

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ESPECIALIZAÇÃO EM WEB E SISTEMAS DE INFORMAÇÃO

FELIPE DALLE MOLLE

UML Data Modeling Profile

Trabalho de Conclusão apresentado como
requisito parcial para a obtenção do grau de
Especialista

Prof. Dr. Carlos Alberto Heuser
Orientador

Prof. Dr. Carlos Alberto Heuser
Coordenador do Curso

Porto Alegre, dezembro de 2007.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Reitor: Prof. José Carlos Ferraz Hennemann

Vice-Reitor: Prof. Pedro Cezar Dutra Fonseca

Pró-Reitora de Pós-Graduação: Profa. Valquiria Linck Bassani

Diretor do Instituto de Informática: Prof. Flávio Rech Wagner

Coordenador do WEBSIS: Prof. Carlos Alberto Heuser

Bibliotecária-Chefe do Instituto de Informática: Beatriz Regina Bastos Haro

AGRADECIMENTOS

Agradeço a Universidade Federal do Rio Grande do Sul, através do Instituto de Informática pela realização deste curso, que me proporcionou a oportunidade de aprender e estudar com grandes mestres em suas áreas de conhecimento. Ao Professor Dr. Carlos Alberto Heuser, por aceitar me orientar neste trabalho. Seu esforço e dedicação, apresentados durante as várias aulas ao longo do curso, são dignos dos verdadeiros mestres e um exemplo a ser seguido. Ao colega Ricardo Cavedini, pela colaboração e paciência durante os infindáveis trabalhos que fizemos em conjunto ao longo destes dois anos de curso. A Professora Naira Maira Balzaretti, do Instituto de Física desta universidade, pelo seu auxílio com as sugestões didáticas e pontuais durante a definição do modelo a ser empregado no estudo de caso deste trabalho.

Por fim, agradeço a minha família. A minhas filhas Nathalie e Nicole pela tolerância as restrições causadas durante este período de estudos. Saibam que este curso foi uma realização pessoal muito importante para mim. Espero sempre poder retornar todo o carinho que vocês merecem. Lembrem: “Ler e Estudar” são coisas muito importantes a serem feitas durante toda a vida. A minha amada esposa Naira, agradeço simplesmente por tudo. Impossível seria quantificar, tudo o que eu teria de expressar. Portanto, como sei que gostas de resolver criptogramas, e como prova da minha gratidão, abaixo tens um para te divertires por um bom tempo.

“NaameLtAmEaIlDMERyVLMEmAvMeFJTAmEVtyEmEllyMEIeeTmEjAMEbOtM
ELtAmELbmENeedMEPeTaMESjAdMETmTmETssMEyIldMeZmTwmeDjeDmEjKt
MeKTYME“¹

¹ Grato Professor Weber por ensinar as técnicas.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS.....	5
LISTA DE FIGURAS.....	6
LISTA DE TABELAS.....	7
LISTA DE QUADROS.....	9
RESUMO.....	10
ABSTRACT.....	10
1 INTRODUÇÃO.....	11
2 MODELO FISICO DE DADOS.....	14
3 APRESENTANDO O UML PROFILE.....	15
4 ESTUDO DE CASO: UNIVERSIDADE.....	26
4.1 Modelagem de Tabelas.....	26
4.2 Complementando o Modelo Físico.....	31
4.3 Modelo Físico Universidade.....	36
4.4 Análise do Modelo Físico Universidade.....	38
5 CONCLUSÃO.....	44
REFERÊNCIAS.....	45

LISTA DE ABREVIATURAS E SIGLAS

UML	Unified Modeling Language
ER	Entity-Relationship
OMG	Object Management Group
XML	Extensible Markup Language
XMI	XML Metadata Interchange
SGBD	Sistema de Gerenciamento de Banco de Dados
LDM	Logical Data Model
PDM	Physical Data Model
OODB	Object-Oriented Database
ORDB	Object-Relational Database
PK	Primary Key
FK	Foreign Key
AK	Alternate Key
OID	Object Identifier
OCL	Object Constraint Language
Int	Integer
Char	Character
SQL	Structure Query Language
CEP	Código de Endereço Postal
OO	Object-Oriented
ORM	Object-Relational Model
OODB	Object-Oriented Database
ORDB	Object-Relational Database
RDB	Relational Database
BD	Banco de Dados

LISTA DE FIGURAS

Figura 1.1: Resumo do modelo físico de dados proposto	13
Figura 3.1: Exemplo da representação de uma tabela	16
Figura 3.2: Exemplo da representação de uma View	17
Figura 3.3: Exemplo de representação de índices	18
Figura 3.4: Exemplo da representação de uma <i>Store procedure</i>	19
Figura 3.5: Exemplo de representação de chaves de uma tabela	20
Figura 3.6: Exemplo da representação de relacionamentos	23
Figura 3.7: Modelagem de restrições	25
Figura 4.1: Tabela desenvolvida com Argo UML.....	27
Figura 4.2: Tabela com a definição de suas chaves.....	28
Figura 4.3: Definição de <i>Tagged Values</i> no Argo UML.....	29
Figura 4.4: Definição de relacionamentos entre tabelas no Argo UML.....	30
Figura 4.5: Modelo físico de dados Universidade.....	32
Figura 4.6: Modelo físico de dados desenvolvido com Microsoft Office Visio 2007 ...	37
Figura 4.7: Tabelas Alunos e Cursos do modelo físico Universidade.....	39
Figura 4.8: Modelagem do índice associado à tabela Alunos	39
Figura 4.9: Modelagem de tabela associativa.....	40
Figura 4.10: Modelagem de chaves compostas	41
Figura 4.11: Modelagem de uma <i>View</i>	42
Figura 4.12: Modelagem de <i>Store Procedures</i>	43

LISTA DE TABELAS

Tabela 3.1: Estereótipos referentes aos tipos de modelos	15
Tabela 3.2: Estereótipos referentes aos tipos de Armazenamento	15
Tabela 3.3: Estereótipos empregados com <i>Class Boxes</i>	19
Tabela 3.4: Estereótipos empregados em colunas de uma tabela.....	21
Tabela 3.5: <i>Tagged values</i> empregados em colunas de uma tabela	21
Tabela 3.6: Exemplos de chaves de uma tabela	22
Tabela 3.7: Estereótipos empregados em associações entre tabelas	22
Tabela 3.8: Significado das multiplicidades.....	23

LISTA DE QUADROS

Quadro 4.1: Comandos em SQL necessários para criar a Tabela Alunos.....	28
Quadro 4.2: Script SQL para criação das tabelas Alunos e Cursos.....	30
Quadro 4.3: Parte do programa fonte XMI2SQL.CS desenvolvido em C Sharp.....	33
Quadro 4.4: Script SQL para criação das tabelas do modelo físico	33

RESUMO

O objetivo deste trabalho é apresentar o *profile* desenvolvido por Scott W. Ambler para modelagem física de dados com UML. No transcorrer do trabalho foram caracterizados os elementos que compõem este *profile* e o seu emprego apropriado durante a construção de um modelo físico. Também foram desenvolvidos alguns exemplos, sendo que parte destes foram submetidos à ferramenta XMI2SQL. Ela tem a capacidade de interpretar modelos construídos segundo as definições do *profile*, e a partir de arquivos gerados no padrão XMI, criar *scripts* em SQL que podem ser utilizados na montagem da estrutura de tabelas do banco de dados. Finalizando é apresentado um estudo de caso baseado no *profile*, no qual foi feita a modelagem física de parte de um sistema informatizado de uma universidade.

Palavras-Chave: UML, UML *Profile* para modelagem de dados, modelo físico de dados, XMI, SQL.

UML Data Modeling Profile

ABSTRACT

The goal of this work is to present the profile developed by Scott W. Ambler for UML physical data modeling. The elements of this profile were characterized during this work, together with the correct way of constructing a physical data model. Some examples were also developed and submitted to XMI2SQL tool. This tool is able to read XMI files developed in agreement with the profile and it generates SQL scripts that can be used to build the database. Finally, a physical model related to the information system of a university is presented as a study case based on the profile.

Keywords: UML, UML *Profile* for data modeling, Physical Data Model, PDM, XMI, SQL.

1 INTRODUÇÃO

Durante o ciclo de desenvolvimento de softwares são utilizados vários tipos de ferramentas as quais têm a finalidade de auxiliar e apoiar todo este processo. Em muitos casos, ocorre que cada profissional que trabalha nas diversas fases do desenvolvimento utiliza um tipo adequado de ferramenta para aquela fase. Este profissional cria os artefatos necessários para dar forma ao sistema e disponibiliza os mesmos para a fase seguinte. Neste momento de transição podem ocorrer perdas importantes de informações devido à falta de integração entre as ferramentas utilizadas. Considerando um caso de desenvolvimento de sistemas em que é utilizada uma ferramenta UML (Unified Modeling Language – Linguagem de Modelagem Unificada) durante a fase de análise de sistemas e na transição para a fase de projeto de sistemas, é utilizada uma ferramenta de modelagem de dados baseada em ER (Entity-Relationship – Entidade-Relacionamentos). Neste caso, se não houver uma forma na qual as informações sejam transferidas de um modelo para outro, todo o trabalho de modelagem feito durante a análise de sistemas teria que ser refeito na fase de projeto. Um exemplo deste tipo de transferência seria a transformação da estrutura de classes feita na análise para uma estrutura de banco de dados. Neste momento podem ocorrer perdas de informações ou desvios, que por consequência, podem prejudicar todo o desenvolvimento do sistema, e até mesmo, impedir que seja obtido o resultado esperado.

Um dos problemas em se utilizar UML para representar banco de dados é que originalmente ele não foi projetado para este fim. UML é uma ferramenta de modelagem orientada a objetos e não uma ferramenta de projeto de banco de dados. Bancos de dados, na sua grande maioria, são baseados em Entidades e Relacionamentos, e existem no mercado excelentes ferramentas para executar sua modelagem.

Vários pesquisadores já propuseram seus próprios métodos para modelagem de dados em UML (NAIBURG e MAKSIMCHUK, 2001; MULLER, 1999; AMBLER, 2003), bem como vários fabricantes de ferramentas UML desenvolveram em seus produtos a capacidade de preencher esta lacuna. O grande problema é a falta de uma definição, isto é, um padrão definitivo que determine a maneira única de fazer a modelagem de dados com UML.

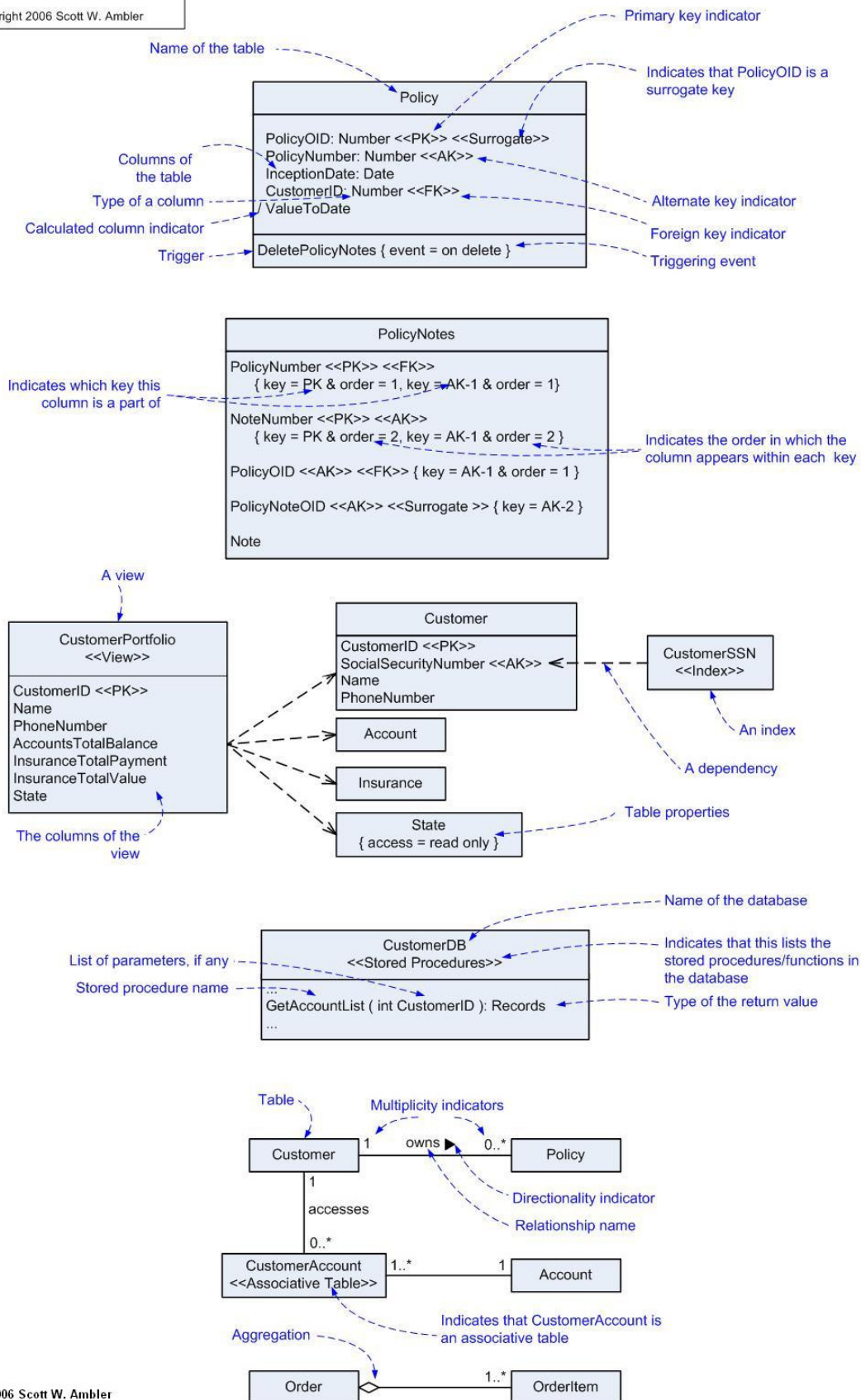
Com o objetivo de tentar definir um padrão, no final do ano de 2005, o *Object Management Group* (OMG) iniciou um *Request for Proposal – Information Management Metamodel* (OMG, 2005), sendo que parte deste abrange as sugestões de propostas para definição de um *UML Profile for Data Modeling*. Um *profile* para UML é um mecanismo que fornece uma ampliação genérica para a construção de modelos em um domínio particular. Neste caso, modelagem de dados.

O objetivo deste trabalho é apresentar o modelo desenvolvido por Scott W. Ambler (AMBLER, 2006B), cuja proposta está sendo avaliada pela própria OMG e poderá servir de base para esta especificação.

A abordagem será feita com base em exemplos desenvolvidos em UML, sendo que alguns serão submetidos à ferramenta XMI2SQL que foi desenvolvida por Eric R. Hartford (HARTFORD, 2003). A ferramenta XMI2SQL foi programada para ler e interpretar arquivos do padrão XML *Metadata Interchange* (XMI), gerando como saída código em *Structured Query Language* (SQL). Arquivos XMI são utilizados como um padrão de intercâmbio entre ferramentas UML.

O padrão proposto por Scott W. Ambler para modelagem de dados com UML vem sendo desenvolvido há bastante tempo. Desde 1997 em suas publicações, ele aborda a necessidade da criação de um modelo de dados para UML. Na figura 1.1 temos uma visão geral da proposta de Scott W. Ambler, onde é apresentado um sumário para o modelo físico de dados em UML.

**Notation Summary:
UML Physical Data Model**
Copyright 2006 Scott W. Ambler



Copyright 2006 Scott W. Ambler

Figura 1.1: Resumo do modelo físico de dados proposto (AMBLER, 2006b).

2 MODELO FISICO DE DADOS

O propósito do modelo físico é servir de especificação para a construção de um banco de dados, portanto nele devem estar contidos os detalhes das estruturas de dados, dos relacionamentos e restrições. O modelo físico de dados é empregado para projetar os esquemas internos que compõem um banco de dados, sendo que nele devem ser descritas as tabelas, as colunas destas tabelas e os relacionamentos entre essas tabelas.

Ao se desenvolver um modelo físico, deve-se levar em consideração qual a arquitetura do banco de dados e o ambiente de *hardware* no qual o banco de dados será utilizado. De uma forma geral, é importante considerar quais as características e capacidades do Sistema de Gerenciamento de Banco de Dados (SGBD) (PONNIAH, 2007).

3 APRESENTANDO O UML PROFILE

A seguir é apresentado em detalhes o *profile* para modelagem de dados em UML (AMBLER, 2006b), que é o objeto de estudo deste trabalho. O primeiro elemento que deve ser especificado é o tipo de modelo que está sendo elaborado. Isto pode ser feito através de um texto descritivo em um elemento Nota da UML ou através de um estereótipo, conforme descrição feita na tabela 3.1.

Tabela 3.1: Estereótipos referentes aos tipos de modelos.

Estereótipo	Modelo
<<Class Model>>	Modelo OO ou ORM
<<Conceptual Data Model>>	Modelo Conceitual de dados
<<Domain Model>>	Modelo de Domínio
<<Logical Data Model>>	Modelo Lógico de dados (LDM)
<<Physical Data Model>>	Modelo Físico de dados (PDM)

No caso do Modelo Físico de Dados, pode-se também definir o tipo de armazenamento conforme apresentados na tabela 3.2.

Tabela 3.2: Estereótipos referentes aos tipos de Armazenamento.

Estereótipo	Tipo de Armazenamento
<<File>>	Arquivo
<<Hierarchical Database>>	Banco de dados Hierárquico
<<Object-Oriented Database>>	Banco de dados Orientado a objetos (OODB)
<<Object-Relational Database>>	Banco de dados Objeto Relacional (ORDB)
<<Network Database>>	Banco de dados em rede
<<Relational Database>>	Banco de dados Relacional (RDB)
<<XML Database>>	Banco de dados XML

Tabelas de um banco de dados são representadas através de *Class Boxes*. Estas podem ser divididas em três seções. Na primeira seção deve constar o nome da tabela e abaixo deste o estereótipo <<Table>>. O emprego do estereótipo <<Table>> no modelo físico de dados é opcional, pois a finalidade principal deste modelo é a definição das tabelas. Na segunda seção, são descritas as colunas da tabela onde também é possível especificar o seu tipo de dado. Se uma coluna faz parte de uma ou mais chaves da tabela, é importante associar um estereótipo cuja finalidade é a de definir o tipo de chave. Alguns exemplos destes estereótipos são: <<PK>>, <<FK>> e <<AK >>. Na terceira seção são definidos os *triggers* associados às tabelas. Abaixo, na figura 3.1, pode-se visualizar a representação de uma tabela e seus componentes de acordo com este *profile*.

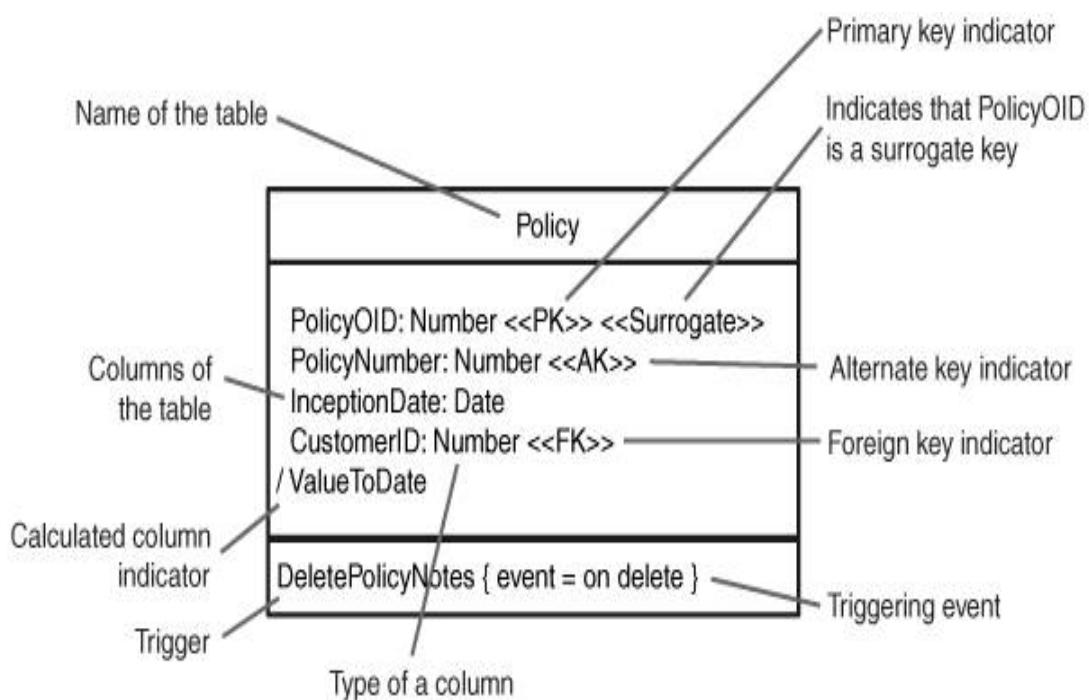


Figura 3.1: Exemplo da representação de uma tabela (AMBLER, 2006a).

Class Boxes também são utilizados na representação de *Views*. Deve ser levado em conta o emprego obrigatório do estereótipo <<View>>, cuja finalidade é facilitar a distinção entre as tabelas, pois nestas, o uso do estereótipo <<Table>> não é obrigatório. Na figura 3.2, pode-se visualizar a representação de uma *View* e sua composição através de informações extraídas de quatro tabelas.

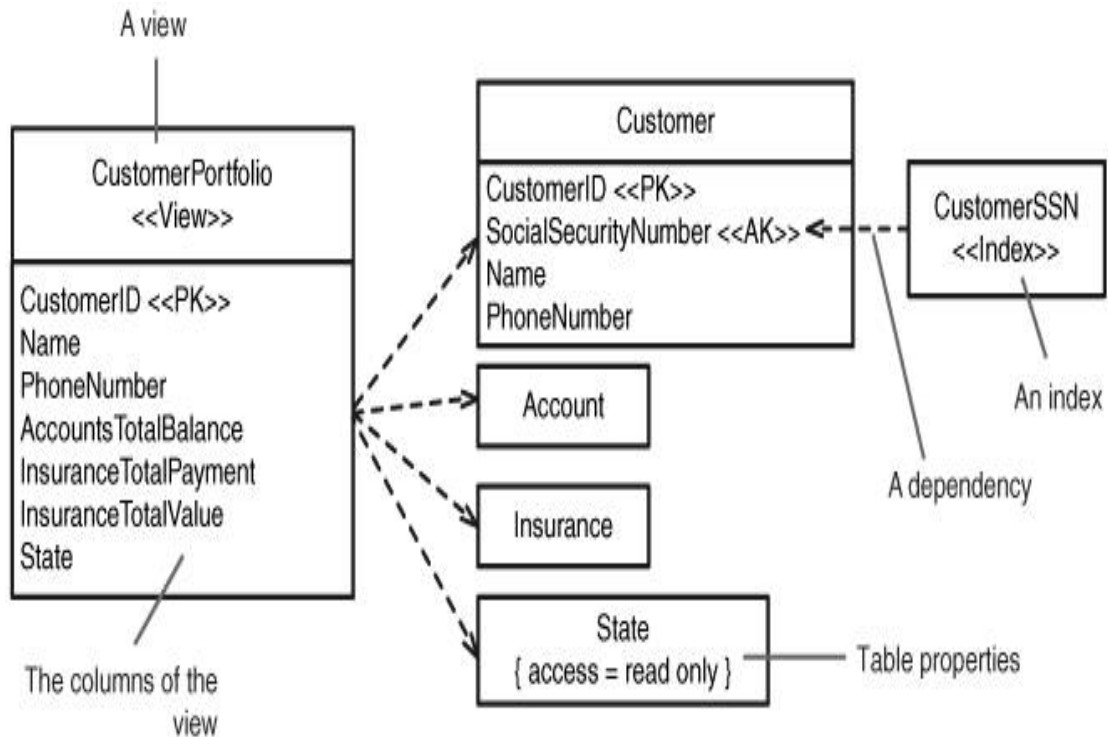


Figura 3.2: Exemplo da representação de uma View (AMBLER, 2006a).

A representação de índices também é feita através de *Class Boxes*. Deve-se nomear o índice na primeira seção do *Class Box* e empregar o estereótipo <<Index>>. A dependência com relação à coluna que compõe um índice pode ser expressa de dois modos. O primeiro modo seria descrevendo a coluna de dependência na segunda seção do *Class Box* e fazer uma ligação com a tabela. O outro modo seria fazendo uma ligação de dependência entre o *Class Box* que representa o índice e conectá-lo diretamente na coluna da tabela que é utilizada para a formação do índice. Na figura 3.3 pode-se verificar o emprego das duas técnicas.

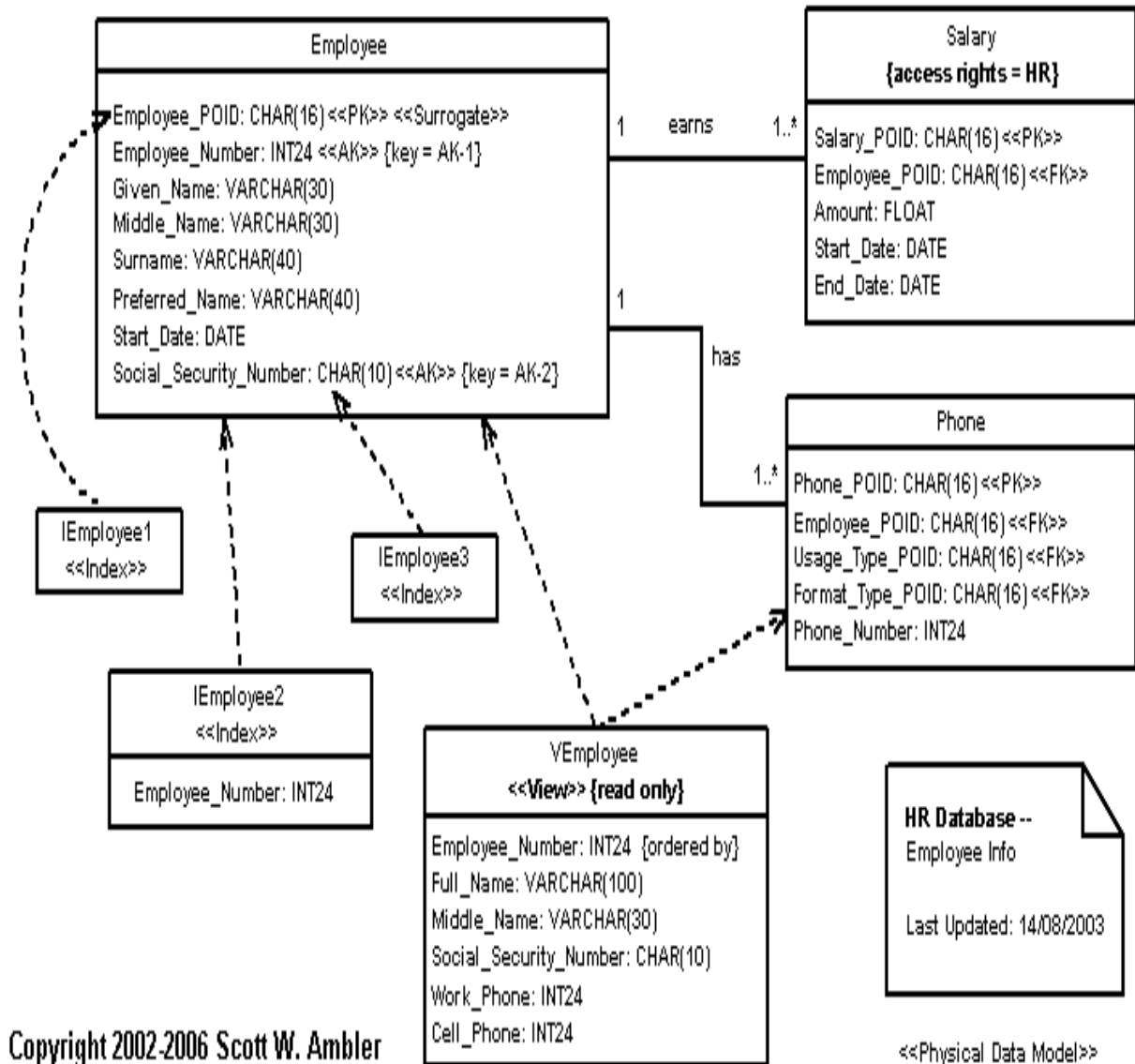


Figura 3.3: Exemplo de representação de índices (AMBLER, 2006b).

Da mesma maneira que os elementos anteriores, as *Stored Procedures* são representadas através de *Class Boxes*. Na primeira seção do *Class Box* deve ser informado um nome para *Stored Procedure* e também deve ser empregado o estereótipo <<Stored Procedures>>. A próxima seção é utilizada para descrever a lista de *Stored Procedures* ou funções, sendo que devem ser especificados os parâmetros de entrada e o tipo de retorno gerado. As *Stored Procedures* podem estar associadas a um banco de dados ou dentro de um pacote (*package*) de um banco de dados. Esta distinção pode ser feita através de um sufixo ao nome atribuído à *Stored Procedure* na primeira seção. Na figura 3.4 está exemplificada uma *Stored Procedure* de acordo com o proposto por este *profile*.

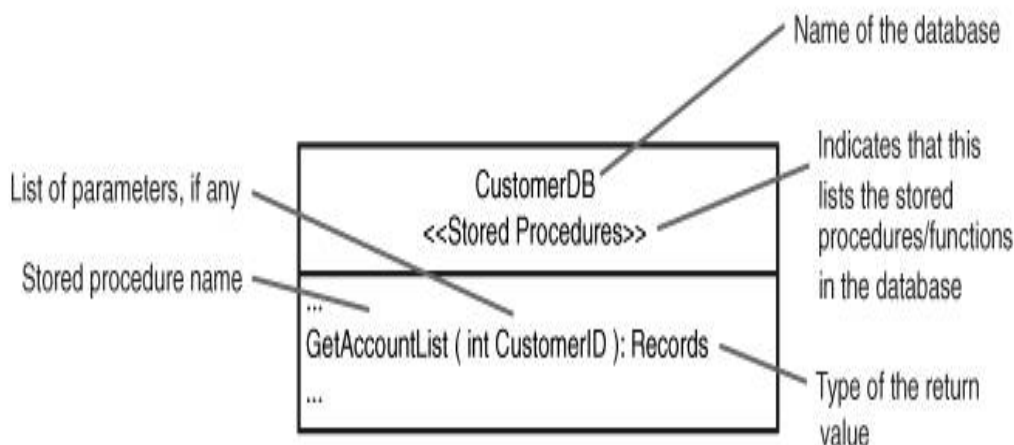


Figura 3.4: Exemplo da representação de uma *Store Procedure* (AMBLER, 2006a).

A seguir é apresentada a tabela 3.3, onde consta uma lista de estereótipos que podem ser utilizados no modelo físico de dados em associação com o elemento *Class Box* da UML

Tabela 3.3: Estereótipos empregados com *Class Boxes*.

Estereótipo	Aplicação
<<Associative Table>>	Na representação de tabelas associativas.
<<Index>>	Na representação de índices.
<<Lookup Table>>	Na definição de tabelas do tipo <i>Lookup Table</i> .
<<Stored Procedures>>	Na definição de <i>Stored Procedures</i> .
<<Table>>	Na representação uma tabela.
<<View>>	Na Definição uma <i>View</i> .

A representação das chaves de uma tabela é feita através de estereótipos e também por intermédio de *tagged values*. *Tagged values* são conjuntos definidos por expressões do tipo “nome = valor”. Os estereótipos são utilizados para definir o tipo de chave, como por exemplo, no caso da determinação de uma chave primária de uma tabela, deve-se utilizar o estereótipo <<PK>> (*Primary Key*). Outro exemplo é a determinação de chaves estrangeiras onde o estereótipo <<FK>> (*Foreign Key*) é o utilizado. Na figura 3.5 pode-se verificar a aplicação destes estereótipos, além de outros.

As *tagged values* são utilizadas para aumentar o nível de detalhamento das chaves da uma tabela, informando o seu tipo e como é feita a composição das mesmas. A composição de uma chave é feita através da especificação da sua ordem. Analisando a figura 3.5, observa-se que a chave primária (PK) é composta por duas colunas. A primeira parte da composição é feita pela coluna *PolicyNumber* e isto é indicado pela

tag value “{key = PK & order = 1, ...}”. A parte final da chave é formada pela coluna *NoteNumber* e está indicada pela *tag value* “{key = PK & order = 2, ...}”.

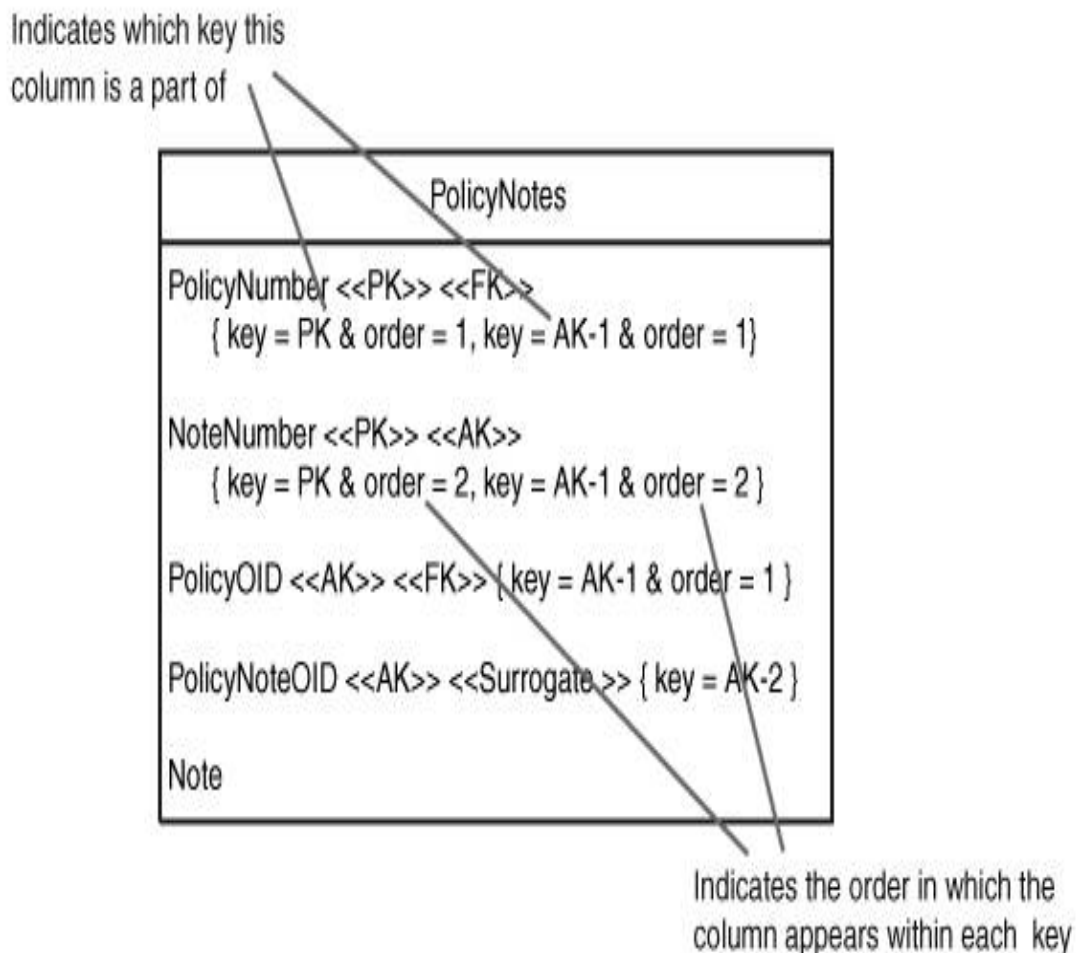


Figura 3.5: Exemplo de representação de chaves de uma tabela (AMBLER, 2006a).

Na tabela 3.4 abaixo, consta a relação de estereótipos que podem ser empregados em conjunto às colunas de uma tabela.

Na tabela 3.5 são apresentadas as *Tagged Values* utilizadas para modelar as chaves de uma tabela. Segundo o autor, estas *Tagged Values* devem ser empregadas como uma notação suplementar, isto é, seu uso não é obrigatório. A intenção é elucidar de forma mais precisa a formação das chaves definidas em uma tabela, principalmente as chaves compostas por duas ou mais colunas.

Tabela 3.4: Estereótipos empregados em colunas de uma tabela.

Estereótipo	Aplicação
<<AK>>	Indicar que a coluna faz parte de uma chave alternada (chave secundária).
<<Auto Generated>>	Indicar que o valor da coluna é gerado automaticamente.
<<Column>>	Define que um atributo é uma coluna (redundante)
<<FK>>	Indicar que a coluna é parte de uma chave estrangeira.
<<Natural>>	Indica que a coluna, devido a suas características, é uma chave natural.
<<Not Null>>	Indica que a coluna não pode conter um valor nulo.
<<Nullable>>	Indica que a coluna pode conter um valor nulo.
<<PK>>	Indica que a coluna é parte da chave primária de uma tabela.
<<Surrogate>>	Indica que a coluna é uma chave substituta.

Tabela 3.5: *Tagged values* empregados em colunas de uma tabela.

Tag	Aplicação	Exemplos
Key	Indicar o tipo de chave que está sendo representada.	key = PK key = FK key = AK-2
Order	Indicar a ordem da uma coluna em uma chave composta.	order = 3
Source column	Indicar a coluna de origem a qual uma chave estrangeira faz referência. Utilizado quando as colunas das tabelas têm nomes diferentes.	source column = IDFunc
Table	Indicar a qual tabela a chave estrangeira faz referência.	table = Funcionários

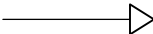


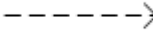

Analisando a figura 3.5, referente à tabela *PolicyNotes*, conclui-se que a composição das chaves pode ser expressa conforme os exemplos descritos na tabela a seguir.

Tabela 3.6: Exemplos de chaves de uma tabela.

Tipo de Chave	Colunas	Tag Value
Chave Primária (PK)	<i>PolicyNumber</i>	{key = PK & order = 1}
	<i>NoteNumber</i>	{key = PK & order = 2}
Primeira Chave Alternada (AK-1)	<i>PolicyOID</i>	{key = AK-1 & order = 1}
	<i>NoteNumber</i>	{key = AK-1 & order = 2}
Segunda Chave Alternada (AK-2)	<i>PolicyNoteOID</i>	{key = AK-2}

Os relacionamentos entre tabelas, também chamados de associações, são representados por linhas sólidas entre as mesmas. Este *profile* define uma série de estereótipos que podem ser utilizados durante a modelagem, porém, o próprio autor recomenda a utilização da representação visual dos mesmos sempre que possível. Na tabela 3.7 estão definidos os estereótipos que podem ser empregados para representar os relacionamentos entre tabelas.

Tabela 3.7: Estereótipos empregados em associações entre tabelas.

Estereótipo	Representação Gráfica	Aplicação
<<Subtype>>		Indicar subtipo/supertipo ou um relacionamento de herança.
<<Aggregation>>		Indicar um relacionamento do tipo agregação.
<<Composition>>		Indicar um relacionamento do tipo composição.
<<Dependency>>		Indicar dependência de uma <i>view</i> ou um índice em relação a uma tabela.
<<Identifying>>		Indicar a relação de identificação entre duas tabelas dependentes.
<<Uni-directional>>		Indicar a única direção do relacionamento entre duas tabelas.
<<Non-Identifying>>		Indicar a relação de não identificação entre duas tabelas independentes.

Na figura 3.6 pode-se ver o emprego da representação gráfica dos relacionamentos entre tabelas. Nela também estão representadas as multiplicidades que ocorrem entre as tabelas. Como exemplo de multiplicidades, a tabela *Customer* pode ter

zero ou muitos registros na tabela *Policy*, da mesma forma que um registro na tabela *Policy* sempre está relacionado a apenas um registro da tabela *Customer*.

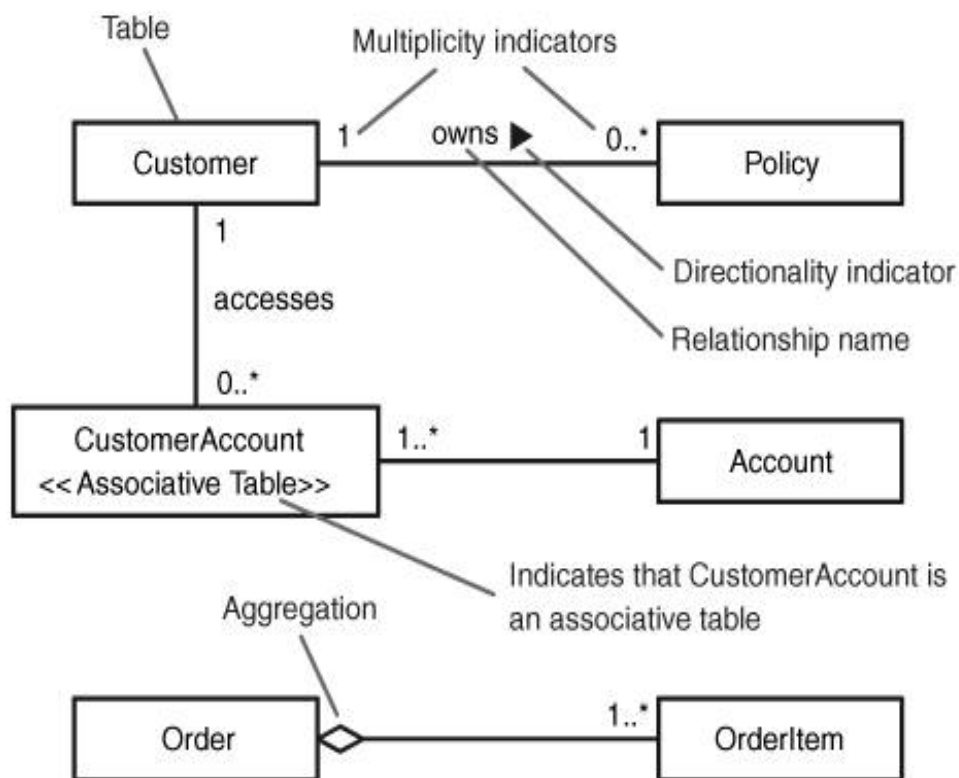


Figura 3.6: Exemplo da representação de relacionamentos (AMBLER, 2006a).

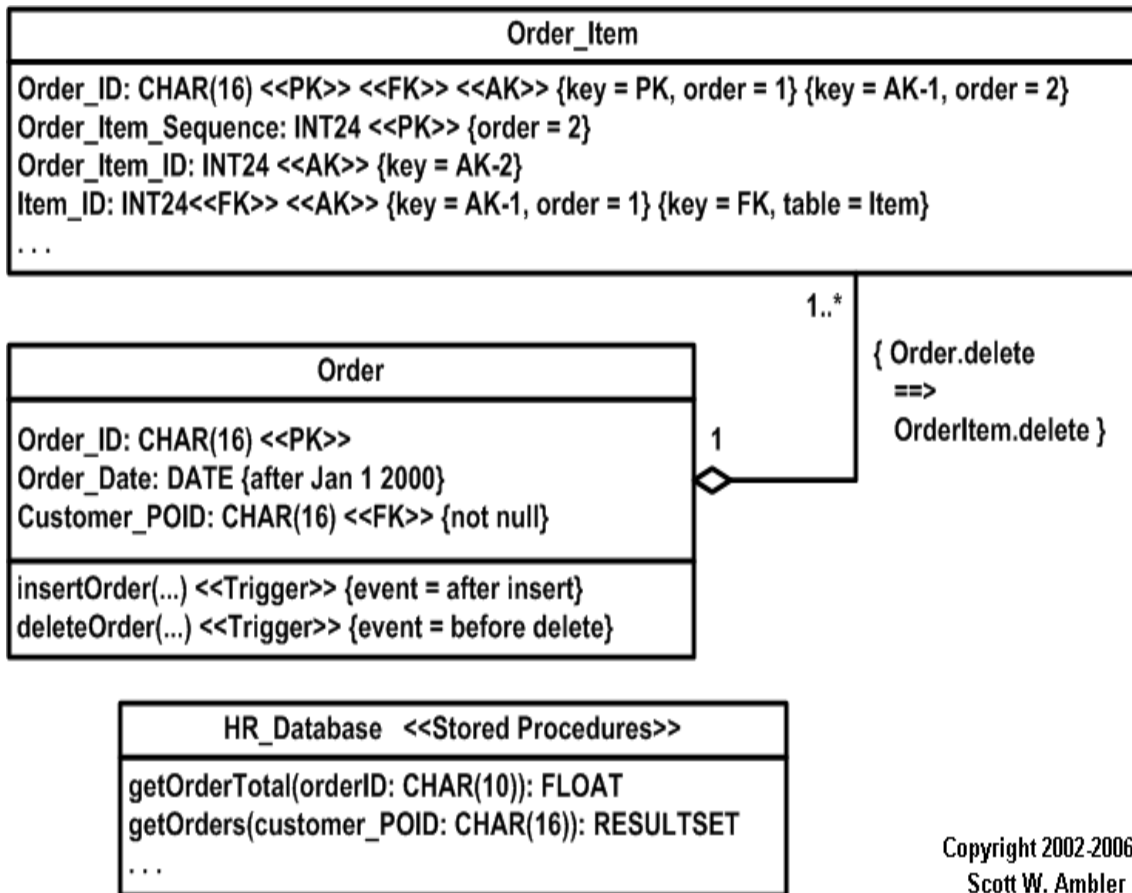
Na tabela 3.8 constam as relações de multiplicidades que podem ser empregadas nos relacionamentos entre tabelas.

Tabela 3.8: Significado das multiplicidades.

Multiplicidade	Significado
0..1	Zero ou um
1	Somente um
0..*	Zero ou mais
1..*	Um ou mais
*	Um ou mais
N	Somente n (onde n > 1)

Multiplicidade	Significado
0..n	Zero a n (onde $n > 1$)
1..n	Um a n (onde $n > 1$)

A especificação de restrições é feita com a utilização da *Object Constraint Language* (OCL). A OCL é uma linguagem declarativa, através da qual são feitas as descrições de regras que se aplicam aos modelos (AMBLER, 2004). Na figura 3.7 existem vários exemplos de utilização de restrições. Na tabela *Order*, está definida na coluna *Order_Date* a restrição “*{after Jan 1 2000}*” que deve ser interpretada como: só devem ser aceitas datas posteriores a 1º de Janeiro de 2000 no campo *Order_Date*. Outro exemplo de utilização de OCL para definir restrições pode ser verificado no relacionamento entre as tabelas *Order* e *Order_Item*, cuja declaração “*{Order.delete==>OrderItem.delete}*” significa que, quando um registro da tabela *Order* for excluído, todos os registros associados a este na tabela *Order_Item*, também devem ser excluídos. Cabe observar que a própria representação gráfica do relacionamento através de uma agregação já tem implícito tal comportamento, porém a especificação de restrições com OCL pode elucidar eventuais ambigüidades que o modelo possa deixar transparecer. Na figura 3.7 pode-se observar também a aplicação de restrições com os *Triggers* definidos para a tabela *Order*. Foram definidos dois *Triggers*: *insertOrder(...)* e *deleteOrder(...)*. O primeiro deverá ser executado após a inserção de um registro na tabela *Order*, conforme definido pela restrição em OCL “*{event = after insert}*”. Já o segundo deverá ser executado antes da exclusão de um registro na tabela *Order*, conforme definido pela restrição em OCL “*{event = before delete}*”.



Copyright 2002-2006
Scott W. Ambler

Figura 3.7: Modelagem de restrições (AMBLER, 2006b).

4 ESTUDO DE CASO: UNIVERSIDADE

A partir de agora será desenvolvido um estudo de caso baseado na utilização dos artefatos que compõem o *profile* proposto. Com base no mesmo, será feita a modelagem física de dados de parte de um sistema de uma universidade, iniciando com apenas uma tabela e, na seqüência, agregando outras, bem como os demais elementos necessários para a composição do modelo. O objetivo é exemplificar o uso do *profile*. Para a elaboração gráfica do modelo físico de dados, foram utilizadas as ferramentas Argo UML e Microsoft Office Visio 2007. Alguns dos exemplos construídos também foram submetidos à ferramenta XMI2SQL, através da exportação de arquivos no padrão XMI que foram gerados a partir do próprio Argo UML. A ferramenta XMI2SQL se mostrou bastante limitada no seu propósito de gerar arquivos SQL tendo como base arquivos XMI. Apenas poucas características do *profile* são aceitas pela ferramenta XMI2SQL, ou seja, o *profile* não está totalmente implementado na ferramenta. Porém é válida a sua utilização neste trabalho, pois serve como uma prova da aplicação do padrão proposto para modelagem física de dados.

4.1 Modelagem de tabelas

Conforme apresentado no início do trabalho, a modelagem de uma tabela de acordo com o *profile* definido por Scott W. Ambler, é feita através de um *Class Box* com o estereótipo <<Table>>, como apresentado na figura 4.1 abaixo:

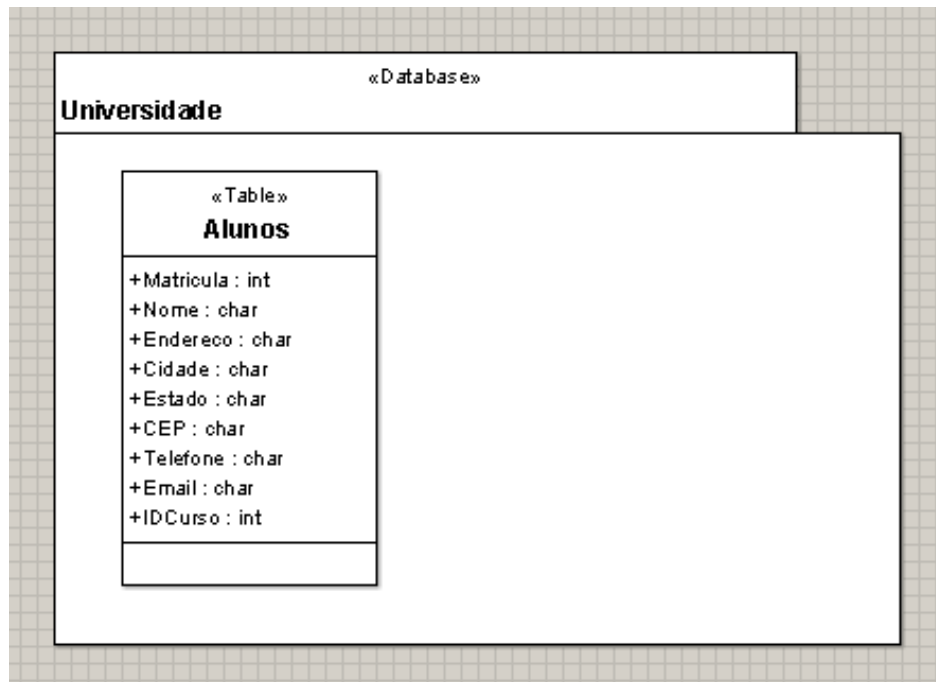


Figura 4.1: Tabela desenvolvida com Argo UML.

As colunas de uma tabela são representadas pela notação padrão de atributos, sendo que os seus tipos são definidos de acordo com o seu conteúdo. Normalmente os tipos são especificados como inteiros (*int*), caracteres (*char*), data (*date*) ou lógico (*boolean*), podendo ser utilizados outros tipos de acordo com o suportado pelo sistema de banco de dados.

Segundo o *profile*, um banco de dados deve ter a mesma representação gráfica de um *Package*, com a aplicação do estereótipo <<Database>>. A mesma representação deve ser utilizada para a criação de seções em bancos de dados. Neste caso, os estereótipos devem refletir a nomenclatura de seções definida por cada fabricante. Exemplos destes estereótipos seriam <<Tablespace>> e <<Partition>>.

Na figura a seguir, é apresentada a modelagem da chave primária de ordenação e também da chave estrangeira da tabela.

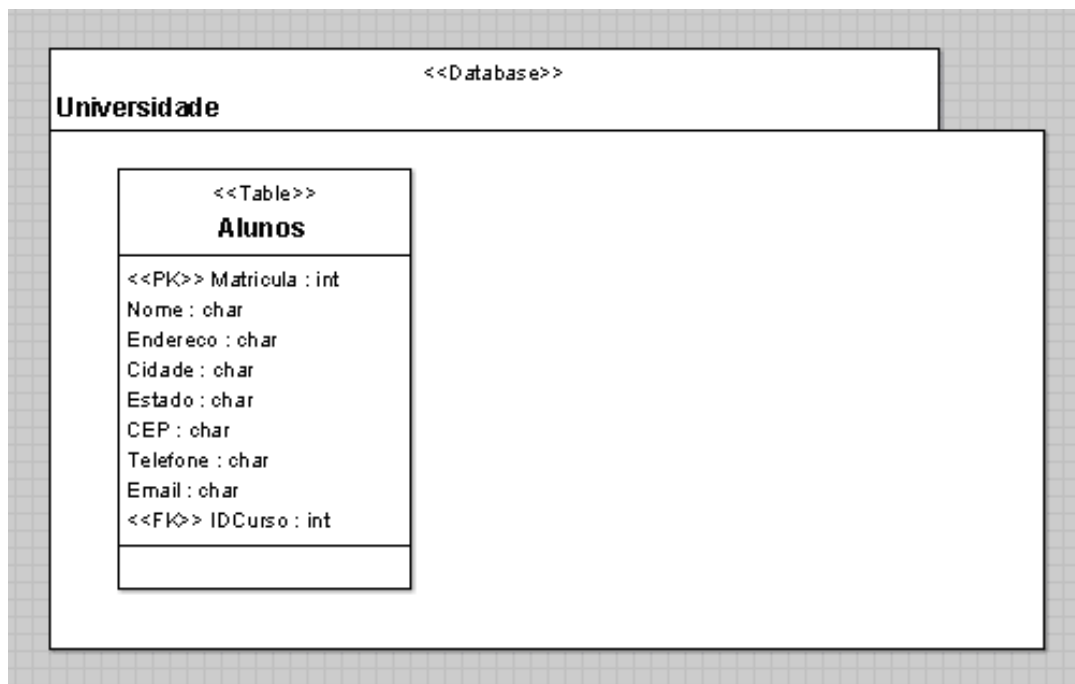


Figura 4.2: Tabela com a definição de suas chaves.

A tabela ilustrada na figura 4.2 foi submetida à ferramenta XMI2SQL. O resultado obtido após o processamento pode ser visto no quadro abaixo, no qual foi gerada a seqüência de comandos SQL necessária para a criação da tabela Aluno.

```

DROP DATABASE Universidade;
CREATE DATABASE Universidade;
USE Universidade;
CREATE TABLE Alunos
(
Matricula int NOT NULL,
Nome char NOT NULL,
Endereco char NOT NULL,
Cidade char NOT NULL,
Estado char NOT NULL,
CEP char NOT NULL,
Telefone char NOT NULL,
Email char NOT NULL,
IDCurso int NOT NULL,

PRIMARY KEY (Matricula)

FOREIGN KEY (IDCurso) REFERENCES Cursos( )
)

```

Quadro 4.1: Comandos em SQL necessários para criar a Tabela Alunos.

Observa-se que todos os comandos necessários para a criação do bando de dados e da tabela foram gerados. As chaves de ordenação foram criadas através da indicação dos estereótipos <<PK>> e <<FK>>. Nota-se que a chave estrangeira (FK) faz referência à tabela Curso, que não consta na figura 4.2. Este relacionamento é obtido com a inclusão de uma *Tagged Value* associada à coluna IDCurso. Na figura 4.3 abaixo, é apresentada a seqüência necessária de procedimentos para realizar este relacionamento entre as tabelas no Argo UML.

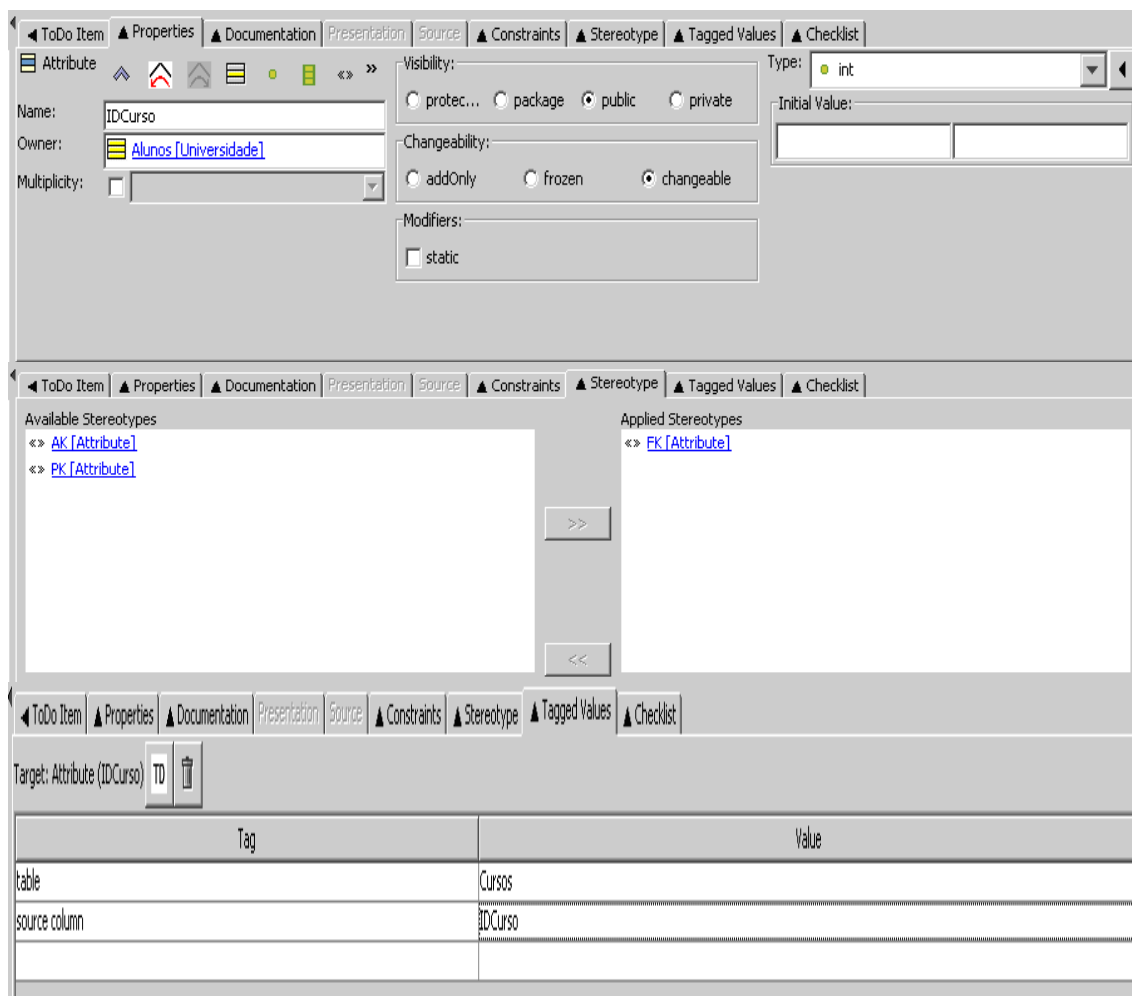


Figura 4.3 Definição de *Tagged Values* no Argo UML.

Prosseguindo com a construção do modelo físico, é acrescentada a tabela Cursos que se relaciona com a tabela Alunos através da chave estrangeira (FK) IDCurso. Observa-se na figura 4.4 que as tabelas estão associadas e também foram incluídas suas multiplicidades.

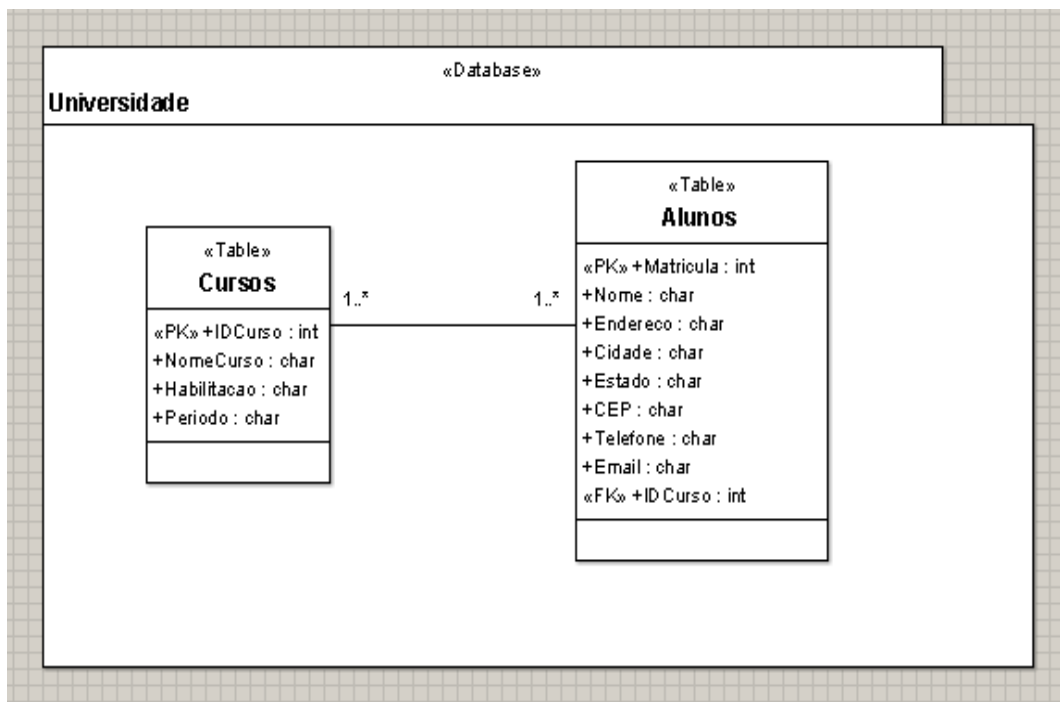


Figura 4.4 Definição de relacionamentos entre tabelas no Argo UML.

Ao submeter as tabelas da figura 4.4 à ferramenta XMI2SQL, foi obtido como resultado a seqüência de comando SQL para a criação das tabelas Alunos e Cursos, bem como suas respectivas chaves. No quadro 4.2 é apresentado o *script* SQL gerado.

```

DROP DATABASE Universidade;
CREATE DATABASE Universidade;
USE Universidade;
CREATE TABLE Alunos
(
  Matricula int NOT NULL,
  Nome char NOT NULL,
  Endereco char NOT NULL,
  Cidade char NOT NULL,
  Estado char NOT NULL,
  CEP char NOT NULL,
  Telefone char NOT NULL,
  Email char NOT NULL,
  IDCurso int NOT NULL,

  PRIMARY KEY (Matricula)

  FOREIGN KEY (IDCurso) REFERENCES Cursos(IDCurso)
)
CREATE TABLE Cursos
  
```

```
(  
IDCurso int NOT NULL,  
NomeCurso char NOT NULL,  
Habilitacao char NOT NULL,  
Periodo char NOT NULL,  
  
PRIMARY KEY (IDCurso)  
  
)
```

Quadro 4.2: Script SQL para criação das tabelas Alunos e Cursos.

4.2 Complementando o Modelo Físico.

A fim de complementar o modelo físico de dados de um sistema para universidade, a partir de agora serão incorporadas ao modelo as tabelas Professores, Departamentos, Unidades, Disciplinas, DisciplinaAluno e DisciplinaProfessor, bem como os relacionamentos entre as mesmas.

Na figura 4.5 é apresentado o resultado final do modelo físico de dados proposto, onde também são expressas as multiplicidades de cada relacionamento.

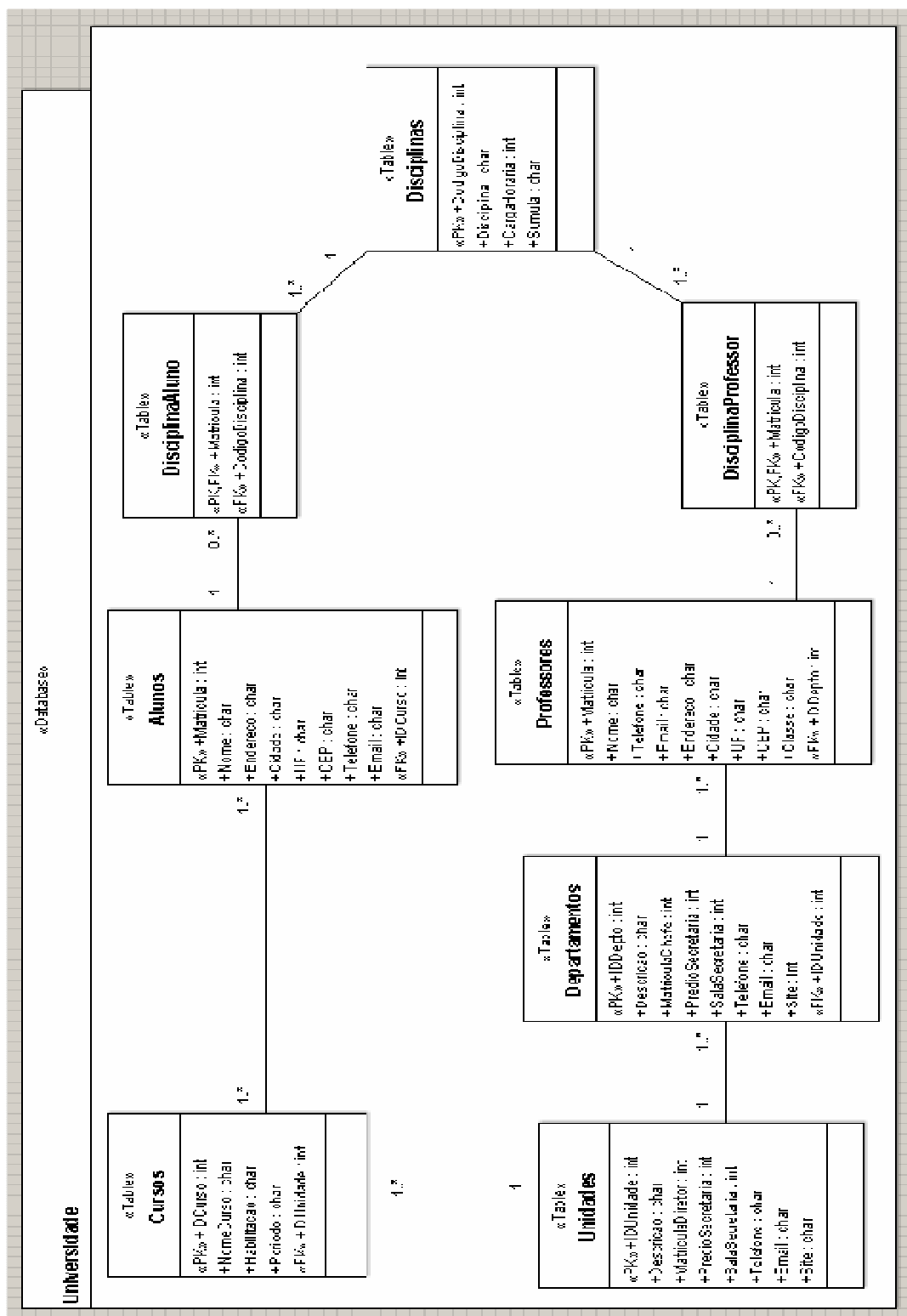


Figura 4.5 Modelo físico de dados Universidade.

Cabe observar, que as tabelas DisciplinaAluno e DisciplinaProfessor deveriam estar modeladas com o estereótipo <<Associative Table>>. Porém, isto não foi possível por restrições de implementação da ferramenta XMI2SQL. Ela somente permite a utilização dos seguintes estereótipos: *Database*, *Table*, *PK*, *FK*, *table* e *source column*.

A análise de parte do programa fonte apresentado no quadro 4.3 comprova tais restrições.

```

    /// <summary>
        /// XMI2SQL's subset of valid stereotypes for modeling data
with
        /// Agile Data's UML Profile for Data Modeling.
        /// </summary>
        static ArrayList validStereotypes = new ArrayList(
            new string[]{
                "Database",
                "Table",
                "PK",
                "FK" });

        /// <summary>
        /// XMI2SQL's subset of valid tagged values for modeling
data with
        /// Agile Data's UML Profile for Data Modeling.
        /// </summary>
        static ArrayList validTaggedValues = new ArrayList(
            new string[]{
                "table",
                "source column" });

```

Quadro 4.3: Parte do programa fonte XMI2SQL.CS desenvolvido em C Sharp.

Em função destas limitações da ferramenta XMI2SQL, a mesma não será mais utilizada no decorrer deste trabalho, visto que somente parte do *profile* proposto pôde ser interpretado pela mesma. Apesar disso, a ferramenta reafirma o conceito de promover a utilização de arquivos no padrão XMI (XML Metadata Interchange) em ferramentas UML, pois o emprego deste tipo de arquivo facilita o intercâmbio de modelos.

Ao submeter o modelo físico da figura 4.5 à ferramenta XMI2SQL, foi obtido como resultado a seqüência de comando SQL para a criação de todas as tabelas que compõem o banco de dados Universidade, bem como suas respectivas chaves primárias e estrangeiras. No quadro 4.4 é apresentado o *script* SQL gerado.

```

DROP DATABASE Universidade;
CREATE DATABASE Universidade;
USE Universidade;
CREATE TABLE Alunos
(
    Matricula int NOT NULL,

```

```
Nome char NOT NULL,  
Endereco char NOT NULL,  
Cidade char NOT NULL,  
UF char NOT NULL,  
CEP char NOT NULL,  
Telefone char NOT NULL,  
Email char NOT NULL,  
IDCurso int NOT NULL,
```

```
PRIMARY KEY (Matricula)
```

```
FOREIGN KEY (IDCurso) REFERENCES Cursos(IDCurso)  
)
```

```
CREATE TABLE Cursos  
(  
IDCurso int NOT NULL,  
NomeCurso char NOT NULL,  
Habilitacao char NOT NULL,  
Periodo char NOT NULL,  
IDUnidade int NOT NULL,
```

```
PRIMARY KEY (IDCurso)
```

```
FOREIGN KEY (IDUnidade) REFERENCES Unidades(IDUnidade)  
)
```

```
CREATE TABLE Unidades  
(  
IDUnidade int NOT NULL,  
Descricao char NOT NULL,  
MatriculaDiretor int NOT NULL,  
PredioSecretaria int NOT NULL,  
SalaSecretaria int NOT NULL,  
Telefone char NOT NULL,  
Email char NOT NULL,  
Site char NOT NULL,
```

```
PRIMARY KEY (IDUnidade)
```

```
)
```

```
CREATE TABLE Professores  
(  
Matricula int NOT NULL,  
Nome char NOT NULL,  
Telefone char NOT NULL,  
Email char NOT NULL,  
Endereco char NOT NULL,
```

```

Cidade char NOT NULL,
UF char NOT NULL,
CEP char NOT NULL,
Classe char NOT NULL,
IDDepto int NOT NULL,

```

```

PRIMARY KEY (Matricula)

```

```

FOREIGN KEY (IDDepto) REFERENCES Departamentos(IDDepto)
)

```

```

CREATE TABLE Departamentos

```

```

(
IDDepto int NOT NULL,
Descricao char NOT NULL,
MatriculaChefe int NOT NULL,
PredioSecretaria int NOT NULL,
SalaSecretaria int NOT NULL,
Telefone char NOT NULL,
Email char NOT NULL,
Site int NOT NULL,
IDUnidade int NOT NULL,

```

```

PRIMARY KEY (IDDepto)

```

```

FOREIGN KEY (IDUnidade) REFERENCES Unidades(IDUnidade)
)

```

```

CREATE TABLE DisciplinaAluno

```

```

(
Matricula int NOT NULL,
CodigoDisciplina int NOT NULL,

```

```

PRIMARY KEY (Matricula)

```

```

FOREIGN KEY (Matricula) REFERENCES Alunos(Matricula)
FOREIGN KEY (CodigoDisciplina) REFERENCES Disciplinas(CodigoDisciplina)
)

```

```

CREATE TABLE DisciplinaProfessor

```

```

(
Matricula int NOT NULL,
CodigoDisciplina int NOT NULL,

```

```

PRIMARY KEY (Matricula)

```

```

FOREIGN KEY (Matricula) REFERENCES Professores(Matricula)
FOREIGN KEY (CodigoDisciplina) REFERENCES Disciplinas(CodigoDisciplina)
)

```

```
CREATE TABLE Disciplinas
(
CodigoDisciplina int NOT NULL,
Disciplina char NOT NULL,
CargaHoraria int NOT NULL,
Sumula char NOT NULL,

PRIMARY KEY (CodigoDisciplina)

)
```

Quadro 4.4 Script SQL para criação das tabelas do modelo físico

4.3 Modelo Físico Universidade.

Continuando com o processo de construção do modelo físico de dados que tem como base parte da estrutura organizacional e funcionamento de uma universidade, serão incluídos a partir de agora os demais elementos propostos pelo *profile*, que são necessários para uma melhor compreensão do mesmo. O objetivo é adicionar ao modelo algumas das principais características do *profile*, como *Views*, *Triggers*, *Store Procedures*, índices e a formação das chaves simples e compostas. Cabe ressaltar, que as tabelas empregadas na composição do modelo não estão totalmente normalizadas. Este critério foi adotado para possibilitar a representação em um único modelo, de grande parte dos elementos que compõem o *profile*. Na figura 4.6 é apresentado o modelo físico de dados proposto na sua forma completa.

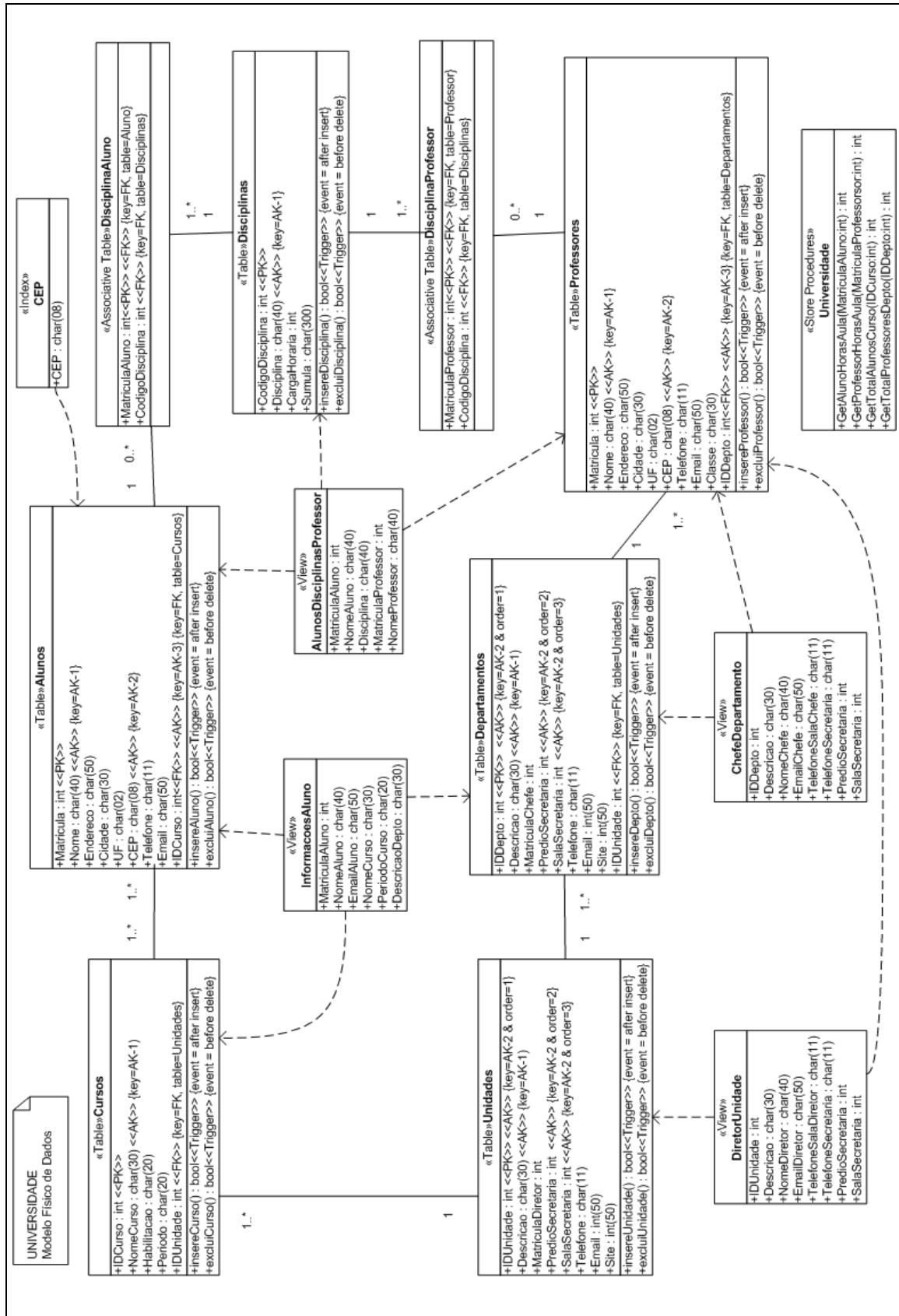


Figura 4.6: Modelo físico de dados desenvolvido com Microsoft Office Visio 2007.

4.4 Análise do Modelo Físico Universidade.

Com a finalidade de enfatizar as técnicas sugeridas pelo *profile*, a seguir partes do modelo acima serão analisadas e comentadas, de maneira que se possa aprimorar o entendimento da proposta de modelagem do autor.

Na tabela Alunos da figura 4.7, pode-se identificar através do estereótipo <<PK>> que a coluna Matricula está sendo designada para compor a chave primária da tabela. A coluna Nome é designada como a primeira chave alternada e isto é determinado pela utilização do estereótipo <<AK>> tendo como complemento a instrução em OCL: “{key=AK-1}”, onde “AK-1” é a indicação de que a coluna é a primeira chave alternada. A coluna CEP é designada como a segunda chave alternada e isto é determinado pela utilização do estereótipo <<AK>> tendo como complemento a instrução em OCL: “{key=AK-2}”, onde “AK-2” é a indicação de que a coluna é a segunda chave alternada. A coluna IDCurso é designada como a terceira chave alternada e isto é determinado pela utilização do estereótipo <<AK>> tendo como complemento a instrução em OCL: “{key=AK-3}”, onde “AK-3” é a indicação de que a coluna é a terceira chave alternada. A coluna IDCurso também é designada como uma chave estrangeira da tabela. Isto é determinado pela utilização do estereótipo <<FK>> tendo como complemento a instrução em OCL: “{key=FK, table=Cursos}”, onde “FK” indica que a coluna é uma chave estrangeira e “table=Cursos” faz referência a qual tabela este relacionamento está associado, no caso a tabela Cursos. O *Trigger* “insereAluno()” tem como retorno de sua execução um valor booleano, indicando se a execução do mesmo foi bem sucedida ou não. Complementando o *Trigger* “insereAluno()” está a instrução em OCL: “{event = after insert}”, que significa que o *Trigger* deve ser executado após cada operação de inclusão de alunos na tabela. O *Trigger* “excluiAluno()” também tem como retorno de sua execução um valor booleano, indicando se a execução do mesmo foi bem sucedida ou não. Complementando o *Trigger* “excluiAluno ()” está a instrução em OCL: “{event = before delete}”, que significa que o *Trigger* deve ser executado antes de cada operação de exclusão de alunos na tabela. Na tabela Cursos as definições com relação ao *profile* são semelhantes as da tabela alunos, o que torna redundante a explicação. Finalizando a análise com relação a figura 4.7, observa-se o emprego do elemento Nota, onde estão informados o nome do banco de dados e o tipo de modelo desenvolvido.

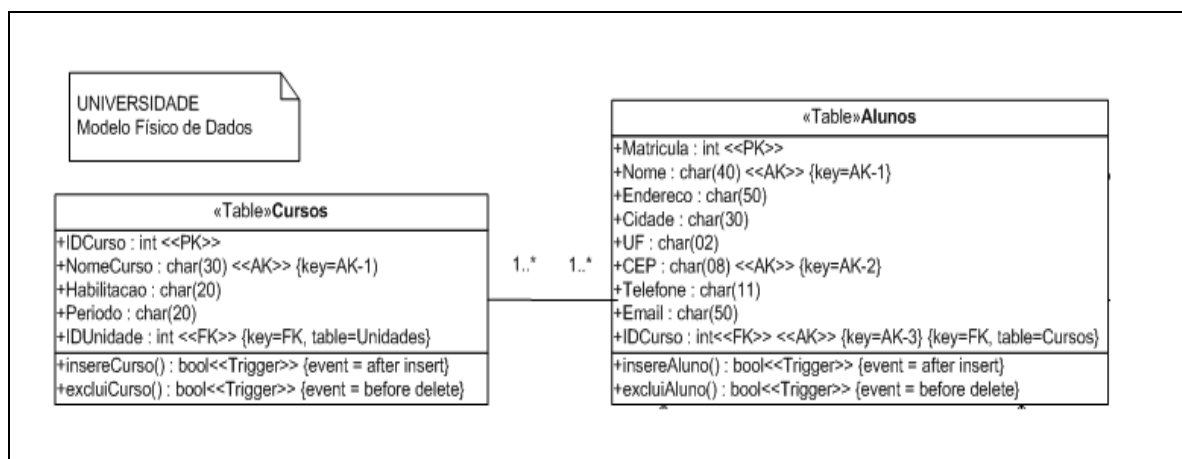


Figura 4.7: Tabelas Alunos e Cursos do modelo físico Universidade.

Na figura 4.8 é feita a representação do índice CEP da tabela Alunos. No *Class Box* com o estereótipo <<Index>>, é indicado o nome da coluna da tabela que é empregada na formação do índice, neste caso a coluna CEP. A seta de dependência indica que o índice CEP está diretamente associado à tabela Alunos.

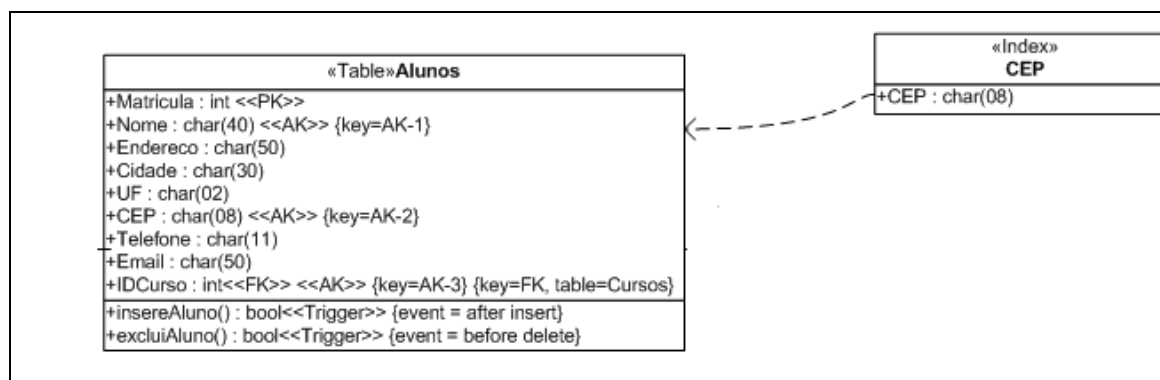


Figura 4.8: Modelagem do índice associado à tabela Alunos.

Na figura 4.9 é feita a modelagem de uma tabela associativa. Neste caso, o meio utilizado como ligação entre as tabelas Disciplina e Professor, é a tabela DisciplinaProfessor que está designada pelo estereótipo <<Associative Table>>. O emprego das chaves estrangeiras (FK) no modelo deixa claro a maneira como é feita esta ligação entre as tabelas. Na tabela DisciplinaProfessor, a coluna MatriculaProfessor é designada como chave primária e também como chave estrangeira. Isto é determinado pela utilização dos estereótipos <<PK>> e <<FK>>, e este último tendo como complemento a instrução em OCL: “{key=FK, table=Professor}”, onde “FK” indica que a coluna é uma chave estrangeira e “table=Professor” faz referência à qual tabela este relacionamento está associado, no caso a tabela Professor. A outra parte da associação é feita pela coluna CodigoDisciplina que também é designada como uma chave estrangeira da tabela. Isto é determinado pela utilização do estereótipo <<FK>> tendo como complemento a instrução em OCL: “{key=FK, table=Disciplinas}”, onde

“FK” indica que a coluna é uma chave estrangeira e “table=Disciplinas” faz referência à qual tabela este relacionamento está associado, no caso a tabela Disciplinas.

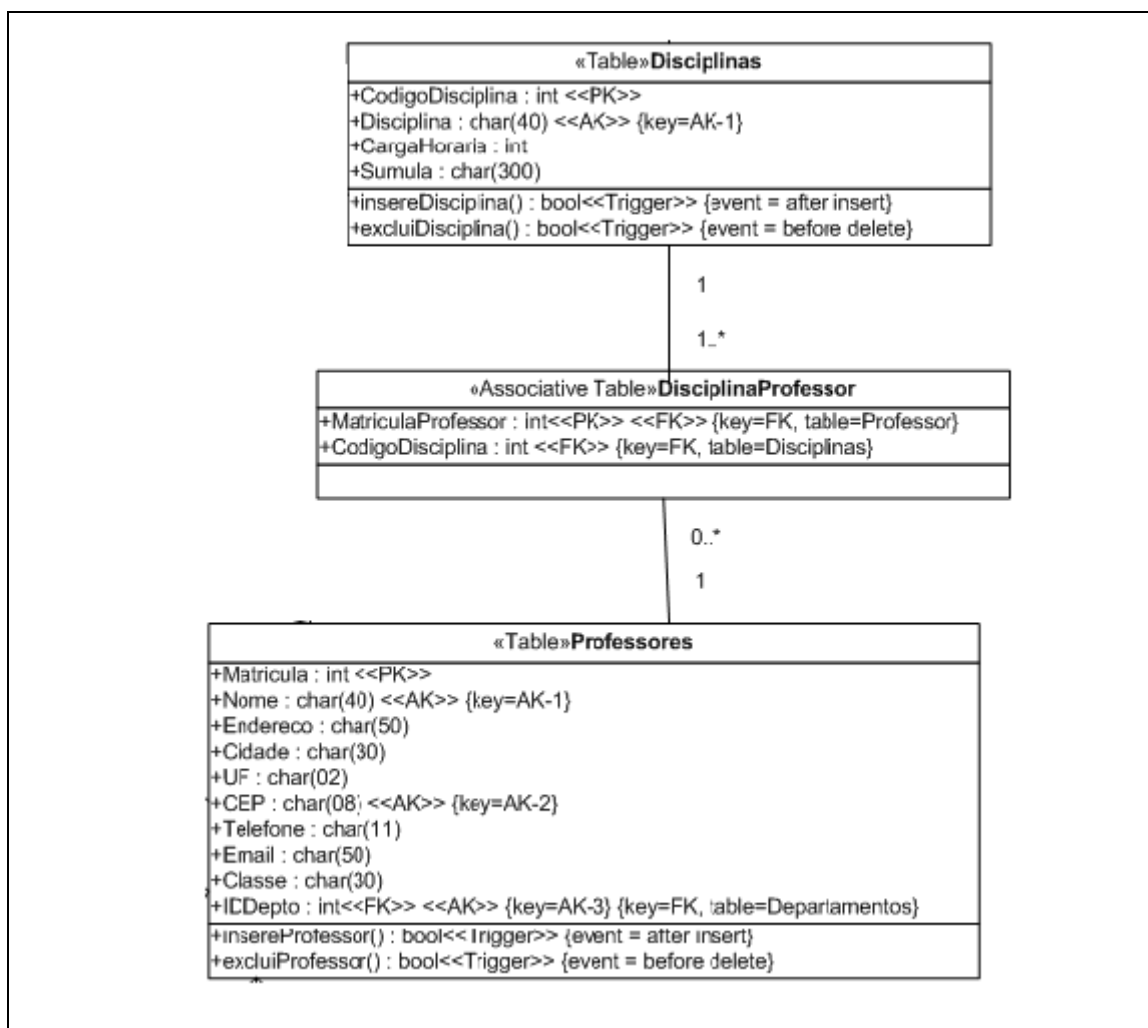


Figura 4.9: Modelagem de tabela associativa.

Na tabela apresentada na figura 4.10 está representada uma chave alternada composta por três colunas. Esta composição é feita pela união das colunas IDDepto, PredioSecretaria e SalaSecretaria. Observando as definições da coluna IDDepto, verifica-se que a mesma é designada como chave primária e chave alternada.

Isto é determinado pela utilização dos estereótipos <<PK>> e <<AK>>, e este último tendo como complemento a instrução em OCL: “{key=AK-2 & order=1}”, onde “AK-2” indica que a coluna faz parte da segunda chave alternada definida na tabela e “order=1” indica que esta coluna é o primeiro elemento na formação da chave composta. Na coluna PredioSecretaria está definida a instrução em OCL: “{key=AK-2 & order=2}”, onde “AK-2” indica que a coluna faz parte da segunda chave alternada definida na tabela e “order=2” indica que esta coluna é o segundo elemento na formação da chave composta. A coluna SalaSecretaria finaliza a modelagem da chave alternada composta, sendo que nela está definida a instrução em OCL: “{key=AK-2 & order=3}”, onde “AK-2” indica que a coluna faz parte da segunda chave alternada

definida na tabela e “*order=3*” indica que esta coluna é o terceiro elemento na formação da chave composta.

«Table»Departamentos
+IDDepto : int <<PK>> <<AK>> {key=AK-2 & order=1}
+Descricao : char(30) <<AK>> {key=AK-1}
+MatriculaChefe : int
+PredioSecretaria : int <<AK>> {key=AK-2 & order=2}
+SalaSecretaria : int <<AK>> {key=AK-2 & order=3}
+Telefone : char(11)
+Email : int(50)
+Site : int(50)
+IDUnidade : int <<FK>> {key=FK, table=Unidades}
+insereDepto() : bool<<Trigger>> {event = after insert}
+excluiDepto() : bool<<Trigger>> {event = before delete}

Figura 4.10: Modelagem de chaves compostas.

Na representação gráfica da *View* InformacoesAluno, disponibilizada pela figura 4.11, observa-se que a mesma é formada com informações extraídas de três tabelas. Da tabela Alunos são utilizadas as colunas Matricula, Nome e Email na formação da *View*. Da tabela Cursos são utilizadas as colunas NomeCurso e Período. A *View* é finalizada com a informação da coluna Descrição que é extraída da tabela Departamentos. As dependências da *View* InformacoesAlunos com relação às três tabelas que são utilizadas na sua formação, são representadas através das setas tracejadas que saem do *Class Box* em direção a estas.

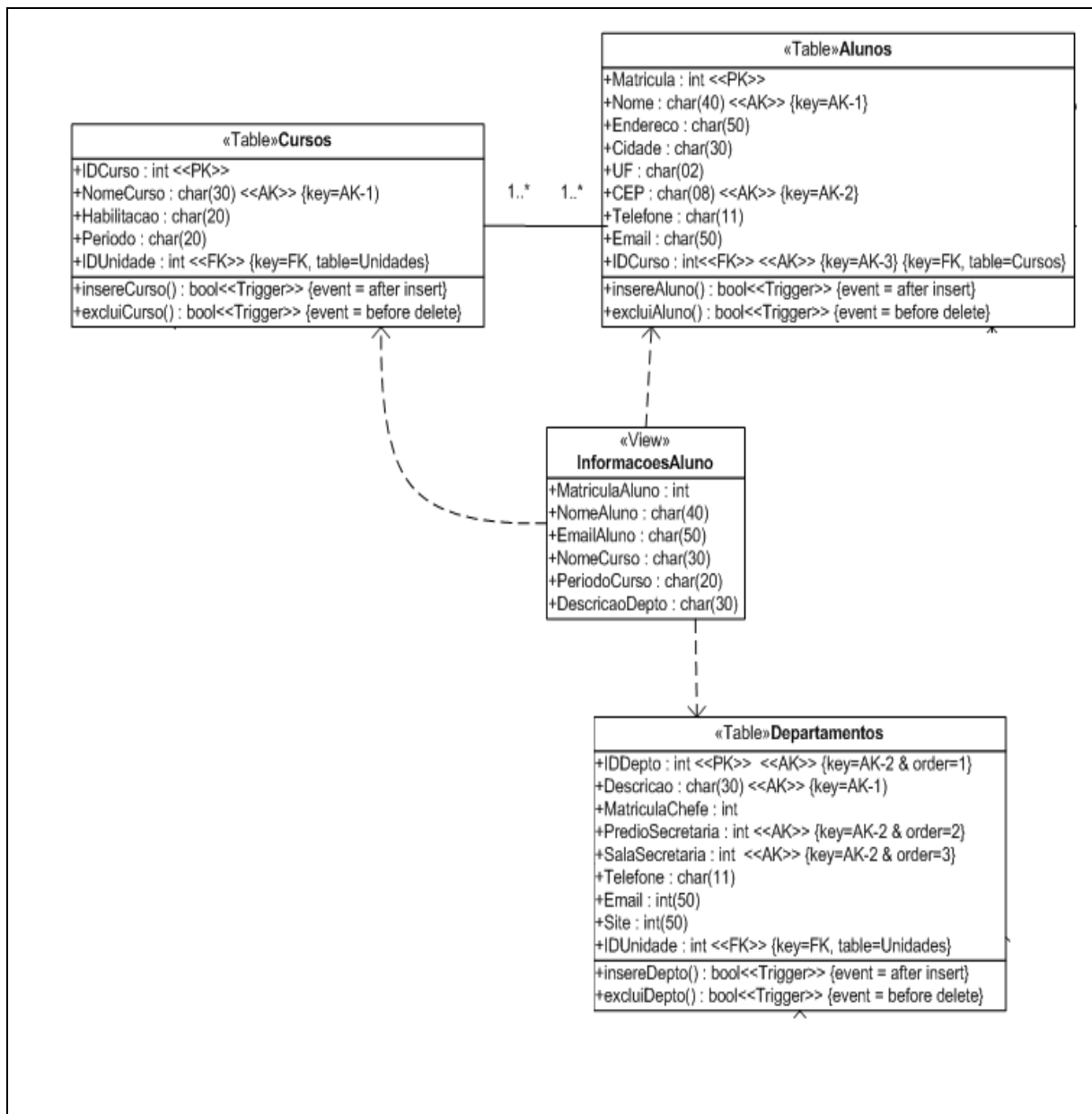


Figura 4.11: Modelagem de uma View.

As *Store Procedures* são agrupadas em um *Class Box* que fica isolado em relação aos demais elementos do modelo físico de dados. Na figura 4.12, estão representadas quatro *Store Procedures* que são executadas no banco de dados Universidade. A primeira é a *GetAlunoHorasAula(MatriculaAluno:int)*, cujo objetivo é retornar o número de horas-aula de um aluno. É informado como parâmetro de pesquisa o número de matrícula do aluno e é obtido como retorno um valor inteiro. A segunda *Store Procedure* é a *GetProfessorHorasAula(MatriculaProfessor:int)*, cujo objetivo é retornar o número de horas-aula de um professor específico. É informado como parâmetro de pesquisa o número de matrícula do professor e é obtido como retorno um valor inteiro. A terceira é a *GetTotalAlunosCurso(IDCurso:int)*, cujo objetivo é retornar o total de alunos matriculados em um curso. É informado como parâmetro na *Store Procedure* a identificação do curso (*IDCurso*) e é obtido como retorno um valor inteiro. A última *Store Procedure* definida no *Class Box* é a

GetTotalProfessoresDepto(IDDepto:int) , cujo objetivo é retornar o total de professores de um departamento. É informado como parâmetro a identificação do departamento (IDDepto) e é obtido como retorno um valor inteiro.

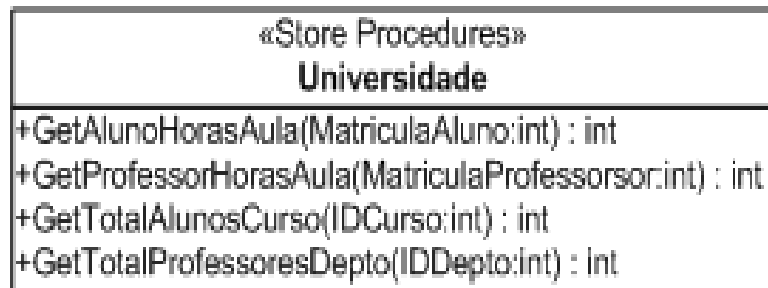


Figura 4.12: Modelagem de *Store Procedures*.

5 CONCLUSÃO

O *profile* proposto por Scott W. Ambler atende as necessidades de analistas e projetistas de sistemas ao permitir uma adequada transição entre as fases de desenvolvimento de sistemas. Estão considerados no *profile*, por exemplo, todos os elementos necessários para se fazer a transposição de uma modelagem de classes do padrão Orientado a Objetos para um padrão em banco de dados relacional, este baseado em tabelas e seus relacionamentos.

Uma vantagem desta proposta em relação a outros *profiles* de modelagem de dados é de que todas as informações importantes e necessárias para a compreensão e posterior implementação em um sistema de banco de dados, podem ser visualizadas em um único documento. Em contra-partida, ocorre que o modelo pode conter uma quantidade enorme de informações, tornando-se de difícil interpretação. Este tipo de problema é facilmente observado na modelagem de chaves alternadas compostas, onde a especificação das mesmas é baseada em OCL e é repetida em várias colunas da tabela. A modelagem de chaves pode ficar extensa e confusa, uma característica não desejada em um modelo que deve ser de fácil compreensão. Portanto, deve-se ter bom senso no emprego dos diversos elementos sugeridos pelo *profile*.

Finalizando, acredito que este *profile*, em função de suas novas características e por levar em consideração diversos elementos já empregados em ferramentas UML existentes, seja uma boa base para o desenvolvimento de um padrão definitivo para modelagem de dados por parte da OMG.

REFERÊNCIAS

- AMBLER, S. W. **Agile Database Techniques - Effective Strategies for the Agile Software Developer**. New York: John Wiley & Sons, 2003.
- AMBLER, S. W. **The Object Primer**. 3rd ed. New York: Cambridge University Press, 2004.
- AMBLER, S. W.; SADALAGE, P. J. **Refactoring Databases: Evolutionary Database Design**. Crawfordsville: Addison Wesley Professional, 2006a.
- AMBLER, S.W. **A UML Profile for Data Modeling**. Toronto: Ambysoft Inc, 2006b. Disponível em: <<http://www.agiledata.org/essays/umlDataModelingProfile.html>>. Acesso em: nov. 2007.
- HARTFORD, E. R **Reducing the Gap Between Database Models and Database Designs: XMI2SQL**. 2003. Proposta e projeto (Master of Science). Computing and Software Systems - Institute of Technology, University of Washington, Tacoma.
- MULLER, R.J. **Database Design for Smarties: Using UML for Data Modeling**. [S.l.]: Morgan Kaufmann Publishers, 1999.
- NAIBURG, E. J.; MAKSIMCHUK, R. A. **UML for Database Design**. [S.l.]: Addison Wesley, 2001.
- OBJECT MANAGEMENT GROUP (OMG). **Request for Proposal – Information Management Metamodel (IMM)**: Needham, 2005. Disponível em: <<http://www.omg.org/cgi-bin/doc?ab/2005-12-2>>. Acesso em: nov. 2007.
- PONNIAH, P. **Data Modeling Fundamentals**. Hoboken: John Wiley & Sons, 2007.