O Modelo Relacional

Carla Geovana do N. Macário[1], Stefano Monteiro Baldo[2]

[1]Embrapa Informática Agropecuária – Av. André Tosello, 209 - Barão Geraldo
Caixa Postal 6041– 13083-886 – Campinas – SP – Brazil

[2]Instituto de Computação – Universidade Estadual de Campinas
Caixa Postal 6176 – 13084-971 – Campinas – SP – Brazil

carla@cnptia.embrapa.br, stefz@terra.com.br

Este trabalho foi produzido como parte da avaliação do curso MO-410 Introdução a Banco de Dados oferecido pelo Instituto de Computação da Unicamp em 2005/1.

Abstract. *The relational model has been the most widely used model in database area. It was proposed in 1970 and revolutionized the database field. Besides the new trends in software, like the object-oriented approach, the relational model is the dominant model in database market. This paper describes the relational model and its main features and advantages, including the SQL commands used to implement them. Also is presented rules to convert an Entity-Relationship diagram into a relational database schema, tailoring the user on the logical design of a database.*

Resumo. *O modelo relacional é atualmente o modelo mais utilizado em banco de dados. Proposto inicialmente em 1970, revolucionou o mercado desta área e, apesar das novas tendências de software como orientação a objetos, continua sendo o modelo dominante no mercado de banco de dados. Este trabalho descreve o modelo relacional, identificando suas principais características e vantagens e apresentando comandos em linguagem SQL para implementação destas características. Também apresenta regras para promover o mapeamento de um esquema no modelo Entidade Relacionamento para o modelo relacional, visando auxiliar o usuário na execução da fase de projeto lógico de um banco de dados.*

1. Introdução

O modelo relacional foi proposto por Edgar Codd em 1970, como uma nova maneira de representação de dados. Neste seu trabalho Codd mostrou que uma visão relacional dos dados permite a sua descrição em uma maneira natural, sem que sejam necessárias estruturas adicionais para sua representação, provendo uma maior independência dos dados em relação aos programas. Em complementação, apresentou bases para tratar problemas como redundância e consistência. Mais tarde, em outro trabalho, Codd definiu uma álgebra relacional e provou, por meio de sua equivalência com o cálculo relacional, que ela era relacionalmente completa, dando fundamentação teórica ao modelo relacional.

Este modelo, por suas características e por sua completitude, mostrou ser uma excelente opção, superando os modelos mais usados àquela época: o de redes e o

hierárquico. A maior vantagem do modelo relacional sobre seus antecessores é a representação simples dos dados e a facilidade com que consultas complexas podem ser expressas.

Em meados dos anos 70 foram desenvolvidos os primeiros sistemas relacionais em projetos da IBM, como o Sistema-R, e da Universidade de Berkeley (Califórnia), que deu origem ao sistema Ingres. Desde então, o uso deste modelo intensificou-se e a partir dos anos 80, o modelo relacional passou a ser dominante na área de banco de dados. Tal crescimento fez o mercado de sistemas relacionais crescer bastante, tornando-se hoje um mercado milionário.

Vários são os fornecedores atuais, dentre os quais podemos citar IBM, Microsoft, Sybase e Oracle, este último uma empresa que começou naquela época e que hoje é provavelmente a líder do mercado de banco de dados relacionais. Existem também sistemas como FireBird, MySQL e PostGres, que com a nova tendência de software livre, vêm sendo utilizados por várias empresas.

Ainda existem sistemas legados mantidos nos modelos antigos, como o IMS DBMS da IBM no modelo hierárquico e o IDS e IDMS no modelo de redes. Há também uma nova abordagem que vem sendo bastante explorada: o modelo orientado a objetos, que tem com exemplos o Objectstore e o Versant . Apesar da orientação a objetos não ser um conceito novo, apenas recentemente, com a proposição de novas metodologias, como OMT e UML, e de linguagens de programação como C++ e Java, seu uso vem se intensificado, tornando-se um padrão em várias aplicações de software. Com isso, a existência de banco de dados orientados a objetos está sendo colocada como uma das revoluções na área de banco de dados. Assim, numa tentativa de reunir vantagens de objetos com as inerentes ao modelo relacional, tem-se os bancos de dados objeto-relacional. Mesmo assim, o modelo relacional ainda é o modelo dominante em seu mercado.

A Linguagem SQL - Structured Query Language – que foi desenvolvida originalmente pela IBM para consulta ao seu Sistema-R, evoluiu, vindo a tornar-se a linguagem mais usada para criação, manipulação e consultas em Sistemas Gerenciadores de Banco de Dados (SGBD) relacionais.

Tal fato levou à definição de uma SQL padrão, permitindo aos seus usuários avaliar a completitude da linguagem oferecida pelos sistemas disponíveis e a identificar diferenças entre características específicas de produtos. Além disso, a padronização garante portabilidade, já que produtos que se baseiam em características padrão tendem a ser mais portáveis.

A primeira versão da SQL padrão foi desenvolvida em 1986 pela American National Standards Institute (ANSI) e recebeu o nome de SQL-86. Em 1989 foi lançada a SQL-89, contendo uma revisão pouco expressiva da linguagem, com a incorporação de restrições de integridade. Já em 1992, a ANSI , em conjunto com a International Standards Organizations (ISO) lançou uma revisão mais significativa, ainda utilizada por muitos dos sistemas disponíveis. Em 1999 foram adicionadas algumas outras extensões, como os gatilhos (*triggers*), gerando a SQL:1999. Apesar desta ser a linguagem padrão atual, já existe uma nova versão, a SQL:2003 também chamada de SQL:200n.

Este trabalho apresenta uma introdução ao modelo relacional, discutindo seus principais conceitos e características, como restrições de integridade, uma propriedade importante deste modelo. Em seguida passa-se à execução do projeto lógico de um banco de dados, mapeando um esquema gerado no projeto conceitual em Modelo Entidade Relacionamento (MER) para o modelo relacional. São dadas regras básicas a serem usadas nesta atividade, tornando-a mais simples. Por fim têm-se uma introdução ao conceito de visão, muito utilizado no modelo relacional, que traz consigo vantagens como independência de dados e segurança. Em todos os tópicos discutidos são utilizados exemplos para facilitar seu entendimento, complementados por comandos em SQL padrão.

2. O Modelo Relacional

No modelo relacional a principal construção para representação dos dados é a **relação**, uma tabela com linhas não ordenadas e colunas. Uma relação consiste de um **esquema** e de uma **instância**. O esquema especifica o nome da relação e o nome e o domínio de cada coluna, também denominada atributo ou campo da relação. O domínio do atributo é referenciado no esquema por seu nome e serve para restringir os valores que este atributo pode assumir. O esquema de uma relação é invárivael ao longo do tempo, sendo modificado apenas por comandos específicos. Um exemplo de esquema de relação é:

Students (*sid*: string, *name*: string, *login*: string, *age*: integer, *gpa*: real).

Neste caso está sendo definida a relação de nome *Students*, com atributos *sid*, *name*, *login*, *age* e *gpa*, cujos domínios são respectivamente *string*, *string*, *string*, *integer* e *real*.

A instância de uma relação é o conjunto de linhas, também denominadas tuplas ou registros, distintas entre si, que compõem a relação em um dado momento. Ela é variável, já que o número de tuplas e o conteúdo de seus atributos podem variar ao longo do tempo. A instância de uma relação deve seguir sempre o seu respectivo esquema, respeitando o número de atributos definidos, bem como os seus domínios. Esta restrição, denominada restrição de domínio, é muito importante. O modelo relacional somente considera relações que satisfaçam esta restrição. Um exemplo de uma instância para o esquema *Students* é ilustrado na Figura 1.

Sid	Name	Login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

Figura 1 – Exemplo de instância da relação Students

O número de tuplas que uma dada instância possui denomina-se cardinalidade da relação e o número de atributos é o seu grau. A instância de relação da Figura 1 tem cardinalidade 3 e grau 5. Note que a cardinalidade é variável, mas o grau não.

Um banco de dados relacional é um conjunto de uma ou mais relações com nomes distintos. O esquema do banco de dados relacional é a coleção dos esquemas de cada relação que compõe o banco de dados.

2.1. Criando e Modificando Relações em SQL

A linguagem SQL padrão usa a palavra TABLE para referenciar uma relação. Um subconjunto desta linguagem forma a **Linguagem de Definição de Dados** (DDL) que compreende comandos básicos para a criação, a remoção e a modificação de relações.

A criação de relações em SQL é feita usando-se o comando CREATE TABLE, com a especificação do respectivo esquema. Por exemplo, para criar a relação *Students* citada anteriormente tem-se:

```
CREATE TABLE Students (sid: CHAR(20), name: CHAR(20), login: CHAR(10),
age: INTEGER, gpa: REAL)
```

Observe que com a execução deste comando está sendo criada apenas a relação, sem que sejam atribuídos quaisquer valores aos seus atributos.

Para a remoção de uma relação do banco de dados usa-se o comando DROP TABLE. Assim, para remover a mesma relação *Students* tem-se:

```
DROP TABLE Students
```

Este comando remove a relação especificada, removendo a informação sobre o seu esquema e também as tuplas da instância atual.

O comando ALTER TABLE é usado para alteração do esquema de uma relação. Ainda considerando a relação *Students* para alterar o seu esquema com a adição de um novo campo *firstYear* cujo domínio é inteiro usa-se o comando:

```
ALTER TABLE Students ADD COLUMN firstYear: integer;
```

A execução deste comando faz com que o esquema da relação seja alterado e com que para cada tupla da instância corrente seja criado um novo atributo de nome *firstYear*, atribuindo a ele o valor *null*.

Um outro subconjunto da linguagem SQL forma a **Linguagem de Manipulação de Dados** (DDL), que compreende comandos básicos para a modificação e a recuperação de dados.

O comando INSERT INTO é usado para adicionar novas tuplas a uma relação. Por exemplo, para inserir uma tupla na relação *Students*, tem-se:

```
INSERT INTO Students (sid, name, login, age, gpa)
   VALUES (53688, 'Smith', 'smith@ee', 18, 3.2)
```

onde os valores descritos por VALUES, correspondem ao valor que cada atributo terá na nova tupla.

As tuplas de uma relação são removidas por meio do comando DELETE FROM. Pode-se remover todas as tuplas de uma relação ou apenas aquelas que satisfaçam uma dada condição. Para remover as tuplas de estudantes cujo nome (*name*) é Smith na relação *Students* executa-se o comando:

```
DELETE FROM Students S WHERE S.name = 'Smith'
```

A alteração do valor de atributos que compõem as tuplas é feita usando-se o comando UPDATE FROM. De forma semelhante ao comando DELETE FROM, pode-se modificar uma tupla específica ou várias delas por meio de um único comando. Por exemplo:

```
UPDATE FROM Students S
   SET S.age = S.age + 1, S.gpa = S.gpa – 1
   WHERE S.sid = 53688
```

Este comando altera apenas a tupla da relação *Students* cujo atributo *sid* tenha valor igual a 53688. Já o comando a seguir altera todas as tuplas cujo atributo *gpa* tenha valor maior ou igual a 3.2.

```
UPDATE FROM Students S SET S.gpa = S.gpa + 0.5 WHERE S.gpa >= 3.2
```

3. Restrições de Integridade sobre Relações

Um bom SGBD deve evitar a entrada de informação incorreta ou inconsistente em sua base de dados, garantindo, com isso, a qualidade da informação inserida. Uma restrição de integridade (RI) é uma condição especificada no esquema da base de dados para restringir a informação a ser armazenada. Ou seja, a RI é uma condição definida que deve ser verdadeira para qualquer instância da base de dados. Se uma instância da base de dados satisfaz todas as RIs especificadas, então ela é uma instância válida. Um bom SGBD garante as RIs, não permitindo a existência de instâncias inválidas.

As RI são especificadas e conferidas em 2 momentos diferentes:

- especificação da RI: se dá na definição do esquema da base de dados pelo usuário ou pelo administrador da base de dados (DBA);
- conferência das RIs: é feita pelo banco de dados toda vez que uma relação é modificada por uma aplicação sendo executada.

O modelo relacional permite a especificação de vários tipos de RIs. Um deles é a restrição de domínio citada anteriormente. Outros tipos serão vistos a seguir.

3.1. Restrições de Chaves

A restrição de chave serve para garantir que as tuplas de uma relação sejam únicas. Para isso, identifica um conjunto mínimo de atributos que devem ter valores diferentes em todas as tuplas de uma instância da relação. Este conjunto de atributos denomina-se **chave candidata** da relação e deve satisfazer os seguintes requisitos:

- não podem existir 2 tuplas diferentes com os mesmos valores para estes atributos, ou seja, a chave identifica unicamente qualquer tupla da relação válida;
- ao retirar-se qualquer atributo componente da chave, ela deixa de identificar unicamente as tuplas.

Se o segundo requisito for violado, então a chave candidata é uma **super-chave**. Por exemplo temos que *sid* é uma chave para a relação *Student*, pois identifica cada estudantes. Já o conjunto {*sid, gpa*} é uma super-chave da relação, pois ao retirar-se o atributo *gpa*, o atributo *sid* continua identificando unicamente as tuplas. O conjunto de todos os atributos de uma relação formam sempre uma super-chave desta relação.

Pela definição de relação, é sempre garantida a existência de uma chave. Entretanto, cada relação pode conter várias chaves candidatas. Cabe ao DBA escolher dentre elas aquela que será a **chave primária**, a ser usada pelo banco de dados em operações de otimização. A escolha desta chave é muito importante e deve ser feita

visando garantir a qualidade dos dados. Por exemplo, na relação *Students*, se *name* fosse escolhido como chave primária, não seria possível a existência de estudantes homônimos, o que talvez não refletisse corretamente os requisitos do sistema. A chave primária não pode assumir valor *null*.

Especificando Restrições de Chaves em SQL

A especificação de uma chave e de uma chave primária em SQL é feita respectivamente, pelos comandos UNIQUE e PRIMARY KEY.

Na execução do comando a seguir está sendo criada a relação *Enrolled*, cuja chave primária é composta pelos atributos *stdid* e *cid*

```
CREATE TABLE Enrolled (stdid CHAR(20), cid CHAR(20), grade CHAR(2),
    PRIMARY KEY (stdid,cid) )
```

Caso fosse desejado que esta mesma relação tivesse uma chave composta pelos atributos *cid* e *grade*, sendo *sid* a chave primária da relação, deveria ser executado o comando:

```
CREATE TABLE Enrolled (stdid CHAR(20), cid CHAR(20), grade CHAR(2),
    UNIQUE (cid, grade), CONSTRAINT EnrolledKey PRIMARY KEY (cid) )
```

O uso de CONSTRAINT no comando serve para nomear uma restrição, facilitando sua identificação para impressão de mensagens de erro numa eventual ocorrência de violação.

Note que neste segundo exemplo cada estudante pode cursar apenas um curso e receber uma única nota para este curso. E ainda que dois estudantes de um mesmo curso não recebem a mesma nota. Percebe-se, então, que quando usada de forma displicente, uma restrição de integridade pode impedir o armazenamento de instâncias de base de dados que surgem na prática.

3.2 Restrições de Chave Estrangeira

No modelo relacional é comum que a informação de uma relação esteja ligada à informação de outra relação. Se uma delas é modificada a outra também deve ser checada e modificada, se for o caso, de maneira a garantir a consistência da informação. Para que o banco de dados possa fazer esta checagem, é especificada uma restrição envolvendo ambas as relações. Esta restrição denomina-se restrição de **chave estrangeira**.

A chave estrangeira é um conjunto de atributos de uma relação que é usado para fazer referência a uma tupla de outra relação, correspondendo à chave primária da relação referenciada. A chave estrangeira deve conter o mesmo número de atributos da chave primária da outra relação, e seus respectivos domínios, mas não necessariamente os mesmos nomes. Além disso, diferente da chave primária, pode assumir valor *null*. Por fim pode referenciar a relação que a contém, se necessário.

Considerando a relação *Enrolled* apresentada anteriormente, queremos garantir que apenas estudantes com registro válido (*sid*) podem ser matriculados nos cursos. Para isso, cria-se uma chave estrangeira em *Enrolled* que corresponde à chave primária da relação *Students*, conforme ilustra a Figura 2:

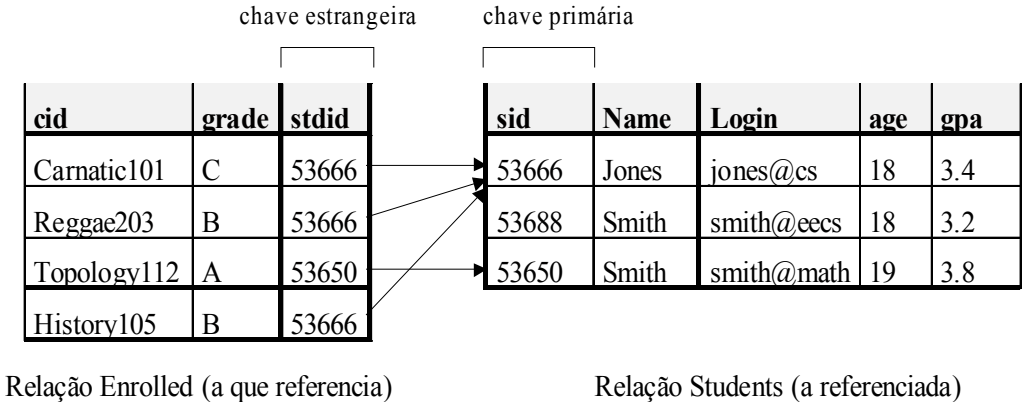

Figura 2 – Exemplo de chave estrangeira

Neste caso, se a aplicação tentar inserir em *Enrolled* uma tupla de um estudante de *stdid* 53673, o sistema de banco de dados irá rejeitar a operação, pois não existe um estudante em *Students* com este *sid*. Da mesma forma, ao tentar excluir de *Students* o estudante com *sid* 53650, o sistema não deve permitir a exclusão, pois ele está sendo referenciado por *Enrolled*. Uma outra ação possível para este caso seria a exclusão de todas as tuplas de *Enrolled* que façam referência a este estudante.

Se todas as restrições de chave estrangeiras definidas no banco de dados são garantidas, a sua **integridade referencial** é alcançada, ou seja, garante-se que não há referências pendentes.

Especificando Restrições de Chave Estrangeira em SQL

O comando FOREIGN KEY identifica a chave estrangeira na criação da relação. Assim, para definir esta restrição para o exemplo da Figura 2 usa-se o comando:

```
CREATE TABLE Enrolled (stdid CHAR(20), cid CHAR(20), grade CHAR(2),
                       PRIMARY KEY (stdid,cid),
FOREIGN KEY (stdid) REFERENCES Students )
```

Com este comando garante-se que somente estudantes registrados em *Students* possam ser matriculados em cursos. Outra restrição colocada por este comando, por meio da chave primária, é que cada estudante pode ter apenas uma nota por curso.

3.3. Restrições Gerais

Restrições de domínio, de chave primária e de chave estrangeiras são consideradas como parte fundamental do modelo de dados relacional. Entretanto, elas não são suficientes para especificar outras restrições mais genéricas, como, por exemplo, a definição de intervalos de valores para determinados atributos.

Para estes outros casos são usadas restrições de tabelas e de assertivas. As restrições de tabelas são associadas a um única tabela e são checadas sempre que a tabela é alterada. Já as restrições de assertivas são associadas a várias tabelas e são checadas sempre que uma destas tabelas é modificada.

4. Garantindo as Restrições de Integridade

Como já visto anteriormente, as RIs são especificadas quando uma relação é criada e são checadas sempre que uma relação é modificada. O impacto de RIs de domínio, de chave primária e de chave estrangeiras é direto. Ou seja, sempre que um comando de inserção, exclusão ou atualização causa uma violação de RI, ele é rejeitado.

Considerando as relações *Students* e *Enrolled* já definidas, sendo *stdid* uma chave estrangeira em *Enrolled* que faz referência a *Students*. O que deveria ser feito se uma tupla de *Enrolled* com um id de estudante não existente fosse inserida? Ou então uma tupla de *Students* com *sid* nulo (*null*)? O sistema deveria apenas rejeitá-las. Mas o que deveria ser feito se uma tupla de *Student* referenciada por *Enrolled* fosse removida? Neste caso com certeza o sistema estaria violando as restrições de integridade referencial. Para evitar isso, a linguagem SQL provê alternativas de tratamento para estas violações. São elas:

- rejeitar a remoção da tupla de *Students* que é referenciada por *Enrolled*;
- remover também todas as tuplas de *Enrolled* que referenciam a tupla de *Students* a ser removida;
- atribuir um valor padrão válido ao *stdid* das tuplas de *Enrolled* que referenciam a tupla de *Students* a ser removida;
- atribuir o valor null ao *stdid* das tuplas de *Enrolled* que referenciam a tupla de *Students* removida, denotando 'desconhecido' ou 'não aplicável'; entretanto, como neste exemplo *stdid* é parte da chave primária de *Enrolled* , esta alternativa estaria violando a RI de chave primária e portanto não poderia ser aplicada.

Estas opções são válidas também para o caso de atualizações na relação *Students*.

Os comandos em SQL para implementar estas ações são:

- NO ACTION, que é a opção padrão e que rejeita a operação sendo executada;
- CASCADE, que remove a tupla da relação referenciada e todas as tuplas que fazem referência à ela;
- SET NULL / SET DEFAULT , que atribui um valor à chave estrangeira da tupla referenciada.

A seleção da alternativa a ser utilizada é feita no momento da especificação da RI. Para o exemplo utilizado, supondo-se que no caso de exclusão seja escolhida a segunda alternativa e no caso de atualização a primeira delas, a criação da tabela *Enrolled* se daria pelo comando:

```
CREATE TABLE Enrolled (stdid CHAR(20), cid CHAR(20), grade CHAR(2),
                                        PRIMARY KEY (stdid,cid), FOREIGN
KEY (stdid) R  EFERENCES Students
            ON DELETE CASCADE        ON UPDATE NO ACTION )
```

4.1. Transações e Restrições

Uma transação é um programa que é executado pelo banco de dados e que pode conter vários comandos de acesso à base de dados, como consultas, inserções, atualizações, etc. Caso um destes comandos da transação viole algumas das restrições de integridade especificadas o tratamento padrão é rejeitar a sua execução. Entretanto, esta abordagem algumas vezes pode ser muito inflexível, devendo ser dado outro tratamento à situação.

Um exemplo seria, considerando as relações *Enrolled* e *Students*, a situação em que para o estudante ser registrado, ou seja, ter seu id, ele deve estar matriculado em um curso. Entretanto, para o curso existir, deve haver pelo menos um estudante matriculado. Percebe-se pelas restrições existentes neste caso, que ao tentar-se inserir a primeira tupla de qualquer das relações, ocorrerá violação e, portanto, as operações não serão completadas. A única maneira de conseguir realizar a primeira inserção em alguma delas seria postergando a checagem da restrição, que normalmente ocorreria ao final da execução do comando INSERT.

A linguagem SQL permite então que a checagem de uma restrição possa ser feita em modo imediato (IMMEDIATE) ou postergado (DEFERRED). Para isso usa-se o comando SET, indicando a restrição e o seu modo. Por exemplo o comando SET CONSTRAINT EnrolledKey DEFERRED, faz com que a restrição de nome *EnrolledKey* definida anteriormente seja checada somente no momento de efetivação (*commit*) da transação.

5. Consultando Dados Relacionais

Uma consulta em uma base de dados relacionais, normalmente referenciada como *query*, é uma questão sobre os dados da base, cuja resposta consiste em uma nova relação que contém o resultado na forma de tuplas. Por exemplo, usando as relações *Students* e *Enrolled*, pode-se querer saber quantos estudantes são maiores de 18 anos ou quantos estudantes estão matriculados em um determinado curso.

O comando em SQL usado para realização de consultas é o SELECT. Para encontrar todos os estudantes com 18 anos na instância de relação da Figura 1, executa-se o comando SELECT * FROM Students S WHERE S.age=18. A relação resposta é apresentada na Figura 3:

Sid	**Name**	**Login**	**age**	**gpa**
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2

Figura 3 – Relação resultante de uma consulta

O símbolo '*' na consulta indica que a relação resultante deve conter todos os atributos existentes na relação consultada. Caso fosse desejado obter apenas o nome e a senha destes estudantes, o comando a ser exeutado seria o descrito abaixo que produziria o resultado ilustrado na Figura 4.

SELECT *name, login* FROM Students S WHERE S.age=18

Name	**Login**

Jones	jones@cs
Smith	smith@eecs

Figura 4 – Relação resultante de uma consulta

6. Projeto Lógico da Base de Dados: indo do Modelo ER para o Modelo Relacional

O Modelo Entidade Relacionamento (MER) é adequado para representar um projeto de banco de dados em alto nível. Uma vez pronto o esquema da base de dados em MER, é preciso converter este esquema para o esquema de banco de dados a ser usado, geralmente no modelo relacional. Para isso, existe um conjunto de regras que direcionam o usuário nesta atividade, tratando, inclusive, restrições que não foram cobertas no esquema conceitual. Estas regras serão discutidas a seguir, abrangendo os principais elementos do MER.

6.1. Conjuntos Entidade

Um conjunto entidade (CE) é mapeado no modelo relacional com uma relação. Os atributos desta relação serão os mesmos do CE, bem como seus respectivos domínios. Da mesma forma, a chave primária também é mantida.

Como exemplo, considere o CE *Employees* ilustrado na Figura 5, cujos atributos são *ssn*, *name* e *lot*, sendo *ssn* a chave primária.

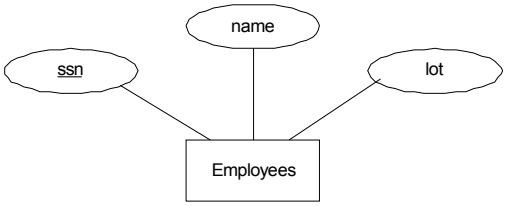

Figura 5 – CE Employees

O comando SQL para criar a relação correspondente é

CREATE TABLE Employees (ssn CHAR(11), name CHAR(20), lot INTEGER, PRIMARY KEY (ssn)).

6.2. Conjuntos Relacionamento

O mapeamento de conjuntos relacionamento (CR) para o modelo relacional pode ser feito de duas maneiras, abordadas a seguir.

Conjuntos Relacionamento sem Restrições

Um conjunto relacionamento (CR) sem restrições é mapeado numa relação de maneira semelhante ao conjunto entidade. Entretanto, a chave primária de cada entidade envolvida irá compor os atributos da nova relação como chaves estrangeiras, juntamente com os atributos descritivos. A chave primária desta relação será composta pelas chaves estrangeiras.

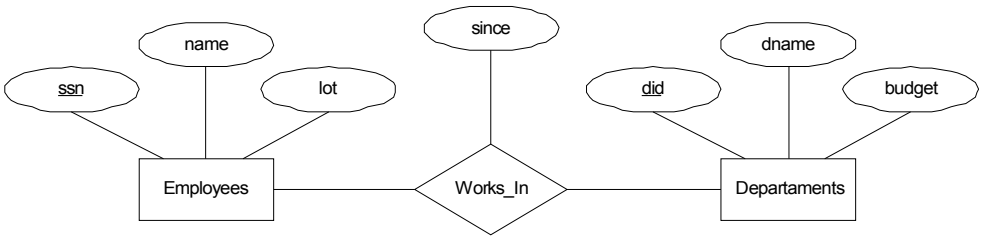

Figura 6 – CR Works_In

Para o exemplo do CR Works_In ilustrado na Figura 6, o seguinte comando SQL será usado no mapeamento:

```
CREATE TABLE Works_In( ssn CHAR(11), did INTEGER, since DATE,
        PRIMARY KEY (ssn, did),
        FOREIGN KEY (ssn) REFERENCES Employees,
        FOREIGN KEY (did) REFERENCES Departments)
```

No caso de auto-relacionamento a relação a ser criada conterá 2 ocorrências da chave primária da entidade envolvida, as quais comporão a chave primária da relação.

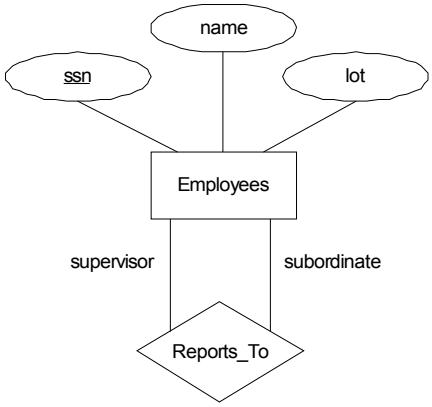

Figura 7 – CR Reports_To

Assim, o mapeamento do conjunto auto-relacionamento Reports_To ilustrado na Figura 7 para o modelo relacional é feito pelo comando:

```
CREATE TABLE Reports_To(supervisor_ssn CHAR(11), subordinate_ssn CHAR(11),
        PRIMARY KEY (supervisor_ssn, subordinate_ssn),
        FOREIGN KEY (supervisor_ssn) REFERENCES Employees (ssn),
        FOREIGN KEY (subordinate_ssn) REFERENCES Departments(ssn))
```

Conjuntos Relacionamento com Restrições de Chave

Existem 2 opções para o mapeamento de CR com restrição de chave para o modelo relacional. A primeira delas é semelhante àquela já apresentada, criando uma relação para representar o CR, com chaves estrangeiras para as relações envolvidas. Esta abordagem nem sempre é interessante, pois faz com que seja necessária a combinação de relações para responder às consultas do usuário, podendo se tornar uma atividade lenta.

A segunda opção é embutir o CR em qualquer das relações referentes às entidades envolvidas, usando uma chave estrangeira para referenciar a outra entidade. Os atributos descritivos também passam a integrar a relação que recebeu o CR. Esta solução é mais vantajosa por permitir a obtenção mais rápida de uma resposta. A única

ressalva é que podem ocorrer tuplas nas quais os atributos referentes ao CR estejam vazios.

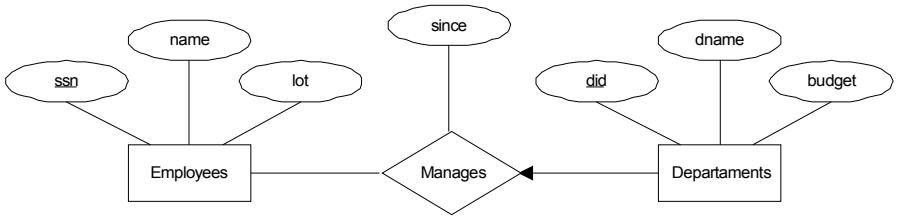

Figura 8 – CR Manages

Considerando a segunda opção, o mapeamento do CR Manages da Figura 8 para o modelo relacional é feito pelo comando:

```
CREATE TABLE Dept_Mgr(did INTEGER, dname CHAR(20), budget REAL,
        ssn CHAR(11), since DATE,
        PRIMARY KEY (did),
        FOREIGN KEY (ssn) REFERENCES Employees))
```

Esta abordagem pode ser aplicada a CR que envolvam mais de 2 CE. Em geral, se o CR envolve N CE e algumas delas tem restrição de chave, o CR pode ser embutido nesta CE.

Conjuntos Relacionamento com Restrições de Participação

A abordagem é a mesma que no caso anterior, mas com algumas particularidades. Considerando que no exemplo da Figura 8 existisse uma restrição de participação relacionada ao gerente: todo departamento tem de ter sempre um gerente. O comando a ser utilizado seria:

```
CREATE TABLE Dept_Mgr(did INTEGER, dname CHAR(20), budget REAL,
        ssn CHAR(11) NOT NULL, since DATE,
        PRIMARY KEY (did),
        FOREIGN KEY (ssn) REFERENCES Employees ON DELETE NO ACTION)
```

Neste caso a restrição de participação é garantida pelo NOT NULL usado no atributo *ssn* indicando que ele não pode assumir valores *null*. Ou seja, que ele tem sempre um valor associado. Já o comando NO ACTION, que é padrão, garante que o empregado não pode ser excluído da relação de empregados, se ele estiver como gerente do departamento.

6.3. Conjuntos de Entidades Fracas

O conjunto de entidades fracas tem sempre participação binária do tipo 1:N, bem como uma restrição de chave e uma de participação total. Levando-se isso em conta, a abordagem utilizada nos casos anteriores é ideal. Entretanto, por se tratar de uma entidade dependente de outra, tem chave do tipo parcial.

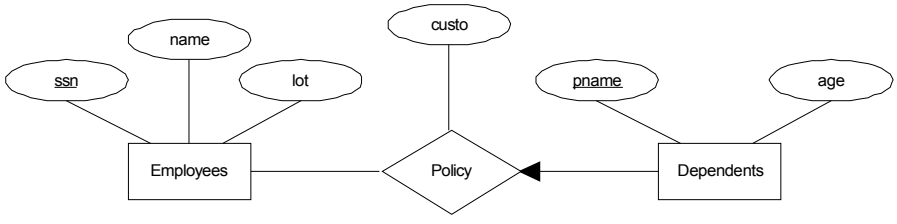

Figura 9 – Conjunto Entidade Fraca

Para o exemplo de conjunto entidade fraca ilustrado na Figura 9 o mapeamento para o modelo relacional seria feito por meio do comando:

```
CREATE TABLE Dept_Policy(pname CHAR(20), age INTEGER, cost REAL,
        ssn CHAR(11),
        PRIMARY KEY (pname, ssn),
        FOREIGN KEY (ssn) REFERENCES Employees ON DELETE CASCADE)
```

Note que neste caso, a chave da relação passou a ser composta pela chave primária da entidade fraca (*pname*) e pela chave primária da relação que a contém (*ssn*, em *Employees*), sendo esta última também uma chave estrangeira. O comando CASCADE, garante que se o empregado for excluído da relação *Employees*, todos os seus dependentes também o serão.

6.4.Traduzindo Hierarquias de Classe

Semelhante ao que ocorre com os CR, existem 2 abordagens para tratar as hierarquias ISA. Na primeira delas, deve ser criada, além da relação referente à super entidade, uma relação para cada especialização do CE, mantendo seus atributos e acrescentando uma chave estrangeira que referencia a super entidade. A chave estrangeira desta relação é também a sua chave primária. Caso a relação correspondente à super entidade seja removida, as relações das entidades especializadas também devem ser removidas, usando para isso o comando CASCADE.

A segunda abordagem sugere criar uma relação para cada especialização do CE, mas não para a super entidade. Neste caso cada relação criada conterá, além dos seus atributos, os atributos da super entidade.

A primeira abordagem é mais genérica e também a mais usada. Sua principal vantagem é permitir que a relação referente à super classe seja utilizada independente das relações referentes às entidades especializadas. Já a segunda abordagem obriga que a informação seja sempre obtida a partir de uma das entidades especializadas, não permitindo a existência da super classe.

6.5. Traduzindo diagramas ER com Agregação

Neste casos o mapeamento é simples e semelhante àqueles discutidos anteriormente: o CR da agregação é traduzido em uma nova relação, sendo sua chave primária é composta pela chave primária dos CE envolvidos. Da mesma forma, o CR que envolve esta agregação também é traduzido em uma nova relação, que terá dentre seus atributos a chave primária da relação que representa a agregação.

7. Visões

A visão é uma tabela cujas linhas não são explicitamente armazenadas na base de dados, mas sim computadas, quando necessário, a partir de uma definição em termos de tabelas da base de dados, denominada tabelas base. O comando em SQL usado para isso é o CREATE VIEW.

Considere as relações *Students* e *Enrolled* discutidas anteriormente. Suponha que exista um interesse constante em recuperar o nome e o identificador dos estudantes que tem nota B em algum curso e também o identificador deste curso. Uma visão pode

ser utilizada neste caso, sem que seja necessária a criação de uma tabela para isso. Então para o exemplo descrito poderia ser usado o comando:

```
CREATE VIEW B-Students (name, sid, course) AS SELECT S.sname, S.sid, E.cid
FROM Students S, Enrolled E WHERE S.sid = E.sid AND E.grade = 'B'
```

A visão definida neste caso é composta por 3 campos (*name*, *sid* e *course*) cujos domínios correspondem aos mesmos das relações *Students* (*name* e *sid*) e *Enrolled* (*course*).

7.1. Visões, Independência de Dados e Segurança

O mecanismo de visões provê uma independência lógica dos dados no modelo relacional. Ela pode ser usada para definir relações em um esquema externo que mascare mudanças ocorridas no esquema conceitual da base de dados das aplicações. Por exemplo, se o esquema de uma dada relação for alterado, pode ser definida uma visão que represente o esquema antigo, permitindo que aplicações continuem executando normalmente sem grandes modificações.

Visões também são muito úteis no contexto de segurança. Com elas é possível permitir que determinados grupos de usuários acessem somente os dados que eles tem permissão para uso. Por exemplo, permitir que estudantes vejam apenas o nome e login de outros estudantes, mas não sua senha ou sua nota em um curso.

7.2. Atualizações em Visões

Uma das vantagens do uso de visões é permitir o gerenciamento da apresentação do dados aos usuários, sem que eles tenham de se preocupar com a maneira como eles estão fisicamente armazenados na base de dados. Isto normalmente funciona adequadamente, já que a visão pode ser usada exatamente como uma relação, permitindo a definição de consultas sobre os dados que a compõem. Esta vantagem acaba levando a uma restrição, pois como se também se trata de uma relação, é natural o desejo de atualização os seus dados. Entretanto, como a visão é uma tabela virtual, a sua atualização deve incidir sobre a tabela base, o que nem sempre é possível, devido a problemas como ambigüidade e restrições de integridade.

Necessidade de Restringir Atualizações de Visões

É possível atualizar ou inserir uma linha em uma visão. No entanto, nem sempre esta atualização fará parte dela. Isto ocorre porque na verdade a execução destes comandos promovem uma atualização ou inserção nas tabelas base e dependendo da condição estabelecida na definição da visão, a linha pode não ser selecionada para compor a tabela virtual, não refletindo a alteração feita.

Assim, por questões práticas, a linguagem SQL permite a atualização de visões apenas em casos muito específicos, não permitindo, por exemplo, a atualização de colunas calculadas, de visões compostas por agregados de linhas ou geradas a partir de junções.

7.3. Destruindo e Alterando Visões

Uma visão, de maneira semelhante ao que ocorre com as relações, também pode ser removida do banco de dados. O comando em SQL utilizado para isso é o DROP VIEW.

Note que a remoção de uma relação também pode levar à remoção de uma visão caso tenha sido usado algum comando que estabeleça esta restrição, como o CASCADE. Já o comando RESTRICT evita que uma relação seja excluída quando da existência de uma visão associada a ela.

Referências Utilizadas

Ramarkrishnan, R. and Gehrke, J. (2003) “Data Management Systems”, McGraw-Hill, 3rd edition.

Guimarães, C. C. (2003) “Fundamentos de banco de dados: modelagem, projeto e linguagem SQL”, Editora da Unicamp.