Mapeamento de Modelos ER para DDL da SQL

Alberto Hokari, Ricardo Terra

Departamento de Ciência da Computação
Universidade Federal de Lavras (UFLA), Brasil

ah_009@computacao.ufla.br, terra@dcc.ufla.br

Resumo. *Entidade Relacionamento (ER) é um modelo importante visto em disciplinas de projeto de banco de dados. É amplamente empregado em projetos conceituais de banco de dados e adotado em muitas das ferramentas de modelagem existentes. Além disso, é comum nos projetos de disciplinas de banco de dados, utilizar-se do modelo de dados Relacional no processo Ensino-aprendizagem. Dessa forma, o processo de projeto se inicia por um diagrama conceitual de alto-nível, passando por um diagrama conceitual de nível lógico e, por fim, a implementação do banco de dados utilizando a DDL da SQL. Embora essa seja uma prática correta, o problema é que seguir esse processo, de análise e correção de dezenas de projetos de alunos, é uma tarefa que pode demandar muito tempo do professor. Diante desse cenário, este artigo propõe uma metodologia para mapear modelos ER para DDL da SQL, o que pode (i) auxiliar professores na correção de exercícios e trabalhos; e (ii) auxiliar profissionais da área a criar um projeto físico de forma prática e ágil.*

1. Introdução

Uma das abordagens de modelagem mais conhecidas é a *Entidade Relacionamento* (ER) [1, 2]. O modelo ER é baseado em uma percepção de um mundo real que consiste em uma coleção de objetos básicos, chamados *entidades*, e as *relações* entre esses objetos. Uma entidade é uma "coisa" ou "objeto" no mundo real que é distinguível dos outros objetos [3].

Esse modelo e suas variações costumam ser utilizados em projetos conceituais de banco de dados e adotados em muitas das ferramentas de modelagem existentes [4]. Uma dessas ferramentas é o TerraER[1], na qual é um software voltado ao meio acadêmico, mais especificamente no auxílio ao aprendizado de disciplinas de modelagem conceitual de banco de dados [5, 6]. Nesse contexto, é comum nos projetos de disciplinas de banco de dados, utilizar-se do modelo de dados Relacional no processo Ensino-aprendizagem. Dessa forma, o processo de projeto se inicia por um diagrama conceitual de alto-nível, passando por um diagrama conceitual de nível lógico e, por fim, a implementação do banco de dados utilizando a *Data Definition Language* (DDL) da *Structured Query Language* (SQL). Embora essa seja uma prática correta, o ***problema*** é que seguir esse processo, de análise e correção de dezenas de projetos de alunos, é uma tarefa que pode demandar muito tempo do professor.

Diante desse cenário, o ***objetivo*** deste artigo é a proposta de uma extensão para a ferramenta TerraER que possibilita o mapeamento de modelos ER para DDL

[1] http://www.terraer.com.br/

da SQL. Basicamente, construções do modelo ER são mapeadas para instruções DDL. A ***motivação*** por trás deste artigo, portanto, se resume em: (i) auxiliar professores na correção de exercícios e trabalhos; e (ii) auxiliar profissionais da área a criar um projeto físico de forma prática e ágil. as as construções do modelo ER foram mapeadas para instruções DDL.

Este artigo está organizado como a seguir. A Seção 2 apresenta os conceitos fundamentais para o entendimento deste estudo. A Seção 3 descreve o processo de mapeamento de modelos ER para DDL da SQL, incluindo a metodologia utilizada, a tabela de mapeamento ER/DDL, a implementação da extensão e limitações. A Seção 4 apresenta um estudo de caso controlado em que é avaliada o mapeamento da DDL da SQL de um modelo bancário que contempla grande parte das construções ER. Por fim, a Seção 5 conclui e apresenta trabalhos futuros.

2. *Background*

Esta seção apresenta conceitos fundamentais ao entendimento deste estudo, tais como modelo ER, modelo ER estendido, ferramenta TerraER e a linguagem SQL.

2.1. Modelo ER

Segundo Garcia-Molina [7], no modelo Entidade Relacionamento, a estrutura do dado é representada graficamente como um diagrama usando três principais tipos de elementos: entidades, atributos e relacionamentos.

Entidade: Uma *entidade* é um objeto abstrato de algum tipo, e uma coleção de entidades similares forma um *conjunto de entidades* [7]. Chen [1] classifica essas entidades como *entidades fortes* e *entidades fracas*. Uma entidade fraca é uma entidade cuja existência depende de alguma forma de outra entidade, no sentido de que ela não poder existir se essa outra entidade também não existir. Por exemplo, na Figura 1, temos a entidade `DEPENDENTE` na qual é dependente da entidade `FUNCIONARIO`, pois um dependente não existe sem um funcionário. Ao contrário, uma entidade forte é uma entidade que não é fraca, ou seja, a entidade `FUNCIONARIO` da Figura 1 não depende de nenhuma outra para existir.[2]

Atributos: Uma entidade é representada por um conjunto de *atributos*. Os atributos são propriedades descritivas possuídas por membro de um conjunto de entidades. A designação de um atributo para um conjunto de entidades expressa que o banco de dados armazena informações semelhantes concernentes a cada entidade no conjunto de entidades; entretanto, cada entidade pode ter seu próprio valor para cada atributo [3]. Além disso, segundo Date [2], os atributos podem ser (i) simples ou compostos; (ii) chaves; (iii) univalorados ou multivalorados; e (iv) básicos ou derivados.

[2]Embora alguns autores utilizam o termo "entidade regular" ao invés de "entidade forte" [2], este trabalho adota o termo "entidade forte".

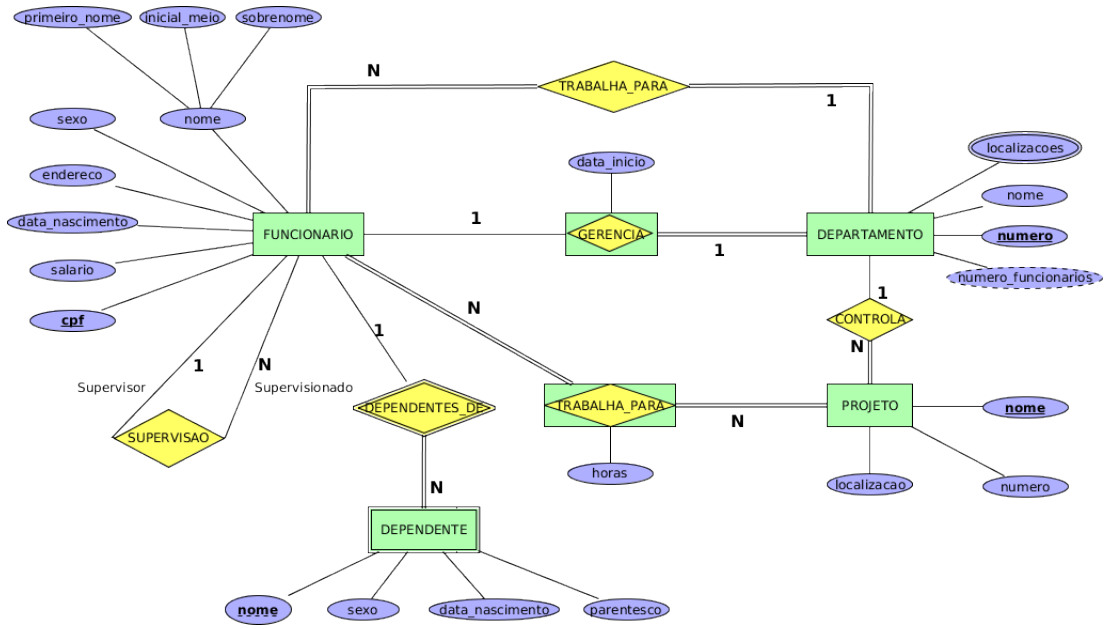


Figura 1. Diagrama ER (adaptado de [4]).

A Figura 1 ilustra cada um desses atributos. Por exemplo, a entidade `FUNCIONARIO` possui *salario* como atributo simples; *nome* como atributo composto, pois *nome* é composto de *primeiro_nome*, *inicial_meio* e *sobrenome*; e *cpf* como atributo chave. Na entidade `DEPARTAMENTO` temos *localizacoes* como atributo multivalorado, pois um departamento pode-se encontrar em diferentes lugares; e ainda em `DEPARTAMENTO` temos *numero_funcionarios* como atributo derivado, ou seja, podemos obter a quantidade de empregados através do somatório de todos os funcionários da entidade `FUNCIONARIO`.

Relacionamentos: *Relacionamentos* são conexões entre duas ou mais entidades [7]. As entidades envolvidas em determinado relacionamento são ditas *participantes* desse relacionamento. O número de participantes em determinado relacionamento é chamado *grau* desse relacionamento.

Seja R um tipo de relacionamento que envolve o tipo de entidade E como participante. Se toda instância de E participa de pelo menos uma instância de R, então a participação de E em R é considerada *total*; do contrário ela é considerada *parcial*. Um relacionamento ER pode ser de *um para um*, de *um para muitos* ou de *muitos para muitos*. Na Figura 1, a entidade `DEPARTAMENTO` tem um relacionamento com a entidade `PROJETO`, ou seja, um departamento controla pelo menos um projeto. Dessa forma, temos um relacionamento binário de um para muitos (1:N) com restrição de participação total no lado N.

2.2. Modelo ER Estendido

Desde o final da década de 1970, projetistas de aplicações de banco de dados têm tentado projetar esquemas de banco de dados mais precisos, que reflitam as propriedades de dados e restrições com mais precisão [4]. Diante desse fato, em 1986, Teory et al. [8] propuseram uma extensão ao modelo ER acrescentando os conceitos de relacionamentos de classe/subclasse, herança de tipo, especialização/generalização e suas restrições.

Na Figura 2, por exemplo, um membro da entidade FUNCIONARIO pode ter o emprego do tipo SECRETARIA, TECNICO ou ENGENHEIRO, o que caracteriza a entidade FUNCIONARIO como *superclasse ou supertipo* e, de forma análoga, as entidades SECRETARIA, TECNICO e ENGENHEIRO como *subclasses ou subtipos*. Vale ressaltar ainda que as entidades membros de uma subclasse (SECRETARIA, TECNICO e ENGENHEIRO) *herdam* todos os atributos da entidade como um membro da superclasse (FUNCIONARIO).

A Figura 2 exemplifica também o conceito de generalização e especialização. A *generalização* se refere ao processo de definição de um tipo de entidade generalizado com base nos tipos de entidades dados, porém, especificamente nessa figura, pode-se observar o caso de *especialização*, na qual definimos um *conjunto de subclasses* (SECRETARIA, TECNICO e ENGENHEIRO) de um tipo de entidade (FUNCIONARIO).

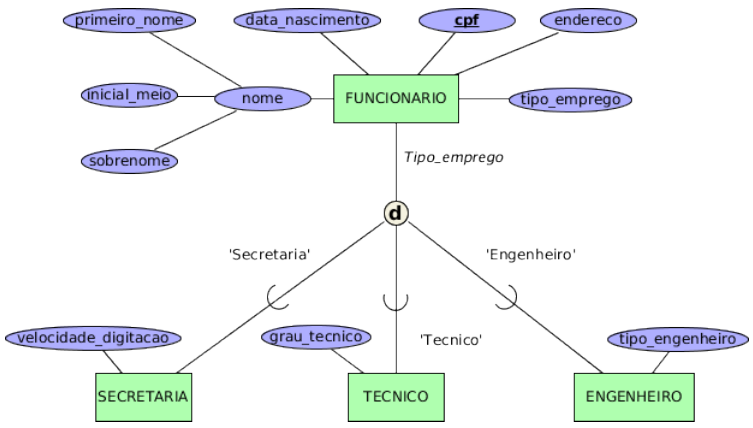


Figura 2. Diagrama ER Estendido (adaptado de [4]).

Duas outras restrições podem se aplicar a uma especialização. A primeira é a *restrição de disjunção* que especifica que as subclasses da especialização devem ser disjuntas. Isso significa que uma entidade pode ser um membro de, *no máximo*, uma das subclasses da especialização. Se as subclasses não forem restringidas a serem disjuntas, seus conjuntos de entidades podem ser *sobrepostos*, isto é, a mesma entidade pode ser um membro de mais de uma subclasse da especialização. A Figura 2 apresenta o caso de especialização com a restrição de disjunção, na qual o tipo de trabalho do FUNCIONARIO pode ser somente de SECRETARIA, ou de TECNICO, ou de ENGENHEIRO.

A segunda restrição sobre a especialização é chamada de *restrição de completude*, que pode ser total ou parcial. Uma restrição de *especialização total* especifica que *toda* entidade na superclasse precisa ser um membro de pelo menos uma subclasse na especialização. Já a *especialização parcial* permite que uma entidade não pertença a qualquer uma das subclasses [4]. Logo, temos quatro restrições possíveis na especialização: (i) disjunção, total; (ii) disjunção, parcial; (iii) sobreposição, total; e (iv) sobreposição, parcial. A Figura 2 exemplifica o caso de especialização com restrição de disjunção e restrição de completude parcial (caso *ii* supracitado), onde o tipo de trabalho do FUNCIONARIO pode ser somente de SECRETARIA, ou de TECNICO, ou de ENGENHEIRO, ou de nenhum dos três já que a restrição de completude é parcial.

2.3. TerraER

O TerraER é uma ferramenta de código aberto, voltada ao meio acadêmico, mais especificamente no auxílio ao aprendizado de disciplinas de modelagem conceitual de banco de dados [5, 6]. TerraER permite criar modelos conceituais mais condizentes ao que os professores lecionam na disciplina de banco de dados. Isso pode ser constatado através da Figura 3, na qual a barra de ferramentas de objetos possui atalhos para criação de elementos do modelo ER na notação Chen estendida, adotada por Elmasri e Navathe [5, 4].

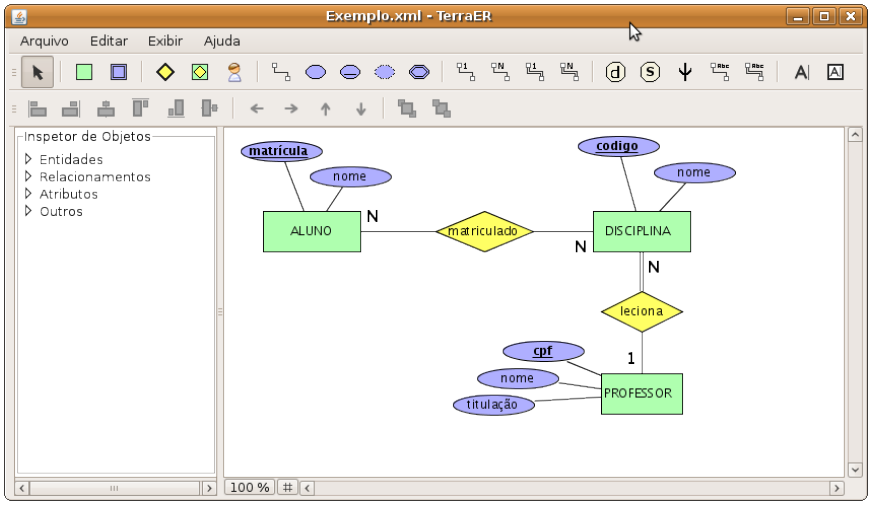


Figura 3. Ferramenta TerraER

Portanto, devido à facilidade de criação de modelo conceituais e pelo fato de a ferramenta já ser adotada em mais de 30 Instituições de Ensino Superior (IES), a ferramenta foi escolhida para implementar o mapeamento de modelos ER para DDL da SQL de forma a auxiliar professores na correção de exercícios e trabalhos e auxiliar profissionais da área a criar um projeto físico de forma prática e ágil.

2.4. SQL

A Linguagem de Consulta Estruturada (SQL) é uma linguagem utilizada para organizar, gerenciar e recuperar dados armazenados em um banco de dados. A IBM desenvolveu a versão original da SQL, originalmente chamada Sequel, como parte do projeto *System R* no começo dos anos 70. Desde então, a linguagem Sequel evoluiu e seu nome mudou para SQL. Além de muitos SGDBs (Sistemas Gerenciadores de Banco de Dados) atualmente suportarem a linguagem SQL, a SQL se tornou claramente o padrão de linguagem de bancos de dados relacionais [3]. Particularmente, este projeto adotou o SGBD Oracle.

A SQL é utilizada para controlar as funções que um SGBD provê aos seus usuários e é composta por várias partes:

- *Linguagem de Definição de Dados (DDL):* A DDL permite especificar, não apenas um conjunto de relações, mas também informações sobre cada relação, como o esquema para cada relação, os tipos de valores associados com cada atributo, restrições de integridade, o conjunto de índices a serem mantidos por cada relação, a informação de segurança e autorização para cada relação e a estrutura de armazenamento físico de cada relação no disco.

- *Linguagem de Manipulação de Dados (DML):* A DML fornece a habilidade de consultar informações do banco de dados, e inserir, excluir, e modificar tuplas no banco de dados.
- *Integridade:* A DDL inclui comandos para especificar restrições de integridade na qual os dados armazenados no banco de dados devem satisfazer. Atualizações na qual violam as restrições de integridade não são permitidas.
- *Definição de Visão:* A DDL inclui comandos para definição de visões. As visões são "relações virtuais" na qual não fazem parte do modelo lógico, mas são visíveis aos usuários. A relação virtual não é pré-computada e armazenada, mas é calculada executando a consulta sempre que a relação virtual é usada.
- *Controle de transação:* A SQL inclui comandos para especificar o começo e fim de transações.
- *SQL embutida e SQL dinâmica:* A SQL embutida e dinâmica definem como as declarações SQL podem ser incluídas dentro de linguagens de programação de propósito geral, como C, C++ e Java.
- *Autorização:* A DDL inclui comandos para especificar direitos de acesso a relações e visões.

É importante mencionar que algumas ferramentas de modelagem, como o MySQL Workbench[3], permitem a engenharia reversa do modelo físico (um esquema em linguagem SQL) para o modelo lógico (um diagrama *Information Engineering*).

3. Processo de Mapeamento de Modelos ER para DDL da SQL

Nos projetos de disciplinas de banco de dados, é comum utilizar o modelo de dados Relacional no processo Ensino-aprendizagem. Dessa forma, o processo de projeto se inicia por um diagrama conceitual de alto-nível, passando por um diagrama conceitual de nível lógico e, por fim, a implementação do banco de dados utilizando a DDL da SQL. Embora essa seja uma prática correta, o problema é que seguir esse processo, de análise e correção de dezenas de projetos de alunos, é uma tarefa que pode demandar muito tempo do professor. Portanto, nesta seção, é proposta uma solução para esse problema através da criação de uma extensão para a ferramenta TerraER, cuja finalidade é mapear o modelo ER para DDL da SQL e, consequentemente, auxiliar os professores na correção de projetos e profissionais da área a criarem um projeto físico de forma ágil e prática.

3.1. Metodologia

Para que a extensão pudesse ser criada, primeiramente foi elaborada uma tabela de mapeamento na qual cada elemento ER é mapeado à sua DDL correspondente; os detalhes dessa tabela são descritos na Seção 3.2. Inicialmente, para que a DDL gerada seja uniforme, houve a padronização dos seguintes termos:

- As tabelas têm seu nome em letras maiúsculas.

[3]https://www.mysql.com/products/workbench

- Um atributo criado para relacionar duas tabelas tem seu nome formado por <nome−atributo>_<nome−tabela>.
- Uma atributo chave tem seu nome definido como PK_<nome−tabela>.
- Uma restrição de integridade referencial tem seu nome definido como FK_<nome−tabela>.
- Um relacionamento M:N entre duas tabelas tem seu nome formado por <nome−tabela1>_<nome−tabela2>.
- O atributo multivalorado tem seu nome composto por <nome_tabela>_<atributo_multivalorado>.
- Caso existam dois atributos chave para uma mesma tabela, a segunda chave será UNIQUE e seu nome será formado por UQ_<nome−tabela>. O sufixo "_<XX>" será incluído caso haja mais de duas chaves, onde XX é um contador que começa com 01.

A partir da tabela de mapeamento, foi desenvolvido o Algoritmo 1 que recebe como entrada um modelo ER e, a cada iteração, gera a DDL correspondente a um elemento do modelo. Ao final da execução, a DDL gerada corresponde ao modelo ER.

Algoritmo 1 Algoritmo de conversão de modelo ER para DDL

Entrada: Modelo ER ($modelo$)
Saída: DDL correspondente ao modelo (ddl)

$ddl :=$ " "
$ddl := ddl + gerarTabelas(modelo)$ ▷ Os atributos simples são criados com a tabela
$ddl := ddl + gerarChavePrimaria(modelo)$
$ddl := ddl + gerarChaveParcial(modelo)$
$ddl := ddl + gerarGeneralizacaoEspecializacao(modelo)$
$ddl := ddl + gerarRelacionamentos(modelo)$ ▷ Tal como entidades relacionamento
$ddl := ddl + gerarAtributoMultivalorado(modelo)$
$ddl := ddl + gerarAtributoDerivado(modelo)$
if $(modelo = \emptyset)$ **then**
  return ddl;
else
  $error()$;
end if

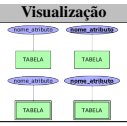
3.2. Tabela de Mapeamento

Dentre as diversas possibilidades de mapeamento de um diagrama conceitual de alto-nível para o nível lógico, as Tabelas de Mapeamento 6, 7 e 8 (veja Apêndice A) denotam uma das possibilidades de mapeamento possíveis. As regras foram criadas com base nas regras de mapeamento descritas por Earp e Bagui [9], porém com algumas modificações. Por exemplo, em algumas situações, gatilhos foram criados a fim de obter uma representação mais fiel ao realizar a transformação. Um subconjunto dessas regras é detalhado nas Tabelas 1, 2, 3, 4 e 5.

Regra de mapeamento #1: Criação de tabelas. Crie uma nova tabela para cada entidade forte e fraca.[4] Também são criados os atributos simples da tabela.

[4] As entidades subclasses de uma generalização e especialização são criadas nessa regra de mapeamento.

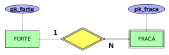
Tabela 1. Regra de Mapeamento #1

Regra	Elemento ER	Visualização	DDL Equivalente
#1	Criação de Tabelas		`CREATE TABLE TABELA (nome_atributo tipo_atributo nulidade_atributo);`

Regra de mapeamento #4: Criação de chaves parciais. Para relacionar a entidade fraca com sua entidade forte, inclua a chave primária da entidade forte na relação fraca. A chave primária da relação fraca será a chave primária da relação forte concatenada com a chave parcial.

Tabela 2. Regra de Mapeamento #4

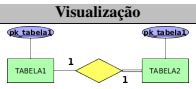
Regra	Elemento ER	Visualização	DDL Equivalente

#4	Chaves Parciais		`ALTER TABLE FRACA ADD pk_forte tipo_atributo NOT NULL;`
			`ALTER TABLE FRACA ADD CONSTRAINT PK_FRACA PRIMARY KEY (pk_fraca, pk_forte);`
			`ALTER TABLE FRACA ADD CONSTRAINT FK_FRACA FOREIGN KEY (pk_forte) REFERENCES FORTE (pk_forte);`

Regra de mapeamento #6: Relacionamento 1:1 (Participação Total). Quando um dos lados do relacionamento possui participação total e o outro participação parcial, armazene a chave primária do lado com restrição de participação parcial no lado com restrição de participação total.

Tabela 3. Regra de Mapeamento #6

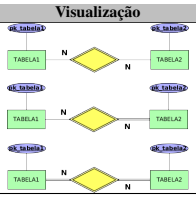
Regra	Elemento ER	Visualização	DDL Equivalente

#6	Relacionamento 1:1 (Participação Total)		`ALTER TABLE TABELA2 ADD pk_tabela1 tipo_atributo NOT NULL;`
			`ALTER TABLE TABELA2 ADD CONSTRAINT FK_TABELA2 FOREIGN KEY ( pk_tabela1) REFERENCES TABELA1 (pk_tabela1);`
			`ALTER TABLE TABELA2 ADD CONSTRAINT UQ_TABELA2 UNIQUE (pk_tabela1);`

Regra de mapeamento #10: Relacionamento M:N (Participação Parcial/Total do Lado N/Total em Ambos os Lados). Para cada relacionamento M:N, cria-se uma nova tabela com as chaves primárias de cada uma das duas entidades que estão sendo relacionadas no relacionamento M:N. A chave primária dessa nova tabela será a concatenação das entidade proprietárias. Inclui-se quaisquer atributos que o relacionamento M:N possa ter nessa nova tabela.

Tabela 4. Regra de Mapeamento #10

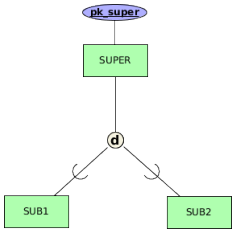
Regra	Elemento ER	Visualização	DDL Equivalente

#10	Relacionamento M:N (Participação Parcial / Total do Lado N / Total em Ambos os Lados)		`CREATE TABLE TABELA1_TABELA2( pk_tabela2 tipo_atributo NOT NULL, pk_tabela1 tipo_atributo NOT NULL);`
			`ALTER TABLE TABELA1_TABELA2 ADD CONSTRAINT PK_TABELA1_TABELA2 PRIMARY KEY (pk_tabela1,pk_tabela2);`
			`ALTER TABLE TABELA1_TABELA2 ADD CONSTRAINT FK_TABELA1_TABELA2 FOREIGN KEY (pk_tabela1) REFERENCES TABELA1 (pk_tabela1);`
			`ALTER TABLE TABELA1_TABELA2 ADD CONSTRAINT FK2_TABELA1_TABELA2 FOREIGN KEY (pk_tabela2) REFERENCES TABELA2 (pk_tabela2);`

Regra de mapeamento #17: Mapeando generalizações e especializações com subclasses disjuntas com restrições de participação parcial. Inclua a chave primária da entidade de generalização nas relações de especialização. A chave primária das relações de especialização será a mesma chave primária da relação de generalização. Aplica-se também alguns *gatilhos* em alguns dos casos para que as restrições possam ser respeitadas. Particularmente, no caso de generalização/especialização com subclasses disjuntas e participação parcial, o *gatilho* garante que uma entidade seja membro de no máximo uma das subclasses de especialização.

Tabela 5. Regra de Mapeamento #17

Regra	Elemento ER	Visualização	DDL Equivalente

#17	Generalização e Especialização (Disjunção - Participação Parcial)		`ALTER TABLE SUB1 ADD pk_super tipo_atributo NOT NULL;`
			`ALTER TABLE SUB1 ADD CONSTRAINT PK_SUB1 PRIMARY KEY (pk_super)`
			`REFERENCES SUPER (pk_super);`
			`ALTER TABLE SUB1 ADD CONSTRAINT FK_SUB1 FOREIGN KEY (pk_super)`
			`REFERENCES SUPER (pk_super) ON DELETE CASCADE;`
			`ALTER TABLE SUB2 ADD pk_super tipo_atributo NOT NULL;`
			`ALTER TABLE SUB2 ADD CONSTRAINT PK_SUB2 PRIMARY KEY (pk_super)`
			`REFERENCES SUPER (pk_super);`
			`ALTER TABLE SUB2 ADD CONSTRAINT FK_SUB2 FOREIGN KEY (pk_super)`
			`REFERENCES SUPER (pk_super) ON DELETE CASCADE;`
			
			`CREATE OR REPLACE TRIGGER genspecTrigger`^{i} `AFTER INSERT OR DELETE`
			`OR UPDATE ON SUB`^{i} `%i=1,j=2 or i=2,j=1`
			`REFERENCING NEW AS n OLD as o FOR EACH ROW`
			`DECLARE X number;`
			`BEGIN`
			`IF INSERTING THEN`
			`SELECT COUNT(*) INTO X FROM SUB`^{j} `c WHERE c.pk_super = :n.`
			`pk_super;`
			`IF (X != 0) THEN RAISE_APPLICATION_ERROR(-20000, 'Violacao`
			`detectada!'); END IF;`
			`END IF;`
			`END;`

3.3. Implementação do Módulo no TerraER

De forma a concretizar a extensão, na UI (Interface do Usuário) do TerraER, foi criada a ação "Gerar DDL" que usa o método`actionPerformed` da classe `GenerateDDLAction`. O método `actionPerformed` invoca os métodos `generateTables`, `generatePrimaryKey`, `generatePartialKey`, `generateGenSpec`, `generateRelationships`, `generateMultivaluedAttribute` e `generateDerivedAttribute`, na qual cada método executa um passo do Algoritmo 1. A ideia é permitir ao usuário ver a DDL gerada de seu modelo ER, conforme pode ser observado na Figura 4. Também foram feitas modificações para armazenar os tipos e nulidade dos atributos (veja o canto inferior direito da Figura 4). A classe `GenerateDDLAction` está publicamente disponível em: `https://github.com/rterrabh/TerraER`.

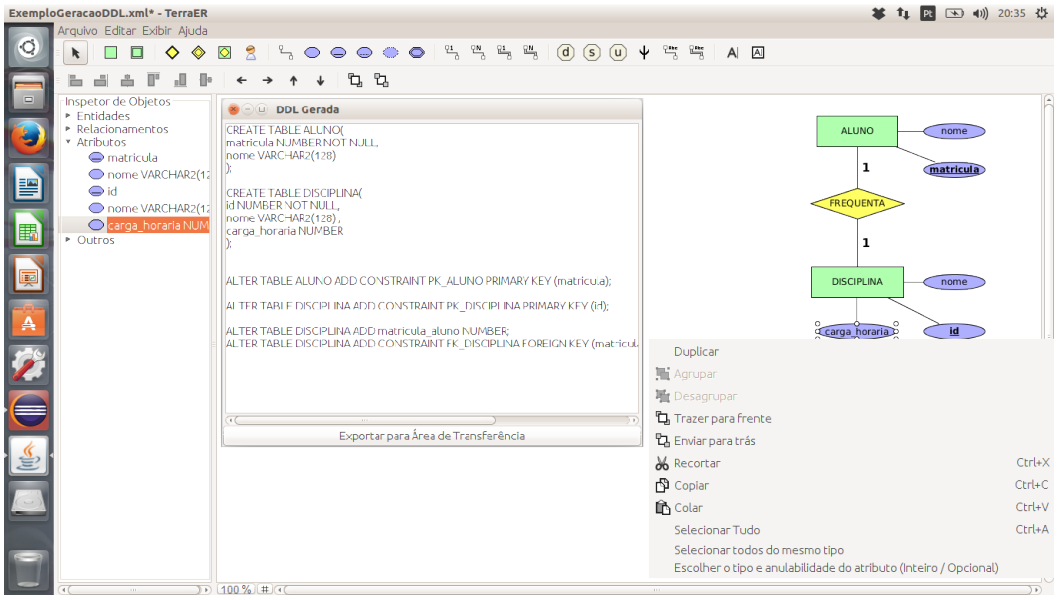


Figura 4. Mapemeamento de um modelo ER para DDL da SQL pelo TerraER

3.4. Limitações

A extensão apresentada neste artigo abrange a maioria dos elementos presentes no modelo ER estendido, porém algumas funcionalidades ainda não foram implementadas como atributos compostos, atributos chave compostos, atributos complexos, entidades fracas que

contém outras entidades fracas, auto-relacionamento, relacionamentos ternários e n-ários, e o operador união.

4. Avaliação Controlada

Como forma de avaliação da extensão desenvolvida, foi elaborado um modelo ER na qual abrange a maioria das construções presentes no modelo ER estendido. As subseções abaixo detalham o modelo criado e a avaliação do processo proposto.

4.1. Modelo Bancário

A Figura 5 ilustra um modelo bancário elaborado para avaliar o processo proposto, o qual foi implementado como uma extensão da ferramenta TerraER. Observa-se que tal modelo possui a maioria das construções do modelo ER e do modelo ER estendido[5], com exceção das limitações descritas na Seção 3.4.

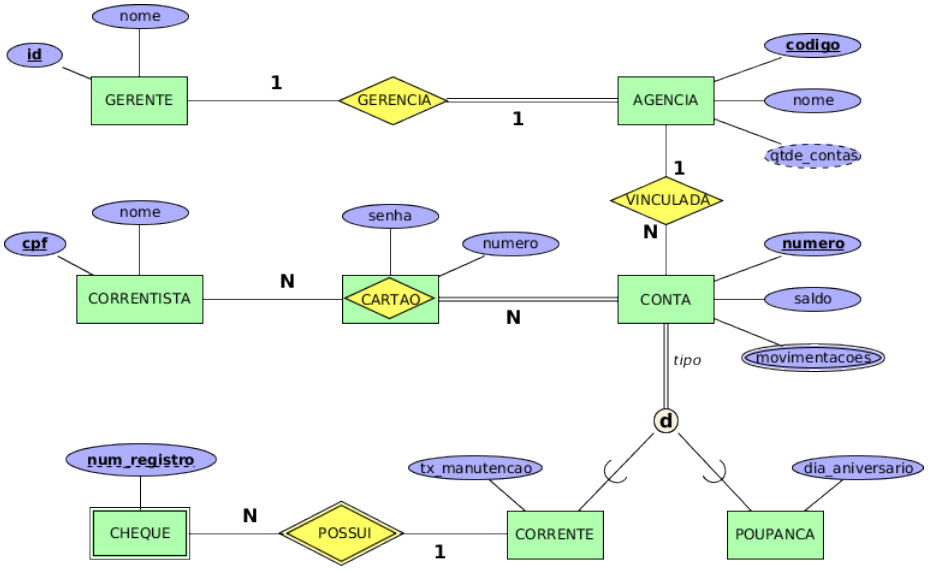


Figura 5. Modelo Bancário

Pode-se constatar as construções que fazem parte do modelo ER, como entidades fortes (`GERENTE`, `AGENCIA`, `CORRENTISTA`, `CONTA`, `CORRENTE` e `POUPANCA`), entidade fraca (`CHEQUE`), entidade relacionamento (`CARTAO`), atributos (*nome*, *senha*, *numero*, *saldo*, *tx_manutencao* e *dia_aniversario*), chaves primárias (*id*, *codigo*, *cpf* e *numero*), chaves parciais (*canhoto*), atributo derivado (*qtde_contas*), atributo multivalorado (*movimentações*), relacionamento 1:1 com participação total em um dos lados (`GERENTE` gerencia `AGENCIA`), relacionamento 1:N com participação parcial (`CONTA` vinculada a `AGENCIA`), relacionamento fraco (`CORRENTE` possui `CHEQUE`), e relacionamento M:N (`CORRENTISTA`/`CARTAO`/`CONTA`).

[5]Com exceção de alguns elementos do modelo ER, como atributos compostos, atributos chave compostos, atributos complexos, entidades fracas que contém outras entidades fracas, auto-relacionamento, relacionamento 1:1 com participação parcial, relacionamento 1:1 com participação total em ambos os lados, relacionamento 1:N com participação total no lado N e relacionamento 1:N com participação total em ambos os lados. Há também algumas exceções no modelo ER estendido como generalização/especialização com restrição de disjunção e restrição de completude parcial, generalização/especialização sobrepostas e restrição de completude parcial e total.

Uma das construções do modelo ER estendido também se faz presente nesta figura através da generalização/especialização, mais especificamente a entidade `CONTA` é especializada nas subclasses `CORRENTE` e `POUPANCA`, possui restrição de disjunção e restrição de completude total.

4.2. Avaliação do Processo Proposto

O Algoritmo 1 e as Tabelas de Mapeamento 6, 7 e 8 (Apêndice A) permitem que a extensão proposta gere modelos ER de forma automática. Nesta seção, o processo proposto é aplicado no modelo bancário descrito na Seção 4.1, cuja DDL resultante é reportada no Apêndice B.

Portanto, após a aplicação do processo no modelo bancário, foram realizados testes para validar a DDL gerada de forma automática, os quais estão detalhados no Apêndice C. Para fins ilustrativos, um pequeno subconjunto da DDL gerada e dos testes são descritos a seguir.

Regra de mapeamento #1: Para cada entidade forte e fraca presentes no diagrama, deve-se criar uma tabela. A Listagem 1 demonstra como a DDL deve ser gerada. A linha 1 exibe a regra de mapeamento e a linha 2 ilustra um exemplo da aplicação da regra na entidade forte `GERENTE` presente na Figura 5.

```
1 --CREATE TABLE TABELA (nome_atributo tipo_atributo nulidade_atributo);
2 CREATE TABLE GERENTE (id NUMBER NOT NULL, nome VARCHAR2(100));
```

Listagem 1. Regra de Mapeamento #1: Criação da Tabela `GERENTE`

Testes: Na Listagem 2, a criação da entidade forte `GERENTE` é testada por meio da inserção de valores em sua tabela. As linhas 1 a 10 testam a inserção de valores na entidade forte `GERENTE` contendo o atributo *id* nulo. Como no momento de criação foi especificado que o atributo *id* não poderia ser nulo, isso acarreta em um erro. Já as linhas 12 a 21 fazem a inserção de valores corretamente, garantindo que a entidade forte seja criada corretamente.

```
1  DECLARE
2   gerenteId GERENTE.id%TYPE;
3   gerenteNome GERENTE.nome%TYPE;
4  BEGIN
5    INSERT INTO GERENTE (id, nome) VALUES (null, 'Arthur') RETURNING
6    id, nome INTO gerenteId, gerenteNome;
7
8    dbms_output.put_line(gerenteId);
9    dbms_output.put_line(gerenteNome);
10 END;
11
12 DECLARE
13  gerenteId2 GERENTE.id%TYPE;
14  gerenteNome2 GERENTE.nome%TYPE;
15 BEGIN
16   INSERT INTO GERENTE (id, nome) VALUES (1, 'Alan') RETURNING
17   id, nome INTO gerenteId2, gerenteNome2;
18
19   dbms_output.put_line(gerenteId2);
20   dbms_output.put_line(gerenteNome2);
21 END;
```

Listagem 2. Teste da Regra de Mapeamento #1: Criação da Tabela `GERENTE`

Regra de mapeamento #4: Para relacionar a entidade fraca com a sua entidade forte, deve-se incluir a chave primária da entidade forte na relação fraca. A chave primária da

relação fraca será a chave primária da relação forte concatenada com a chave parcial. A Listagem 3 demonstra como a DDL deve ser gerada. As linhas 1 a 3 apresentam a regra de mapeamento, e as linhas 4 a 6 demonstram a DDL gerada da entidade fraca CHEQUE presente na Figura 5.

```
1 --ALTER TABLE FRACA ADD pk_forte tipo_atributo NOT NULL;
2 --ALTER TABLE FRACA ADD CONSTRAINT PK_FRACA PRIMARY KEY (pk_fraca, pk_forte);
3 --ALTER TABLE FRACA ADD CONSTRAINT FK_FRACA FOREIGN KEY (pk_forte) REFERENCES FORTE (pk_forte);
4 ALTER TABLE CHEQUE ADD numero_conta NUMBER NOT NULL;
5 ALTER TABLE CHEQUE ADD CONSTRAINT PK_CHEQUE PRIMARY KEY (num_registro, numero_conta);
6 ALTER TABLE CHEQUE ADD CONSTRAINT FK_CHEQUE FOREIGN KEY (numero_conta) REFERENCES CONTA
      (numero);
```

Listagem 3. Regra de Mapeamento #4: Criação de Chaves Parciais na Tabela CHEQUE

Testes: Na Listagem 4, são inseridos valores na entidade fraca CHEQUE de forma que a saída exiba esses valores inseridos, consequentemente garantindo a corretude do método de criação da chave parcial. Observe que, no atributo *numero_conta*, é inserido um valor que se refere a entidade CORRENTE, o que evidencia a característica de entidade fraca. As linhas 1 a 10 demonstram o primeiro teste, na qual é feito a inserção de valor nulo no atributo *num_registro*, o que gera um erro uma vez que o atributo foi declarado de forma a não aceitar valores nulos. Já as linhas 12 a 21 compõem o segundo teste, na qual o valor inserido no atributo *numero_conta* se refere a uma entidade forte que não existe, o que leva a uma violação de restrição de integridade de chave estrangeira. Por fim, as linhas 23 a 32 apresentam a inserção correta de valores na entidade fraca CHEQUE e os valores inseridos são exibidos na saída.

```
1  DECLARE
2   chequeNumRegistro CHEQUE.num_registro%TYPE;
3   chequeCcNumero CHEQUE.numero_conta%TYPE;
4  BEGIN
5    INSERT INTO CHEQUE (num_registro, numero_conta) VALUES (null, 616279) RETURNING
6    num_registro, numero_conta INTO chequeNumRegistro, chequeCcNumero;
7
8    dbms_output.put_line(chequeNumRegistro);
9    dbms_output.put_line(chequeCcNumero);
10 END;
11
12 DECLARE
13  chequeNumRegistro2 CHEQUE.num_registro%TYPE;
14  chequeCcNumero2 CHEQUE.numero_conta%TYPE;
15 BEGIN
16   INSERT INTO CHEQUE (num_registro, numero_conta) VALUES (1, 616260) RETURNING
17   num_registro, numero_conta INTO chequeNumRegistro2, chequeCcNumero2;
18
19   dbms_output.put_line(chequeNumRegistro2);
20   dbms_output.put_line(chequeCcNumero2);
21 END;
22
23 DECLARE
24  chequeNumRegistro3 CHEQUE.num_registro%TYPE;
25  chequeCcNumero3 CHEQUE.numero_conta%TYPE;
26 BEGIN
27   INSERT INTO CHEQUE (num_registro, numero_conta) VALUES (1, 616279) RETURNING
28   num_registro, numero_conta INTO chequeNumRegistro3, chequeCcNumero3;
29
30   dbms_output.put_line(chequeNumRegistro3);
31   dbms_output.put_line(chequeCcNumero3);
32 END;
```

Listagem 4. Teste da Regra de Mapeamento #4: Criação de Chaves Parciais na Tabela CHEQUE

Regra de mapeamento #6: Quando um dos lados do relacionamento possui participação total e o outro participação parcial, armazene a chave primária do lado com restrição de participação parcial no lado com restrição de participação total. A DDL para essa regra é apresentada na Listagem 5. As linhas 1 a 3 apresentam a regra de mapeamento. Como a Figura 5 especifica que um `GERENTE` deve gerenciar uma `AGENCIA`, as linhas 4 a 6 ilustram a DDL gerada na qual corresponde a regra, pois, na entidade `AGENCIA`, é criada uma chave estrangeira na qual referencia a entidade `GERENTE`.

```
1 --ALTER TABLE TABELA2 ADD pk_tabela1 tipo_atributo NOT NULL;
2 --ALTER TABLE TABELA2 ADD CONSTRAINT FK_TABELA2 FOREIGN KEY (pk_tabela1)
3 --REFERENCES TABELA1 (pk_tabela1);
4 --ALTER TABLE TABELA2 ADD CONSTRAINT UQ_TABELA2 UNIQUE (pk_tabela1);
5 ALTER TABLE AGENCIA ADD id_gerente NUMBER NOT NULL;
6 ALTER TABLE AGENCIA ADD CONSTRAINT FK_AGENCIA FOREIGN KEY (id_gerente)
7 REFERENCES GERENTE (id);
8 ALTER TABLE AGENCIA ADD CONSTRAINT UQ_AGENCIA UNIQUE (id_gerente);
```

Listagem 5. Regra de Mapeamento #6: Criação de Relacionamento 1:1 (Participação Total) na Tabela `AGENCIA`

Testes: No primeiro teste (linhas 1 a 12), realiza-se a inserção de valores na entidade `AGENCIA`, porém define-se o valor do atributo *código* como nulo, o que acarreta em um erro devido ao fato do atributo não aceitar valores nulos. Já as linhas 14 a 26 compõem o segundo teste, cuja finalidade é verificar a restrição de integridade de chave estrangeira. Observa-se que, se não for indicado um valor válido para o atributo *id_gerente*, o erro é imediatamente observado na inserção. Por fim, as linhas 28 a 40 indicam o último teste, na qual todos os valores são inseridos corretamente e são exibidos na saída.

```
1  DECLARE
2   agenciaCodigo AGENCIA.codigo%TYPE;
3   agenciaNome AGENCIA.nome%TYPE;
4   agenciaGerenteId AGENCIA.id_gerente%TYPE;
5  BEGIN
6    INSERT INTO AGENCIA (codigo, nome, id_gerente) VALUES (null, 'Bradesco', 1) RETURNING
7    codigo, nome, id_gerente INTO agenciaCodigo, agenciaNome, agenciaGerenteId;
8    dbms_output.put_line(agenciaCodigo);
9    dbms_output.put_line(agenciaNome);
10   dbms_output.put_line(agenciaGerenteId);
11 END;
12
13 DECLARE
14  agenciaCodigo2 AGENCIA.codigo%TYPE;
15  agenciaNome2 AGENCIA.nome%TYPE;
16  agenciaGerenteId2 AGENCIA.id_gerente%TYPE;
17 BEGIN
18   INSERT INTO AGENCIA (codigo, nome, id_gerente) VALUES (3646, 'Santander', null)
19   RETURNING codigo, nome, id_gerente
20   INTO agenciaCodigo2, agenciaNome2, agenciaGerenteId2;
21   dbms_output.put_line(agenciaCodigo2);
22   dbms_output.put_line(agenciaNome2);
23   dbms_output.put_line(agenciaGerenteId2);
24 END;
25
26 DECLARE
27  agenciaCodigo3 AGENCIA.codigo%TYPE;
28  agenciaNome3 AGENCIA.nome%TYPE;
29  agenciaGerenteId3 AGENCIA.id_gerente%TYPE;
30 BEGIN
31   INSERT INTO AGENCIA (codigo, nome, id_gerente) VALUES (3646, 'Banco do Brasil', 1)
32   RETURNING codigo, nome, id_gerente
33   INTO agenciaCodigo3, agenciaNome3, agenciaGerenteId3;
34   dbms_output.put_line(agenciaCodigo3);
35   dbms_output.put_line(agenciaNome3);
36   dbms_output.put_line(agenciaGerenteId3);
37 END;
```

Listagem 6. Teste da Regra de Mapeamento #6: Criação de Relacionamento 1:1 (Participação Total) na Tabela `AGENCIA`

5. Considerações Finais

Entidade Relacionamento (ER) é um modelo importante visto em disciplinas de projeto de banco de dados e adotado principalmente no processo Ensino-aprendizagem. Dessa forma, o processo de projeto se inicia por um diagrama conceitual de alto-nível, passando por um diagrama conceitual de nível lógico e, por fim, a implementação do banco de dados utilizando a DDL da SQL. Embora essa seja uma prática correta, o problema é que seguir esse processo, de análise e correção de dezenas de projetos de alunos, é uma tarefa que pode demandar muito tempo do professor.

Diante disso, este artigo propôs uma metodologia para o mapeamento de modelos ER para DDL da SQL. O processo consiste em converter construções do modelo ER estendido em DDL da SQL com base em regras de uma tabela de mapeamento ER/DDL. Assim, foi projetado um algoritmo que sistematicamente executa as regras da tabela de mapeamento, o qual foi integrado à ferramenta TerraER para possibilitar a avaliação do processo proposto. Um modelo bancário – que contempla a maioria das construções ER – foi então elaborado com o propósito de avaliar a DDL gerada.

Como principais contribuições deste trabalho, espera-se (i) auxiliar professores na correção de exercícios e trabalhos; e (ii) auxiliar profissionais da área a criar um projeto físico de forma prática e ágil. Como trabalhos futuros, pretende-se implementar as funcionalidades de atributos compostos, atributos chave compostos, atributos complexo, entidades fracas que contém outras entidades fracas, auto-relacionamento, relacionamentos ternários e n-ários, e o operador união.

Referências

[1] Chen, Peter Pin Shan: *The Entity-Relationship Model - Toward a Unified View of Data*. ACM Transactions on Database Systems, 1(1):9–36, 1976.

[2] Date, C. J.: *Introdução a Sistemas de Banco de Dados*. Elsevier, 8ª edição, 2004.

[3] Silberschatz, Abraham, Henry F. Korth e S. Sudarshan: *Sistema de Banco de Dados*. Elsevier, 5ª edição, 2006.

[4] Elmasri, Ramez e Shamkant B. Navathe: *Sistemas de Banco de Dados*. Pearson Addison Wesley, 6ª edição, 2011.

[5] Rocha, Henrique e Ricardo Terra: *TerraER: Uma Ferramenta voltada ao Ensino do Modelo de Entidade-Relacionamento*. Em *VI Escola Regional de Banco de Dados (ERBD)*, páginas 1–4, 2010.

[6] Rocha, Henrique e Ricardo Terra: *TerraER - an Academic Tool for ER Modeling*. Methods and Tools, 1(3):38–41, 2013.

[7] Garcia-Molina, Hector: *Database Systems: The Complete Book*. Pearson Prentice Hall, 2ª edição, 2009.

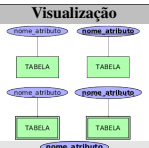
[8] Teorey, Toby J., Dongqing Yang e James P. Fry: *A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model*. ACM Computing Surveys, 18(2):197–222, 1986.

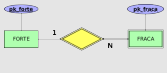
[9] Bagui, Sikha e Richard Earp: *Database Design Using Entity-Relationship Diagrams*. CRC Press, 2ª edição, 2012.

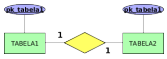
A. Tabela de Mapeamento

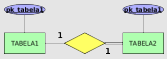
As Tabelas 6, 7 e 8 apresentam as tabelas de mapeamento. Em cada linha de cada tabela, encontram-se respectivamente o número da regra de mapeamento, a descrição do elemento ER, a sua visualização na ferramenta TerraER e, por fim, a DDL que o algoritmo proposto gera para tal elemento. Por exemplo, a regra de mapeamento #1 se refere a criação de tabelas, podendo ser uma entidade forte ou fraca.

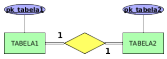
Tabela 6. Tabela de Mapeamento 1/3

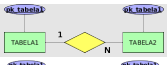
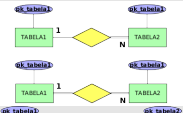
Regra	Elemento ER	Visualização	DDL Equivalente
#1	Criação de Tabelas		`CREATE TABLE TABELA (nome_atributo tipo_atributo nulidade_atributo);`
#2	Atributo Chave		`ALTER TABLE TABELA ADD CONSTRAINT PK_TESTE PRIMARY KEY ( nome_atributo);`
#3	Atributo Chave (Mais de uma Chave)		`ALTER TABLE TABELA ADD CONSTRAINT PK_TABELA PRIMARY KEY ( nome_atributo1);` `ALTER TABLE TABELA ADD CONSTRAINT UQ_TABELA UNIQUE ( nome_atributo2);`

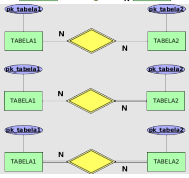
#4	Chaves Parciais		`ALTER TABLE FRACA ADD (pk_forte tipo_atributo NOT NULL);`
			`ALTER TABLE FRACA ADD CONSTRAINT PK_FRACA PRIMARY KEY (pk_fraca, pk_forte);`
			`ALTER TABLE FRACA ADD CONSTRAINT FK_FRACA FOREIGN KEY (pk_forte) REFERENCES FORTE (pk_forte);`

#5	Relacionamento 1:1 (Participação Parcial)		`ALTER TABLE TABELA2 ADD pk_tabela1 tipo-atributo;`
			`ALTER TABLE TABELA2 ADD CONSTRAINT FK_TABELA2 FOREIGN KEY ( pk_tabela1) REFERENCES TABELA1 (pk_tabela1);`
			`ALTER TABLE TABELA2 ADD CONSTRAINT UQ_TABELA2 UNIQUE (pk_tabela1);`

#6	Relacionamento 1:1 (Participação Total)		`ALTER TABLE TABELA2 ADD pk_tabela1 tipo_atributo NOT NULL;`
			`ALTER TABLE TABELA2 ADD CONSTRAINT FK_TABELA2 FOREIGN KEY ( pk_tabela1) REFERENCES TABELA1 (pk_tabela1);`
			`ALTER TABLE TABELA2 ADD CONSTRAINT UQ_TABELA2 UNIQUE (pk_tabela1);`

#7	Relacionamento 1:1 (Participação Total em Ambos os Lados)		`CREATE TABLE TABELA1_TABELA2(pk_tabela1 tipo_atributo NOT NULL, pk_tabela2 tipo_atributo NOT NULL);`
			`ALTER TABLE TABELA1_TABELA2 ADD CONSTRAINT PK_TABELA1_TABELA2 PRIMARY KEY (pk_tabela1);`
			`ALTER TABLE TABELA1_TABELA2 ADD CONSTRAINT FK_TABELA1_TABELA2 FOREIGN KEY (pk_tabela1) REFERENCES TABELA1 (pk_tabela1);`
			`ALTER TABLE TABELA1_TABELA2 ADD CONSTRAINT FK2_TABELA1_TABELA2 FOREIGN KEY (pk_tabela2) REFERENCES TABELA2 (pk_tabela2);`
			`ALTER TABLE TABELA1_TABELA2 ADD CONSTRAINT UQ_TABELA1_TABELA2 UNIQUE (pk_tabela2);`

#8	Relacionamento 1:N (Participação Parcial)		`ALTER TABLE TABELA2 ADD pk_tabela1 tipo_atributo;` `ALTER TABLE TABELA2 ADD CONSTRAINT FK_TABELA2 FOREIGN KEY ( pk_tabela1) REFERENCES TABELA1 (pk_tabela1);`
#9	Relacionamento 1:N (Participação Total do Lado N/Total em Ambos os Lados)		`ALTER TABLE TABELA2 ADD pk_tabela1 tipo_atributo NOT NULL;` `ALTER TABLE TABELA2 ADD CONSTRAINT FK_TABELA2 FOREIGN KEY ( pk_tabela1) REFERENCES TABELA1 (pk_tabela1);`

#10	Relacionamento M:N (Participação Parcial / Total do Lado N / Total em Ambos os Lados)		`CREATE TABLE TABELA1_TABELA2( pk_tabela2 tipo_atributo NOT NULL, pk_tabela1 tipo_atributo NOT NULL );`
			`ALTER TABLE TABELA1_TABELA2 ADD CONSTRAINT PK_TABELA1_TABELA2 PRIMARY KEY (pk_tabela1,pk_tabela2);`
			`ALTER TABLE TABELA1_TABELA2 ADD CONSTRAINT FK_TABELA1_TABELA2 FOREIGN KEY (pk_tabela1) REFERENCES TABELA1 (pk_tabela1);`
			`ALTER TABLE TABELA1_TABELA2 ADD CONSTRAINT FK2_TABELA1_TABELA2 FOREIGN KEY (pk_tabela2) REFERENCES TABELA2 (pk_tabela2);`

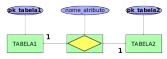
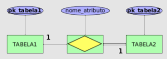
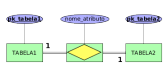
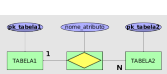
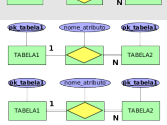
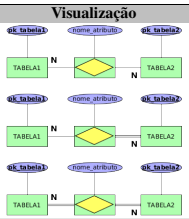
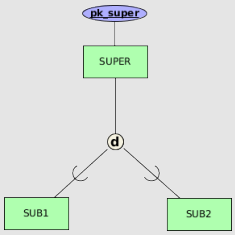
#11	Entidade Relacionamento 1:1 (Participação Parcial)		`REGRA #5 +` `ALTER TABLE TABELA2 ADD nome_atributo tipo-atributo nulidade_atributo;`
#12	Entidade Relacionamento 1:1 (Participação Total)		`REGRA #6 +` `ALTER TABLE TABELA2 ADD nome_atributo tipo-atributo nulidade_atributo;`
#13	Entidade Relacionamento 1:1 (Participação Total em Ambos os Lados)		`REGRA #7 +` `ALTER TABLE TABELA1_TABELA2 ADD nome_atributo tipo-atributo nulidade_atributo;`
#14	Entidade Relacionamento 1:N (Participação Parcial)		`REGRA #8 +` `ALTER TABLE TABELA2 ADD nome_atributo tipo-atributo nulidade_atributo;`
#15	Entidade Relacionamento 1:N (Participação Total do Lado N/Total em Ambos os Lados)		`REGRA #9 +` `ALTER TABLE TABELA2 ADD nome_atributo tipo-atributo nulidade_atributo;`

Tabela 7. Tabela de Mapeamento 2/3

Regra	Elemento ER	Visualização	DDL Equivalente
#16	Entidade Relacionamento M:N (Participação Parcial / Total do Lado N / Total em Ambos os Lados)		REGRA #10 + ALTER TABLE TABELA1_TABELA2 ADD nome_atributo tipo-atributo nulidade_atributo;

#17	Generalização e Especialização (Disjunção - Participação Parcial)		ALTER TABLE SUB1 ADD pk_super tipo_atributo NOT NULL;
			ALTER TABLE SUB1 ADD CONSTRAINT PK_SUB1 PRIMARY KEY (pk_super)
			REFERENCES SUPER (pk_super);
			ALTER TABLE SUB1 ADD CONSTRAINT FK_SUB1 FOREIGN KEY (pk_super)
			REFERENCES SUPER (pk_super) ON DELETE CASCADE;
			ALTER TABLE SUB2 ADD pk_super tipo_atributo NOT NULL;
			ALTER TABLE SUB2 ADD CONSTRAINT PK_SUB2 PRIMARY KEY (pk_super)
			REFERENCES SUPER (pk_super);
			ALTER TABLE SUB2 ADD CONSTRAINT FK_SUB2 FOREIGN KEY (pk_super)
			REFERENCES SUPER (pk_super) ON DELETE CASCADE;
			
			CREATE OR REPLACE TRIGGER genspecTrigger_SUB^i AFTER INSERT OR
			DELETE OR UPDATE ON SUB^i %i=1,j=2 or i=2,j=1
			REFERENCING NEW AS n OLD as o FOR EACH ROW
			DECLARE X number;
			BEGIN
			IF INSERTING THEN
			SELECT COUNT(*) INTO X FROM SUB^j c WHERE c.pk_super = :n.
			pk_super;
			IF (X != 0) THEN RAISE_APPLICATION_ERROR(-20000, 'Violacao␣
			detectada!'); END IF;
			END IF;
			END;

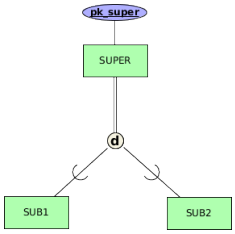
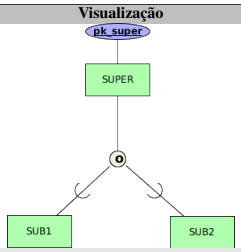
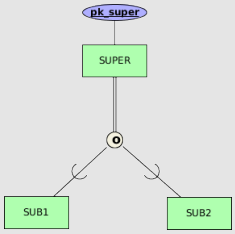
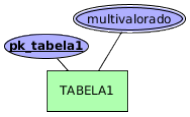
#18	Generalização e Especialização (Disjunção - Participação Total)		ALTER TABLE SUB1 ADD pk_super tipo_atributo NOT NULL;
			ALTER TABLE SUB1 ADD CONSTRAINT PK_SUB1 PRIMARY KEY (pk_super)
			REFERENCES SUPER (pk_super);
			ALTER TABLE SUB1 ADD CONSTRAINT FK_SUB1 FOREIGN KEY (pk_super)
			REFERENCES SUPER (pk_super) INITIALLY DEFERRED DEFERRABLE;
			ALTER TABLE SUB2 ADD pk_super tipo_atributo NOT NULL;
			ALTER TABLE SUB2 ADD CONSTRAINT PK_SUB2 PRIMARY KEY (pk_super)
			REFERENCES SUPER (pk_super);
			ALTER TABLE SUB2 ADD CONSTRAINT FK_SUB2 FOREIGN KEY (pk_super)
			REFERENCES SUPER (pk_super) INITIALLY DEFERRED DEFERRABLE;
			
			CREATE OR REPLACE TRIGGER genspecTrigger_SUPER AFTER INSERT OR
			DELETE OR UPDATE ON SUPER
			REFERENCING NEW AS n OLD as o FOR EACH ROW
			DECLARE X number; Y number;
			BEGIN
			IF INSERTING THEN
			SELECT COUNT(*) INTO X FROM SUB1 c WHERE c.pk_super = :n.
			pk_super;
			SELECT COUNT(*) INTO Y FROM SUB2 c WHERE c.pk_super = :n.
			pk_super;
			IF (X + Y < 1) THEN RAISE_APPLICATION_ERROR(-20000, 'Violacao␣
			detectada!'); END IF;
			END IF;
			END;
			
			CREATE OR REPLACE TRIGGER genspecTrigger_SUB^j AFTER INSERT OR
			DELETE OR UPDATE ON SUB^i %i=1,j=2 or i=2,j=1
			REFERENCING NEW AS n OLD as o FOR EACH ROW
			DECLARE X number; Y number;
			BEGIN
			IF INSERTING THEN
			SELECT COUNT(*) INTO X FROM SUB^j c WHERE c.pk_super = :n.
			pk_super;
			IF (C1 != 0) THEN RAISE_APPLICATION_ERROR(-20001, 'Violacao␣
			detectada!');END IF;
			END IF;
			IF DELETING THEN
			SELECT COUNT (*) INTO Y FROM SUPER c WHERE c.pk_super = :o.
			pk_super;
			IF (Y != 0) THEN RAISE_APPLICATION_ERROR(-20002, 'Violacao␣
			detectada!'); END IF;
			END IF;
			IF UPDATING THEN
			IF (:n.pk_super != :o.pk_super) THEN
			SELECT COUNT (*) INTO Y FROM SUPER c WHERE c.pk_super = :o.
			pk_super;
			IF (Y != 0) THEN RAISE_APPLICATION_ERROR(-20003, 'Violacao␣
			detectada!'); END IF;
			SELECT COUNT(*) INTO X FROM SUB^j c WHERE c.pk_super = :n.
			pk_super;
			IF (X != 0) THEN RAISE_APPLICATION_ERROR(-20004, 'Violacao␣
			detectada!'); END IF;
			END IF;
			END IF;
			END;

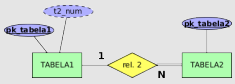
Tabela 8. Tabela de Mapeamento 3/3

Regra	Elemento ER	Visualização	DDL Equivalente

#19	Generalização e Especialização (Sobreposição - Participação Parcial)		`ALTER TABLE SUB1 ADD pk_super tipo_atributo NOT NULL;`
			`ALTER TABLE SUB1 ADD CONSTRAINT PK_SUB1 PRIMARY KEY (pk_super)`
			`REFERENCES SUPER (pk_super);`
			`ALTER TABLE SUB1 ADD CONSTRAINT FK_SUB1 FOREIGN KEY (pk_super)`
			`REFERENCES SUPER (pk_super) ON DELETE CASCADE;`
			`ALTER TABLE SUB2 ADD pk_super tipo_atributo NOT NULL;`
			`ALTER TABLE SUB2 ADD CONSTRAINT PK_SUB2 PRIMARY KEY (pk_super)`
			`REFERENCES SUPER (pk_super);`
			`ALTER TABLE SUB2 ADD CONSTRAINT FK_SUB2 FOREIGN KEY (pk_super)`
			`REFERENCES SUPER (pk_super) ON DELETE CASCADE;`

#20	Generalização e Especialização (Sobreposição - Participação Total)		`ALTER TABLE SUB1 ADD pk_super tipo_atributo NOT NULL;`
			`ALTER TABLE SUB1 ADD CONSTRAINT PK_SUB1 PRIMARY KEY (pk_super)`
			`REFERENCES SUPER (pk_super);`
			`ALTER TABLE SUB1 ADD CONSTRAINT FK_SUB1 FOREIGN KEY (pk_super)`
			`REFERENCES SUPER (pk_super) INITIALLY DEFERRED DEFERRABLE;`
			`ALTER TABLE SUB2 ADD pk_super tipo_atributo NOT NULL;`
			`ALTER TABLE SUB2 ADD CONSTRAINT PK_SUB2 PRIMARY KEY (pk_super)`
			`REFERENCES SUPER (pk_super);`
			`ALTER TABLE SUB2 ADD CONSTRAINT FK_SUB2 FOREIGN KEY (pk_super)`
			`REFERENCES SUPER (pk_super) INITIALLY DEFERRED DEFERRABLE;`
			
			`CREATE OR REPLACE TRIGGER genspecTrigger_SUPER AFTER INSERT OR`
			`DELETE OR UPDATE ON SUPER`
			`REFERENCING NEW AS n OLD as o FOR EACH ROW`
			`DECLARE X number; Y number;`
			`BEGIN`
			`IF INSERTING THEN`
			`SELECT COUNT(*) INTO X FROM SUB1 c WHERE c.pk_super = :n.`
			`pk_super;`
			`SELECT COUNT(*) INTO Y FROM SUB2 c WHERE c.pk_super = :n.`
			`pk_super;`
			`IF (X + Y < 1) THEN RAISE_APPLICATION_ERROR(-20000, 'Violacao`
			`detectada!'); END IF;`
			`END IF;`
			`END;`
			
			`CREATE OR REPLACE TRIGGER genspecTrigger_SUB`^{j} `AFTER INSERT OR`
			`DELETE OR UPDATE ON SUB`^{i} `%i=1,j=2 or i=2,j=1`
			`REFERENCING NEW AS n OLD as o FOR EACH ROW`
			`DECLARE X number;`
			`BEGIN`
			`IF DELETING THEN`
			`SELECT COUNT(*) INTO X FROM SUB`^{j} `c WHERE c.pk_super = :n.`
			`pk_super;`
			`IF (X < 1) THEN RAISE_APPLICATION_ERROR(-20001, 'Violacao`
			`detectada!'); END IF;`
			`END IF;`
			`END;`

#21	Atributo Multivalorado		`CREATE TABLE TABELA1_MULTIVALORADO(`
			`pk_tabela1 tipo_atributo NOT NULL,`
			`pk_multivalorado tipo_atributo NOT NULL,`
			`atributo tipo_atributo nulidade_atributo`
			`);`
			`ALTER TABLE TABELA1_MULTIVALORADO ADD CONSTRAINT`
			`PK_TABELA1_MULTIVALORADO PRIMARY KEY (pk_multivalorado);`
			`ALTER TABLE TABELA1_MULTIVALORADO ADD CONSTRAINT`
			`FK_TABELA1_MULTIVALORADO FOREIGN KEY (pk_tabela1) REFERENCES`
			`TABELA1 (pk_tabela1);`

#22	Atributo Derivado		`CREATE OR REPLACE VIEW VW_TABELA1 AS (`
			`SELECT t1.pk_tabela1, COUNT(t2.pk_tabela2) AS t2_num`
			`FROM TABELA2 t2, TABELA1 t1`
			`WHERE t1.pk_tabela1 = t2.pk_tabela1`
			`GROUP BY t1.pk_tabela1`
			`);`

B. DDL do Modelo Bancário

A Listagem 7 apresenta a DDL do modelo bancário gerada automaticamente pela extensão. É possível constatar que a DDL é fiel ao modelo bancário ilustrado na Figura 5, pois o Algoritmo 1 utiliza cada regra de mapeamento ilustrado nas Tabelas de Mapeamento 6, 7 e 8. Comentários foram adicionados ao *script* SQL para demonstrar quais são as regras utilizadas para gerar cada elemento ER. Por exemplo, as linhas 1 a 33 apresentam a DDL gerada para criação de tabelas na qual utiliza a *Regra de Mapeamento #1* presente na Tabela de Mapeamento 6.

```
1  -- Inicio da Regra de Mapeamento #1
2  CREATE TABLE GERENTE(
3  id NUMBER NOT NULL,
4  nome VARCHAR2(128)
5  );
6
7  CREATE TABLE AGENCIA(
8  codigo NUMBER NOT NULL,
9  nome VARCHAR2(128)
10 );
11
12 CREATE TABLE CONTA(
13 saldo NUMBER ,
14 numero NUMBER NOT NULL
15 );
16
17 CREATE TABLE CORRENTE(
18 tx_manutencao NUMBER
19 );
20
21 CREATE TABLE POUPANCA(
22 dia_aniversario NUMBER
23 );
24
25 CREATE TABLE CORRENTISTA(
26 nome VARCHAR2(128) ,
27 cpf NUMBER NOT NULL
28 );
29
30 CREATE TABLE CHEQUE(
31 num_registro NUMBER NOT NULL
32 );
33 -- Fim da Regra de Mapeamento #1
34
35 -- Inicio da Regra de Mapeamento #2
36 ALTER TABLE GERENTE ADD CONSTRAINT PK_GERENTE PRIMARY KEY (id);
37
38 ALTER TABLE AGENCIA ADD CONSTRAINT PK_AGENCIA PRIMARY KEY (codigo);
39
40 ALTER TABLE CONTA ADD CONSTRAINT PK_CONTA PRIMARY KEY (numero);
41
42 ALTER TABLE CORRENTISTA ADD CONSTRAINT PK_CORRENTISTA PRIMARY KEY (cpf);
43 -- Fim da Regra de Mapeamento #2
44
45 -- Inicio da Regra de Mapeamento #4
46 ALTER TABLE CHEQUE ADD numero_conta NUMBER NOT NULL;
47 ALTER TABLE CHEQUE ADD CONSTRAINT PK_CHEQUE PRIMARY KEY (num_registro, numero_conta);
48 ALTER TABLE CHEQUE ADD CONSTRAINT FK_CHEQUE FOREIGN KEY (numero_conta) REFERENCES CONTA (numero);
49 -- Fim da Regra de Mapeamento #4
50
51 -- Inicio da Regra de Mapeamento #18
52 ALTER TABLE CORRENTE ADD numero_conta NUMBER NOT NULL;
53 ALTER TABLE CORRENTE ADD CONSTRAINT PK_CORRENTE PRIMARY KEY (numero_conta) REFERENCES CONTA (numero);
54 ALTER TABLE CORRENTE ADD CONSTRAINT FK_CORRENTE FOREIGN KEY (numero_conta) REFERENCES CONTA (numero) ON DELETE
       CASCADE;
55 ALTER TABLE POUPANCA ADD numero_conta NUMBER NOT NULL;
56 ALTER TABLE POUPANCA ADD CONSTRAINT PK_POUPANCA PRIMARY KEY (numero_conta) REFERENCES CONTA (numero);
57 ALTER TABLE POUPANCA ADD CONSTRAINT FK_POUPANCA FOREIGN KEY (numero_conta) REFERENCES CONTA (numero) ON DELETE
       CASCADE;
58
59 CREATE OR REPLACE TRIGGER genspecTrigger_CONTA AFTER INSERT OR DELETE OR UPDATE ON CONTA
60 REFERENCING NEW AS n OLD as o FOR EACH ROW
61 DECLARE X0 number; X1 number;
62 BEGIN
63   IF INSTERTING THEN
64     SELECT COUNT(*) INTO X0 FROM CORRENTE c WHERE c.numero_conta = :n.numero;
65     SELECT COUNT(*) INTO X1 FROM POUPANCA c WHERE c.numero_conta = :n.numero;
66     IF(X0 + X1 < 1) THEN RAISE_APPLICATION_ERROR(-20000, 'Violacao detectada!'); END IF;
67   END IF;
68 END;
69
70 CREATE OR REPLACE TRIGGER genspecTrigger_CORRENTE AFTER INSERT OR DELETE OR UPDATE ON CORRENTE
71 REFERENCING NEW AS n OLD as o FOR EACH ROW
```

```
DECLARE X0 number; X1 number;
BEGIN
  IF INSERTING THEN
    SELECT COUNT(*) INTO X1 FROM POUPANCA c WHERE c.numero_conta = :n.numero_conta;
    IF(X1 != 0) THEN RAISE_APPLICATION_ERROR(-20000, 'Violacao detectada!'); END IF;
  END IF;
  IF DELETING THEN
    SELECT COUNT(*) INTO X0 FROM CONTA c WHERE c.numero = :o.numero_conta;
    IF(X0 != 0) THEN RAISE_APPLICATION_ERROR(-20001, 'Violacao detectada!'); END IF;
  END IF;
  IF UPDATING THEN
    IF(:n.numero_conta != :o.numero_conta) THEN
      SELECT COUNT(*) INTO X0 FROM CONTA c WHERE c.numero = :o.numero_conta;
      IF(X0 != 0) THEN RAISE_APPLICATION_ERROR(-20002, 'Violacao detectada!'); END IF;
      SELECT COUNT(*) INTO X1 FROM POUPANCA c WHERE c.numero_conta = :n.numero_conta;
      IF(X1 != 0) THEN RAISE_APPLICATION_ERROR(-20003, 'Violacao detectada!'); END IF;
    END IF;
  END IF
END;

CREATE OR REPLACE TRIGGER genspecTrigger_POUPANCA AFTER INSERT OR DELETE OR UPDATE ON POUPANCA
REFERENCING NEW AS n OLD as o FOR EACH ROW
DECLARE X0 number; X1 number;
BEGIN
  IF INSERTING THEN
    SELECT COUNT(*) INTO X1 FROM CORRENTE c WHERE c.numero_conta = :n.numero_conta;
    IF(X1 != 0) THEN RAISE_APPLICATION_ERROR(-20004, 'Violacao detectada!'); END IF;
  END IF;
  IF DELETING THEN
    SELECT COUNT(*) INTO X0 FROM CONTA c WHERE c.numero = :o.numero_conta;
    IF(X0 != 0) THEN RAISE_APPLICATION_ERROR(-20005, 'Violacao detectada!'); END IF;
  END IF;
  IF UPDATING THEN
    IF(:n.numero_conta != :o.numero_conta) THEN
      SELECT COUNT(*) INTO X0 FROM CONTA c WHERE c.numero = :o.numero_conta;
      IF(X0 != 0) THEN RAISE_APPLICATION_ERROR(-20006, 'Violacao detectada!'); END IF;
      SELECT COUNT(*) INTO X1 FROM CORRENTE c WHERE c.numero_conta = :n.numero_conta;
      IF(X1 != 0) THEN RAISE_APPLICATION_ERROR(-20007, 'Violacao detectada!'); END IF;
    END IF;
  END IF
END;
-- Fim da Regra de Mapeamento #18

-- Inicio da Regra de Mapeamento #6
ALTER TABLE AGENCIA ADD id_gerente NUMBER NOT NULL;
ALTER TABLE AGENCIA ADD CONSTRAINT FK_AGENCIA FOREIGN KEY (id_gerente) REFERENCES GERENTE (id);
ALTER TABLE AGENCIA ADD CONSTRAINT UQ_AGENCIA UNIQUE (id_gerente);
-- Fim da Regra de Mapeamento #6

-- Inicio da Regra de Mapeamento #8
ALTER TABLE CONTA ADD codigo_agencia NUMBER;
ALTER TABLE CONTA ADD CONSTRAINT FK_CONTA FOREIGN KEY (codigo_agencia) REFERENCES AGENCIA (codigo);
-- Fim da Regra de Mapeamento #8

-- Inicio da Regra de Mapeamento #16
CREATE TABLE CARTAO(cpf_correntista NUMBER NOT NULL, numero_conta NUMBER NOT NULL);
ALTER TABLE CARTAO ADD CONSTRAINT PK_CARTAO PRIMARY KEY (cpf_correntista, numero_conta);
ALTER TABLE CARTAO ADD CONSTRAINT FK_CARTAO FOREING KEY (cpf_correntista) REFERENCES CORRENTISTA (cpf);
ALTER TABLE CARTAO ADD CONSTRAINT FK2_CARTAO FOREING KEY (numero_conta) REFERENCES CONTA (numero);
ALTER TABLE CARTAO ADD numero NUMBER NOT NULL;
ALTER TABLE CARTAO ADD senha NUMBER NOT NULL;
-- Fim da Regra de Mapeamento #16

-- Inicio da Regra de Mapeamento #21
CREATE TABLE CONTA_MOVIMENTACOES(
  numero_conta NUMBER NOT NULL,
  pk_movimentacoes NUMBER NOT NULL,
  movimentacoes VARCHAR2(128) NOT NULL
);

ALTER TABLE CONTA_MOVIMENTACOES ADD CONSTRAINT PK_CONTA_MOVIMENTACOES PRIMARY KEY (pk_movimentacoes);
ALTER TABLE CONTA_MOVIMENTACOES ADD CONSTRAINT FK_CONTA_MOVIMENTACOES FOREIGN KEY (numero_conta) REFERENCES CONTA (
    numero);
-- Fim da Regra de Mapeamento #21

-- Inicio da Regra de Mapeamento #22
CREATE OR REPLACE VIEW VW_AGENCIA AS (
SELECT a.codigo, COUNT (c.numero) AS qtde_contas FROM AGENCIA a, CONTA c WHERE c.codigo_agencia = a.codigo GROUP BY
    a.codigo
);
-- Fim da Regra de Mapeamento #22
```

Listagem 7. DDL do Modelo Bancário

C. *Script* de Testes

A Listagem 8 apresenta os testes de validação referentes a DDL gerada automaticamente do modelo bancário ilustrado na Figura 5. Comentários foram inseridos de forma a identificar quais elementos ER da DDL são testados. Por exemplo, as linhas 3 a 25 testam a entidade forte `GERENTE`.

```
1  SET serveroutput ON;
2
3  --Inicio do teste da entidade GERENTE
4  DECLARE
5   gerenteId GERENTE.id%TYPE;
6   gerenteNome GERENTE.nome%TYPE;
7  BEGIN
8    INSERT INTO GERENTE (id, nome) VALUES (null, 'Arthur') RETURNING id, nome INTO gerenteId, gerenteNome;
9
10   dbms_output.put_line(gerenteId);
11   dbms_output.put_line(gerenteNome);
12 END;
13 /
14
15 DECLARE
16  gerenteId2 GERENTE.id%TYPE;
17  gerenteNome2 GERENTE.nome%TYPE;
18 BEGIN
19   INSERT INTO GERENTE (id, nome) VALUES (1, 'Alan') RETURNING id, nome INTO gerenteId2, gerenteNome2;
20
21   dbms_output.put_line(gerenteId2);
22   dbms_output.put_line(gerenteNome2);
23 END;
24 /
25 --Fim do teste da entidade GERENTE
26
27 --Inicio do teste da entidade AGENCIA
28 DECLARE
29  agenciaCodigo AGENCIA.codigo%TYPE;
30  agenciaNome AGENCIA.nome%TYPE;
31  agenciaGerenteId AGENCIA.id_gerente%TYPE;
32 BEGIN
33   INSERT INTO AGENCIA (codigo, nome, id_gerente) VALUES (null, 'Bradesco', 1) RETURNING codigo, nome, id_gerente
       INTO agenciaCodigo, agenciaNome, agenciaGerenteId;
34
35   dbms_output.put_line(agenciaCodigo);
36   dbms_output.put_line(agenciaNome);
37   dbms_output.put_line(agenciaGerenteId);
38 END;
39 /
40
41 DECLARE
42  agenciaCodigo2 AGENCIA.codigo%TYPE;
43  agenciaNome2 AGENCIA.nome%TYPE;
44  agenciaGerenteId2 AGENCIA.id_gerente%TYPE;
45 BEGIN
46   INSERT INTO AGENCIA (codigo, nome, id_gerente) VALUES (3646, 'Santander', null) RETURNING codigo, nome, id_gerente
       INTO agenciaCodigo2, agenciaNome2, agenciaGerenteId2;
47
48   dbms_output.put_line(agenciaCodigo2);
49   dbms_output.put_line(agenciaNome2);
50   dbms_output.put_line(agenciaGerenteId2);
51 END;
52 /
53
54 DECLARE
55  agenciaCodigo3 AGENCIA.codigo%TYPE;
56  agenciaNome3 AGENCIA.nome%TYPE;
57  agenciaGerenteId3 AGENCIA.id_gerente%TYPE;
58 BEGIN
59   INSERT INTO AGENCIA (codigo, nome, id_gerente) VALUES (3646, 'Banco do Brasil', 1) RETURNING codigo, nome,
       id_gerente INTO agenciaCodigo3, agenciaNome3, agenciaGerenteId3;
60
61   dbms_output.put_line(agenciaCodigo3);
62   dbms_output.put_line(agenciaNome3);
63   dbms_output.put_line(agenciaGerenteId3);
64 END;
65 /
66 --Fim do teste da entidade AGENCIA
67
68 --Inicio do teste da entidade CORRENTISTA
69 DECLARE
70  correntistaCpf CORRENTISTA.cpf%TYPE;
71  correntistaNome CORRENTISTA.nome%TYPE;
72 BEGIN
73   INSERT INTO CORRENTISTA (cpf, nome) VALUES (null, 'Joseph') RETURNING cpf, nome INTO correntistaCpf,
       correntistaNome;
74
75   dbms_output.put_line(correntistaCpf);
76   dbms_output.put_line(correntistaNome);
77 END;
78 /
```

```
79
80  DECLARE
81   correntistaCpf2 CORRENTISTA.cpf%TYPE;
82   correntistaNome2 CORRENTISTA.nome%TYPE;
83  BEGIN
84    INSERT INTO CORRENTISTA (cpf, nome) VALUES (12345678910, 'John') RETURNING cpf, nome INTO correntistaCpf2,
          correntistaNome2;
85
86    dbms_output.put_line(correntistaCpf2);
87    dbms_output.put_line(correntistaNome2);
88  END;
89  /
90  --Fim do teste da entidade CORRENTISTA
91
92  --Inicio do teste da entidade CORRENTE
93  DECLARE
94   correnteTxManutencao CORRENTE.tx_manutencao%TYPE;
95   correnteContaNumero CORRENTE.numero_conta%TYPE;
96  BEGIN
97    INSERT INTO CORRENTE (tx_manutencao, numero_conta) VALUES (6, null) RETURNING tx_manutencao, numero_conta INTO
          correnteTxManutencao, correnteContaNumero;
98
99    dbms_output.put_line(correnteTxManutencao);
100   dbms_output.put_line(correnteContaNumero);
101 END;
102 /
103
104 DECLARE
105  correnteTxManutencao2 CORRENTE.tx_manutencao%TYPE;
106  correnteContaNumero2 CORRENTE.numero_conta%TYPE;
107 BEGIN
108   INSERT INTO CORRENTE (tx_manutencao, numero_conta) VALUES (5, 616279) RETURNING tx_manutencao, numero_conta INTO
          correnteTxManutencao2, correnteContaNumero2;
109
110   dbms_output.put_line(correnteTxManutencao2);
111   dbms_output.put_line(correnteContaNumero2);
112 END;
113 /
114 --Fim do teste da entidade CORRENTE
115
116 --Inicio do teste da entidade POUPANCA
117 DECLARE
118  poupancaDiaAniversario POUPANCA.dia_aniversario%TYPE;
119  poupancaContaNumero POUPANCA.numero_conta%TYPE;
120 BEGIN
121   INSERT INTO POUPANCA (dia_aniversario, numero_conta) VALUES (23, null) RETURNING dia_aniversario, numero_conta
          INTO poupancaDiaAniversario, poupancaContaNumero;
122
123   dbms_output.put_line(poupancaDiaAniversario);
124   dbms_output.put_line(poupancaContaNumero);
125 END;
126 /
127
128 DECLARE
129  poupancaDiaAniversario2 POUPANCA.dia_aniversario%TYPE;
130  poupancaContaNumero2 POUPANCA.numero_conta%TYPE;
131 BEGIN
132   INSERT INTO POUPANCA (dia_aniversario, numero_conta) VALUES (23, 616280) RETURNING dia_aniversario, numero_conta
          INTO poupancaDiaAniversario2, poupancaContaNumero2;
133
134   dbms_output.put_line(poupancaDiaAniversario2);
135   dbms_output.put_line(poupancaContaNumero2);
136 END;
137 /
138 --Fim do teste da entidade POUPANCA
139
140 --Inicio do teste da entidade CONTA
141 DECLARE
142  contaNumero CONTA.numero%TYPE;
143  contaSaldo CONTA.saldo%TYPE;
144  contaCodigoAgencia CONTA.codigo_agencia%TYPE;
145 BEGIN
146   INSERT INTO CONTA (numero, saldo, codigo_agencia) VALUES (null, 3000, 3646) RETURNING numero, saldo,
          codigo_agencia INTO contaNumero, contaSaldo, contaCodigoAgencia;
147
148   dbms_output.put_line(contaNumero);
149   dbms_output.put_line(contaSaldo);
150   dbms_output.put_line(contaCodigoAgencia);
151 END;
152 /
153
154 DECLARE
155  contaNumero2 CONTA.numero%TYPE;
156  contaSaldo2 CONTA.saldo%TYPE;
157  contaCodigoAgencia2 CONTA.codigo_agencia%TYPE;
158 BEGIN
159   INSERT INTO CONTA (numero, saldo, codigo_agencia) VALUES (616278, 3000, 3646) RETURNING numero, saldo,
          codigo_agencia INTO contaNumero2, contaSaldo2, contaCodigoAgencia2;
160
161   dbms_output.put_line(contaNumero2);
162   dbms_output.put_line(contaSaldo2);
163   dbms_output.put_line(contaCodigoAgencia2);
164 END;
```

```
/

DECLARE
 contaNumero3 CONTA.numero%TYPE;
 contaSaldo3 CONTA.saldo%TYPE;
 contaCodigoAgencia3 CONTA.codigo_agencia%TYPE;
BEGIN
  INSERT INTO CONTA (numero, saldo, codigo_agencia) VALUES (616279, 3000, 3646) RETURNING numero, saldo,
       codigo_agencia INTO contaNumero3, contaSaldo3, contaCodigoAgencia3;

  dbms_output.put_line(contaNumero3);
  dbms_output.put_line(contaSaldo3);
  dbms_output.put_line(contaCodigoAgencia3);
END;
/

DECLARE
 contaNumero4 CONTA.numero%TYPE;
 contaSaldo4 CONTA.saldo%TYPE;
 contaCodigoAgencia4 CONTA.codigo_agencia%TYPE;
BEGIN
  INSERT INTO CONTA (numero, saldo, codigo_agencia) VALUES (616280, 5000, 3646) RETURNING numero, saldo,
       codigo_agencia INTO contaNumero4, contaSaldo4, contaCodigoAgencia4;

  dbms_output.put_line(contaNumero4);
  dbms_output.put_line(contaSaldo4);
  dbms_output.put_line(contaCodigoAgencia4);
END;
/
--Fim do teste da entidade CONTA

--Inicio do teste da restricao de disjuncao
DECLARE
 poupancaDiaAniversario3 POUPANCA.dia_aniversario%TYPE;
 poupancaContaNumero3 POUPANCA.numero_conta%TYPE;
BEGIN
  INSERT INTO POUPANCA (dia_aniversario, numero_conta) VALUES (23, 616279) RETURNING dia_aniversario, numero_conta
       INTO poupancaDiaAniversario3, poupancaContaNumero3;

  dbms_output.put_line(poupancaDiaAniversario3);
  dbms_output.put_line(poupancaContaNumero3);
END;
/
--Fim do teste da restricao de disjuncao

--Inicio do teste da entidade CHEQUE
DECLARE
 chequeNumRegistro CHEQUE.num_registro%TYPE;
 chequeCcNumero CHEQUE.numero_conta%TYPE;
BEGIN
  INSERT INTO CHEQUE (num_registro, numero_conta) VALUES (null, 616279) RETURNING num_registro, numero_conta INTO
       chequeNumRegistro, chequeCcNumero;

  dbms_output.put_line(chequeNumRegistro);
  dbms_output.put_line(chequeCcNumero);
END;
/

DECLARE
 chequeNumRegistro2 CHEQUE.num_registro%TYPE;
 chequeCcNumero2 CHEQUE.numero_conta%TYPE;
BEGIN
  INSERT INTO CHEQUE (num_registro, numero_conta) VALUES (1, 616260) RETURNING num_registro, numero_conta INTO
       chequeNumRegistro2, chequeCcNumero2;

  dbms_output.put_line(chequeNumRegistro2);
  dbms_output.put_line(chequeCcNumero2);
END;
/

DECLARE
 chequeNumRegistro3 CHEQUE.num_registro%TYPE;
 chequeCcNumero3 CHEQUE.numero_conta%TYPE;
BEGIN
  INSERT INTO CHEQUE (num_registro, numero_conta) VALUES (1, 616279) RETURNING num_registro, numero_conta INTO
       chequeNumRegistro3, chequeCcNumero3;

  dbms_output.put_line(chequeNumRegistro3);
  dbms_output.put_line(chequeCcNumero3);
END;
/
--Fim do teste da entidade CHEQUE

--Inicio do teste da entidade relacionamento CARTAO
DECLARE
 cartaoSenha CARTAO.senha%TYPE;
 cartaoNumero CARTAO.numero%TYPE;
 cartaoCpfCorrentista CARTAO.cpf_correntista%TYPE;
 cartaoNumeroConta CARTAO.numero_conta%TYPE;
BEGIN
  INSERT INTO CARTAO (senha, numero, cpf_correntista, numero_conta) VALUES (1234, 1234567890123456, null, 616279)
       RETURNING senha, numero, cpf_correntista, numero_conta INTO cartaoSenha, cartaoNumero, cartaoCpfCorrentista,
       cartaoNumeroConta;
```

```
250
251   dbms_output.put_line(cartaoSenha);
252   dbms_output.put_line(cartaoNumero);
253   dbms_output.put_line(cartaoCpfCorrentista);
254   dbms_output.put_line(cartaoNumeroConta);
255 END;
256 /
257
258 DECLARE
259  cartaoSenha2 CARTAO.senha%TYPE;
260  cartaoNumero2 CARTAO.numero%TYPE;
261  cartaoCpfCorrentista2 CARTAO.cpf_correntista%TYPE;
262  cartaoNumeroConta2 CARTAO.numero_conta%TYPE;
263 BEGIN
264   INSERT INTO CARTAO (senha, numero, cpf_correntista, numero_conta) VALUES (1234, 1234567890123456, 12345678910,
        null) RETURNING senha, numero, cpf_correntista, numero_conta INTO cartaoSenha2, cartaoNumero2,
        cartaoCpfCorrentista2, cartaoNumeroConta2;
265
266   dbms_output.put_line(cartaoSenha2);
267   dbms_output.put_line(cartaoNumero2);
268   dbms_output.put_line(cartaoCpfCorrentista2);
269   dbms_output.put_line(cartaoNumeroConta2);
270 END;
271 /
272
273 DECLARE
274  cartaoSenha3 CARTAO.senha%TYPE;
275  cartaoNumero3 CARTAO.numero%TYPE;
276  cartaoCpfCorrentista3 CARTAO.cpf_correntista%TYPE;
277  cartaoNumeroConta3 CARTAO.numero_conta%TYPE;
278 BEGIN
279   INSERT INTO CARTAO (senha, numero, cpf_correntista, numero_conta) VALUES (1234, 1234567890123456, 12345678910,
        616279) RETURNING senha, numero, cpf_correntista, numero_conta INTO cartaoSenha3, cartaoNumero3,
        cartaoCpfCorrentista3, cartaoNumeroConta3;
280
281   dbms_output.put_line(cartaoSenha3);
282   dbms_output.put_line(cartaoNumero3);
283   dbms_output.put_line(cartaoCpfCorrentista3);
284   dbms_output.put_line(cartaoNumeroConta3);
285 END;
286 /
287 --Fim do teste da entidade relacionamento CARTAO
288
289 --Inicio do teste da atributo multivalorado MOVIMENTACOES
290 DECLARE
291  movimentacoesNConta CONTA_MOVIMENTACOES.numero_conta%TYPE;
292  movimentacoesId CONTA_MOVIMENTACOES.pk_movimentacoes%TYPE;
293  movimentacoesMovimentacoes CONTA_MOVIMENTACOES.movimentacoes%TYPE;
294 BEGIN
295   INSERT INTO MOVIMENTACOES (numero_conta, pk_movimentacoes, movimentacoes) VALUES (null, 1, 'Deposito 300.00')
        RETURNING numero_conta, pk_movimentacoes, movimentacoes INTO movimentacoesNConta, movimentacoesId,
        movimentacoesMovimentacoes;
296
297   dbms_output.put_line(movimentacoesNConta);
298   dbms_output.put_line(movimentacoesId);
299   dbms_output.put_line(movimentacoesMovimentacoes);
300 END;
301 /
302
303 DECLARE
304  movimentacoesNConta2 CONTA_MOVIMENTACOES.numero_conta%TYPE;
305  movimentacoesId2 CONTA_MOVIMENTACOES.pk_movimentacoes%TYPE;
306  movimentacoesMovimentacoes2 CONTA_MOVIMENTACOES.movimentacoes%TYPE;
307 BEGIN
308   INSERT INTO MOVIMENTACOES (numero_conta, pk_movimentacoes, movimentacoes) VALUES (616279, null, 'Deposito 300.00')
         RETURNING numero_conta, pk_movimentacoes, movimentacoes INTO movimentacoesNConta2, movimentacoesId2,
        movimentacoesMovimentacoes2;
309
310   dbms_output.put_line(movimentacoesNConta2);
311   dbms_output.put_line(movimentacoesId2);
312   dbms_output.put_line(movimentacoesMovimentacoes2);
313 END;
314 /
315
316 DECLARE
317  movimentacoesNConta3 CONTA_MOVIMENTACOES.numero_conta%TYPE;
318  movimentacoesId3 CONTA_MOVIMENTACOES.pk_movimentacoes%TYPE;
319  movimentacoesMovimentacoes3 CONTA_MOVIMENTACOES.movimentacoes%TYPE;
320 BEGIN
321   INSERT INTO MOVIMENTACOES (numero_conta, pk_movimentacoes, movimentacoes) VALUES (616279, 1, 'Deposito 300.00')
        RETURNING numero_conta, pk_movimentacoes, movimentacoes INTO movimentacoesNConta3, movimentacoesId3,
        movimentacoesMovimentacoes3;
322
323   dbms_output.put_line(movimentacoesNConta3);
324   dbms_output.put_line(movimentacoesId3);
325   dbms_output.put_line(movimentacoesMovimentacoes3);
326 END;
327 /
328 --Fim do teste da atributo multivalorado MOVIMENTACOES
```

Listagem 8. *Script* de Teste