL2:

```
NONSEQUENCED VALID
SELECT a2.Gerente
FROM   (VALID
        SELECT e1.CodEmp
        FROM   Empregado e1, Empregado e2
        WHERE  e1.Nome    = 'Jorge' AND
               e1.Gerente = e2.CodEmp AND
               e2.Nome    = 'Paulo') a1,
        Empregado a2
WHERE VALID(a2) meets VALID(a1);
```

A tradução do comando ATSQL2 acima para SQL é a seguinte:

```
CREATE TABLE aux1(vts, vte, CodEmp) AS
SELECT e1.vts, e1.vte, e1.CodEmp
FROM    Empregado e1, Empregado e2
WHERE   e1.Nome    = 'Jorge' AND
        e1.Gerente = e2.CodEmp AND
        e2.Nome    = 'Paulo';

SELECT~a2.Gerente
FROM   aux1 a1, Empregado a2
WHERE  a1.CodEmp = a2.CodEmp AND
       a2.vte = a1.vts;

DROP TABLE aux1;
```

O resultado desta consulta é o número 111, que corresponde ao empregado de nome João.

Figura 3.1: Exemplo de tradução de ATSQL2 para SQL (Adaptado de Steiner (1998)).

implementadas usando o padrão SQL. O algoritmo para a operação unitemporal de conjunto diferença aqui apresentado é baseado em ChronoLog (BÖHLEN, 1994).

A operação temporal de conjunto união

Considerando duas relações de tempo de validade união-compatíveis R_1 e R_2 com os atributos

$$R_1 =< A_1, A_2, ..., A_n, vts\_\#\$, vte\_\#\$ >$$

$$R_2 =< B_1, B_2, ..., B_n, vts\_\#\$, vte\_\#\$ >$$

A operação união de tempo de validade $\cup^v$ de R_1 e R_2 é igual a mesma operação união não-temporal:

$$R_1 \cup^v R_2 \equiv R_1 \cup R_2$$

A operação temporal de conjunto união não realiza a operação de coalescência das tuplas equivalentes com períodos de tempo sobrepostos. Assim, a relação resultante pode conter tuplas equivalentes com sobreposição de períodos de tempo.

A operação temporal de conjunto diferença

Considerando duas relações de tempo de validade união-compatíveis R_1 e R_2 com os atributos

$$R_1 =< A_1, A_2, ..., A_n, vts\_\#\$, vte\_\#\$ >$$

$$R_2 =< B_1, B_2, ..., B_n, vts\_\#\$, vte\_\#\$ >$$

A operação diferença de tempo de validade de R_1 e R_2,

$$R_1 := R_1 -^v R_2,$$

pode ser traduzida nos seguintes comandos SQL (STEINER, 1998):

```
INSERT INTO R1
 SELECT a0.vts_#$, a1.vts_#$, a0.A1, a0.A2, ..., a0.An
   FROM R1 a0, R2 a1
   WHERE a1.vts_#$ > a0.vts_#$ AND
         a1.vts_#$ < a0.vte_#$ AND
         a0.A1 = a1.B1 AND
         a0.A2 = a1.B2 AND
         ...           AND
         a0.An = a1.Bn;

INSERT INTO R1
 SELECT a1.vte_#$, a0.vte_#$, a0.A1, a0.A2, ..., a0.An
   FROM R1 a0, R2 a1
   WHERE a1.vte_#$ > a0.vts_#$ AND
         a1.vte_#$ < a0.vte_#$ AND
         a0.A1 = a1.B1 AND
         a0.A2 = a1.B2 AND
         ...           AND
         a0.An = a1.Bn;

DELETE FROM R1 a0
WHERE EXISTS (SELECT a1.*
              FROM R2 a1
            WHERE ((a0.vts_#$ >= a1.vts_#$ AND a0.vts_#$ < a1.vte_#$) OR
                   (a1.vts_#$ >= a0.vts_#$ AND a1.vts_#$ < a0.vte_#$))
```

```
AND
a0.A1 = a1.B1 AND
a0.A2 = a1.B2 AND
...           AND
a0.An = a1.Bn;
```

A operação unitemporal de conjunto diferença $R_1 -^v R_2$ retorna as tuplas em R_1 com intervalos de tempo durante os quais não possam ser encontradas tuplas equivalentes em R_2. Presumindo que **t1** é uma tupla da tabela R_1 e que possua um intervalo de tempo de validade como está representado na figura 3.2. A tupla **t2** é equivalente na tabela R_2. Existem dois casos distintos no algoritmo acima. O primeiro caso considera uma possível parte não sobreposta do intervalo de tempo de validade no início da tupla **t1**. O segundo caso considera uma possível parte não sobreposta no final da tupla **t1**. O primeiro comando *INSERT* adiciona tuplas na tabela R_1 para a qual uma tupla equivalente exista em R_2 e cujo intervalo de tempo de validade se sobreponha como representado no caso 1. Essas tuplas recebem rótulos temporais com intervalos de tempo não sobrepostos da tupla **t1** com um intervalo *[t1.vts_#$ - t2.vts_#$)*. O segundo comando *INSERT* faz o mesmo para as tuplas em R_1 para as quais uma tupla equivalente exista em R_2 e cujo intervalo de tempo de validade se sobreponha como representado no caso 2. Essas tuplas recebem rótulos temporais com intervalos de tempo não sobrepostos da tupla **t1**, com um intervalo *[t2.vte_#$ - t1.vte_#$)* . Esta abordagem também resolve o caso em que o intervalo de tempo de validade da tupla **t2** está contido no intervalo de tempo de validade da tupla **t1**. O próximo passo é deletar todas as tuplas da tabela R_1 para as quais existam tuplas equivalentes em R_2 com sobreposição no intervalo de tempo.

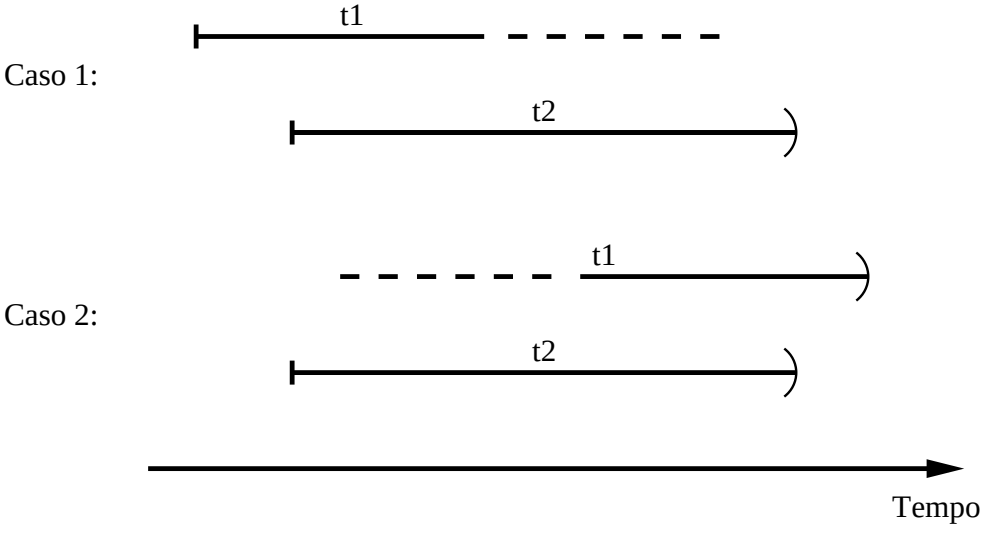

Figura 3.2: Os dois casos de sobreposição dos intervalos de tempo considerados no cálculo da operação temporal de conjunto diferença (Adaptado de Steiner (1998)).

No algoritmo apresentado acima, a tabela R_1 é modificada. No caso de R_1 ser uma tabela definida pelo usuário, a operação temporal de conjunto diferença de duas tabelas deve ser calculada em uma tabela auxiliar. Sendo assim, a tabela definida pelo usuário deve ser copiada antes do cálculo da operação temporal de conjunto diferença.

A operação temporal de conjunto intersecção

A operação temporal de conjunto intersecção pode ser obtida através de duas operações temporais de conjunto diferença consecutivas. Considerando duas relações de tempo de validade união-compatíveis R_1 e R_2 com os atributos

$$R_1 =< A_1, A_2, ..., A_n, vts\_\#\$, vte\_\#\$ >$$

$$R_2 =< B_1, B_2, ..., B_n, vts\_\#\$, vte\_\#\$ >$$

O conjunto intersecção de tempo de validade de R_1 e R_2, $R_1 \cap^v R_2$, pode ser escrito como

$$R_1 -^v (R_1 -^v R_2)$$

Outra maneira de efetuar a operação temporal de conjunto intersecção é através de uma junção temporal (STEINER, 1998):

```
SELECT DISTINCT A1, A2, ..., An,
       GREATEST(R1.vts_#$, R2.vts_#$) vts_#$,
       LEAST(R1.vte_#$, R2.vte_#$) vte_#$
FROM R1, R2
WHERE R1.A1 = R2.B1 AND
      R1.A2 = R2.B2 AND
      ...           AND
      R1.An = R2.Bn AND
      GREATEST(R1.vts_#$, R2.vts_#$) < LEAST(R1.vte_#$, R2.vte_#$)
```

A intersecção dos períodos de tempo de validade é calculada usando as funções *GREATEST* e *LEAST*. Estas estão disponíveis no *Oracle*[1] e retornam respectivamente o maior e o menor valor em uma lista de valores. Calculando o maior limite inferior e o menor limite superior dos intervalos de tempo envolvidos resulta na intersecção dos mesmos se o resultado do limite inferior for menor que o resultado do limite superior. Essa condição é verificada na cláusula *WHERE*.

A operação temporal de seleção

A operação temporal de seleção é igual a uma operação não-temporal de seleção. Entretanto, ela é estendida com predicados adicionais para comparação temporal. Como foi proposto em (SNODGRASS et al., 1993), a operação temporal de seleção deve suportar os predicados *=, precedes, overlaps, meets* e *contains*. Quando traduzidos para o padrão SQL, os predicados devem ser transformados em condições conforme definido abaixo.

- ***X precedes Y*** se e somente se ***end(X) ¡ begin(Y)***.
- ***X = Y*** se e somente se ***begin(X) = begin(Y) and end(X) = end(Y)***.
- ***X meets Y*** se e somente se ***end(X) = begin(Y)***.
- ***X overlaps Y*** se e somente se ***begin(X) ¡ begin(Y) and end(X) ¡ end(Y) and end(X) ¿ begin(Y)***.

[1] As funções *GREATEST* e *LEAST* não fazem parte do padrão SQL, mas o mesmo resultado pode ser obtido usando a expressão CASE disponível no SQL-92 (MELTON; SIMON, 1993).

- ***X contains Y*** se e somente se ***begin(Y) ¿= begin(X) and end(Y) ¡= end(X)***.

Finalmente, as funções *begin* e *end* devem ser suportadas de modo que retornem o limite inferior e o limite superior de um intervalo de tempo respectivamente.

A operação temporal de projeção

A operação temporal de projeção é igual a uma operação não-temporal de projeção. A única diferença é se o rótulo temporal é ou não membro da lista de atributos para projeção.

No caso da compatibilidade ascendente temporal, não há rótulos temporais na projeção resultante, pois o resultado deve compreender somente as tuplas que correspondem ao instante *now*. No caso da consulta seqüenciada, o rótulo temporal é membro por *default*. Uma consulta de tempo de validade retorna implicitamente o rótulo temporal do tempo de validade. No caso de uma consulta não-seqüenciada, os atributos temporais são tratados como atributos definidos pelo usuário e devem ou não aparecer no resultado, dependendo se os mesmos aparecem ou não na lista de projeção especificada pelo usuário. O rótulo temporal do tempo de validade das tuplas em uma relação R pode ser referenciado por *VALID(R)*.

A operação temporal de produto cartesiano

A operação temporal de produto cartesiano combina tuplas de duas relações que possuam períodos de tempo de validade comuns. Considerando duas relações de tempo de validade R_1 e R_2, com os atributos

$$R_1 =< A_1, A_2, ..., A_n, vts\_\#\$, vte\_\#\$ >$$

$$R_2 =< B_1, B_2, ..., B_n, vts\_\#\$, vte\_\#\$ >$$

O produto cartesiano de tempo de validade de R_1 e R_2, $R_1 \times^v R_2$, pode ser traduzido no seguinte comando SQL (STEINER, 1998):

```
SELECT A1, A2, ..., An, B1, B2, ..., Bn,
       GREATEST(R1.vts_#$, R2.vts_#$) vts_#$,
       LEAST(R1.vte_#$, R2.vte_#$) vte_#$
       FROM R1, R2
WHERE GREATEST(R1.vts_#$, R2.vts_#$) < LEAST(R1.vte_#$, R2.vte_#$)
```

Subqueries

Subqueries são consultas que aparecem no corpo de outra consulta. Esse é outro método pelo qual os dados de diferentes relações podem ser relacionados (MELTON; SIMON, 1993). É comum o uso de *subqueries* juntamente com os predicados *EXISTS*, *IN*, *NOT EXISTS* e *NOT IN*. *Subqueries* correlacionadas são aquelas que referenciam valores de atributos de tabelas da consulta principal. No exemplo abaixo, considerando a tabela da figura 3.1, deseja-se obter os empregados que ganham menos que outros.

```
VALID
  SELECT *
  FROM Empregado e1
  WHERE EXISTS (SELECT *
                FROM Empregado e2
                WHERE e1.Salario < e2.Salario);
```

O comando ATSQL2 acima deve retornar a seguinte tabela:

VALID	CodEmp	Nome	Salário	Gerente
[1999/10/21 - 1999/10/29)	111	João	870,00	111
[1999/10/21 - 1999/11/1)	113	Jorge	830,00	111
[1999/11/1 - ∞)	113	Jorge	830,00	112
[1999/10/21 - ∞)	112	Paulo	910,00	111

As consultas com *subqueries* correlacionadas devem ser traduzidas em operações temporais de conjunto. Uma consulta com *subquery EXISTS*, *NOT EXISTS*, *IN* ou *NOT IN* pode ser traduzida em uma operação temporal de conjunto diferença entre a consulta principal e a *subquery*. A consulta do exemplo acima possui uma *subquery EXISTS* com a correlação *e1.Salario ¡ e2.Salario*. A expressão de álgebra da consulta é a seguinte:

$$Empregado -^{v} (Empregado -^{v} \pi Empregado_{e1}.* \\ (\sigma e1.salario < e2.salario(Empregado_{e1} \times^{v} Empregado_{e2})))$$

Primeiro, o produto cartesiano da tabela *Empregado* com ela própria é calculado retornando a combinação de cada tupla da tabela externa com cada uma da tabela interna juntamente com seus períodos comuns de tempo de validade. Segundo, na tabela resultante, as tuplas são selecionadas de acordo com o critério de seleção *e1.salario ¡ e2.salario*. Terceiro, todos os atributos da tabela interna são projetados e a tabela é retornada contendo os salários para o qual existam outros mais altos.

Uma consulta com *subquery NOT EXISTS* ou *NOT IN*, por sua vez, pode ser traduzida de forma muito similar. Por exemplo, para conhecer a história dos mais altos salários, com base na mesma tabela da figura 3.1, pode-se utilizar uma *subquery NOT EXISTS*:

```
VALID
  SELECT *
  FROM Empregado e1
  WHERE NOT EXISTS (SELECT *
                    FROM Empregado e2
                    WHERE e1.Salario < e2.Salario);
```

A consulta acima deve retornar a seguinte tabela:

VALID	CodEmp	Nome	Salário	Gerente
[1999/10/21 - ∞)	112	Paulo	910,00	111
[1999/10/29 - ∞)	111	João	930,00	111

A expressão de álgebra da consulta é a seguinte:

$$Empregado -^{v} \pi Empregado_{e1}. * (Empregado_{e1} \bowtie_{e1.salario<e2.salario} Empreegado_{e2})$$

Primeiro, o produto cartesiano de tempo de validade da tabela *Empregado* com ela própria é calculado. Segundo, as tuplas são selecionadas se a condição *e1.salario ¡ e2.salario* seja verdadeira. Para essas duas operações, a notação $\bowtie^{v}_{e1.salario<e2.salario}$ é usada. Terceiro, a tabela resultante deve conter somente os valores de atributos do primeiro argumento da operação temporal de produto cartesiano. O resultado intermediário contém todas as tuplas para as quais um empregado pode ser encontrado

com um salário alto (com os períodos de tempo de validade correspondentes). Finalmente, uma operação temporal de conjunto diferença a partir da tabela *Empregado* deverá retornar o resultado desejado.

Coalescência

A operação de coalescência[2] é usada para calcular os intervalos maximais do tempo de validade de tuplas com valores de atributos idênticos. Essa operação especial não existe na álgebra relacional não-temporal.

Em ATSQL2, a palavra (*PERIOD*) é usada para denotar que um conjunto de tuplas deve coalesc*er*. Esta palavra pode aparecer após qualquer referência a uma tabela derivada na cláusula *FROM* ou no final de uma consulta temporal ATSQL2.

A proposta para traduzir a operação de coalescência de ATSQL2 para SQL é atualizar o limite superior do intervalo de tempo das tuplas idênticas para que todas as tuplas equivalentes entre si fiquem com o mesmo limite superior. A atualização deve se repetir até que não exista mais tuplas a serem atualizadas. Esse *loop* requer programação adicional para ser alcançado, pois não existe no padrão SQL. O último passo, é deletar as tuplas para as quais existam outras tuplas equivalentes que possuam o período de tempo de validade completo. Em outras palavras, deve-se deletar as tuplas equivalentes que possuam intervalos que sejam menores (contidos) no intervalo de outra tupla igual mas que possua um intervalo de tempo maior.

Considerando uma relação de tempo de validade R_1 com os seguintes atributos

$$R_1 =< A_1, A_2, ..., A_n, vts\_\#\$, vte\_\#\$ > .$$

A operação de coalescência da relação $R_1 := coalesce(R_1)$, é realizado como segue (STEINER, 1998):

```
** REPEAT **
    UPDATE R1 A0
      SET (A0.VTE_#$) = (SELECT MAX(A1.VTE_#$)
                         FROM  R1 A1
                         WHERE A0.C0 = A1.C0
                         AND   A0.C1 = A1.C1
                         AND   A0.VTE_#$ >= A1.VTS_#$
                         AND   A0.VTE_#$ <  A1.VTE_#$)
      WHERE EXISTS (SELECT *
                    FROM  R1 A1
                    WHERE A0.C0 = A1.C0
                    AND   A0.C1 = A1.C1
                    AND   A0.VTE_#$ >= A1.VTS_#$
                    AND   A0.VTE_#$ <  A1.VTE_#$);
    ** UNTIL NO UPDATES **

    DELETE FROM R1 A0
      WHERE EXISTS (SELECT *
                    FROM  R1 A1
                    WHERE A0.C0 = A1.C0
                    AND   A0.C1 = A1.C1
                    AND   A1.VTS_#$ < A0.VTS_#$
                    AND   A1.VTE_#$ = A0.VTE_#$);
```

[2]A palavra coalescência é latina, vem de *co* + *alescere* = crescer juntos, em latim. Basicamente é a transformação de um todo difuso em um conjunto concreto. Diz-se que os gases coalescem, quando assumem forma líquida. Pode ser também unir em um conjunto, unir ou unir-se para um fim ou objetivo comum, unir forças ou combinar elementos.

Por exemplo, analisando a tabela *Empregado* abaixo, verifica-se a existência de duas atualizações no salário de *João:*

VALID	CodEmp	Nome	Salário	Gerente
[1999/10/21 - ∞)	112	Paulo	910,00	111
[1999/10/21 - 1999/10/29)	111	João	870,00	111
[1999/10/29 - ∞)	111	João	930,00	111
[1999/10/21 - 1999/11/1)	113	Jorge	830,00	111
[1999/11/1 - ∞)	113	Jorge	830,00	112

Considerando a tabela acima, deseja-se conhecer o período de tempo maximal dos dados armazenados sobre o empregado *João* na tabela *Empregado* sem considerar o atributo *Salário*. O comando em ATSQL2 é o seguinte:

```
VALID
 (SELECT CodEmp, Nome, Gerente
  FROM Empregado
  WHERE Nome = 'João') (PERIOD);
```

A tabela resultante é a seguinte:

VALID	CodEmp	Nome	Gerente
[1999/10/21 - ∞)	111	João	111

3.3 Conclusões sobre o capítulo

Este capítulo apresentou um resumo das linguagens TSQL2 e ATSQL2. A TSQL2 foi a primeira tentativa de especificar um padrão de uma linguagem de consulta temporal. Segundo Steiner (1995), em 1994, Richard Snodgrass iniciou o trabalho junto aos comitês SQL3 da ANSI e ISO para propor uma nova parte da SQL3 chamada SQL/Temporal, a qual foi formalmente aprovada em julho de 1995. A idéia era usar TSQL2 para o padrão SQL/Temporal. Entretanto, o *design* da TSQL2 recebeu severas críticas:

- a operação de coalescência é implícita não permitindo duplicatas;
- a sintaxe é confusa;
- não há semântica formalmente definida;
- não há implementação.

A idéia de integralidade temporal introduzida em (BÖHLEN; MARTI, 1994; BÖHLEN, 1994) foi usada para avaliar a TSQL2 e levou ao redesenho da linguagem. A nova linguagem foi chamada de ATSQL2 (*Applied* TSQL2).

Em 1994, Andreas Steiner desenvolveu um SGBD bitemporal chamado ChronoSQL, o qual trata consultas de tempo de validade e de tempo de transação, bem como suporta consultas bitemporais. Em 1995, Steiner foi convidado para participar do projeto ATSQL2. Durante essa colaboração, ele migrou seu protótipo ChronoSQL para o protótipo *TimeDB* (STEINER, 1995) que implementa a linguagem ATSQL2. Além da influência do ChronoSQL no tratamento do tempo de validade

e do tempo de transação em comandos temporais, a implementação do protótipo *TimeDB* também ajudou a clarear e a refinar a linguagem ATSQL2 como um todo.

Neste trabalho, optou-se por utilizar a linguagem ATSQL2 pelos seguintes motivos:

- facilidade de tradução de comandos temporais para SQL;
- possui similaridade sintática com o padrão SQL;
- existe um algoritmo de tradução de comandos ATSQL2 para SQL (STEINER, 1998);
- a linguagem ATSQL2 possui implementação (STEINER, 1995).

A facilidade de tradução de comandos ATSQL2 para SQL é alcançada pela introdução dos *timeflags* na frente dos comandos SQL, bastando verificar o primeiro *token* do comando. Se ele for igual a um dos *timeflags* da ATSQL2, trata-se de um comando ATSQL2 e passível de verificação de sintaxe e de tradução. Senão, trata-se de um comando SQL.

A similaridade sintática também é outra vantagem viabilizada pela utilização dos *timeflags*. Além de facilitar a tradução dos comandos temporais para SQL, a similaridade sintática propicia uma fácil compreensão para os programadores e usuários conhecedores da linguagem SQL.

Quanto a utilização do algoritmo de tradução de comandos ATSQL2 para SQL neste trabalho, o mesmo sofrerá pequenas modificações adequadas ao modelo de dados proposto para a extensão temporal de OASIS.

A implementação do protótipo *TimeDB* com base no algoritmo de tradução apresentado prova que é viável a sua utilização com base em SGBDs convencionais.

4 OASIS E SUA EXTENSÃO TEMPORAL

Este capítulo contém uma introdução a linguagem OASIS (*Open and Active Specification of Information Systems*) (LETELIER, 1998; PASTOR; SALVERT, 1995; PASTOR et al., 1997, 1998; TORRES, 1998) e a extensão da mesma com aspectos temporais (ZANATTA, 2000). O capítulo apresenta os conceitos conforme descrito em Zanatta (2000). O texto original sofreu pequenas adaptações para adequação a este trabalho.

4.1 OASIS

OASIS é uma abordagem formal para a especificação de modelos conceituais, seguindo o paradigma orientado a objetos, enriquecida através de regras semânticas.

4.1.1 Notação usada na apresentação de OASIS

A seguir é apresentada a notação utilizada em OASIS. Essa notação é extraída de Letelier (1998) e Pastor e Salvert (1995).

- símbolo terminal ou caracteres entre ' ';
- símbolo não terminal entre <>;
- símbolos opcionais entre [ ];
- símbolos alternativos separados por |.

As simplificações utilizadas para apresentar OASIS são as seguintes:

- sendo x um elemento em particular, para descrever um bloco de elementos x, usa-se $< bloco\_x >$;
- para uma lista de elementos x, usa-se $< lista\_x >$;
- para uma seqüência de elementos x, usa-se $< sec\_x >$.

Estas simplificações são definidas conforme apresentação abaixo:

```
<bloco_x> := <bloco_x> <x> | <x>
<lista_x> := <lista_x> ',' <x> | <x>
<sec_x>   := <sec_x> <x> ';' | <x> ';'
```

São listados abaixo, alguns símbolos não terminais usados freqüentemente:

- não terminal < *formula* >: é uma fórmula bem formada em lógica de predicados de primeira ordem, baseada em valores constantes, atributos e/ou parâmetros de ações;

- não terminal < *expressao* >: é uma expressão booleana ou aritmética baseada em valores constantes, atributos e/ou parâmetros de ações;

- não terminal < *valor* >: é um elemento de algum valor dos domínios permitidos.

4.1.2 Modelo conceitual

Em OASIS, um modelo conceitual corresponde a um conjunto de elementos de especificação, incluindo domínios, classes simples, classes complexas e interfaces (LETELIER, 1998; PASTOR; SALVERT, 1995).

O modelo conceitual de um sistema de informação corresponde em OASIS à especificação de um esquema conceitual, que está definido como segue:

```
<def_esquema_conceitual> ::= conceptual_schema <id_esquema>
                             [ <bloco_def_domínio> ]
                               <bloco_def_classe>
                             [ <bloco_def_classe_complexa> ]
                               <bloco_def_interface>
                               end_conceptual_schema
```

Nas próximas subseções, serão apresentadas em detalhes as especificações de cada um dos blocos mencionados acima.

4.1.3 Tipos de dados (domínios)

Em OASIS há vários tipos de dados: *nat*, *int*, *real*, *bool*, *char*, *string*, *date*, *time* e *blob* ("*binary large objects*"). O especificador tem a possibilidade de declarar outros tipos abstratos de dados e utilizá-los.

Por exemplo, neste trabalho, são definidos tipos para representar os preços que uma fita de vídeo obteve na vídeo-locadora desde que esta foi cadastrada até o dia de hoje (data do sistema). Para isto, é definido um atributo "preço" cujo domínio são os números reais com dimensão temporal por intervalo de tempo.

A seguir é apresentado a sintaxe para especificação de um domínio.

```
<def_domínio>   ::= domain <id_domínio>
                    '(' <lista_id_domínio> ')'
                    [import <lista_id_domínio>] ';'
                    [var <lista_def_variáveis>] ';'
                    functions <sec_def_funções>
                    equations <sec_def_equações>
                    end domain
<def_variáveis>  ::= <id_variáveis> ':' <domínio>
<def_funções>    ::= <id_funções>
   ['('lista_def_parâmetros')']':'<domínio>
<def_equações>   ::= <fórmula_equacional>
```

Os tipos de dados genéricos podem ser especificados através de uma lista de domínios entre parênteses, indicando que tais tipos de dados estão parametrizados por outros tipos base.

A seção de *import* permite utilizar tipos abstratos de dados previamente definidos.

A seção *equations* é uma seqüência de fórmulas bem formadas da lógica equacional, do tipo $ter1 = ter2$, onde $ter1$ e $ter2$ são terminações que descrevem como trabalham as funções.

4.1.4 Especificação da classe

Uma classe é composta por um nome da classe e um *template*. O *template* da classe está dividido em seções e parágrafos. A sintaxe para definir uma classe é a seguinte:

```
<def_classe>    ::=     class <id_classe>
                        <mecanismos_identificação>
                        <atributos_constantes>
                        [ <atributos_variáveis> ]
                        [ <atributos_derivados> ]
                        [ <derivações> ]
                        [ <restrições_integridade> ]
                        <eventos>
                        [ <operações> ]
                        [ <gatilhos> ]
                        [ <avaliações> ]
                        [ <pré-condições>]
                        [ <protocolos> ]
                        end class
```

Além das seções opcionais mostradas acima, se uma classe for definida por herança a partir de outra classe, então todas as seções são opcionais.

Se $< id\_classe >$ coincide com o identificador do esquema conceitual assume-se que se trata da descrição de propriedades globais associadas ao da classe implicitamente estabelecida por agregação de todas as classes definidas no esquema conceitual.

As próximas subseções descrevem detalhadamente os templates da classe. As operações e os protocolos não são descritos neste trabalho pelo fato de não serem utilizados.

Mecanismos de identificação

É uma função que estabelece correspondência entre valores de atributos de um objeto e seu oid (objeto identificador). Uma mesma classe pode ter distintos mecanismos de identificação.

Os mecanismos de identificação apresentam a seguinte sintaxe:

```
<mecanismos_identificação> ::= identification
                               <sec_def_identificador>
<def_identificador>        ::= <id_identificador>
                               ':' '('<lista_id_atributo>')'
```

Um mecanismo de identificação é composto por uma lista de atributos constantes. A lista de valores associada a estes atributos constantes, deve determinar unicamente um objeto da classe.

Para exemplificar, supondo que em uma vídeo-locadora exista a classe usuário. Nesta classe, o número do usuário é baseado no atributo constante "n_usuário", conforme apresentado abaixo.

```
class usuario
identification
```

```
        número_usuario: (n_usuario);
constant attributes
        n_usuario: string;
```

Atributos

O estado dos objetos é observado através dos valores de seus atributos. Os atributos são propriedades estruturais das instâncias de uma classe e possuem um tipo associado.

Em OASIS distinguem-se três tipos de atributos:

- atributos constantes: aqueles que recebem seu valor quando se cria o objeto e não mudam com o passar do tempo;
- atributos variáveis: aqueles cujos valores podem alterar quando ocorre uma ação relevante;
- atributos derivados: que são derivados a partir de outros atributos.

Para cada atributo definido existem funções e operações associadas, disponíveis para serem usadas em especificações do *template*, e que são úteis para manipular atributos cuja cardinalidade é maior que um.

Atributos constantes e variáveis utilizam uma mesma sintaxe definida abaixo.

```
<atributos_constantes>  ::= constant attributes
                            <sec_def_atributo_cons_ou_var>
<atributos_variáveis>   ::= variable attributes
                            <sec_def_atrib_cons_ou_var>
<def_atrib_cons_ou_var> ::= <id_atributo> ':' <domínio>
                            '('<lista_valor_por_default>')']
                            [towards <cardinalidade>
                               [<def_representação>]]
<valor_por_default>     ::= <valor>
<cardinalidade>         ::= '('<card_min>','<card_max>')'
<def_representação>     ::= list '['<def_índice>']'| set|bag
<def_índice>            ::= <intervalo> | <lista_corrente>
```

Os atributos podem ter um valor por *default* quando se cria o objeto. O especificador pode introduzir entre parênteses este valor por *default* na declaração do atributo.

A cláusula *towards* $< cardinalidade >$ apresenta as cardinalidades vistas desde que o objeto tenha o domínio primitivo do atributo. Se não especificar assume-se *towards* (0,1).

Deve-se garantir que $< card\_min > <= < card\_max >$.

Os valores dos atributos constantes e variáveis são parâmetros implícitos do evento *new* do objeto. Se para algum atributo tem-se $< card\_min > >= 1$, então é obrigatório que a partir da criação do objeto deste atributo a condição seja satisfeita, isto é, tenha pelo menos essa quantidade de valores. Se o parâmetro associado a um atributo constante ou variável não está instanciado no evento *new* e existe um valor por *default* declarado, então o atributo receberá este valor.

Quando o atributo é multivalorado ($< card\_max > > 1$), *list*, *set* e *bag* permitem indicar qual é a representação escolhida. Em *list* e *bag* permitem elementos duplicados, set não. Em *list* a declaração ¡def_índice¿ permite definir a forma de

referência usada para acessar os elementos da lista. Se não se especifica *list*, *set* e *bag*, assume-se que é do tipo *set*.

Por exemplo, é definida abaixo, a declaração do atributo "telefone" com domínio *string*, multivalorado (*towards* (0,*)) e com representação *list*, na classe usuário da vídeo-locadora.

```
Variable attributes
telefone:string towards (0,*) list [1..*];
```

O asterisco (*) indica que não existe restrição a respeito da cardinalidade máxima (em towards), e no máximo valor do número (em list).

A sintaxe para especificação dos atributos derivados é a seguinte:

```
<atributos_derivados>   ::= derived attributes
                            <sec_def_atrib_deriv>
<def_atrib_deriv>       ::= <id_atributo> ':' <domínio>
```

Para atributos derivados, o domínio declarado deve ser consistente com sua especificação na seção *derivations*, na qual determina-se o tipo de atributo (constante ou variável), sua cardinalidade e representação.

Consideremos que exista um atributo derivado na classe multa, que irá guardar o cálculo do valor da multa conforme especificação a seguir:

```
derived attributes
valor_multa: nat;
```

Derivações

São usadas para especificar um atributo derivado. A sintaxe para especificação das derivações é apresentada a seguir:

```
<derivações>           ::= derivations <sec_def_atrib_deriv>
<sec_def_atrib_deriv>  ::= ['{'<fórmula>'}']< atribuições>
<atribuições>          ::= <id_atributo> ':=' <expressão>
```

Para exemplificar, considera-se a definição abaixo, uma derivação na classe multa para definir o valor do atributo derivado "valor_multa", que depende do número de dias que o usuário ficou em atraso na vídeo-locadora.

```
{dias_atraso = 2} valor_multa = fita.preço + 2 * valor;
```

O atributo "valor" armazena o valor da multa diária. Esse exemplo expressa que se a quantidade de dias em atraso na devolução da fita for igual a dois ({dias_atraso = 2}), o valor da multa ("valor_multa") será o preço da fita ("fita.preço") multiplicado pelo valor da multa diária ("valor"). Esse mesmo exemplo pode ser definido da seguinte forma:

```
valor_multa:={day(currenttime)-day(data_mul) * valor};
```

Dessa forma, a quantidade de dias em atraso é calculada pelo próprio sistema.

Restrições de integridade

São fórmulas baseadas no estado do objeto e que devem ser satisfeitas para cada um de seus estados. Podem ser classificadas em estáticas (referem-se a somente um estado) e dinâmicas (relacionam diferentes estados). Operadores temporais tradicionais são utilizados para especificar restrições de integridade dinâmicas. A sintaxe para especificação de restrições de integridade é a seguinte:

```
<restrições_integridade>::= constraints <sec_def_restrições>
<def_restrições>        ::=
                        [always ] '{' <fórmula> '}'
                        | sometimes '{' <fórmula> '}'
                        | next '{' <fórmula> '}'
                        | <fórmula_depois> since <fórmula_antes>
                        | <fórmula_antes> until <fórmula_depois>
                        | sometimes [ '('<timeout> ')' ] <fórmula_depois>
                                        since <fórmula_antes>
                        | always [ '('<delay> ')' ] <fórmula_depois>
                                        since <fórmula_antes>
<fórmula_antes>         ::= '{' <fórmula> '}'
<fórmula_depois>        ::= '{' <fórmula> '}'
<timeout>               ::= <op_rel_timeout> <natural> <unidade_tempo>
<delay>                 ::= <op_rel_delay> <natural> <unidade_tempo>
<op_rel_timeout>        ::= '<' | '<='
<op_rel_delay>          ::= '>' | '>='
<unidade_tempo>         ::= seconds | minutes | hours | days | weeks | months
                          | years
```

Se na verificação das restrições de integridade algumas delas não forem satisfeitas, o estado do objeto deve ser o último estado no qual se cumpriu a restrição de integridade. Os estados posteriores a última situação de integridade são ignorados.

Uma restrição de integridade que utiliza o operador *sometimes* sem especificar o limite de tempo, tem um sentido equivalente a de uma pré-condição para o evento *destroy* do objeto.

Considerando o exemplo da vídeo-locadora a seguir é apresentada uma restrição de integridade que pode ser definida na classe fita para restringir o número de fitas, o qual deve ser diferente de vazio para garantir a existência de fitas.

```
Constraints
       {n_fita <> '' }
```

Pela extensão temporal proposta em (ZANATTA, 2000), não é necessário o uso dos operadores temporais nas restrições de integridade. A sintaxe fica definida como foi proposta na primeira versão de OASIS (PASTOR, 1992b).

Eventos

Um evento ocorre em um instante de tempo. Sua ocorrência resulta em algumas ações, que podem causar mudanças de estado em alguns objetos. Abaixo é apresentada a sintaxe para especificação de eventos:

```
<eventos>   ::= events
                <sec_def_evento>
<def_evento>::=
          <id_evento> ['('lista_def_parâmetro')']
          [new | destroy ]
```

```
          [alias for <operações_implícitas> | <coordenação>]
<coordenação>::= calling to members
                 <lista_serviço_componente>
                 | sharing with members
                 <lista_serviço_componente>
```

As cláusulas *new* e *destroy* indicam respectivamente a criação e destruição de instâncias. Ambos eventos são únicos para cada classe. O evento *new* tem implicitamente como parâmetros, os valores para os atributos constantes e variáveis do objeto.

O evento *new* cria o objeto e deve ser o primeiro evento da vida do objeto. O evento *destroy* é a causa da destruição de um objeto. Quando uma classe não tem o evento *destroy* declarado assume-se que esta classe é "imortal".

Há uma distinção entre eventos privados (quando não tem a cláusula *calling to members* ou *sharing with members*), eventos implicados (com a cláusula *calling to members*) e eventos compartilhados (com a cláusula *sharing with members*). Os eventos privados são serviços oferecidos ou requeridos por um objeto e cuja ocorrência depende somente do comportamento do objeto. Nos eventos implicados a ocorrência do evento no objeto agregado implica necessariamente na ocorrência do correspondente serviço em seus objetos componentes. Quando se trata de eventos compartilhados, esta implicação se dá em ambos os sentidos.

Os eventos implicados e compartilhados somente se definem em uma classe agregada. Em ambos os casos, deve-se estabelecer uma correspondência entre o evento do agregado e o serviço do componente, mediante a definição da lista $<$ *lista_servico_componente* $>$.

Consideremos a definição abaixo da classe usuário da vídeo-locadora onde existem, entre outros, o evento para cadastrar um usuário ("insere_usu") e para removelo ("remove_usu") da vídeo-locadora.

```
class usuario
...
events
       insere_usu new;
       remove_usu destroy;
```

4.2 Extensão de OASIS

Esta seção apresenta a extensão temporal dos aspectos estáticos e dinâmicos de OASIS.

Na extensão temporal (ZANATTA, 2000) feita em OASIS é utilizada ordem no tempo linear. A duração particular de cada *chronon* foi deixada livre para ser escolhida no momento da definição do modelo de dados, o qual será apresentado no capítulo 5.

Os atributos variáveis temporais recebem rótulos na forma de elementos temporais, utilizando o tempo de validade, para melhor modelar a realidade.

Para representar os eventos temporais é adotado somente o tempo de transação, uma vez que o que realmente interessa para a aplicação é o instante no tempo em que ocorreu o evento. O elemento temporal utilizado como rótulo para os eventos é o instante no tempo.

Na primeira subseção é apresentada uma simplificação para a definição de atributos em OASIS. Na segunda subseção são apresentadas as alternativas de extensão

de OASIS. Tendo em vista a alternativa escolhida para estender OASIS são descritos os domínios temporais na terceira subseção, e a definição das classes temporais na quarta subseção. Na quinta subseção, as fórmulas de OASIS são estendidas para suportar o aspecto tempo. Na sexta e última subseção, é apresentado o OASIS Temporal, isto é, OASIS com aspectos temporais.

4.2.1 Simplificação da especificação para atributos em OASIS

Em OASIS não é possível definir atributos cujo valor é um objeto. Quando for necessário, cria-se uma classe e define-se como uma agregação que contém a classe que seria atributo.

No exemplo da vídeo-locadora, supondo que exista uma classe endereço com seus respectivos atributos conforme especificação a seguir.

```
class endereço

identification
        ref: (cod);

constant attributes
        cod: string;

variable attributes
        rua: string;
        numero: nat;
        cidade: string;
        estado: string;
        cep: string;
        complemento: string;
events
        insere_endereço new;
        remove_endereço destroy;
end class
```

Deseja-se especificar que um usuário pode ter um ou mais endereços (residencial, comercial e outros). Isto significa que a classe endereço é agregada da classe usuário. A especificação desta agregação em OASIS é a seguinte:

```
usuario aggregation of
        dynamic inclusive endereco
                towards (1,*) from (1,1) list [1..*]
```

Quando os objetos do componente podem modificar-se, a agregação é definida como *dynamic*.

Para tornar especificações de OASIS mais sucintas e fáceis de entender, introduz-se aqui uma simplificação da notação. Nesta simplificação, admitem-se atributos cujo valor é um objeto (não somente literais como em OASIS original).

Usando a notação simplificada, na classe usuário passa a ser definido um atributo "ende" cujo valor é um objeto (classe endereço). Esta definição é apresentada abaixo:

```
class usuario

variable attributes
        ende: endereco;
        ...
```

Essa simplificação é utilizada na extensão temporal.

4.2.2 Alternativas de extensão

Para estender OASIS com aspectos temporais existem duas alternativas.

Uma delas é modificar o formalismo original OASIS. Isto significa estender a sintaxe e alterar a semântica. OASIS em sua primeira versão (PASTOR, 1992b) foi formalizado através de Teorias de Primeira Ordem. Na segunda versão (PASTOR; SALVERT, 1995), OASIS foi baseado em uma variante da Lógica Dinâmica que permite representar os operadores de obrigação, proibição e permissão usados na Lógica Deôntica (DEONTIC LOGIC, 1984; MEYER, 1988). Em sua última versão (LETELIER, 1998), o formalismo foi mantido.

A alternativa em questão seria modificar seu formalismo básico para uma lógica temporal (MAIOCCHI; PERNICI, 1991b; KOWALSKI; SERGOT, 1985). Em uma lógica temporal é possível fazer referência aos estados dados passados, presentes e futuros da aplicação.

Modelos orientados a objetos encapsulam estado e comportamento através da definição de classes. Assim a segunda alternativa é definir a extensão temporal de OASIS, utilizando o próprio modelo para a extensão. Isto significa definir classes temporais e domínios temporais utilizando a semântica de OASIS.

Neste trabalho, preferiu-se a segunda alternativa, para não modificar a semântica da linguagem.

Nas subseções que seguem, a extensão de OASIS para aspectos temporais é definida na forma de:

- domínios temporais;
- classes temporais e;
- extensão das fórmulas e expressões.

4.2.3 Domínios temporais

Em OASIS, domínios são conjuntos de valores que os atributos podem assumir. Para tratamento do tempo, são definidos os seguintes domínios temporais:

- *time_point*;
- *closed_time_interval*;
- *begin_open_time_interval*;
- *end_open_time_interval*;
- *time_interval*.

OASIS admite uma seção *¡equations¿*, para definição das equações que definem formalmente as funções. Preferiu-se neste trabalho definir as funções apenas de maneira informal.

A seguir são apresentadas as funções que operam sobre domínios temporais.

Domínio "time_point"

O domínio *"time_point"*, foi definido para representar um ponto no tempo. Os pontos no tempo são dados em segundos (*default*).

Foram definidas funções que operam com pontos no tempo, para transformá-los em minutos, horas, dias. Outras funções podem ser definidas conforme a necessidade do usuário.

```
domain time_point
functions
        in_sec (the_point:time_point) : int;
        in_year (the_point:time_point) : int;
        in_minute (the_point:time_point) : int;
        in_hour (the_point:time_point) : int;
        in_month (the_point:time_point) : int;
        in_day (the_point:time_point) : int;
        in_time_stamp (the_point:time_point) : string;
        sec_in_time_point (seconds:int): time_point;
        year_in_time_point (year:int): time_point;
        hour_in_time_point (hour:int): time_point;
        minute_in_time_point (minute:int): time_point;
        month_in_time_point (month:int): time_point;
        day_in_time_point (day:int): time_point;
        time_stamp_in_time_point (the_point:string): time_point;
        addition (the_point1, the_point2:time_point): time_point;
        difference (the_point1, the_point2:time_point): time_point;
end domain
```

A função *"in_sec (the_point: time_point) : int"*, recebe um ponto no tempo com domínio *"time_point (the_point)"* e retorna o ponto no tempo em segundos com domínio inteiro. As funções *"in_year", "in_hour", "in_month", "in_day", "in_minute"* são semelhantes a função *"in_sec"*, retornando o ponto no tempo respectivamente em anos, horas, meses, dias e minutos, todos com domínio inteiro.

A função *"in_time_stamp (the_point: time_point) : string"*, recebe um ponto no tempo com domínio *"time_point (the_point)"* e retorna o ponto no tempo com domínio *string* no formato dia/mês/ano-hora:minuto:segundo.

As funções *"sec_in_time_point", "minute_in_time_point", "hour_in_time_point", "day_in_time_point", "month_in_time_point", "year_in_time_point"* recebem o ponto no tempo em segundos, minutos, horas, dias, meses e anos respectivamente, transformando-o em um ponto no tempo com domínio *"time_point"*.

A função *"time_stamp_in_time_point (the_point: string) : time_point"*, recebe um ponto no tempo (*"the_point"*) com domínio *string* no formato dia/mês/ano-hora:minuto:segundo e, retorna o ponto no tempo com domínio *time_point*.

As funções *"addition"* e *"difference"* recebem dois pontos do tempo com domínio *"time_point"* e retornam um novo ponto no tempo que é, respectivamente, a soma e a subtração dos dois pontos no tempo com domínio *"time_point"*. Na função *"difference"* o primeiro ponto deverá ter subtraído o segundo.

Constantes definidas para o domínio "time_point"

A constante "n", definida abaixo, com domínio inteiro, representa a quantidade de anos, meses, dias, horas, minutos ou segundos.

```
<n> [year | month | day | hour | minute |seconds]
```

A seguir, é apresentada uma constante para representar um ponto no tempo:

- "ano/mês/dia-hora:minuto:segundo";
- "ano": é um número inteiro de 4 algarismos que representa o ano do ponto no tempo;
- "mês": é um número inteiro pertencente ao intervalo de 01 (janeiro) a 12 (dezembro);
- "dia": é um número inteiro que pode variar de 1 até 31 dependendo do mês ao qual irá referenciar;
- "hora": é um número pertencente ao intervalo de 0 (zero) a 23 (vinte e três), representando a hora do ponto no tempo;
- "minuto": é um número pertencente ao intervalo de 0 (zero) a 59 (cinqüenta e nove) representado a quantidade de minutos;
- "segundo": é um número pertencente ao intervalo de 0 (zero) a 59 (cinqüenta e nove) representado a quantidade de segundos.

Notações infixadas para o domínio "time_point"

- 1999/12/04: significa que o ano é 1999, o mês 12, o dia 04 e implicitamente a hora, o minuto e o segundo igual a zero;
- 3 months: representa 3 (três) meses;
- 5 years: representa 5 (cinco) anos;
- *(the_point1 + the_point2)*: equivale a escrever *"addition(the_point1, the_point2)"*;
- *(the_point1 - the_point2)*: equivale a escrever *"difference(the_point1, the_point2)"*;
- "+" : equivalente a função *"addition"*, soma dois pontos no tempo e;
- "-" : equivalente a função *"difference"*, subtrai dois pontos no tempo.

Na função *"difference"* (ou na notação infixada) o primeiro parâmetro (*"the_point1"*) irá subtrair o segundo (*"the_point2"*).

Para exemplificar, consideremos a notação infixada abaixo:

```
2000/02/06 - 5 months;
```

Essa notação é equivalente a ativação da função "difference(2000/02/06, 5 months)". A partir do ano de 2000, do mês 02 e do dia 06, pretende-se tirar 5 meses.

Domínio "time_interval"

O domínio *"time_interval"* é definido como a união dos domínios *"closed_time_interval"*, *"begin_open_time_interval"* e *"end_open_time_inter-val"* definidos nesta subseção.

```
domaintime_interval
functions
pertence(the_point:time_point,interval:time_interval):boolean;
is_begin_open(interval:time_interval):boolean;
is_end_open(interval:time_interval):boolean;
endpoint(interval:time_interval):time_point;
beginpoint(interval:time_interval):time_point;
enddomain
```

A função *"pertence"*, verifica a partir de um intervalo de tempo (*"interval"*) e um ponto no tempo (*"the_point"*), se este ponto pertence ao intervalo. O resultado da função é booleano.

A função *"is_begin_open"* verifica a partir de um intervalo de tempo (*"interval"*) se este intervalo é aberto no inicio. O resultado da função é booleano.

A função *"is_end_open"* verifica a partir de um intervalo de tempo (*"interval"*) se o intervalo é aberto no final. O resultado da função é booleano.

As funções *"endpoint"* e *"beginpoint"* recebem um intervalo de tempo e retornam, respectivamente, o último ponto e o primeiro ponto.

A seguir é apresentado um exemplo:

```
pertence (1992, [1990, 1998]);
```

A função *"pertence"* verifica se o ponto 1992 pertence ao intervalo de tempo [1990, 1998]. O resultado é verdadeiro (*true*).

Domínio "closed_time_interval"

O domínio *"close_time_interval"* é definido para representar um intervalo de tempo fechado, isto é, quando sabe-se o início e o final do intervalo.

```
domain closed_time_interval
functions
 add_end (interval:time_interval, end_point:time_point): time_interval;
 add_begin (begin_point:time_point, interval:time_interval): time_interval;
 subtract_end (interval: time_interval, end_point:time_point): time_interval;
 subtract_begin(begin_point:time_point,interval:time_interval):time_interval;
 open_end (interval:time_interval, end_point:time_point): time_interval;
 open_begin (begin_point:time_point, interval:time_interval): time_interval;
end domain
```

As funções que podem ser aplicadas a esse domínio são as seguintes:

- *"add_end"* e *"subtract_end"*: aumentar e diminuir o final do intervalo;
- *"add_begin"* e *"subtract_begin"*: aumentar e diminuir o início do intervalo e;
- *"open_begin"* e *"open_end"*: alterar o intervalo, deixando-o aberto no início ou aberto no final;

Para resolver estas funções é necessário que elas recebam como parâmetros o intervalo original que tem domínio intervalo de tempo (*"interval"*) e o ponto que será alterado no início (*"begin_point"*) ou no final do intervalo (*"end_point"*) com domínio *time_point*.

Para exemplificar consideremos, a ativação abaixo, da função *"subtract_end"*:

```
subtract_end ([1999/05/05-15:30:35, 2000/06/30-12:00:30], 4 months);
```

A função acima subtrai 4 (quatro) meses do final do intervalo. De 2000/06/30-12:00:30 para 2000/02/30-12:00:30.

Domínio "begin_open_time_interval"

Para representar o intervalo de tempo aberto no início foi definido o domínio *begin_open_time_interval*.

```
domain begin_open_time_interval
functions
 close_begin (interval:time_interval, begin_point:time_point): time_interval;
 add_end (interval:time_interval, end_point:time_point): time_interval;
end domain
```

Este domínio apresenta duas funções: a função *"close_begin"* que recebe um intervalo de tempo (*"interval"*) e um ponto no tempo (*"begin_point"*) fechando o intervalo no início com o ponto recebido; a função *"add_end"* que recebe um intervalo de tempo (*"interval"*) e um ponto no tempo (*"end_point"*) estendendo o intervalo no final com o ponto recebido.

A seguir é apresentada a ativação da função *"add_end"*:

```
add_end([1990/01/01-18:25:12, 2000/02/20-15:10:45], 2000/06/08-12);
```

A função acima estende o intervalo de tempo no final (2000/05/20-15:10:45) pelo ponto no tempo determinado (2000/06/08-12). O resultado é: [1999/01/01-18:25:12,2000/06/08-12].

Domínio "end_open_time_interval"

O domínio *end_open_time_interval* representa o intervalo de tempo aberto no final.

```
domain end_open_time_interval
functions
 close_end (interval:time_interval, end_point:time_point): time_interval;
 add_begin (interval:time_interval, begin_point:time_point): time_interval;
end domain
```

Este domínio apresenta duas funções: a função *"close_end"* que recebe um intervalo de tempo (*"interval"*), um ponto no tempo (*"end_point"*) e fecha o intervalo no final com o ponto recebido; a função *"add_begin"* que recebe um intervalo de tempo (*"interval"*), um ponto no tempo (*"begin_point"*) e estende o intervalo no início com o ponto recebido.

Abaixo é apresentado um exemplo da função *"add_begin"*:

```
add_begin([1990/01/01-18:25:12, 2000/05/20-15:10:45], 1995);
```

A função *"add_begin"* estende o inicio do intervalo (1990/01/01-18:25:12) com ponto no tempo 1995. O resultado é: [1995, 2000/05/20-15:10:45]

Constantes definidas para o domínio "time_interval"

- ">>": significa que o intervalo está aberto no final;
- "<<": significa que o intervalo está aberto no início;
- [*begin_point*, *end_point*]: representa um intervalo de tempo fechado, onde o primeiro argumento é o valor inicial do intervalo (*"begin_point"*) e o segundo o valor final (*"end_point"*);
- [*begin_point*, >>]: representa um intervalo de tempo aberto no final (>>), sendo o início do intervalo determinado pelo ponto no tempo *"begin_point"*;
- [<<, *end_point*]: representa um intervalo de tempo aberto no início (<<), sendo o final do intervalo determinado pelo ponto no tempo *"end_point"*;

A seguir é apresentado um exemplo da ativação função *"close_begin"* utilizando a constante:

```
close_begin ([<<, 2000] , 1990);
```

Esta função acima fecha o início do intervalo (<<) com o ponto no tempo 1999. O resultado é: [1990,2000].

Notações infixadas para o domínio "time_interval"

(*the_point* ? *interval*): onde "?" é símbolo da lógica, representando a palavra *"pertence"*;

Essa notação verifica se um ponto (*"the_point"*) pertence a um intervalo (*"interval"*). É equivalente a escrever: *"pertence (the_point, interval)"*.

Um exemplo utilizando a notação infixada:

```
1992 ? [1990, 1998 ];
```

A notação acima verifica se o ponto 1992 pertence ao intervalo de tempo [1990,1998]. O resultado é verdadeiro (*true*). Equivalente a ativação da função *"pertence (1992, [1990, 1998])"*;

4.2.4 Classes temporais

Além de incluir domínios temporais, a extensão temporal de OASIS permite a definição de classes temporais.

Em OASIS uma classe é composta por um conjunto de instâncias, um mecanismo de identificação para elas e um template comum a todas as instâncias.

O comportamento de um objeto pode ser alterado devido à ocorrência de um evento (fato ocorrido em um determinado instante de tempo). Neste caso é importante que o modelo permita apresentar a história desses eventos. Para os eventos que são declarados como temporais é guardado o registro das suas ocorrências, associando a cada ocorrência o seu tempo de transação.

Consideremos que um sistema possua um relógio e que a aplicação tenha que guardar o valor do relógio sempre atualizado. Para isto, implicitamente em todas as classes, é definido um atributo denominado *currenttime* com domínio do tipo date que recebe como valor inicial o valor do relógio. Para que esse atributo seja atualizado é definido um evento denominado *tick* que se encarregará de atualiza-lo.

Nas seções que seguem estão definidas as seguintes classes temporais:

- *it_nat*;
- *it_nat_value*;
- *it_int*;
- *it_int_value*;
- *temporal_event*;
- *event_occurrence_list*;
- *temporal_event_occurrence*.

OASIS foi concebido para especificar uma única aplicação, isto significa que não apresenta o conceito de genericidade (MEYER, 1997). Este conceito permite utilizar tipos parametrizados. Por exemplo, se OASIS fosse genérico, seria permitido definir uma classe "it¡tipo¿" e o "tipo" poderia ser inteiro, real, caracter entre outros domínios.

Assim, as classes apresentadas devem ser consideradas apenas como esqueletos para as classes que compõem uma aplicação. Por exemplo, a classe *"it_nat"* serve de esqueleto para as classes *"it_float"*, *"it_string"* e assim sucessivamente. O mesmo se aplica as demais classes.

A seguir são apresentadas as classes temporais.

Classe "it_nat"

A classe *"it_nat"* é definida para representar os intervalos temporais para os números naturais.

```
class it_nat
variable attributes
      the_intervals: it_nat_value list [1,*]
      the_point: time_point;

derived attributes
      first_point: time_point;
      last_point: time_point;
the_values (the_intervals): nat list [1,*];
      min_value: nat;
      max_value: nat;
      actual_value: nat;
      value_at_time (the_point): nat;
      values_at_interval (interval): nat list [1,*];
      pair_values_interval_at_interval (interval): time_interval list [1,*];
      interval_at_values (value): nat list [1,*];

events
      insert_interval (interval, value);
      remove_interval (interval, value);
      remove_all_intervals;
      from_now_new_value (interval, value);

valuations
      [insert_interval] := insert (the_intervals[interval],value);
      [remove_interval] := remove (the_intervals[interval], value);
```

```
      [remove_all_intervals] := remove_all (the_intervals);
      [from_now_new_value (valor)] the_intervals[interval], value:= valor;

preconditions
      insert_interval (interval, value)
           if (not exists a_interval in the_intervals
            where (not is_end_open (a_interval) impl
                   (endpoint (a_interval) > beginpoint (interval))
                    and (beginpoint(a_interval) < endpoint (interval)))
            and  (not exists a_interval in the_intervals
                where not is_begin_open (a_interval) impl
                      (endpoint (a_interval) > beginpoint (interval))
                      and (beginpoint(a_interval) < endpoint (interval)))

end class
```

Em OASIS a especificação de atributos derivados dá-se através de uma cláusula *derivations* e a especificação da semântica de um evento através de cláusulas *valuations* e *preconditions*.

Para simplificar o presente texto optou-se por apresentar a semântica dos atributos derivados de maneira informal.

A classe *"it_nat"* apresenta um atributo variado do tipo objeto (*"the_inter-vals"*), que é uma lista onde ficam registrados todos os intervalos de tempo.

Os atributos derivados são os seguintes:

- *"first_point"*, retorna o primeiro intervalo no tempo (*"interval"*) pertencente à lista de intervalos (*"the_intervals"*). A função *"beginpoint"*, definida no domínio *"time_interval"*, retorna o primeiro ponto do intervalo em questão;
- *"last_point"* é semelhante ao *"first_point"*: retorna o último intervalo (*"interval"*) pertencente à lista de intervalos (*"the_intervals"*);
- *"the_values"* retorna a lista de valores que existem em todos intervalos (*"the_intervals"*);
- *"min_value"* e *"max_value"*, retornam respectivamente, o menor e o maior valor do ponto no tempo;
- *"actual_value"* retorna o valor contido no ponto no tempo atual (*currenttime*);
- *"value_at_time"* recebe um ponto no tempo e retorna o valor correspondente ao ponto;
- *"values_at_interval"* recebe um intervalo de tempo e retorna os valores contidos no intervalo;
- *"pair_values_interval_at_interval"* recebe um intervalo de tempo e retorna para cada valor o intervalo de tempo correspondente;
- *"interval_at_values"* recebe um valor e retorna o intervalo que contém esse valor.

Os eventos são definidos abaixo:

- *"remove_interval"* recebe um intervalo e um valor (*"interval"*, *"value"*) removendo-os da lista de intervalos (*"the_intervals"*);
- *"remove_all_intervals"* remove toda a lista de intervalos (*"the_intervals"*);
- *"from_now_new_value"* recebe um intervalo e um valor (*"interval"*, *"value"*) definindo que a partir do instante *now* o intervalo passa a ter um novo valor (*"value"*:=valor);
- *"insert_interval"* recebe um intervalo e um valor (*"interval"*, *"value"*) inserindo o intervalo com seu respectivo valor se a pré-condição permitir.

A pré-condição não permite que o intervalo sobreponha outro intervalo já existente na lista.

Classe "it_nat_value"

A classe *"it_nat_value"* apresenta um atributo para especificar o intervalo de tempo (*"interval"*) e o seu respectivo valor (*"value"*).

```
class it_nat_value
variable attributes
        value: nat;
        interval: time_interval;
end class
```

A classe *"it_nat_value"* armazena um intervalo (*"interval"*) e um valor (*"value"*). Esse par intervalo-valor tem que ser anexado a lista de intervalos (*"the_intervals"*) que está definida na classe *"it_nat"*. Portanto, a classe *"it_nat_value"* é agregada da classe *"it_nat"*, conforme a definição abaixo.

```
it_nat aggregation of
        dynamic inclusive it_nat_value
                towards (1,*) from (1,1) list [1,*]
```

Classe "it_int"

As classes *"it_int"* e *"it_int_value"* são apresentadas a seguir somente para mostrar sua diferença com as classes anteriormente definidas *("it_nat"* e *"it_nat_va-lue")*. Esta diferença é o domínio, isto é, as primeiras classes definidas apresentam intervalo de tempo para o domínio dos naturais e estas para o domínio dos inteiros. Se houver necessidade de utilizar intervalo de tempo para o tipo caracter, as classe *"it_char"* e *"it_char_value"* devem ser definidas e assim sucessivamente.

```
class it_int
variable attributes
       the_intervals: it_int_value list [1,*]

derived attributes
       first_point (interval) : time_point;
       last_point (interval) : time_point;
       the_values (the_intervals): int list [1,*];
       min_value (the_values): int;
       max_value (the_values) : int;
       actual_value: int;
       value_at_time (the_point): int;
```

```
        values_at_interval (interval): int list [1,*];
        pair_values_interval_at_interval (interval): time_interval list [1,*];
        interval_at_values (value): int list [1,*];

events
        insert_interval (interval, value);
        remove_interval (interval, value);
        remove_all_intervals;
        from_now_new_value (interval, value);

valuations
        [insert_interval] := insert (the_intervals[interval],value);
        [remove_interval] := remove (the_intervals[interval], value);
        [remove_all_intervals] := remove_all (the_intervals);
        [from_now_new_value (valor)] the_intervals[interval], value:= valor;

preconditions
        insert_interval (interval, value)
              if (not exists a_interval in the_intervals
               where (not is_end_open (a_interval) impl
                      (endpoint (a_interval) > beginpoint (interval))
                       and (beginpoint(a_interval) < endpoint (interval)))
               and (not exists a_interval in the_intervals
                   where not is_begin_open (a_interval) impl
                          (endpoint (a_interval) > beginpoint (interval))
                         and (beginpoint(a_interval) < endpoint (interval)))

end class
```

Classe "it_int_value"

```
class it_int_value
variable attributes
        value: int;
        interval: time_interval;
end class
```

Classe "temporal_event"

A classe temporal "temporal_event" foi definida para representar eventos temporais.

```
class temporal_event
variable attributes
        the_event_list: event_occurrence_list list [1,*]

derived attributes
event_time_point: event_occurrence_list;

events
occurs;

end class
```

Esta classe apresenta um atributo *"the_event_list"* que é do tipo objeto (*class "event_occurrence_list"*), que gera uma lista das ocorrências dos eventos.

O atributo derivado *"event_time_point"* gera uma lista dinamicamente dos pontos no tempo em que o evento ocorreu. A seguir é apresentada a sintaxe deste atributo:

```
(<ref_objeto>.<id_evento>)
```

Onde ¡ref_objeto¿.¡id_evento¿ indica qual o evento que ocorreu. Esse evento tem que ser declarado como temporal.

O resultado da derivação é uma lista de pontos no tempo (*"event_occurrence-_list"*), nos quais o evento em questão foi disparado pelo objeto.

O evento *"occurs"*, é disparado quando um evento é ativado.

Classe "event_occurrence_list"

A classe *"event_occurrence_list"* define a ocorrência de um evento (quem ativou o evento e quando este evento ocorreu).

```
class event_occurrence_list
variable attributes
        the_events : temporal_event_occurrence;
events
        insert;
end class
```

O registro é feito através do evento *"insert"*.

Classe "temporal_event_occurrence"

A classe *"temporal_event_occurrence"* apresenta dois atributos conforme especificação a seguir:

```
class temporal_event_occurrence
variable attributes
        when : time_point;
        actor: object;
end class
```

O atributo *"when"* define quando ocorreu o evento, e o atributo *"actor"* define quem ativou o evento.

Na semântica de OASIS existe o conceito de "ações" que incluem três elementos: referência do cliente, referência do servidor e descrição do serviço. Assim, uma ação está definida pela tupla (cliente, servidor, serviço). Esse conceito é demonstrado na classe *"temporal_event_occurrence"*, através do atributo *"actor"*, que é do tipo objeto, isto é, o cliente que ativou a ocorrência do evento temporal.

O relacionamento entre as classes *"temporal_event"*, *"event_occurrence_list"* e *"temporal_event_occurrence"* são apresentadas a seguir através da especificação OASIS.

```
temporal_event  aggregation of
dynamic event_occurrence_list
                towards (1,*) from (1,1) list [1,*]

event_occurrence_list aggregation of
        dynamic inclusive temporal_event_occurrence
                towards (1,*) from (1,1) list [1,*]
```

4.2.5 Extensão das fórmulas de OASIS

As fórmulas em OASIS foram estendidas para inserir as expressões *after*, *before* que são utilizadas, principalmente, em especificações temporais. Além disso, as expressões *impl*, *for all* e *exists* foram estendidas para tratamentos de listas.

Através da extensão temporal de OASIS, é possível determinar eventos que ocorram condicionalmente a eventos que ocorreram no passado ou a estados que o objeto assumiu no passado.

```
<fórmula> ::= '(' <fórmula> ')' | not '(' <fórmula> ')'
              | <fórmula> and <fórmula>
              | <fórmula> or <fórmula>
              | <fórmula> impl <fórmula>
              | <time_point> after <time_point>
              | <time_point> before <time_point>
              | for all <atributo/componente> in <lista>
                            [where '{'<fórmula>'}']
              | exists <atributo/componente> in <lista>
                            [where '{'<fórmula>'}']
              | <expressão_aritmética> <op_rel> <expressão_aritmética>
              | true | false
<op_rel>  ::= '=' | '<>'| '<' | '<=' | '>' | '>='
```

A seguir, é apresentado um exemplo de uma pré-condição que condiciona a ocorrência de um evento a estados que o objeto assumiu no passado.

Na vídeo-locadora existem três categorias de usuário: ouro, prata e bronze. A categoria atual de um usuário é indicada pelo atributo "categoria". Quando um usuário associa-se a locadora, ele assume a categoria bronze.

Para que um usuário tenha um desconto de 10% no pagamento da locação de uma fita deve ser disparado o evento "desconto". A pré-condição para a ocorrência deste evento é que o usuário não tenha estado em débito durante o último ano. O fato de um usuário estar em débito é indicado pelo atributo temporal "devedor", que é verdadeiro quando o usuário encontra-se nesta condição.

Abaixo está especificada uma pré-condição que garante que um usuário não obtenha desconto se tiver sido devedor durante o último ano.

```
preconditions
  desconto (preço*0,90) if
             (categoria="ouro" or categoria="prata")
           impl for all d_value in devedor.the_values [now-(1 year), now]
             (not d_value)
```

A pré-condição especifica que o evento "desconto" somente pode ocorrer, caso a categoria seja ouro ou prata e que deve valer para todos os valores do atributo "devedor" durante o último ano, não sendo este valor verdadeiro (o usuário não esteve no estado de devedor).

4.2.6 Especificação para atributos e eventos temporais

Nesta subseção é mostrado como atributos e eventos temporais são especificados. Em outros termos, é mostrado como os conceitos das subseções anteriores são usados para especificar uma classe.

Utilizando a sintaxe original de OASIS, a especificação de um atributo temporal ("preço") e de um evento temporal ("emprestar") na classe fita da vídeo-locadora é a seguinte:

```
classe fita
...
variable attributes
        localização : string;
```

```
        lançamento: bool (true);
        preço: it_nat;
        emprestar: temporal_event;
...
events
        insere_fita new;
        ...
```

"Preço" é um atributo temporal com domínio *"it_nat"* representando o domínio dos números naturais com dimensão temporal por intervalo de tempo.

"Emprestar" é um evento temporal. Isto significa que o sistema irá manter o registro das ocorrências deste evento, associando a cada uma delas o seu tempo de transação.

Nesta sintaxe, para referenciar a ocorrência de um evento temporal, é necessário usar o termo abaixo:

```
fita.emprestar.occurs;
```

Ou seja, como eventos temporais são declarados como sendo atributos variáveis, seu tratamento difere dos eventos não temporais. Para contornar esta restrição e homogeneizar o tratamento de atributos e eventos temporais, introduz-se uma nova sintaxe para definição destes elementos através de seções para atributos temporais (temporal attributes) e eventos temporais (temporal events).

A sintaxe para especificação de classe passa a ser a seguinte:

```
<def_classe>    ::=    class <id_classe>
                       <mecanismos_identificação>
                       <atributos_constantes>
                       [ <atributos_variáveis> ]
                       [ <atributos_temporais>]
                       [ <atributos_derivados> ]
                       [ <derivações> ]
                       [ <restrições_integridade> ]
                       <eventos>
                       <eventos temporais>
                       [ <operações> ]
                       [ <gatilhos> ]
                       [ <avaliações> ]
                       [ <pré-condições>]
                       [ <protocolos> ]
                       end class
```

A classe fita mencionada anteriormente é especificada com a inserção das cláusulas temporais para atributos e eventos como segue:

```
variable attributes
        localização : string;
        lançamento: bool (true);
        ...
temporal attributes
        preço: nat;
        ...
events
        insere_fita new;
        ...
temporal events
        emprestar;
        ...
```

Esta notação pode ser considerada como uma forma abreviada da notação original de OASIS, sem extensão da sua semântica.

5 MODELO DE DADOS PARA SUPORTE À EXTENSÃO TEMPORAL DE OASIS NA ABORDAGEM RELACIONAL

Este capítulo discute algumas alternativas para a definição de um modelo de dados para suporte à extensão temporal (ZANATTA, 2000) feita na linguagem OASIS. O capítulo propõe um modelo de dados para atributos variáveis temporais e outro para eventos temporais. O modelo de dados proposto baseia-se na utilização de sistemas gerenciadores de banco de dados relacionais devido a ferramenta *CASE OO-Method* (PASTOR, 1992a; WAGNER; THOMA, 1996; PASTOR et al., 1997) fazer uso desse tipo de SGBD como meio de persistência de objetos.

5.1 Alternativas de implementação de modelos de dados temporais

A primeira consideração a ser feita com relação ao modelo de dados é como armazenar a informação sobre os elementos temporais daqueles atributos definidos como temporais na especificação formal feita na linguagem OASIS. Existem várias formas de acrescentar a dimensão temporal a um modelo de dados. Diversos modelos de dados temporais baseados no modelo relacional já foram propostos (THE HISTORICAL RELATIONAL DATA MODEL , HRDM; NAVATHE; AHMED, 1993; HSQL: A HISTORICAL QUERY LANGUAGE, 1993; SNODGRASS et al., 1996a,b; SNODGRASS, 1995). Dentre estes, (SNODGRASS et al., 1996a,b) constituem o foco principal desta pesquisa por apresentarem maior compatibilidade com a linguagem SQL. Esta compatibilidade é alcançada devido a características e conceitos apresentados na seção 3.2, a qual trata da linguagem ATSQL2.

5.1.1 Mesma tabela para um ou mais atributos temporais

A primeira alternativa para um modelo de dados que forneça suporte temporal é utilizar parte da abordagem de generalização temporal (STEINER, 1998). Isto significa que todas as colunas (linha inteira) de cada linha da tabela passam a ser temporais. Cada linha recebe um rótulo temporal que contém o intervalo do tempo de validade. Como visto anteriormente, um intervalo temporal é fechado no limite inferior e aberto no superior. Nessa abordagem não há uma tabela separada para armazenar a história das atualizações de cada linha da tabela. Assim, até mesmo os comandos SQL não-temporais deverão ser filtrados e modificados para selecionar somente a última versão de cada linha da tabela, tal como o conceito de compat-

ibilidade ascendente temporal (SNODGRASS et al., 1996b), que foi apresentado na seção 3.2. Entretanto, a vantagem seria aproveitar na íntegra os algoritmos de tradução de ATSQL2 para SQL apresentados na subseção 3.2.2. Outra importante consideração é que qualquer atualização na tabela deverá gerar uma nova linha, não importa qual seja o atributo atualizado. Isto pode ser uma vantagem se considerado o ponto de vista da disponibilidade de informação sobre os diversos estados da tabela no tempo. Também, pode ser uma desvantagem se considerado o ponto de vista do maior espaço de armazenamento utilizado. A figura 5.1 mostra um exemplo desta forma de implementação.

Consideremos a seguinte tabela não-temporal chamada **Empregado** *abaixo.*

Codigo	Nome	Salario	Cargo
231	Maria	2500	10
132	João	2000	11

Em 2 janeiro de 2001 foi feita a migração da tabela **Empregado** *de não-temporal para temporal com o seguinte comando em ATSQL2:*

```
ALTER TABLE Empregado ADD VALID;
```

O comando acima altera o esquema e o conteúdo da tabela **Empregado** *de modo que os tempos de validade das linhas são registrados com o conteúdo* **[02/01/2001 - ∞)**. *Em 29 de janeiro do mesmo ano, o seguinte comando SQL é executado:*

```
UPDATE Empregado SET Salario = 2300 WHERE Nome = 'João';
```

Tabela **Empregado** *após a migração e atualização:*

VALID	Codigo	Nome	Salario	Cargo
[02/01/2001 - ∞)	231	Maria	2500	10
[02/01/2001 - 29/01/2001)	132	João	2000	11
[29/01/2001 - ∞)	132	João	2300	11

Figura 5.1: Mesma tabela para um ou mais atributos temporais.

5.1.2 Uma nova tabela para um ou mais atributos temporais

Uma segunda alternativa seria armazenar em uma tabela separada, aqueles atributos definidos como temporais, juntamente com os rótulos temporais que contém o intervalo do tempo de validade. Como a tabela original não-temporal permanece inalterada em sua estrutura, os comandos SQL não-temporais não necessitam ser modificados para selecionar somente o estado mais atual da tabela. Outra vantagem é que somente aqueles atributos definidos como temporais deverão ter a sua história armazenada, o que deverá economizar espaço de armazenamento. A desvantagem seria a necessidade de adaptar o algoritmo de tradução de ATSQL2 para SQL de forma que os comandos acrescentem a tabela que contém a história das atualizações, mediante uma operação de junção, aos comandos traduzidos. A figura 5.2 mostra um exemplo desta forma de implementação.

Consideremos a seguinte tabela não-temporal chamada **Empregado** *abaixo.*

Codigo	Nome	Salario	Cargo
231	Maria	2500	10
132	Joao	2000	11

Para registrar a história das alterações de salário e de cargo existe uma outra tabela.

VALID	Codigo	Salario	Cargo
[02/01/2001 - ∞)	231	2500	10
[02/01/2001 - 29/01/2001)	132	2000	11
[29/01/2001 - ∞)	132	2300	11

Figura 5.2: Uma nova tabela para um ou mais atributos temporais.

5.1.3 Uma nova tabela para cada atributo temporal

A terceira alternativa seria armazenar cada atributo definido como temporal em uma tabela separada. Isto significa que deverá existir uma tabela para cada atributo temporal, além da tabela original. A vantagem desse modelo é que não haverá necessidade de realizar a operação de *coalescing* para eliminar rótulos temporais consecutivos diferentes para tuplas de valor equivalente, pois nunca haverá duas linhas com o mesmo valor devido à inexistência de outros atributos temporais que poderiam estar causando a inserção de uma nova linha em função de suas atualizações. A desvantagem seria a grande quantidade de tabelas existentes no banco de dados e a dificuldade de expressar consultas que envolvam vários atributos temporais. Também, existiria a necessidade de adaptar o algoritmo de tradução de ATSQL2 para SQL. A figura 5.3 mostra um exemplo desta forma de implementação.

Consideremos a seguinte tabela não-temporal chamada **Empregado** *abaixo.*

Codigo	Nome	Salario	Cargo
231	Maria	2500	10
132	Joao	2000	11

Para registrar a história das alterações de salário e de cargo existem duas novas tabelas, uma para cada atributo temporal.

VALID	Codigo	Salario
[02/01/2001 - ∞)	231	2500
[02/01/2001 - 29/01/2001)	132	2000
[29/01/2001 - ∞)	132	2300

VALID	Codigo	Cargo
[02/01/2001 - ∞)	231	10
[02/01/2001 - ∞)	132	11

Figura 5.3: Uma nova tabela para cada atributo temporal.

5.1.4 Eliminação dos atributos temporais da tabela de origem

Uma variante dos dois modelos anteriores é simplesmente a eliminação dos atributos temporais da tabela original. Essa pequena economia de espaço de armazenamento dificulta as consultas que necessitam apenas da última situação de um atributo temporal, pois deverão requerer junções e condições adicionais para obter os valores correspondentes ao instante *now* dos atributos temporais em questão. A figura 5.4 mostra um exemplo desta forma de implementação.

Consideremos a seguinte tabela não-temporal chamada **Empregado** *abaixo, sem os atributos temporais respectivos ao salário e cargo.*

Codigo	Nome
231	Maria
132	Joao

Os atributos temporais aparecem somente na tabela temporal.

VALID	Codigo	Salario	Cargo
[02/01/2001 - ∞)	231	2500	10
[02/01/2001 - 29/01/2001)	132	2000	11
[29/01/2001 - ∞)	132	2300	11

Figura 5.4: Eliminação dos atributos temporais da tabela de origem.

5.2 Proposta de modelo para implementação da extensão temporal de OASIS

No capítulo 4 observa-se que a extensão temporal (ZANATTA, 2000) feita em OASIS contempla somente o tempo de validade. Isto significa que devemos acrescentar somente o intervalo que corresponde ao tempo de validade. Na prática, para armazenar o intervalo, são necessários dois novos atributos, os quais deverão ser sempre definidos com os nomes *VTS_#$* e *VTE_#$* e que correspondem respectivamente aos tempos de início e fim do tempo de validade. A seqüência de caracteres *_#$*, que é utilizada no final dos nomes dos atributos, evita ou torna muito improvável que exista outro atributo, definido pelo usuário, que possua o mesmo nome. Em última análise, os nomes *VTS_#$* e *VTE_#$* devem ser vistos como palavras reservadas do modelo.

5.2.1 Definição da tabela temporal

Com relação à questão sobre manter o intervalo do tempo de validade na mesma ou em outra tabela, optou-se pela criação de uma outra tabela, que deverá possuir no mínimo quatro atributos definidos: *(VTS_#$, VTE_#$, OID, < NomeAtributo1 >, [< NomeAtributo2 >, ...])*. Para facilitar o entendimento no decorrer desta dissertação, a nova tabela será chamada de *tabela temporal* e a tabela original de *tabela não-temporal*. No presente modelo, os nomes dos atributos variáveis temporais deverão receber o sufixo *_#$* na tabela temporal.

Quanto ao nome de cada tabela temporal, estas deverão possuir o mesmo nome da respectiva tabela não-temporal acrescido da seqüência de caracteres *_#$*. Na

figura 5.5 é apresentado um esquema na forma de tabelas para melhor compreensão.

Consideremos a seguinte tabela não-temporal de nome **Empregado** *e que possui o salário definido como temporal:*

OID	Nome	Salario	Cargo	Setor
231	Henry	2.000,00	10	12
420	Carla	1.800,00	9	12

A tabela temporal que armazena história dos salários recebe o nome **Empregado_#$** *e possui os seguintes atributos:*

VTS_#$	VTE_#$	OID_#$	Salario_#$
01/08/1988	31/12/2000	231	1.500,00
31/12/2000	∞	231	2.000,00
01/05/1997	31/12/1999	420	1.000,00
31/12/1999	∞	420	1.800,00

Figura 5.5: Exemplo do modelo de dados para atributos variáveis temporais.

5.2.2 Duração do *chronon*

Em Zanatta (2000) a duração particular de um *chronon* foi deixada livre, a ser escolhida no momento da implementação do modelo de dados. Assim sendo, é vantajoso que o modelo suporte várias granularidades. Isto significa que os atributos *VTS_#$* e *VTE_#$*, para que possam suportar diferentes granularidades, devem ser armazenados com uma granularidade pequena, por exemplo, um segundo. Determina-se então, que os atributos *VTS_#$* e *VTE_#$* devem ser definidos como sendo do tipo inteiro com tamanho 16 e conterão o número de segundos desde a hora zero do dia 1º de janeiro do ano 1 até o instante no tempo que os mesmos estão representando. Logo, o *software* tradutor de ATSQL2 para SQL deve efetuar a conversão das datas para segundos de forma apropriada. A linguagem C possui funções para essa finalidade.

5.2.3 Representação do limite superior em um intervalo aberto

Outra consideração importante é como representar um intervalo aberto em *VTE_#$*. Ele deve representar o momento presente. Os dados associados ao valor representado por "∞" em *VTE_#$* são equivalentes à última situação em uma tabela não-temporal. Quando contiver "∞", é necessário que *VTE_#$*, ao ser comparado com outro rótulo temporal, sempre seja maior ou igual em relação ao outro atributo. Assim, define-se que "∞" deve ser representado por um valor constante e que o referido valor seja igual ao maior valor possível para um campo do tipo inteiro de 16 algarismos. Desse modo, o valor 9999999999999999, quanto atribuído à *VTE_#$*, representa um intervalo aberto. O modelo não suporta intervalos abertos no início do tempo de validade.

5.2.4 Indexação

Outra importante questão a ser analisada no modelo é quais atributos devem receber um índice associado na tabela temporal. A utilização ou não do índice deverá influenciar apenas na otimização do acesso aos dados históricos armazenados na tabela temporal. Por isso, a criação do índice será deixada livre, a ser feita a critério do DBA. Entretanto, recomenda-se que a criação do índice seja feita apenas com um índice por tabela temporal, concatenando os atributos *OID* e *VTS_#$*. Desse modo, obteremos uma ordenação automática do tempo para cada *OID*. Tal índice, deve garantir também a restrição de integridade de chave ao ser definido como único ou como chave primária, pois a combinação de *OID* e *VTS_#$* nunca poderá se repetir na mesma tabela.

5.2.5 Definição da visão temporal

Finalmente, para simplificar o *software* de tradução de ATSQL2 para SQL, o modelo prevê a criação de uma visão que faça a junção da tabela não-temporal com a tabela temporal. O nome da visão é o mesmo da tabela temporal acrescido dos caracteres "V_" na frente. A figura 5.6 apresenta a seqüência de comandos completa de criação de uma tabela que possua atributos variáveis temporais.

Tabela de nome **Empregado** *com as colunas Cargo, Setor e Salario definidos como atributos variáveis temporais.*

```
CREATE TABLE EMPREGADO (
  OID        NUMBER (10) PRIMARY KEY,
  NOME       CHAR   (32) NOT NULL,
  CARGO      NUMBER (03),
  SETOR      NUMBER (03),
  SALARIO    NUMBER (05,2) );

CREATE TABLE EMPREGADO_#$ (
  VTS_#$     NUMBER (16),
  VTE_#$     NUMBER (16),
  OID_#$     NUMBER (10),
    REFERENCES EMPREGADO (OID),
  CARGO_#$   NUMBER (03),
  SETOR_#$   NUMBER (03),
  SALARIO_#$ NUMBER (05,2),
  CONSTRAINT EMPREGADO_#$_PK
    PRIMARY KEY (OID_#$, VTS_#$) );

CREATE VIEW V_EMPREGADO_#$
  (VTS_#$, VTE_#$, OID_#$, NOME_#$, CARGO_#$, SETOR_#$,
   SALARIO_#$)
AS
  SELECT VTS_#$, VTE_#$, OID, NOME, CARGO_#$, SETOR_#$,
   SALARIO_#$
  FROM EMPREGADO, EMPREGADO_#$
  WHERE EMPREGADO.OID = EMPREGADO_#$.OID_#$;
```

Figura 5.6: Exemplo da seqüência de comandos para criação de uma tabela com atributos variáveis temporais

5.3 Modelo de dados para suporte a eventos temporais

Um evento é um instante isolado no tempo (SOO; SNODGRASS, 1995). A extensão temporal feita em OASIS (ZANATTA, 2000) permite que sejam especificados eventos temporais. Por exemplo, em uma vídeo-locadora o "empréstimo" de uma fita constitui um evento e passaria a ser considerado temporal. Isto significa que o registro de todas as ocorrências do evento deve ser mantido. Desta forma, o usuário pode consultar o histórico das ocorrências do evento através de uma linguagem de consulta temporal. Com isto, é possível obter informações, tais como, o fato de um usuário da vídeo-locadora poder receber um cartão de cliente especial, caso tenha alugado um determinado número de fitas em um período, ou um filme poder ter seu preço de aluguel reduzido, caso ocorram poucos empréstimos em um período.

Como a execução de serviços e transações são processos atômicos, o que interessa é o instante no tempo em que ocorreu determinado evento. Sendo assim, o modelo deve registrar o tempo de transação em que um evento de uma determinada classe ocorreu para um objeto específico.

5.4 Alternativas de implementação para suporte a eventos temporais

Em Meneses et al (1999) são apresentadas duas alternativas para definir o modelo de dados para dar suporte ao registro de eventos temporais em OASIS. Neste trabalho foram identificadas três alternativas para definir o modelo de dados de eventos temporais:

1. criar apenas uma tabela para registrar todos os eventos de todas as classes;
2. criar uma tabela para cada classe que possua ao menos um evento;
3. criar uma tabela para cada evento de cada classe.

A opção pela primeira, pela segunda ou pela terceira alternativa deve influenciar significativamente na elaboração do algoritmo de tradução da linguagem temporal para o padrão SQL, como explicado nas subseções a seguir.

5.4.1 Uma tabela para todos os eventos

A primeira alternativa é criar apenas uma tabela para registrar todos os eventos de todas as classes de uma base de dados. Nesse modelo o nome da tabela permanece o mesmo em todas as consultas, a cláusula *WHERE* deve ser mais detalhada para selecionar apenas as linhas que se referem a um determinado evento de uma classe. A figura 5.7 apresenta uma tabela capaz de armazenar informação sobre as ocorrências de quaisquer eventos em um sistema de vídeo-locadora.

5.4.2 Uma tabela para cada classe

Na segunda alternativa deve existir uma tabela do tipo evento para cada classe que possua pelo menos um evento definido. Isso significa que deve haver um esquema para atribuição de nomes para as tabelas do tipo evento. A cláusula *WHERE* não deve conter o critério de seleção do nome da classe, pois o mesmo já está resolvido na referência ao nome da tabela do tipo evento. A figura 5.8 apresenta duas tabelas que registram as ocorrências de todos os eventos de cada classe respectiva.

Tabela de nome **e_Evento_#$** *de um sistema de vídeo-locadora. No dia* **18/01/2002** *o* **usuário** *de código* **231** *efetuou um* **empréstimo** *e uma* **fita** *de código* **1230** *foi* **emprestada** *no dia* **18/01/2002**. *O mesmo* **usuário** *efetuou uma* **devolução** *no dia* **19/01/2002** *e a mesma* **fita** *foi* **devolvida** *nessa* **mesma data**.

TT	Classe	Evento	OID
18/01/2002	Usuario	Emprestar	231
19/01/2002	Usuario	Devolver	231
18/01/2002	Fita	Emprestar	1230
19/01/2002	Fita	Devolver	1230

Figura 5.7: Uma tabela para todos os eventos.

Tabelas de nomes **e_Usuario_#$** *e* **e_Fita_#$** *representando as ocorrências dos mesmos eventos da figura 5.7.*

Tabela e_Usuario_#$

TT	Evento	OID
18/01/2002	Emprestar	231
19/01/2002	Devolver	231

Tabela e_Fita_#$

TT	Evento	OID
18/01/2002	Emprestar	1230
19/01/2002	Devolver	1230

Figura 5.8: Uma tabela para cada classe.

5.4.3 Uma tabela para cada evento

A terceira e última alternativa é criar uma tabela para cada um dos eventos temporais de cada classe. A escolha torna bastante complexa a atribuição de nomes para as tabelas do tipo evento, pois estas devem possuir um esquema de atribuição de nomes envolvendo o nome da classe mais o nome do evento, o que aumenta bastante a possibilidade de algum nome de tabela do tipo evento entrar em conflito com outro nome de tabela definido pelo usuário. A cláusula *WHERE*, entretanto, é simplificada, pois os nomes da classe e do evento já estão resolvidos no nome da tabela do tipo evento. A figura 5.9 apresenta quatro tabelas, cada uma registrando as ocorrências de apenas um evento.

Uma tabela para cada um dos eventos representados nas figuras 5.7 e 5.8.

Tabela e_Usuario_Emprestar_#$

TT	OID
18/01/2002	231

Tabela e_Usuario_Devolver_#$

TT	OID
19/01/2002	231

Tabela e_Fita_Emprestar_#$

TT	OID
18/01/2002	1230

Tabela e_Fita_Devolver_#$

TT	OID
19/01/2002	1230

Figura 5.9: Uma tabela para cada evento.

5.5 Proposta de um modelo de dados para eventos temporais

Para definir o modelo de dados para suporte a eventos temporais optou-se pela criação de apenas uma tabela para registrar todos os eventos. A escolha simplifica a implementação dos gatilhos que atualizam a tabela de eventos e evita a execução de comandos de definição de dados para cada tabela que possua eventos definidos. Com isso, exclui-se também a necessidade de implementação de um sistema de atribuição de nomes para as tabelas evento. Como explicado na seção anterior, a única desvantagem é que as consultas devem conter critérios de seleção adicionais para determinar as classes e eventos que estão sendo consultados.

5.5.1 Definição da tabela evento

A tabela única para registro de eventos deve possuir quatro atributos, são eles: *TT*, *Classe*, *Evento* e *OID*. Define-se que o nome da tabela evento é *e_Evento_#$*. O prefixo *e_* e o sufixo *_#$*, acrescentados ao nome, tem a finalidade de evitar conflitos com outros nomes definidos pelo usuário no banco de dados. A seguir, o comando de definição de dados da tabela única para registro de eventos (Figura 5.10).

```
CREATE TABLE e_Evento_#$ (
  TT         NUMBER (16),
  CLASSE     CHAR(30),
  EVENTO     CHAR(30),
  OID        NUMBER(16) );
```

Figura 5.10: Comando de definição de dados da tabela evento.

5.5.2 Indexação

A tabela de eventos *e_Evento_#$* provavelmente receberá a inserção de uma grande quantidade de linhas. Pela utilização de índices, pode-se evitar a inserção de linhas idênticas, bem como diminuir o tempo de resposta das consultas.

Para evitar a inserção de linhas idênticas e, ao mesmo tempo, diminuir o tempo de resposta das consultas e ainda obter uma ordenação automática pelo tempo de transação para cada objeto, é necessário a criação de uma chave primária que concatene todas as colunas da tabela de eventos. Para isso, recomenda-se a criação de uma chave primária que concatene, nessa ordem, os atributos C*lasse*, *Evento*, *OID* e *TT* (Figura 5.11).

```
CREATE UNIQUE INDEX e_Evento_#$_PK
  (Classe, Evento, OID, TT);
```

Figura 5.11: Criação do índice da tabela de eventos.

6 MAPEAMENTO DE ATSQL2 PARA SQL

Este capítulo apresenta uma extensão da ATSQL2 para tratamento de eventos temporais, discute algumas formas de implementação do *software* tradutor de ATSQL2 para SQL e apresenta, através de exemplos, uma maneira de efetuar o mapeamento de comandos de consulta ATSQL2 para SQL. O mapeamento é adequado ao modelo de dados proposto no capítulo 5. Como visto anteriormente, a abordagem de generalização temporal (STEINER, 1998), por ser muito radical, não será adotada. Ao invés de converter tabelas não-temporais em temporais e, por esse motivo, ser obrigado a modificar até mesmo os comandos SQL convencionais, optou-se pela criação de uma tabela separada para armazenar a história das atualizações, ficando a tabela original (não-temporal) sem qualquer modificação em sua estrutura.

6.1 Extensão da ATSQL2 para tratamento de eventos temporais

A extensão da ATSQL2 introduz um novo *timeflag* chamado *EVENT*, que deverá aparecer na frente do comando *SELECT*. Após o *timeflag* podem aparecer as palavras *ALL*, *FIRST*, *LAST* ou um intervalo de tempo. A palavra *ALL*, que é assumida por *default*, indica que todos os eventos devem ser selecionados. A palavra *FIRST* indica que somente o primeiro evento deve ser selecionado. A palavra *LAST* informa que somente o último evento deve ser selecionado. A especificação do intervalo pode ser feita no mesmo lugar das palavras *ALL*, *FIRST* e *LAST* para indicar que somente os eventos que ocorreram dentro do intervalo devem ser selecionados. Após, entre parênteses, devem ser informados o nome da classe, o nome do evento e o *OID* do objeto. No lugar do *OID* pode-se ainda especificar uma *subquery* que resulte em um *OID*. Finalmente, pode ser especificado um comando *SELECT* qualquer que contenha uma tabela com nome igual ao nome da classe. A Figura 6.1 apresenta a sintaxe do comando.

```
EVENT [ALL]|FIRST|LAST|<exp intervalo>
 (<classe>,<nome_evento>,<oid>|(<subquery)>)
 [ SELECT ... ]
```

Figura 6.1: Extensão da ATSQL2 para tratamento de eventos temporais.

Caso não apareça o comando *SELECT*, a saída do comando será uma relação com apenas uma coluna contendo os instantes no tempo de cada evento. Por outro lado, caso seja fornecido o comando *SELECT*, a saída do comando deverá ser o

resultado da junção dos instantes do evento com as colunas referenciadas no comando *SELECT*. No exemplo abaixo é apresentado um comando que consulta o último evento ***Emprestar*** de cada usuário categoria ***ouro*** de uma vídeo-locadora:

```
EVENT LAST
  (Usuario, Emprestar, (SELECT OID
                        FROM USUARIO
                        WHERE CATEGORIA = 'OURO'))
   SELECT NOME
   FROM USUARIO;
```

Consideremos as tabelas a seguir como base para execução do comando acima.

Tabela Usuario:

OId	Nome	Categoria
231	João	OURO
123	Ana	PRATA
1230	Silvia	OURO
456	Lúcia	PRATA

Tabela e_Eventos_#$:

TT	Classe	Evento	OID
18/01/2002	Usuario	Emprestar	231
19/01/2002	Usuario	Devolver	231
18/01/2002	Fita	Emprestar	1230
19/01/2002	Fita	Devolver	1230
20/01/2002	Usuario	Emprestar	231
20/01/2002	Usuario	Emprestar	1230

A saída do comando é a seguinte:

e.TT	e.Nome
20/01/2002	João
20/01/2002	Sílvia

6.2 Formas de implementação do software tradutor

Analisando pelo ponto de vista da arquitetura cliente/servidor de banco de dados, um software adicional precisa ser implementado entre o cliente e o servidor. O *software* intermediário fará a tradução da ATSQL2 (escrita pelo usuário utilizando o *software* cliente) para o padrão SQL (utilizado pelos sistemas de bancos de dados interpretado pelo servidor). O software tradutor receberá como entrada uma consulta escrita em ATSQL2 e fará a tradução para a SQL correspondente àquela consulta, colocando-a como saída final do processo.

O uso do *software* tradutor deverá acrescentar duas importantes vantagens quando da execução de consultas que envolvem aspectos temporais:

- evita que o servidor de banco de dados tenha que interpretar uma linguagem de consulta temporal, o que implicaria na substituição dos SGBDs convencionais por outros capazes de executar uma linguagem de consulta temporal;

- evita que o usuário escreva comandos de consulta SQL complexos envolvendo aspectos temporais, pois uma consulta com aspectos temporais, quando traduzida para a SQL, resulta em um código bem maior e mais complexo.

Existem diferentes formas ou lugares onde o *software* tradutor pode ser implementado. Dependendo da forma de implementação escolhida haverá vantagens e desvantagens. A seguir, são apresentadas as diferentes formas de implementação possíveis:

1. implementação no cliente na forma de um programa convencional, o qual deve ser chamado pelo usuário tal como um comando do sistema operacional local. O programa recebe como entrada uma consulta em ATSQL2 e escreve na saída em SQL;

2. implementação no cliente na forma de um programa residente em memória. O programa atende requisições monitorando uma ação do usuário, lendo da área de transferência (padrão *Windows*) a consulta temporal escrita pelo usuário e escrevendo, de volta, na área de transferência a consulta convertida para SQL;

3. implementação de um programa cliente completo, tal como SQL*Plus da Oracle, a diferença é que antes de enviar os comandos para o SGBD a tradução já seria efetuada;

4. implementação em servidor de conversão independente. O programa é implementado na forma de um *daemon* que monitora um determinado número de porta (padrão TCP/IP) do nível de transporte da arquitetura TCP/IP. O *daemon* solicita os serviços da camada de transporte do protocolo de rede para estabelecimento da conexão, recebimento da consulta temporal, entrega da consulta em SQL e encerramento da conexão. Neste caso, um software cliente implementado na primeira ou na segunda forma de implementação, apresentadas anteriormente, seria necessário;

5. implementação no cliente na forma de um redirecionador. O programa monitora o tráfego entre cliente e servidor verificando se o comandos possuem algum dos *timeflags* da ATSQL2. Se o resultado da verificação for verdadeiro, o redirecionador, ao invés de passar a consulta diretamente para o servidor, desvia a consulta para o programa responsável pela tradução, a qual, somente após traduzida, será entregue ao servidor. Este tipo de implementação requer a modificação do *middleware* de rede, tal como o SQL*Net da Oracle ou um *middleware* padrão ODBC. O *middleware* ficaria responsável por analisar o tráfego e desviar ou não para o *software* tradutor;

6. implementação no servidor de banco de dados. O funcionamento é idêntico ao ítem 5, com a única diferença de que o redirecionador e o *software* de tradução estão sendo executados no mesmo computador onde está sendo executado o SGBD.

A primeira forma de implementação distingue-se das demais por possuir somente o núcleo para resolução do problema, o qual deverá ser reutilizado pelas demais formas de implementação. As demais formas apresentam outras vantagens e desvantagens, além do critério de facilidade de implementação, o qual definiu a ordem em que

foram apresentadas. Se as formas de implementação forem analisadas cuidadosamente, outros critérios, tais como performance do servidor, performance do cliente e tráfego de rede serão identificados. Entretanto, o fator mais importante seria a capacidade do *software* tradutor traduzir os comandos ATSQL2 ou deixar como estão os comandos SQL de forma automática e transparente antes que eles sejam interpretados pelo SGBD. Assim, as aplicações já existentes continuariam funcionando sem qualquer modificação.

As formas de implementação números 5 e 6, apresentadas acima, são as únicas que possuem a capacidade de transparência. Na prática, essas duas formas requerem a implementação de mais uma camada de *software* (*middleware*) que funcionaria como um *gateway,* simulando as *interfaces* de *middleware* tanto do cliente quanto do servidor, de modo a fazê-las "pensar" que estão interagindo com as *interfaces* reais.

6.3 Algoritmo de tradução

O algoritmo de tradução de consultas ATSQL2 para SQL é o mesmo apresentado em (STEINER, 1998) com algumas modificações adequadas ao modelo de dados proposto e apresentado nesta dissertação.

O conceito de compatibilidade ascendente (BÖHLEN; JENSEN; SNODGRASS, 1995) e compatibilidade ascendente temporal (SNODGRASS et al., 1996b) não requerem a intervenção do software tradutor, pois o conceito de generalização temporal não foi adotado em sua totalidade no modelo de dados proposto no presente trabalho, ficando a história das atualizações dos atributos variáveis temporais armazenada em uma tabela separada e não necessitando, portanto, da modificação dos próprios comandos SQL para selecionar somente a última situação das tabelas.

Para descobrir se um comando está escrito em ATSQL2 ou SQL basta verificar o primeiro *token* do comando. Se ele for igual a *VALID*, *NONSEQUENCED* ou *EVENT*, trata-se de um comando ATSQL2 e passível de verificação de sintaxe e de tradução. Senão, trata-se de um comando SQL e deve ser repassado sem qualquer modificação para o SGBD.

6.3.1 Exemplo de tradução de um comando seqüenciado

A seguir é apresentado o exemplo de um comando seqüenciado, escrito em ATSQL2, e como é feita a tradução para SQL. No exemplo considera-se a existência de uma tabela de nome *Empregado* que possui a coluna *Salário* definida como temporal. Segundo o modelo de dados proposto, apresentado anteriormente, a história das atualizações de salário dos empregados será armazenada em outra tabela, cujo nome deverá ser *Empregado_#$*. O modelo também prevê a criação de uma visão que deverá efetuar a junção da tabela *Empregado* com a tabela *Empregado_#$* e o seu nome será *v_Empregado_#$*. Para melhor compreensão, as tabelas, a visão e o comando com a respectiva tradução estão representadas a seguir:

Tabela ***Empregado***:

OID	Nome	Salario	Setor
231	Maria	2500	10
132	João	2000	11

Tabela ***Empregado_#$****:*

VTS_#$	VTE_#$	OID_#$	Salario_#$	Setor_#$
02/01/2001	∞	231	2500	10
02/01/2001	29/01/2001	132	2000	11
29/01/2001	∞	132	2300	11

Visão ***v_Empregado_#$***:

VTS_#$	VTE_#$	OID_#$	Nome_#$	Salario_#$	Setor_#$
02/01/2001	∞	231	Maria	2500	10
02/01/2001	29/01/2001	132	João	2000	11
29/01/2001	∞	132	João	2300	11

O seguinte comando seqüenciado:

```
VALID
  SELECT Nome, Salario
  FROM Empregado
  WHERE Salario > 2300;
```

Resulta na seguinte tradução:

```
SELECT VTS_#$ VTS, VTE_#$ VTE, Nome_#$ Nome, Salario_#$ Salario
  FROM v_Empregado_#$
  WHERE Salario_#$ > 2300;
```

O resultado da consulta é o seguinte:

VTS	VTE	Nome	Salario
02/01/2001	∞	Maria	2500
29/01/2001	∞	João	2300

Conforme mostrado acima, percebe-se que a visão *v_Empregado_#$* possui todas as colunas renomeadas de modo que o sufixo *_#$* apareça ao final do nome de todas as colunas. O comando traduzido para SQL renomeia as colunas novamente para os seus nomes originais para não prejudicar as aplicações que as utilizam.

6.3.2 Exemplo de tradução de um comando não-seqüenciado

Como visto anteriormente, um comando não-seqüenciado é aquele que compara diferentes estados da base de dados. Via de regra, esse tipo de consulta é feita com a utilização de produtos cartesianos. No exemplo que será mostrado a seguir, deve-se considerar as mesmas tabelas da subseção anterior.

Deseja-se encontrar quando os dados sobre os empregados foram alterados com respeito ao tempo de validade. Por exemplo, a alteração de ***salário*** *de* ***João*** *ficou registrada na tabela temporal* ***Empregado_#$****. Isto significa que devemos encontrar as tuplas contendo informação sobre os mesmos empregados que possuem intervalos de tempo de validade adjacentes (meeting).*

```
NONSEQUENCED VALID
  SELECT BEGIN(VALID(a2)), a2.Nome
  FROM Empregado a1, Empregado a2
  WHERE a1.OID = a2.OID
  AND   VALID(a1) meets VALID(a2);
```

Comando traduzido para SQL:

```
SELECT a2.VTS_#$ a2.VTS, a2.Nome_#$ a2.Nome
  FROM v_Empregado_#$ a1, v_Empregado_#$ a2
  WHERE a1.OID_#$ = a2.OID_#$
  AND   a1.VTE_#$ = a2.VTS_#$;
```

O resultado da consulta é o seguinte:

a2.VTS	a2.Nome
29/01/2001	João

6.3.3 Exemplo estendido

Nesta subseção será mostrado como um comando ATSQL2 complexo pode ser traduzido para SQL. O comando a ser traduzido contém comandos seqüenciados, *subqueries*, operação de conjunto união e operação de coalescência.

Nesse exemplo pretende-se mostrar o tempo de validade com períodos maximais (coalescência), o nome, o salário e o setor dos empregados que possuem o maior salário do setor 10 e os mesmos dados sobre os empregados que possuem o menor salário do setor 11 com base nas tabelas abaixo. O exemplo foi adaptado de Steiner (1998).

Tabela *Empregado*:

OID	Nome	Salário	Setor	Cargo
231	Maria	2500	10	1
132	João	2000	11	1
250	Carla	1000	10	2
500	José	1500	11	3

Tabela *Empregado_#$*:

VTS_#$	VTE_#$	OID_#$	Salario_#$	Setor_#$	Cargo_#$
02/01/2001	31/12/2001	231	2500	10	1
31/12/2001	∞	231	2700	10	1
02/01/2001	29/01/2001	132	2000	11	1
29/01/2001	∞	132	2300	11	1
28/02/2001	31/12/2001	250	1000	10	2
31/12/2001	∞	250	1200	10	2
01/01/2001	31/12/2001	500	1500	11	3
31/12/2001	31/01/2002	500	1700	11	3
31/01/2002	∞	500	1700	11	2

Para obter a consulta o comando ATSQL2 é o seguinte:

```
VALID
  (SELECT e1.Nome, e1.Salario, e1.Setor
   FROM Empregado e1
   WHERE e1.Setor = 10
   AND NOT EXISTS (SELECT {*}
                   FROM Empregado e2
                   WHERE e1.Salario < e2.Salario
                   AND e2.Setor = 10)
 UNION
  SELECT e1.Nome, e1.Salario, e1.Setor
   FROM Empregado e1
   WHERE e1.Setor = 11
   AND NOT EXISTS (SELECT {*}
                   FROM Empregado e2
                   WHERE e1.Salario > e2.Salario
                   AND e2.Setor = 11)) (PERIOD);
```

O resultado desta consulta é a seguinte tabela:

e1.VTS	e1.VTE	e1.Nome	e1.Salario	e1.Setor
31/12/2001	∞	Maria	2500	10
01/01/2001	31/12/2001	José	1500	11

O *timeflag VALID* refere-se ao resultado da união dos dois blocos *SELECT-FROM-WHERE*, os quais devem ser avaliados separadamente. Ao resultado da união deve ser aplicada a operação de *coalescing*, pois a palavra (*PERIOD*) aparece somente no final do comando, após o fechamento dos parênteses que unem os dois blocos *SELECT-FROM-WHERE*. As *subqueries* são avaliadas traduzindo-se cada uma delas nas operações de conjunto diferença de tempo de validade, conforme descrito anteriormente. A expressão de álgebra do comando pode ser expressa da seguinte maneira:

$$coalesce(\pi[vts, vte, Nome, Salario, Setor](aux1 -^v aux2) \cup^v \pi[vts, vte, Nome, Salario, Setor](aux3 -^v aux4))$$

A expressão de álgebra é avaliada em diversas etapas. Primeiro, a tabela *aux1* é criada contendo o resultado do primeiro comando *SELECT* sem a *subquery:*

```
CREATE TABLE aux1 (vts_#$, vte_#$,
                   nome_#$, salario_#$, setor_#$)
AS
SELECT vts_#$, vte_#$, nome_#$, salario_#$, setor_#$
FROM v_Empregado_#$
WHERE Setor_#$ = 10;
```

A próxima etapa é criar a tabela *x_Empregado_#$* com o resultado do produto cartesiano da visão *v_Empregado_#$* com ela própria executando os critérios de seleção da *subquery* e, em seguida, criar a tabela *aux2* com a projeção de todos os atributos pertencentes ao primeiro argumento do produto cartesiano:

```
CREATE TABLE x_Empregado_#$ (vts_#$, vte_#$,
     vts#$_1, vte#$_1, oid#$_1, nome#$_1, salario#$_1, setor#$_1, cargo#$_1,
     vts#$_2, vte#$_2, oid#$_2, nome#$_2, salario#$_2, setor#$_2, cargo#$_2
AS
SELECT DISTINCT GREATEST(e1.vts_#$, e2.vts_#$) vts_#$,
                LEAST   (e1.vte_#$, e2.vte_#$) vte_#$,
     e1.vts_#$, e1.vte_#$, e1.oid_#$, e1.nome_#$, e1.salario_#$, e1.setor_#$,
     e2.vts_#$, e2.vte_#$, e2.oid_#$, e2.nome_#$, e2.salario_#$, e2.setor_#$,
FROM v_Empregado_#$ e1, v_Empregado_#$ e2
WHERE GREATEST(e1.vts_#$, e2.vts_#$) < LEAST(e1.vte_#$, e2.vte_#$)
AND   e1.Salario_#$ < e2.Salario_#$
AND   Setor_#$ = 10;

CREATE TABLE aux2 (vts_#$, vte_#$,
                   nome_#$, salario_#$, setor_#$)
AS
SELECT vts#$_1, vte#$_1,
       nome#$_1, salario#$_1, setor#$_1
FROM x_Empregado_#$;

DROP TABLE x_Empregado_#$;
```

Agora é possível calcular a operação de conjunto diferença temporal das tabelas *aux1* e *aux2*.

```
INSERT INTO aux1
  SELECT a1.vts_#$, a2.vts_#$,
         a1.nome_#$, a1.salario_#$, a1.setor_#$
  FROM aux1 a1, aux2 a2
  WHERE a2.vts_#$     > a1.vts_#$
  AND   a2.vts_#$     < a1.vte_#$
  AND   a1.nome_#$    = a2.nome_#$
  AND   a1.salario_#$ = a2.salario_#$
  AND   a1.setor_#$   = a2.setor_#$;

INSERT INTO aux1
  SELECT a2.vte_#$, a1.vte_#$,
         a1.nome_#$, a1.salario_#$, a1.setor_#$
  FROM aux1 a1, aux2 a2
  WHERE a2.vte_#$     > a1.vts_#$
  AND   a2.vte_#$     < a1.vte_#$
  AND   a1.nome_#$    = a2.nome_#$
  AND   a1.salario_#$ = a2.salario_#$
  AND   a1.setor_#$   = a2.setor_#$;
```

```
DELETE FROM aux1 a1
WHERE EXISTS (SELECT a2.{*}
              FROM aux2 a2
              WHERE ((a1.vts_#$ >= a2.vts_#$ AND a1.vts_#$ < a2.vte_#$) OR
                     (a2.vts_#$ >= a1.vts_#$ AND a2.vts_#$ < a1.vte_#$))
              AND   a1.nome_#$    = a2.nome_#$
              AND   a1.salario_#$ = a2.salario_#$
              AND   a1.setor_#$   = a2.setor_#$;
```

A próxima etapa é efetuar a projeção dos rótulos temporais e dos atributos *Nome*, *Salario* e *Setor* conforme especificado no comando *SELECT* que faz parte do primeiro argumento da operação de união. O resultado da projeção é armazenado na tabela *aux5*.

```
CREATE TABLE aux5 (vts_#$, vte_#$, nome_#$, salario_#$, setor_#$)
AS
SELECT vts_#$, vte_#$, nome_#$, salario_#$, setor_#$
FROM aux1;
```

Estando a primeira parte da operação de união concluída, pode-se efetuar os mesmos procedimentos para obter o resultado do segundo argumento da operação de união, visto que o mesmo é praticamente idêntico ao primeiro argumento, mudando apenas a cláusula *WHERE*.

```
CREATE TABLE aux3 (vts_#$, vte_#$,
                   nome_#$, salario_#$, setor_#$)
AS
SELECT vts_#$, vte_#$, nome_#$, salario_#$, setor_#$
FROM v_Empregado_#$
WHERE Setor_#$ = 11;

CREATE TABLE x_Empregado_#$ (vts_#$, vte_#$,
     vts#$_1, vte#$_1, oid#$_1, nome#$_1, salario#$_1, setor#$_1, cargo#$_1,
     vts#$_2, vte#$_2, oid#$_2, nome#$_2, salario#$_2, setor#$_2, cargo#$_2
AS
SELECT DISTINCT GREATEST(e1.vts_#$, e2.vts_#$) vts_#$,
                LEAST   (e1.vte_#$, e2.vte_#$) vte_#$,
     e1.vts_#$, e1.vte_#$, e1.oid_#$, e1.nome_#$, e1.salario_#$, e1.setor_#$,
     e2.vts_#$, e2.vte_#$, e2.oid_#$, e2.nome_#$, e2.salario_#$, e2.setor_#$,
FROM v_Empregado_#$ e1, v_Empregado_#$ e2
WHERE GREATEST(e1.vts_#$, e2.vts_#$) < LEAST(e1.vte_#$, e2.vte_#$)
AND   e1.Salario_#$ > e2.Salario_#$
AND   Setor_#$ = 11;

CREATE TABLE aux4 (vts_#$, vte_#$,
                   nome_#$, salario_#$, setor_#$)
AS
SELECT vts#$_1, vte#$_1,
       nome#$_1, salario#$_1, setor#$_1
FROM x_Empregado_#$;

DROP TABLE x_Empregado_#$;

INSERT INTO aux3
  SELECT a1.vts_#$, a2.vts_#$,
         a1.nome_#$, a1.salario_#$, a1.setor_#$
  FROM aux3 a1, aux4 a2
```

```
  WHERE a2.vts_#$     > a1.vts_#$
  AND   a2.vts_#$     < a1.vte_#$
  AND   a1.nome_#$    = a2.nome_#$
  AND   a1.salario_#$ = a2.salario_#$
  AND   a1.setor_#$   = a2.setor_#$;

INSERT INTO aux3
  SELECT a2.vte_#$, a1.vte_#$,
         a1.nome_#$, a1.salario_#$, a1.setor_#$
  FROM aux3 a1, aux4 a2
  WHERE a2.vte_#$     > a1.vts_#$
  AND   a2.vte_#$     < a1.vte_#$
  AND   a1.nome_#$    = a2.nome_#$
  AND   a1.salario_#$ = a2.salario_#$
  AND   a1.setor_#$   = a2.setor_#$;

DELETE FROM aux3 a1
WHERE EXISTS (SELECT a2.*
              FROM aux4 a2
              WHERE ((a1.vts_#$ >= a2.vts_#$ AND a1.vts_#$ < a2.vte_#$) OR
                     (a2.vts_#$ >= a1.vts_#$ AND a2.vts_#$ < a1.vte_#$))
              AND   a1.nome_#$    = a2.nome_#$
              AND   a1.salario_#$ = a2.salario_#$
              AND   a1.setor_#$   = a2.setor_#$;

CREATE TABLE aux6 (vts_#$, vte_#$, nome_#$, salario_#$, setor_#$)
AS
SELECT vts_#$, vte_#$, nome_#$, salario_#$, setor_#$
FROM aux3;
```

Estando concluídos os dois resultados de ambos argumentos da operação de união, pode-se finalmente efetuar essa operação de conjunto armazenando o resultado na tabela *uAux*.

```
CREATE TABLE uAux (vts_#$, vte_#$,
                   nome_#$, salario_#$, setor_#$)
AS
SELECT *
FROM aux5
UNION
SELECT *
FROM aux6;
```

Finalmente, deve-se realizar a operação de coalescência na tabela intermediária *uAux* e apresentar o resultado para o usuário.

```
** REPEAT **
UPDATE uAux e1
  SET (e1.vte_#$) = SELECT MAX(e1.vte_#$)
                    FROM uAux a1
                    WHERE e1.nome_#$    = a1.nome_#$
                    AND   e1.salario_#$ = a1.salario_#$
                    AND   e1.setor_#$   = a1.setor_#$
                    AND   e1.vte_#$    >= a1.vts_#$
                    AND   e1.vte_#$     < a1.vte_#$)
  WHERE EXISTS (SELECT *
```

```
                FROM uAux a1
                WHERE e1.nome_#$    = a1.nome_#$
                AND   e1.salario_#$ = a1.salario_#$
                AND   e1.setor_#$   = a1.setor_#$
                AND   e1.vte_#$    >= a1.vts_#$
                AND   e1.vte_#$     < a1.vte_#$);
** UNTIL NO UPDATES **

DELETE FROM uAux e1
  WHERE EXISTS (SELECT *
                FROM uAux a1
                WHERE e1.nome_#$    = a1.nome_#$
                AND   e1.salario_#$ = a1.salario_#$
                AND   e1.setor_#$   = a1.setor_#$
                AND   a1.vts_#$     < e1.vts_#$
                AND   a1.vte_#$     = e1.vte_#$);

SELECT vts_#$ vts, vte_#$ vte,
       nome_#$    Nome,
       salario_#$ Salario,
       setor_#$   Setor
FROM uAux e1;
```

6.4 Tradução de consultas a eventos temporais

Neste trabalho foi proposta uma extensão da linguagem ATSQL2 para facilitar as consultas a eventos temporais (figura 6.1). A implementação do *software* tradutor deve verificar a presença do *timeflag EVENT* no início do comando para verificação da sintaxe e posterior tradução para SQL. As traduções que serão apresentadas a seguir, na forma de exemplos, não focalizam a eficiência e são de fácil entendimento.

No exemplo abaixo, deseja-se selecionar todos os eventos ocorridos do evento *Emprestar* da classe *Usuario.* O *OID* do usuário é o número *231.*

```
EVENT
  (Usuario, Emprestar, 231);
```

O comando acima possui a seguinte tradução em SQL:

```
SELECT e.TT
FROM  e_Eventos_#$ e
WHERE e.Classe = 'USUARIO'
AND   e.Evento = 'EMPRESTAR'
AND   e.OID = 231;
```

Agora, o mesmo comando cujo *OID* do usuário é desconhecido:

```
EVENT
  (Usuario, Emprestar, (SELECT OID
                        FROM USUARIO
                        WHERE Nome = 'JOAO DA SILVA'));
```

A tradução é a seguinte:

```
SELECT e.TT
FROM e_Eventos_#$ e
WHERE e.Classe = 'USUARIO'
AND   e.Evento = 'EMPRESTAR'
AND   e.OID IN (SELECT OID
                FROM USUARIO
                WHERE NOME = 'JOAO DA SILVA');
```

Uma consulta com a projeção de outros atributos além dos tempos de transação:

```
EVENT
  (Usuario, Emprestar, (SELECT OID
                        FROM USUARIO
                        WHERE CATEGORIA = 'OURO'))
   SELECT NOME
   FROM USUARIO;
```

A respectiva tradução:

```
SELECT e.TT, NOME
FROM e_Eventos_#$ e, Usuario
WHERE e.OID = Usuario.OID
AND   e.Classe = 'USUARIO'
AND   e.Evento = 'EMPRESTAR'
AND   e.OID IN (SELECT OID
                FROM USUARIO
                WHERE CATEGORIA = 'OURO');
```

Uma consulta com o último evento de cada usuário categoria ouro:

```
EVENT LAST
  (Usuario, Emprestar, (SELECT OID
                        FROM USUARIO
                        WHERE CATEGORIA = 'OURO'))
   SELECT NOME
   FROM USUARIO;
```

A tradução é a seguinte:

```
SELECT e.MAX(TT), Nome
FROM e_Eventos_#$ e, Usuario
WHERE e.OID = Usuario.OID
AND   e.Classe = 'USUARIO'
AND   e.Evento = 'EMPRESTAR'
AND   e.OID IN (SELECT OID
                FROM USUARIO
                WHERE CATEGORIA = 'OURO')
GROUP BY e.OID;
```

Uma consulta com especificação de intervalo:

```
EVENT [2001-01-01 - NOW)
  (Usuario, Emprestar, (SELECT OID
                        FROM USUARIO
```

```
                        WHERE CATEGORIA = 'OURO'))
    SELECT NOME
    FROM USUARIO;
```

A tradução da consulta com especificação de intervalo:

```
SELECT TT, Nome
FROM e_Eventos_#$ e, Usuario
WHERE e.OID = Usuario.OID
AND   e.Classe = 'USUARIO'
AND   e.Evento = 'EMPRESTAR'
AND   e.TT BETWEEN 20010101 and sysdate
AND   e.OID IN (SELECT OID
                FROM USUARIO
                WHERE CATEGORIA = 'OURO')
GROUP BY e.OID;
```

Nas traduções apresentadas acima é importante observar alguns padrões de tradução que sempre deverão ocorrer dependendo do tipo de consulta:

1. nas consultas com instrução *SELECT* a cláusula *FROM* deverá conter a tabela que aparece no nome da classe. Caso contrário, um erro de sintaxe deve ser retornado;

2. nas consultas cujo *OID* é desconhecido, a *subquery* deve ser traduzida sempre com o operador *IN* no lugar do operador *=;*

3. sempre que o comando apresentar a instrução *SELECT*, a tradução deve acrescentar o critério de junção da tabela e_Eventos_#$ com a tabela cujo nome aparece no nome da classe na forma *e.OID = ¡nome-tabela¿.OID;*

4. nas consultas que apresentem as palavras *FIRST* ou *LAST*, deve-se acrescentar a cláusula *GROUP BY e.OID* no final do comando traduzido;

5. as consultas que possuem especificação de intervalo devem acrescentar o operador lógico relacional *BETWEEN* na cláusula *WHERE*.

7 CONCLUSÃO

A extensão de OASIS com aspectos temporais e a inexistência de SGBDs que possuam um modelo de dados e uma linguagem de consulta temporais constituem o problema que foi discutido na introdução do presente trabalho. A solução encontrada foi a utilização de SGBDs atuais juntamente com um sistema de tradução de comandos temporais para comandos SQL, além de um modelo de dados para suporte.

Este trabalho apresentou a revisão dos conceitos gerais para a representação de aspectos temporais, bem como a linguagem OASIS e a sua extensão temporal. Um panorama da linguagem TSQL2 foi apresentado. A linguagem ATSQL2 foi estudada destacando-se o algoritmo de tradução para SQL.

No capítulo cinco foi proposto um modelo de dados para suporte à extensão temporal de OASIS. O modelo proposto possibilita a utilização do algoritmo de tradução de ATSQL2 para SQL sem necessidade de traduzir inclusive os comandos SQL cujas tabelas referenciadas sejam temporais. O conceito de generalização temporal, entre outras características, transforma a própria tabela não-temporal em temporal. Essa característica não preserva a tabela original (não-temporal). O modelo de dados proposto contorna o problema com a utilização de duas tabelas e uma visão que faz a junção de ambas. Tal como implementado no protótipo *TimeDB*, o qual implementa a linguagem ATSQL2 e os conceitos de generalização temporal, a visão mencionada é equivalente a apenas uma tabela temporal.

O capítulo seis apresentou uma extensão da linguagem ATSQL2 para tratamento de consultas a eventos temporais, discutiu algumas formas de implementação do *software* tradutor e apresentou uma maneira de efetuar o mapeamento de comandos de consulta ATSQL2 para SQL, bem como a tradução de comandos de consulta à eventos temporais. O capítulo é importante para futuros trabalhos que visam a implementação do *software* tradutor.

As principais contribuições do presente trabalho são a definição do modelo de dados adequado à extensão temporal de OASIS, a extensão da linguagem ATSQL2 para tratamento de consultas a eventos temporais de OASIS e o mecanismo de tradução de comandos ATSQL2 para SQL, também adequado ao modelo de dados proposto.

Quanto às limitações, pode-se dizer que o mecanismo de mapeamento de ATSQL2 para SQL é funcional mas sem preocupação com desempenho e não foi feita a implementação do *software* tradutor. A extensão da linguagem ATSQL2 para tratamento de eventos temporais de OASIS não permite *subqueries* escritas em ATSQL2 para recuperar um *OID* desconhecido. Igualmente, essa limitação ocorre também nas consultas a eventos temporais que possuam um comando *SELECT* para agregar

mais informação no resultado, ou seja, somente comandos SQL convencionais são permitidos nesse ponto do comando. O trabalho não apresenta uma linguagem de definição de dados para facilitar a criação ou modificação de tabelas temporais. Os mecanismos de atualização de dados também não são tratados.

Os trabalhos futuros que poderão ser realizados são a modificação do mecanismo de tradução de comandos ATSQL2 para SQL visando aumentar o desempenho, a implementação do *software* tradutor, a definição de uma linguagem de definição de dados com respectiva tradução para SQL e os mecanismos de atualização de dados.

REFERÊNCIAS

AL, C. S. J. et. A Consensus Glossary of Temporal Database Concepts. **SIGMOD Record (ACM Special Interest Group on Management of Data)**, [S.l.], v.23, n.1, p.52–64, Mar. 1994.

AN OO METHODOLOGICAL Approach for Making Automated Prototyping Feasible. In: INTERNATIONAL CONFERENCE ON DATABASE AND EXPERT SYSTEMS APPLICATIONS, DEXA, 7., 1996. **Database and Expert Systems Applications**. New York: Springer-Verlag, 1996. xv, 921p. (Lecture Notes in Computer Science, v.1134).

ARAPIS, C. **Specifying Object Interactions**. [S.l.]: Centre Universitaire d'Informatique, University of Geneva, 1991. Object Composition, Working paper.

BÖHLEN, M. H. **Managing Temporal Knowledge in Deductive Databases**. 1994. Tese (Doutorado em Ciência da Computação) — Departement Informatik, ETH Zürich, Zurich.

BÖHLEN, M. H.; JENSEN, C. S.; SNODGRASS, R. T. Evaluating the Completeness of TSQL2. In: INTERNATIONAL WORKSHOP ON TEMPORAL DATABASES, 1995. **Recent Advances In Temporal Databases**: procedings. Berlin: Springer-Verlag, 1995. p.153–174.

BÖHLEN, M. H.; MARTI, R. On the Completeness of Temporal Database Query Languages. In: INTERNATIONAL CONFERENCE ON TEMPORAL LOGIC, 1., 1994, Berlin. **Proceedings...** Berlin: Springer, 1994. p.283–300. (LNAI, v.827).

CLIFFORD, J.; CROKER, A. **The Historical Relational Data Model (HRDM) Revisited**. [S.l.]: Benjamin/Cummings, 1993. p.6–27.

EDELWEISS, N.; OLIVEIRA, J. P. M. **Modelagem de Aspectos Temporais de Sistemas de Informacao**. Recife, PE, Brazil: UFPE-DI, 1994.

GABBAY, D.; GUENTHER, F. (Ed.). **Deontic Logic**. Dordrecht: D. Reidel Publishing Co., 1984. p.605–714.

GABBAY, D. M.; MCBRIEN, P. Temporal Logic & Historical Databases. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES, 1991, Barcelona, Spain. **Proceedings...** San Mateo: Morgan Kaufmann, 1991. p.423–430.

KOWALSKI, R.; SERGOT, M. **A logic based calculus of events**. London: Dept. Computing, Imperial College, 1985.

LARMAN, C. **Applying UML and Patterns**: an introduction to object-oriented analysis and design. [S.l.]: Prentice Hall, 1997.

LETELIER, P. **OASIS version 3.0**: um enfoque formal para o modelo conceitual orientado a objetos. Valência: Universidad Politecnica de Valencia, Serviço de Publicaciones, 1998.

MAIOCCHI, R.; PERNICI, B. Temporal Data Management Systems: a comparative view. [S.l.], v.3, n.4, p.504–524, Dec. 1991.

MAIOCCHI, R.; PERNICI, B. Temporal Data Management Systems In Real-Time Systems: a comparative view. **IEEE Transactions on Knowledge and Data Engineering**, New York, v.3, n.5, Sept. 1991.

MELTON, J.; SIMON, A. R. **Understanding the new SQL**: a complete guide. Los Altos, CA: USA: Morgan Kaufmann, 1993. xxiv, 536p. (Morgan Kaufmann series in data management systems).

MENESES, C. A.; PASTOR, O.; INSFRÁN, E.; HEUSER, C. A. Entornos Automaticos de Produccion de Software a Partir de Modelos Conceptuales Orientados a Objetos y Temporales. **Proyecto IDEAS**, [S.l.], 1999.

MEYER, B. **Object-Oriented Software Construction**. 2nd ed. Englewood Cliffs: Prentice-Hall, 1997.

MEYER, J.-J. C. A Different Approach to Deontic Logic. Deontic Logic Viewed as a Variant of Dynamic Logic. **Notre Dame Journal of Formal Logic**, [S.l.], v.29, p.109–136, 1988.

NAVATHE, S. B.; AHMED, R. **Temporal extensions to the relational Model and SQL**. Redwood City, California: Benjamin/Cummings, 1993. p.92-109.

PASTOR, O. OO-Method: an object oriented methodology for software production. In: INTERNATIONAL CONFERENCE ON DATABASE AND EXPERT SYSTEMS APPLICATIONS, DEXA, 3., 1992. **Database and Expert Systems Applications**. New York: Springer-Verlag, 1992. p.121–127.

PASTOR, O. **Diseno y Desarrollo de un Entorno de Produccion Automatica de Software basado en el modelo OO**. 1992. Tese (Doutorado em Ciência da Computação) — Department de Sistemes Informàtics i Computació, Universitat Politècnica de Valencia, Valencia.

PASTOR, O.; HEUSER, C.; GOMEZ, J.; INSFRÁN, E. Ingenieria de Ambientes Software: combinando expresividades temporales y objetuales en la fase de modelizacion conceptual. In: JORNADAS DE TRABAJO EN INGENIERIA DEL SOFTWARE, 1997, San Sebastian, Spain. **Actas**. [S.l.: s.n.], 1997.

PASTOR, O.; HEUSER, C.; INSFRÁN, E.; EDELWEISS, N. Especificacion de Temporalidad en Modelos Conceptuales Orientados a Objetos. In: CONFERENCIA LATINOAMERICANA DE INFORMATICA, 24., 1998, Quito, Ecuador. **Memorias**. Quito: Pontificia Universidad Catolica del Ecuador, 1998. v.1, p.371-381.

PASTOR, O.; INSFRÁN, E.; PELECHAMO, V.; ROMERO, J.; MERSEGUER, J. OO-Method: an oo software production environment combining conventional and formal methods. In: INTERNATIONAL CONFERENCE ON ADVANCED INFORMATION SYSTEMS ENGINEERING, CAISE, 9., Barcelona. **Proceedings...** Berlin: Springer-Verlag, 1997. (Lecture Notes in Computer Science, v.1250).

PASTOR, O.; SALVERT, I. **OASIS version2 (2.2)**: a class-definition language to model information systems using an object-oriented approach. Valencia: Universidad Politecnica de Valencia, Serviço de Publicaciones, 1995.

SIMONETO, E. O. **Uma Proposta para a Incorporação de Aspectos Temporais no Projeto Lógico de Banco de Dados em SGBDs Relacionais**. 1998. Tese (Mestrado em Ciência da Computação) — Instituto de Informática, PUCRS, Porto Alegre.

SNODGRASS, R.; DYRESON, C. E.; JENSEN, C. S.; KLINE, N.; SO, L.; WHELAN, J. **MultiCal System, Release 1.0**. [S.l.: s.n.], 1993.

SNODGRASS, R. T.; BÖHLEN, M. H.; JENSEN, C. S.; STEINER, A. **Adding Transaction Time to SQL/Temporal Change Proposal**. [S.l.]: International Organization for Standardization, 1996. (JTC1/SC21/WG3 DBL MAD-146r2).

SNODGRASS, R. T.; BÖHLEN, M. H.; JENSEN, C. S.; STEINER, A. **Adding Valid Time to SQL/Temporal**. [S.l.]: International Organization for Standardization, 1996. (JTC1/SC21/WG3 DBL MAD-146r2 21/11/96, (change proposal)).

SNODGRASS, R. T. (Ed.). **The TSQL2 Temporal Query Language**. [S.l.]: Kluwer, 1995.

SOO, M.; SNODGRASS, R. **Mixed Calendar Query Language Support for Temporal Constants**. Tucson: Department of Computer Science, The University of Arizona, 1995.

STEINER, A. **The TimeDB Temporal Database Prototype**. 1995. Disponível em: <http://www.timeconsult.com>. Acesso em: 30 jan. 2002.

STEINER, A. **A Generalisation Approach to Temporal Data Models and their Implementation**. 1998. 160p. Tese (Doutorado em Ciência da Computação) — Swiss Federal Institute of Technology, Zurich.

TANSEL, A. U.; CLIFFORD, J.; GADIA, S.; JAJODIA, S.; SEGEV, A.; SNODGRASS, R. (Ed.). **HSQL**: a historical query language. Redwood City, Ca: Benjamin/Cummings, 1993.

TORRES, P. L. (Ed.). **OASIS version 3.0**: um enfoque formal para o modelo conceitual orientado a objetos. [S.l.]: Universidad Politecnica de Valencia, Serviço de Publicaciones, 1998.

ZANATTA, M. M. P. **Extensão de um Modelo OO Formal com Aspectos Temporais**. 2000. Tese (Mestrado em Ciência da Computação) — Instituto de Informática, UFRGS, Porto Alegre.