PREVOD KONCEPTUALNEGA MODELA V SQL

Iztok Savnik

prevajanje entitet

- SQL tabela z isto informacijsko vsebino kot originalna entiteta iz katere je bila razvita
- transformacija se uporablja za entitete z binarnim razmerjem
  - N-N
  - 1-N na „ena“ strani
  - 1-1 na obeh straneh
- entitete, ki so povezane z binarno rekurzivno relacijo tipa N-N
- entitete s ternarnim razmerjem ali z razmerjem višjega reda ali z generalizacijsko hierarhijo

odvisne entitete

- SQL tabela z vgnezdenim tujim ključem entitete starša
- transformacija se vedno naredi za entitete, ki imajo binarno razmerje
  - 1-N za entiteto na N (otrok) strani
  - 1-1 za eno od entitet
  - za vsako entiteto, ki ima binarno rekurzivno razmerje tipa 1-1 ali 1-N
- to je eden izmed dveh načinov na katere načrtovalska orodja obravnavajo razmerja

pretvorba razmerij

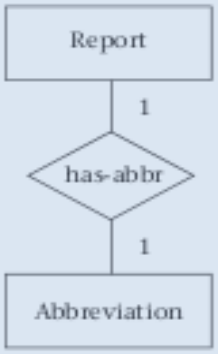
- SQL tabela izpeljana iz razmerja
  - vsebuje tuje ključe vseh entitet v razmerju
- transformacija se naredi za
  - binarna razmerja tipa N-N
  - razmerja, ki so rekurzivna in N-N
  - razmerja, ki so ternarna ali višjega reda
- to je drugi najbolj pogost način na katerega orodja obravnavajo razmerja v ER in UML
  - razmerje N-N se lahko definira samo s tabelo, ki vsebuje tuje ključe entitet v razmerju
  - ta tabela lahko vsebuje tudi atribute originalnega razmerja

uporaba null vrednosti

- null vrednosti so dovoljene v SQL tabelah za tuje ključe, ki povezujejo opcijske entitete
- null vrednosti niso dovoljene v SQL tabelah za tuje ključe , ki povezujejo obvezne entitete
- null vrednosti niso dovoljene za vse ključe v SQL tabeli, ki je bila izpeljana iz razmerja tipa N-N, ker so smiselne samo celotne vrstice

binarna razmerja

- pogledali si bomo primere binarnih razmerij glede na številnost razmerij
- kako prevedemo entitete? kako prevedemo binarno razmerje? kako prevedemo obvezno oz. opcijsko razmerje? katere SQL gradnike uporabljamo?
- imamo tri tipe razmerij glede na strukturo: 1-1, 1-N in N-N
- vsaka vloga v razmerju je lahko opcijska ali mandatorna



Every report has one abbreviation, and every abbreviation represents exactly one report.

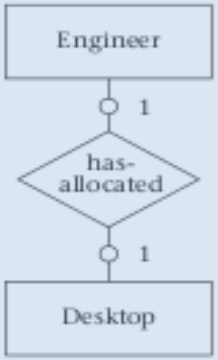
```
create table report
        (report_no integer,
         report_name varchar(256),
         primary key(report_no);
create table abbreviation
        (abbr_no char(6),
         report_no integer not null unique,
         primary key (abbr_no),
         foreign key (report_no) references report
             on delete cascade on update cascade);
```

(a) One-to-one, both entities mandatory

Every department must have a manager, but an employee can be a manager of at most one department.

```
create table department
        (dept_no integer,
         dept_name char(20),
         mgr_id char(10) not null unique,
         primary key (dept_no),
         foreign key (mgr_id) references employee
             on delete set default on update cascade);
create table employee
        (emp_id char(10),
         emp_name char(20),
         primary key (emp_id));
```

(b) One-to-one, one entity optional, one mandatory

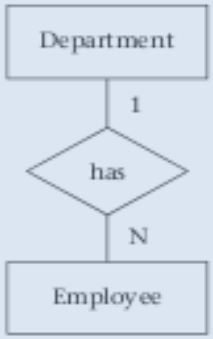


Some desktop computers are allocated to engineers, but not necessarily to all engineers.

```
create table engineer
        (emp_id char(10),
         desktop_no integer,
         primary key (emp_id));
create table desktop
        (desktop_no integer,
         emp_id char(10),
         primary key (desktop_no),
         foreign key (emp_id) references engineer
             on delete set null on update cascade);
```

(c) One-to-one, both entities optional

Figure 5.1 ER model: one-to-one binary relationship between two entities



Every employee works in exactly one department, and each department has at least one employee.

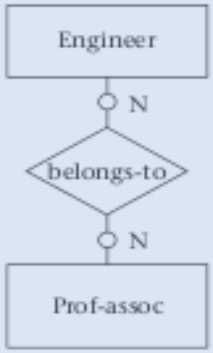
```
create table department
        (dept_no integer,
         dept_name char(20),
         primary key (dept_no));
create table employee
        (emp_id char(10),
         emp_name char(20),
         dept_no integer not null,
         primary key (emp_id),
         foreign key (dept_no) references department
            on delete set default on update cascade)
```

(d) One-to-many, both entities mandatory

Each department publishes one or more reports. A given report may not necessarily be published by a department.

```
create table department
        (dept_no integer,
         dept_name char(20),
         primary key (dept_no));
create table report
        (report_no integer,
         dept_no integer,
         primary key (report_no),
         foreign key (dept_no) references department
            on delete set null on update cascade);
```

(e) One-to-many, one entity optional, one mandatory



Every professional association could have none, one, or many engineer members. Each engineer could be a member of none, one, or many professional associations.

```
create table engineer
        (emp_id char(10),
         primary key (emp_id));
create table prof_assoc
        (assoc_name varchar(256),
         primary key (assoc_name));
create table belongs_to
        (emp_id char(10),
         assoc_name varchar(256),
         primary key (emp_id, assoc_name),
         foreign key (emp_id) references engineer
            on delete cascade on update cascade,
         foreign key (assoc_name) references prof_assoc
            on delete cascade on update cascade);
```

(f) Many-to-many, both entities optional

Figure 5.1 *(continued)*

Every report has one abbreviation, and every abbreviation represents exactly one report.

```
create table report
        (report_no integer,
         report_name varchar(256),
         primary key(report_no));
create table abbreviation
        (abbr_no char(6),
         report_no integer not null unique,
         primary key (abbr_no),
         foreign key (report_no) references report
            on delete cascade on update cascade);
```

(a) one-to-one, both entities mandatory

Every department must have a manager, but an employee can be a manager of at most one department.

```
create table department
        (dept_no integer,
         dept_name char(20),
         mgr_id char(10) not null unique,
         primary key (dept_no),
         foreign key (mgr_id) references employee
            on delete set default on update cascade);
create table employee
        (emp_id char(10),
         emp_name char(20),
         primary key (emp_id));
```

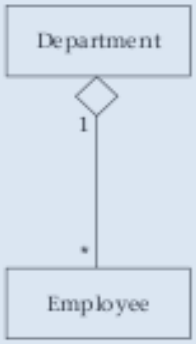
(b) one-to-one, one entity optional, one mandatory

Some desktop computers are allocated to engineers, but not necessarily to all engineers.

```
create table engineer
        (emp_id char(10),
         desktop_no integer,
         primary key (emp_id));
create table desktop
        (desktop_no integer,
         emp_id char(10),
         primary key (desktop_no),
         foreign key (emp_id) references engineer
            on delete set null on update cascade);
```

(c) one-to-one, both entities optional

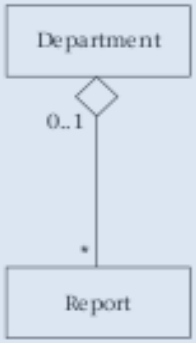
Figure 5.2 UML: one-to-one binary relationship between two entities



Every employee works in exactly one department, and each department has at least one employee.

```
create table department
        (dept_no integer,
         dept_name char(20),
         primary key (dept_no));
create table employee
        (emp_id char(10),
         emp_name char (20),
         dept_no integer not null,
         primary key (emp_id),
         foreign key (dept_no) references department
            on delete set default on update cascade);
```

(d) one-to-many, both entities mandatory



Each department publishes one or more reports. A given report may not necessarily be published by a department.

```
create table department
        (dept_no integer,
         dept_name char(20),
         primary key (dept_no));
create table report
        (report_no integer,
         dept_no integer,
         primary key (report_no),
         foreign key (dept_no) references department
            on delete set null on update cascade);
```

(e) one-to-many, one entity optional, one mandatory

Every professional association could have none, one, or many engineer members. Each engineer could be a member of none, one, or many professional associations.

```
create table engineer
        (emp_id char(10),
         primary key (emp_id));
create table prof_assoc
        (assoc_name varchar(256),
         primary key (assoc_name));
create table belongs_to
        (emp_id char(10),
         assoc_name varchar(256),
         primary key (emp_id, assoc_name),
         foreign key (emp_id) references engineer
            on delete cascade on update cascade,
         foreign key (assoc_name) references prof_assoc
            on delete cascade on update cascade);
```

(f) many-to-many, both entities optional

Figure 5.2 *(continued)*

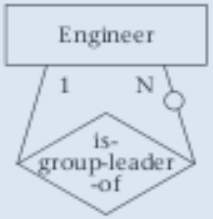
rekurzivna razmerja

- binarno razmerje na eni sami entiteti zahteva definicijo parov primerkov iste entitete
- pari so lahko popolnoma opcijski ali na drugi strani obvezni
- pari entitet so definirani s tujimi ključi v tabeli s katero je predstavljeno rekurzivno razmerje
- pari so lahko definirani v svoji tabeli ali so znotraj tabele entitete odvisno od tipa razmerja

Any employee is allowed to be married to another employee in this company.

```
create table employee
        (emp_id char(10),
         emp_name char(20),
         spouse_id char(10),
         primary key (emp_id),
         foreign key (spouse_id) references employee
             on delete set null on update cascade);
```

(a) One-to-one, both sides optional



Engineers are divided into groups for certain projects. Each group has a leader.

```
create table engineer
        (emp_id char(10),
         leader_id char(10) not null,
         primary key (emp_id),
         foreign key (leader_id) references engineer
             on delete set default on update cascade);
```

(b) One-to-many, one side mandatory, many side optional

Each employee has the opportunity to coauthor a report with one or more other employees, or to write the report alone.

```
create table employee
        (emp_id char(10),
         emp_name char(20),
         primary key (emp_id));
create table coauthor
        (author_id char(10),
         coauthor_id char(10),
         primary key (author_id, coauthor_id),
         foreign key (author_id) references employee
             on delete cascade on update cascade,
         foreign key (coauthor_id) reference employee
             on delete cascade on update cascade);
```

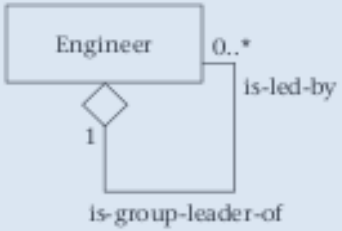
(c) Many-to-many, both sides optional

Figure 5.3 ER model: binary recursive relationship

Any employee is allowed to be married to another employee in this company.

```
create table employee
        (emp_id char(10),
         emp_name char(20),
         spouse_id char(10),
         primary key (emp_id),
         foreign key (spouse_id) references employee
            on delete set null on update cascade);
```

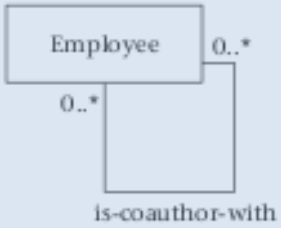
(a) one-to-one, both sides optional



Engineers are divided into groups for certain projects. Each group has a leader.

```
create table engineer
        (emp_id char(10),
         leader_id char(10) not null,
         primary key (emp_id),
         foreign key (leader_id) references engineer
            on delete set default on update cascade);
```

(b) one-to-many, one side mandatory, many side optional



Each employee has the opportunity to coauthor a report with one or more other employees, or to write the report alone.

```
create table employee
        (emp_id char(10),
         emp_name char(20),
         primary key (emp_id));
create table coauthor
        (author_id char(10),
         coauthor_id char(10),
         primary key (author_id, coauthor_id),
         foreign key (author_id) references employee
            on delete cascade on update cascade,
         foreign key (coauthor_id) references employee
            on delete cascade on update cascade);
```

(c) many-to-many, both sides optional

Figure 5.4 UML: binary recursive relationship

ternarna razmerja

- imamo štiri možne strukture glede na tip razmerja: 1-1-1, 1-1-N, 1-N-N in N-N-N
- ogledali si bomo vse 4 možnosti
- število „ena“ strani razmerja določa število funkcijskih odvisnosti
- vsi tuji ključi morajo biti definirani s „cascade“:

A technician uses exactly one notebook for each project. Each notebook belongs to one technician for each project. Note that a technician may still work on many projects and maintain different notebooks for different projects.

```
create table technician (emp_id char(10),
                         primary key (emp_id));
create table project (project_name char(20),
                      primary key (project_name));
create table notebook (notebook_no integer,
                       primary key (notebook_no));
create table uses_notebook (emp_id char(10),
                            project_name char(20),
                            notebook_no integer not null,
                            primary key (emp_id, project_name),
                            foreign key (emp_id) references technician
                               on delete cascade on update cascade,
                            foreign key (project_name) references project
                               on delete cascade on update cascade,
                            foreign key (notebook_no) references notebook
                               on delete cascade on update cascade,
                            unique (emp_id, notebook_no),
                            unique (project_name, notebook_no));
```

uses_notebook

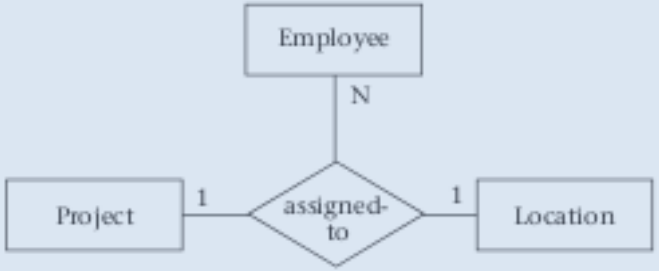
emp_id	project_name	notebook_no
35	alpha	5001
35	gamma	2008
42	delta	1004
42	epsilon	3005
81	gamma	1007
93	alpha	1009
93	beta	5001

Functional dependencies

emp_id, project_name → notebook_no
emp_id, notebook_no → project_name
project_name, notebook_no → emp_id

(a) One-to-one-to-one ternary relationship

Figure 5.5 ER model: ternary and *n*-ary relationships



Each employee assigned to a project works at only one location for that project, but can be at a different location for a different project. At a given location, an employee works on only one project. At a particular location, there can be many employees assigned to a given project.

```
create table employee (emp_id char(10),
                       emp_name char(20),
                       primary key (emp_id));
create table project (project_name char(20),
                      primary key (project_name));
create table location (loc_name char(15),
                       primary key (loc_name));
create table assigned_to (emp_id char(10),
                          project_name char(20),
                          loc_name char(15) not null,
                          primary key (emp_id, project_name),
                          foreign key (emp_id) references employee
                             on delete cascade on update cascade,
                          foreign key (project_name) references project
                             on delete cascade on update cascade,
                          foreign key (loc_name) references location
                             on delete cascade on update cascade,
                          unique (emp_id, loc_name));
```

assigned_to

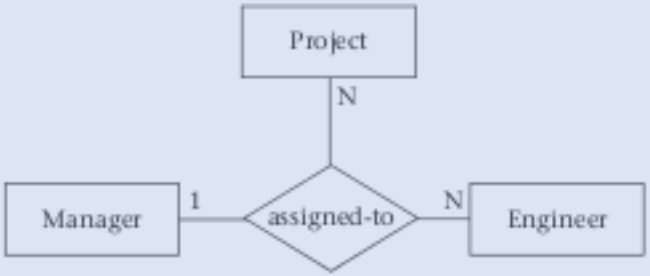
emp_id	project_name	loc_name
48101	forest	B66
48101	ocean	E71
20702	ocean	A12
20702	river	D54
51266	river	G14
51266	ocean	A12
76323	hills	B66

Functional dependencies

emp_id, loc_name → project_name
emp_id, project_name → loc_name

(b) One-to-one-to-many ternary relationships

Figure 5.5 *(continued)*



Each engineer working on a particular project has exactly one manager, but a project can have many managers and an engineer may have many managers and many projects. A manager may manage several projects.

```
create table project (project_name char(20),
                      primary key (project_name));
create table manager (mgr_id char(10),
                      primary key (mgr_id));
create table engineer (emp_id char(10),
                       primary key (emp_id));
create table manages (project_name char(20),
                      mgr_id char(10) not null,
                      emp_id char(10),
                      primary key (project_name, emp_id),
                      foreign key (project_name) references project
                         on delete cascade on update cascade,
                      foreign key (mgr_id) references manager
                         on delete cascade on update cascade,
                      foreign key (emp_id) references engineer
                         on delete cascade on update cascade);
```

manages

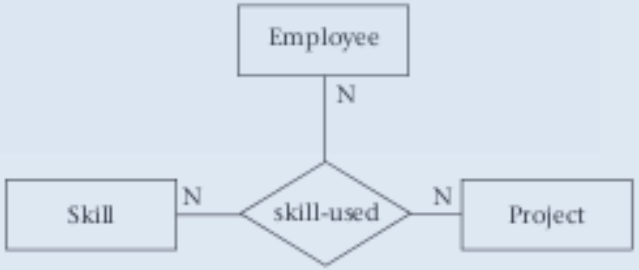
project_name	emp_id	mgr_id
alpha	4106	27
alpha	4200	27
beta	7033	32
beta	4200	14
gamma	4106	71
delta	7033	55
delta	4106	39
iota	4106	27

Functional dependency

project_name, emp_id → mgr_id

(c) One-to-many-to-many ternary relationships

Figure 5.5 *(continued)*



Employees can use different skills on any one of many projects, and each project has many employees with various skills.

```
create table employee (emp_id char(10),
                       emp_name char(20),
                       primary key (emp_id));
create table skill (skill_type char(15),
                    primary key (skill_type));
create table project (project_name char(20),
                      primary key (project_name));
create table skill_used (emp_id char(10),
                         skill_type char(15),
                         project_name char(20),
                         primary key (emp_id, skill_type, project_name),
                         foreign key (emp_id) references employee
                            on delete cascade on update cascade,
                         foreign key (skill_type) references skill
                            on delete cascade on update cascade,
                         foreign key (project_name) references project
                            on delete cascade on update cascade);
```

skill_used

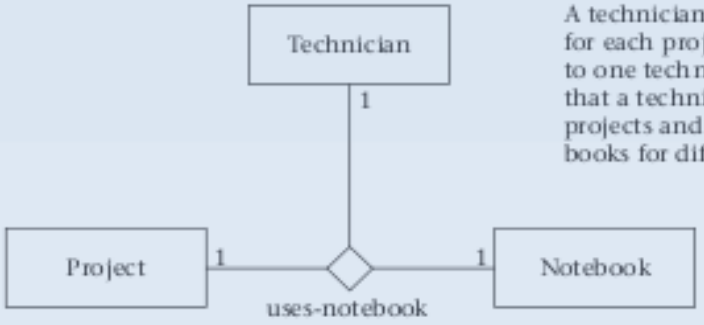
emp_id	skill_type	project_name
101	algebra	electronics
101	calculus	electronics
101	algebra	mechanics
101	geometry	mechanics
102	algebra	electronics
102	set theory	electronics
102	geometry	mechanics
105	topology	mechanics

Functional dependencies

None

(d) Many-to-many-to-many ternary relationships

Figure 5.5 *(continued)*



A technician uses exactly one notebook for each project. Each notebook belongs to one technician for each project. Note that a technician may still work on many projects and maintain different notebooks for different projects.

```
create table technician (emp_id char(10),
                         primary key (emp_id));
create table project (project_name char(20),
                      primary key (project_name));
create table notebook (notebook_no integer,
                       primary key (notebook_no));
create table uses_notebook (emp_id char(10),
                            project_name char(20),
                            notebook_no integer not null,
                            primary key (emp_id, project_name),
                            foreign key (emp_id) references technician
                               on delete cascade on update cascade,
                            foreign key (project_name) references project
                               on delete cascade on update cascade,
                            foreign key (notebook_no) references notebook
                               on delete cascade on update cascade,
                            unique (emp_id, notebook_no),
                            unique (project_name, notebook_no));
```

uses_notebook

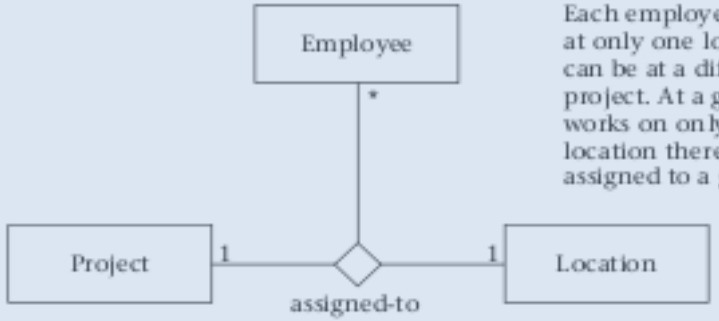
emp_id	project_name	notebook_no
35	alpha	5001
35	gamma	2008
42	delta	1004
42	epsilon	3005
81	gamma	1007
93	alpha	1009
93	beta	5001

Functional dependencies

emp_id, project_name → notebook_no
emp_id, notebook_no → project_name
project_name, notebook_no → emp_id

(a) one-to-one-to-one ternary association

Figure 5.6 UML: ternary and *n*-ary relationships



Each employee assigned to a project works at only one location for that project, but can be at a different location for a different project. At a given location, an employee works on only one project. At a particular location there can be many employees assigned to a given project.

```
create table employee (emp_id char(10),
                       emp_name char(20),
                       primary key (emp_id));
create table project (project_name char(20),
                      primary key (project_name));
create table location (loc_name char(15),
                       primary key (loc_name));
create table assigned_to (emp_id char(10),
                          project_name char(20),
                          loc_name char(15) not null,
                          primary key (emp_id, project_name),
                          foreign key (emp_id) references employee
                              on delete cascade on update cascade,
                          foreign key (project_name) references project
                              on delete cascade on update cascade,
                          foreign key (loc_name) references location
                              on delete cascade on update cascade,
                          unique (emp_id, loc_name));
```

assigned_to

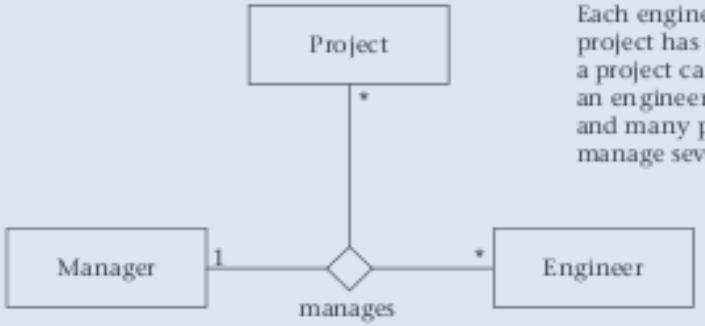
emp_id	project_name	loc_name
48101	forest	B66
48101	ocean	E71
20702	ocean	A12
20702	river	D54
51266	river	G14
51266	ocean	A12
76323	hills	B66

Functional dependencies

emp_id, loc_name → project_name
emp_id, project_name → loc_name

(b) one-to-one-to-many ternary associations

Figure 5.6 *(continued)*



Each engineer working on a particular project has exactly one manager, but a project can have many managers and an engineer may have many managers and many projects. A manager may manage several projects.

```
create table project (project_name char(20),
                      primary key (project_name));
create table manager (mgr_id char(10),
                      primary key (mgr_id));
create table engineer (emp_id char(10),
                       primary key (emp_id));
create table manages (project_name char(20),
                      mgr_id char(10) not null,
                      emp_id char(10),
                      primary key (project_name, emp_id),
                      foreign key (project_name) references project
                          on delete cascade on update cascade,
                      foreign key (mgr_id) references manager
                          on delete cascade on update cascade,
                      foreign key (emp_id) references engineer
                          on delete cascade on update cascade);
```

manages

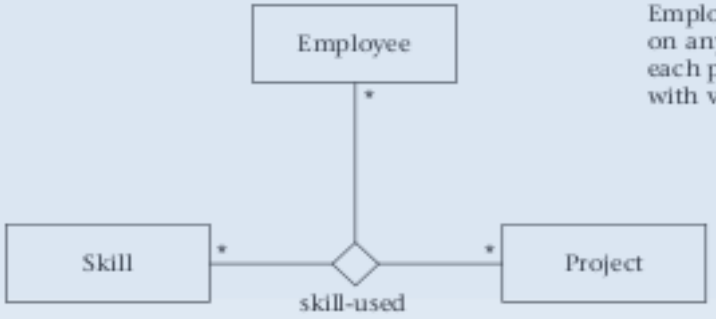
project_name	emp_id	mgr_id
alpha	4106	27
alpha	4200	27
beta	7033	32
beta	4200	14
gamma	4106	71
delta	7033	55
delta	4106	39
iota	4106	27

Functional dependency

project_name, emp_id → mgr_id

(c) one-to-many-to-many ternary association

Figure 5.6 *(continued)*



Employees can use different skills on any one of many projects, and each project has many employees with various skills.

```
create table employee (emp_id char(10),
                       emp_name char(20),
                       primary key (emp_id));
create table skill (skill_type char(15),
                    primary key (skill_type));
create table project (project_name char(20),
                      primary key (project_name));
create table skill_used (emp_id char(10),
                         skill_type char(15),
                         project_name char(20),
                         primary key (emp_id, skill_type, project_name),
                         foreign key (emp_id) references employee
                            on delete cascade on update cascade,
                         foreign key (skill_type) references skill
                            on delete cascade on update cascade,
                         foreign key (project_name) references project
                            on delete cascade on update cascade);
```

skill_used

emp_id	skill_type	project_name
101	algebra	electronics
101	calculus	electronics
101	algebra	mechanics
101	geometry	mechanics
102	algebra	electronics
102	set-theory	electronics
102	geometry	mechanics
105	topology	mechanics

Functional dependencies

None

(d) many-to-many-to-many ternary association

Figure 5.6 *(continued)*

generalizacija

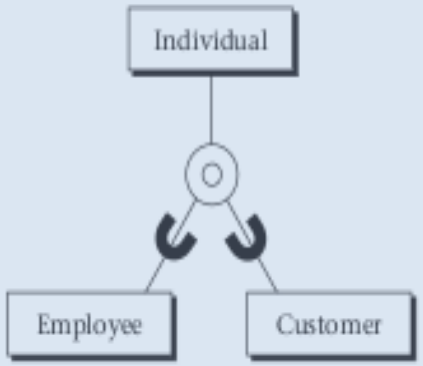
- prevod generalizacijske hierarhije sestavljene iz supertipa in podtipov lahko vodi do več SQL tabel
- tabela izpeljana iz supertipa vsebuje ključ entitete supertipa in vse skupne atribute hierarhije
- vsaka tabela izpeljana iz entitet podtipa vsebuje ključ entitete supertipa in samo atribute specifične za podtip
- integriteto pri popravljanju je potrebno ohranjati tako, da vsi popravki, vstavljanja in brisanja spreminjajo tako tabelo super tipa kot tudi pripadajočo tabelo podtipa
  - uporabiti je potrebno omejitev cascade za tuje ključe
  - popravek ključa supertipa zahteva popravek v hierarhiji
  - popravek opisnega atributa zahteva popravek samo enega zapisa

generalizacija (2)

- transformacijska pravila so ista za primer prekrivanja kot tudi za disjunktne podtipe
- drugi pristop je definicija ene same tabele, ki vsebuje vse atribute tako podtipa kot tudi nadtipa
  - celotna hierarhija v eni tabeli
  - po potrebi uporabimo null vrednosti
- tretji pristop je ena tabela za en podtip vendar porinemo skupne atribute navzdol k tabelam podtipov
- imamo prednosti in slabosti teh treh pristopov
- programska orodja za načrtovanje navadno uporabljajo vse tri možnosti

generalizacija (3)

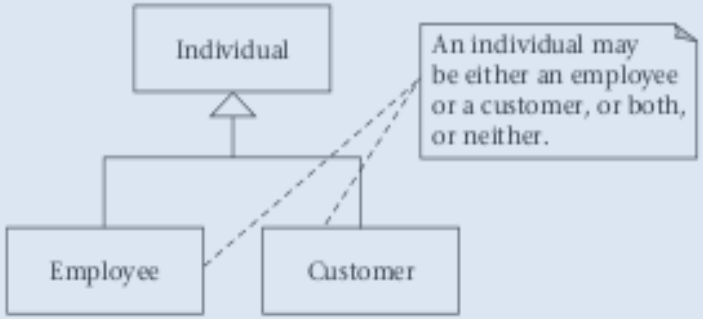
- praktični sistemi velikokrat uporabijo klasifikacijski atribut v nadtipu s katerim lahko ločimo med instancami podtipov
- če imamo več-nivojsko hierarhijo potem lahko uporabimo več klasifikacijskih atributov
- pristop kombinira sistem tipov s klasifikacijsko hierarhijo definirano glede na vrednost izbranih atributov



An individual may be either an employee or a customer, or both, or neither.

```
create table individual (indiv_id char(10),
                         indiv_name char(20),
                         indiv_addr char(20),
                         primary key (indiv_id));
create table employee (emp_id char(10),
                       job_title char(15),
                       primary key (emp_id),
                       foreign key (emp_id) references individual
                          on delete cascade on update cascade);
create table customer (cust_no char(10),
                       cust_credit char(12),
                       primary key (cust_no),
                       foreign key (cust_no) references individual
                          on delete cascade on update cascade);
```

Figure 5.7 ER model: generalization and aggregation




```
create table individual (indiv_id char(10),
                         indiv_name char(20),
                         indiv_addr char(20),
                         primary key (indiv_id));
create table employee (emp_id char(10),
                       job_title char(15),
                       primary key (emp_id),
                       foreign key (emp_id) references individual
                           on delete cascade on update cascade);
create table customer (cust_no char(10),
                       cust_credit char(12),
                       primary key (cust_no),
                       foreign key (cust_no) references individual
                           on delete cascade on update cascade);
```

Figure 5.8 UML: generalization and aggregation

agregacija

- prevod agregacijske hierarhije tudi vodi do ločenih tabel za ločene podtipe
- agregacijo obravnavamo kot razmerje in uporabimo enaka pravila za prevajanje v tabele
- v primeru agregacije nimamo skupnih atributov in integritetnih omejitev, ki jih je potrebno vzdrževati
- glavna funkcija agregacije je definicija abstrakcije, ki lahko pomaga pri integraciji shem
- v UML predstavlja agregacija relacijo kompozicije, ki ustreza šibkim entitetam

koraki transformacije

- pretvori vsako entiteto v tabelo s ključem in opisnimi atributi entitete
- pretvori vsako binarno in rekurzivno razmerje tipa N-N v tabelo s ključi entitet(e) in opisnimi atributi razmerja
- pretvori vsako ternarno ali relacijo višjega reda v tabelo

prevod entitet

- 1-N razmerje med dvema entitetama
  - dodaj ključ entitete na „ena“ strani k tabeli otroka
- 1-1 razmerje med dvema entitetama:
  - dodaj ključ ene entitete v tabelo druge, kjer ima vlogo tujega ključa
  - dodajanje tujega ključa je lahko v poljubno smer
- strategije:
  - uniformno dodajanje ključa otroku
  - osnovana na učinkovitosti: dodaj tuj ključ v tabelo, ki ima manj vrstic

prevod entitet (2)

- vsaka entiteta v generalizacijski hierarhiji se spremeni v tabelo
- vsaka od teh tabel vsebuje ključ nadrejene entitete (v realnosti)
- primarni ključi podrejenih entitet so tudi tuji ključi
- tabela supertipa vsebuje opisne atribute, ki so skupni vsem izbranim entitetam
- druge tabele vsebujejo opisne atribute specifične za entitete podtipa

prevod entitet (3)

- SQL gradniki za prevod so not null, unique, in foreign key
- primarni ključ mora biti definiran za vsako tabelo
  - lahko eksplicitno iz ključa entitete v ER diagrama
  - privzet ključ je sestavljen iz vseh atributov
  - primarni ključ ==> not null in unique
- vsi SUPB se ne držijo standarda
  - dobro je eksplicitno definirati not null za vse atribute ključa

prevod razmerja N-N

- binarna razmerja N-N se prevedejo v tabelo, ki vsebuje ključe entitet in atribute razmerij
- tabela vsebuje zveze med instancami ene in druge entitete
- katerikoli atribut te zveze se predstavi z opisnim atributom tabele
- SQL gradniki za pretvorbo:
  - not null
  - omejitev unique se ne uporablja za individualne atribute, ker so vsi kluči sestavljeni iz tujih ključev entitet v razmerju
  - primary key + foreign key
-

prevod ternarnega razmerja

- vsaka ternarno razmerje se prevede v tabelo
- ternarno ali n-arno razmerje se definira s kolekcijo n primarnih ključev entitet povezanih v razmerju
- tabela vsebuje tudi opisne atribute odvisne od ključa sestavljenega iz n primarnih ključev
- SQL gradniki za pretvorbo:
  - not null -- vsi atributi ključa!
  - omejitev unique se ne uporablja za posamezne atribute, ker so vsi kluči sestavljeni iz tujih ključev entitet v razmerju
  - primary key + foreign key