Departamento de Ciência da Computação

Universidade Federal de Juiz de Fora

DCC060 – Banco de Dados

MATERIAL DE APOIO

Mapeamento do Modelo de Dados Conceitual para o Modelo Relacional e SQL

PROF. TARCÍSIO DE SOUZA LIMA

- Existe uma evolução natural dos modelos de dados ER e UML para um esquema relacional. Esta evolução é tão natural que se admite a modelagem de dados conceitual como etapa inicial efetiva do desenvolvimento de um BD relacional.

- Ferramentas genéricas de projeto de software dão suporte à conversão automática desses modelos de dados conceituais para definições de tabelas SQL e restrições de integridade específicas do fornecedor.

- Vamos considerar aqui que as aplicações são transacionais, ou seja, OLTP (*Online Transaction Processing*) e não OLAP (*Online Analytical Processing*)

- As transformações básicas podem ser descritas em termos dos 3 tipos de tabelas que elas produzem:
  - Transformação 1: tabela SQL com o mesmo conteúdo de informação da entidade original da qual é derivada;
  - Transformação 2: tabela SQL com a inclusão da chave estrangeira da entidade pai;
  - Transformação 3: tabela SQL derivada de um relacionamento, contendo as chaves estrangeiras de todas as entidades no relacionamento.

- **Transformação 1:** tabela SQL com o mesmo conteúdo de informação da entidade original da qual é derivada.

  Sempre ocorre para:

  - entidades com relacionamentos binários (associações) N:N, 1:N no lado "um" (pai) ou 1:1 em qualquer lado;
  - entidades com relacionamentos recursivos binários que são N:N; e
  - entidades com qualquer relacionamento ternário ou de maior grau ou hierarquia de generalização.

- **Transformação 2:** tabela SQL com a inclusão da chave estrangeira da entidade pai.

  Sempre ocorre para:

  - entidades com relacionamentos binários 1:N para a entidade no lado “muitos” (filho),
  - relacionamentos 1:1 para uma das entidades e
  - cada entidade com um relacionamento recursivo binário que seja 1:1 ou 1:N.

  OBS.: esta é uma das duas maneiras mais comuns de como as ferramentas de projeto tratam os relacionamentos, pedindo ao usuário para definir uma chave estrangeira na tabela filha que corresponda a uma chave primária na tabela pai.

- **Transformação 3:** tabela SQL derivada de um relacionamento, contendo as chaves estrangeiras de todas as entidades no relacionamento.

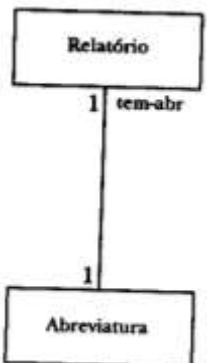
  Sempre ocorre para:

  - relacionamentos que são binários N:N, recursivos ou não e
  - relacionamentos que são de grau ternário ou maior.

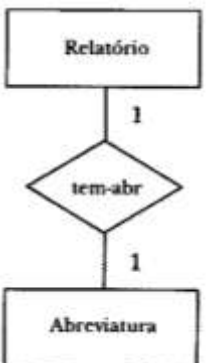
  OBS.: Um relacionamento N:N só pode ser definido em termos de uma tabela que contém chaves estrangeiras correspondentes às chaves primárias das duas entidades associadas. Essa nova tabela também pode conter os atributos do relacionamento original. P.ex., um relacionamento “matriculado_em” entre as entidades “Aluno” e “Curso” pode ter os atributos “semestre” e “nota”, que estão associados a uma inscrição em particular de um aluno em um determinado curso.

- Tratamento de valores nulos da SQL nestas transformações:
  - Nulos são permitidos em uma tabela SQL para chaves estrangeiras de entidades opcionais (referenciadas) associadas.
  - Nulos não são permitidos em uma tabela SQL para chaves estrangeiras de entidades obrigatórias (referenciadas) associadas.
  - Nulos não são permitidos para qualquer chave em uma tabela SQL derivada de um relacionamento N:N, pois somente entradas de linhas completas fazem sentido na tabela.

- **Relacionamento binário 1:1, ambas entidades obrigatórias – cada entidade vira uma tabela e a chave de qualquer entidade pode aparecer na tabela da outra entidade como uma chave estrangeira.**



UML

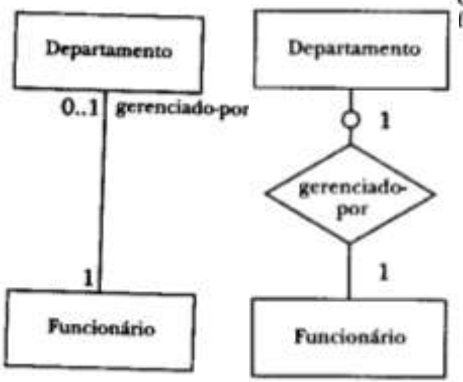


MER

Cada relatório tem uma abreviatura, e cada abreviatura representa exatamente um relatório.

```
create table relatório
        (num_relat integer,
        nome_relat varchar(256),
        primary key(num_relat));

ou
create table abreviatura
        (num_abrev char(6),
        num_relat integer not null unique,
        primary key (num_abrev),
        foreign key (num_relat) references relatório
            on delete cascade on update cascade);
```

- **Relacionamento binário 1:1, uma das entidades em um relacionamento opcional – Departamento deve conter a chave estrangeira da outra entidade em sua tabela transformada.**



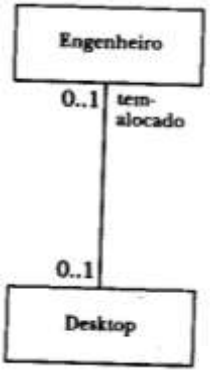
UML MER

Cada departamento precisa ter um gerente, mas um funcionário pode ser um gerente de no máximo um departamento.

```
create table departmento
        (num_depto integer,
        nome_depto char(20),
        id_gerente char(10) not null unique,
        primary key (num_depto),
        foreign key (id_gerente) references funcionário
          on delete set default on update cascade);

create table funcionário
        (id_func char(10),
        nome_func char(20),
        primary key (id_func));
```

OBS.: **Funcionário** também pode conter uma chave estrangeira (num_depto) com nulos permitidos, mas isto exigiria mais espaço de armazenamento devido ao maior número de instâncias.

- **Relacionamento binário 1:1, ambas as entidades opcionais – qualquer entidade pode conter a chave estrangeira embutida da outra entidade, com nulos permitidos nas chaves estrangeiras.**



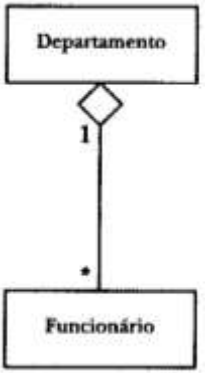
UML

MER

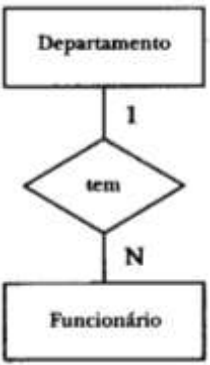
Alguns computadores de desktop são alocados a engenheiros, mas não necessariamente a todos os engenheiros.

```
create table engenheiro
        (id_func char(10),
        num_desktop integer,
        primary key (id_func));

ou

create table desktop
        (num_desktop integer,
        id_func char(10),
        primary key (num_desktop),
        foreign key (id_func) references engenheiro
          on delete set null on update cascade);
```

- **Relacionamento binário 1:N, com obrigatório ou opcional no lado "muitos" ou no lado "um" – em todos os casos a chave estrangeira precisa aparecer no lado "muitos", que representa a entidade filha, com nulos permitidos para chaves estrangeiras apenas nos casos em que o lado "um" é opcional.**



UML



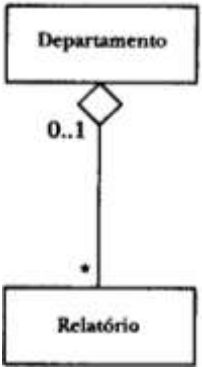
MER

Cada funcionário trabalha em exatamente um departamento, e cada departamento tem pelo menos um funcionário.

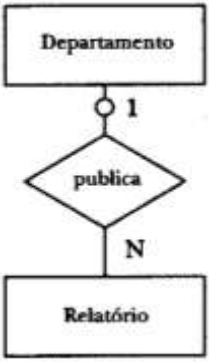
```
create table departamento
        (num_depto integer,
        nome_depto char(20),
        primary key (num_depto));

create table funcionário
        (id_func char(10),
        nome_func char(20),
        num_depto integer not null,
        primary key (id_func),
        foreign key (num_depto) references departamento
          on delete set default on update cascade);
```

Nulos não permitidos

- **Relacionamento binário 1:N, uma entidade opcional, uma obrigatória.**



UML



MER

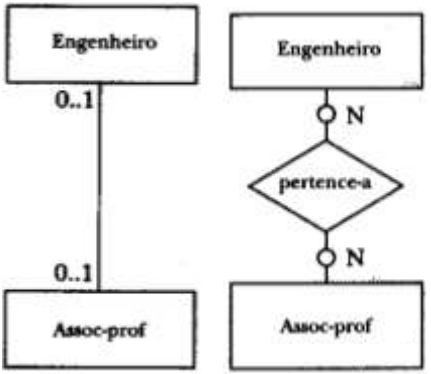
Cada departamento publica um ou mais relatórios. Determinado relatório pode não necessariamente ser publicado por um departamento.

```
create table departamento
        (num_depto integer,
        nome_depto char(20),
        primary key (num_depto));

create table relatório
        (num_relat integer,
        num_depto integer,
        primary key (num_relat),
        foreign key (num_depto) references departamento
          on delete set null on update cascade);
```

Neste caso os nulos são permitidos

OBS.: As restrições de chave estrangeira são definidas de acordo com o significado específico do relacionamento e podem variar de um relacionamento para o outro.

- **Relacionamento binário N:N, ambas as entidades opcionais ou obrigatórias – exige uma nova tabela contendo as chaves primárias das duas entidades. A cláusula *not null* precisa aparecer para as chaves estrangeiras nos dois casos.**

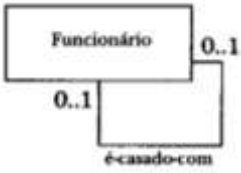


UML MER

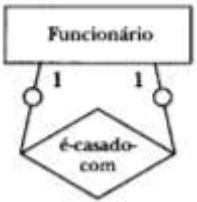
Cada associação profissional pode ter nenhum, um ou muitos membros engenheiros. Cada engenheiro poderia ser membro de nenhuma, uma ou muitas associações profissionais.

```
create table engenheiro
        (id_func char(10),
        primary key (id_func));

create table assoc-prof
        (nome_assoc varchar(256),
        primary key (nome_assoc));

create table pertence_a
        (id_func char(10),
        nome_assoc varchar(256),
        primary key (id_func, nome_assoc),
        foreign key (id_func) references engenheiro
          on delete cascade on update cascade,
        foreign key (nome_assoc) references assoc-prof
          on delete cascade on update cascade);
```

- **Relacionamento 1:1 com uma única entidade indica alguma forma de acoplamento (opcional ou obrigatório) entre ocorrências dessa entidade, conforme indicado pelo nome do relacionamento – em todos os casos a chave da entidade acoplada aparece como chave estrangeira na tabela resultante. Os dois atributos de chave são retirados do mesmo domínio, mas recebem diferentes nomes para indicar o seu uso exclusivo.**



UML

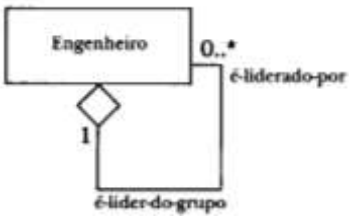


MER

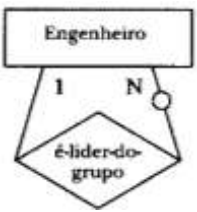
Um funcionário pode ser casado com outro funcionário nesta empresa.

```
create table funcionário
    (id_func char(10),
    nome_func char(20),
    id_cônjuge char(10),
    primary key (id_func),
    foreign key (id_cônjuge) references funcionário
      on delete set null on update cascade);
```

- **Relacionamento 1:N – exige uma chave estrangeira na tabela resultante. As restrições de chave estrangeira podem variar de acordo com a especificidade do relacionamento.**



UML

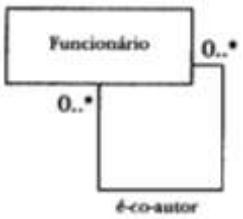


MER

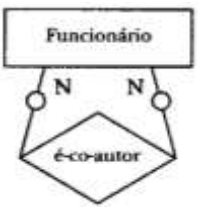
Os engenheiros são divididos em grupos para certos projetos. Cada grupo possui um líder.

```
create table engenheiro
        (id_func char(10),
        id_líder char(10) not null,
        primary key (id_func),
        foreign key (id_líder) references engenheiro
            on delete set default on update cascade);
```

- **Relacionamento N:N (opcional ou obrigatório) – resulta em uma nova tabela; nos dois casos, as chaves estrangeiras são definidas como "not null".**



UML



MER

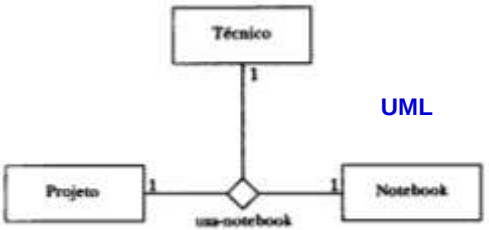
Cada funcionário tem a oportunidade de ser co-autor de um relatório com um ou mais outros funcionários, ou de escrever o relatório sozinho.

```
create table funcionário
        (id_func char(10),
        nome_func char(20),
        primary key (id_func));

create table co-autor
        (id_autor char(10),
        id_coautor char(10),
        primary key (id_autor, id_coautor),
        foreign key (id_autor) references funcionário
          on delete cascade on update cascade,
        foreign key (id_coautor) reference funcionário
          on delete cascade on update cascade);
```

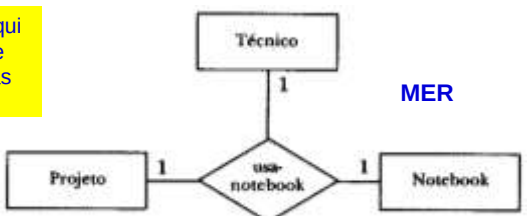
OBS.: As restrições de chave estrangeira sobre delete e update sempre precisam ser propagadas, pois cada entrada na tabela SQL depende do valor atual ou da existência da chave primária referenciada.

- **Relacionamento ternário, do tipo 1:1:1 – a tabela SQL resultante consiste em 3 chaves distintas possíveis. Esse arranjo representa o fato de que 3 DFs são necessárias para descrever esse relacionamento.**

Um técnico usa exatamente um notebook para cada projeto. Cada notebook pertence a um técnico para cada projeto. Observe que um técnico ainda pode trabalhar em muitos projetos e manter diferentes notebooks para diferentes projetos.



OBS.: A restrição de opcionalidade não é usada aqui porque todas as *n* entidades precisam participar de cada instância do relacionamento para satisfazer as restrições da DF.



- **Relacionamento ternário, do tipo 1:1:1**

```
create table técnico (id_func char(10),
                      primary key (id_func));

create table projeto (nome_projeto char(20),
                      primary key (nome_projeto));

create table notebook (num_notebook integer,
                       primary key (num_notebook));

create table usa_notebook (id_func char(10),
                           nome_projeto char(20),
                           num_notebook integer not null,
                           primary key (id_func, nome_projeto),
                           foreign key (id_func) references técnico
                              on delete cascade on update cascade,
                           foreign key (nome_projeto) references projeto
                              on delete cascade on update cascade,
                           foreign key (num_notebook) references notebook
                              on delete cascade on update cascade,
                           unique (id_func, num_notebook),
                           unique (nome_projeto, num_notebook));
```

- **Relacionamento ternário, do tipo 1:1:1**

usa_notebook

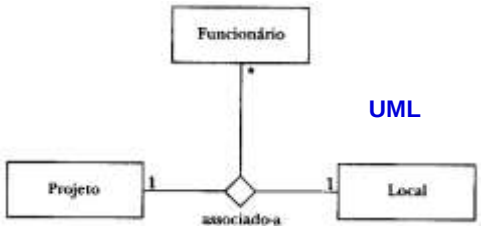
id-func	nome-projeto	num-notebook
35	alpha	5001
35	gamma	2008
42	delta	1004
42	epsilon	3005
81	gamma	1007
93	alpha	1009
93	beta	5001

Dependências funcionais

id_func, nome_projeto → num_notebook
id_func, num_notebook → nome_projeto
nome_projeto, num_notebook → id_func

- **Relacionamento ternário, do tipo 1:1:N – em geral, o número de entidades com conectividade "um" determina o limite inferior sobre o número de DFs; neste caso, existem 2 DFs.**

Cada funcionário associado para um projeto trabalha em apenas um local para esse projeto, mas pode estar em um local diferente para um projeto diferente. Em determinado local, um funcionário trabalha em apenas um projeto. Em um local em particular, pode haver muitos funcionários associados a um determinado projeto.



- **Relacionamento ternário, do tipo 1:1:N**

```
create table funcionário (id_func char(10),
                          nome_func char(20),
                          primary key (id_func));

create table projeto (nome_projeto char(20),
                      primary key (nome_projeto));

create table local (nome_loc char(15),
                    primary key (nome_loc));

create table associado_a (id_func char(10),
                          nome_projeto char(20),
                          nome_loc char(15) not null,
                          primary key (id_func, nome_projeto),
                          foreign key (id_func) references funcionário
                            on delete cascade on update cascade,
                          foreign key (nome_projeto) references projeto
                            on delete cascade on update cascade,
                          foreign key (nome_loc) references local
                            on delete cascade on update cascade,
                          unique (id_func, nome_loc));
```

- **Relacionamento ternário, do tipo 1:1:N**

associado_a

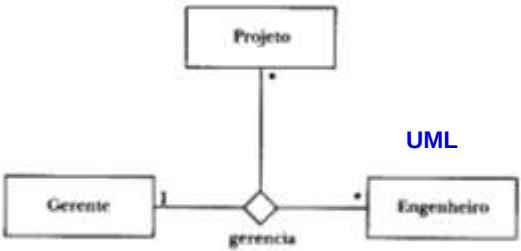
id-func	nome-projeto	nome-loc
48101	floresta	B66
48101	oceano	E71
20702	oceano	A12
20702	rio	D54
51266	rio	G14
51266	oceano	A12
76323	montanhas	B66

Dependências funcionais

id_func, nome_loc → nome_projeto
id_func, nome_projeto → nome_loc

- **Relacionamento ternário, do tipo 1:N:N – existe apenas uma DF.**

Cada engenheiro trabalhando em um determinado projeto tem exatamente um gerente, mas um projeto pode ter muitos gerentes, e um engenheiro pode ter muitos gerentes e muitos projetos. Um gerente pode gerenciar vários projetos.



- **Relacionamento ternário, do tipo 1:N:N**

```
create table projeto (nome_projeto char(20),
                      primary key (nome_projeto));

create table gerente (id_gerente char(10),
                      primary key (id_gerente));

create table engenheiro (id_func char(10),
                         primary key (id_func));

create table gerencia (nome_projeto char(20),
                       id_gerente char(10) not null,
                       id_func char(10),
                       primary key (nome_projeto, id_func),
                       foreign key (nome_projeto) references projeto
                         on delete cascade on update cascade,
                       foreign key (id_gerente) references gerente
                         on delete cascade on update cascade,
                       foreign key (id_func) references engenheiro
                         on delete cascade on update cascade);
```

- **Relacionamento ternário, do tipo 1:N:N**

gerencia

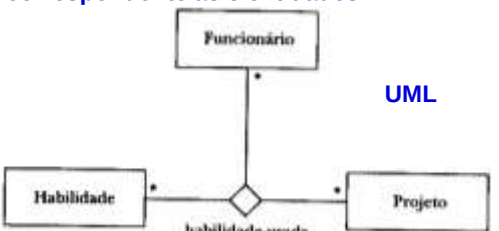
nome-projeto	id-func	id-gerente
alpha	4106	27
alpha	4200	27
beta	7033	32
beta	4200	14
gamma	4106	71
delta	7033	55
delta	4106	39
iota	4106	27

Dependências funcionais

nome_projeto, id_func → id_gerente

- **Relacionamento ternário, do tipo N:N:N – a tabela de relacionamentos é uma única chave composta, sem considerar os atributos próprios do relacionamento; neste caso, a chave é o composto de todas as 3 chaves correspondente às 3 entidades associadas.**

Os funcionários podem usar diferentes habilidades em qualquer um dos vários projetos, e cada projeto possui muitos funcionários com diversas habilidades.



- **Relacionamento ternário, do tipo N:N:N**

```
create table funcionário (id_func char(10),
                          nome_func char(20),
                          primary key (id_func));

create table habilidade (tipo_habilidade char(15),
                         primary key (tipo_habilidade));

create table projeto (nome_projeto char(20),
                      primary key (nome_projeto));

create table habilidade-usada (id_func char(10),
                               tipo_habilidade char(15),
                               nome_projeto char(20),
                               primary key (id_func, tipo_habilidade, nome_projeto),
                               foreign key (id_func) references funcionário
                                 on delete cascade on update cascade,
                               foreign key (tipo_habilidade) references habilidade
                                 on delete cascade on update cascade,
                               foreign key (nome_projeto) references projeto
                                 on delete cascade on update cascade);
```

- **Relacionamento ternário, do tipo N:N:N**

habilidade-usada

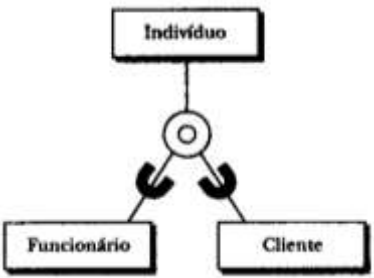
id-func	tipo_habilidade	nome-projeto
101	álgebra	eletrônica
101	cálculo	eletrônica
101	álgebra	mecânica
101	geometria	mecânica
102	álgebra	eletrônica
102	teoria dos conjuntos	eletrônica
102	geometria	mecânica
105	topologia	mecânica

Dependências funcionais

Nenhuma

OBS.: As restrições de chave estrangeira sobre delete e update para relacionamentos ternários transformados em tabelas SQL sempre precisam ser propagadas, pois cada entrada na tabela SQL depende do valor atual (ou da existência) da chave primária referenciada.

- **Abstração de generalização – 1ª opção: produzir tabelas SQL separadas para a entidade genérica ou supertipo e para cada um dos subtipos.**



MER

UML

- **Abstração de generalização – 1ª opção: produzir tabelas SQL separadas para a entidade genérica ou supertipo e para cada um dos subtipos.**

```
create table individuo (id_indiv char(10),
                        nome_indiv char(20),
                        end_indiv char(20),
                        primary key (id_indiv));
create table funcionário (id_func char(10),
                          nome_cargo char(15),
                          primary key (id_func),
                          foreign key (id_func) references individuo
                            on delete cascade on update cascade);
create table cliente (num_cliente char(10),
                      cred_cliente char(12),
                      primary key (num_cliente),
                      foreign key (num_cliente) references individuo
                        on delete cascade on update cascade);
```

A tabela derivada de uma entidade supertipo contém a chave da entidade supertipo e todos os atributos comuns. Cada tabela derivada das entidades subtipo contém a chave da entidade supertipo e apenas os atributos que são específicos e esse subtipo.

- **Abstração de generalização – 1ª opção: produzir tabelas SQL separadas para a entidade genérica ou supertipo e para cada um dos subtipos.**

OBSERVAÇÃO 1: A integridade de atualização é mantida exigindo-se que todas as inserções e exclusões ocorram na tabela do supertipo e na tabela do subtipo relevante – ou seja, a propagação da restrição de chave estrangeira precisa ser usada.

OBSERVAÇÃO 2: Se a atualização é na chave primária da tabela supertipo, então todas as tabelas subtipo, além da tabela supertipo, precisam ser atualizadas. Uma atualização em um atributo não-chave afeta o supertipo ou uma tabela subtipo, mas não ambos.

OBSERVAÇÃO 3: As regras de transformação (e regras de integridade) são iguais para as generalizações de subtipo disjuntas e sobrepostas.

- **Abstração de generalização – 2ª opção: ter uma única tabela que inclui todos os atributos do supertipo e dos subtipos (a hierarquia inteira em uma tabela), com os nulos sendo usados quando for necessário.**
- **Abstração de generalização – 3ª opção: uma tabela para cada subtipo, levando-se os atributos comuns para os subtipos específicos.**
- **Existem vantagens e desvantagens em cada uma dessas abordagens e ferramentas de software que admitem todas as três opções.**
- **Os profissionais de BD normalmente acrescentam um discriminador ao supertipo – atributo que possui um valor separado para cada supertipo e indica qual subtipo usar para obter mais informações.**
- **A abstração de agregação também produz uma tabela separada para a entidade supertipo e cada entidade subtipo. Contudo não existem atributos comuns e restrições de integridade a serem mantidas.**

- **Múltiplos relacionamentos entre *n* entidades sempre são considerados completamente independentes.**
- **Relacionamentos binários 1:1, 1:N ou recursivos que resultem em tabelas equivalentes ou que difiram apenas no acréscimo de uma chave estrangeira podem simplesmente ser unificados em uma única tabela contendo todas as chaves estrangeiras.**
- **Relacionamentos N:N ou ternários que resultam em tabelas SQL costumam ser exclusivos e não podem ser unificados.**

- **Entidades fracas diferem das entidades apenas em sua necessidade de chaves de outras entidades para estabelecer sua unicidade. Fora isto possuem as mesmas propriedades de transformação que as entidades e nenhuma regra especial é necessária.**

- **Quando uma entidade fraca for derivada de duas ou mais entidades no diagrama ER, ela poderá ser transformada diretamente em uma tabela sem nenhuma alteração.**

- **Etapas básicas de transformação do diagrama ER para tabelas SQL:**
  - **Etapa 1: transformar cada entidade em uma tabela contendo os atributos-chave e não-chave da entidade;**
  - **Etapa 2: transformar cada relacionamento binário N:N, recursivo ou não, em uma tabela com as chaves das entidades e os atributos do relacionamento;**
  - **Etapa 3: transformar cada relacionamento ternário ou *n*-ário de nível superior em uma tabela.**

- Se houver um relacionamento 1:N entre duas entidades, acrescente a chave da entidade do lado “um” (o pai) à tabela filho como uma chave estrangeira.

- Se houver um relacionamento 1:1 entre duas entidades, acrescente a chave de uma das entidades à tabela de outra entidade como uma chave estrangeira, em qualquer direção.

  - Manter o relacionamento pai-filho mais natural colocando a chave do pai na tabela filho ou

  - Estratégia baseada na eficiência: acrescente uma chave estrangeira à tabela com menos linhas.

- Cada entidade em uma hierarquia de generalização é transformada em uma tabela. Cada uma dessas tabelas contém uma chave da entidade supertipo; na realidade, as chaves primárias do subtipo também são chaves estrangeiras. A tabela supertipo também contém valores não-chaves específicos de cada entidade subtipo.

- Os construtores da SQL para essas transformações podem incluir restrições **not null**, **unique** e **foreign key**.
- Uma chave primária precisa ser especificada para cada tabela, seja explicitamente a partir das chaves no diagrama ER ou pela composição de todos os atributos como chave *default*.
- A designação de chave primária implica que o atributo é **not null** e **unique**.
- Nem todos os SGBDs seguem o padrão ANSI para esta questão – em alguns sistemas pode ser possível criar uma chave primária que pode ser nula. Por isso recomenda-se especificar o "not null" explicitamente para todos os atributos-chave.

- Cada relacionamento binário N:N é transformado em uma tabela contendo as chaves das entidades e os atributos do relacionamento. A tabela resultante mostrará a correspondência entre instâncias específicas de uma entidade com as de outra entidade.

- Qualquer atributo desta correspondência (como o escritório eleito que um engenheiro tem em uma associação profissional) é considerado dado da interseção e é acrescentado à tabela como um atributo não-chave.

- Os construtores SQL para essa transformação podem incluir restrições **not null**.

- A restrição **unique** não é usada aqui, pois todas as chaves são composições das chaves primárias das entidades participantes associadas ao relacionamento.

- As restrições para chave primária e chave estrangeira são obrigatórias, pois a tabela é definida como uma composição das chaves primárias das entidades associadas.

- Cada relacionamento ternário (ou *n*-ário mais alto) é transformado em uma tabela. São definidos como uma coleção das *n* chaves primárias das entidades associadas nesse relacionamento, possivelmente com alguns atributos não-chaves que são dependentes da chave formada pela composição das *n* chaves primárias.

- Os construtores SQL para essa transformação precisam incluir restrições **not null** pois a opcionalidade não é permitida.

- A restrição **unique** não é usada para atributos individuais, pois todas as chaves são composições das chaves primárias das entidades participantes associadas ao relacionamento.

- As restrições para chave primária e chave estrangeira são exigidas porque a tabela é definida como uma composição das chaves primárias das entidades associadas. A cláusula **unique** também precisa ser usada para definir chaves alternativas que normalmente ocorrem em relacionamentos ternários.

- Uma tabela derivada de um relacionamento *n*-ário possui *n* chaves estrangeiras.