# Anexo II
# Mapeamento entre modelos

Esta seção apresenta e caracteriza o processo de mapeamento entre o nível conceitual e o nível lógico e indica as alternativas consideradas mais convenientes considerando um esquema de fragmentação. Os passos de mapeamento aqui apresentados são baseados em regras gerais de mapeamentos encontradas em (AMBLER, 2000; AMBLER, 2001; BATINI *et al*., 1992; HEUSER, 2001)

## Motivação

O objetivo do mapeamento a ser apresentado é fazer refletir os aspectos semânticos presentes no esquema conceitual, inerentes a um sistema, para uma modelagem baseada no modelo Objeto-Relacional e relacional. Desta forma, será possível garantir um maior grau de fidelidade entre as diversas fases do projeto até o seu esquema de fragmentação final. Para realizar o mapeamento do nível conceitual para o nível lógico, três pontos principais foram abordados: mapeamento de métodos e atributos, mapeamento de classes e mapeamento de relacionamentos. Cada um destes pontos será apresentado e discutido separadamente nesta sessão.

Como o modelo Objeto-Relacional suporta todas as características do relacional, acrescido de aspectos do modelo OO, o seu mapeamento será um subconjunto dos passos aqui descritos, posteriormente explanado.

## Mapeamento Esquema Conceitual OO ➔ Esquema Relacional

Para realizar o mapeamento do esquema conceitual para o modelo relacional, é necessário analisar os diversos aspectos que evidenciam incompatibilidades entre estes modelos. Entre aqueles evidenciados em (CHAUDHRI & OSMON, 1997b; STONEBRAKER *et al*., 1990; STONEBRAKER & MOORE, 1995), podemos citar como aspectos diferenciadores: atributos e métodos, herança e relacionamentos entre classes ou tabelas. Estes aspectos necessitam ser analisados e, de acordo com a sua particularidade, um procedimento específico deve ser adotado para realização do mapeamento.

### Mapeamento de atributos e métodos

Para o mapeamento de atributos e métodos devem ser tratados os seguintes casos:

- **Atributos Multivalorados**

No caso do mapeamento de atributos multivalorados, é necessário criar uma tabela extra para armazenar todos os valores dos atributos multivalorados. Além disso, é necessário estabelecer um relacionamento *1* x *n* entre a tabela criada para o atributo multivalorado e a tabela que representa a classe vinda do esquema conceitual.

Como exemplo, podemos citar uma classe Pessoa com um atributo multivalorado Filhos. Este atributo seria uma lista de cadeias de caracteres contendo os nomes de todos os filhos daquela pessoa.

- **Objetos Complexos e Tipos de Dados Abstratos (ADT)**

Objetos complexos e Extensões de tipos de dados básicos (tipos de dados abstratos ou *Abstract Data Types - ADT* ) são formas de estruturação e armazenamento dos dados utilizados no modelo OO (CATTEL, 1994; CATTEL, 1999). Nestas extensões é possível aumentar o universo das funcionalidades permitidas, pois se elimina a simulação de dados complexos a partir do uso exclusivo de tipos de dados básicos, melhorando o desempenho do sistema como um todo. Outro ponto a ser destacado é a forma mais intuitiva e natural destas estruturas, tornando mais fácil o entendimento e a utilização das mesmas (STONEBRAKER *et al*., 1990).

Para realizar o mapeamento destes tipos de objetos, é necessário identificar e analisar as suas estruturas. Assim, deve-se decidir sobre a forma de mapeamento do mesmo. A abordagem mais usual é mapear o objeto em diversas colunas da tabela.Um exemplo disso seria um objeto complexo *Endereço* formado pelos seguintes atributos: *Rua, Número, Cidade, Estado*. Este objeto seria mapeado em quatro colunas adicionais na tabela do projeto lógico. Outro exemplo seria um atributo *Localização*, definido no esquema conceitual com dois atributos *latitude* e *longitude* do tipo inteiro e um método *Distância()* que retorna a distância entre dois objetos. Neste caso, o mapeamento seria restrito a criação dos campos latitude e longitude na tabela relacional, não sendo possível mapear o método diretamente. O mapeamento de métodos é detalhado no próximo tópico.

- **Métodos**

Para realizar o mapeamento de métodos, é necessário antes classificá-los. A depender desta classificação será decidida a forma de mapeamento. Para isto foram definidos dois grupos de métodos:

- Métodos que fazem acesso a atributos exclusivos da classe ou herdados de uma superclasse. Neste caso, os métodos podem ser mapeados em atributos derivados (atributos cujos valores dependem da manipulação dos valores existentes em outros atributos). Um exemplo seria um método *Idade()* que, através de um atributo *Data_nascimento*, retorna a idade de um determinado indivíduo. Para o seu mapeamento seria criado um atributo derivado *Idade*. No entanto, nada impede que sejam criadas funções para representar tais métodos. Neste ponto faz-se necessário analisar as implicações de ambas as alternativas. Ao implementar um campo calculado, deve-se levar em consideração o custo de mantê-lo atualizado e de que forma isto poderia interferir no desempenho do sistema. Esta decisão fica a cargo do projetista responsável pelo mapeamento.
- Métodos que necessitam fazer acesso a atributos ou métodos de outras classes. Este tipo de método deve ser mapeado em procedimentos armazenados (*Stored Procedures*) ou funções. As informações agora passam a ser obtidas através de consultas disparadas por estes mecanismos. Alguns SGBD Relacionais, como o Oracle (GIETZ, 2001), já dispõem de uma linguagem própria para acessos procedurais aos dados ou permitem utilizar internamente linguagens de alto nível como Java para a implementação destes mecanismos. Desta forma, o poder expressivo destes mecanismos é bastante aumentado.

### Mapeamento de classes para tabelas

Para realizar o mapeamento de classes, definidas num esquema conceitual, para tabelas, que fazem parte do modelo lógico/físico, o maior problema concentra-se na questão da herança entre as classes. Dentro da literatura consultada, existem três alternativas de mapeamento de hierarquias de classes para tabelas: Mapeamento de várias classes para uma única tabela; Uma tabela criada por classe concreta; Uma tabela criada por cada classe.

- **Mapeamento da hierarquia em somente uma tabela**

Nesta abordagem, todos os atributos de todas as classes que fazem parte de uma certa hierarquia existente no modelo são mapeados em somente uma tabela. Esta alternativa gera um grande desperdício de espaço, pois tuplas que representam uma determinada classe possuiriam atributos oriundo(s) de outras classe(s), não pertencendo ao seu escopo de utilização. Adicionalmente a isto, é necessário criar algum tipo de mecanismo que informe qual classe estará sendo instanciada em cada tupla da tabela. Para isto existem duas opções:

- Criar um campo que possuirá um domínio pré-definido de valores e informará qual classe está sendo armazenada naquela tupla da tabela (esta solução pode vir a gerar mais uma tabela para armazenar o domínio e mais um relacionamento)
- Criar colunas na estrutura da tabela que sirvam como indicadores (*flags*) da classe a que pertence o objeto armazenado na tupla. Por exemplo, numa hierarquia formada por Pessoa, Estudante e Professor, seriam criados três campos adicionais: um para indicar se a instância em questão é uma pessoa, um estudante ou um Professor.

Esta estratégia de mapeamento para hierarquias de classes ignora toda a identificação e levantamento das características de entidades participantes da aplicação. Impede-se a aplicação de predicados lógicos de forma localizada sobre as classes que realmente estão sendo consultadas pelas operações de seleção de forma natural.

Como o enfoque do trabalho não é o simples mapeamento para um modelo lógico, mas o mapeamento de um esquema conceitual que será mapeado para o modelo lógico e distribuído em seguida, seria muito difícil executar o mapeamento dos fragmentos utilizando-se somente uma tabela centralizada.

A FH somente se beneficiaria nesta abordagem caso existisse um predicado lógico sobre o campo que informa a classe sendo instanciada entre as tuplas. Como este tipo de predicado lógico nunca aconteceria na definição de consultas no nível conceitual sobre o modelo de classes, tal predicado não seria considerado.

- **Mapeamento de cada classe para uma tabela correspondente**

Nesta alternativa de mapeamento, todas as classes do modelo OO, a despeito do fato de serem concretas ou abstratas, são mapeadas para tabelas correspondentes. Assim, tabelas formadas estritamente com atributos de superclasses abstratas (não instanciáveis) seriam criadas junto às tabelas de subclasses compostas somente com atributos particulares a cada uma das subclasses. Relacionamentos artificiais teriam que ser criados para que todas as informações referentes a uma determinada tupla pudessem ser recuperadas. Um exemplo é dado através da Figura 27, na qual uma hierarquia de classes é mapeada para três tabelas. Nesta abordagem, para qualquer consulta requisitando o atributo *Name* e qualquer outro atributo em particular das tabelas *Employee* e *Student* seria obrigada a realizar uma junção com a tabela *Person*.

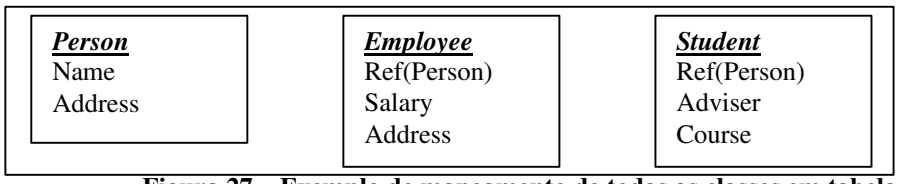


**Figura 27 – Exemplo de mapeamento de todas as classes em tabelas**

A grande dificuldade desta estratégia encontra-se na gerência das tabelas criadas e no desempenho do sistema. Será necessário definir uma grande quantidade de relacionamentos que não seriam naturais. Neste caso, haveria grande dificuldade por parte dos usuários em construírem consultas com junções entre várias tabelas. Para simples consultas a subclasses, numa árvore de herança com somente um nível, obrigatoriamente ter-se-ia que se fazer uma junção com a tabela da superclasse. À medida que a especialização aumenta dentro da hierarquia, maior será o número de junções a serem realizadas entre as classes que fazem parte da hierarquia (STONEBRAKER *et al*., 1990). Como o trabalho objetiva gerar esquemas de fragmentação que consigam melhorar o desempenho das aplicações que fazem acesso aos PDBDs, esta não seria a alternativa mais indicada.

- **Mapeamento somente das classes concretas para tabelas**

Nesta última abordagem, somente as classes concretas são mapeadas em tabelas específicas. Os campos em comum originários das superclasses são repetidos nas tabelas das classes concretas mapeadas. Embora haja a replicação destes campos comuns em várias tabelas, não haverá desperdício de espaço, pois os atributos específicos serão mapeados nas tabelas específicas.

Esta alternativa foi a que ofereceu maior compatibilidade com PDBDs, pois diminui o número de junções necessárias para recuperação dos dados, favorecendo um melhor desempenho da aplicação. Ao mesmo tempo, garante, ao contrário da primeira abordagem de mapeamento de hierarquias, a separação dos atributos nas tabelas correspondentes, permitindo que os predicados lógicos possam ser aplicados pela FH nas devidas tabelas.

Um dos grandes problemas desta alternativa reside no fato de desconsiderar a similaridade entre entidades. Desta forma, dados idênticos semanticamente terminam por serem mapeados em mais de uma tabela. A gerência de eventuais mudanças na hierarquia e no modelo torna-se também mais difícil, já que todas as tabelas deverão ser atualizadas, ao invés de realizar somente a atualização numa superclasse. Além disso, consultas envolvendo superclasses devem utilizar o operador de União entre as tabelas criadas a partir das subclasses. Embora este seja um custo adicional para a execução da consulta, uma operação de União é caracterizada por ser menos custosa que uma operação de Junção.

Esta abordagem é mais indicada para hierarquias onde as classes filhas possuem mais atributos que as superclasses (AMBLER, 2000; AMBLER, 2001). Com o crescimento da complexidade das aplicações, os objetos tendem a se especializar ainda mais, elevando o número de atributos específicos das subclasses, sendo esta a abordagem que mais beneficia esta tendência.

Entre as três abordagens analisadas, esta evidencia-se a que menos prejudica o PDBD e o desempenho da aplicação, sendo assim considerada a melhor escolha para o trabalho em questão e adotada para o desenvolvimento da avaliação. Ao mesmo tempo, esta alternativa pouco fere a modelagem conceitual, não se afastando da modelagem original.

### Mapeamento de Relacionamentos

O mapeamento de relacionamentos OO para o modelo relacional somente apresenta incompatibilidade no caso de relacionamentos *n* x *m*. Em todos outros casos – relacionamentos *1* x *n* e *1*x *1* - o mapeamento acontece naturalmente e de forma direta. Para que seja possível representar a semântica definida no esquema conceitual em relacionamentos *n* x *m*, faz-se necessária a criação de tabelas artificiais que servem para garantir o relacionamento vários para vários entre as classes do esquema conceitual e,

caso seja necessário, armazenar dados referentes a este relacionamento. Um exemplo para isto é o relacionamento entre *Estudante* e *Turma* (Figura 28).

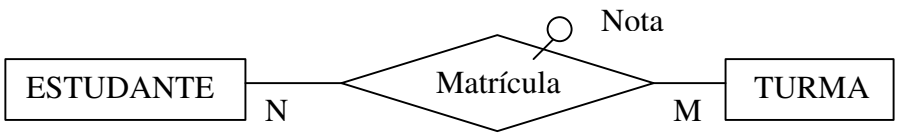


**Figura 28 – Exemplo de Relacionamento NxM**

Para que o mesmo possa ser implementado em um modelo lógico, é necessário criar uma terceira tabela (Matrícula) que armazene numa mesma tupla o identificador da tabela *Estudante* assim como o da tabela *Turma*. Um outro dado a ser armazenado na tabela criada para expressar o relacionamento seria a nota do estudante naquela turma, por exemplo (este dado, no modelo OO, poderia ser armazenado num atributo multivalorado da classe *Estudante*). Terá-se então como fruto do mapeamento as seguintes tabelas:

*ESTUDANTE(Inscrição,Nome,Telefone)  Turma(NoTurma, Sala, Horário,Matéria)*

*Matrícula(Inscrição, NoTurma, Nota)*

## Pontos de Diferença: OO x Relacional

Com este estudo sobre mapeamento, várias diferenças entre o modelo OO e o modelo relacional são evidenciadas, entre elas:

- Utilização de chaves identificadoras, relacionadas com o domínio da aplicação, para fornecer unicidade às tuplas das tabelas e permitir o relacionamento entre estas tabelas, ao invés de OIDS, identificadores artificiais, sem valor semântico para o sistema como um todo, mas um mecanismo que garante a unicidade entre qualquer objeto do sistema,
- Necessidade de criação de várias tabelas para eventualmente representar uma só classe,
- Criação de tabelas artificiais (sem significado semântico para o modelo) para suportar relacionamentos *n* x *m*, pois o modelo relacional não suporta referências múltiplas entre duas tabelas. Logo, faz-se necessário criar uma tabela extra que possa armazenar as chaves de ambas as tabelas envolvidas no relacionamento,

- Uso de Normalização do modelo para evitar redundância dos dados, já que não há suporte a atributos multivalorados como Conjuntos (*Sets*) e *Bags,*
- Necessidade de criação de campos extras para representar atributos complexos e/ou métodos,
- Uso de Procedimentos Armazenados e Funções para substituição de métodos.

## Mapeamento Esquema Conceitual OO ➔ Modelo OR

Como já citado, o mapeamento do modelo OO para o modelo Objeto-relacional é muito mais simples devido ao fato deste modelo suportar diversas características do esquema conceitual OO.

Um dos pontos de diferença entre estes modelos é a questão de relacionamentos *n* x *m* entre tabelas. Neste caso, o mesmo tratamento aplicado ao modelo relacional deve ser mantido para o mapeamento deste tipo de relacionamento no modelo OO para o modelo Objeto-Relacional. Ou seja, uma tabela artificial deve ser criada para armazenar as chaves das duas tabelas com múltiplos relacionamentos entre si.

Nos demais pontos que diferenciam o modelo relacional do OO, o modelo Objeto-Relacional oferece alternativas de formas intuitivas e naturais de mapeamento para os demais casos de impedância. É possível utilizar herança assim como atributos complexos, atributos multivalorados e métodos de forma direta.