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

# *Definição de um Gerenciador para o Modelo de Dados Temporal TF-ORM*

por

PATRÍCIA NOGUEIRA HÜBLER

Dissertação submetida à avaliação,
como requisito parcial para a obtenção do grau de Mestre
em Ciências da Computação

Profª. Dra. Nina Edelweiss
Orientadora

Porto Alegre, agosto de 2000.

# CATÁLOGO NA PUBLICAÇÃO

Hübler, Patrícia Nogueira

Mapeamento de Regras de Transição de Estados do Modelo TF-ORM, através da Implementação de um Banco de Dados Temporal / Patrícia Nogueira Hübler - Porto Alegre: PPGC da UFRGS, 2000.

10608p.: il.

Dissertação (Mestrado) - Universidade Federal do Rio Grande do Sul, Programa de Pós-Graduação em Computação, Porto Alegre, BR-RS 2000. Orientador: Prof.a. Dr. Nina Edelweiss

1. Banco de Dados, 2. Banco de Dados Temporal, 3. Mapeamento, 4. Modelos de Dados Temporais, 5. TF-ORM,

# Agradecimentos

*À Profa. Dra. Nina Edelweiss, pela confiança, incentivo, força em todos os momentos. Por estar sempre disposta a esclarecer minhas dúvidas e ouvir minhas angústias. Pela sua doçura e atenção, muito obrigada!*

*Ao CNPq pela bolsa concedida no início do desenvolvimento deste trabalho.*

*Aos funcionários do Instituto de Informática que estão sempre dispostos a esclarecer nossas dúvidas.*

*À minha eterna orientadora: Tanisi Pereira de Carvalho. Uma pessoa maravilhosa, dinâmica. A pessoa que mais me incentivou no início deste trabalho. Obrigada minha Mestre!*

*Ao ex-Coordenador do Curso de Informática da Universidade Luterana do Brasil - Campus Gravataí, Vinicius Gadis Ribeiro, meu "padrinho" por todo carinho e confiança depositados. Vini, Muito Obrigada!*

*Agradeço ao atual coordenador do curso de Informática da Universidade Luterana do Brasil - Campus Gravataí, Marco Antônio da Rocha, por me disponibilizar tempo para o desenvolvimento deste trabalho.*

*Aos amigos do CTG Laço da Amizade por compreenderem minhas faltas aos ensaios e minha angústia no decorrer dos anos. Sempre me acolheram e me apoiaram. Vocês são mais que especiais para mim. Vocês fazem parte da minha história!*

*Aos meus irmãos por suportarem momentos de ira e falta de humor.*

*Agradeço a todos os meus amigos por ouvirem minhas lamúrias e resmungos. Gente, não sei como vocês me agüentaram!*

*Agradeço a Deus pelas pessoas maravilhosas que estão ao meu redor, pelo carinho e carisma de cada uma.*

# Oferecimento

*Muitas vezes as palavras são traiçoeiras e dizemos coisas que não gostaríamos de dizer. As pessoas que mais sofreram com isto foram meus pais, Maria Zilá e Victor Hugo. Nem por isso deixaram de me incentivar e valorizar o meu trabalho. Foram carinhosos, dedicados...*

*Tantos fins-de-semana sem acompanhar a família, muitos dos quais vocês se privavam para me incentivar e me prestar companhia.*

*As palavras de amor não saem facilmente, mas tenham certeza, amo muito vocês!*

*Poucas pessoas possuem a compreensão, o apoio, o incentivo que vocês sempre me deram. Espero, um dia, poder passar tudo o que aprendi com vocês para meus filhos e saber que vocês sentem-se orgulhosos com isto.*

***Amo muito vocês!!!***

# Sumário

# Lista de Abreviaturas

| | |
|---|---|
| BD | Banco de Dados |
| BDT | Banco de Dados Temporal |
| ER | Entidade Relacionamento |
| ERT | Entidade Relacionamento Temporal |
| PL/SQL | Procedural Language/Structured Query Language |
| SGBD | Sistema Gerenciador de Banco de Dados |
| SGBDH | Sistema Gerenciador de banco de Dados Histórico |
| SQL | Structured Query Language |
| TV | Tempo de Validade |
| TT | Tempo de Transação |

# Lista de Figuras

# Lista de Tabelas

# Resumo

Há alguns anos, a necessidade de armazenar a história das informações e o período no qual as mesmas são válidas ou não no mundo real, está crescendo. As próprias leis vigentes no país fazem com que isto seja uma necessidade quando, por exemplo, criam uma tabela de tempo de validade para o armazenamento de documentos fiscais ou de recursos humanos.

Neste âmbito destaca-se a importância do estudo de modelos de dados temporais que gerenciem este tipo de informação. Tais modelos definem a forma como as informações são organizadas, mantidas e recuperadas. A implementação destes modelos, entretanto, é realizada sobre bancos de dados (BD) convencionais, uma vez que ainda não existe um BD totalmente temporal. O mapeamento de um modelo temporal sobre um convencional não impede que sejam satisfeitas todas as necessidades de representação temporal, desde que seja realizado de forma coerente.

As diferentes informações temporais que podem ser utilizadas para a representação de tempo são o tempo de transação e o tempo de validade. Quando se deseja uma representação completa da realidade, utilizam-se bancos de dados bitemporais, através dos quais é possível recuperar todas as informações passadas, presentes e futuras.

Este trabalho apresenta o mapeamento de um modelo de dados bitemporal (modelo TF-ORM) para um SGBD convencional (Oracle). Este mapeamento compreende, além das informações temporais, todas as características do modelo, dentre as quais: classes e papéis, mensagens, regras, propriedades dinâmicas e estáticas. É apresentada, ainda, a definição de um gerenciador temporal, o qual busca automatizar o que é implementado.

Complementando o estudo realizado, são apresentadas sugestões de operações a serem realizadas pelos desenvolvedores de aplicações temporais quando da manutenção das informações. As operações de inserção, atualização e remoção em uma base temporal são analisadas, independente do tipo de BD temporal implementado. Um estudo de caso é apresentado para validar todas as definições realizadas.

**Palavras-chave:** Banco de Dados, Banco de Dados Temporal, Mapeamento, Modelos de Dados Temporais, TF-ORM

**Title: *"Definition of a Manager for a TF-ORM Temporal Data Model"***

# Abstract

The need for storing the information's history and the period in which they are still valid in the real world has been growing in the last few years. The very existing laws in the country make it necessary when, for instance, a validity temporal table is created for storing business or human resources documents.

Therefore, the study of temporal data models which manage this kind of information has become very important. Such models define the way in which the information is organized, kept and recovered. The implementation of these models, though, is carried out over conventional data bases (DB), once there is no a totally temporal DB. The mapping of a temporal model over a conventional one allows all of its requirements to be met, as long as it is made coherently.

The different temporal information that can be used for time representation are the transaction time and the validity time. Whenever a complete representation of the reality is wanted, bitemporal databases are used. These bases make it possible to recover all the past information as well as present and future information.

This work presents the mapping of a temporal data model (TF-ORM model) for a conventional DBMS (Oracle). Besides the temporal information, this mapping comprehends all the model's characteristics, such as: classes and roles, messages, rules, dynamic and static properties. It is also presented a definition of a temporal manager which aims at automating what is implemented.

In addition to this study, suggestions of operations to be carried out by the developers of temporal applications for the maintenance of the information are presented. The operations of inserting, updating and removing on temporal base are analyzed, apart from the kind of DB implemented. A case study is presented in order to validate all the definitions made.

**Keywords:** Database, Temporal Database, Mapping, Temporal Data Model, TF-ORM

# 1 Introdução

Bancos de dados temporais (BDTs) permitem armazenar e recuperar todos os estados de um objeto, registrando sua evolução ao longo do tempo [EDE 94,98, ETZ 98, ÖZS 95, TAN 93, ZAN 97]. Para isso, informações temporais devem ser associadas aos dados. Estas informações correspondem ao tempo de validade (tempo no qual a informação será válida no banco de dados) e/ou tempo de transação (tempo no qual a informação foi inserida no banco de dados). Dependendo do tipo de informação temporal considerada, os bancos de dados podem ser classificados em [SNO 85, EDE 94, JEN 98]:

- *banco de dados instantâneo* - cada modificação feita no banco de dados faz com que o valor anteriormente armazenado seja perdido;
- *banco de dados de tempo de transação* - cada informação armazenada possui um rótulo temporal (*timestamp*) associado a ela, representando o tempo de sua inserção no BD;
- *banco de dados de tempo de validade* - utiliza o tempo de validade como rótulo temporal, para representar o período no qual a informação é válida;
- *banco de dados bitemporal* - associa a cada informação tanto o tempo de validade quanto o tempo de transação;
- *banco de dados multitemporal* - pode associar às informações tempo de transação, tempo de validade ou ambos.

Diferentes modelos de dados possuem características temporais [CLI 87, SAR 90, SNO 87, ELM 93, ANT 97, WUU 93, LIN 93]. Alguns associam aos dados apenas informações referentes ao tempo de transação, outros, ao tempo de validade e, outros, ainda, tanto ao tempo de transação quanto ao tempo de validade. Estes modelos de dados são, geralmente, extensões de modelos convencionais, sendo a variação do tempo representada por valores de rótulos temporais, os quais podem ser adicionados às tuplas ou aos atributos. Estes podem ser:

- *pontos no tempo*: um único valor, o qual, representa o início da validade da informação;
- *intervalos temporais*: dois valores, identificando um intervalo, representando o tempo inicial e final da validade da informação;
- *elementos temporais*: conjuntos disjuntos de intervalos temporais.

A utilização de um modelo de dados temporal para a especificação de uma aplicação não implica, necessariamente, na utilização de um SGBD específico para o modelo. Bancos de dados (BD) comerciais podem ser utilizados se existir um mapeamento adequado entre o modelo temporal e o banco de dados utilizado. Este enfoque, está sendo adotado para o modelo de dados temporal orientado a objetos TF-ORM [EDE 93, 94a].

A implementação requer uma estratégia específica para sua representação, de modo a tornar o gerenciamento dos dados históricos independente da intervenção do usuário ou programador de aplicações.

Algumas implementações de modelos de dados temporais já foram realizadas sobre bancos de dados convencionais [CAV 95, SIM 98, TIM 99, BRA 94, ARR 94, ORE 00]. Entretanto, estas visam apenas a representação dos dados temporais nas bases convencionais, não se preocupando com operações de manutenção das informações históricas.

As operações realizadas para manter a base de dados – inserção, atualização e exclusão – utilizadas em bases de dados convencionais são, da mesma forma, utilizadas em bases de dados temporais. Entretanto, para a manutenção correta do tempo, algumas regras adicionais devem ser criadas e sua implementação deve ser adaptada. Estas operações são executadas de diferentes formas, dependendo do tipo de BDT implementado. Desta forma, existe a necessidade de, antes de realizar o mapeamento, determinar qual o tipo de BDT a ser implementado.

## 1.1 Apresentação do Trabalho

Este trabalho tem por objetivo analisar a forma de implementar BDTs em BDs convencionais, partindo de um modelo de dados temporal. São apresentadas diferentes possibilidades para a realização de operações na base de dados, bem como a definição da implementação de mensagens e regras quando se trata de bancos de dados temporais.

Um BDT que manipula regras e mensagens foi implementado para validar o mapeamento realizado. É utilizado o modelo TF-ORM neste estudo, o qual possui, como características básicas, regras de integridade e transição de estados, além de possibilitar a recuperação de um maior número possível de histórias (informações), por ser um modelo bitemporal. Assim, através do modelo escolhido, obtém-se a implementação de um BDT bitemporal. O BD comercial escolhido foi o Oracle, um banco de dados muito utilizado por empresas de grande porte, as quais necessitam manter muitas informações históricas.

O presente trabalho está organizado da seguinte forma:

- no capítulo 2 encontram-se exemplos de modelos de dados temporais: relacionais, orientados a objetos e entidade-relacionamento. São apresentadas suas principais características;

- exemplos de implementações de modelos temporais em SGBDs convencionais são apresentados no capítulo 3;

- a definição da implementação de operações a serem executadas sobre os diferentes tipos de BDTs é apresentada no capítulo 4, bem como sugestões de atuação em determinados casos;

- o modelo TF-ORM, modelo utilizado na implementação, é descrito no capítulo 5. São apresentadas suas características, operadores e definições;

- para validar o mapeamento e a implementação realizada, definiu-se um estudo de caso, apresentado no capítulo 6;

- o capítulo 7 apresenta o mapeamento do modelo TF-ORM para um SGBD convencional como o Oracle;

- no capítulo 8, encontram-se exemplos da implementação realizada;

- conclusões e propostas de trabalhos futuros são apresentadas no capítulo 9.

# 2 Bancos de Dados Temporais

Muitas aplicações requerem informações não apenas do presente, mas também do passado e do futuro. Os atributos de um objeto podem assumir diferentes valores ao longo do tempo e o conjunto destes valores constitui a história deste objeto. Uma base de dados temporal é definida como uma base de dados que mantém as histórias dos objetos, isto é, passado, presente e, possivelmente, futuro das informações.

Neste capítulo serão apresentados alguns modelos de dados temporais, suas principais características e conceitos. Estes modelos são, na maioria das vezes baseados em modelos já existentes, tais como relacionais, orientados a objetos ou, do tipo entidade-relacionamento.

Os diferentes tipos de bancos de dados existentes também serão apresentados, enfatizando suas diferenças e formas de recuperar as informações.

## 2.1 Modelos de Dados

O tempo pode ser definido de duas formas: tempo de validade (TV) e tempo de transação (TT). O TV corresponde ao tempo no qual um fato é válido na realidade. Os TVs podem corresponder ao presente, passado e futuro. O TT corresponde ao momento no qual a informação é inserida no BD. As duas dimensões de tempo são ortogonais, pois possuem semânticas diferentes.

A variação do tempo em modelos temporais é, geralmente, representada por rótulos temporais associados aos dados. Estes podem ser: pontos no tempo, intervalos ou elementos temporais (conjuntos disjuntos de intervalos temporais). Os rótulos temporais podem ser adicionados às tuplas ou aos atributos.

Diversos modelos de dados temporais foram propostos, sendo a maioria extensões de modelos já existentes, adicionando a estes, características relacionadas à manipulação do tempo. De acordo com a forma utilizada para representar o tempo, os modelos temporais podem ser de tempo de transação (quando utilizado o TT), de tempo de validade (quando utilizado o TV) ou bitemporais (quando utilizados tanto TT, quanto TV).

Dentre as várias características encontradas nos modelos de dados, a possibilidade de recuperar informações armazenadas em um banco de dados é fundamental. O tempo pode ser envolvido em consultas temporais de três diferentes formas [EDE 94a]:

1. recuperar valores de propriedades cujo domínio é temporal;
2. referir-se a um determinado instante ou intervalo temporal;
3. recuperar valores com base em restrições temporais.

O poder de expressão da linguagem de consulta de um modelo temporal é importante. Muitas linguagens de consulta dos modelos de dados temporais são baseadas no SQL. Alguns critérios foram adotados nos modelos relacionais temporais para completar e melhorar a representação e manipulação dos dados temporais.

### 2.1.1 Modelos de Dados Temporais Relacionais

A representação do tempo em modelos relacionais temporais é feita, geralmente, através de atributos adicionais, denominados de rótulos temporais (TT e/ou TV). Existem diversos modelos temporais relacionais, sendo os primeiros a surgiu na literatura.

Dentre todos os modelos existentes, pode-se citar o HRDM (*The Historical Relational Data Model* ), por ter sido o primeiro modelo de dados temporal apresentado, surgindo em 1987 [CLI 87]. É um modelo de dados relacional

O modelo HRDM, introduziu o conceito de *lifespans*. Permite associação de tempo a tuplas, atributos ou relações. Cada tupla é associada a um *lifespan*, podendo recuperar valores apenas em pontos no tempo deste *lifespan*. A cada atributo de um esquema relacional, incluindo os atributos chave, é designado um domínio de valores, o qual é formado por um conjunto de valores atômicos e um *lifespan*. Através destes *lifespans*, é possível representar todos os estados de um objeto: passado, presente e futuro [CLI 93, CLI 93a]. Este modelo, utiliza o tempo de validade para representar as informações temporais.

Outra extensão temporal de um modelo relacional é o HSQL [SAR 90]. O HSQL associa o tempo apenas a atributos. Ele adota uma visão orientada a estados da base de dados histórica correspondente, onde os estados dos objetos são definidos pelos valores dos seus atributos. Os estados prevalecem sobre um intervalo de tempo, no qual nenhum valor de atributo é alterado. Adota-se, assim, a utilização de instâncias com intervalos temporais.

Mantendo a estrutura básica de um comando SQL, o HSQL apresenta facilidades para definição, armazenamento, recuperação e atualizações das relações históricas. A linguagem suporta comparações entre instantes de tempo e operações sobre intervalos. As tuplas destas relações contém rótulos temporais [SAR 93]. Este modelo, como o HRDM, também utiliza o tempo de validade para representar as informações temporais.

A linguagem de consulta temporal TQuel, apresentada por [SNO 87], é uma linguagem para recuperação de informações temporais. O modelo correspondente, suporta tanto tempo de validade, quanto tempo de transação. Como nas linguagens de consulta convencionais, a linguagem TQuel também suporta tempo definido pelo usuário (consistindo de propriedades temporais definidas explicitamente pelos usuários em um domínio temporal e manipuladas pelos programas de aplicação).

O TQuel associa o tempo a atributos adicionais [SNO 93], sendo que toda a relação temporal é representada através de uma relação instantânea que apresenta um atributo temporal. As tuplas recebem, opcionalmente, um rótulo temporal. O modelo possui ainda relações do tipo evento, onde as tuplas recebem um único rótulo temporal

(representando ocorrências instantâneas) e do tipo intervalo, onde as tuplas recebem um par de rótulos temporais, correspondendo ao TV e um par de rótulos temporais, correspondendo ao TT (representando um estado válido durante um intervalo temporal).

Em [SNO 95] é apresentada a linguagem de consulta TSQL2, definida com o objetivo de ser a linguagem de consulta temporal de consenso. Suporta três dimensões temporais: tempo definido pelo usuário, tempo de validade e tempo de transação. Neste modelo estão presentes os conceitos de instante e período de tempo.

### 2.1.2 Modelos de Dados Temporais Entidade-Relacionamento (E-R)

Os modelos de dados temporais Entidade-Relacionamento utilizam entidades, relacionamentos e atributos do modelo padrão (E-R), associando o tempo a estas formas de representação. Dentre eles, pode-se citar o TEER, o TempER e o TER. Alguns destes modelos serão apresentados neste item.

O modelo TEER (*Temporal EER*), [ELM 93], é uma adaptação do modelo EER (*Enhanced Entity-Relationship)* [ELM 89] incluindo a dimensão temporal. Tal modelo, utiliza intervalos como unidade temporal básica. O modelo TEER é estendido ao incluir informações temporais para entidades, relacionamentos, superclasses, subclasses e atributos. Cada entidade deste modelo é associada com um elemento temporal, representando a duração de vida (*lifespan*) da entidade. Cada entidade tem um conjunto de atributos básicos e a cada atributo é associado um domínio de valores. O *lifespan* pode ser um intervalo de tempo contínuo ou a união de intervalos de tempos disjuntos.

Outra extensão dos modelos temporais do tipo entidade-relacionamento é o TempER (*Temporal Entity Relationship*) [ANT 97], o qual permite representar a associação entre elementos temporalizados e não temporalizados. Para tanto, adota o pressuposto que todas as entidades, sejam elas temporalizadas ou não, apresentam uma dimensão temporal, ou seja, uma "existência" ou validade temporal [ANT 98]. No caso das entidades temporalizadas esta existência é um subconjunto de pontos do eixo temporal. Em virtude disto são chamadas de *entidades transitórias*. Em relação às entidades não temporalizadas, é assumido que "existem" durante todo o eixo temporal, ou seja, a sua validade temporal é constante, implícita e igual a todo o eixo temporal. Por isto são denominadas entidades perenes.

### 2.1.3 Modelos de Dados Temporais Orientados a Objetos

Os modelos de dados temporais orientados a objetos utilizam as primitivas básicas da orientação a objetos para manipular e armazenar informações. Trabalham com classes, atributos e métodos.

O modelo de dados OODAPLEX, conforme [WUU 93], apresenta as principais características de um modelo orientado a objetos: identidade de objetos, encapsulamento através do conceito de tipos abstratos de dados, tipos de objetos complexos, herança, polimorfismo.

Todo objeto apresenta uma identidade única, imutável durante toda a existência do objeto e independente das propriedades e do comportamento deste objeto. Todos os objetos são abstratos, consistindo de uma interface (o tipo do objeto) e de uma implementação (a representação do objeto). As propriedades dos objetos, os relacionamentos entre objetos e as operações sobre os objetos são todos modelados de uma maneira uniforme através de funções aplicadas aos objetos.

No modelo, objetos que apresentam propriedades e comportamento similares são agrupados em tipos. O tipo especifica o conjunto de funções que podem ser aplicados às instâncias do tipo. OODAPLEX suporta o encapsulamento, uma vez que os objetos somente podem ser manipulados através das funções definidas para seus tipos [WUU 93]. A associação de tempo no modelo é feita tanto nos atributos como nos objetos como um todo. A associação de tempo aos atributos permite que as propriedades de um objeto variem assincronamente com o tempo. Possibilita, também, o versionamento de objetos.

Este modelo apresenta uma linguagem de consulta poderosa que combina, de uma maneira uniforme, a recuperação e a atualização de objetos individuais e de agregados de objetos, a recuperação e a navegação associativa, a manipulação de objetos existentes e a criação de novos objetos. Pode ser utilizada para resolver consultas temporais e não temporais.

Outra extensão dos modelos de dados temporais orientados a objetos é o modelo TOODM [LIN 93] que manipula TT e TV. O modelo trabalha sobre três tipos de objetos: (i) objetos do tipo TIME - objetos utilizados para representar as dimensões temporais; (ii) objetos invariantes no tempo - que não são compostos por nenhum objeto do tipo TIME e (iii) objetos variantes no tempo - objetos complexos compostos por, pelo menos, um objeto do tipo TIME.

A linguagem de consulta do TOODM [LIN 93, BRA 94] permite consultas que retornam: (i) valores de tempo (objetos do tipo TIME); (ii) valores invariantes no tempo e (iii) objetos, classes ou estados temporais.

O TF-ORM (*Temporal Functionality in Objects With Roles Model*) [EDE 93, 94a] é um modelo de dados orientado a objetos que utiliza o conceito de papéis para representar os diferentes comportamentos de um objeto. As informações temporais estão associadas aos dois valores de tempo: TT e TV, o que torna o TF-ORM um modelo de dados bitemporal.

O modelo de dados TF-ORM, com sua linguagem de consulta também baseada na linguagem SQL, apresenta alguma influência do TQuel.

## 2.2 Bancos de Dados

Dependendo da possibilidade de representação de informações históricas, os bancos de dados foram classificados por [SNO 85] em:

- **Bancos de Dados Instantâneos**: são os bancos de dados tradicionais, onde apenas a última definição dos valores é armazenada. Toda vez que um valor é alterado, ou

seja, um novo valor é definido, o anterior é perdido. Não apresentam suporte para informações temporais, podendo apenas ser tratados através de tempos definidos pelos usuários e através de aplicações externas, (não permitem consultas temporais);

- **Banco de Dados de Tempo de Transação** ("*rollback databases*"): todos os valores definidos neste tipo de banco de dados ficam armazenados permanentemente, sendo válidos a partir do momento da sua definição. O tempo em que foi feita a definição do valor no banco de dados, ou seja, o seu tempo de transação, é associado a cada valor armazenado no banco de dados. A recuperação dos valores atuais das informações armazenadas e das informações definidas em algum instante no passado é feita de acordo com o instante de sua definição (tempo de transação). Não é possível fazer alterações de dados passados ou futuros. São bancos de dados orientados à implementação, uma vez que o tempo de transação é fornecido pelo sistema gerenciador do banco de dados [EDE 94];
- **Banco de Dados de Tempo de Validade** ("*Historical Databases*"): representam melhor a semântica da realidade do que os bancos de dados de tempo de transação. Modelam a realidade associando a cada informação o seu tempo de validade na aplicação e não no instante de sua definição. Considerando o tempo de validade associado às informações, podem ser feitas alterações de informações válidas em momentos presentes, passados ou futuros;
- **Bancos de Dados Bitemporais** ("*Temporal Databases*"): combinam bancos de dados de tempo de validade com os de tempo de transação. Armazenam não somente o conhecimento atual dos dados, mas também todos os estados passados do banco de dados. Possibilitam consultas a respeito de valores atuais, passados e futuros, tanto para o atual estado de validade das informações como para estados passados do banco de dados;
- **Bancos de Dados Multitemporais**: utilizam simultaneamente características de bancos de dados de tempo de transação, de tempo de validade e bitemporais. Possibilitam consultas ao passado, presente e futuro das informações.

## 2.3 Considerações Finais

Neste capítulo foram apresentados conceitos relativos à representação temporal, assim como alguns modelos de dados temporais existentes na literatura, os quais surgiram da necessidade de representar o tempo nos bancos de dados convencionais.

De acordo com o rótulo temporal utilizado (TT/TV), diferentes tipos de bancos de dados foram identificados, sendo apresentadas suas características e formas de recuperação das informações relacionadas ao tempo. Pode-se concluir que os modelos de dados bitemporais, como por exemplo TQuel, TSQL2, TOODM e TF-ORM, em relação aos demais modelos estudados, apresentam uma representação da realidade mais completa, pois tratam informações tanto do passado, quanto do presente e do futuro.

Neste trabalho será utilizado como modelo para a implementação do banco de dados temporal no SGBD objeto-relacional ORACLE, o modelo TF-ORM, por ser um

modelo temporal orientado a objetos, bitemporal. Um estudo mais aprofundado sobre este modelo será apresentado nos próximos capítulos.

Quanto à forma de representação do tempo, as representações através de pontos no tempo ou de intervalos temporais, são equivalentes. Serão implementados intervalos temporais, associando, às informações, seus tempos iniciais e finais, considerando o tempo variando de forma contínua.

# 3 Implementação de Bancos de Dados Temporais em SGBDs Convencionais

Atualmente, muitos pesquisadores têm se dedicado a estudar a temporalidade das informações, pois a necessidade de implementação de bancos de dados temporais pode ser identificada em diferentes áreas: sistemas geográficos, sistemas de manufatura e muitos outros. Estes sistemas, em geral, necessitam armazenar informações em diferentes estados, fazer previsões, análises e consultas a dados passados e/ou futuros.

A implementação de bancos de dados temporais em SGBDs convencionais deve-se à inexistência de um SGBD totalmente temporal. Com a criação de mapeamentos apropriados pode-se utilizar SGBDs convencionais para a representação de modelos temporais, sem que se necessite de SGBDs específicos para tal tarefa.

A implementação requer uma estratégia específica para sua representação, de modo a tornar o gerenciamento dos dados históricos independente da intervenção do usuário ou programador de aplicações.

Nos próximos itens serão apresentados exemplos de implementações de bancos de dados temporais em SGBDs relacionais e orientados a objetos.

## 3.1 Níveis de Implementação

Levando-se em consideração uma implementação a ser desenvolvida, os recursos necessários disponíveis no SGBD e o modelo de dados temporal utilizado, diferentes níveis de implementação foram identificados [CAV 95]:

- **Nível 1**: o SGBD hospedeiro não suporta regras. Tais regras referem-se a qualquer controle de integridade das informações. Considera-se, desta forma, o modelo sem qualquer restrição. Estas devem ser implementadas através de programas de aplicação;
- **Nível 2**: o SGBD hospedeiro suporta regras. Entretanto, qualquer outra característica própria do modelo como mensagens, por exemplo se for um modelo orientado a objetos, devem ser implementadas através de programas de aplicação;
- **Nível 3**: o SGBD suporta todas as características do modelo. Desta forma, considera-se o modelo em sua totalidade.

## 3.2 Implementação de Bancos de Dados Temporais em SGBDs Relacionais

Os bancos de dados relacionais são os mais utilizados para a implementação de bancos de dados temporais por serem, também, os mais utilizados comercialmente. A

implementação neste tipo de banco de dados é feita através da inclusão de atributos especiais para representar o tempo (TT, TV ou Bitemporal).

### 3.2.1 A Implementação de Simonetto

[SIM 98] propõe uma maneira de incorporar aspectos temporais no projeto lógico de bancos de dados, considerando as atuais funcionalidades oferecidas pelos SGBDs relacionais dominantes do mercado, sem que o tratamento temporal dos dados interferisse no trabalho dos que não precisam (ou não devem) tomar conhecimento desses aspectos temporais. Para tanto, a solução automatizou a manipulação da parcela temporal do banco de dados, sendo utilizado somente o TT.

Simonetto apresenta a necessidade do modelo ser transparente ao usuário. Devido a isto, implementou um banco de dados temporal de tempo de transação.

A implementação foi realizada sobre os SGBDs Ingres e Sybase, buscando satisfazer os requisitos do modelo proposto.

A representação do tempo no modelo deu-se na forma de intervalos no tempo, facilitando a recuperação de informações temporais. Para representar o tempo de transação foram acrescentados dois atributos temporais adicionais: início de tempo de transação (I_TT) e final do tempo de transação (F_TT) às informações que variam com o tempo.

A granularidade de tempo utilizada no modelo foi a de segundos.

### 3.2.2 O Projeto ORES

O projeto ORES [ORE 00] buscou criar um SGBDT que suportasse a representação amigável e eficiente da manipulação do conhecimento temporal. O ORES foi desenvolvido sobre o banco de dados comercial INGRES.

Este projeto:

- desenvolveu uma fundamentação formal da representação temporal;
- desenvolveu uma linguagem de consulta temporal, a qual é consistida pelo SQL;
- desenvolveu técnicas e ferramentas amigáveis para a manipulação, definição e validação de dados temporais;
- realizou um estudo de caso para validar suas observações.

A implementação foi realizada considerando, apenas, o tempo de validade das informações, e possibilita, também, consultas pela Internet.

### 3.2.3 A Implementação de Cavalcanti - Ingres

[CAV 95] propõe uma abordagem para a implementação do modelo TF-ORM utilizando SGBDs relacionais. Tal abordagem foi validada sobre o SGBD relacional Ingres.

Para a implementação, [CAV 95] optou pela representação através de atributos adicionais, cujos domínios são instantes de tempo, devido à simplicidade e maior facilidade de manipulação desse tipo de atributo em SGBDs relacionais.

Nas operações de inserção, o TT é atualizado automaticamente pelo sistema. O TV deve ser solicitado ao usuário caso seu valor não seja definido em uma operação de inserção e/ou atualização. Quando uma operação de inserção se referir a um novo objeto ou novo papel de objeto já existente, os valores das propriedades *object_instance* ou *role_instance* (do modelo TF-ORM) devem ser devidamente inseridos.

As operações de atualização devem ser tratadas de forma mais específica, pois em um banco de dados temporal nenhum dado que se torne antigo pode ser perdido. Para tanto, todas as operações de atualização devem ser seguidas de operações de inserção. Tal operação deve garantir que valores novos sejam inseridos e valores antigos sejam mantidos.

As operações de remoção devem ser redirecionadas para não remover de fato os dados, mas inserir o valor de *end_object* ou *end_role* e atualizar o fim do tempo de validade das propriedades. Isto faz com que as operações deixem de valer no instante da remoção.

Na implementação utilizando o Ingres [CAV 95], classes e papéis foram mapeados para tabelas do banco de dados relacional.

Quando se utiliza SGBDs relacionais para implementar modelos com maior capacidade de representação, como os modelos orientados a objetos por exemplo, podem surgir problemas de desempenho devido à necessidade de mapeamentos específicos. Conforme [CAV 95], quando utiliza-se a representação temporal, esta situação tende a ser agravada.

### 3.2.4 O Banco de Dados Temporal Comercial - TimeDB

O BDT TimeDB [TIM 99] é um SGBD relacional baseado na linguagem de consulta SQL. Foi desenvolvido utilizando SICSTus Prolog.

O TimeDB suporta a linguagem de consulta temporal ATSQL2, a qual foi desenvolvida para o próprio BDT. É executado como uma camada do SGBD relacional comercial Oracle. As declarações ATSQL2 são compiladas em seqüência. Isto garante a portabilidade entre diferentes plataformas e diferentes SGBDS. Providencia, também, declarações bitemporais.

## 3.3 Implementação de Bancos de Dados Temporais em SGBDs OO

A evolução das aplicações faz com que surja a necessidade de gerar condições para que a realidade seja melhor representada. Os modelos orientados a objetos permitem uma representação mais precisa, atendendo às requisições dessas novas aplicações. A implementação neste tipo de banco de dados é feita através da inclusão de

atributos especiais para representar o tempo (TT, TV ou Bitemporal) nas classes do banco de dados.

### 3.3.1 A Implementação do Modelo TOODM

Em [BRA 94] é apresentada uma implementação do modelo TOODM sobre o banco de dados $O_2$. Tal implementação visou aliar as funções básicas de gerenciamento temporal às funções de evolução de esquemas e de gerenciamento de objetos apresentadas pelo $O_2$.

Durante o processamento de mapeamento dos comandos do usuário em comandos $O_2$, a camada de gerenciamento temporal precisa garantir [BRA 94]:

- a manutenção da semântica das especificações do usuário (garantir a correta execução das solicitações do usuário);
- as restrições de integridade temporal do modelo TOODM;
- a geração de comandos $O_2$ sintaticamente corretos.

A implementação assume que pode-se associar, aos objetos, as duas dimensões temporais: TT e/ou TV. Fornecendo a identificação e resolução de problemas inerentes à incorporação de tempo em um BDOO.

### 3.3.2 O Projeto TOOBIS

O TOOBIS - *Temporal Object-Oriented Databases in Information System* [TOO 99] é um projeto de criação de um SGBDOO competitivo, o qual suporta a representação e recuperação de dados temporais. Os objetivos do TOOBIS são:

- estender as funcionalidades oferecidas pelo SGBDOO $O_2$, suportando TT e TV;
- providenciar uma análise temporal e uma definição da metodologia, pela adaptação de metodologias existentes correspondendo aos aspectos temporais;
- generalizar as aplicações desenvolvidas para ele e buscar novas possibilidades nos setores da indústria.

O TOOBIS foi criado sobre o SGBDOO $O_2$ por ser este banco de dados o líder entre os bancos de dados orientados a objetos na Europa e por estarem os conceitos de orientação a objetos se difundindo no mundo [TOO 99].

### 3.3.3 A Implementação de Arruda

Arruda em [ARR 94] apresenta a implementação do modelo TF-ORM sobre o SGBDOO $O_2$. Procura definir uma forma genérica para a implementação, estudando as principais características do SGBDOO escolhido.

Desta forma, implementou o modelo utilizando relacionamento das classes e papéis via atributos, não pela herança (característica de OO), por não suportar a

instanciação de um mesmo objeto em papéis distintos ou múltiplas instâncias de um mesmo papel.

As propriedades definidas no modelo TF-ORM foram implementadas como atributos no SGBDOO. Os estados do modelo TF-ORM foram implementados como atributos dinâmicos.

### 3.3.4 A Implementação de Cavalcanti - Postgres

[CAV 95] implementou o modelo TF-ORM também no Postgres, o qual apresentou maiores facilidades de implementação, especialmente, nos mecanismos de *passagem de tempo* e de suporte às consultas temporais.

As classes do TF-ORM foram implementadas como as classes do Postgres, e as propriedades estáticas e os estados como atributos de classes. Esta solução, entretanto, necessita de um identificador adicional para classes e papéis, para relacioná-los com suas propriedades dinâmicas. As mensagens, embora não sejam suportadas diretamente, podem ser implementadas como funções escritas na linguagem de programação hospedeira e associadas a cada classe específica.

## 3.4 Considerações Finais

Este capítulo apresentou algumas implementações de bancos de dados temporais existentes. Tais implementações são realizadas sobre BDs convencionais, sendo apresentadas implementações sobre BDs relacionais e orientados a objetos. Até mesmo o BDT comercial TimeDB é implementado sobre um BD convencional: Oracle.

Os diferentes tipos de mapeamentos estudados referem-se à incorporação de tempo aos bancos de dados comerciais, sendo que, em alguns casos, o tempo foi tratado como sendo, apenas, de TT e, em outros casos, de TV. Alguns dos mapeamentos apresentados utilizam ambos TT e TV, implementando bancos de dados *bitemporais*.

Como apresentado no capítulo 2, um modelo ou BD bitemporal representa melhor a realidade. Sendo assim, o mapeamento desenvolvido neste trabalho buscará representar a realidade de forma completa, ou seja, desenvolver um mapeamento que não seja apenas utilizado por um banco de dados temporal específico, mas, que possa ser utilizado tanto por um *bitemporal*, quanto por um *multitemporal*.

As implementações apresentadas não citam a criação de regras para a manutenção das informações. Este trabalho, desenvolvido através das definições do modelo TF-ORM, definirá um mapeamento para as regras de transição de estados deste modelo, bem como, algumas regras para a gerência de bancos de dados temporais, incluindo as operações de inserção, atualização e remoção de suas informações.

# 4 Gerência de Dados em Bancos de Dados Temporais

Os bancos de dados convencionais possibilitam a gerência das informações através de operações básicas: inserção, atualização, exclusão e seleção. Tais operações são encontradas em todos os bancos de dados comerciais. Nestes bancos de dados, as operações de atualização e remoção podem ser realizadas sobre qualquer tupla, desde que o usuário possua permissão para tal.

Para que as atividades relacionadas à manutenção da base possam ser executadas, os BDs tradicionais possuem suas próprias restrições de integridade, as quais buscam manter a consistência da base de dados. Tais restrições são importantes por fornecerem garantias aos usuários quanto à veracidade das informações armazenadas, pois, caso um dado seja pesquisado em um banco de dados convencional, todos os usuários interessados por esta informação desejam obter o mesmo resultado.

Os bancos de dados temporais (BDTs) também gerenciam as informações através das mesmas operações básicas: inserção, atualização, exclusão e seleção. Entretanto, nos BDTs as operações de atualização não são realizadas da mesma forma. Nenhuma informação pode ser substituída por outra, devendo ambas ficar armazenadas no BD.

A exclusão, em um BDT, pode ser de duas formas: exclusão lógica e exclusão física. A exclusão lógica, a qual norteia o conceito deste tipo de banco de dados, é realizada encerrando-se a "*vida*" da informação.

A exclusão física é utilizada quando se deseja remover fisicamente uma informação que não é mais relevante. Esta operação é conhecida por *vacuuming* e é realizada raramente, somente quando se quer diminuir o número de dados armazenados. Entretanto, deve-se estar ciente de que estas informações serão perdidas. Desta forma, pode-se criar restrições, dando direitos aos administradores da base e cuidando para que a mesma permaneça consistente. A operação de *vacuuming* pode ser realizada de 3 formas:

- a informação válida anteriormente será válida durante toda a validade da informação excluída; ou
- a informação válida posteriormente iniciará sua validade quando a validade da informação excluída iniciou; ou
- o tempo no qual esta informação era válida poderá ser utilizado por outra tupla que será inserida.

Como nos BD convencionais, para que as atividades relacionadas à manutenção da base possam ser realizadas, os BDs temporais também possuem suas próprias regras de gerência, as quais buscam manter a consistência da base de dados. Tais regras dependem do tipo de banco de dados temporal a ser utilizado: TT, TV ou bitemporal.

A seguir, é feita uma análise detalhada das operações para cada um dos tipos de BDT e das ações que o gerenciador deve realizar.

## 4.1 Bancos de Dados Temporais de Tempo de Transação

Os BDT de TT, como apresentado no capítulo 2, mantêm o passado das informações. As informações podem ser recuperadas de acordo com o tempo no qual foram inseridas na base de dados. A principal característica deste tipo de BD é a transparência temporal aos usuários, uma vez que os tempos são fornecidos pelo SGBD.

Com o objetivo de ter uma visão geral da estrutura das informações, considera-se a figura 4.1, onde os valores de 1 a 3 correspondem aos tempos de transação em uma base de dados e os valores de A a C são informações inseridas nesta base. Assim:

- **A**: foi inserida no BD no momento 1 e deixou de ser válida no momento 2;
- **B**: foi inserida no BD no momento 2 e deixou de ser válida no momento 3;
- **C**: foi inserida no BD no momento 3 e será válida até que uma nova informação seja inserida na base de dados.

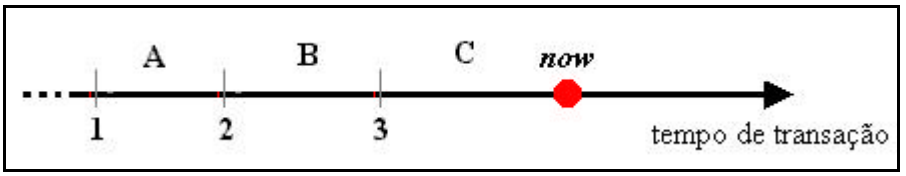



FIGURA 4.1 - BDT de Tempo de Transação: Visão Geral

### 4.1.1 Inserção em um Banco de Dados Temporal de Tempo de Transação

A solicitação de inserção de uma nova informação em um BDT de TT é executada de diferentes formas conforme o tipo de rótulo temporal utilizado.

*- Pontos no Tempo*

Quando se utiliza pontos no tempo para representar o tempo de "vida" de uma informação, o instante da transação é anexado à informação.

TABELA 4.1 - Exemplo Inserção Pontos no Tempo

| Código | Endereço | TT |
|---|---|---|
| 001 | Rua das Oliveiras, 400 | 01/12/1998 |
| 002 | Rua das Figueiras, 500 | 10/01/2000 |
| ... | ... | ... |

Na tabela 4.1 são apresentadas duas informações já inseridas, juntamente com seus TTs. Desta forma se conclui que a informação de código 002 será válida a partir do dia 10/01/2000.

- *Intervalo Temporal*

Quando se utiliza intervalos temporais para representar o tempo de "vida" de uma informação, o rótulo temporal da mesma deve possuir o instante no qual inicia sua transação (quando está sendo inserida) e o instante no qual encerra sua transação.

TABELA 4.2 - Exemplo Inserção Intervalos Temporais

| **Código** | **Endereço** | **TT_inicial** | **TT_final** |
|---|---|---|---|
| 001 | Rua das Oliveiras, 400 | 01/12/1998 | *Null* |
| 002 | Rua das Figueiras, 500 | 10/01/2000 | *Null* |
| ... | ... | ... | ... |

Na tabela 4.2 são apresentadas as mesmas informações da tabela 4.1. Entretanto, os instantes de tempo inicial e final são registrados, representando quando inicia e quando encerra sua "vida". Neste caso, *null* indica que continua válido. No exemplo, a tupla de Código 002 terá o seu *endereço* válido de 10/01/2000 até infinito (representado por *null*). O infinito apenas é substituído quando houver uma atualização na base de dados e novas informações, referentes a este mesmo Código 002, ferem inseridas. Estas atualizações serão exploradas no item 4.1.2.

- *Elemento Temporal*

Para a inserção de informações, utilizando elemento temporal, se procede da mesma forma que para intervalo temporal, pois um elemento nada mais é que um conjunto de intervalos. Assim, a inserção de uma nova informação será a inserção do primeiro intervalo.

### 4.1.2 Atualização em um Banco de Dados Temporal de Tempo de Transação

Nos BDT não existe o termo "sobrescrever" uma informação. Desta forma, as atualizações são transformadas em novas inserções na base de dados. Deve-se criar uma nova tupla, sendo que esta tupla apenas será válida a partir do momento da sua criação. E, além disso, deve-se encerrar a validade da tupla anterior.

- *Pontos no Tempo*

Tratando-se de pontos no tempo, a nova tupla é inserida com o TT associado. Isto implica na finalização da validade da última tupla que tinha sida inserida. Sabe-se quanto tempo uma tupla foi válida pela comparação dos tempos de transação de duas tuplas.

TABELA 4.3 - Exemplo Pontos no Tempo

| **Código** | **Endereço** | **TT** |
|---|---|---|
| .... | ... | ... |
| 001 | Rua das Oliveiras, 400 | 01/12/1998 |
| 001 | Rua das Figueiras, 500 | 10/01/2000 |
| ... | ... | ... |

Na tabela 4.3 pode-se observar que a *pessoa* de código *001* teve seu *endereço* alterado em dois instantes diferentes do tempo. O primeiro, no dia *01/12/1998* e o

segundo, no dia *10/01/2000*. Desta forma, pode-se concluir que esta informação de *endereço*: *Rua das Oliveiras, 400,* foi válida para o sistema do dia 01/12/1998 até o dia 09/01/2000.

- *Intervalo Temporal*

O tempo de transação final da nova informação deve receber um valor que indique que esta informação será válida até que outra seja inserida (por exemplo: *null*). Além disso, a informação válida antes de uma nova ser inserida deve ser finalizada, fazendo com que seu tempo final de transação receba o tempo inicial de transação da tupla inserida menos uma unidade da granularidade temporal utilizada.

TABELA 4.4 - Exemplo Intervalos

| **Código** | **Endereço** | **TT Inicial** | **TT Final** |
|---|---|---|---|
| .... | ... | ... | ... |
| 001 | Rua das Oliveiras, 400 | 01/12/1998 | 09/01/2000 |
| 001 | Rua das Figueiras, 500 | 10/01/2000 | *Null* |
| ... | ... | ... | ... |


Por exemplo, na tabela 4.4, o apontador 1 indica o tempo de transação final de uma tupla, o qual foi preenchido automaticamente pelo SGBD quando a tupla de *endereço*: "Rua das Figueiras, 500" foi inserida. O tempo de transação final desta nova tupla apenas será preenchido quando outra tupla for inserida.

- *Elemento Temporal*

A atualização em BDT que utilizam elementos temporais é semelhante à dos intervalos temporais. Sua única diferença, é quando se trata da inserção de novas informações com o mesmo valor, o que dará origem a intervalos disjuntos de TT nos quais a mesma informação é válida. Quando ocorrer a inserção de informações com diferentes valores, procede-se da mesma forma que para intervalos temporais.

### 4.1.3 Remoção em um Banco de Dados Temporal de Tempo de Transação

Toda vez que houver um pedido de remoção para uma determinada informação armazenada na base, o mesmo deve resultar em uma finalização do tempo de "vida" da tupla, gerando uma exclusão lógica.

- *Pontos no Tempo*

Como uma tupla em um BDT de TT, que utiliza pontos no tempo, possui apenas o instante no qual a mesma iniciou sua validade, é difícil representar sua "exclusão" da base, pois não existe uma forma de encerrar sua validade. Pode-se realizar esta exclusão de duas formas:

- utilizar um atributo adicional que informe o "estado" da informação, podendo ser: (i) ***A*** - ativo: representando que a informação é válida no BDT; (ii) ***I*** - inativo: representando a exclusão lógica da informação do BDT;

- definir um valor *null* específico para o domínio a partir do instante desejado. Por exemplo: definir um valor *vazio* para um determinado atributo.

- *Intervalo Temporal*

Diferente da representação que utiliza pontos no tempo, um BDT com intervalos temporais apresenta a possibilidade de encerrar a validade da tupla quando desejado. Isto é realizado, atribuindo-se ao tempo de transação final, o instante desejado, o que representará que a informação não é mais válida no BDT.

- *Elemento Temporal*

Procede-se da mesma forma já citada para intervalos temporais.

## 4.2 Bancos de Dados Temporais de Tempo de Validade

Um banco de dados temporal de tempo de validade trata informações que serão válidas em um determinado instante de tempo, não necessariamente no instante de sua inserção à base de dados. Estes tempos são fornecidos pelos usuários.

Considerando a figura 4.2, onde os valores de 1 a 4 correspondem aos tempos de validade em uma base de dados e os valores de A a D são informações históricas inseridas nesta base, pode-se concluir que a informação:

- **A**: iniciou sua validade no BD no momento 1 e deixou de ser válida no momento 2;
- **B**: iniciou sua validade no BD no momento 2 e deixou de ser válida no momento 3;
- **C**: iniciou sua validade no BD no momento 3 e deixou de ser válida no momento 4;
- **D**: iniciou sua validade no BD no momento 4 e será válida até que uma nova informação seja inserida na base de dados.

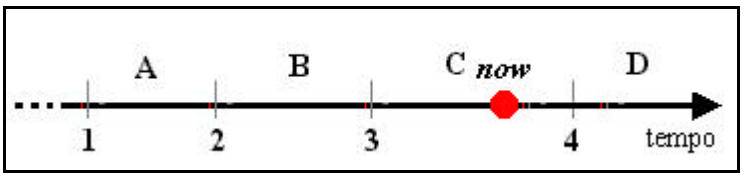



FIGURA 4.2 - BDT de Tempo de Validade: Estado Inicial

### 4.2.1 Inserção em Bancos de Dados Temporais de Tempo de Validade

A solicitação de inserção de uma nova informação em um BDT de TV, conforme os de TT, também é executada de diferentes formas de acordo com o tipo de rótulo temporal utilizado.

- *Pontos no Tempo*

O usuário deve fornecer um único tempo de validade da informação. Este pode pertencer ao passado, presente ou futuro. A mesma será válida a partir do instante definido. A tabela 4.5 apresenta um exemplo para esta definição.

TABELA 4.5 - Exemplo Inserção Pontos no Tempo BDT TV

| **Código** | **Endereço** | **TV** |
|---|---|---|
| 001 | Rua das Oliveiras, 400 | 01/12/1998 |
| 002 | Rua das Figueiras, 500 | 10/01/2000 |
| ... | ... | ... |

- *Intervalo Temporal*

O usuário deve fornecer o tempo de validade inicial e o final da informação que está sendo inserida. A informação será válida somente no intervalo definido. A tabela 4.6 apresenta um exemplo para esta definição.

TABELA 4.6 - Exemplo Inserção Intervalos Temporais BDT TV

| **Código** | **Endereço** | **TV_inicial** | **TV_final** |
|---|---|---|---|
| 001 | Rua das Oliveiras, 400 | 01/12/1998 | 10/03/2000 |
| 002 | Rua das Figueiras, 500 | 10/01/2000 | 05/05/2000 |
| ... | ... | ... | ... |

- *Elemento Temporal*

Para a inserção de informações, utilizando elemento temporal em BDT de TV, se procede da mesma forma que para intervalo temporal, pois um elemento nada mais é que um conjunto de intervalos. A inserção de uma nova informação será a inserção do primeiro intervalo.

### 4.2.2 Atualização em Bancos de Dados Temporais de Tempo de Validade

A atualização em BDT de TV resulta em novas inserções na base de dados, uma vez que as informações não podem ser substituídas por se tratar de um banco de dados que visa manter a história das informações.

- *Pontos no Tempo*

Utilizando pontos no tempo, o tempo de validade da tupla será representado por um único valor. A seguir são analisadas as diferentes possibilidades de atualização, conforme o TV definido.

- *TV maior do que todos os TVs associados a informações do BD*

Quando o TV do novo valor que está sendo definido, é maior que todos os TVs das demais informações que compõem a base, esta é a última informação inserida ao BD.

O TV da nova informação indica quando a mesma iniciará sua validade e, da mesma forma, quando a anterior deverá encerrar a sua. A figura 4.3 (a) apresenta o estado de uma base de dados, referente a um determinado conjunto de valores de um mesmo atributo, antes da inserção de uma nova informação.

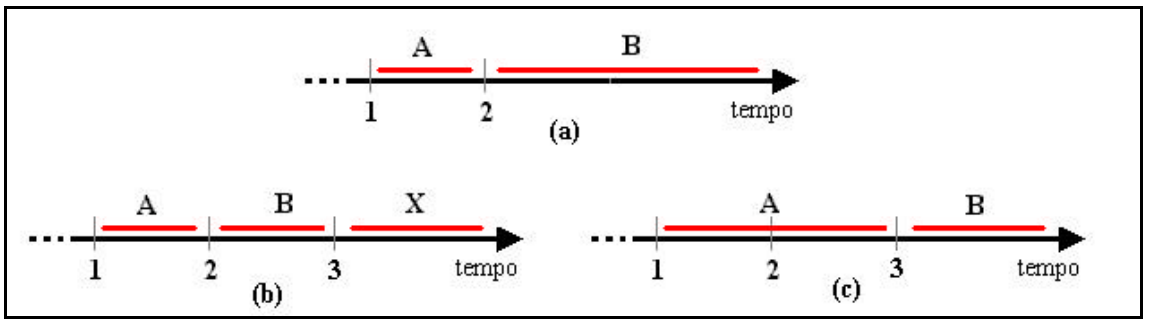



FIGURA 4.3 - Conjunto de Valores de um Atributo

Na figura 4.3, A vale de 1 até 2 (1 chronon) e B vale de 2 em diante.

O banco de dados assume a representação apresentada na figura 4.3 (b), inserindo a seguinte informação: (X,3), onde:

- **A** permanece com a mesma representação da figura 4.3;
- **B** vale de 2 até 3;
- **X**, que foi inserida, vale de 3 em diante.

Neste caso, a semântica é a inserção de um novo valor.

Quando, por algum acaso, deseja-se corrigir o início da validade da última informação inserida, o banco de dados assume a representação apresentada na figura 4.5, atualizando a informação B para: (B,3), onde:

- **A** vale de 1 a 3, sendo extendido até que B torne-se válida;
- **B** vale de 3 em diante.

Neste caso, a semântica é a alteração (correção) do início da validade da informação B para mais tarde.

- *TV igual ao maior rótulo temporal*

Este item indica que existirá um valor cujo TV será igual ao do novo que está sendo definido.

Neste caso, o SGBD deve substituir fisicamente o último valor que tinha sido definido, pois caso contrário, o BD ficaria com duas informações diferentes para o mesmo rótulo temporal.

Assim, inserindo o conjunto (X,2) a base de dados fica com a seguinte representação, conforme figura 4.4 (b), sendo que a base original é apresentada em 4.4 (a).

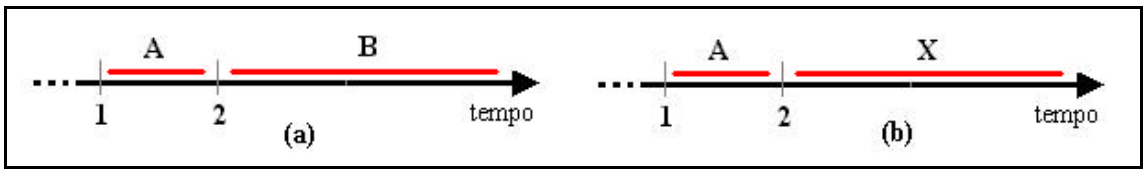



FIGURA 4.4 - Susbtituição do Último Valor

A operação corresponde a correção do último valor definido.

- *TV menor que o último valor definido*

Neste caso, o TV da nova informação que está sendo inserida, é menor que o maior TV das demais informações que compõem a base.

Para esta situação, o SGBDT tem duas opções:

a) substituir fisicamente o último valor definido, cuja semântica é a correção do valor e do tempo do último definido;

b) encerrar a validade da informação quando a última inicia, cuja semântica é a inserção de um valor válido em um intervalo anterior.

A figura 4.5 apresenta estas duas possibilidades, considerando que a informação (X, 1.5) está sendo inserida.

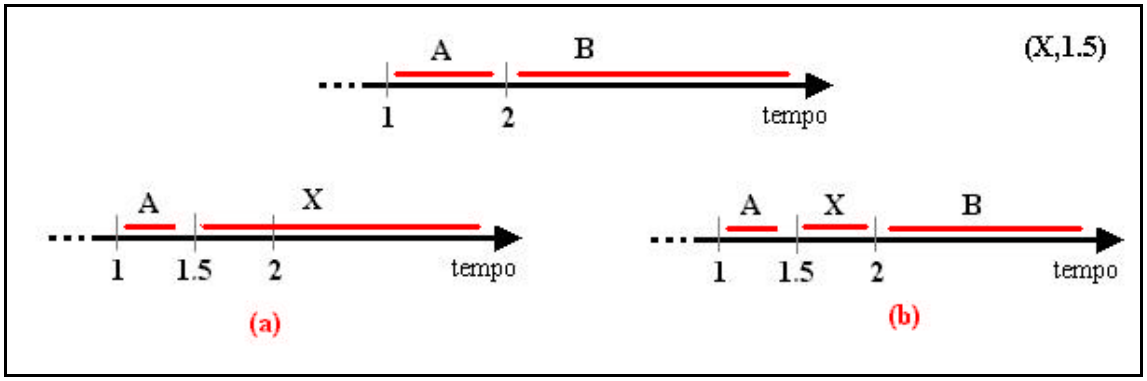



FIGURA 4.5 - Possibilidades de Ações a serem Tomadas pelo SGBDT

Na figura 4.5, item (a) a base de dados possui os seguintes valores: (A,1) e (X,1.5), apresentando a substituição da última informação inserida. No item (b) a base de dados possui os seguintes valores: (A,1), (X,1.5) e (B,2), sendo que a informação inserida X encerra sua validade quando a última inicia.

Além das ações tomadas pelo SGBDT, existe a necessidade de definição do tipo de informação que será inserida. Para isto, também, existem duas possibilidades:

a) valor da informação que está sendo inserida é igual ao último. Neste caso, apenas terá a alteração do TV da informação, fazendo com que a última inserida tenha seu início anteriormente;

b) valor da informação que está sendo inserida é diferente do último. Neste caso, deverá se definir quais atitudes serão tomadas quando da inserção da informação, seguindo as possibilidades apresentadas na figura 4.5;

- *TV coincide com início de algum anterior*

Neste caso, o TV da nova informação que está sendo inserida é igual a qualquer TV de uma informação que compõe a base.

A figura 4.6 (a) apresenta a estrutura de uma base de dados antes de sofrer atualização.

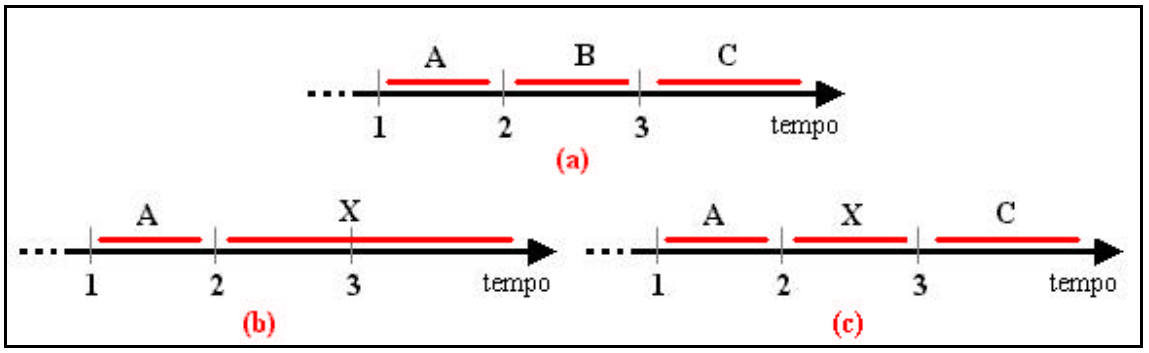



FIGURA 4.6 - Base de Dados e Possibilidades de Atualização

Existem duas possibilidades quando se deseja inserir a informação (X,2) sobre a base de dados:

a) considerar esta como a última válida e eliminar fisicamente todas as informações posteriores, cuja semântica é a correção da base a partir de um ponto;

b) eliminar fisicamente somente a informações coincidente, cuja semântica é a correção deste valor.

Estas possibilidades são apresentadas na figura 4.6 itens (b) e (c).

Na figura 4.6, item (b) a base de dados possui os seguintes valores: (A,1) e (X,2), apresentando a substituição das últimas informações inseridas. No item (c) a base de dados possui os seguintes valores: (A,1), (X,2) e (C,3), sendo que a informação inserida X substitui a informação B e encerra sua validade quando a C inicia.

- *TV dentro de algum intervalo anterior*

Neste caso, o TV do novo valor que está atualizando uma propriedade já inserida, está dentro de algum intervalo formado por duas informações já inseridas, sendo que a informação com tempo exatamente posterior que o TV inserido não é o último. Uma base de dados já consistente é apresentada na figura 4.7.

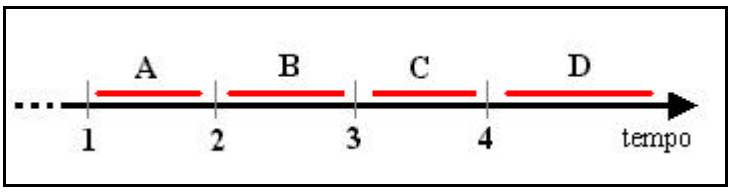



FIGURA 4.7 - Base de Dados Consistente

Existem duas possibilidades básicas para a atualização desta base de dados: atualizar o TV de alguma informação, como por exemplo antecipar a validade da **B** ou atualizar a base com um novo valor.

- antecipar a validade de **B**: neste caso, pode-se inserir a seguinte informação: (B,1.5), onde a base de dados apresentará a estrutura apresentada na figura 4.8. A semântica deste item é a atualização do TV de uma informação;

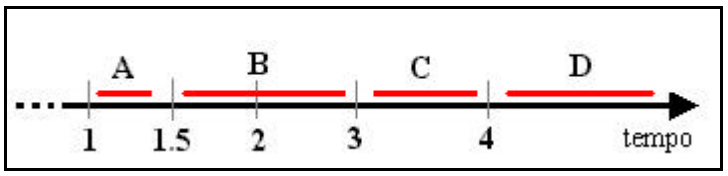



FIGURA 4.8 - Antecipação da Validade de B

- atualizar a base de dados com um novo valor e um novo TV: neste caso, deve-se definir a atitude do SGBDT, pois existem duas possibilidades - (a) a informação inserida vale até que outra torne-se válida, cuja semântica é a inserção de um novo valor até que outro torne-se válido; (b) a informação substituirá todas as demais informações existentes na base a partir de sua existência, cuja semântica é a substituição de todos os valores da base a partir do TV da informação que está sendo inserida. Conforme apresenta a figura 4.9.

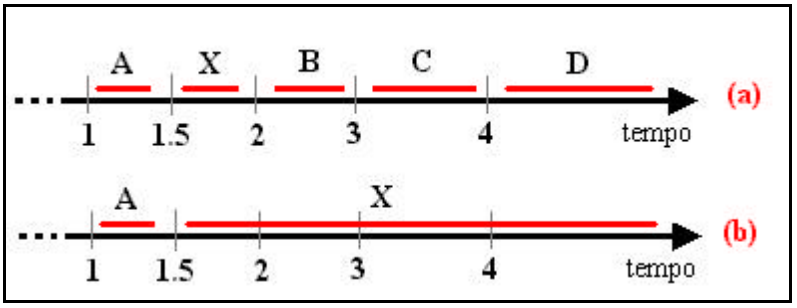



FIGURA 4.9 - Possibilidades para Atualização com Valores Diferentes

- *Intervalo Temporal*

Utilizando intervalos temporais, o usuário deve fornecer o TVI (tempo de validade inicial) e o TVF (tempo de validade final).

- *TVI maior que o maior TVF*

Neste caso, o TVI da informação que está sendo inserida é maior que o TVF da última informação inserida, indicando que a mesma será a última.

Quando se trata de intervalos podem existir intervalos abertos (valendo até infinito, ou seja, até que outra informação seja inserida) ou fechados (com valores definidos). Serão apresentadas características para estas duas realidades:

- *intervalo aberto*: última informação válida na base possui TVF igual a infinito (representações como: *null*, >>, ∞, etc.). A base da figura 4.10 apresenta esta realidade, onde existem duas informações: (A,10,20) e (B,20,>>). Neste caso, a inserção de uma informação (X,30,>>) faz com que o intervalo anterior seja encerrado, ou seja, (B,20,29) e que X seja válido até que outra informação seja inserida. Ao utilizar intervalos abertos, não significa que sua utilização seja permanente, pode-se utilizar um intervalo fechado assim que se desejar. A inserção da informação X é apresentada na figura 4.11.

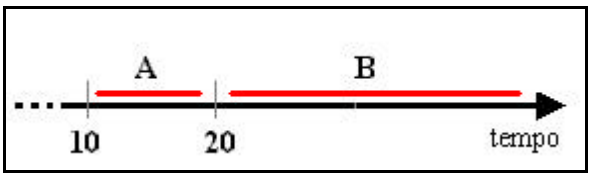



FIGURA 4.10 - Informações com seus Intervalos

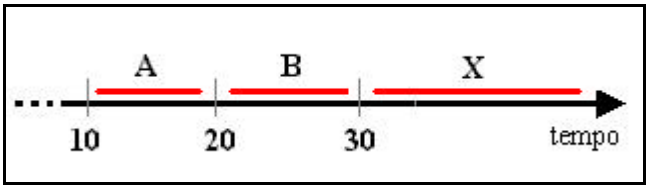



FIGURA 4.11 - Inserção de uma Informação com Intervalo Aberto

- *intervalo fechado*: o TVI da informação que está sendo inserida pode ser exatamente maior que o TVF da informação já inserida, ou pode ser simplesmente maior, fazendo com que exista um intervalo onde não existe um valor válido para esta situação, como apresenta a **f**gura 4.12 nos itens **a** e **b** respectivamente, considerando uma base de dados original onde: (A,10,19) e (B,20,29).

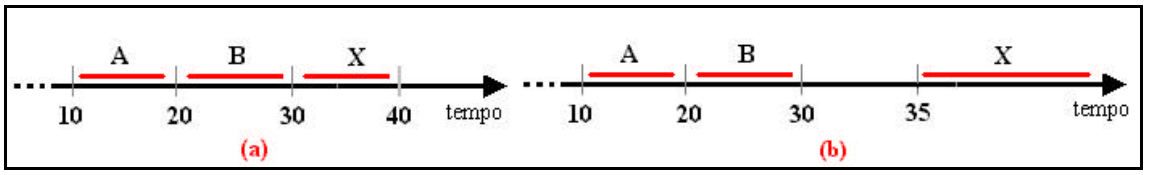



FIGURA 4.12 - Inserção de Informação Onde a Anterior Possui Intervalo Fechado

No item **a** da figura 4.12 foi inserido (X,30,40), desta forma, todos os instantes de tempo possuem algum valor definido. Entretanto, no item **b**, foi inserido (X,35, >>), como poderia ter sido inserido (X, 35, 40), o que faz com que exista um intervalo de 30 a 34 onde não existem valores válidos para esta informação.

- *TVI igual a qualquer TVI definido*

Para este caso, onde o TVI da informação que está sendo inserida é igual a qualquer TVI de uma informação já inserida, existem três variações para o TVF. Estas variações são analisadas a seguir.

- TVF menor que qualquer TVF já inserido: quando uma informação com TVI igual a qualquer outro TVI já definido na base de dados for inserido e o TVF fornecido é menor que qualquer outro já inserido, a informação cujo TVF é maior conitnuará sendo válida na base de dados, sendo que seu TVI será adiado. Entretanto, todas as outras informações deverão ser fisicamente excluídas da base. A figura 4.13, item (a), apresenta um exemplo para esta realidade, cuja semântica é a atualização de uma informação com a prorrogação da validade inicial de um dado já inserido à base.

- TVF igual a qualquer TVF já inserido: neste caso, existe a exclusão física de uma ou mais informações já inseridas na base, observando se o valor que está sendo inserido não é igual ao valor de uma das informação afetadas. Caso isto seja verdade, a semântica é a correção de um valor já definido, caso contrário, é a substituição física de informação da base de dados. A figura 4.13, item (b), apresenta esta realidade.

- TVF válido até o infinito: da mesma forma que o item anterior, este caso pode ser evidenciado para duas realidades: (i) valor igual a algum já definido - correção da validade de uma informação ou (ii) valor diferente aos já definidos - correção do valor e substituição física dos demais. A figura 4.13 (c) apresenta esta realidade.

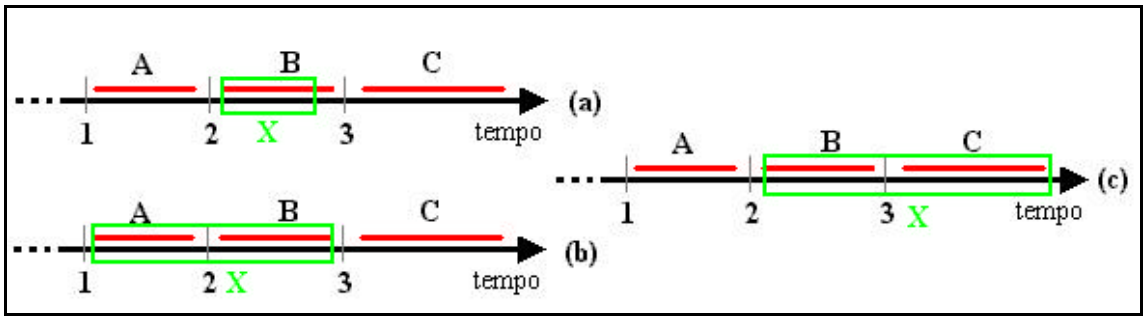



FIGURA 4.13 - TVI Igual a Qualquer Outro já Definido

- *TVI maior que qualquer TVI definido*

Para este caso, onde o TVI da informação que está sendo inserida é maior a qualquer TVI de uma informação já inserida, existem três variações para o TVF, como apresentado no caso anterior. Estas variações são analisadas a seguir.

- TVF menor que qualquer TVF já inserido: quando uma informação com TVI maior que qualquer outro TVI já definido na base de dados for inserido e o TVF fornecido é menor que qualquer outro já inserido, existirão dois intervalos diferentes nos quais a informação anterior à atualização é válida. A figura 4.14, item (a), apresenta um exemplo para esta realidade, cuja semântica é a inserção de uma informação entre um intervalo, sendo que a informação anterior não é perdida.

- TVF igual a qualquer TVF já inserido: neste caso, a informação, cujo TVI é menor que o da informação que está sendo inserida, não será perdida, apenas as demais (se for o caso) serão fisicamente substituídas pela nova informação. A figura 4.14, item (b), apresenta esta realidade, cuja semântica é a alteração do TVF de uma informação para mais cedo e a substituição física das demais (caso existam).

- TVF válido até o infinito: da mesma forma que o item anterior, existe a substituição física de alguns valores definidos na base e o término da validade da primeira informação afetada é antecipado. A figura 4.13 (c) apresenta esta realidade, cuja semântica é a substituição física dos últimos valores da base, sem perder a primeira afetada.

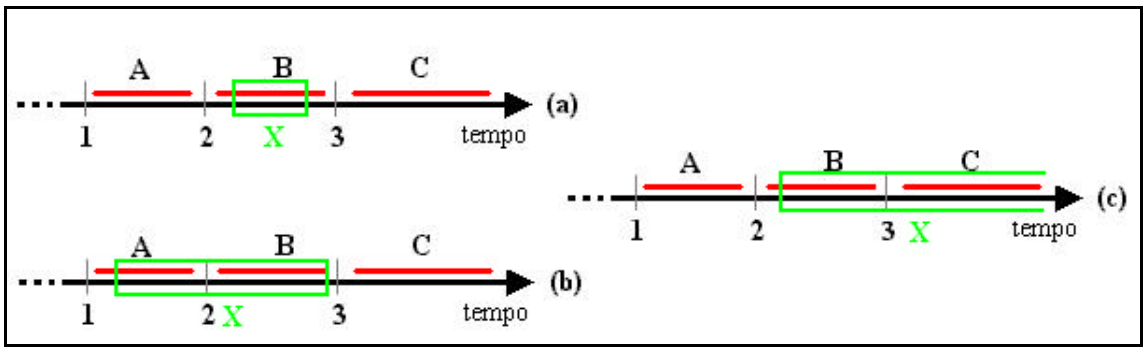



FIGURA 4.14 - TVI Maior que Qualquer TVI já Definido

*- Elemento Temporal*

Tratando-se de elementos temporais, como já mencionado anteriormente, procede-se de forma semelhante que para intervalos temporais. Desta forma, para cada novo intervalo, no qual a informação será válida, deverão ser realizados os teste apresentados acima. O que irá gerar maior diferença, será quando da implementação.

A figura 4.15 apresenta um exemplo de intervalos temporais, considerando o mesmo valor para uma informação válido em diferentes instantes do tempo. Nesta figura, a informação *departamento = vendas* é válida em três momentos diferentes no BD. Inicialmente, é válida no intervalo (10,30), após, de (40, 50) e, finalmente de 60 até que outro intervalo seja criado ou que outro valor seja definido para a informação *departamento.* Desta forma, os testes apresentandos para os intervalos temporais são válidos para os elementos temporais, uma vez que os mesmos são formados por conjuntos de intervalos.

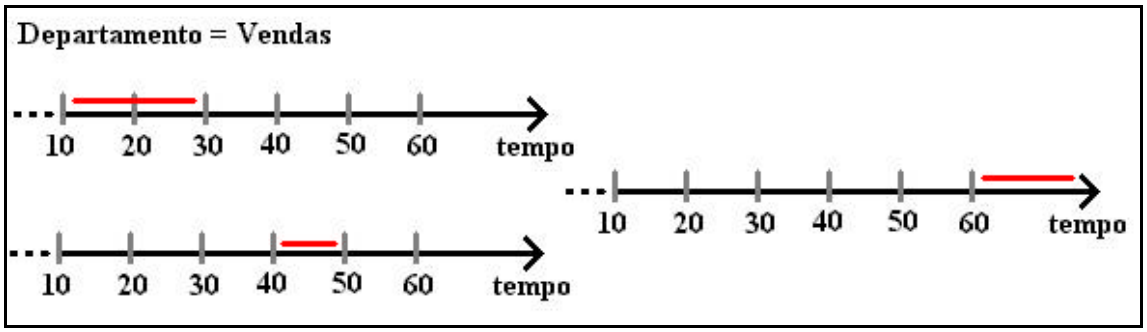



FIGURA 4.15 - Exemplo de Elemento Temporal

### 4.2.3 Remoção em um Banco de Dados Temporal de Tempo de Validade

A validade das informações, sendo inserida pelo usuário, facilita a manutenção da base quando refere-se à exclusão.

A exclusão em uma base de dados temporal ocorre quando o usuário não deseja que uma informação permaneça sendo válida. Desta forma, encerra-se sua validade.

Entretanto, quando a representação do tempo é realizada por pontos no tempo se necessita de algo a mais que represente esta exclusão, pois a informação possui apenas uma informação referente ao tempo de validade e não duas como nos intervalos. Neste caso, sugere-se a inclusão de um atributo que identifique se esta informação permanece logicamente inserida na base ou não, um exemplo, seria a utilização de *null*.

## 4.3 Bancos de Dados Temporais Bitemporais

Os bancos de dados bitemporais, apresentando características de bancos de dados de tempo de transação e de tempo de validade, deverão manipular ambos os tempos. Desta forma, diferentes operações deverão ser executadas quando forem inseridas, atualizadas e/ou removidas informações da base de dados. O tempo de transação é fornecido pelo SGBD quando uma nova informação é inserida e o tempo de validade é fornecido pelo usuário.

A figura 4.16 é formada por três ambientes: (i) informações com tempo de transação; (ii) informações com tempo de validade e (iii) informações com ambos os tempos: TT e TV, sendo chamado de bitemporal. Analisando-se apenas o ambiente bitemporal pode-se ter uma completa visualização dos três ambientes. Portanto, os valores de 1 a 8 correspondem aos TV na base de dados, os valores de I a VIII correspondem aos TT na base de dados e os valores de A a H são informações inseridas a esta base, pode-se concluir que a informação:

- **A**: foi inserida ao BD no momento I, sendo válida de 1 a 2;
- **B**: foi inserida ao BD no momento II, sendo válida de 2 a 3;
- **...**
- **H**: foi inserida ao BD no momento VIII, sendo válida de 8 até que uma nova informação seja inserida na base de dados.

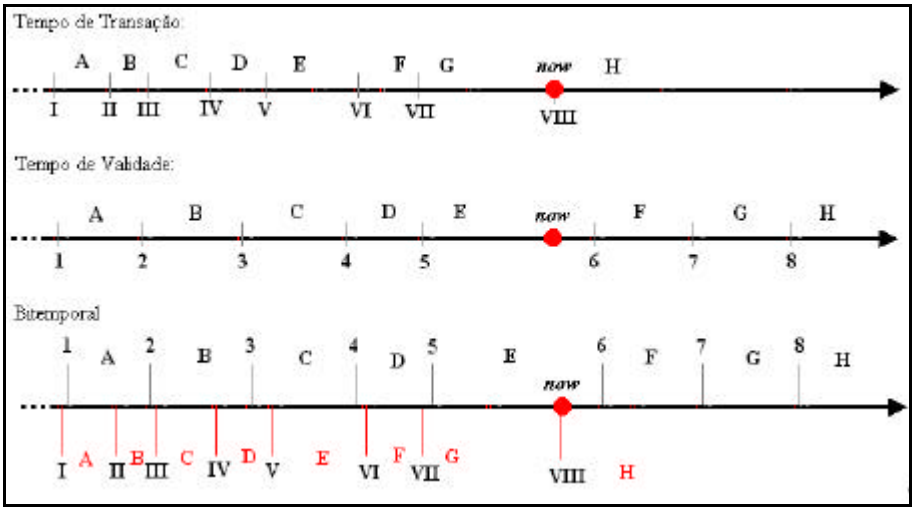



FIGURA 4.16 - BDT Bitemporal

Como apresentado na figura 4.16, o tempo de transação nunca pertencerá ao futuro, uma vez que é fornecido pelo SGBD no momento da inserção da informação.

Neste item serão apresentadas as possíveis operações em um BDT bitemporal, analisando o que ocorre quando as informações serão inseridas, atualizadas e removidas.

### 4.3.1 Inserção em um Banco de Dados Temporal Bitemporal

É a primeira definição de um valor para um atributo. Qualquer informação que seja inserida em uma base de dados bitemporal deverá receber seus respectivos TT (fornecido pelo SGBD) e TV (fornecido pelo usuário).

- *Pontos no Tempo*

O usuário deve fornecer um único tempo de validade à informação. O tempo de transação será fornecido pelo SGBD. A tabela 4.7 apresenta um exemplo para esta definição, com valores para dois atributos.

TABELA4.7 - Exemplo Inserção Pontos no Tempo BDT Bitemporal

| **Código** | **Endereço** | **TV** | **TT** |
|---|---|---|---|
| 001 | Rua das Oliveiras, 400 | 01/12/1998 | 01/12/1998 |
| 002 | Rua das Figueiras, 500 | 10/10/2000 | 03/01/2000 |
| ... | ... | ... | ... |

- *Intervalo Temporal*

O usuário deve fornecer o tempo de validade inicial e o final da informação que está sendo inserida. O SGBD fornecerá os tempos de transação inicial e final da tupla, sendo que o final somente será fornecido quando uma nova instância, desta informação, for inserida, gerada por uma atualização da base de dados. A tabela 4.8 apresenta um exemplo para esta definição. Quando o usuário não fornecer o TVF, ele será igual a infinito (valendo até que outra informação seja definida).

TABELA4.8 - Exemplo Inserção Intervalos Temporais BDT Bitemporal

| **Código** | **Endereço** | **TV_inicial** | **TV_final** | **TT_inicial** | **TT_final** |
|---|---|---|---|---|---|
| 001 | Rua das Oliveiras, 400 | 01/12/1998 | 10/03/2000 | 01/12/1998 | *null* |
| 002 | Rua das Figueiras, 500 | 10/10/2000 | 05/12/2000 | 03/02/2000 | *null* |
| ... | ... | ... | ... | ... | ... |

- *Elemento Temporal*

Como nos demais tipos de BDs, o elemento temporal é igual ao intervalo temporal, tratando-se da inserção de novas informações à base.

### 4.3.2 Atualização em um Banco de Dados Temporal Bitemporal

A atualização em um BDT bitemporal segue as mesmas características apresentadas por um BDT de TV. Entretanto, as informações não serão perdidas ou

excluídas da base em resultado a uma atualização, uma vez que o TT fornecido às informações possibilita a distinção de duas ou mais que possuam TV iguais.

Deste modo, qualquer definição de novo valor é realizada, sendo a semântica dos valores definida pelo TT.

### 4.3.3 Remoção em um Banco de Dados Temporal Bitemporal

A exclusão em uma base de dados bitemporal ocorrerá quando o usuário encerrar a validade de uma informação. No caso da utilização de intervalos ou elementos, atribuir um tempo ao tempo de validade final. No caso de pontos no tempo, deverá ser utilizado um atributo que indique quando a informação está logicamente inserida ou não. Ou seja, a exclusão em um BDT Bitemporal ocorre da mesma forma que em um BDT de tempo de validade.

O tempo de transação permanece o mesmo, pois é o tempo no qual a informação foi inserida na base de dados.

## 4.4 Considerações Finais

Este capítulo apresentou as operações de manutenção (inserção, atualização e remoção) e suas características em uma base de dados temporal. Como nos bancos de dados convencionas, existem restrições que buscam manter a base de dados consistente.

Diferentes casos foram apresentados para os BDTs de tempo de transação, de tempo de validade e bitemporais. Os mesmos basearam-se, principalmente, na atualização das informações, as quais, em BDTs geram novas inserções. Desta forma, a atualização resulta em novas operações de inserção à base.

Através deste capítulo, pode-se comparar e definir quais as operações que serão permitidas sobre uma base de dados temporal a ser implementada. O projetista desta base pode definir inserções de informações válidas apenas no presente, no passado, no futuro ou realizar suas próprias combinações.

# 5 O Modelo de Dados TF-ORM

TF-ORM (*Temporal Functionality in Objects With Roles Model*) [EDE 93, 94a] é um modelo de dados temporal orientado a objetos, que utiliza o conceito de papéis para representar diferentes comportamentos de um objeto. Permite a modelagem dos aspectos estáticos e dinâmicos de uma aplicação, pois considera todos os estados do objeto e associa informações temporais às propriedades que podem mudar de valor ao longo do tempo. É um modelo bitemporal, ou seja, suporta tanto tempo de transação, quanto tempo de validade.

O conceito de papéis tem por objetivo separar a representação dos aspectos dinâmicos de um objeto, dos seus aspectos estáticos. O objeto poderá assumir, ao longo de sua existência, um ou mais papéis, em diferentes tempos, continuando a ser uma instância de uma única classe. O objeto, com o conceito de papéis, pode apresentar, simultaneamente, diversas instâncias de um mesmo papel, o que possibilita a representação de variações do seu comportamento. E, ainda, instâncias de papéis diferentes no mesmo instante de tempo.

Muitas vezes, o conceito de papéis é confundido com o de subclasses, cuja diferença fundamental está na identidade dos objetos [EDE 94]. Quando representações através de papéis são utilizadas, o objeto existe como instância da classe, independentemente do fato de estar ou não desempenhando um ou mais papéis. Considere o exemplo: se um determinado objeto *empregado* fosse representado como uma subclasse de *pessoa*, este objeto apenas poderia ser criado no momento em que esta pessoa tivesse um emprego. Utilizando a representação de empregado através de um papel, o objeto pessoa existe independentemente desta pessoa possuir um emprego.

## 5.1 Classes Agente, Recurso e Processo

As classes podem ser de três tipos distintos, pré-definidos: *classes agente*, *recurso* e *processo*.

***Classes de Recursos**:* definem a estrutura de um recurso (documento, dado ou objeto manipulado) em termos dos papéis que este recurso pode apresentar durante seu ciclo de vida.

***Classes Agente**:* uma classe agente representa as pessoas. É representada de maneira semelhante a uma classe de recurso, sendo incluída a definição de um conjunto de decisões que o agente pode tomar.

***Classes de Processos**:* integram as classes de recursos e de agentes, permitindo a descrição do trabalho realizado em termos de sua organização e da cooperação entre os agentes envolvidos. A classe de processos representa a manipulação de informações através dos procedimentos executados. Nesta classe, o conceito de papéis também é utilizado. A descrição é realizada através destes papéis com propriedades, mensagens, estados e regras. Os procedimentos são divididos em tarefas e cada papel, com exceção

do papel básico, corresponde a uma tarefa. As operações sobre uma determinada tarefa são descritas através de mensagens que podem ser recebidas e/ou enviadas pelo papel.

## 5.2 Definição de Classes e de Papéis

Uma classe é definida por um nome único em toda a especificação de classes e por um conjunto de papéis:

$$\text{Classe Agente} = (Nca, Pa_0, Pa_1, \ldots, Pa_i, \ldots, Pa_n)$$
$$\text{Classe Recurso} = (Ncr, Pr_0, Pr_1, \ldots, Pr_i, \ldots, Pr_n)$$
$$\text{Classe Processo} = (Ncp, Pp_0, Pp_1, \ldots, Pp_i, \ldots, Pp_n)$$

Onde:

Nca = Nome de classe agente

Ncr = Nome de classe recurso

Ncp = Nome de classe processo

$Pa_x$ = Conjunto de papéis agente.

$Pr_x$ = Conjunto de papéis recurso.

$Pp_x$ = Conjunto de papéis processo.

Cada papel de uma classe de recurso ($Pr_i$) e de uma classe de processo ($Pp_i$) consiste de um nome ($Np_i$), um conjunto de propriedades ($Pr_i$) do papel, um conjunto de estados abstratos $S_i$ que o objeto pode apresentar neste papel, um conjunto de mensagens $M_i$ que o objeto pode receber e enviar de um conjunto de regras $R_i$ (regras de transição de estado e regras de integridade) [EDE 94a]:

$$P_i = < Np_i, Pr_i, S_i, M_i, R_i >$$

Para as classes do tipo agente, além das características já apresentadas, na definição de papel pode aparecer um conjunto $D_i$ de decisões:

$$P_i = < Np_i, Pr_i, S_i, D_i, M_i, R_i >$$

Existem dois tipos de propriedades que podem ser identificadas em aplicações que evoluem com o passar do tempo:

- **Propriedades estáticas**: que nunca mudam de valor. Somente um valor pode ser definido para uma propriedade estática, permanecendo o mesmo durante toda a existência do objeto;
- **Propriedades Dinâmicas**: podem representar diversos valores durante a existência do objeto. Todos os valores definidos para uma propriedade dinâmica devem ficar armazenados permanentemente, pressupondo a implementação de um banco de dados temporal.

A cada propriedade é associado um domínio, estando disponível, além dos domínios usuais (inteiro, real, string, conjunto, objetos de outra classe) um conjunto de atributos temporais.

## 5.3 Definição de Mensagens

Todas as mensagens que uma instância de um papel pode enviar ou receber devem ser definidas, com seus parâmetros e origem/destino. Deve ser definido um domínio para cada parâmetro, podendo ser utilizado qualquer dos domínios definidos para as propriedades.

Mensagens podem ser enviadas ou recebidas de classes ou de papéis. A identificação de receptor/emissor é feita através do nome da classe (opcional, caso o contexto deixe isto claro) e do papel.

Uma mensagem recebida de uma classe seria enviada pelo papel básico desta classe. Uma mesma mensagem pode ser recebida de mais de um emissor, e pode ser enviada simultaneamente a mais de um receptor. A definição de que uma mensagem pode ser recebida de mais de um papel significa que somente um destes papéis necessita enviar a mensagem para que sua regra de transição seja ativada. Já no envio o conceito é outro - ao ser ativada por uma regra de transição, cópias desta mensagem são enviadas a cada um dos papéis definidos como destino.

Uma mensagem pode ser enviada por uma instância de um papel a si mesmo (*to itself*).

Quando o emissor/receptor da mensagem não estiver definido na modelagem, a mensagem deve ser declarada como tendo o mundo externo (*External_World*) como interface. Deste modo é possível modelar aplicações em vários níveis de detalhamento.

## 5.4 Definição de Decisões

As classes de agentes podem apresentar um conjunto de decisões. Uma decisão pode apresentar parâmetros, devendo para cada um ser definido um domínio. As decisões representam a parcela de trabalho não estruturado, modelada na aplicação.

## 5.5 Definição de Regras

Dois tipos de regras podem ser definidas em TF-ORM: regras de transição de estados e regras de integridade. Todas as regras recebem um nome, nome este que é utilizado somente para facilitar a modelagem - através deste nome pode-se identificar qual o funcionamento da regra.

### 5.5.1 Regras de Transição de Estados

As regras de transição de estados definem quais são as mudanças de estados que podem ocorrer em instâncias de um papel. A forma mais geral de uma regra de transição de estados é a seguinte:

*$r_n$: state($s_1$), msg(mi) Þ msg(mo), msg(m1), ..., msg(mn), state($s_2$); (<condição de transição> )*

Esta regra tem o seguinte significado: quando a instância recebe a mensagem *mi*, se esta instância estiver no estado $s_1$ e se a condição de transição for satisfeita, então esta instância envia as mensagem *mo, m1, ..., mn* e muda de estado, assumindo o estado $s_2$.

Nenhum dos elementos acima é obrigatório na definição de uma regra de transição de estados. As seguintes combinações podem ocorrer:

- o estado inicial s1 não é definido - a regra será ativada sempre que chegar a mensagem mi, se a condição for verdadeira, independentemente do estado que a instância estiver desempenhando;
- a mensagem de chegada mi não é definida - a regra será ativada sempre que a instância estiver no estado s1, devendo a condição ser verdadeira;
- nenhuma mensagem de saída é definida - ocorre uma transição de estado sem envio de mensagem;
- o estado final s2 não é definido - a regra, quando ativada, envia alguma mensagem sem que ocorra uma transição de estado;
- a condição não é definida - a ativação da regra fica restrita à chegada da mensagem estando a instância no estado inicial.

Formas alternativas desta regra podem ser utilizadas, relativas à chegada de múltiplas mensagens:

*$r_n$: state($s_1$), {msg($mi_1$), msg($mi_2$), ... , msg($mi_m$)} Þ msg(mo), state($s_2$);*
*( <condição de transição> )*

Neste caso, a transição vai ocorrer somente depois que chegarem todas as mensagens $mi_1$, $mi_2$, ..., $mi_m$. A chegada destas mensagens pode ocorrer em qualquer ordem, em tempos diferentes.

Além disso, uma regra de transição pode ser acionada por uma tomada de decisão:

*$r_n$: state($s_1$), decision($d_1$) Þ msg(mo), state($s_2$); ( <condição de transição> )*

A condição de transição é escrita em lógica temporal, estando detalhada em [EDE 93a].

### 5.5.2 Regras de Integridade

As regras de integridade devem ser sempre satisfeitas por todas as instâncias do papel. Apresentam a seguinte forma:

*constraint ( <pré-condição> Þ <pós-condição> )*

Sempre que a pré-condição de uma regra de integridade for satisfeita, sua pós-condição também o deve ser. Deste modo, estados inconsistentes do banco de dados não são permitidos. As regras de integridade de um papel devem ser verificadas a cada vez que uma regra de transição de estados for ativada - se for constatada a quebra da integridade do banco de dados, esta transição deverá ser desfeita, restaurando os valores anteriores das propriedades modificadas pelo(s) método(s) que implementa(m) a(s) mensagem(s) recebida(s). Além disso, deve ser evitado que a(s) mensagem(s) de saída seja(m) enviada(s).

Entre as diferenças de uma regra de transição e uma regra de integridade está o momento de sua avaliação. As regras de integridade devem ser avaliadas toda vez que um novo estado do banco de dados é criado - correspondendo ao momento logo após a execução de uma transição. O momento de avaliação de uma regra de transição, é anterior à execução da transição. As regras de integridade detectam a ocorrência de uma violação de integridade, enquanto que as regras de transição não permitem a ocorrência de violações.

## 5.6 Identificador de Instâncias

Diversas instâncias de uma mesma classe podem estar presentes simultaneamente ao se tratar de um modelo orientado a objetos. A manipulação das diferentes instâncias, tanto de classes quanto de papéis, requer uma identificação de instâncias.

Um identificador interno, único no sistema, é associado ao objeto, pelo sistema gerenciador do banco de dados, quando da criação do mesmo. Cada instância de um papel, deste objeto, será associada a um identificador, que também apresentará valores únicos no sistema.

Foram definidas duas propriedades especiais para representar este identificador único:

- *old*: presente no papel básico de todas as classes, armazena o valor do identificador de cada objeto da classe;
- *rld*: presente em todos os papéis básicos, armazena os identificadores de cada papel instanciado.

## 5.7 Papel Básico

Descreve as características iniciais de uma instância e as propriedades globais que controlam sua evolução. Somente apresenta uma instância. Suas propriedades aplicam-se a todos os demais papéis.

O papel básico apresenta algumas mensagens pré-definidas, dentre elas:

- *create-object*: criação de uma instância de uma classe (um objeto);
- *suspend-object* e *resume-object*: suspensão e retomada do estado de uma instância.

Os estados do papel básico também são pré-definidos:

- *active*: quando o objeto está ativo. Ao ser criado, um objeto assume este estado;
- *suspended*: quando está temporariamente desabilitado a enviar e a receber mensagens. A única mensagem que pode ser recebida neste estado é a que o coloca novamente no estado ativo.

## 5.8 Superclasse OBJECT

O TF-ORM apresenta uma classe chamada OBJECT, a qual desempenha o papel de superclasse para todas as demais classes definidas. O papel básico desta superclasse contém três propriedades [EDE 94]:

- *old*: propriedade estática, destinada a armazenar o identificador de instância da classe;
- *class-instance*: propriedade dinâmica, destinada à armazenar o início da vida da instância da classe e seus períodos de validade;
- *class-end*: propriedade dinâmica, destinada à armazenar o momento em que deixa de existir.

A superclasse apresenta um papel, denominado ROLE, onde também são definidas três propriedades:

- *rld*: propriedade estática, cuja finalidade é armazenar o identificador de instância de um papel;
- *role-instance*: propriedade dinâmica, cuja finalidade é armazenar o momento de criação de uma instância de um papel e seus períodos de validade;
- *role-end*: propriedade dinâmica, cuja finalidade é registrar o momento a partir do qual a instância do papel deixa de existir.

As mensagens:

- *create-object, resume-object, suspend-object* e *kill* podem ser recebidas por qualquer papel básico das classes derivadas desta superclasse;
- *add-role, resume-role, suspend-role* e *terminate-role* podem ser recebidas por qualquer papel que não seja um papel básico de classes derivadas.

## 5.9 Representação Temporal e Elemento Temporal Primitivo

O tempo, no modelo TF-ORM é linearmente ordenado, variando de forma discreta. A variação dos valores armazenados em um banco de dados que implemente este modelo é considerada na forma escada - uma vez definido um valor, este permanece válido até que outro valor seja definido.

O ponto no tempo foi o elemento temporal primitivo adotado para a representação dos aspectos temporais no TF-ORM.

O *chronon* sugerido para este modelo de dados foi o minuto considerado como a menor granularidade temporal adequada para atividades humanas. Entretanto, a definição do *chronon* a ser utilizado fica a critério da implementação. A definição completa de um ponto no tempo é o instante temporal, formado por (ano, mês, dia) e um horário dentro desta data (hora e minuto).

## 5.10 Linguagem de Consulta

A linguagem de consulta do TF-ORM baseia-se na linguagem SQL, apresentando, também, alguma influência da TQuel [SNO 87]. A forma geral de um comando de recuperação de dados é dada pelas duas estruturas que definem a linguagem de consulta TF-ORM [EDE 94a], através dos exemplos a seguir:

```
SELECT <cláusula de especificação>
FROM <cláusula de identificação>
WHERE <cláusula de busca>
[ON <cláusula de instante temporal>

SELECT <cláusula de especificação>
FROM <cláusula de identificação>
WHEN <cláusula de busca>
[ON <cláusula de instante temporal>
```

Nestas:

- a cláusula de especificação define a estrutura do resultado solicitado;
- a cláusula de identificação identifica os objetos sobre os quais vai ser efetuada a busca;
- a cláusula de busca apresenta a condição que deve ser satisfeita pelos objetos a serem recuperados;
- A cláusula de instante temporal define a data correspondente a uma história passada, mudando o referencial de tempo e execução da consulta para o momento definido e utilizando, como base, as informações conhecidas naquele momento.

No TF-ORM, utiliza-se as cláusula WHEN e WHERE para incorporar aspectos temporais às consultas. A escolha de uma das duas cláusulas define o universo de busca:

- WHERE: a informação é buscada no banco de dados instantâneo correspondente ao instante de tempo considerado. Pode-se referir ao estado atual do banco de dados ou a um estado de uma história passada;
- WHEN: aumenta o universo de busca para todas as informações da história considerada (atual ou passada), incluindo dados passados e futuros, além dos atuais.

Tanisi Carvalho desenvolveu uma proposta para o mapeamento da linguagem de consulta do modelo TF-ORM para bancos de dados relacionais em [CAR 97].

## 5.11 Considerações Finais

Este capítulo apresentou as principais características do modelo de dados temporal TF-ORM, por ser o escolhido para este trabalho. Foram estudadas as diversas características do modelo, dentre as quais, os papéis que fazem com que o TF-ORM se diferencie dos demais modelos orientados a objetos.

Este estudo possibilita a realização do mapeamento das classes, papéis, propriedades dinâmicas e estáticas, mensagens, decisões e regras de transição de estados do modelo TF-ORM, para uma base de dados relacional.

# 6 Estudo de Caso – Prontuário Médico

Com o objetivo de exemplificar as consultas no modelo TF-ORM, escolheu-se como estudo de caso um prontuário médico. Este estudo de caso foi escolhido por apresentar um grande número de informações que mudam de valor com o passar do tempo, onde as informações temporais tornam-se importantes. No processo de análise foram identificados os agentes envolvidos, os recursos manipulados e os procedimentos executados [BAR 99].

O médico é identificado por seu CREMERS e sua especialidade. O paciente é identificado pelo código de seu prontuário médico. Médico e paciente possuem os dados pessoais (nome, data de nascimento, sexo, estado civil) e o endereço como propriedades em comum. No modelo TF-ORM essas propriedades fazem parte da classe PESSOA e os papéis dessa classe são médico e paciente.

Um prontuário médico apresenta um conjunto de informações relativo ao paciente e seu problema. Esse conjunto pode ser dividido em:

- ***Identificação***: dados de identificação do paciente. Os dados básicos, como nome e endereço, são preenchidos pela secretária antes da primeira consulta. Outras informações que podem afetar o diagnóstico, como religião, educação e moradia, devem ser preenchidas pelo próprio médico durante a consulta. Todas as informações pertencentes à identificação do paciente são apresentadas a seguir:
    - número da ficha clínica (número identificador do paciente);
    - dados pessoais:
        - nome,
        - data de nascimento,
        - sexo,
        - profissão,
        - estado civil;
    - endereço:
        - rua,
        - número,
        - complemento,
        - bairro,
        - cidade,
        - estado,
    - telefone 1,
    - telefone 2,
    - fax.
- ***Anamnese***: dados da queixa do paciente. É uma forma padronizada de registrar a entrevista. Deve detalhar como a queixa principal ocorreu e como evoluiu até o momento da consulta, revisar outros sintomas que possam ter significado clínico mas que, eventualmente, tenham sido esquecidos ou menos valorizados [BAR 99]. Revisar como foi o desenvolvimento, a ocorrência de patologias prévias e seu manejo; pesquisar a presença de sintomas ou diagnósticos significativos na família

ou círculo social próximo; definir as condições sociais que favorecem ou desfavorecem o paciente ou modificam o risco e o prognóstico das patologias em questão. Abaixo, encontram-se as informações necessárias para a Anamnese.

- queixa principal:
  - queixa
  - início;
  - intensidade;
  - freqüência;
  - duração;
- fatores:
  - agravantes;
  - atenuantes;
- ocorrência predominante:
  - diurna/noturna;
  - em repouso/em exercício;
  - posição ereta/sentada/deitada);
- história pregressa da doença atual:
  - outros episódios (sim: intensidade, duração);
  - houve tratamento (sim: descrição do tratamento).
- antecedentes
  - pessoais:
    - hábitos (fumo, álcool, drogas, esportes, automedicação);
    - cirurgias anteriores;
    - doenças anteriores.
  - familiares:
    - doenças (asma, diabetes, hipertensão arterial, obesidade, outras).

- ***Exame Físico***: medidas e verificações realizadas pelo médico no consultório. As informações que devem conter no item Exame Físico do paciente são:

  - sinais vitais:
    - tensão arterial (T.A),
    - freqüência cardíaca (F.C.),
    - freqüência respiratória (F.R.),
    - hipótese diagnóstica (H.D.);
  - estágios:
    - inspeção,
    - palpação,
    - percussão;
    - ausculta;
    - torácica pulmonar;
    - torácica cardiológica;
  - revisão de sistemas:
    - respiratório,
    - circulatório,
    - urinário,
    - digestivo;

- hipótese diagnóstica.

- ***Exames e laudos***: solicitações de exames clínicos feitas pelo médico. As informações necessárias ao item Exames e Laudos são:
  - exame;
  - respectivo laudo.
- ***Tratamento***: descrição do tratamento realizado; Possui apenas e item *Descrição*, o qual deve ser preenchido pelo médico.
- ***Retorno***: acompanhamento da evolução do quadro e solicitações de exames físicos dirigidos. A seguir, os itens necessários para o retorno do paciente.
  - evolução;
  - exame físico dirigido.

Devido à grande quantidade de informações existentes no prontuário, a modelagem TF-ORM contém apenas a identificação de pacientes e médicos, bem como a Anamnese, incluindo algumas mensagens e regras.

No item a seguir, através do modelo TF-ORM, será apresentada a modelagem do estudo de caso proposto.

## 6.1 Modelagem no Modelo TF-ORM

Esta modelagem tem o objetivo de apresentar um estudo de caso que possa validar a implementação de um banco de dados temporal, baseado nas regras do modelo.

É importante ressaltar que nem todas as informações foram modeladas, por se tratar de um estudo extenso. As informações mais importantes para a validação da implementação, ou seja, algumas classes, estados, decisões, mensagens e regras foram modeladas e estão sendo apresentadas no anexo 2. Neste item, apenas são apresentadas algumas estruturas importantes para validarem a implementação.

### 6.1.1 Modelagem de Classes e Papéis

A ***identificação***, do item anterior, foi modelada em uma classe e três papéis. Na classe, encontram-se todas as informações em comum aos papéis. A classe foi denominada de PESSOA e seus respectivos papéis: médico, funcionário e paciente. A seguir, apresenta-se a estrutura básica para esta declaração.

```
agent class(
    PESSOA,
  < Base_role,
      static properties = {...}
      dynamic properties = {...}
      rules = {...},
  >,
```

< **Paciente**,
static properties = {...}
dynamic properties = {...}
states = { ...}
messages = {...},
decisions = {…},
rules = {... }
>,
< **Medico**,
static properties = { ...}
dynamic properties = {...}
messages = {…}
states = { ...}
decisions = {... }
rules = {... }
>,
< **Funcionário**,
static properties = {...}
dynamic properties = { ... }
decisions = { ... }
messages = { ... }
states = { ...}
rule = { ... }
>)

Da mesma forma que a ***identificação***, a ***consulta***, ***anamnese*** e demais itens apresentados anteriormente, também foram mapeados para classes e papéis. A representação completa encontra-se no anexo 2.

### 6.1.2 Mapeamento das Propriedades Estáticas e Dinâmicas

Todas as propriedades estáticas (que não variam com o passar do tempo) e dinâmicas (que variam com o passar do tempo) definidas no estudo de caso anterior, foram mapeadas para o modelo TF-ORM. Tais propriedades são representadas da seguinte forma:

**static properties** = {
(sexo, {F,M}),
(dataNasc, date),
(dataObito, date) }
**dynamic properties** = {
(nome, string),
(end, string),
(cidade, string),
(UF, string),
(fone, string),
(fax, string) }

O mapeamento do modelo TF-ORM para as demais propriedades do estudo de caso, encontram-se no anexo 2.

### 6.1.3 Mapeamento das Mensagens, Decisões, Regras e Estados

As mensagens, decisões e regras, as quais devem determinar ações a serem realizadas sobre as informações, também foram mapeadas para o Oracle, desta forma, a representação das mesmas deve ser realizada como segue:

**messages** = {
valores_iniciais (NroFicha:string, Sexo:{F,M}, DataNasc:date, Nome:string, End: string, Cidade: string, Uf: string, Fone: string, Fax: string, Profissao: string, Religiao: string, Educacao: string, Moradia: string) from CONTROLE.PacienteCtrl,
update_dataObito (NroFicha: string, NovaDataObito: date) to CONTROLE.PacienteCtrl,
... },
**decisions** = {
novo_nome (nome: string),
nova_religiao (religiao: string)
…},
**rules** = {
inicio:
msg(add_role) ⇒ state(cadastro),
inicializacao:
state(cadastro), msg(valores_iniciais (NroFicha, Sexo, DataNasc, Nome, End, Cidade, Uf, Fone, Fax, Profissao, Religiao, Educacao, Moradia)) ⇒ state(emConsulta);
(not exists Rid (value(rid, nroFicha) = NroFicha)),
mudança _de_nome:
state(fora_consulta), decision(novo_nome(Nome))⇒state(fora_consulta ),
requis_obito:
msg(update_dataObito(nroFicha, newDataObito)) ⇒ state( noCeu ); (state (oid, rid) = emConsulta OR state (oid, rid) = esperaRetorno OR state (oid, rid) = foraConsulta),
… }

As mensagens e decisões são os disparadores de uma determinada ação e as regras são restrições que permitirão, ou não, que uma determinada ação seja executada. Algumas regras e mensagens adicionais podem ser encontradas no anexo2.

Os estados são informações adicionais sobre determinado objeto. Dependendo do estado em que um objeto se encontrar uma determinada regra poderá ser executada ou não. Sua declaração é simples:

**states** = { cadastro, emConsulta, esperaRetorno, foraConsulta, noCeu }

## 6.2 Considerações Finais

O estudo de caso apresentado neste capítulo forneceu uma visão completa da modelagem de informações através do modelo TF-ORM. O prontuário médico é uma

realidade completa, a qual manipula informações diferenciadas. Desta forma, todas as características do modelo foram exploradas. Dentre elas: classes, papéis, regras, mensagens, decisões e estados.

Esta modelagem possibilitou o estudo concreto do modelo, principalmente no que diz respeito às regras de transição de estados, que serão utilizadas neste trabalho.

O estudo de caso não foi modelado de forma completa, por ser um estudo muito extenso. Entretanto, todas as características necessárias para sua implementação foram exploradas.

# 7 Mapeamento no Oracle

Este capítulo descreve como o mapeamento do modelo TF-ORM para o SGBD Oracle é implementado. Um estudo sobre o Oracle é apresentado no Anexo 1, com o objetivo de apresentar a estrutura das operações e características utilizadas neste trabalho.

A utilização deste SGBD para hospedar a implementação realizada surgiu da necessidade de encontrar um sistema que fosse capaz de suportar as regras do modelo TF-ORM. O Oracle, com todos seus mecanismos – *triggers*, *stored procedures*, *stored functions e views* – satisfaz este requisito. Além disso, é um SGBD comercial muito utilizado que, apresentando característica de SGBDs relacionais, fornece uma estrutura clara para sua utilização. Seu tratamento de datas, além dos demais tipos de dados existentes, também foi decisivo, devido à necessidade de manipulação de um grande número de datas e, em alguns casos, horas, minutos e, talvez, segundos.

O mapeamento é realizado visando manipular tanto o TT quanto o TV das informações, implementando um banco de dados bitemporal.

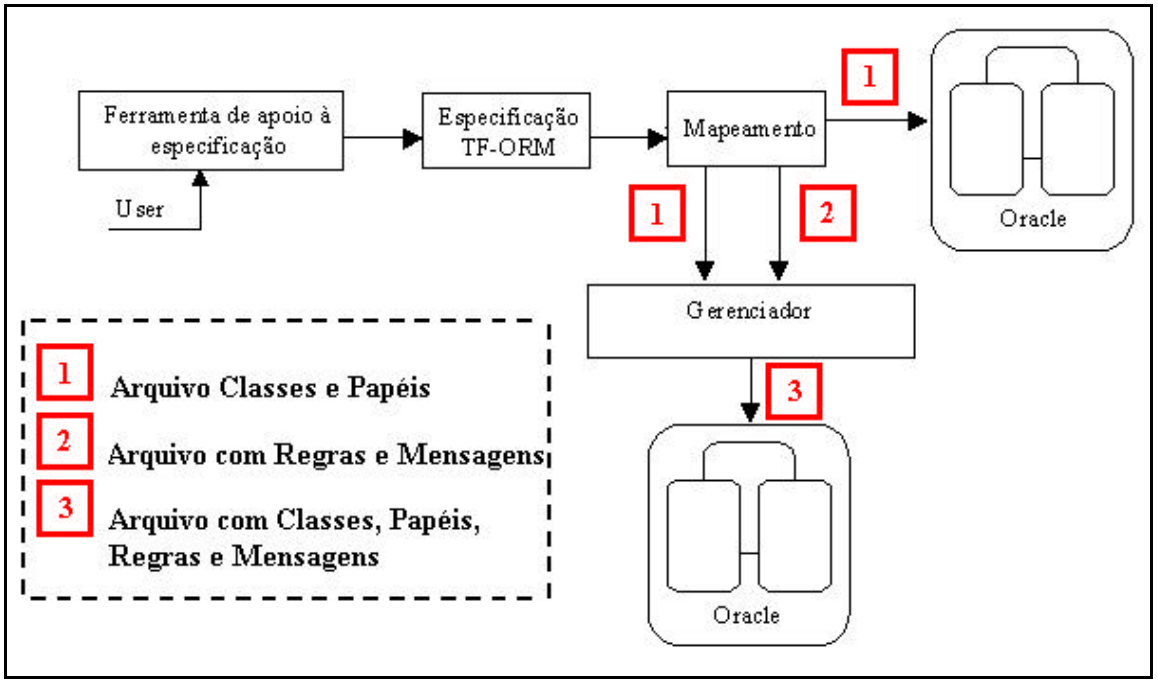



FIGURA 7.1 - Esquema do Funcionamento do Gerenciador

O gerenciador, apresentado na figura 7.1, é o responsável pela realização do mapeamento do modelo TF-ORM para o Oracle. Através de critérios pré-definidos, os quais são apresentados no decorrer deste capítulo, os componentes do modelo TF-ORM são transformados em PL/SQL. Nesta mesma figura é apresentado o funcionamento completo da realidade, na qual está inserida o gerenciador. Conforme apresentado na figura, o gerenciador busca informações de dois arquivos gerados pela ferramenta de apoio à especificação, definida por [DEM 00]. Estas informações são passadas para um interpretador (faz parte do gerenciador), para que as classes e papéis sejam devidamente

identificadas e seja possível gerar arquivos para serem interpretados por um banco de dados relacional.

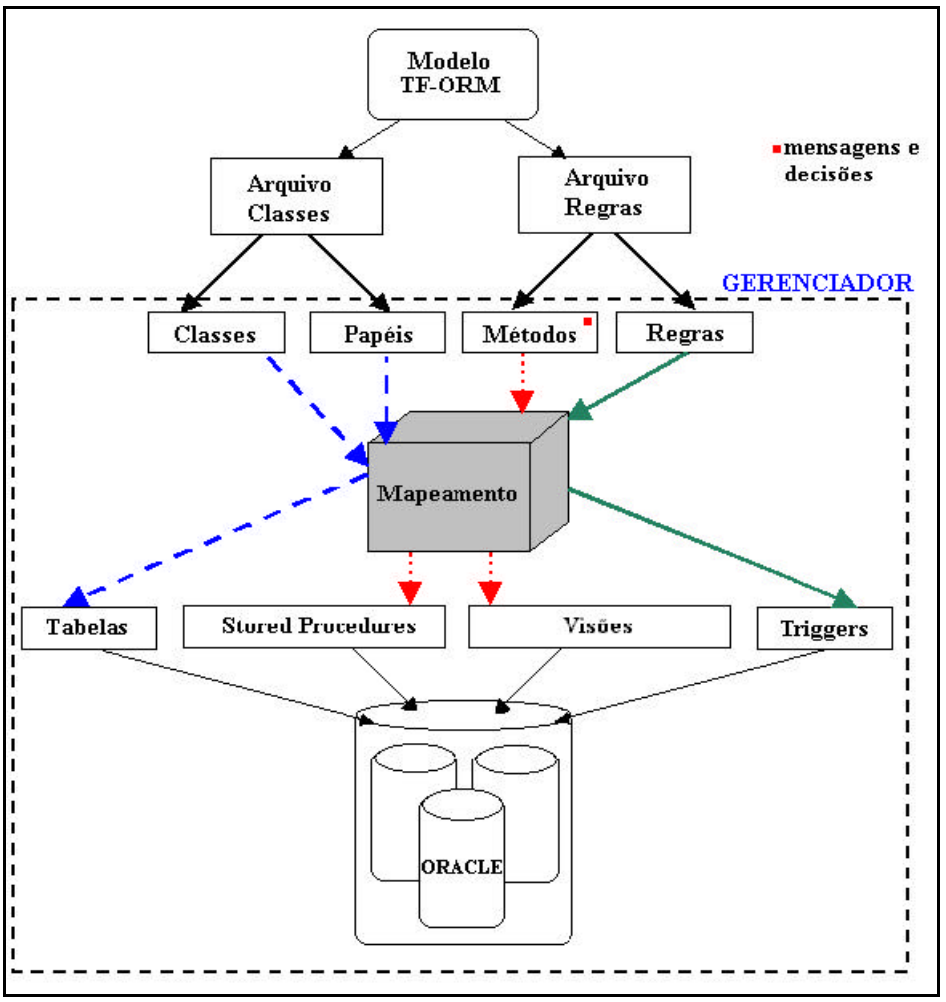



FIGURA 7.2 – Mapeamento dos Componentes do Modelo TF-ORM para o Oracle

O mapeamento do modelo TF-ORM para o SGBD Oracle, é realizado conforme apresenta a figura 7.2: (i) classes e papéis, mapeadas para tabelas; (ii) métodos – mensagens e decisões – mapeados para *stored procedures* (procedimentos) ou *views* (visões); (iii) regras de gerência da base, mapeadas para *triggers*. Todo o mapeamento será detalhado a seguir.

## 7.1 Mapeamento de Classes e Papéis

O mapeamento de classes e papéis segue a mesma estrutura definida por [CAV 95]. Já foi criada uma ferramenta de apoio à especificação para o Oracle [DEM 00]. Esta ferramenta, disponibiliza uma interface para que o usuário crie uma especificação TF-ORM, da qual, são gerados dois arquivos:

1. arquivo com o mapeamento já realizado de classes e papéis em linguagem PL/SQL – linguagem do Oracle;

2. arquivo com as mensagens, decisões e regras do modelo TF-ORM, as quais serão mapeadas para o Oracle através do gerenciador criado.

O arquivo com classes e papéis, apesar de já estar mapeado e pronto para a implementação, será utilizado pelo interpretador, para a criação de mensagens, regras e decisões.

A seguir, será apresentada a forma como as classes e os papéis são implementados.

As classes e papéis são mapeados, um para cada tabela, com seus atributos próprios. As propriedades estáticas de uma classe ou papel são todas mapeadas para uma única tabela denominada ***BASE***, conforme apresenta figura 7.3.

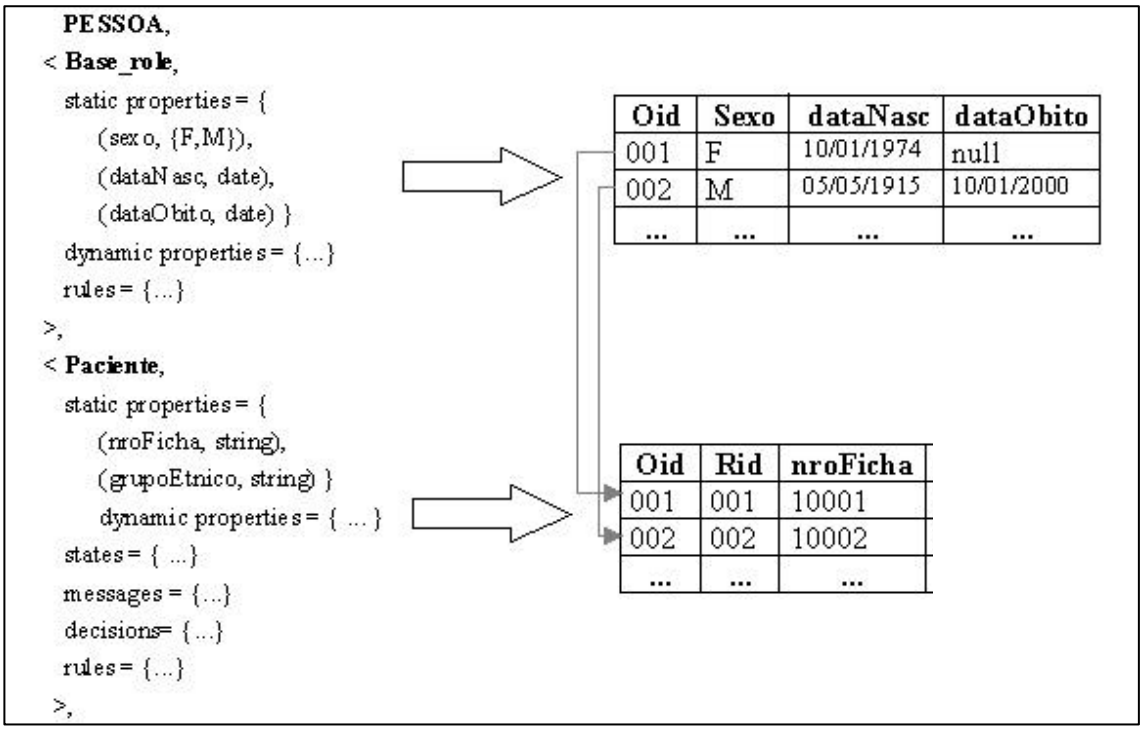



FIGURA 7.3 - Criação das Tabelas Base de Classes e Papéis

Em uma tabela base estão todos os atributos que não variam com o passar do tempo, incluindo-se o identificador da classe ou papel. Por exemplo, na figura 7.3, existem duas tabelas:

- a tabela base de *Pessoa*, com seus atributos estáticos: Oid, Sexo, dataNasc e dataObito, onde Oid é o identificador (único) da tabela (classe no TF-ORM);
- a tabela base de *Paciente*, com seus atributos estáticos: Oid, Rid e nroFicha. Esta tabela, sendo um papel de Pessoa, possui o identificador desta classe (Oid), conforme indicam as setas da figura, fazendo referência à tabela origem (chave estrangeira) e Rid, sendo o identificador (único) da tabela (papel no TF-ORM).

Pode-se afirmar, então, que a "união" entre classes e papéis, quando transformados em tabelas, é realizada através da inclusão do *OID* (identificador) da classe na tabela base do papel (que possui as propriedades estáticas).

Cada propriedade dinâmica será mapeada para uma tabela em separado, a qual conterá o próprio atributo dinâmico, o identificador da classe ou papel ao qual pertence e os atributos de tempo. Exemplos que serão apresentados a seguir, utilizam-se de propriedades dinâmicas definidas no estudo de caso, como é o caso de *nome*, *profissão*, *educação*, etc..

No caso desta implementação, os atributos temporais são: tempo de validade inicial (v_timei), tempo de validade final (v_timef), tempo de transação inicial (t_timei) e tempo de transação final (t_timef). O modelo TF-ORM utiliza pontos no tempo. Entretanto, a implementação utiliza intervalos para que as consultas feitas sobre a base de dados sejam simplificadas. A figura 7.4 apresenta um exemplo do mapeamento de atributos dinâmicos para tabelas.

É importante ressaltar que os estados de classes e papéis também foram implementados como atributos dinâmicos, uma vez que possuem seu valor alterado com o passar do tempo. Cada classe e cada papel possui seu próprio conjunto de estados. Desta forma, foram criadas tabelas para cada um deles. No caso de estados de papéis, os mesmos possuem o identificador *Rid*, tratando-se de classes, possuem o identificador *Oid*, conforme apresentado na figura 7.4.

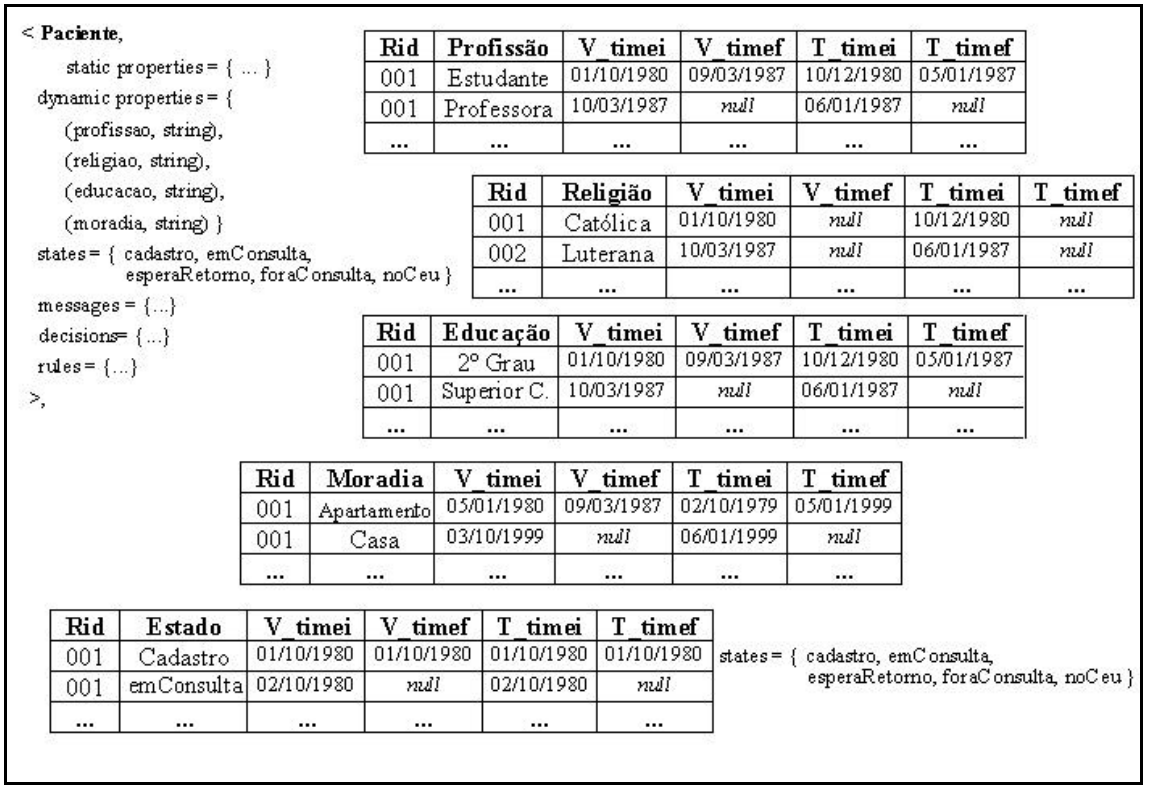



FIGURA 7.4 - Mapeamento dos Atributos Dinâmicos de Classes ou Papéis

Observa-se que as tabelas de atributos dinâmicos, criadas na figura 7.4, referem-se a papéis, pois possuem o identificador Rid. Caso fossem criadas sobre classes, possuiriam o identificador: Oid. A comunicação destes papéis com suas respectivas classes é dada pela existência do identificador Oid na tabela ***BASE***.

Cada uma destas tabelas dinâmicas: *Profissão*, *Religião*, *Educação*, *Moradia* e *Estado*, refere-se a uma única tabela base: *Paciente*, sendo que, no banco de dados,

serão chamadas de: ***Paciente_profissão***, ***Paciente_religião***, ***Paciente_educação***, ***Paciente_moradia*** e ***Paciente_estado***. Sua respectiva tabela base foi apresentada na figura 7.3.

Ao se criar as tabelas originadas de classes e papéis, deve-se tomar o cuidado de utilizar nomes coerentes para as mesmas, pois elas serão utilizadas para identificar os atributos componentes de uma mensagem, regra ou decisão . Ou seja:

- nomear as tabelas bases (com atributos estáticos) com o mesmo nome das classe e papéis;
- nomear os atributos dinâmicos com os nomes das classes e papéis, seguidos de *under line* (_) e o nome do atributo dinâmico. Por exemplo: Pessoa_Nome.

Os identificadores (Oid e Rid) das tabelas são controlados por restrições próprias do banco, sendo definidos como chaves primária e estrangeira em seus devidos casos. Como, para os estados, quando da definição do modelo, já se sabe quais seus possíveis valores, os mesmos são definidos sobre a forma de restrições no Oracle. Tais restrições são chamadas de *Check Constraints*, fazendo com que o atributo estado possua somente os valores pré-definidos durante a implementação.

Os objetos, quando criados, possuem um determinado tempo de vida, definidos da mesma forma que os atributos dinâmicos. Sendo assim, tabelas adicionais são criadas para conter a história de cada objeto na base de dados, tendo a estrutura apresentada pela tabela 7.1. Esta tabela é denominada *NomeClasse/Papel*_DYNAMIC. Esta história é atualizada através das propriedades pré-definidas: *class_instance* e *role_instance*.

TABELA 7.1 - Tabela de Histórias dos Objetos da Tabela Pessoa_nome

| ***Oid*** | ***t_timei*** | ***t_timef*** | ***v_timei*** | ***v_timef*** |
|---|---|---|---|---|
| 01 | 01/10/1999 | *null* | 01/10/1999 | *null* |
| 02 | 01/10/1999 | 09/01/2000 | 01/10/1999 | 09/01/2000 |
| 03 | 10/10/1999 | *null* | 10/10/1999 | *null* |
| ... | ... | ... | ... | ... |

Para as classes, o identificador utilizado na tabela representada pela figura 7.1 é o OID, enquanto que para os papéis, o identificador é o RID.

## 7.2 Implementação de Classes e Papéis

A estrutura básica para a criação de classes e papéis foi definida no item anterior. A criação de uma tabela base, ou seja, da tabela que conterá os atributos estáticos de uma classe, foi realizada da seguinte forma no Oracle:

```
CREATE TABLE <Tabela base> (
Atributo          tipo
Atributo          tipo
Restrições de Integridade);
```

Por exemplo, a criação da classe PESSOA apresentada no estudo de caso é descrita a seguir:

```
CREATE TABLE pessoa(
```

```
oid             number(3)     constraint pk_oid_pessoa PRIMARY KEY,
sexo            varchar2(2),
dataNasc        date,
dataObito       date);
```

Algumas restrições de integridade são mantidas através de mecanismos do banco. No exemplo anterior, criou-se uma chave primária para o atributo OID, onde o nome da *constraint*, referente à chave primária, foi implementado seguindo a seguinte norma, baseando-se no exemplo anterior:

- **pk**: indicando primary key;
- **fk**: indicando foreign key;
- **oid/rid**: campo chave da tabela;
- **pessoa**: classe ou papel base à qual pertence o atributo ou atributo dinâmico gerador de uma tabela.

A tabela 7.2 exemplifica uma tabela dinâmica com valores, bem como ilustra o preenchimento da base com os valores referentes ao tempo de transação e ao tempo de validade.

TABELA 7.2 – Tabela Pessoa_Nome

| ***Oid*** | ***Nome*** | ***T_timei*** | ***t_timef*** | ***v_timei*** | ***v_timef*** |
|---|---|---|---|---|---|
| 001 | Maria Silva | 03/01/1996 | 22/07/1998 | 03/01/1996 | 29/08/1998 |
| 002 | João | 05/12/1997 | *null* | 10/12/1997 | *null* |
| 001 | Maria Souza | 23/07/1998 | *null* | 30/08/1998 | *null* |

Os tempos de transação das informações são fornecidos pelo SGBD quando da inclusão da tupla na base de dados. O tempo de transação final da pessoa de *Oid* 002, cujo nome é João, somente será fornecido caso uma nova tupla para esta pessoa, referente ao atributo dinâmico nome for inserida, cujo valor será o tempo de transação inicial da tupla inserida, menos uma unidade da granularidade temporal utilizada.

É importante salientar que o tempo de validade inicial das tuplas é fornecido pelo usuário quando da inclusão, como característica dos bancos de dados de tempo de validade. Entretanto, para que esta operação torne-se mais transparente para o usuário, caso ele não forneça a validade inicial, esta deve ser considerada como sendo a data atual.

O tempo de validade final será fornecido quando uma nova tupla for inserida pelo usuário, sendo que o mesmo receberá o tempo de validade inicial da nova tupla inserida diminuída de 1 unidade da granularidade utilizada. A granularidade utilizada neste trabalho será a de 30 minutos, por representar o estudo de caso realizado (marcação de consultas).

O comando para criar as tabelas dinâmicas é apresentado a seguir:

```
CREATE TABLE <atributo dinâmico> (
Atributo id          tipo,
Atributo din         tipo,
V_timei              date,
V_timef              date,
T_timei              date,
```

```
T_timef          date);
```

A criação na base de dados de um atributo dinâmico é feito da seguinte forma:

```
CREATE TABLE pessoa_nome(
oid              number(3),
nome             varchar2(30),
v_timei          date,
v_timef          date,
t_timei          date,
t_timef          date,
constraint fk_oid_pessoa_nome FOREIGN KEY(oid) references pessoa,
constraint pk_pessoa_nome PRIMARY KEY(oid,v_timei,t_timei));
```

Para a criação de um papel utilizou-se a seguinte definição:.

```
CREATE TABLE paciente(
oid              number(3),
rid              number(3),
nroFicha         number(5),
constraint fk_oid_paciente FOREIGN KEY(oid) references pessoa,
constraint pk_rid_paciente PRIMARY KEY(rid));
```

O estado de um papel é criado da seguinte forma:

```
CREATE TABLE paciente_states(
rid         number(5),
state       varchar2(2) constraint ck_state_pac CHECK(state in('CA', 'EC', 'ER', 'FC',
            'NC')),
v_timei     date,
v_timef     date,
t_timei     date,
t_timef     date,
constraint fk_rid_paciente_estado FOREIGN KEY(rid) references paciente,
constraint pk_paciente_estado PRIMARY KEY(rid,v_timei,t_timei));
```

É importante ressaltar que as chaves primárias de tabelas dinâmicas são criadas a partir do identificador da classe (Oid) ou do papel (Rid), somando-se ao tempo de validade inicial (v_timei) e ao tempo de transação inicial (t_timei), pois subentende-se que duas informações não serão válidas ao mesmo tempo. Entretanto, caso isto ocorra, não serão inseridas ao mesmo tempo.

A partir do estudo apresentado no capítulo 4, decidiu-se implementar gatilhos que permitissem a atualização e exclusão de informações em qualquer instante do tempo. Ou seja, no caso de atualizações, devem ser geradas novas inserções. Desta forma, o gatilho não deve permitir o comando de *Update* e, sim, comandos de inserção sobre informações com o mesmo atributo dinâmico. No caso de exclusões, o gatilho não deve permitir a exclusão física, o comando *Delete* e, sim, a exclusão lógica, ou seja, encerrar a validade da tupla que se deseja excluir.

## 7.3 Mapeamento de Mensagens e Decisões

As mensagens e decisões são as ativadoras das regras de transição do modelo TF-ORM. Foram implementadas através da utilização dos comandos *stored procedures* (procedimentos) e *views* (visões).

Os procedimentos foram utilizados para implementar mensagens e decisões que inserem, alteram ou excluem informações da base de dados.

As visões foram utilizadas para implementar mensagens e decisões que selecionam informações da base de dados. Foram escolhidas por oferecerem flexibilidade ao usuário quando o mesmo necessitar restringir o escopo da consulta. A forma de implementação manual destas mensagens e decisões é apresentada no item 7.3.1. Uma proposta de implementação automática é apresentada no item 7.3.2.

O modelo TF-ORM pode manipular diferentes classes e/ou papéis em uma mesma mensagem. Desta forma, surgiu a necessidade de definir um interpretador para as mesmas, o que será apresentado a seguir.

### 7.3.1 Mapeamento Manual de Mensagens e Decisões

Inicialmente, o mapeamento da modelagem TF-ORM para o Oracle foi realizado de forma manual, com o objetivo de se encontrar uma forma de otimização para o mesmo. Manualmente, este mapeamento é realizado analisando as mensagens TF-ORM, interpretando de quais tabelas os atributos envolvidos na mensagem fazem parte e criando arquivos – *scripts* através de PL/SQL.

Por exemplo, a mensagem:

valores_iniciais (NroFicha:string, Sexo:{F,M}, DataNasc:date, Nome:string, End: string, Cidade: string, Uf: string, Fone: string, Fax: string, Profissao: string, Religiao: string, Educacao: string, Moradia: string) from CONTROLE.PacienteCtrl,

foi implementada manualmente, seguindo-se as etapas:

1. interpretação da mensagem, buscando encontrar o significado da mensagem: (i) inserção de valores aos atributos; (ii) atualização de atributos; (iii) exclusão de valores; ou (iv) seleção de valores;

2. interpretação dos atributos, identificando a quais tabelas pertencem determinados atributos;

3. identificação de atributos estáticos e dinâmicos, associando todos os atributos estáticos com uma mesma tabela base e todos os atributos dinâmicos com tabelas separadas.

Na figura 7.5, encontra-se a representação do processo manual do mapeamento de mensagens, no qual a análise é feita através do conhecimento prévio e da interpretação da mensagem como um todo.

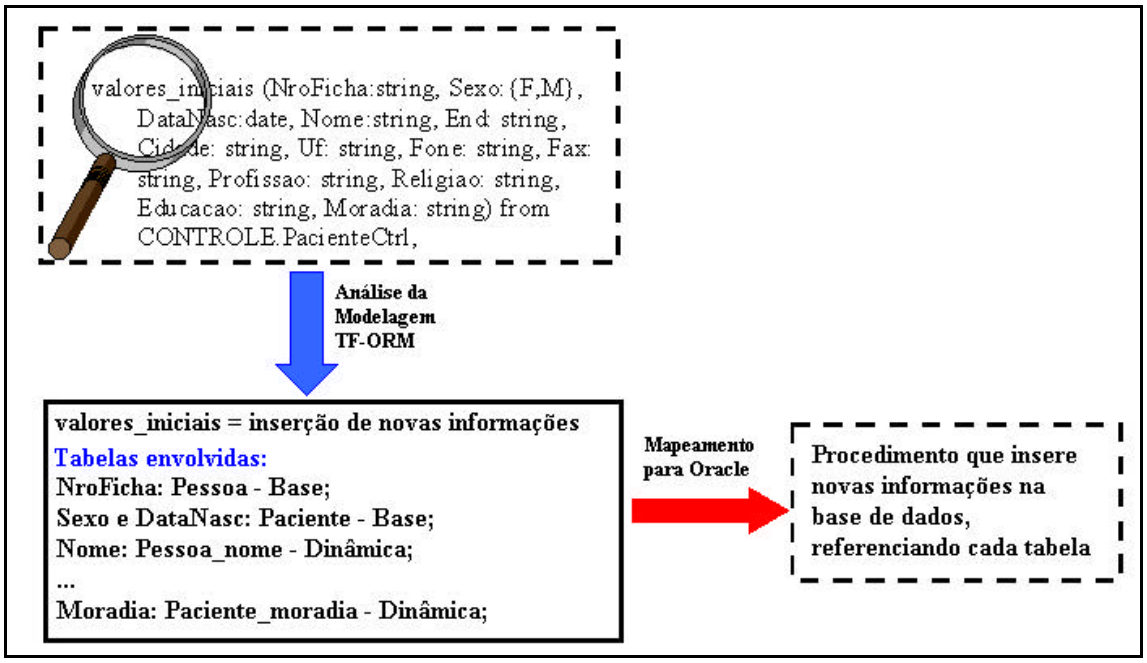



FIGURA 7.5 – Mapeamento Manual de Mensagens

### 7.3.2 Mapeamento Automático de Mensagens e Decisões

Conforme apresentado no item anterior, a análise "visual" da mensagem é possível através da interpretação do seu significado. Entretanto, para que esta análise seja realizada por um sistema de computador, não se procede da mesma forma, uma vez que este sistema não possuirá o conhecimento suficiente.

Os estudos comprovaram que o mapeamento automático não é possível para qualquer tipo de mensagem. Portanto, para que uma mensagem possa ser automaticamente implementada por um gerenciador, primeiro se definiu um padrão para a sua declaração, apresentado a seguir e, também, se definiu um interpretador, o qual tem o objetivo de agilizar a busca de informações quando do mapeamento e implementação de regras do modelo TF-ORM no Oracle. Para isto, deve-se armazenar dados referentes aos atributos estáticos e dinâmicos das classes e papéis, os quais são: nome do atributo, tipo do atributo (se estático ou dinâmico) e classe/papel origem. O interpretador une os atributos de uma mensagem à sua respectiva classe ou papel. Ele busca informações do arquivo de regras, cujo fluxo foi apresentado na figura 7.1, para obter as informações necessárias sobre cada mensagem.

O arquivo contendo as regras é gerado a partir da modelagem original do TF-ORM, sendo separado da mesma quando da definição de tabelas. Não possui as declarações de atributos estáticos e dinâmicos, uma vez que estes são armazenados em um arquivo a parte (definição de tabelas), já podendo ser mapeado para o Oracle. Exemplo de parte deste arquivo é encontrado na figura 7.6, onde pode-se constatar a existência do nome da classe/papel e suas respectivas mensagens, regras e decisões.

Desta forma, o interpretador busca as informações no arquivo de regras, com o objetivo de compará-lo com as informações coletadas sobre os atributos dinâmicos e estáticos. Estas informações são armazenadas sobre a forma de uma tabela,

possibilitando, após a comparação com o arquivo de regras, a criação das mensagens através de procedimentos ou visões na base de dados.

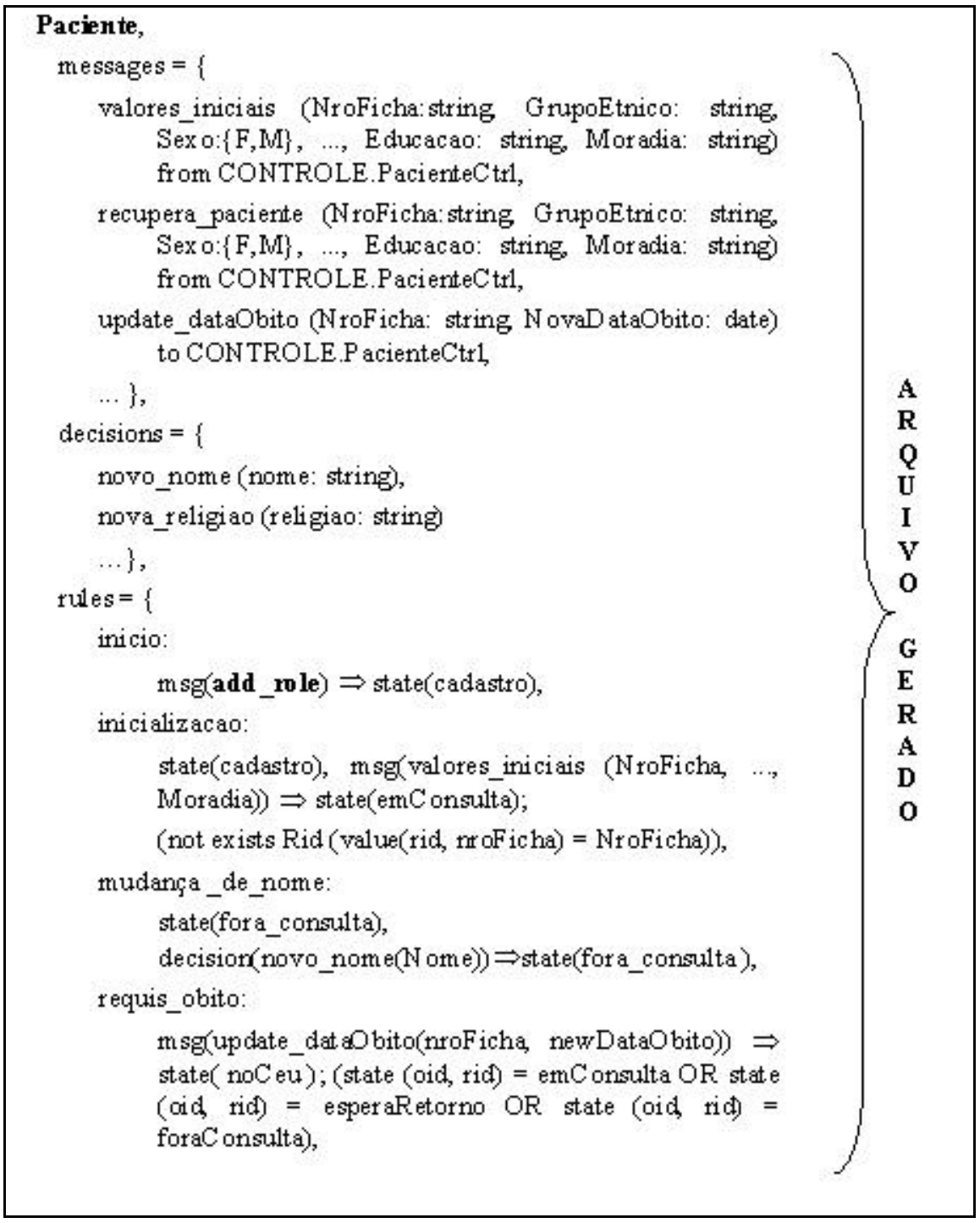
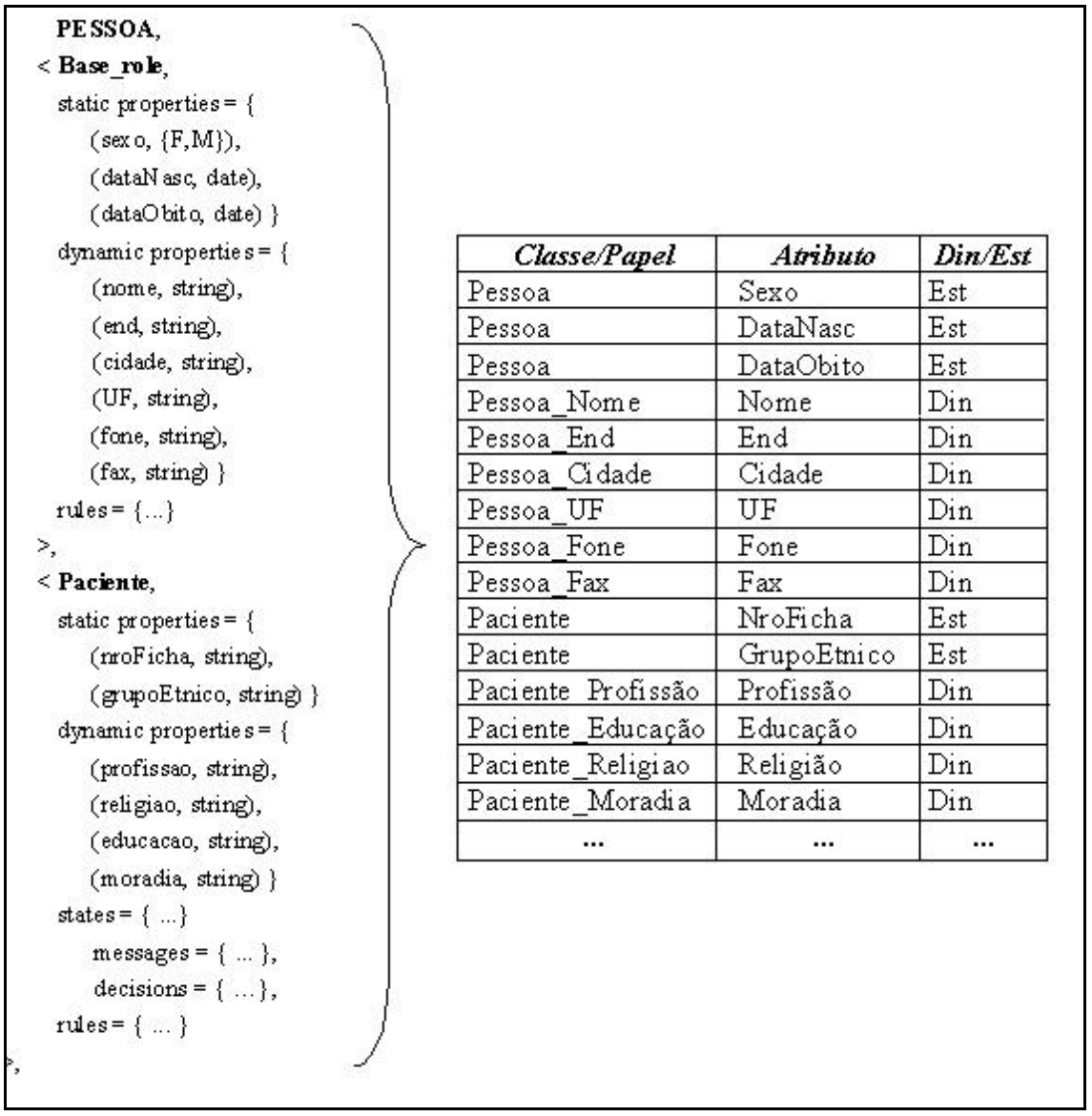



FIGURA 7.6 - Arquivo Gerado pela Ferramenta de Apoio à Especificação

A tabela criada pelo identificador com as informações dos atributos estáticos e dinâmicos é apresentada na figura 7.7.




| *Classe/Papel* | *Atributo* | *Din/Est* |
|---|---|---|
| Pessoa | Sexo | Est |
| Pessoa | DataNasc | Est |
| Pessoa | DataObito | Est |
| Pessoa_Nome | Nome | Din |
| Pessoa_End | End | Din |
| Pessoa_Cidade | Cidade | Din |
| Pessoa_UF | UF | Din |
| Pessoa_Fone | Fone | Din |
| Pessoa_Fax | Fax | Din |
| Paciente | NroFicha | Est |
| Paciente | GrupoEtnico | Est |
| Paciente_Profissão | Profissão | Din |
| Paciente_Educação | Educação | Din |
| Paciente_Religiao | Religião | Din |
| Paciente_Moradia | Moradia | Din |
| ... | ... | ... |

FIGURA 7.7 – Tabela com Atributos Estáticos e Dinâmicos

Por exemplo, considere a mensagem a seguir, definida para o estudo de caso proposto:

> valores_iniciais (NroFicha:string, Sexo:{F,M}, DataNasc:date, Nome:string, End: string, Cidade: string, Uf: string, Fone: string, Fax: string, Profissao: string, Religiao: string, Educacao: string, Moradia: string) from CONTROLE.PacienteCtrl,

O mapeamento automático desta mensagem seria praticamente impossível, pois um gerenciador não saberia qual a ação a ser executada, bem como quais as classes a serem manipuladas. Para tanto, o gerenciador busca as informações necessárias na tabela apresentada na figura 7.7.

Sabendo que esta mensagem pertence ao papel ***Paciente***, bem como quais são os atributos estáticos e dinâmicos, pois existe variação na nomenclatura das tabelas, fica simples buscar seus atributos para criar o procedimento no banco, pois a variação será nas tabelas ***Pessoa***, ***Paciente*** e nos seus respectivos atributos dinâmicos.

O comando utilizado no Oracle para criar os procedimentos é apresentado a seguir:

```
CREATE PROCEDURE<NOME> (PARÂMETROS)
AS BEGIN
UPDATE / INSERT...
SET...
WHERE ...
END;
```

Considerando ainda a mesma mensagem apresentada anteriormente, sua implementação no Oracle é realizada conforme apresenta a figura 7.8.

```
create procedure add_paciente(Oid number,Rid number, NroFicha number,Sexo varchar2,
Nome varchar2,Ende varchar2, Cidade Varchar2, UF varchar2,Fone varchar2, Fax varchar2,
DataNasc date,Profissao varchar2, Religiao varchar2, Educacao varchar2,Moradia varchar2,
Val_ini date)
as begin
    insert into pessoa
    values(Oid,Sexo,DataNasc,NULL);
    insert into paciente
    values(Oid,Rid,NroFicha);
    if(Val_ini is NULL)
        then insert into pessoa_nome
            values(Oid,Nome,SYSDATE,NULL,SYSDATE,NULL);
            insert into pessoa_end
            values(Oid,Ende,Val_ini,NULL,SYSDATE,NULL);
            insert into pessoa_cidade
            values(Oid,Cidade,Val_ini,NULL,SYSDATE,NULL);
            insert into pessoa_uf
            values(Oid,UF,Val_ini,NULL,SYSDATE,NULL);
            insert into pessoa_fone
            values(Oid,Fone,Val_ini,NULL,SYSDATE,NULL);
            insert into pessoa_fax
            values(Oid,Fax,Val_ini,NULL,SYSDATE,NULL);
            insert into paciente_profissao
            values(Rid,Profissao,Val_ini,NULL,SYSDATE,NULL);
            insert into paciente_religiao
            values(Rid,Religiao,Val_ini,NULL,SYSDATE,NULL);
            insert into paciente_educacao
            values(Rid,Educacao,Val_ini,NULL,SYSDATE,NULL);
            insert into paciente_moradia
            values(Rid,Moradia,Val_ini,NULL,SYSDATE,NULL);
            insert into paciente_states
            values(Rid,'AC',Val_ini,NULL,SYSDATE,NULL);
        END IF;
    if(Val_ini is NOT NULL)
        then insert into pessoa_nome
            values(Oid,Nome,to_date(Val_ini,'DD-MM-YY HH:MI'),NULL,SYSDATE,NULL);
            insert into pessoa_end
            values(Oid,Ende,to_date(Val_ini,'DD-MM-YY HH:MI'),NULL,SYSDATE,NULL);
            insert into pessoa_cidade
            values(Oid,Cidade,to_date(Val_ini,'DD-MM-YY HH:MI'),NULL,SYSDATE,NULL);
            insert into pessoa_uf
            values(Oid,UF,to_date(Val_ini,'DD-MM-YY HH:MI'),NULL,SYSDATE,NULL);
            insert into pessoa_fone
            values(Oid,Fone,to_date(Val_ini,'DD-MM-YY HH:MI'),NULL,SYSDATE,NULL);
            insert into pessoa_fax
            values(Oid,Fax,to_date(Val_ini,'DD-MM-YY HH:MI'),NULL,SYSDATE,NULL);
            insert into paciente_profissao
            values(Oid,Profissao,to_date(Val_ini,'DD-MM-YY HH:MI'),NULL,SYSDATE,NULL);
            insert into paciente_religiao
            values(Oid,Religiao,to_date(Val_ini,'DD-MM-YY HH:MI'),NULL,SYSDATE,NULL);
            insert into paciente_educacao
            values(Oid,Educacao,to_date(Val_ini,'DD-MM-YY HH:MI'),NULL,SYSDATE,NULL);
            insert into paciente_moradia
            values(Oid,Moradia,to_date(Val_ini,'DD-MM-YY HH:MI'),NULL,SYSDATE,NULL);
            insert into paciente_states
            values(Rid,'AC',Val_ini,NULL,SYSDATE,NULL);
        END IF;
end;
```

FIGURA 7.8 - Criação de Mensagem no Oracle

Visualizando a figura 7.8 surge a pergunta: como saber quais operações devem ser realizadas com os atributos da mensagem? Uma forma para solucionar este problema é a definição de um padrão na nomenclatura das mensagens, ou seja:

- para mensagens que insiram informações na base de dados o nome das mesmas deve vir precedido da palavra: **INSERT**;

  Por exemplo, a regra *valores_iniciais*, ficaria:

  ***Insert**_valores_iniciais (NroFicha:string, Sexo:{F,M}, DataNasc:date, Nome:string, End: string, Cidade: string, Uf: string, Fone: string, Fax: string, Profissao: string, Religiao: string, Educacao: string, Moradia: string) from CONTROLE.PacienteCtrl,*

- mensagens que atualizem a base de dados, o nome das mesmas deve vir precedido da palavra: **UPDATE**;

  Por exemplo, a regra *atualiza_dataObito*, ficaria:

  ***Update**_atualiza_dataObito (NroFicha: string, NovaDataObito: date) from PESSOA.Paciente ,*

- mensagens que excluam informações da base de dados, o nome das mesmas deve vir precedido da palavra: **DELETE**;

  Por exemplo, a regra *exclui_paciente*, ficaria:

  ***Delete**_exclui_paciente (NroFicha: string) from PESSOA.Paciente ,*

- mensagens que recuperem informações da base de dados, o nome das mesmas deve vir precedido da palavra: **SELECT**.

  Por exemplo, a regra *recupera_valores*, ficaria:

  ***Select**_recupera_valores (NroFicha:string, Sexo:{F,M}, DataNasc:date, Nome:string, End: string, Cidade: string, Uf: string, Fone: string, Fax: string, Profissao: string, Religiao: string, Educacao: string, Moradia: string) to PESSOA.Paciente,*

As mensagens que selecionam informações da base de dados são implementadas utilizando o conceito de visões do Oracle, uma vez que as mesmas oferecem maior flexibilidade ao usuário.

A figura 7.9 apresenta a implementação de uma mensagem que recupera informações no Oracle. A visão apresentada nesta figura, foi criada a partir da seguinte mensagem:

***recupera_valores** (NroFicha:string, Sexo:{F,M}, DataNasc:date, Nome:string, End: string, Cidade: string, Uf: string, Fone: string, Fax: string, Profissao: string, Religiao: string, Educacao: string, Moradia: string) to PESSOA.Paciente,*

```
CREATE VIEW recupera_paciente(Oid, rid, nroFicha, sexo, dataNasc, nome,
ende, cidade, uf, fone, fax, profissao, religiao, educacao, moradia, v_timei, v_timef,
t_timei, t_timef)

AS SELECT p.oid, pa.rid, pa.nroFicha, p.sexo, p.dataNasc, pn.nome, pe.ende,
pc.cidade, pu.uf, pf.fone, pfa.fax, pp.profissao, pr.religiao, ped.educacao,
pm.moradia,

  GREATEST(pn.v_timei, pe.v_timei, pc.v_timei, pu.v_timei, pf.v_timei,
pfa.v_timei, pp.v_timei, pr.v_timei, ped.v_timei, pm.v_timei),

  LEAST(pn.v_timef, pe.v_timef, pc.v_timef, pu.v_timef, pf.v_timef, pfa.v_timef,
pp.v_timef, pr.v_timef, ped.v_timef, pm.v_timef),

  GREATEST(pn.t_timei, pe.t_timei, pc.t_timei, pu.t_timei, pf.t_timei, pfa.t_timei,
pp.t_timei, pr.t_timei, ped.t_timei, pm.t_timei),

  LEAST(pn.t_timef, pe.t_timef, pc.t_timef, pu.t_timef, pf.t_timef, pfa.t_timef,
pp.t_timef, pr.t_timef, ped.t_timef, pm.t_timef)

FROM pessoa p, paciente pa, pessoa_nome pn, pessoa_end pe, pessoa_cidade
pc, pessoa_uf pu, pessoa_fone pf, pessoa_fax pfa, paciente_profissao pp,
paciente_religiao pr, paciente_educacao ped, paciente_moradia pm

WHERE p.Oid = pa.oid AND
      p.Oid = pn.oid AND
      p.Oid = pe.oid AND
      p.Oid = pu.oid AND
      p.Oid = pf.oid AND
      p.Oid = pfa.oid AND
      pa.Rid = pp.Rid AND
      pa.Rid = pr.Rid AND
      pa.Rid = ped.Rid AND
      pa.Rid = pm.Rid
;
```

FIGURA 7.9 - Visão no Oracle

Existem informações adicionais que foram inseridas à visão e que não estão presentes na mensagem original, como por exemplo, os tempos de validade e transação das informações. Para a recuperação destas informações, foram utilizados os operadores do Oracle GREATEST e LEAST para fornecerem, respectivamente, as informações sobre a maior e a menor data fornecidas dentre um conjunto de datas. Isto possibilita a consulta sobre informações válidas ao mesmo tempo.

Da mesma forma que os procedimentos, para que as visões sejam criadas, deve-se analisar a mensagem original através do identificador, o qual fornecerá os nomes das tabelas às quais os atributos pertencem.

Além das mensagens definidas pelo usuário quando da criação de uma modelagem TF-ORM, existem as do modelo que representam características próprias,

como por exemplo: CREATE_OBJECT, a qual determina a criação da instância de uma classe, neste caso, tabela; END_OBJECT, a qual determina a finalização da existência de uma instância; ADD_ROLE, a qual determina a criação da instância de um papel; TERMINATE_ROLE, a qual determina a finalização da existência de um papel que, da mesma forma que as demais, devem ser automaticamente criadas para cada tabela pelo gerenciador. Estas mensagens atualizam a tabela de histórias (DYNAMIC) dos objetos quando de sua criação e finalização. Exemplos destes procedimentos são apresentados no capítulo 8.

## 7.4 Mapeamento de Regras

As regras tratadas neste mapeamento compreendem as de transição de estado do modelo TF-ORM. As de integridade do modelo não serão tratadas neste trabalho, pois podem ser representadas pelas regras de transição de estados.

O acionamento dos gatilhos é automático. Entretanto, eles não conseguem representar de forma completa a estrutura das regras de transição de estados, uma vez que são manipuladas várias tabelas ao mesmo tempo. Desta forma, optou-se pela sua utilização nas regras de manutenção da base de dados, sobre tabelas separadas, ou seja, naqueles casos onde se deseja manter a integridade da base temporal, seguindo-se as regras apresentadas no capítulo 4 e, mais especificamente, aquelas que permitem a atualização e remoção lógica sobre toda a base de dados.

Os procedimentos apresentam uma maior versatilidade para a criação das regras, através da utilização de parâmetros. Entretanto, seu acionamento não é transparente. Mas, pode-se simular o funcionamento das regras de transição do modelo TF-ORM com o auxílio da aplicação a ser desenvolvida.

Quando um procedimento é invocado através de uma declaração do próprio banco, ele utiliza parâmetros para receber e retornar valores desta requisição, podendo chamar outros procedimentos e, inclusive, acionar gatilhos. Estes, por sua vez, são associados a tabelas específicas e acionados automaticamente quando tuplas, associadas às mesmas, são inseridas, atualizadas ou excluídas da base. Eles não possuem parâmetros e não podem ser invocados por uma declaração de "chamada". Podem chamar procedimentos e acionar outros gatilhos.

Suponha a seguinte estrutura para uma regra:

R1: State(S1), M1(s) ➔ State(S2), M2(s); <condição>

Esta regra é implementada da seguinte forma:

- **State(S1)**: é implementado através de um atributo especial que identifica o estado das tuplas. Estes estados são definidos através de uma das restrições do próprio SGBD, as *check constraints.* São implementados como atributos dinâmicos, em uma tabela a parte.
- **M1(s)**: criadas através de procedimentos, quando originárias de operações de inserção, atualização ou exclusão e visões, quando originárias de operações de consulta. As mensagens são responsáveis por disparar outros procedimentos ou

gatilhos, nos quais estão implementadas todas as regras descritas para a consistência do estudo de caso. A implementação dos procedimentos deve ser feita de forma separada, ou seja, deve existir um procedimento por regra escrita na definição do modelo TF-ORM.

- **State(S2)**: é a alteração do atributo de estado. Ou seja, verificando a(s) regra(s): se o atributo está no estado S1, ao ter recebido a(s) mensagem(ns) M1(s), ele passa para o estado S2 e envia a(s) mensagem(ns) M2(s). Para passar para o próximo estado, simplesmente, realiza-se a operação de *update* sobre o atributo desejado.
- **M2(s)**: Estas mensagens são as direcionadas para o usuário. Podem, também, ser visões ou procedimentos que executarão novas operações.

Todas as regras de transição de estados possuem suas mensagens implementadas através de procedimentos. Os gatilhos são utilizados para implementar regras que mantenham a base íntegra.

As operações, assim chamadas no decorrer deste trabalho, nada mais são do que as mensagens que disparam uma determinada regra, possibilitando a atualização de atributos e o mapeamento das regras de transição de estados do modelo TF-ORM.

As regras são criadas por partes. Uma vez que as mensagens já foram definidas pelo gerenciador e que os estados são tabelas em separado, uma por classe, basta unir as mensagens e fazer com que as mesmas tomem a estrutura da regra definida pelo TF-ORM. Isto pode ser realizado de duas formas: criando procedimentos adicionais, os quais contêm a estrutura da regra com as chamadas para os demais procedimentos e verificação do tabela de estados ou realizado pela aplicação, o que é mais coerente, uma vez que se pode verificar quais as mensagens que foram acionadas.

Exemplificando o parágrafo anterior:

1. Regras definidas através de um novo procedimento – estrutura:

*Criando novo procedimento:*

*Chamando procedimentos(mensagens) – um ou vários;*

*Verificando estado das classes;*

*Chamando novos procedimentos – um ou vários;*

*Alterando estados de uma classe;*

Não existe uma liberdade neste tipo de definição, ou seja, uma nova mensagem apenas deve ser enviada, caso o estado correto do objeto esteja definido. Nesta implementação o mapeamento está sendo direcionado ao correto envio das mensagens.

2. Regras definidas através da aplicação – estrutura:

*If Mensagem 1*

*Then if Mensagem 2*

*Else is Estado(S1)*

*Then Aciona mensagem 3 and Troca Estado(S2)*

*End if*

Neste tipo de implementação existe uma maior flexibilidade para o usuário no que diz respeito ao acionamento das mensagens. Entretanto, o desenvolvedor deve criar a estrutura correta das regras na aplicação.

Neste trabalho foi escolhida a segunda opção apresentada, ou seja, deixar que a aplicação determine a seqüência de execução da regra, uma vez que a estrutura da mesma é preservada, bem como, a filosofia das regras do modelo TF-ORM. Exemplos da implementação realizada para o estudo de caso pode ser encontrado no capítulo 8.

### 7.4.1 Mapeamento Manual de Regras de Transição de Estados

Manualmente, este mapeamento é realizado analisando-se as regras TF-ORM, interpretando-as e verificando quais mensagens serão as disparadoras de quais regras, bem como, sobre quais atributos os mesmos interagem.

Conforme definido anteriormente, o acionamento ou não de uma regra será gerado pela aplicação definida para um determinado caso. Desta forma, se consegue utilizar a estrutura correta da regra TF-ORM, bem como representar seus reais objetivos.

Por exemplo, a regra:

**analisa_medico**:

state (ativo), msg (atribui_medico (NroFicha, Especialidade)) ⇒ state (analisandoCaso),

foi implementada da seguinte forma:

1. foram definidos todos os estados possíveis para o atributo *state* da tabela (classe) médico;

2. foi criada a mensagem *atribui_médico*, com parâmetros NroFicha e Especialidade, sabendo que ***NroFicha*** pertence à tabela base e ***Especialidade*** pertence à tabela dinâmica *medico_especialidade*.

Um trecho de programa que aciona esta regra segue a seguinte estrutura:

SE *medico.state* = ativo ENTÃO

SE *msg.atribui_medico = TRUE* ENTÃO

*Medico.state := analisandoCASO;*

FIMSE

### 7.4.2 Mapeamento Automático de Regras de Transição de Estados

O mapeamento automático de regras pode ser utilizado, mas deve-se estar ciente que a estrutura da regra não será seguida, pois o acionamento será feito pelo nome da regra. Por exemplo, a regra com nome ***inicialização*** e estrutura:

> state(cadastro), msg(valores_iniciais (NroFicha, Sexo, DataNasc, Nome, End, Cidade, Uf, Fone, Fax, Profissao, Religiao, Educacao, Moradia)) ⇒ state(emConsulta);
>
> (not exists Rid (value(rid, nroFicha) = NroFicha)),

será acionada seguindo-se as seguintes etapas:

1. definição das atributos de estado para a classe paciente, a qual refere-se esta regra;

2. criação da mensagem *valores_iniciais* com seus respectivos atributos de entrada;

3. criação do procedimento *inicialização* o qual deverá acionar uma função que retorne se, no momento desejado, o atributo *state* é igual a cadastro, acionar o procedimento que foi criado para *valores_iniciais*, atualizar o valor do atributo *state* de paciente para *emConsulta*, além de, somente realizar todas estas operações, caso a condição apresentada seja satisfeita, ou seja, somente executa caso o *NroFicha* que se está inserindo ainda não exista.

Através das etapas apresentadas anteriormente, nota-se a dificuldade em manter o objetivo de uma regra de transição de estados, a qual busca identificar, através de mensagens acionadas, qual regra será aplicada. Não se deve limitar o acionamento de uma regra por uma seqüência pré-definida.

### 7.4.3 Mapeamento Automático de Regras de Gerência

Para todas as tabelas criadas na base de dados, deve-se criar regras que façam o gerenciamento dos dados. Tais regras visam manter a integridade da base temporal, como por exemplo:

- atualização dos tempos de transação inicial das tuplas: atualizados quando uma informação é inserida à base de dados;
- atualização dos tempos de transação final das tuplas: atualizados quando uma nova informação é inserida na base, fazendo com que a informação anterior seja finalizada, uma vez que trabalha-se com pontos no tempo, simulando-se intervalos temporais;
- atualização do tempo de validade inicial de uma tupla: fornecido como a data atual, para ser transparente ao usuário, quando o mesmo não deseja fornecer esta informação;
- atualização do tempo de validade final de uma tupla: para indicar que a informação deixou de ser válida ou que foi excluída da base de dados, neste caso, deve-se, também, atualizar a tabela de informações sobre o objeto.

Estas regras foram criadas através de gatilhos (*triggers)*, uma vez que são acionadas por operações realizadas sobre tabelas específicas.

A seguir é apresentado um exemplo da definição de um gatilho, o qual foi criado para atualizar os atributos *t_timei* (tempo de transação inicial) com a data atual, *t_timef* (tempo de transação final) da tupla inserida anteriormente e *v_timef* (tempo de validade final) da tupla inserida anteriormente, toda vez que se inserir uma informação na tabela *pessoa_nome*, conforme figura 7.10.

```
create trigger insere_pessoa_nome
after
 insert on pessoa_nome
declare
 no number;
 dd date;

begin
  select oid, v_timei
  into no, dd
  from pessoa_nome
  where t_timef is NULL and t_timei = SYSDATE;

  update pessoa_nome
  set v_timef = dd - 1
  where oid = no and v_timef is null and v_timei <> dd;

  update pessoa_nome
  set t_timei = SYSDATE
  where t_timef is NULL;

  update pessoa_nome
  set t_timef = SYSDATE
  WHERE oid = no and t_timef is NULL and v_timei <> dd;

end;
```

FIGURA 7.10 – Definição de um Trigger

## 7.5 Interface do Gerenciador do Mapeamento

Nesta seção é apresentada a estrutura básica de uma interface para o gerenciador, o qual possibilita a busca de dados de uma modelagem TF-ORM e os transforma em informações a serem implementadas pelo Oracle ou por qualquer outro SGBD relacional, sofrendo poucas alterações. A definição do gerenciador, foi apresentada na figura 7.1.

Uma interface para este gerenciador deve conter pelo menos as instruções a seguir (figura 7.11). É apresentada, ainda, uma proposta da forma de implementar as instruções relacionadas:

- *Gera Arquivo Identificadores* – busca informações da modelagem TF-ORM e cria uma tabela (armazenada em arquivo), conforme apresenta a figura 7.7. O nome do arquivo que será gerado pelo identificador será fornecido em *Nome Arquivo de Saída*;

- *Busca Arquivo Mensagens e Regras* –encontra o arquivo gerado pela ferramenta de apoio à especificação, cujo nome deve ser enviado para o campo *Nome Arquivo de Entrada*;
- *Processa Indentificação* – após os dois campos estarem preenchidos, se pressiona este botão, o qual irá disparar um processo para gerar um arquivo intermediário com as informações coletadas dos dois arquivos: de identificadores e de mensagens e regras;
- *Cria Scripts* – geração dos *scripts* propriamente ditos, pressiona-se este botão, fazendo com que um processo interno seja disparado e crie os *scripts* que serão executados no Oracle.

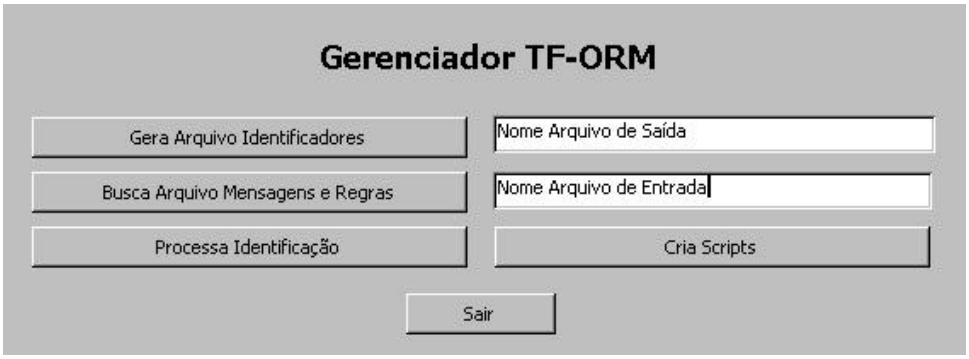



FIGURA 7.11 – Interface Básica para o Gerenciador

## 7.6 Considerações Finais

Neste capítulo foi apresentada a forma como o modelo TF-ORM é mapeado para o Oracle, incluindo o mapeamento de:

- **classes e papéis** para tabelas;
- **mensagens e decisões** para procedimentos e funções, possibilitando uma exata representação destes mecanismos no banco;
- **regras**: os estados são implementados para tabelas e as mensagens utilizadas na sua estrutura são implementadas como procedimentos.

A implementação da estrutura das regras não foi realizada utilizando-se apenas recursos do banco. Se verificou a necessidade de algumas características obtidas, apenas, por programação, como definir a correta estrutura de uma regra.

A utilização apenas dos recursos relacionais do Oracle foi por obter uma maior flexibilidade, podendo-se, com poucas alterações, realizar migrações para outras bases relacionais, uma vez que são as mais utilizadas comercialmente.

Os gatilhos foram utilizados para a criação de regras que mantenham a base de dados consistente (regras de gerência), cuidando para que os tempos de transação e validade inicial e final permaneçam consistentes durante toda a existência do objeto.

O estudo apresentado no capítulo 4, referente à forma de gerenciar as informações em uma base de dados temporal, foi essencial nas decisões tomadas neste capítulo. Este estudo possibilitou a definição dos momentos nos quais as operações de atualização e exclusão na base temporal seriam realizadas.

Portanto, neste capítulo se comprovou a viabilidade da definição e implementação de um mapeamento, do modelo de dados temporal TF-ORM, para um SGBD convencional.

# 8 Implementação no Oracle

Neste capítulo, são apresentados exemplos da implementação do estudo de caso no Oracle. Os exemplos principais estão baseados na criação de tabelas (classes e papéis do TF-ORM), de procedimentos (regras e mensagens do TF-ORM) de visões (mensagens de recuperação de várias informações) de funções (mensagens de recuperação de uma única informação) e de gatilhos (regras para manterem a base de dados consistente)

Todo o mapeamento do TF-ORM para o Oracle foi gerado através de arquivos (*scripts*) que são executados dentro do SGBD para que sejam criadas as tabelas, procedimentos, funções e visões. A criação destes arquivos foi adotada pela facilidade de verificação posterior, de quais comandos foram executados sobre quais atributos, tabelas, etc., fazendo com que uma provável correção de código seja facilitada.

## 8.1 Implementação dos *Scripts*

Este item demonstra como os arquivos foram executados no Oracle, para que o mapeamento possa ser utilizado.

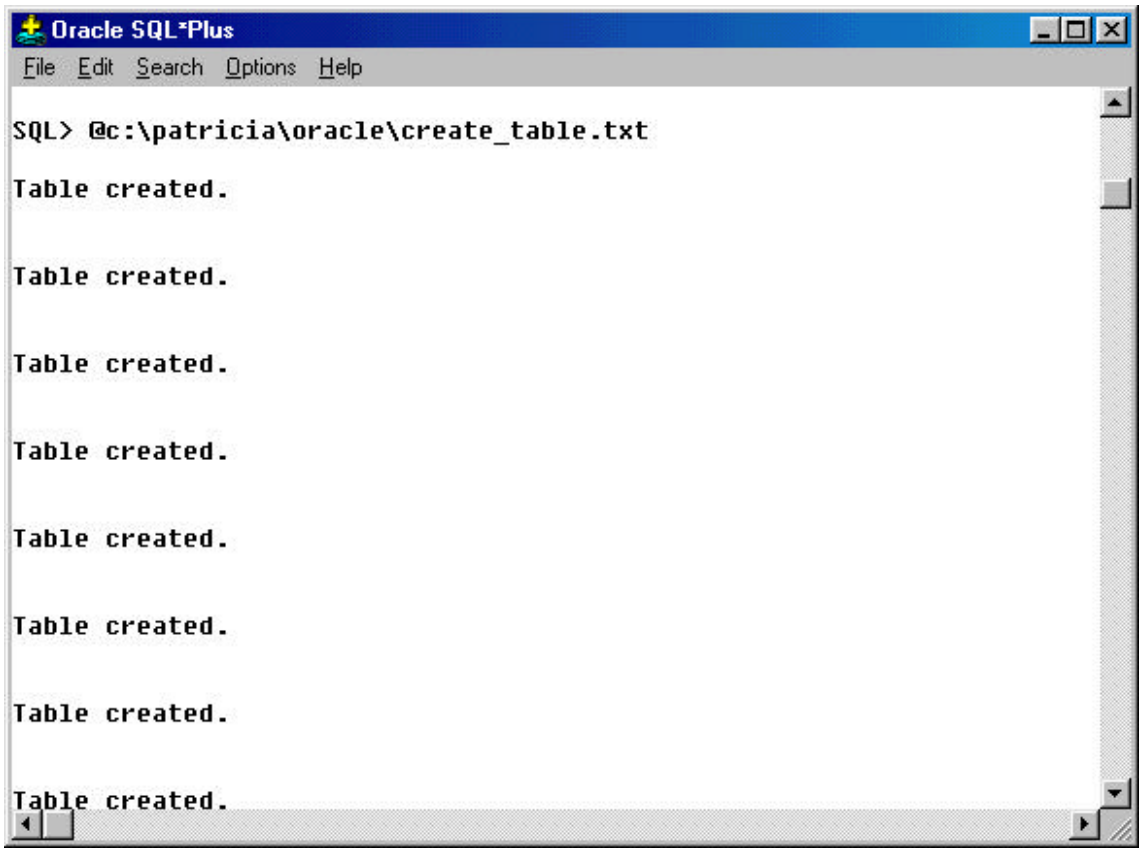



FIGURA 8.1 – Criação de Tabelas no Oracle

A figura 8.1 apresenta a chamada do arquivo que contém todas as definições de tabelas, bem como as mensagens que indicam que as tabelas foram criadas com êxito. Parte do arquivo de definição destas tabelas é apresentado na figura 8.2. Este arquivo é gerado pela ferramenta de apoio à especificação [DEM 00], através da qual foram criadas tabelas para todas as classes e papéis apresentados na modelagem TF-ORM (Anexo 2), bem como para todas as propriedades dinâmicas de cada um destes elementos.

Uma informação importante, contida na figura 8.2, é a visualização da definição dos tempos de validade e transação de estados em algumas tabelas, os quais indicam que estas foram geradas através de propriedades dinâmicas. Além da apresentação da criação de chaves primárias das tabelas dinâmicas (definida como sendo formada pelo identificador da tabela, tempo de validade inicial e tempo de transação inicial) e estrangeiras (referência à tabela "pai").

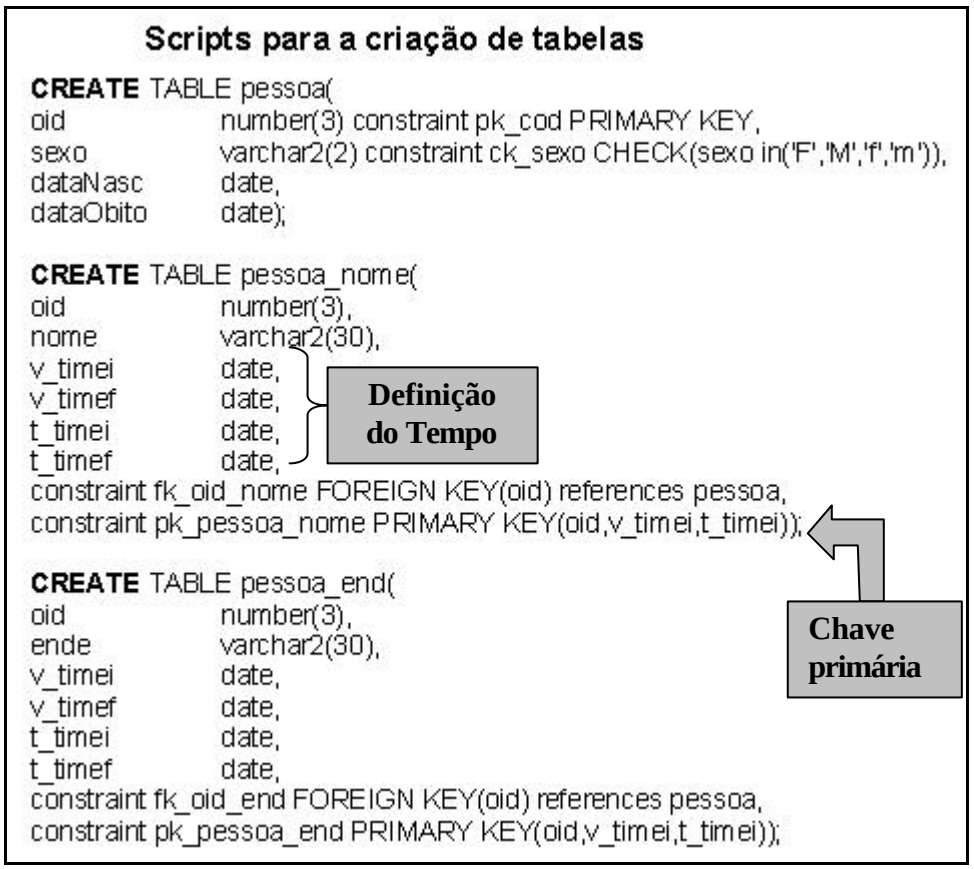



FIGURA 8.2 – Exemplo do Conteúdo do Arquivo de Definição de Tabelas

No caso de procedimentos, os mesmos foram criados levando-se em consideração o tipo de mensagem a ser definida: inserção, atualização e/ou remoção. Deve-se considerar que, toda vez que um procedimento para atualização for definido, ele será criado através de comandos de inserção, como apresenta a figura 8.3. Nesta figura, o apontador 1 apresenta o nome do arquivo gerado com o procedimento em questão e o apontador 2, indica a execução do arquivo, buscando possíveis erros de definição. Tais erros incluem: má definição de parâmetros, tabelas envolvidas e não declaradas, etc.

Quando um procedimento é criado e está livre de erros, a única mensagem retornada é: ***Procedure Created***.

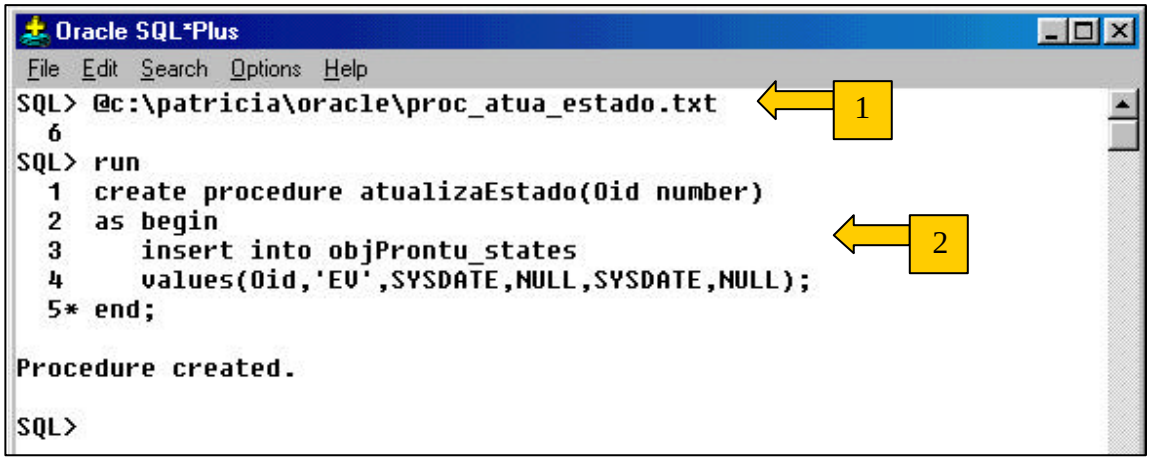



FIGURA 8.3 – Procedimento de Atualização de Valores

A figura 8.4 apresenta a criação de um procedimento que insere novos valores em diversas tabelas. Isto somente é possível através da definição do interpretador, o qual agiliza a busca de informações através da comparação dos dados contidos na mensagem com os atributos estáticos e dinâmicos definidos na modelagem TF-ORM, como componentes de classes e papéis. Manualmente, isto pode ser facilmente identificado. Entretanto, automaticamente, isto não é uma verdade.

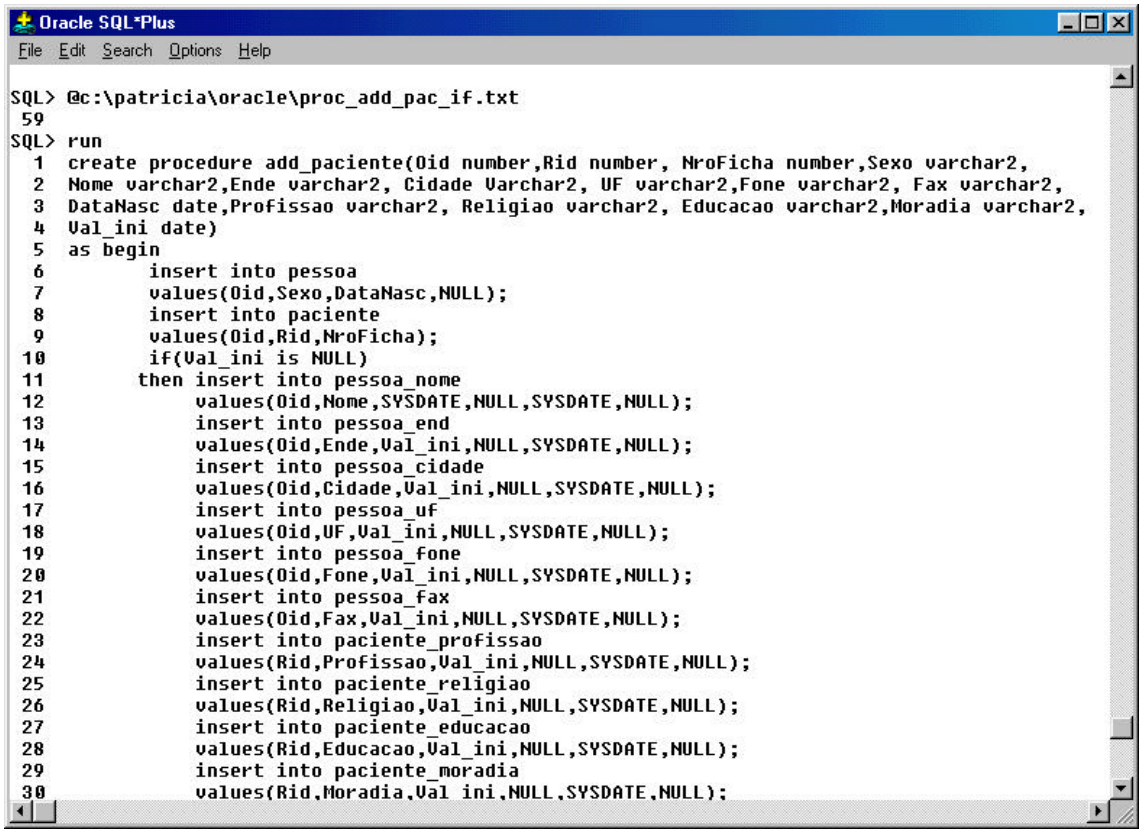



FIGURA 8.4 – Definição do Procedimento de Inserção em Diversas Tabelas

A definição completa deste procedimento foi apresentada na figura 7.8, no capítulo 7.

A inserção, da figura 8.4, é realizada em 13 tabelas diferentes, entre tabelas base (apenas com atributos estáticos) e tabelas dinâmicas (incluindo informações referentes ao tempo). Devido à transparência que se deseja, utilizou-se, ainda, os comandos IF do PL/SQL, pois, caso o usuário não forneça o tempo de validade inicial, o mesmo deve ser o momento atual.

A figura 8.5 apresenta a implementação de um procedimento (mensagem do TF-ORM) que visa atualizar o nome de uma pessoa. Este mesmo procedimento será utilizado para exemplificar as consultas na base temporal no próximo item.

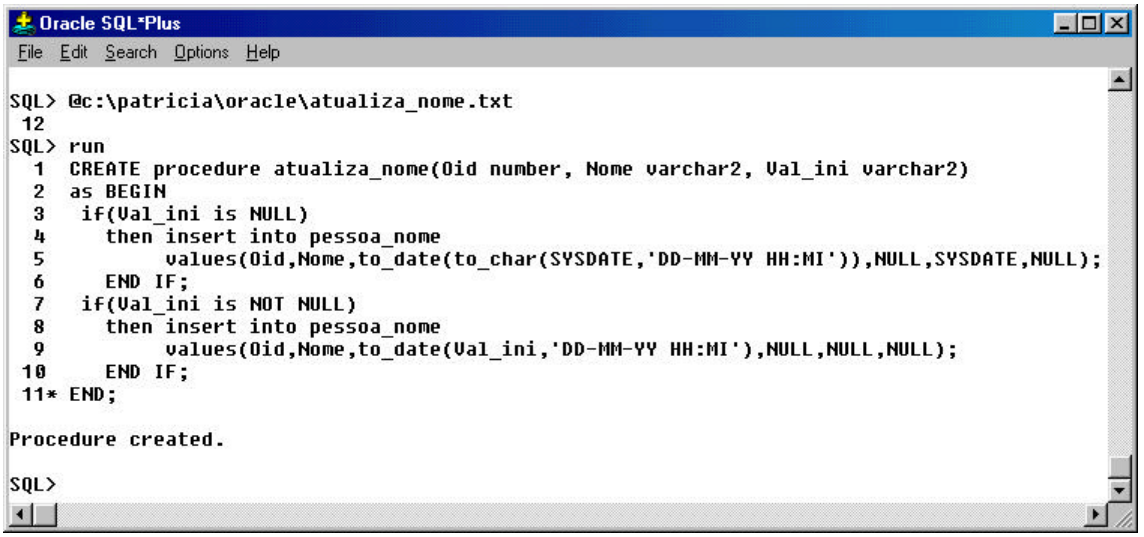



FIGURA 8.5 – Procedimento de Atualização da Base Temporal

A figura 8.6 apresenta a implementação da visão apresentada na figura 7.9, no capítulo 7.

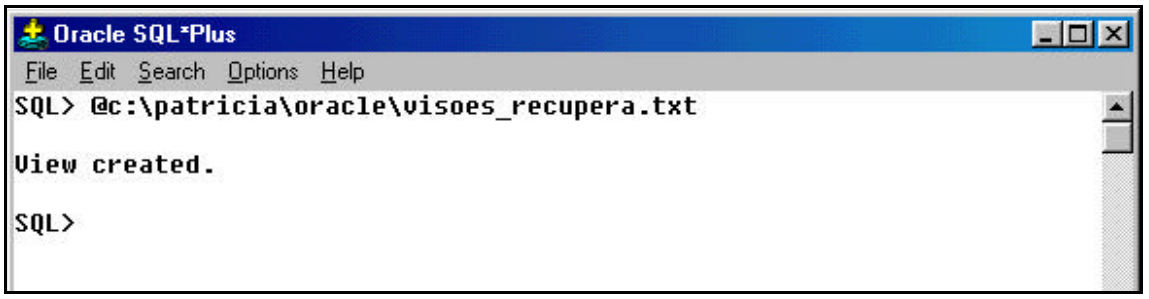



FIGURA 8.6 – Criação de uma visão no Oracle

Conforme comentado no capítulo 7, os gatilhos foram utilizados para as regras de gerência da base de dados, entre as quais encontram-se as regras que mantêm a base temporal consistente, como a atualização dos tempos de validade final e transação inicial e final, após a inserção de uma informação. É importante ressaltar que estes gatilhos são criados para cada tabela e acionados através de uma operação realizada sobre a mesma, como é o caso da situação apresentada pela figura 8.7.

O gatilho apresentado na figura 8.7 é executado após uma inserção na tabela *pessoa_nome*, o qual atualiza:

- o tempo de validade final da tupla anteriormente inserida, com o tempo de validade inicial da tupla inserida, menos uma unidade de granularidade temporal utilizada;
- o tempo de transação final da tupla anteriormente inserida, com o tempo de transação final da tupla inserida;
- o tempo de transação inicial da tupla inserida com o momento atual.

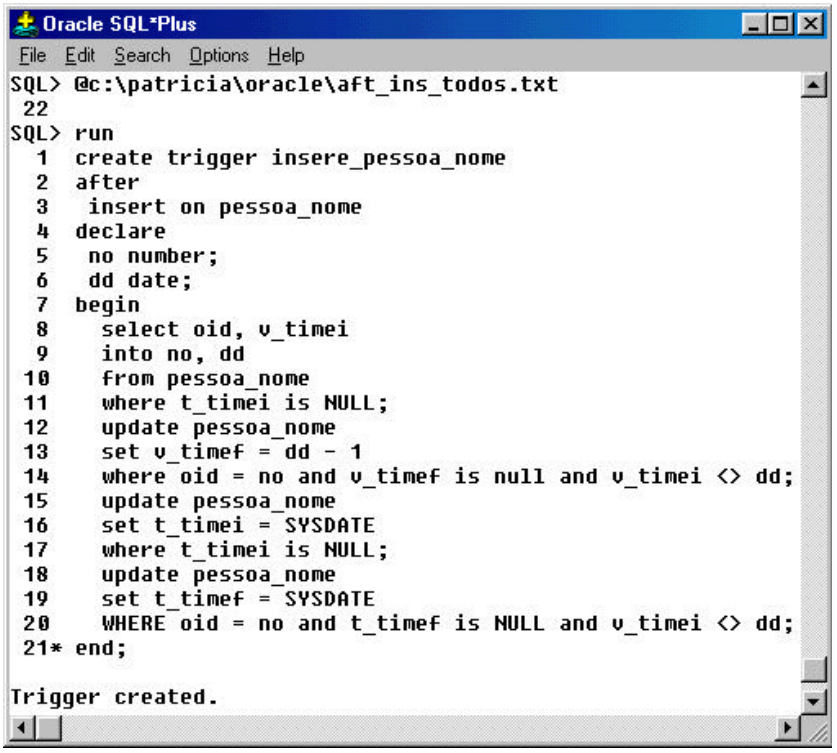



FIGURA 8.7 – Definição de um Gatilho

## 8.2 Execução no Banco de Dados Temporal

Nesta seção, são apresentadas execuções das implementações realizadas no item anterior, bem como consultas às tabelas criadas no SGBD. Tais atividades pretendem comprovar a implementação realizada, bem como a atuação de procedimentos, regras de transição de estados (simuladas) e regras de gerência.

A figura 8.8 apresenta uma consulta à base vazia, onde nenhuma informação foi inserida.

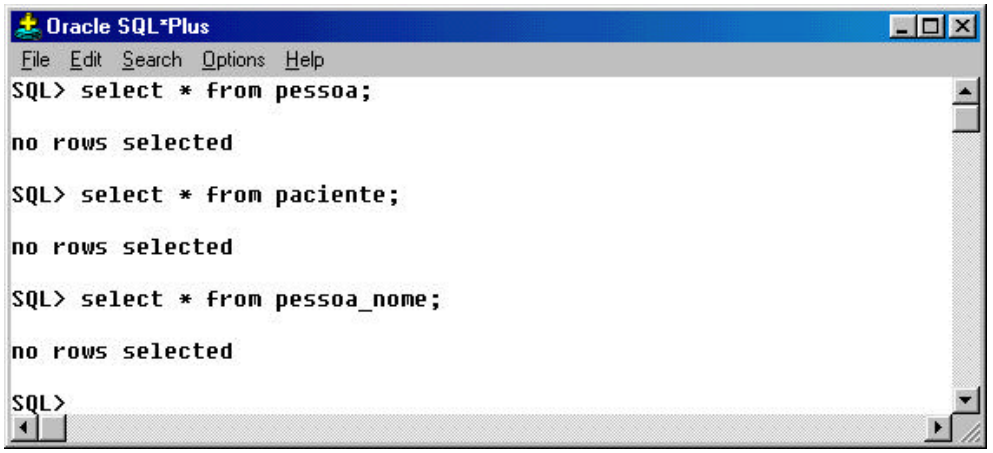



FIGURA 8.8 – Consulta à Base de Dados Temporal Vazia

Ao se utilizar o procedimento criado para inserção de novas informações na base temporal, as tabelas são populadas, como apresenta a figura 8.9. Sempre que uma nova instância de um objeto é criada, a tabela DYNAMIC é populada, através da execução do procedimento criado para a mensagem TF-ORM CREATE_OBJECT conforme apresenta a figura 8.10.

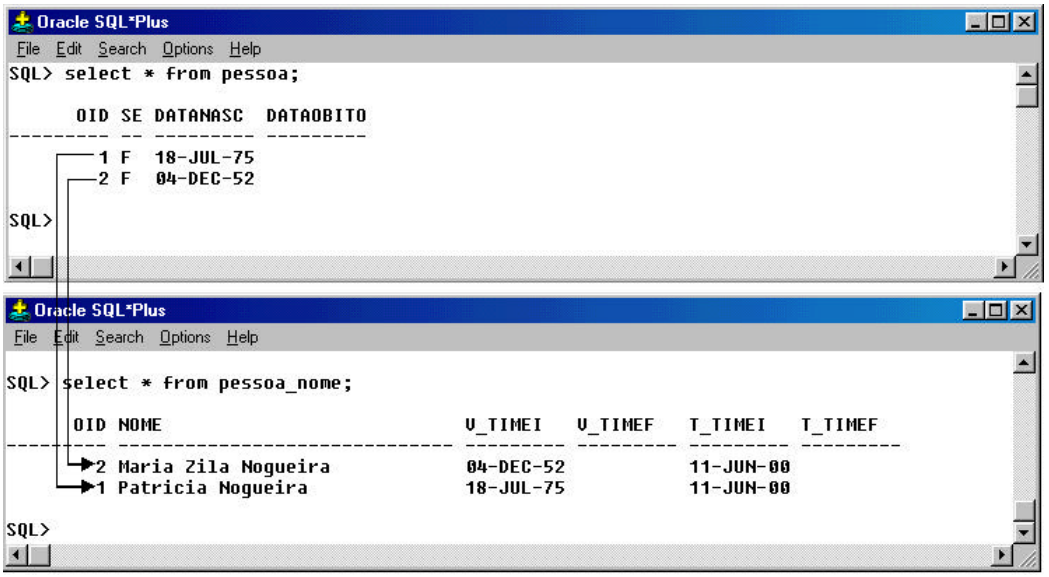



FIGURA 8.9 – Primeira Consulta Realizada sobre a Base Temporal

Na figura 8.9, verifica-se que as informações pertencentes à tabela *pessoa_nome* são informações novas e que fazem referência à tabela pessoa. Desta forma, como a tabela *pessoa_nome* é dinâmica, novas informações que forem inseridas para as mesmas pessoas de código 1 e 2 serão decorrentes de atualizações na base de dados, através da execução do procedimento apresentado na figura 8.11.

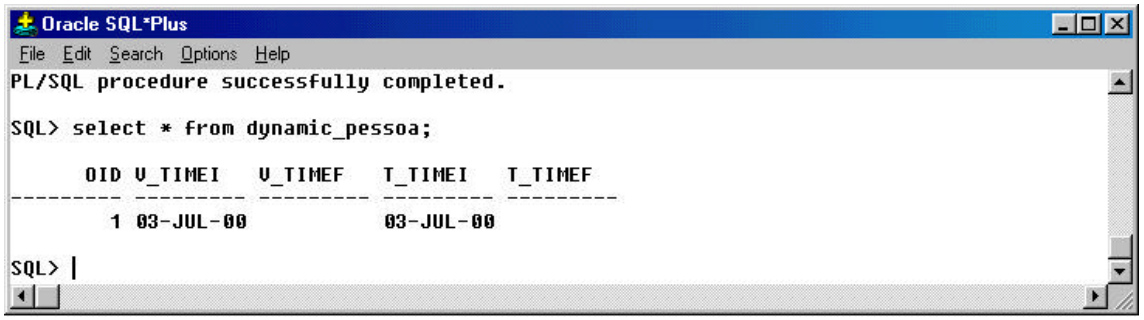



FIGURA 8.10 – Populando a Tabela Dynamic de Pessoa

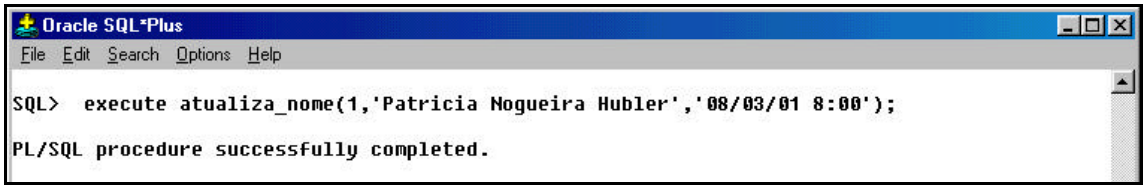



FIGURA 8.11 – Execução do Procedimento de Atualização do Nome

A partir do instante que uma informação é atualizada na base, as anteriores, também, devem ser atualizadas. Ou seja, os tempos de transação e validade são atualizados pelo gatilho que foi criado, sem que o usuário necessite fazer isto. Partindo da execução, quantas vezes forem necessárias, do procedimento apresentado na figura 8.11, a tabela pessoa_nome, apresenta as seguintes informações, conforme figura 8.12.

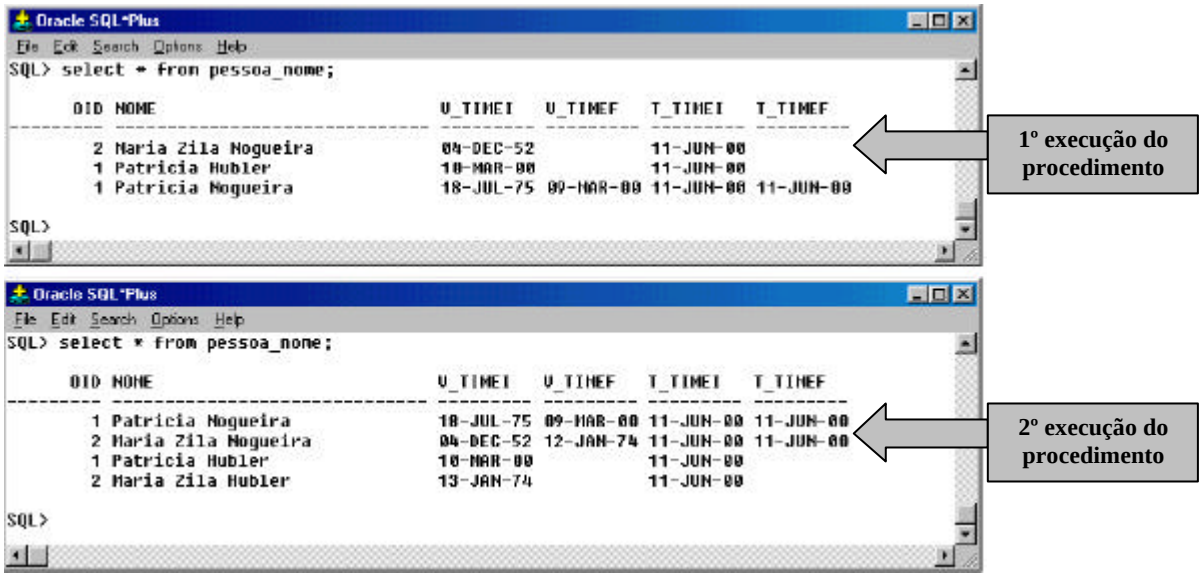



FIGURA 8.12 – Consulta Após Execução do Procedimento de Atualização

Na figura 8.12 existem algumas informações importantes. Pode-se fazer uma comparação entre a figura 8.9, onde os tempos de validade e transação final eram nulos e a figura 8.12, onde alguns tempos já foram atualizados – pelo gatilho. Na figura 8.12, primeira parte, o procedimento de atualização havia sido executado uma única vez, o que gerou a modificação na tupla mais antiga da pessoa de código 1. Enquanto que, na segunda parte, a modificação foi na pessoa de código 2, pois esta teve seu nome alterado. As duas tuplas que foram inseridas pelo procedimento de atualização apenas

terão seus tempos modificados quando novas tuplas forem inseridas para as mesmas pessoas.

A exclusão na base temporal desenvolvida neste trabalho, é a exclusão lógica, a qual faz com que o tempo de validade e transação final da tabela DYNAMIC, do objeto em questão, seja preenchido, conforme apresenta a figura 8.13, para o objeto pessoa. É importante ressaltar, que esta atualização também é realizada através de um procedimento, criado para este fim, por ser uma mensagem padrão do TF-ORM.

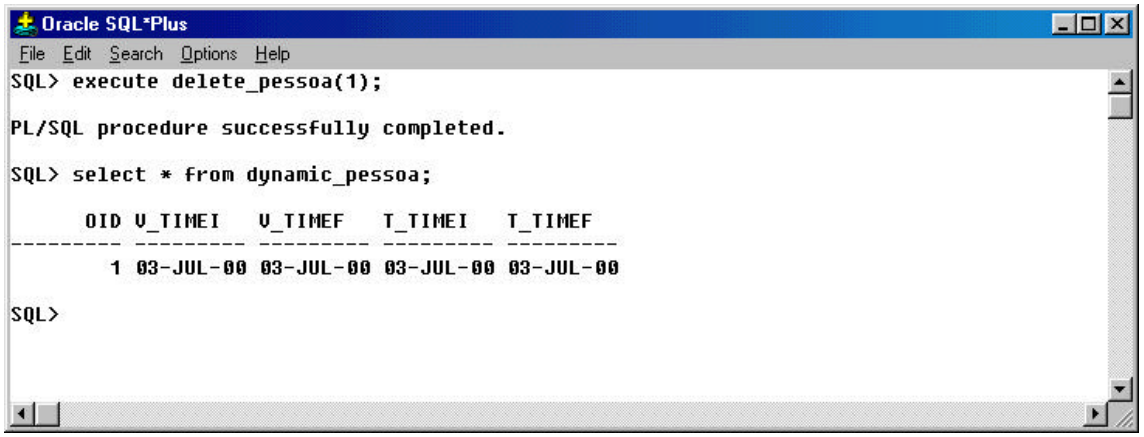



FIGURA 8.13 – Exclusão Lógica

A figura 8.14 apresenta a inclusão, na tabela DYNAMIC, de mais uma instância.

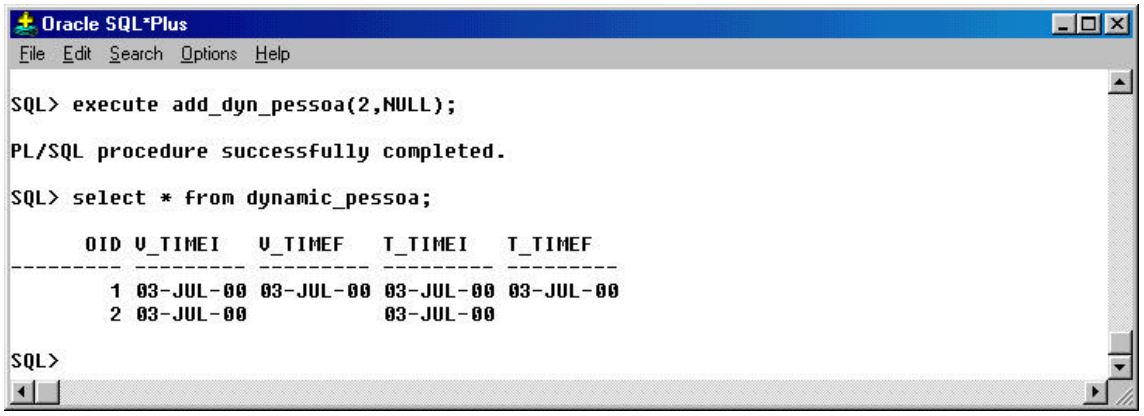



FIGURA 8.14 – Inserção de uma Nova Instância na Tabela Dynamic

A visão criada para a mensagem *recupera_valores*, é executada, retornando apenas os valores que ainda estão "vivos" na base de dados, ou seja, que ainda não foram excluídos, conforme figura 8.15.

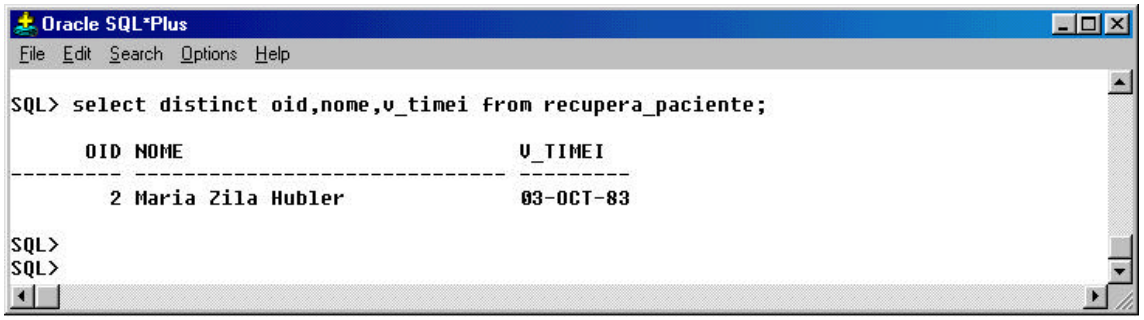



FIGURA 8.15 – Execução da Visão para Recuperar um Paciente Específico

Uma consulta para verificar que a granularidade utilizada realmente é o minuto, é apresentada na figura 8.16.

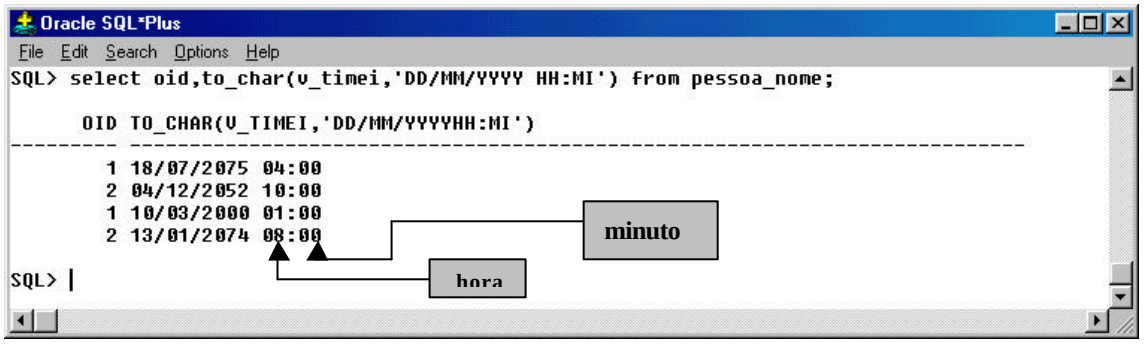



FIGURA 8.16 – Confirmação da Granularidade - Minuto

## 8.3 Considerações Finais

Este capítulo apresentou alguns exemplos da implementação sobre o mapeamento realizado, buscando validá-lo e, ao mesmo tempo, comprovar sua funcionalidade. Foi possível verificar o funcionamento dos procedimentos, visões e gatilhos sobre o estudo de caso definido no capítulo 6.

# 9 Conclusão

Aplicações que utilizam sistemas de informação apresentam dados que são armazenados em bancos de dados. A constatação de que as aplicações evoluem ao longo do tempo levou à necessidade de armazenar informações não só do momento atual, mas também do passado e do presente, levando ao desenvolvimento de bancos de dados temporais. Estes permitem armazenar e recuperar todos os estados de um objeto, registrando sua evolução ao longo do tempo.

Existem muitas bases de dados sendo utilizadas e, muitas vezes, a evolução temporal, embora necessária, não está sendo considerada. Surge, então, a necessidade de implementação de uma base temporal que satisfaça esta necessidade. Algumas implementações de modelos temporais sobre BDs convencionais já foram realizadas. Entretanto, estas implementações não apresentam um tratamento para regras que mantenham a base de dados temporal íntegra.

O objetivo deste trabalho foi a implementação de um banco de dados temporal em um SGBD convencional, no caso o Oracle, utilizando um modelo de dados temporal baseado em regras, o TF-ORM. Para possibilitar esta implementação, o modelo sofreu um mapeamento na busca da automatização e definição de um gerenciador.

Foram utilizados somente os recursos relacionais do Oracle, devido à grande utilização, ainda nos dias de hoje, de bases relacionais. A associação de tempo às informações, na maioria das vezes, surgirá em aplicações que já estejam em funcionamento, fazendo com que exista a solicitação de utilização da mesma plataforma.

## 9.1 Contribuições

As seguintes contribuições podem ser identificadas neste trabalho:

**a)** ***Análise da forma de gerenciar operações em um BDT, implementado sobre um BD convencional***

Este trabalho apresentou uma análise de diversas situações que podem ocorrer no gerenciamento de BDTs, quando implementadas sobre BDs convencionais. Foram considerados os três tipos de rótulos temporais que podem ser utilizados e, para cada um deles, foi feita uma análise de como gerenciar as operações de inserção, atualização e remoção de informações. Em alguns casos foram identificadas inconsistências, sendo sugeridas algumas ações alternativas para o gerenciador.

Os casos levantados neste trabalho podem servir de base para futuras implementações de BDTs sobre BDs convencionais. Uma análise das situações que podem ocorrer pode levar à definição do tipo de BDT a ser implementado (de tempo de transação, de validade, ou bitemporal) e de como podem executadas as operações de manutenção da base de dados. Para limitar os problemas devidos à manutenção dos

dados, o projetista pode limitar as inserções de informações apenas no presente, apenas no passado, apenas no futuro ou realizar suas próprias combinações.

A identificação de como devem ser manipuladas as diferentes operações é o primeiro passo para a implementação de um gerenciador temporal, que realmente execute as funções que são esperadas de um banco de dados temporal – armazenar todo o passado das informações, mantendo sua consistência.

A definição das possíveis operações que podem ser realizadas sobre determinados bancos de dados e seus respectivos tempos, foram essenciais para definir a implementação da base temporal.

**b)** ***Implementação de um BDT baseado em regras***

Para a implementação de um BDT sobre um BD relacional, as principais estruturas do modelo TF-ORM, um modelo de dados temporal orientado a objetos com papéis, foram utilizadas. Desde classes e papéis até regras de transição de estados foram analisadas para a implementação. Estas estruturas sofreram um mapeamento para o PL/SQL, linguagem do SGBD Oracle, o qual compreendeu as seguintes etapas:

- mapeamento das classes e papéis para tabelas;
- mapeamento das mensagens e decisões para procedimentos ou visões;
- mapeamento das regras de transição de estados – os estados que compõem uma regra foram mapeados para tabelas a parte, cujo tipo foi definido através de restrições do Oracle: *Check Constraints*; as mensagens que compõem uma regra foram mapeadas, conforme já citado, para procedimentos ou visões. Para representar a estrutura de uma regra, se optou por deixar para a aplicação, uma vez que outras formas de implementação alterariam o real significado da regra de transição de estados, pois estariam forçando a execução de mensagens e/ou visões.

**c)** ***Proposta de um mapeamento automático***

Através deste trabalho, foi possível a definição de algumas características da modelagem que podem ser automatizadas, tornando-se completamente transparentes aos usuários. Estas características foram analisadas e referem-se, principalmente, à criação de mensagens e regras de forma automática, utilizando-se de padronizações quando da definição destas propriedades. As classes e papéis já são implementadas a partir do modelo TF-ORM, sem uma nova intervenção do usuário.

## 9.2 Trabalhos Futuros

A continuação deste trabalho é evidente, uma vez que o tempo, associado às informações, é a cada dia mais necessário. Sugere-se, como trabalhos futuros:

a) *implementação de BDT em um BD orientado a objetos* - conforme apresentado, este trabalho utilizou apenas as características relacionais do SGBD Oracle, fazendo com que parte dos recursos deste banco não fossem utilizados. Como um dos possíveis

trabalhos futuros a serem desenvolvidos, sugere-se a implementação de um BD temporal orientado a objetos, utilizando, da mesma forma, regras de transição e gerência dos dados;

b) *utilização de regras de integridade* - as regras de integridade do modelo TF-ORM não foram tratadas neste trabalho, sendo um excelente ponto de partida para uma nova implementação;

c) *implementação do gerenciador automático* - deve ser implementado o gerenciador automático, definido neste trabalho, utilizando como entrada os arquivos gerados pela ferramenta de apoio à especificação. Uma interface, com as características mínimas, foi sugerida no capítulo 7.

# Anexo 1 - SGBD Objeto-Relacional Oracle

A escolha de um sistema gerenciador de bancos de dados para implementar o modelo TF-ORM baseou-se na utilização da linguagem SQL para manipulação das informações. O banco de dados escolhido foi o Oracle, por ser um banco de dados comercial bastante conhecido, além de utilizar o SQL.

A maioria dos SGBDs comerciais utiliza a linguagem de consulta SQL, cuja versão original foi desenvolvida no Laboratório de Pesquisas da IBM, hoje conhecido como Almaden. O SQL, antes chamado de SEQUEL, é uma linguagem de consulta estruturada (*Structured Query Language*). Em 1986 foi divulgado, pelo American National Standard Institute (ANSI), um padrão SQL. A partir de então, esta linguagem estabeleceu-se como o padrão de banco de dados relacional. De acordo com [KOR 95] a linguagem SQL possui diversas partes:

- Linguagem de Definição de Dados (*Data Definition Language* – DDL) – A SQL DDL fornece comandos para definição de esquemas, remoções, criação de índices e modificação de esquemas de uma relação (tabela);
- Linguagem de Manipulação de Dados Interativa (*Interactive Data Manipulation Language* – DML) – A DML inclui uma linguagem de consulta baseada na álgebra e no cálculo relacional. Possuindo comandos de consulta, inserção, remoção e alteração de tuplas;
- Linguagem de Manipulação de Dados Embutida (*Embedded Data Manipulation Language*) – Projetada para ser utilizada em linguagens de programação como PL/I, Cobol, Pascal, Fortran e C;
- Definição de Visões (*View Definition*) – Comandos para definição de visões;
- Autorização (*Authorization*) – Comandos de especificação e autorização de acesso a relações (tabelas) e visões, incluídos na SQL DDL;
- Integridade (*Integrity*) – Possui uma forma limitada para verificação de integridade. Esta é realizada através do controle de chaves e das formas de relacionamento entre as tabelas.;
- Controle de Transações (*Transaction Control*) – Inclui comandos para início e fim das transações. Estas correspondem a uma coleção de operações que executam uma única função lógica numa aplicação de banco de dados.

Um banco de dados relacional é uma forma extremamente simples de encontrar e manter os dados utilizados em uma corporação. Não é nada mais que uma coleção de tabelas de dados. Todas as tabelas apresentam um cabeçalho para cada coluna e tuplas com informações simples. Assim, um BD relacional pode ser sofisticado e poderoso. O Oracle 8 é um banco de dados objeto-relacional que suporta todas as características de um relacional, além de suportar conceitos e características próprias.

Um banco de dados objeto-relacional estende a capacidade dos relacionais para suportar os conceitos da orientação a objetos. O Oracle 8 pode ser utilizado, simplesmente, como um SGBD relacional ou como um objeto-relacional, com características de orientação a objetos.

Para que uma aplicação seja desenvolvida em Oracle 8 e utilizada de forma rápida e eficiente, os usuários e desenvolvedores devem utilizar uma linguagem comum, entendida por ambos. Desde a primeira versão, o Oracle tem sido baseado no modelo relacional, de fácil entendimento, assim, os usuários entendem facilmente o que ele faz e como faz. Muitas aplicações são desenvolvidas visando as facilidades do Oracle.

A informação é armazenada através de tabelas. Estas tabelas caracterizam-se por possuírem: colunas, tuplas e um nome que as identifique.

A inclusão dos conceitos de orientação a objetos no Oracle 8 não obriga a sua utilização quando um BD é implementado.

O custo de se utilizar objetos está baseado, principalmente, na adição da complexidade de sistemas e no tempo para se aprender tal metodologia. Entretanto, a capacidade OO incluída no Oracle é construída facilmente através do modelo relacional.

A linguagem de consulta do Oracle é chamada de PL/SQL. Nesta linguagem os comandos padrão do SQL permanecem, mas são adicionados novos comandos aumentando o poder de expressão da linguagem.

O SQL embutido é suportado pelos precompiladores do Oracle. O precompilador interpreta as declarações em SQL e as traduz para declarações que possam ser entendidas pelos compiladores de linguagens procedurais: (i)Precompilador Pro*Ada; (ii) Precompilador Pro*C/C++; (iii) Precompilador Pro*COBOL; (iv) Precompilador Pro*FORTRAN; (v) Precompilador Pro*Pascal; (vi) Precompilador Pro*PL/I; (vii) Precompilador Pro Java.

Nos próximos itens, serão apresentadas algumas características do Oracle como tipos de dados e primitivas do PL/SQL.

### Tipos de Dados do Oracle

Alguns tipos de dados do SGBD ORACLE são apresentados a seguir:

- **Caracteres** - utilizando-se as palavras: VARCHAR2(X) ou CHAR(X), onde X representa o número de caracteres que compõe o string. Sendo que para VARCHAR2 o tamanho máximo de caracteres é 2.000 e para CHAR é 255. Para ambos, o número mínimo é 1;

- **Números** – utilizando-se a palavra NUMBER(p,s), onde "p" corresponde à precisão (pertencendo ao intervalo de 1 a 38) e "s" à escala (intervalo de –84 a 127). A precisão e escala são utilizadas para armazenamento de números com ponto flutuante. Além destes, armazena zeros, números positivos e negativos;

- **Data** – utilizando a palavra (DATE). Para cada valor do tipo data pode-se armazenar informações do tipo: século, ano, mês, dia, hora, minuto e segundo. O padrão do tipo data é: 'DD-MON-YY' (Dia – Mês – Ano, onde o mês é dado pelas suas três primeiras letras). Para alterar este padrão, numa determinada seção, utiliza-se o comando: ALTER SESSION SET NLS_DATE_FORMAT = <data> especificando o tipo desejado. A data atual é fornecida pela cláusula SYSDATE. O tipo data permite operações aritméticas: adicionar dias, subtrair, multiplicar, dividir.

- **ROWID** – Cada tupla possui um endereço na base de dados. Esta coluna é denominada de ROWID. Seus valores são apresentados em strings hexadecimais, representando o endereço de cada tupla.

O Oracle suporta diferentes tipos de dados. Os tipos de dados abstratos são aqueles que se consistem de um ou mais subtipos. Desta forma, quando inicia-se a restrição para um tipo de dado padrão do Oracle como: ***number, date*** e ***varchar2***, os tipos abstratos de dados podem fornecer maior precisão à descrição destes dados. Por exemplo, um tipo de dados abstrato para endereço pode consistir das seguintes colunas apresentadas a seguir:

```
Rua       VARCHAR2(50)
Cidade    VARCHAR2(25)
Estado    CHAR(2)
CEP       NUMBER
```

Quando uma tabela for criada com informações referentes ao endereço, pode-se criar uma coluna que utilize este tipo de dado abstrato. Tipos de dados abstratos podem ser aninhados, contendo referências para outros tipos de dados, também abstratos.

Pode-se utilizar os mesmos para criar objetos tabelas. Em cada um destes, as colunas das tabelas são mapeadas para colunas do tipo de dados abstrato.

- **Tabelas aninhadas:** as tabelas aninhadas são uma coleção de tuplas, representadas como uma coluna dentro da tabela principal. Para cada registro da tabela principal, a tabela aninhada pode conter múltiplas tuplas.

- **Varying Arrays:** é, como uma tabela aninhada, uma coleção. Um *Varying Arrays* é um conjunto de objetos, cada um com o mesmo tipo de dados. O tamanho deste array é determinado quando da sua criação.

- **Large Objects** (LOB): é a capacidade de armazenar grandes volumes de dados. Os tipo de dados LOB são divididos em: BLOB, CLOB, NCLOB e BFILE. O BLOB é utilizado para dados binários e pode ter até 4GB de tamanho. O CLOB armazena caracteres e pode armazenar dados maiores que 4GB. O NCLOB é utilizado para armazenar dados CLOB para conjuntos de caracteres. Os dados

para BLOB, CLOB e NCLOB são armazenados dentro do BD. Assim, pode-se obter uma simples tupla no BD com tamanho superior a 4GB. O quarto tipo de dados LOB, o BFILE, é um ponteiro para um arquivo externo. Os arquivos existentes no sistema operacional são referenciados por BFILEs; o BD apenas mantém um ponteiro para o arquivo. O tamanho do arquivo externo é delimitado apenas pelo sistema operacional. Pode-se utilizar múltiplos LOBs por tabela.

Para se criar um tipo abstrato de dados utiliza-se a primitiva: ***create type***, como apresentado abaixo:

```
Create type ENDEREÇO_TY as object
(Rua       VARCHAR2(50),
 Cidade    VARCHAR2(25),
 Estado    CHAR(2),
 CEP       NUMBER);
```

### O Tipo Date e suas Operações

O DATE é um tipo de dado do Oracle, assim como CHAR e NUMBER o são. Entretanto, ele tem propriedades únicas. O tipo de dados DATE é armazenado em um formato interno especial no Oracle o qual não inclui apenas o mês, dia e ano, mas, também, hora, minuto e segundo. Este, apresenta algumas características que devem ser apresentadas:

- ***Sysdate***: primitiva interna que fornece a data corrente e a hora do sistema operacional do computador;
- ***Dual***: é uma pequena, mas útil tabela do Oracle criada para testar funções ou realizar cálculos rápidos;
- ***Funções de datas***: a seguir, são apresentadas algumas funções próprias do Oracle utilizadas para trabalhar com datas:
    - **ADD_MONTHS(date,count)**: adiciona o número de meses para a data;
    - **GREATEST(date1,date2,date3,...)**: fornece a maior data de uma lista de datas;
    - **LAST_DAY(date)**: fornece a data do último dia do mês no qual a data está;
    - **MONTHS_BETWEEN(date2,date1)**: fornece date2 - date1 em meses (pode gerar meses fracionários);
    - **NEXT_DAY(date,'day')**: fornece a data do próximo dia depois de *date*, onde '*day*' é '*Monday*', '*Tuesday*', e assim por diante;
    - **NEW_TIME(date,'this','other')**: fornece a data (e horário) em uma determinada área: *this*. t*his* será substituído por três letras, as quais abreviam a área corrente. o*ther* será substituído pela abreviatura de três letras para uma outra área de tempo, para a qual deseja-se saber a hora e data. A seguir, são apresentadas as possíveis área de tempo:

| | |
|---|---|
| AST/ADT | • Atlantic standard/daylight time |
| BST/BDT | • Bering standard/daylight time |
| CST/CDT | • Central standard/daylight time |
| EST/EDT | • Eastern standard/daylight time |
| GMT | • Greenwich mean time |
| HST/HDT | • Alaska-Hawaii standard/daylight time |
| MST/MDT | • Mountain standard/daylight time |
| NST | • Newfoundland standard time |
| PST/PDT | • Pacific standard/daylight time |
| YST/YDT | • Yukon standard/daylight time |

    - **ROUND(date,'format')**: sem formato (*format*) especificado, arredonda uma data para 12:00;

- **TRUNC(date,'format')**: sem formato (*format*) especificado, seta uma data para 12:00;
- **TO_CHAR(date,'format')**: reformata a data de acordo com o formato (*format*);
- **TO_DATE(string,'format')**: converte uma string em um dado formato (*format*) para uma data dentro do Oracle.

Qualquer uma destas funções podem ser combinadas quando utilizadas.

### Primitivas do SQL - PL/SQL

O Oracle, utilizando como DML primitiva o SQL, possui em sua linguagem própria, o PL/SQL, muitas características do SQL original. Desta forma, serão apresentadas neste item as principais características desta DML, sua forma de declaração e utilização.

### Inserindo Informações

O comando INSERT permite a inserção de uma tupla de informações diretamente dentro de uma tabela (ou indiretamente, se for realizada sobre uma visão):

```
INSERT INTO tabela VALUES (valores);
```

A palavra VALUES deve preceder a lista de dados a serem inseridos. Uma *string* deve estar entre aspas simples. Números são apresentados sem qualquer elemento adicional. Cada campo é separado por vírgula e estes devem estar na mesma ordem das colunas da tabela. As datas, no formato *default*, devem estar entre aspas simples. Caso deseja-se inserir datas em qualquer outro formato, deve-se utilizar a função TO_DATE.

Quando deseja-se inserir colunas em qualquer outra ordem que não seja o padrão da declaração das colunas na tabela, deve-se descrever, antes dos valores, uma lista na ordem desejada dos dados.

Quando se insere, atualiza ou remove dados de uma banco de dados, pode-se desfazer estas operações através da execução de um *rollback*. Isto pode ser muito importante quando um erro é descoberto. O processo de finalizar ou desfazer uma ou várias operações é controlado por dois comandos do SQLPLUS: ***commit*** e ***rollback***.

### Removendo Informações

Remover uma ou várias tuplas de uma tabela requer o comando DELETE. A cláusula WHERE é essencial para remover somente as tuplas desejadas. Um DELETE sem WHERE removerá todas as tuplas da tabela.

Desta forma, a representação básica deste comando é apresentada a seguir:

```
DELETE <nome da tabela>
WHERE <restrição>
```

### Atualizando Informações

O comando UPDATE requer um conjunto específico de valores para cada coluna que se deseja alterar, especificando qual tupla ou tuplas deseja-se afetar pela utilização da cláusula WHERE.

É forma básica de representação deste operador é apresentada a seguir:

```
UPDATE <nome da tabela>
SET <lista de atributos>
WHERE <restrições>
```

### Criando Tabelas

A declaração CREATE TABLE fornece diferentes tipos de restrições em uma tabela: chaves candidatas, chaves primárias, chaves estrangeiras e condições de checagem. Uma cláusula de restrição (***constraint***) pode restringir uma simples coluna ou grupo de colunas de uma tabela. O objetivo destas restrições é fazer com que o Oracle mantenha a integridade do BD. A maioria das restrições é adicionada quando da criação da tabela, outras podem ser desenvolvidas por aplicações específicas.

Existem duas formas para se especificar restrições: como parte da definição da coluna ou no final da declaração da tabela.

Uma chave candidata é a combinação de uma ou mais colunas, cujos valores devem ser únicos em cada tupla da tabela.

O exemplo a seguir apresenta a criação de uma tabela.

```
CREATE TABLE EMP(
Nome        VARCHAR2(30) NOT NULL,
Dnasc       DATE NOT NULL,
End         VARCHAR2(30),
UNIQUE   (Nome,DNasc) );
```

A chave desta tabela é a combinação de Nome e Dnasc. Ambas as colunas estão sendo declaradas como: NOT NULL, ou seja, seus valores não podem ser nulos. Se o NOT NULL não for especificado, a coluna poderá ter valores nulos.

A chave primária de uma tabela, conforme declarado a seguir, é uma das chaves candidatas, a qual recebe algumas características especiais. Deve existir apenas uma chave primária e esta não deve conter valores nulos.

```
CREATE TABLE EMP(
Nome        VARCHAR2(30),
Dnasc       DATE,
End         VARCHAR2(30),
PRIMARY KEY (Nome,DNasc) );
```

Esta declaração do CREATE TABLE tem a mesma função da declaração anterior, com uma única exceção: pode-se ter várias declarações de chaves únicas, entretanto, apenas uma restrição de chave primária.

Para uma única coluna, a declaração de chave primária ou candidate pode ser definida na própria declaração da coluna, conforme a seguir:

```
CREATE TABLE DEPT (
Nome        VARCHAR2(20),
Cod_dept    NUMBER(3) PRIMARY KEY );
```

Uma chave estrangeira é a combinação de colunas com valores baseados nas chaves primárias de outras tabelas. Uma restrição de chave estrangeira também fornece uma restrição de integridade referencial, especificando que os valores da chave estrangeira devem corresponder aos valores atuais de sua respectiva chave primária na outra tabela. A estrutura destas cláusulas pode ser observada abaixo:

```
CREATE TABLE Materiais(
Descrição       VARCHAR2(30),
Cod_mat         NUMBER(3) PRIMARY KEY,
Cod_dept        NUMBER(3)     REFERENCES
DEPT(Cod_dept) );
```

Muitas colunas têm valores que são pertencem a uma certa série, ou que satisfazem certas condições. Com uma *check constraint*, pode-se fornecer expressões que devem sempre ser verdadeira para todas as tuplas na tabela. Por exemplo, um empregado deve ter idade entre 18 e 65 anos, a restrição seria:

```
CREATE TABLE EMP(
Nome            VARCHAR2(30),
Dnasc           DATE,
Age NUMBER   CHECK   (Age   BETWEEN   18
and 65),
End             VARCHAR2(30),
PRIMARY KEY (Nome,DNasc) );
```

As restrições podem ser nomeadas. Se for utilizado um padrão para a nomenclatura, será simples identificar e manter as restrições. O nome de uma restrição deve identificar a tabela sobre a qual ela atua e o tipo da restrição. Por exemplo: a chave primária de EMP pode ser: EMP_PK. Se não foi especificado um nome para a restrição quando da sua criação, o ORACLE gera uma nome. Entretanto, com este nome, não se consegue descobrir qual a tabela possui a restrição e, nem sequer, o tipo de restrição.

No exemplo a seguir, a restrição de chave primária é criada e nomeada, como parte da criação da tabela:

```
CREATE TABLE EMP(
Nome        VARCHAR2(30),
Dnasc       DATE,
Age          NUMBER CHECK (Age BETWEEN
18 and 65),
End          VARCHAR2(30),
CONSTRAINT   EMP_PK   PRIMARY   KEY
(Nome,DNasc) );
```

A cláusula ***constraint*** do comando ***create table*** nomeia a restrição, neste caso: EMP_PK.

**Operadores Adicionais do PL/SQL**

O PL/SQL possui alguns operadores adicionais que serão utilizados em exemplos nos próximos capítulos. Entre eles:

- GREATEST - retorna o maior dentre um conjunto de valores. Trabalha sobre tuplas individuais, elementos de uma mesma tupla;
- LEAST - retorna o menor valor dentre um conjunto de valores.

No PL/SQL existem mecanismos para executar um ou mais comandos, automaticamente, quando uma operação é realizada sobre uma relação do banco de dados. Estes mecanismos são chamados de *triggers* (gatilho). O *trigger* executa uma série de operações, previamente criadas. A ação que dispara o *trigger* pode ser uma inserção, atualização ou exclusão em uma determinada tabela.

Para projetar um *trigger* é necessário especificar as condições sobre as quais o ele deve ser acionado e quais as ações a serem tomadas quando o mesmo é executado. Para criar um *trigger* utiliza-se a cláusula CREATE TRIGGER apresentada a seguir.

```
CREATE TRIGGER < nome >
BEFORE / AFTER
FOR EACH ROW
DECLARE
      <variável>  <tipo>;
< ação origem (delete, insert ou update) > OF
<nome de atributos (update)>
ON < nome da tabela >
BEGIN
     <ações a serem executadas>
END;
```

As restrições BEFORE ou AFTER são utilizadas de acordo com a necessidade do desenvolvedor do *trigger*. Caso deseja-se verificar a consistência dos dados, antes da operação ser executada, por exemplo, a cláusula BEFORE é utilizada. O BEFORE não permite atualização na tabela que está sendo verificada, mas aceita comparações realizadas através da utilização dos operadores *new* (valor atual do atributo) e/ou *old* (valor antigo do atributo). Caso deseja-se atualizar uma tabela, após uma determinada ação, a restrição AFTER é utilizada.

Quando a condição para disparar o *trigger* estiver relacionada ao UPDATE, deve-se especificar sobre quais atributos esta ação se refere. Sendo assim, o *trigger* somente será acionado para os atributos correspondentes. O INSERT e o DELETE necessitam, apenas, do nome da tabela sobre a qual deverão ser acionados.

A cláusula FOR EACH ROW é utilizada quando o *trigger* for disparado antes da execução sobre a tabela (BEFORE) e neste caso, pode-se utilizar os operadores *new* e/ou *old*. Esta cláusula determina que a pesquisa será realizada para cada tupla da tabela correspondente.

Quando utiliza-se variáveis para armazenar valores de atributos retornados por uma consulta, os mesmos devem ser declarados na cláusula DECLARE. Cada variável possuirá um nome e um tipo e deverá ser precedida de ponto-e-vírgula (;). Esta cláusula é utilizada tanto com a restrição BEFORE, quando com a restrição AFTER.

Como nas linguagens de programação procedural, o PL/SQL possibilita a criação de procedimentos, chamados *stored procedures*. Estes são um conjunto de declarações em PL/SQL que

podem ser chamadas pelo nome, ou seja, como nas demais linguagens de programação, um conjunto de comandos utilizados para realizar determinada ação.

Para criar um procedimento deve-se utilizar o comando CREATE PROCEDURE:

```
CREATE PROCEDURE <nome do
procedimento> ( <argumentos>)
AS BEGIN
UPDATE <nome da tabela>
SET <atributo> = <atualização do atributo)
WHERE <condição>;
END;
```

Um procedimento deve ser criado fornecendo, à cláusula, o seu nome, o qual servirá de referência para uma futura execução, e os argumentos que deverão ser passados para o procedimento.

Um procedimento não retorna valor, sendo assim é utilizado, apenas, em operações de inserção e/ou atualização que não necessitem de consultas a tabelas. Não permite que testes sejam realizados sobre os dados, a não ser na cláusula WHERE da operação de atualização (UPDATE), ou seja, a operação SELECT não pode ser realizada dentro de um procedimento.

Para executar um procedimento deve preceder, ao nome do mesmo, a primitiva EXECUTE. O nome do procedimento deve ser seguido dos parâmetros correspondentes.

Caso deseja-se consultar uma tabela deve-se criar uma função específica. Isto é possível utilizando-se o comando CREATE FUNCTION:

```
CREATE FUNCTION <nome>(<parâmetros>)
RETURN <tipo>
IS
        <variável>  <tipo>;
BEGIN
        SELECT <atributo>
        INTO <variável>
        FROM <tabela>
        WHERE <restrição>;
        RETURN(<variável);
END;
```

A função deve ter um nome e, como no procedimento, devem ser especificados os parâmetros para que ela seja executada. Estes parâmetros precisam ser declarados com seus respectivos tipos. Sendo uma função, retorna valor. Isto é apresentado através da primitiva RETURN seguida do tipo que deverá ser igual ao da variável que será retornada. A descrição dos passos de execução da função é formada pelo comando SELECT com suas respectivas cláusulas e operadores e pela primitiva RETURN seguida da variável.

Para que uma função seja executada, diferente de um procedimento, deve-se utilizar a cláusula SELECT. Esta é seguida do nome da função e seus respectivos parâmetros e da cláusula FROM determinando a tabela a ser pesquisada como DUAL.

O único momento em que o Oracle fará distinção entre letras maiúsculas e minúsculas será quando de uma comparação com a base de dados, ou seja, com os valores já armazenados.

# Anexo 2 - Modelagem TF-ORM

## *Modelagem Completa do Estudo de Caso para o Modelo TF-ORM*

agent class(

**PESSOA**,

< **Base_role**,

static properties = {

(sexo, {F,M}),

(dataNasc, date),

(dataObito, date) }

dynamic properties = {

(nome, string),

(end, string),

(cidade, string),

(UF, string),

(fone, string),

(fax, string) }

rules = {

r1: msg(**create_object**) ⇒ msg(**allow_role**(paciente)),

r2: msg(**create_object**) ⇒ msg(**allow_role**(medico)),

>,

< **Paciente**,

static properties = {

(nroFicha, string),}

dynamic properties = {

(profissao, string),

(religiao, string),

(educacao, string),

(moradia, string) }

states = { cadastro, emConsulta, esperaRetorno, foraConsulta, noCeu }

messages = {

valores_iniciais (NroFicha:string, Sexo:{F,M}, DataNasc:date, Nome:string, End: string, Cidade: string, Uf: string, Fone: string, Fax: string, Profissao: string, Religiao: string, Educacao: string, Moradia: string) from CONTROLE.PacienteCtrl,

recupera_paciente (NroFicha:string, Sexo:{F,M}, DataNasc:date, Nome:string, End: string, Cidade: string, Uf: string, Fone: string, Fax: string, Profissao: string, Religiao: string, Educacao: string, Moradia: string) from CONTROLE.PacienteCtrl,

update_dataObito (NroFicha: string, NovaDataObito: date) to CONTROLE.PacienteCtrl,

... },

decisions = {

novo_nome (nome: string),

nova_religiao (religiao: string)

…},

rules = {

inicio:

msg(**add_role**) ⇒ state(cadastro),

inicializacao:

state(cadastro), msg(valores_iniciais (NroFicha, Sexo, DataNasc, Nome, End, Cidade, Uf, Fone, Fax, Profissao, Religiao, Educacao, Moradia)) ⇒ state(emConsulta);

(not exists Rid (value(rid, nroFicha) = NroFicha)),

mudança _de_nome:

state(fora_consulta), decision(novo_nome(Nome))⇒state(fora_consulta ),

requis_obito:

msg(update_dataObito(nroFicha, newDataObito)) ⇒ state( noCeu ); (state (oid, rid) = emConsulta OR state (oid, rid) = esperaRetorno OR state (oid, rid) = foraConsulta),

... }

>,

< **Medico**,

static properties = { (CREMERS, string) }

dynamic properties = {

(especialidade, string) ,

(salario, real) }

messages = {

estuda_valores (QueixaPcpal: string, Inicio: date, Intensidade: string, Frequencia: string, Duracao: string, Agravante: string, Atenuante: string, OcorPredTurno: {dia/noite}, OcorPredAtivid: {repouso/exerc}, OcorPredPosicao: {ereta/sentada/deitada}, HistPregressa: string, AntecPessoal: string, AntecFamilia: string) from ANAMNESE.ObjAnamnese,

…}

states = { ativo, analise, aposentado }

decisions = { pedir_exame(Exame: EXAMEFISICO),

... }

rules = {

analise:

state(ativo), msg(estuda_valores (QueixaPcpal, Inicio, Intensidade, Frequencia, Duracao, Agravante, Atenuante, OcorPredTurno, OcorPredAtivid, OcorPredPosicao, HistPregressa, AntecPessoal, AntecFamilia)) ⇒ state( analise ); state (analise), decision(pedir_exame(Exame)) ⇒ msg (Cria_ObjProntu), msg(Cria_ObjConsulta),

... } >,

< **Funcionário**,

static properties = { (nroF, string) }

dynamic properties = { ... }

decisions = { escolhe_medico (NroFicha: string, Cremers: MEDICO),

... }

messages = {

atribui_medico (NroFicha: string, Especialidade: string ) from Controle.PacienteCtrl,

escolheu_medico(NroFicha: string, Cremers: string) to Controle.PacienteCtrl

... }

states = { ativo, analisandoCaso, aposentado }

rule = {

analisa_medico:

state (ativo), msg (atribui_medico (NroFicha, Especialidade)) ⇒ state (analisandoCaso),

decide_medico:

state (analisandoCaso), decision (escolhe_medico(NroFicha, Cremers)) ⇒ msg(escolheu_medico(NroFicha, Cremers)), state (ativo),

... }

>)

resource class (

**PRONTUARIO**

< **Base_role**,

rules = {

r1: msg(**create_object**) ⇒ state (**active**),

r2: state(**active**) ⇒ **add_role**(objProntuario), state(**active**) }

> ,

< **ObjProntu**,

static properties = { (codProntu, string) },

dynamic properties = {

(vigencia, string),

{nroConsulta **set of** integer},

(atendidoPor, MEDICO) },

messages = {

```
        novoObjProntu (codProntu: string, vigencia: string, nroConsulta: integer) from
            CONSULTA.objProntuCtrl,
        atualizaEstado (codProntu: string) to CONSULTA.objProntuCtrl,
        cria_prontuario (CodProntu: string, Vigencia: string, NroConsulta: integer,
            AtendidoPor: MEDICO) from CONTROLE.PacienteCtrl,
        ... },
      states = { criacao, emVigencia, foraVigencia }
      rules = {
        criacao:
            msg(add_role) ⇒ state(criacao),
        inicializacao:
            state(criacao), msg(novoObjProntu (CodProntu, Vigencia, NroConsulta,
            Medico)) ⇒ state(emVigencia);
            (not exists Rid (value(rid, codProntu) = CodProntu)),
        atualiza:
              state (emVigencia), msg(atualizaEstado (CodProntu) ⇒ state(
              foraVigencia )
        ... }
    >)
process class (
    CONTROLE,
    < Base_role, ... >,
    < MédicoCtrl,
      states = { active }
      messages = {
        defineMedico (OId: string, RId: string) from CONSULTA.ObjConsulta,
        medicoConsulta (OId: string, RId: string, Med:PESSOA.medico) to
            CONSULTA.ObjConsulta,
        ... }
      rules = { ... }
    >)
    < PacienteCtrl,
      states = { active }
      messages = {
        valores_iniciais (NroFicha:string, Sexo:{F,M}, DataNasc:date, Nome:string,
            End: string, Cidade: string, Uf: string, Fone: string, Fax: string, Profissao:
            string, Religiao: string, Educacao: string, Moradia: string) to
            PESSOA.Paciente,
        recupera_valores (NroFicha:string, Sexo:{F,M}, DataNasc:date, Nome:string,
            End: string, Cidade: string, Uf: string, Fone: string, Fax: string, Profissao:
            string, Religiao: string, Educacao: string, Moradia: string) to
            PESSOA.Paciente,
```

atribui_medico (NroFicha: string, Especialidade: string ) to Pessoa.Funcionario,

cria_prontuario (CodProntu: string, Vigencia: string, NroConsulta: integer, AtendidoPor: MEDICO) to PRONTUARIO.ObjProntu,

cria_consulta (NroConsulta: string, Med: PESSOA.Medico, Pac: PESSOA.Paciente) to CONSULTA.ObjConsulta,

reconsulta (OId: string, RId: string, Med: PESSOA.Medico, Pac: PESSOA.Paciente) to CONSULTA.ObjConsulta,

update_dataObito (NroFicha: string, NovaDataObito: date) from EXTERNAL_WORLD,

add_Paciente (NroFicha:string, Sexo:{F,M}, DataNasc:date, Nome:string, End: string, Cidade: string, Uf: string, Fone: string, Fax: string, Profissao: string, Religiao: string, Educacao: string, Moradia: string) from EXTERNAL_WORLD,

... },

rules = {

inicio:

msg(**add_role**) ⇒ state(cadastro),

inicializacao:

state(cadastro), msg( add_Paciente (NroFicha, Sexo, DataNasc, Nome, End, Cidade, Uf, Fone, Fax, Profissao, Religiao, Educacao, Moradia) ⇒ ( **add_role** ( PESSOA.Paciente)), msg (valores_iniciais (NroFicha, Sexo, DataNasc, Nome, End, Cidade, Uf, Fone, Fax, Profissao, Religiao, Educacao, Moradia)), state(emConsulta)),

requis_paciente_obito:

state(active), msg(update_dataObito(nroFicha, newDataObito) ⇒ state(noCeu); (state (oid, rid) = emConsulta OR state (oid, rid) = esperaRetorno OR state (oid, rid) = foraConsulta),

... }

>)

process class (

**CONSULTA composed of** { ANAMNESE, EXAMEFISICO, EXAMELAUDO, TRATAMENTO, RETORNO }

< **Base_role**

rules = { msg(create_object) ⇒ msg(add_role(ObjConsulta)), state(active),

... }

> )

< **ObjConsulta**,

static properties = { (nroConsulta, string) }

dynamic properties = { },

messages = {

define_medico (OId: string, RId: string) to CONTROLE.MedicoCtrl,

medico_consulta (OId: string, RId: string, Med:PESSOA.medico) from CONTROLE.MedicoCtrl,

cria_consulta (NroConsulta: string, Med: PESSOA.Medico, Pac: PESSOA.Paciente) from CONTROLE.PacienteCtrl,

reconsulta (OId: string, RId: string, Med: PESSOA.Medico, Pac: PESSOA.Paciente) from CONTROLE.PacienteCtrl,

cria_anamnese ( Pac: PESSOA.Paciente, Med: PESSOA.Medico, Cons: CONSULTA.ObjConsulta ) to ANAMNESE.ObjAnamnese,

…},

states = { *conjunto de estados* }

rules = {

msg(add_role) ⇒ msg(DadosPaciente), state(espera_indetificacao_paciente),

... }

>)

process class (

**ANAMNESE**,

< **Base_role**

rules = { *conjunto de regras* }

>,

< **ObjAnamnese**,

static properties = { }

dynamic properties = {

(queixaPcpal, string),

(inicio, date),

(intensidade, string),

(frequencia, string),

(duracao, string),

(agravante, string),

(atenuante, string),

(ocorPredTurno, { dia/noite }),

(ocorPredAtivid, { repouso/exerc }),

(ocorPredPosicao, { ereta/sentada/deitada } ),

(histPregressa, string),

(antecPessoal, string),

(antecFamilia, string }

messages = {

valores_iniciais (NroFicha:string, Sexo:{F,M}, DataNasc:date, Nome:string, End: string, Cidade: string, Uf: string, Fone: string, Fax: string, Profissao: string, Religiao: string, Educacao: string, Moradia: string) to PESSOA.Paciente,

recupera_valores (NroFicha:string, Sexo:{F,M}, DataNasc:date, Nome:string, End: string, Cidade: string, Uf: string, Fone: string, Fax: string, Profissao: string, Religiao: string, Educacao: string, Moradia: string) to PESSOA.Paciente,

update_dataObito (NroFicha: string, NovaDataObito: date) from PESSOA.Paciente ,

add_Paciente (NroFicha:string, Sexo:{F,M}, DataNasc:date, Nome:string, End: string, Cidade: string, Uf: string, Fone: string, Fax: string, Profissao: string, Religiao: string, Educacao: string, Moradia: string),

cria_anamnese ( Pac: PESSOA.Paciente, Med: PESSOA.Medico, Cons: CONSULTA.ObjConsulta ) from CONSULTA.ObjConsulta,

}

states = { }

rules = {

inicio:

msg(**add_role**) ⇒ state(cadastro),

inicializacao:

state(cadastro), msg( add_Paciente (NroFicha, Sexo, DataNasc, Nome, End, Cidade, Uf, Fone, Fax, Profissao, Religiao, Educacao, Moradia) ⇒ ( **add_role** ( PESSOA.Paciente)), msg (valores_iniciais (NroFicha, Sexo, DataNasc, Nome, End, Cidade, Uf, Fone, Fax, Profissao, Religiao, Educacao, Moradia)), state(emConsulta)),

requis_paciente_obito:

state(active), msg(update_dataObito(nroFicha, newDataObito) ⇒ state(noCeu); (state (oid, rid) = emConsulta OR state (oid, rid) = esperaRetorno OR state (oid, rid) = foraConsulta),

... }

> )

process class (

**EXAMEFISICO**,

< ...

>)

process class (

**EXAMELAUDO**,

< ...

>)

process class (

**TRATAMENTO**,

< ...

>)

process class (

**RETORNO**,

< ...>)

# Bibliografia


[ANT 97] ANTUNES, Dante Carlos; HEUSER, Carlos A.; EDELWEISS, Nina. TempER: uma abordagem para modelagem temporal de banco de dados. **Revista de Informática Teórica e Aplicada**, Porto Alegre, v.4, n.1, p.49-85, 1997.

[ANT 98] ANTUNES, Dante Carlos; HEUSER, Carlos A.; EDELWEISS, Nina. Modelagem Temporal de Transações em Banco de Dados. In: JORNADAS IBEROAMERICANAS DE ENGENIERIA DE REQUISITOS Y AMBIENTES SOFTWAR, 1998. Torres. **Anais...** Porto Alegre: Instituto de Informática, UFRGS, 1998. v.1, p.50-64.

[ARR 94] ARRUDA, E. H. P. **Um Modelo de Implementação da Linguagem TF-ORM para o SGBD $O_2$.** Porto Alegre: CPGCC - UFRGS, 1994. (TI - 408).

[BAR 99] BARROS, E.; et al. In: **Exame Clínico:** consulta rápida. Porto Alegre: Artmed, 1999. 260p., p. 17-25.

[BRA 94] BRAYNER, Ângelo R.A.; MEDEIROS, Claudia B. Incorporação do tempo em um SGBD orientado a objetos. In: SIMPÓSIO BRASILEIRO DE BANCOS DE DADOS, 9.,1994, São Carlos, SP. **Anais...** São Carlos: USP, 1994.

[CAR 97] CARVALHO, Tanisi Pereira de. **Implementação de Consultas para um Modelo de Dados Temporal Orientado a Objetos.** Porto Alegre: CPGCC da UFRGS, 1997. Dissertação de Mestrado.

[CAV 95] CAVALCANTI, J. M. B et al. Uma abordagem para a implementação de um modelo temporal orientado a objetos usando SGBDs relacionais. In: SIMPÓSIO BRASILEIRO DE BANCOS DE DADOS, 10.,1995, Recife. **Anais...** Recife: UFPE, 1995.

[CLI 87] CLIFFORD, J.; CROCKER, A. The historical relational data model (HRDM) and algebra based on lifespans. **IEEE Transactions on Software Engineering,** New York, v.14, n.7, July 1988. Trabalho apresentado na International Conference on Data Engineering, 1987, Los Angeles, California.

[CLI 93] CLIFFORD, J; CROCKER, A. The Historical Relational Data Model (HRDM) Reviseted. In: TANSEL, A.V. (Ed.). **Temporal Databases:** Theory, Design, and Implementation**.** Bridge Parkway: Benjamin/Cummings, 1993.

[CLI 93a] CLIFFORD, J; CROCKER, A; YUZHILIN, A. On the completeness of query languages for grouped and ungrouped historical data models. In: TANSEL, A.V. (Ed.). **Temporal Databases:** Theory, Design, and Implementation. Bridge Parkway: Benjamin/Cummings, 1993.

[DEM 00] DEMARTINI, G.; EDLEWEISS, N. **Ferramenta de Especificação TF-ORM e Mapeamento para um Banco de Dados Relacional**. Porto Alegre: PPGC da UFRGS, 2000. (RP – 307).

[EDE 93a] EDELWEISS, N.; OLIVEIRA, J.P.M.; CASTILHO, J.M.V. de. A Temporal Logic Language for Temporal Conditions Definition. In: CONFERENCIA INTERNACIONAL DE LA SOCIEDAD CHILENA DE CIENCIA DE LA COMPUTACION, 13., 1993, La Serena, CI. **Actas...** Santiago: Sociedad Chilena de la Computacion, 1993.

[EDE 93] EDELWEISS, N.; Oliveira, J.Palazzo. M. de; PERNECI, B. An Object-Oriented Temporal Model. In: INTERNATIONAL CONFERENCE ON ADVANCED INFORMATION SYSTEMS ENGINEERING - CAISE'93, 5., 1993, Paris. **Proceedings...** Heidelberg: Springer-Verlag, 1993. p.397-415. (Lecture Notes in Computer Science, v.685).

[EDE 94] EDELWEISS, N. **Sistemas de Informação de Escritórios:** Um Modelo para Especificações Temporais. Porto Alegre: CPGCC da UFRGS, 1994. Tese de Doutorado.

[EDE 94a] EDELWEISS, N. Modelagem de Aspectos Temporais de Sistemas de Informação. In: ESCOLA DE COMPUTAÇÃO, 9.,1994, Recife. **Anais...** Recife: UFPE, 1994.

[EDE 98] EDELWEISS, N. Bancos de Dados Temporais: Teoria e Prática. In: JORNADA DE ATUALIZAÇÃO EM INFORMÁTICA, 17; CONGRESSO NACIONAL DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO, 18., 1998. **Anais...** Recife: Sociedade Brasileira de Computação, 1998. p.225-282.

[ETZ 98] ETZION, O.; JAJODIA, S.; SRIPADA, E. (Eds.) **Temporal Databases**: Research and Practice. Berlin: Springer-Verlag, 1998. (Lecture Notes in Computer Science, v. 399).

[ELM 89] ELMASRI, R.; NAVATHE, S. **Fundamentals of Database Systems.** Cidade: Benjamin/Cummings, 1989.

[ELM 93] ELMASRI, R.; WUU, G.T.J. A temporal model and query language for EER databases. In: TANSEL, A.V. (Ed.). **Temporal Databases:** Theory, Design, and Implementation. Bridge Parkway: Benjamin/Cummings, 1993.

[HÜB 99] HÜBLER, P.N.; EDELWEISS, N.; CARVALHO, T.P. Implementação de um Banco de Dados Temporal Utilizando um SGBD Convencional. In: CONFERENCIA LATINOAMERICANA DE INFORMATICA - CLEI, 25., 1999. **Memorias...** Asuncion: Universidad Autonoma de Asuncion, 1999. V1, p 99-110.

[JEN 98] JENSEN, C.S. et al. The Consensus Glossary of Temporal Database Concepts - February 1998 Version. In: ETZION, O.; JAJODIA, S.; SPRIDA, S. (Eds.). **Temporal Databases Research and Practice.** Berlin: Springer-Verlag, 1998. p. 367-405.

[KOR 95] KORTH, Henry; SILBERSCHATZ, Abraham. **Sistema de Bancos de Dados**. 2. ed. São Paulo: Makron Books, 1995.

[LIN 93] LINCOLN, Cesa M.de O. **Incorporação da Dimensão Temporal em Bancos de Dados Orientados a Objetos**. Campinas: Universidade Estadual de Campinas, 1993. Dissertação de Mestrado.

[OLI 95] OLIVEIRA, J.P.M. et al. Implementation of an Object-Oriented Temporal Model. In: DATABASE AND EXPERT SYSTEMS APPLICATIONS CONFERENCE - DEXA, 6., 1995, London, U.K. **Proceedings...** [S.1.:s.n.], 1995.

[ORE 2000] ORES, Temporal Databases System. Disponível por www em http:// www-rocq.inria.fr/ verso/ publications /sigmod/ Theodoulidis: 1994:OTD.html, (2000).

[ÖZS 95] ÖZSOYOGLU, G.; SNODGRASS, R. T. Temporal and real-time databases: a survey. **IEEE Transactions on Knowledge and Data Engineering**, New York, v.7, n.4, p.513-532, Aug.1995.

[SAR 90] SARDA, N. L. Extensions to SQL for historical databases. **IEEE Transactions on Knowledge and Data Engineering**, New York, v.2, n.2, June 1990.

[SAR 93] SARDA, N. L. HSQL: A Historical query language. In: TANSEL, A.V. (Ed.). **Temporal Databases:** Theory, Design, and Implementation. Bridge Parkway: Benjamin/Cummings, 1993.

[SIM 98] SIMONETTO, Eugênio de Oliveira. **Uma Proposta para a Incorporação de Aspectos Temporais, no Projeto Lógico de Bancos de Dados, em SGBDs Relacionais.** Porto Alegre: Pontifícia Universidade Católica do Rio Grande do Sul. 1998. Dissertação de Mestrado.

[SNO 85] SNODGRASS, Richard Thomas; AHN, I. A Taxonomy of Time in Databases. In: ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, 1985, Texas. **Proceedings...** New York: ACM, 1985.

[SNO 87] SNODGRASS, R. The Temporal query language Tquel. **ACM Transactions on Database Systems,** New York, v.12, n.2, p.247-298, June 1987.

[SNO 93] SNODGRASS, R. An Overview of TQuel. In: TANSEL, A.V. (Ed.). **Temporal Databases:** Theory, Design, and Implementation. Bridge Parkway: Benjamin/Cummings, 1993.

[SNO 95] SNODGRASS, R. **The TSQL2 Temporal Query Language.** Boston: Kluwer Academic, 1995.

[TAN 93] TANSEL, A.U. et al. A Generalized Relational Framework for Modeling Temporal Data. In: TANSEL. A.V. (Ed.) **Temporal Databases**: Theory, Design, and Implementation. Bridge Parkway: Benjamin/Cummings, 1993.

[TAN 97] TANSEL, A.U., TIN, E. The expressive power fo temporal relational query languages. **IEEE Transaction on Knowledge and Data Engineering**, New York, v.9, n.1, Jan. 1997.

[TIM 99] TIME DB. **A Temporal Relational DBMS**. Disponível por www em http:// www.timeconsult. com /Software/ Software.html, (1999).

[TOO 99] TOOBIS. **Temporal Objects-Oriented Databases in Information Systems**. Disponível por www em http://www.di.uoa.gr/~toobis/ Description.html, (1999).

[UFR 91] UFRGS. INSTITUTO DE INFORMÁTICA. BIBLIOTECA. **Normas para Apresentação de Monografias ao CPGCC.** Porto Alegre: Instituto de Informática da UFRGS, 1991. 45p.

[WUU 93] WUU, Gene.T.J; DAYAL, Umeshwar. Uniform Model for Temporal and Versioned Object-Oriented Databases. In: TANSEL, A.V. (Ed.).

**Temporal Databases:** Theory, Design, and Implementation. Bridge Parkway: Benjamin/Cummings, 1993.

[ZAN 97] ZANIOLO, C. et al. **Advanced Database Systems**. San Francisco: Morgan Kaufmann, 1997. 574p.