Úvod do databázových systémov

http://www.dcs.fmph.uniba.sk/~plachetk

/TEACHING/DB1

Tomáš Plachetka

Fakulta matematiky, fyziky a informatiky,
Univerzita Komenského, Bratislava

Zima 2023–2024

Cieľ a metodológia navrhovania databáz

• Návrhu databázy predchádza plánovanie, t.j. zber požiadaviek na aplikáciu. Metodológie špecifikácie systémov: SAD (Structured Analysis and Design), DFD (Data-Flow Diagrams), UML (Unified Modeling Language)

• Cieľom návrhu databázy (Database Design) je modelovanie časti reálneho sveta vo zvolenom dátovom modeli, obvykle relačnom. Začína sa s **koncepčným návrhom**, ktorého výsledkom je napr. UML class diagram alebo ER diagram (Entity-Relationship). Nasleduje **logický návrh** (stále nezávislý od hardwaru či softwaru), ktorého výsledkom sú relácie, typy atribútov, bezpečnostný model atď. (sem patrí aj proces normalizácie). Nasleduje **fyzický návrh**, ktorého cieľom je mapovanie logického návrhu na konkrétny DBMS a hardware. Výsledkom sú vytvorené tabuľky, kľúče, indexy, constrainty, užívateľské kontá, procedúry vkladania/vynechávania dát, pohľady (VIEWs), prístupové práva atď.

• Po návrhu nasleduje správa a používanie databázy

Koncepčný návrh

Jazyky na modelovanie reality (väčšinou vizuálna reprezentácia modelu):

- Entitno-relačné diagramy (ER diagramy)
- UML diagramy tried (class diagrams)

Existujú softwarové nástroje, ktoré uľahčujú koncepčný návrh a čiastočne automatizujú následné fázy návrhu:

- Rational Rose (IBM)
- Visio (Microsoft)

Koncepčný návrh: ER diagramy

ER diagramy:

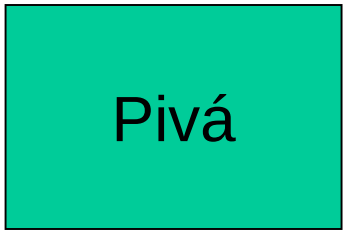
• popisujú aké veci (entity) v modeli vystupujú a ako spolu súvisia

• nemajú **žiadne operácie**, t.j. nepopisujú ako sa veci menia (napríklad ako vznikajú a zanikajú)

• dajú sa kresliť rôznymi spôsobmi, pričom však popisujú ten istý model („klasická“ syntax alebo UML syntax pre diagramy tried)

Koncepčný návrh: ER diagramy

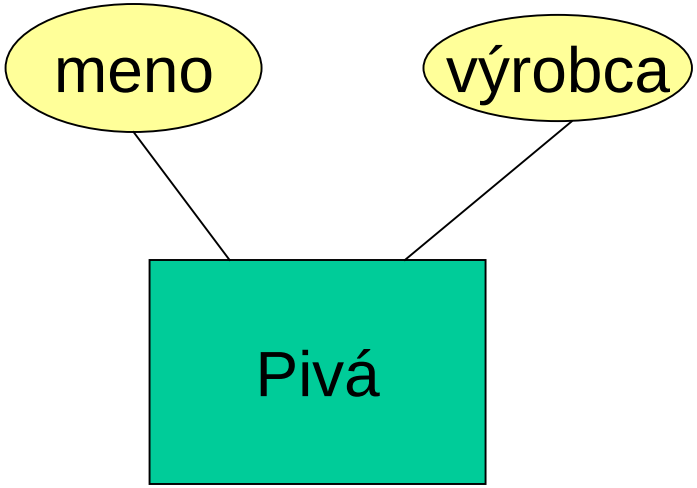
Entity sets



Entity, ktoré modelujeme, sú pivá. Pod tým obdĺžnikom rozumieme množinu pív (entity set), nie jedno konkrétne pivo—preto ten plurál

Koncepčný návrh: ER diagramy

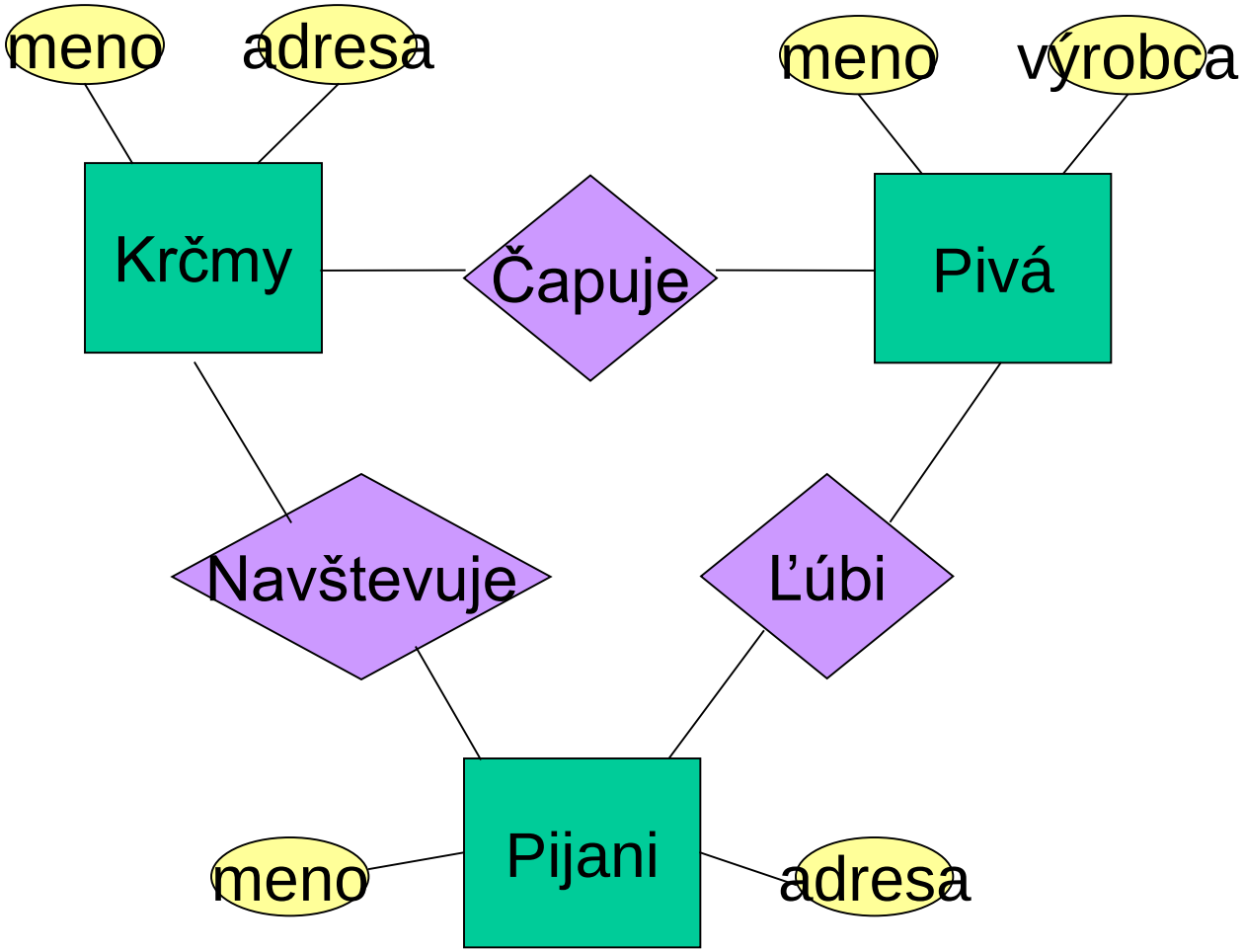
Atribúty



Každé pivo má svoje meno a svojho výrobcu

Koncepčný návrh: ER diagramy

Vzťahy (relationships)



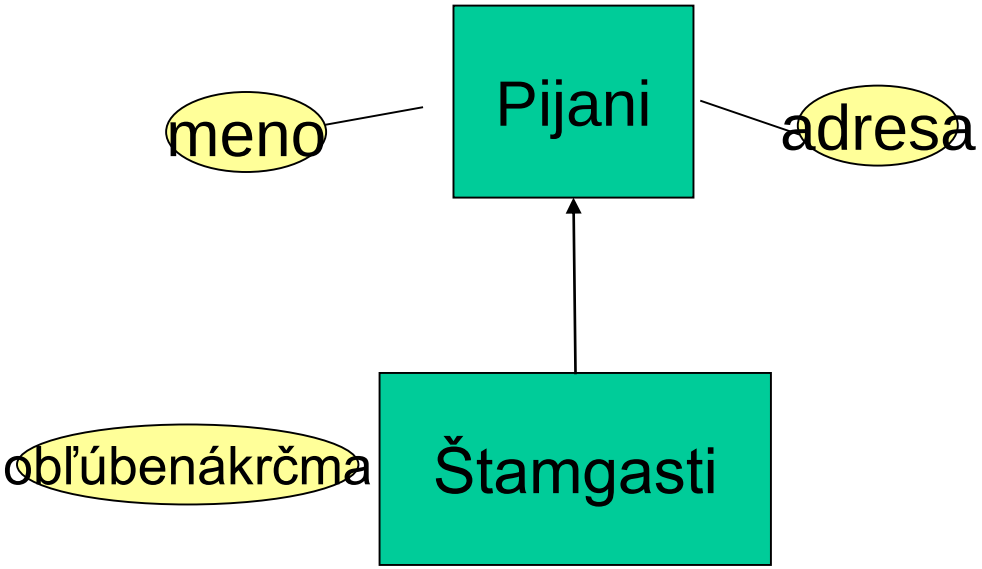
Krčmy čapujú pivá

Pijani ľúbia pivá

Pijani navštevujú krčmy

Koncepčný návrh: ER diagramy

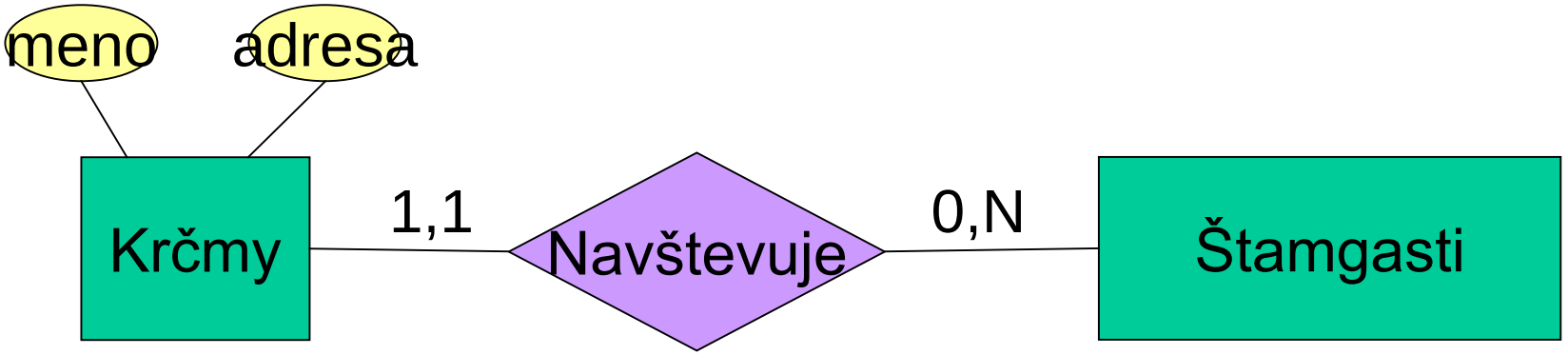
Špecializované entity (is a)



Štamgasti sú pijani, ktorí majú svoje obľúbené krčmy. (Nie každý pijan má svoju obľúbenú krčmu.)

Rozdiel oproti O-O dedeniu: v O-O objekt štamgast patrí iba do podtriedy Štamgasti. V ER objekt štamgast patrí aj do Pijanov

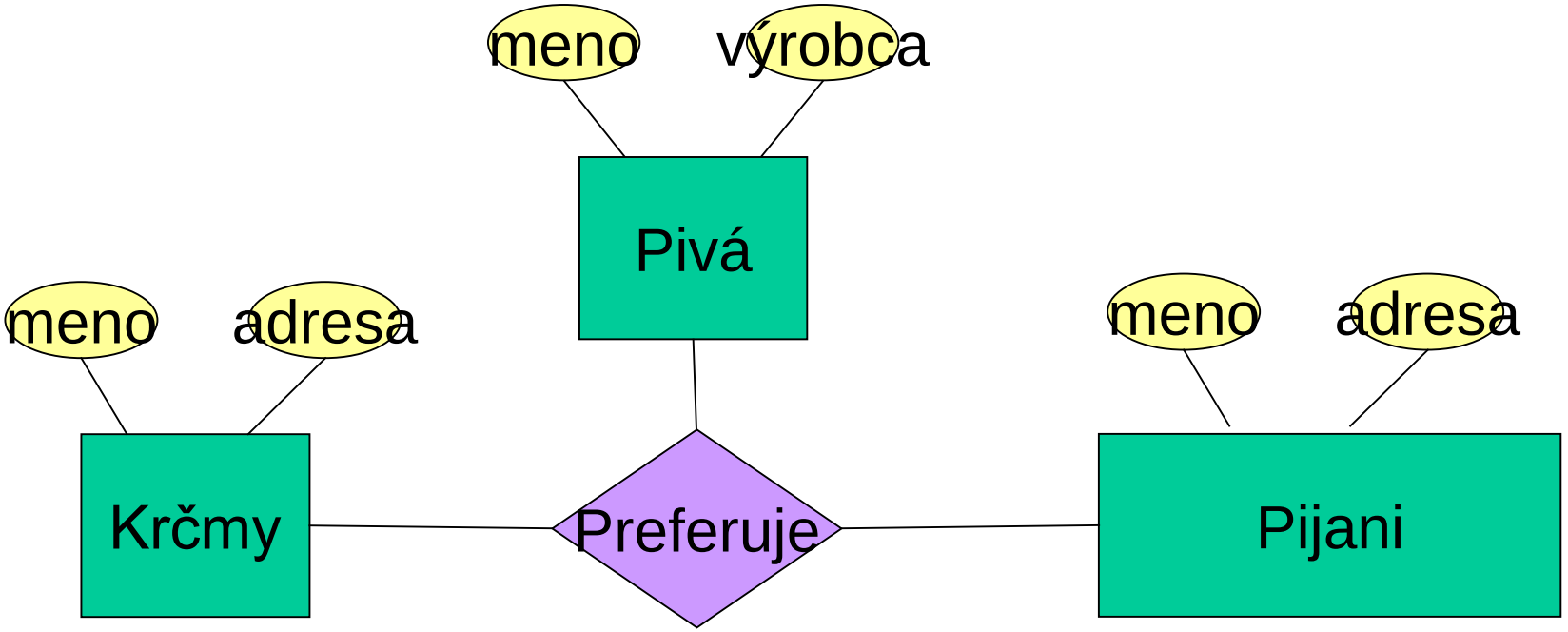
Relácie (relationships) s multiplicitami výskytov



Štamgasti sú pijani, ktorí navštevujú práve jednu krčmu. Tú istú krčmu môže navštevovať ľubovoľne veľa (od 0 po N) štamgastov

Koncepčný návrh: ER diagramy

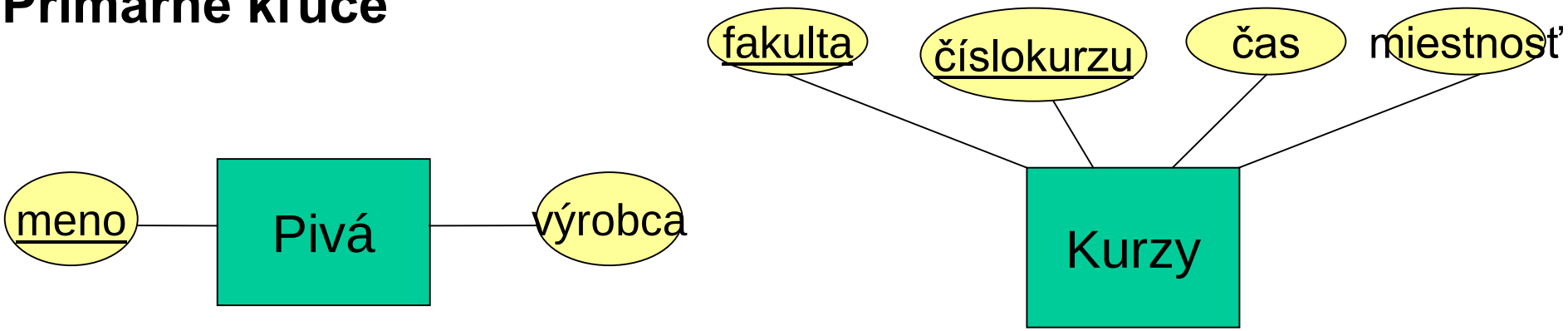
N-árne relácie



Niektorí pijani pijú Staropramen zásadne u Mamuta, ale v krčme U jeleňa pijú zásadne Stein

Koncepčný návrh: ER diagramy

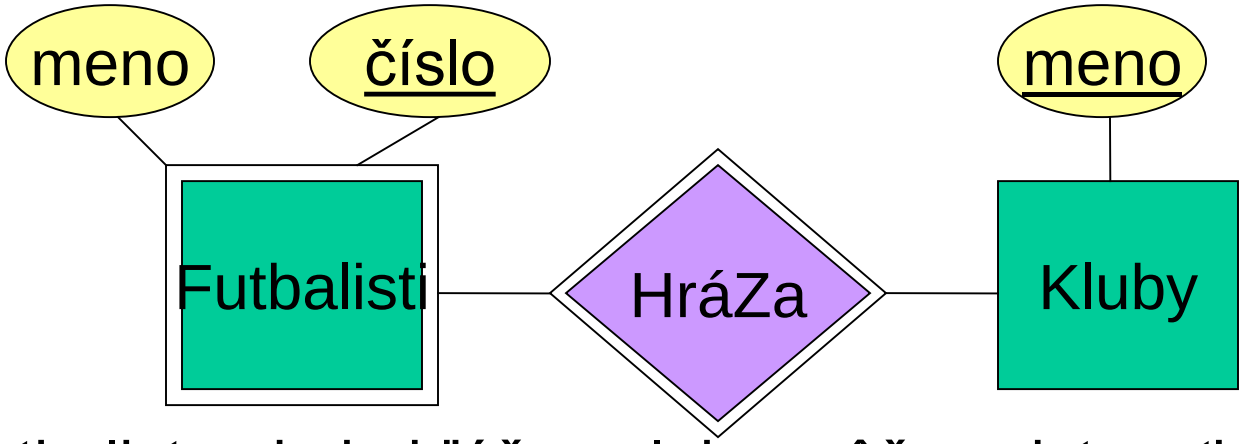
Primárne kľúče



Kľúč je (neformálne) minimálna množina atribútov, ktorá jednoznačne identifikuje entitu. Kľúčov môže byť viac. Primárny kľúč je niektorý z kľúčov a označuje sa počiarknutím atribútov, ktoré ho tvoria. V ER diagramoch sa vyžaduje, aby každý entity set mal nejaký kľúč. V každom prípade je možné do entity set pridať tzv. surrogate key, ktorý slúži len pre ten účel, že je primárnym kľúčom. V špecializovaných entity sets (is-a hierarchia) musí byť primárny kľúč v najvyššej triede zároveň kľúčom vo všetkých podtriedach

Koncepčný návrh: ER diagramy

Slabé entity sets



Meno futbalistu nie je kľúčom, lebo môžu existovať dvaja s rovnakým menom. Číslo tiež nie je kľúčom, lebo hráči rôznych klubov môžu mať na dresoch rovnaké číslo (a navyše môžu byť menovci). **Avšak číslo futbalistu spolu s menom klubu jednoznačne identifikujú futbalistu.** Keďže kľúč pre Futbalisti závisí od inej entity, Futbalisti je slabý entity set, ktorý musí byť podporený vzťahom HráZa a entity setom Kluby

Inými slovami, **existencia futbalistu je závislá na existencii klubu, za ktorý ten futbalista hrá**

Pravidlá dobrého návrhu

1. **Vyhýbať sa redundancii** (Occamova britva)
2. **Vyhýbať sa slabým entity setom**
3. **Nepoužívať entity set, keď sa dá nahradiť atribútom**

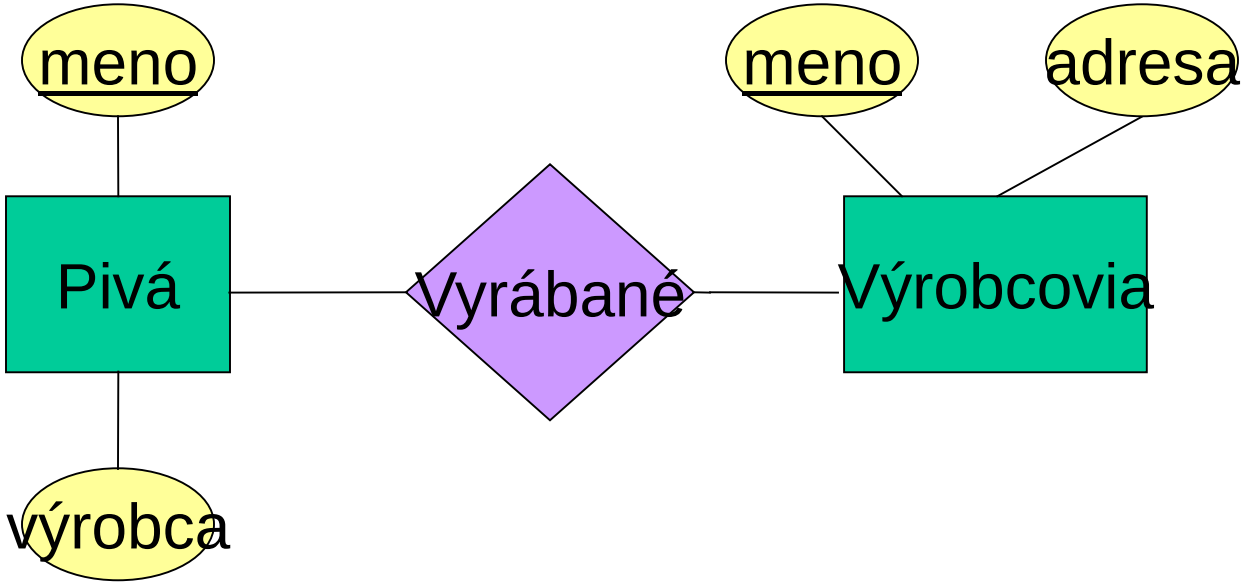
Redundancia znamená, že tá istá vec je vyjadrená niekoľkými rôznymi spôsobmi. Redundancia sa prejaví:

- plytvaním pamäťou (málo vadí)
- potrebou NULL hodnôt (málo vadí)
- **rizikom nekonzistencie (veľmi vadí)**

Slabé entity sets sa prejavia

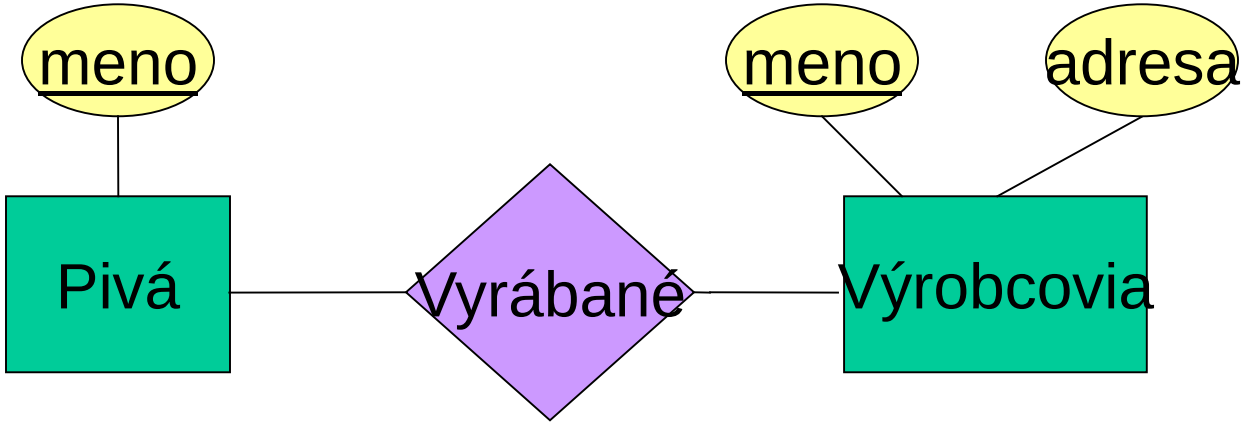
- **anomáliou pri vynechávaní (veľmi vadí)**

Príklad chyby návrhu: redundantná entita



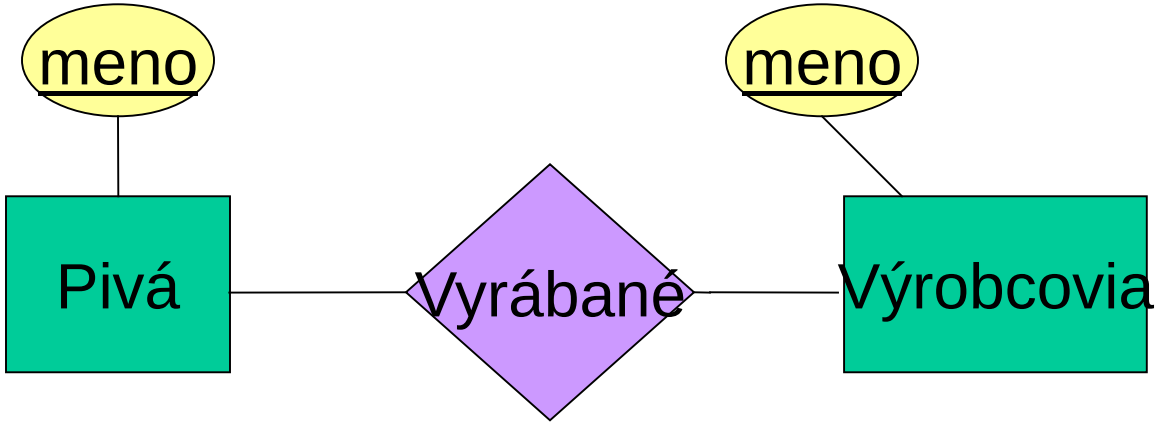
ZLE! Výrobca je aj atribút aj súvisiaci entity set

Príklad chyby návrhu: redundantná entita



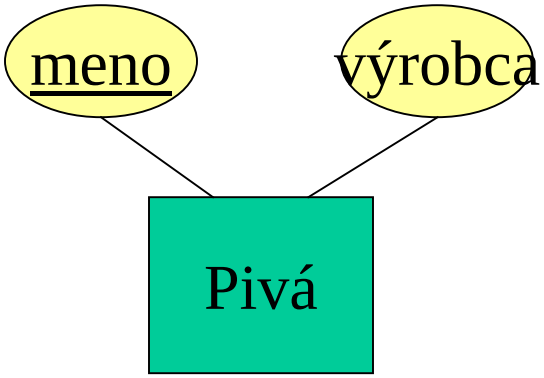
Správne

Príklad chyby návrhu: redundantná entita



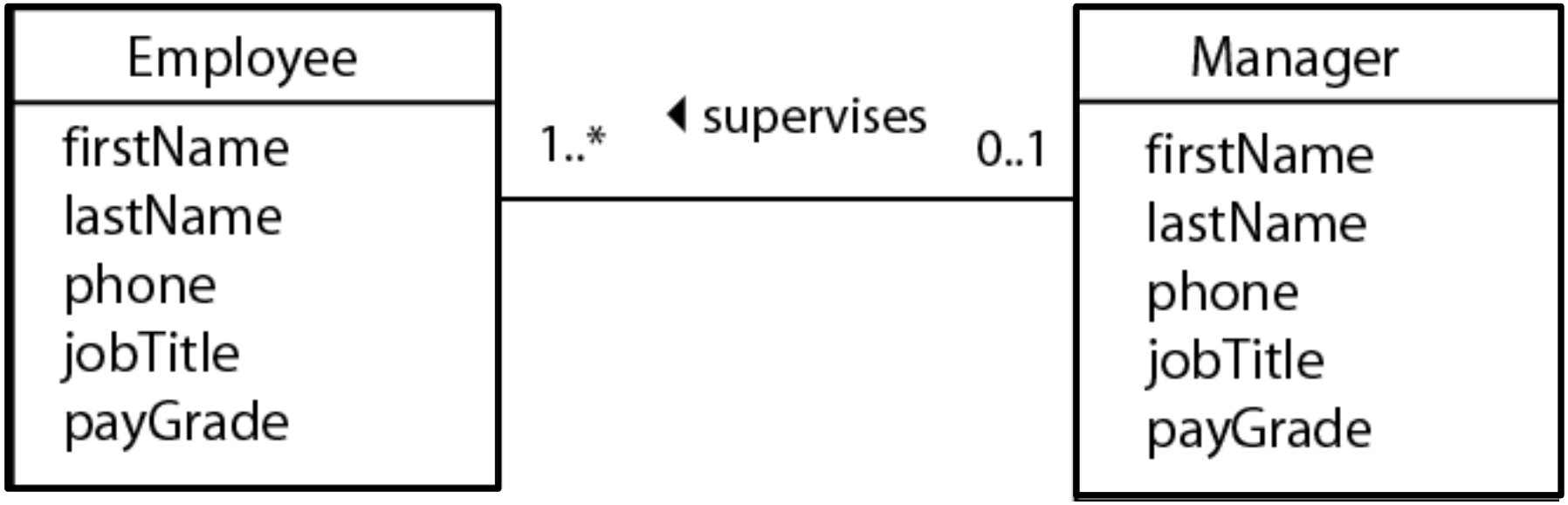
ZLE! Ak nás nezaujímajú adresy výrobcov, ale len ich mená, tak si Výrobcovia nezaslúžia byť entity setom.

Príklad chyby návrhu: redundantná entita



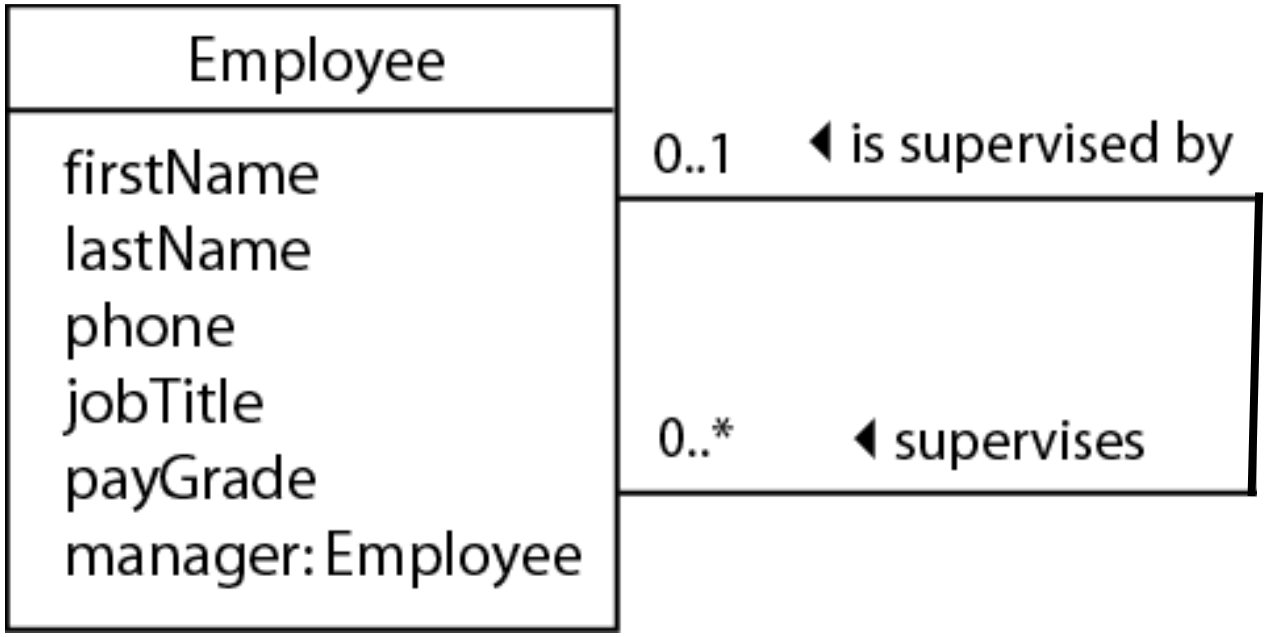
Správne

Príklad chyby návrhu: redundatná entita



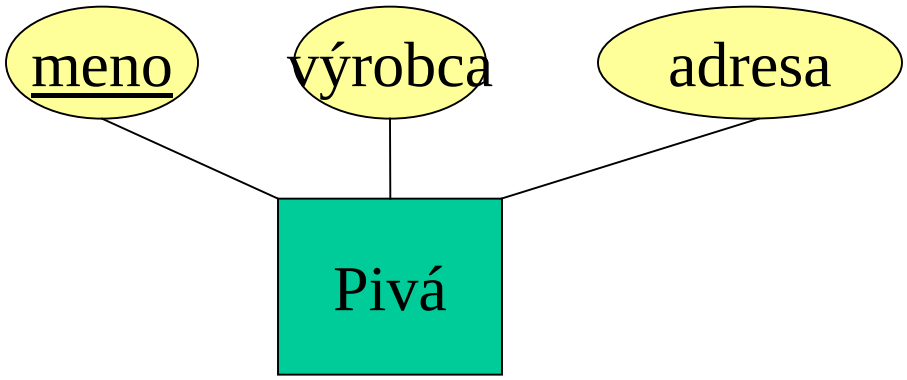
ZLE! Manager je tiež zamestnancom

Príklad chyby návrhu: redundantná entita



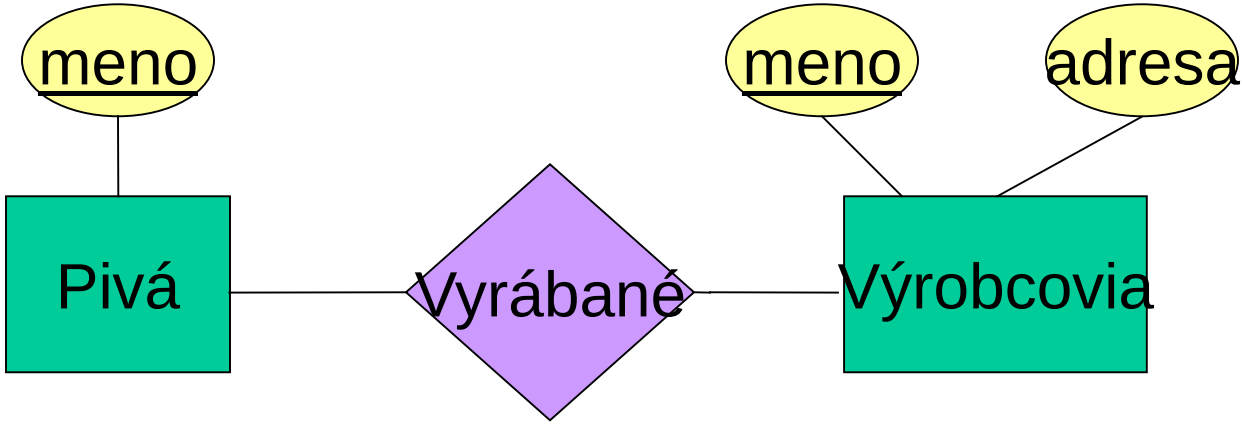
Správne

Príklad chyby návrhu: redundantné data



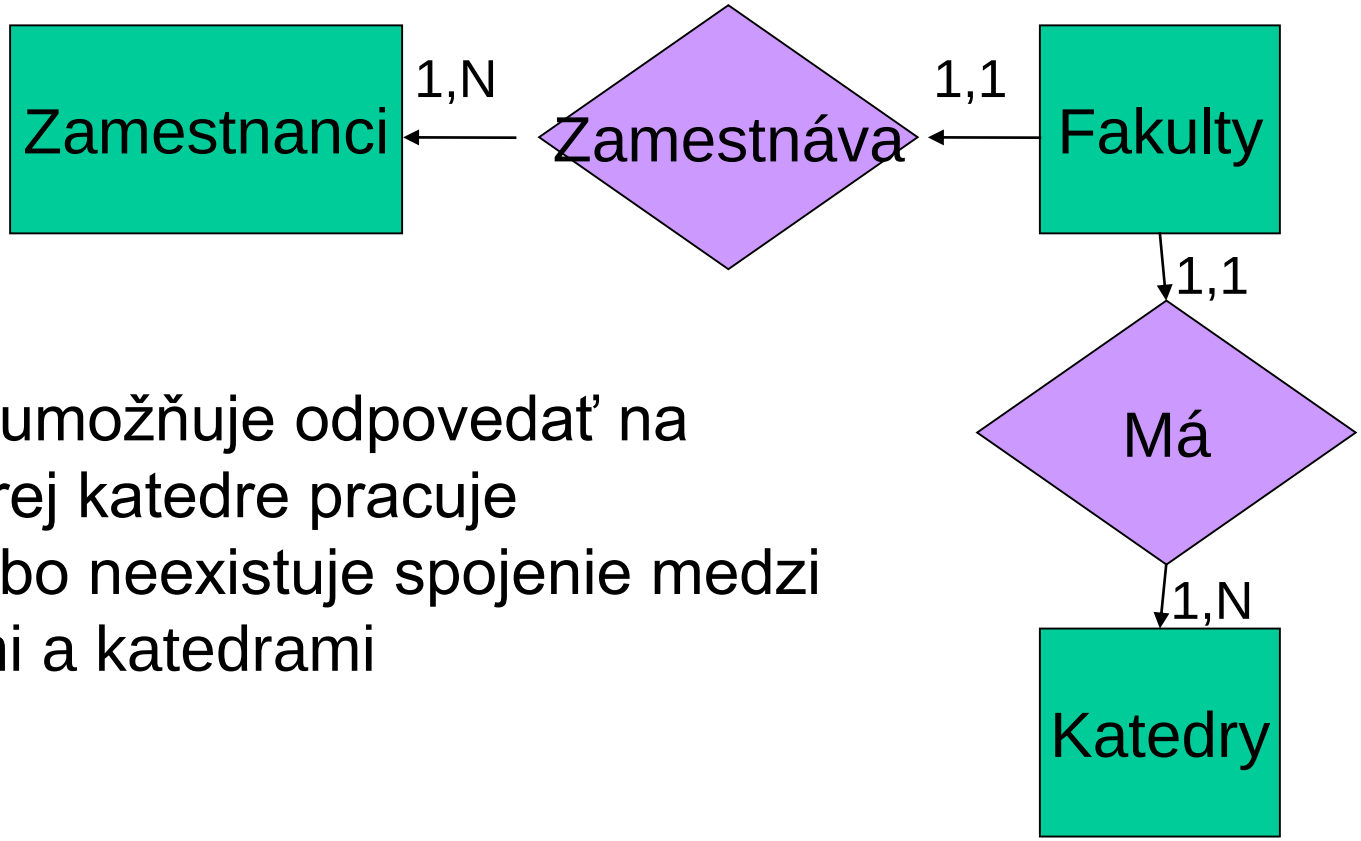
ZLE! Pre každé pivo, ktoré vyrába ten istý výrobca, sa opakuje adresa toho výrobcu. Ešte horšie je, že ak nejaký výrobca prestane na chvíľu vyrábať pivo, tak stratíme jeho adresu

Príklad chyby návrhu: redundantné data



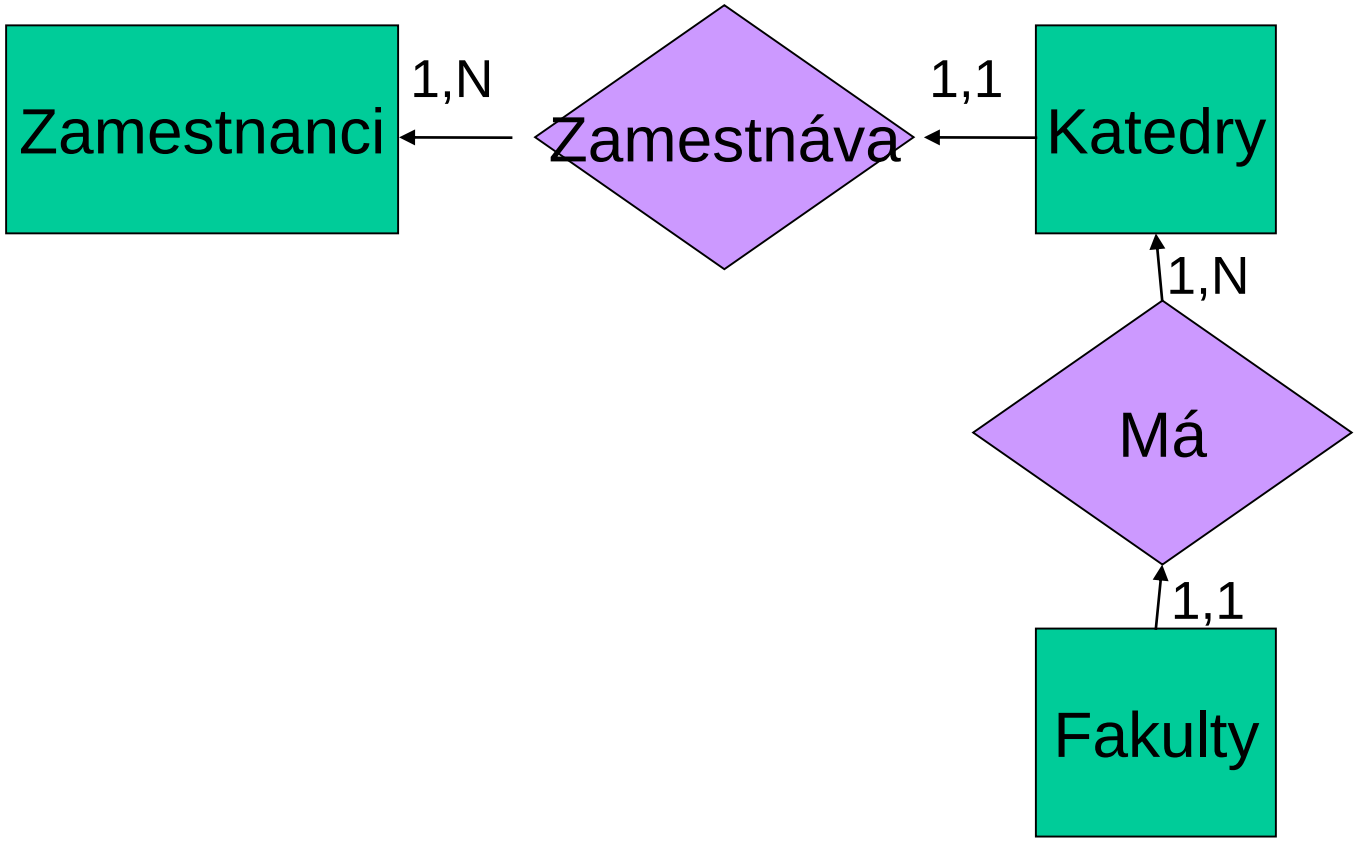
Správne

Príklad chyby návrhu: fan trap (vejár)



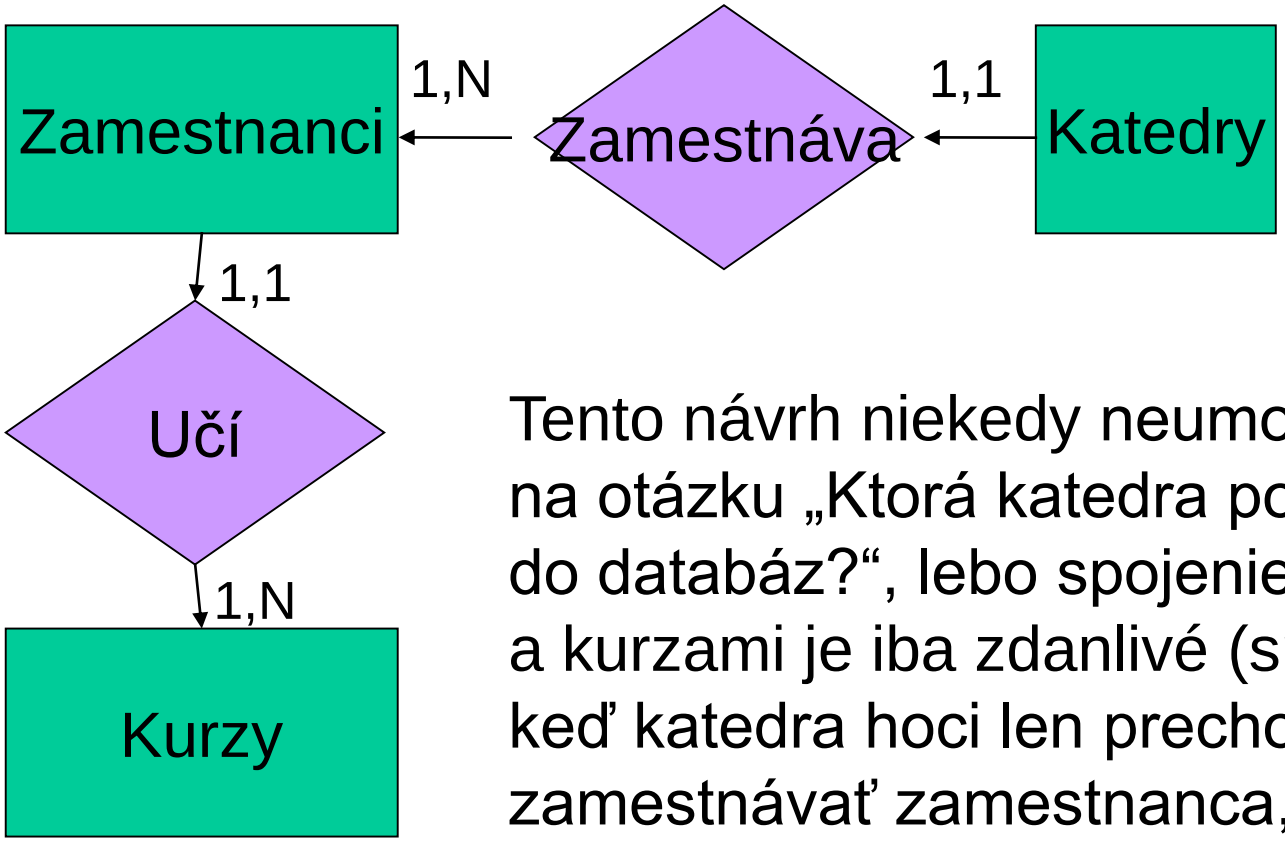
Tento návrh neumožňuje odpovedať na otázku „Na ktorej katedre pracuje Plachetka?“, lebo neexistuje spojenie medzi zamestnancami a katedrami

Príklad chyby návrhu: fan trap (vejár)



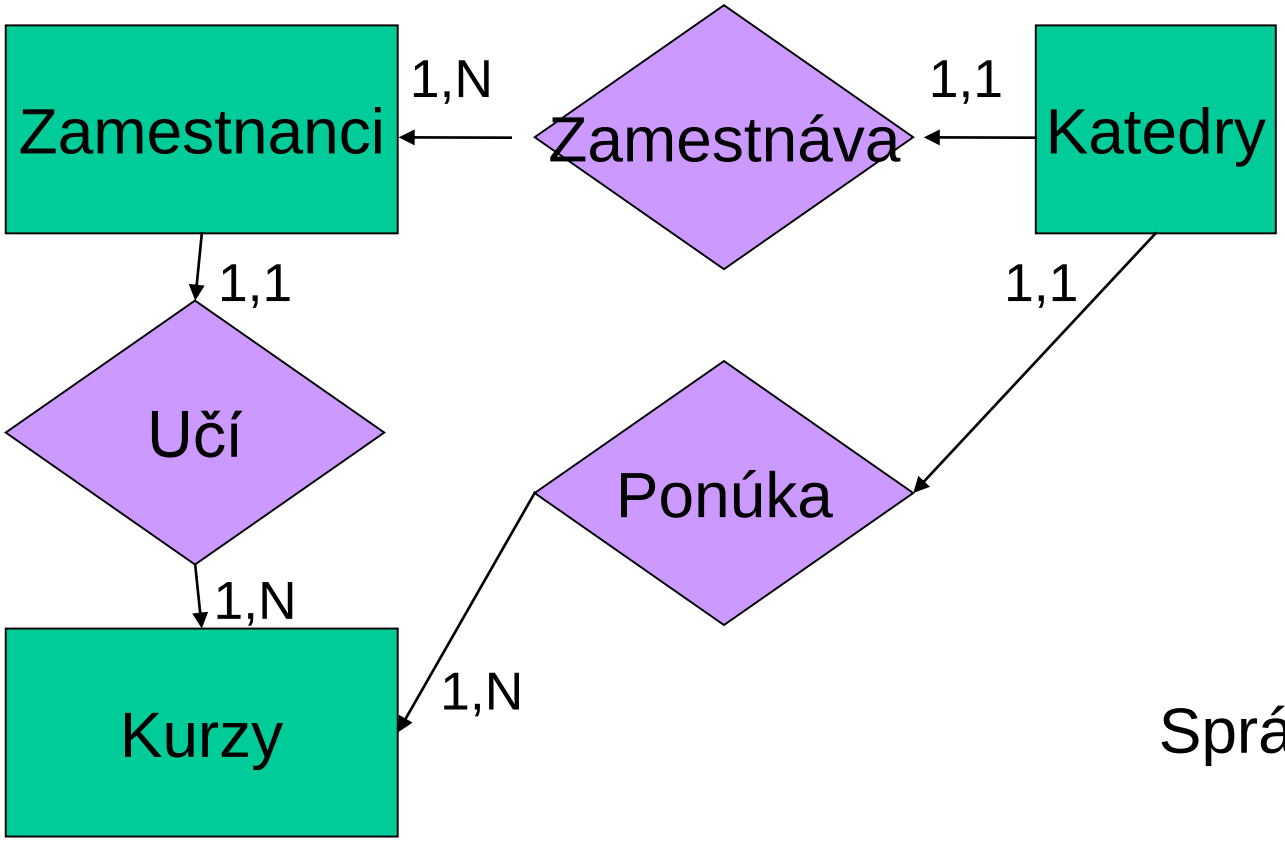
Správne

Príklad chyby návrhu: chasm trap (priepasť)



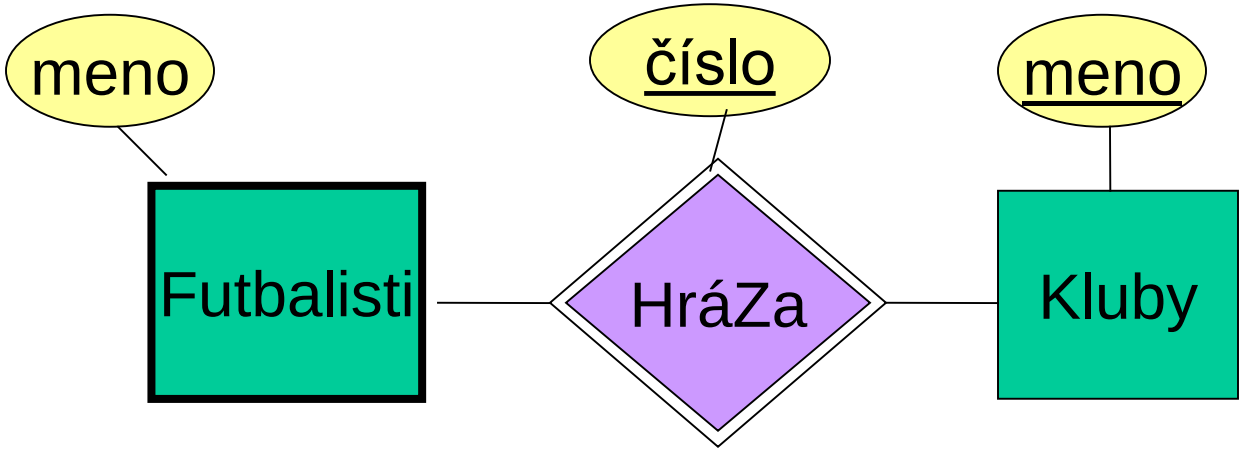
Tento návrh niekedy neumožňuje odpovedať na otázku „Ktorá katedra ponúka kurz Úvod do databáz?“, lebo spojenie medzi katedrami a kurzami je iba zdanlivé (spojenie sa stratí keď katedra hoci len prechodne prestane zamestnávať zamestnanca, ktorý ten kurz práve teraz učí)

Príklad chyby návrhu: chasm trap (priepasť)



Správne

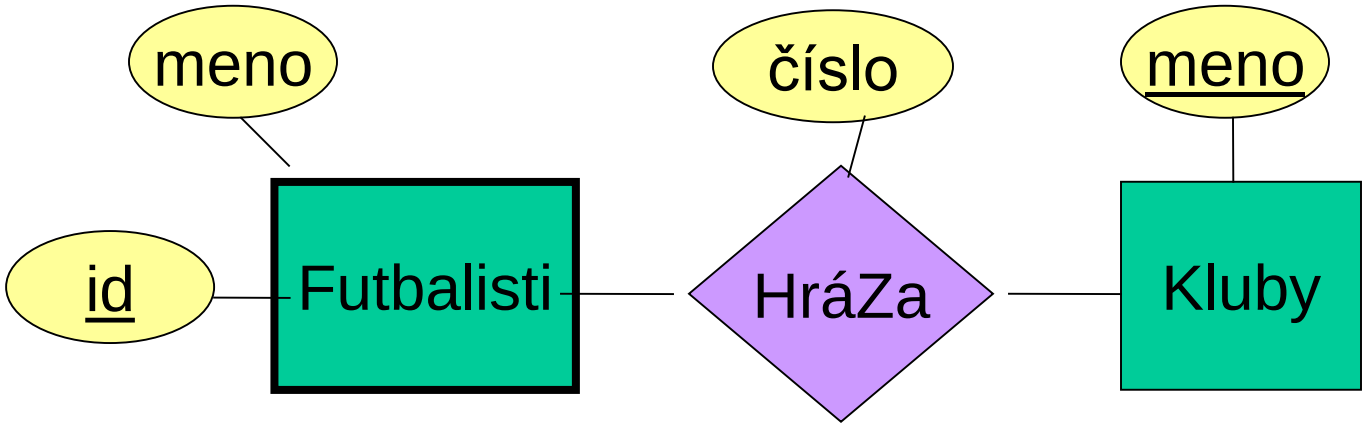
Príklad chyby návrhu: slabé entity sets



Ak kluby nevznikajú a nezanikajú príliš často, použitie slabej entity set sa môže zdať celkom vhodné. Neexistenciu klubu vieme vyjadriť dodatočným atribútom, ak chceme zachovať údaje o hráčoch.

Avšak tento návrh umožňuje len aktuálny pohľad. Ak futbalista prestúpi do iného klubu, tak stratíme informáciu o jeho pôvodnom klube. Vtedy sa zmení **identita futbalistu** (hoci je to stále ten istý chlap). Ak na tom záleží, tak je rozumnejšie futbalistu identifikovať nezávisle na klube (napríklad umelým id)

Príklad chyby návrhu: slabé entity sets



Ani použitie surrogate kľúča nerieši všetky problémy. Ak totiž hľadáme konkrétneho futbalistu, potrebujeme poznať jeho id. To id je síce jednoznačné, ale **odkiaľ sa dozvieme jeho hodnotu**? Toto je dôvod, prečo vzniká potreba **rozšíriť reálny svet o autoritu**, ktorá priraďuje futbalistom jednoznačné identifikátory, napríklad na základe rodných čísiel—takou autoritou je **futbalový zväz**. Futbalový zväz nesmie zverejniť rodné čísla, no v prípade nejednoznačného mena vie futbalový zväz rozlíšiť dvoch futbalistov podľa ďalších znakov (napr. podľa klubovej histórie alebo rodného čísla)

Príklad chyby návrhu: nevhodný primárny kľúč

Príklad nevhodného primárneho kľúča: rodné číslo, PSČ

Výber primárneho kľúča treba starostlivo zvážiť. Často je vhodné použiť surrogate, aj keď prirodzený externý kľúč existuje. Dôvod: **databázový systém nedovoľuje modifikovať primárny kľúč**

Možnosti výberu primárneho kľúča:

- externý kľúč: napr. rodné číslo (ale pozor, ak hrozí omyl pri vkladaní rodného čísla do databázy, tak radšej surrogate key)
- surrogate key: generovaný automaticky databázovým systémom
- substitute primary key: jeden atribút, obvykle skratka (napr. 3-písmenové skratky letísk)

Návrhové vzory

- Oplatí sa poznať dobré riešenia typických situácií
- Zdroje:
  - literatúra
  - vlastné skúsenosti
  - reverzné inžinierstvo fungujúcich systémov

  Napríklad je zaujímavé preštudovať formát GEDCOM, ktorý je v súčasnosti štandardom na uchovávanie genealogických dát
- Metódy v koncepčnej fáze návrhu sú iba **poloformálne**.

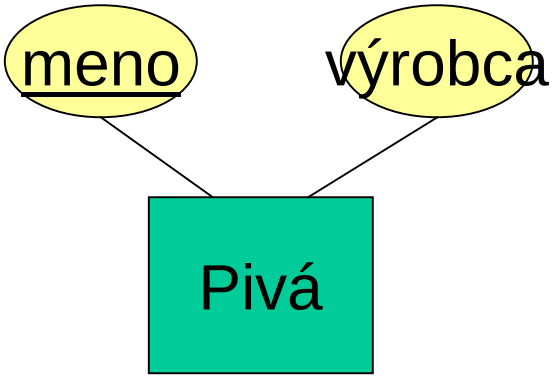
Niekedy je ťažké rozoznať dôležitý poznatok od (zbytočnej?) byrokracie. Lenže taký je reálny svet

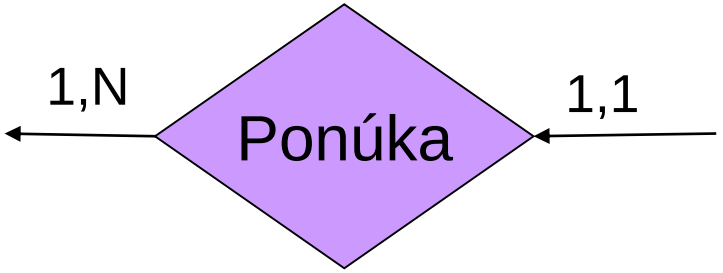
Od diagramov k SQL: Data Definition Language (DDL)

Data Definition Language, DDL, slúži na vytvorenie databázy. Preklad ER diagramov do SQL chápať ako prechod od koncepčného návrhu k logickému návrhu

- **Entity sets sa prekladajú ako tabuľky**
    - V prípade akýchkoľvek pochybností je vhodným primárnym kľúčom surrogate key (*WITH OIDS* alebo *SEQUENCE*)
- **Binárne vzťahy sa prekladajú do binárnych tabuliek** (v prípade many-to-many sú kľúčom oba atribúty, v prípade many-to-one je kľúč na strane one). V prípade vzťahu many-to-one je možné binárnu reláciu nahradiť importovaním cudzieho kľúča do podriadenej (slabej) relácie
- **N-árne vzťahy sa prekladajú do n-árnych tabuliek**

Preklad entity sets a vzťahov do SQL





1.Vytvorenie typov atribútov (SQL-99):
CREATE DOMAIN *name* [**AS**] *data_type*
[**DEFAULT** *expression*]
[*constraint* [...]]

where *constraint* is:
[**CONSTRAINT** *constraint_name*]
{ NOT NULL | NULL | **CHECK** (*expression*) }

Preklad entity sets a vzťahov do SQL

2.Vytvorenie tabuľky:

CREATE [[**GLOBAL** | **LOCAL**] { **TEMPORARY** | **TEMP** }]
TABLE *table_name* (
{ *column_name data_type* [**DEFAULT** *default_expr*] [*column_constraint* [...]]
| *table_constraint*
| **LIKE** *parent_table* [{ **INCLUDING** | **EXCLUDING** } **DEFAULTS**]
} [, ...]
)
[**INHERITS** (*parent_table* [, ...])]
[**WITH OIDS** | **WITHOUT OIDS**]
[**ON COMMIT** { **PRESERVE ROWS** | **DELETE ROWS** | **DROP** }]

OIDS znamená Object Identifiers, t.j „WITH OIDS“ znamená „vytvor surrogate primary key“ (dá sa použiť aj "sequence", t.j. automatický counter)

Preklad entity sets a vzťahov do SQL

3.Constraints na stĺpce:

[**CONSTRAINT** *constraint_name*]
{ NOT NULL | NULL | **UNIQUE** | **PRIMARY KEY** |
CHECK (*expression*) |
REFERENCES *reftable* [(*refcolumn*)] [MATCH FULL | MATCH PARTIAL | MATCH SIMPLE]
[**ON DELETE** *action*] [**ON UPDATE** *action*] }
[**DEFERRABLE** | **NOT DEFERRABLE**] [**INITIALLY DEFERRED** | **INITIALLY IMMEDIATE**]

action :: =
NO ACTION | **SET** NULL | **SET** DEFAULT | **CASCADE**

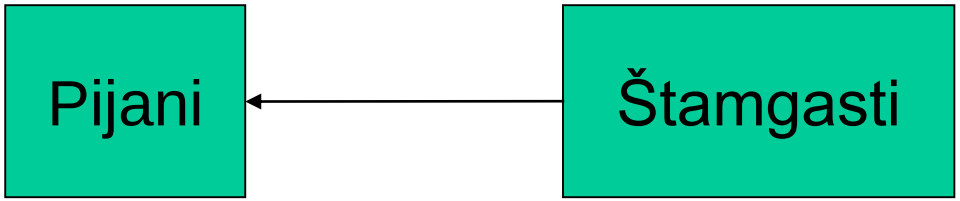
Preklad entity sets a vzťahov do SQL

4.Constraints na celú tabuľku:

[**CONSTRAINT** *constraint_name*]
{ **UNIQUE** (*column_name* [, ...]) |
PRIMARY KEY (*column_name* [, ...]) |
CHECK (*expression*) |
FOREIGN KEY (*column_name* [, ...]) **REFERENCES** *reftable* [(*refcolumn* [, ...])]
[MATCH FULL | MATCH PARTIAL | MATCH SIMPLE] [**ON DELETE** *action*] [**ON UPDATE** *action* [**DEFERRABLE** | **NOT DEFERRABLE**] [**INITIALLY DEFERRED** | **INITIALLY IMMEDIATE**]

action :: =
NO ACTION | **SET** NULL | **SET** DEFAULT | **CASCADE**

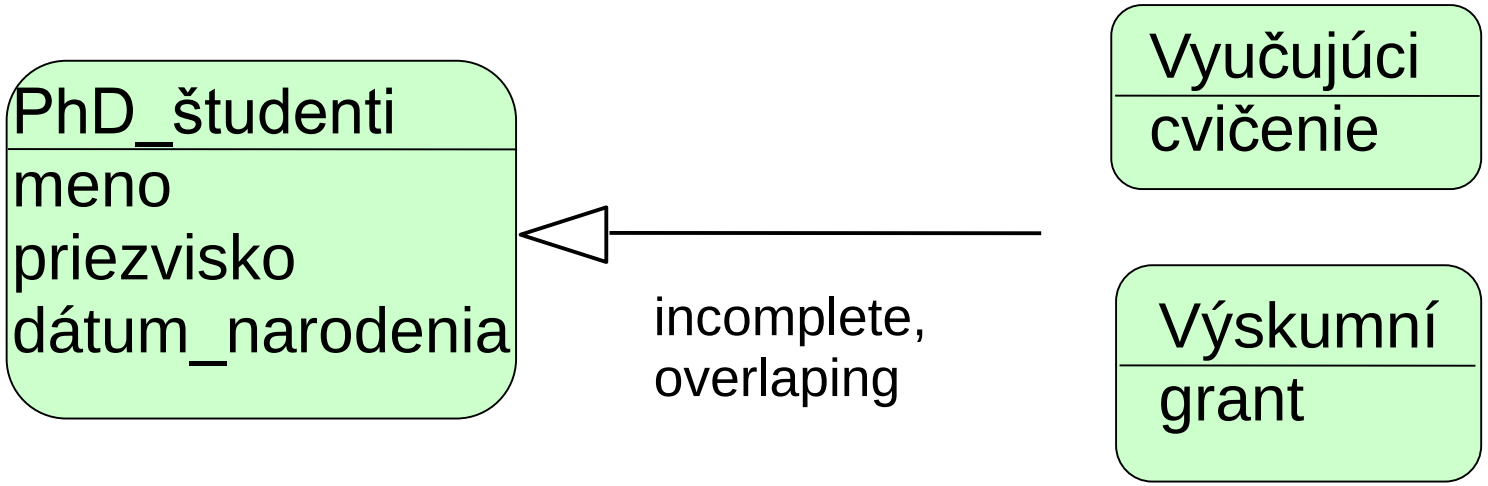
Preklad entity sets a vzťahov do SQL



Špecializácia sa preloží do samostatných tabuliek s rovnakým primárnym kľúčom (ktorý je uložený v oboch tabuľkách)

Preklad entity sets a vzťahov do SQL

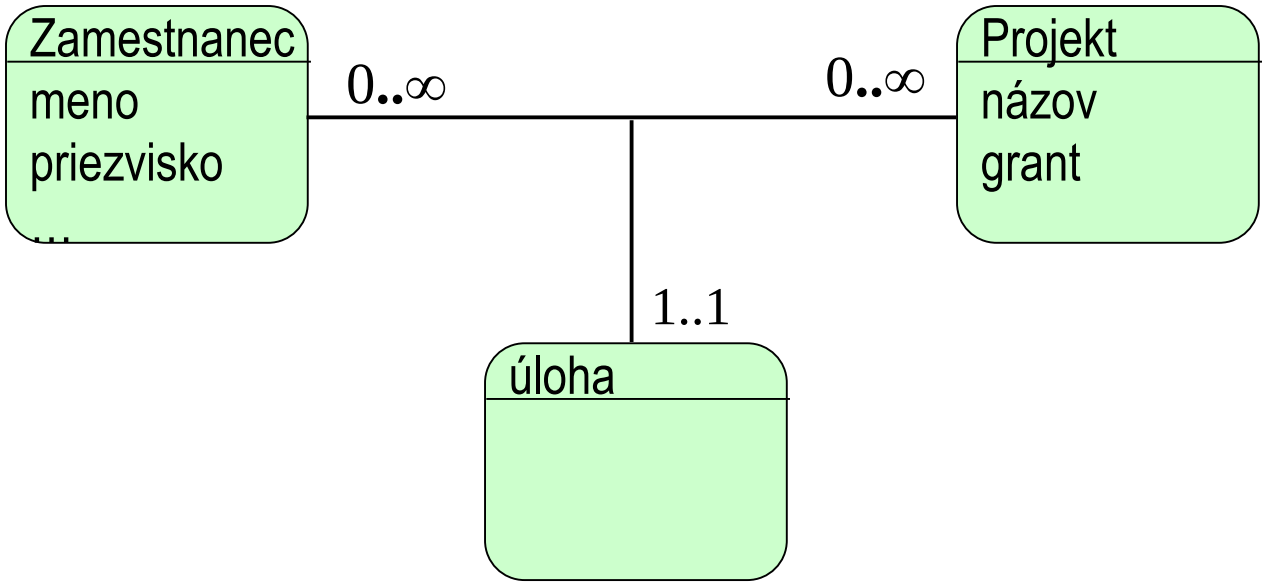
Príklad prekladu špecializácie




```
create table phd_studenti (Ids, Meno, Priezvisko,
  Datum_narodenia, constraint primary key (Ids));
create table vyucujuci (Ids, Cvicenie,
  primary key (Ids),
  check (exists (
  select * from phd_studenti phds where phds.Ids = Ids)));
create table vyskumni (Ids, Grant,
  primary key(Ids), check (exists
  (select * from phd_studenti phds where phds.Ids = Ids)));
```

Preklad entity sets a vzťahov do SQL

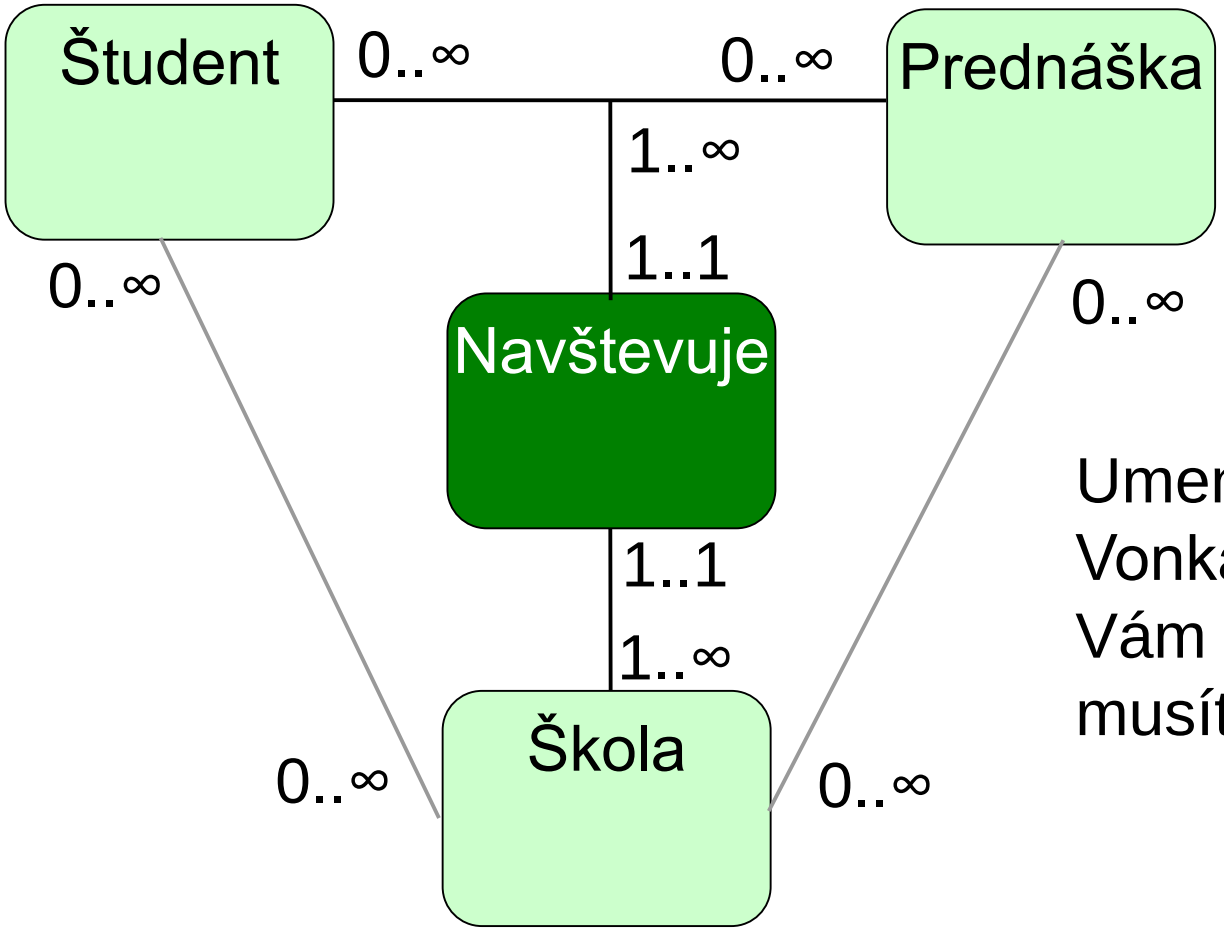
Príklad prekladu ternárneho vzťahu



create table zamestnanec(Idz, Meno, Priezvisko, ...,
  primary key (Idz));
create table projekt(Idp, Meno, Nazov, Grant, ...,
  primary key (Idp));
create table uloha(Idu, Idp, Idz, **primary key**(Idz, Idp, Idu),
  foreign key (Idz) **references** zamestnanec,
  foreign key (Idp) **references** projekt
  on delete cascade);

Preklad entity sets a vzťahov do SQL

Ešte jeden príklad prekladu ternárneho vzťahu

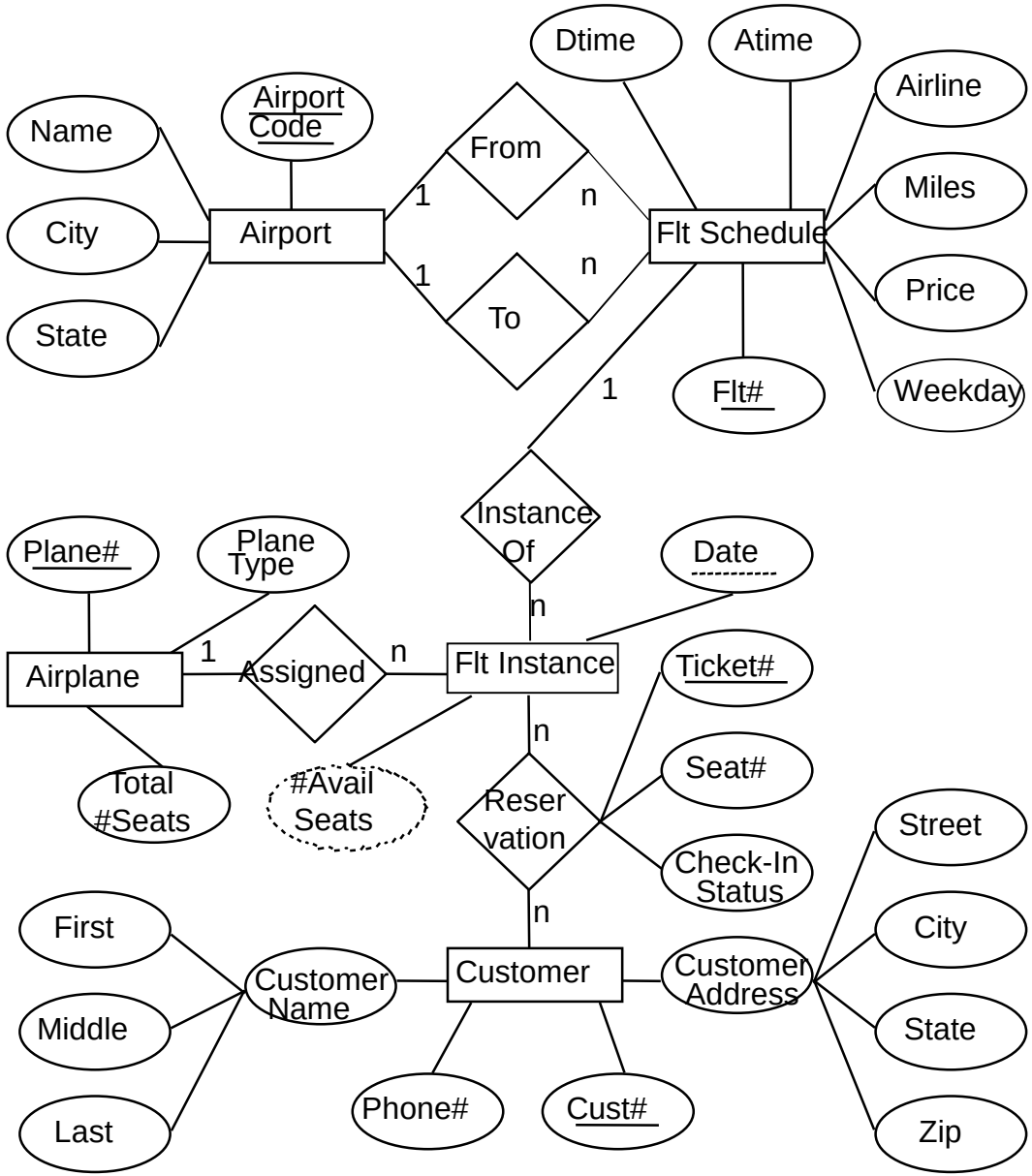


Umenie návrhu:
Vonkajší trojuholník
Vám hovoria, vnútro
musíte objaviť sami

Preklad entity sets a vzťahov do SQL

create table student (Ids, Rodne_cislo, Meno, Priezvisko, … ,
primary key(Ids));
create table prednaska (Idp, Nazov, … , **primary key**(Idp));
create table skola (ICO, … , **primary key**(ICO));
create table navštevuje (Ids, Idp, ICO,
primary key(Ids, Idp, ICO),
foreign key (Ids) references student,
foreign key (Idp) references prednaska,
foreign key (ICO) references skola);

Identifikácia funkčných závislostí (Lee Mark)



Funkčné závislosti v ER-diagrame

AIRPORT ↔ Airportcode

FLT-SCHEDULE ↔ Flt#

FLT-INSTANCE ↔ (Flt#, Date)

CUSTOMER ↔ Cust#

RESERVATION ↔ (Cust#, Flt#, Date)

RESERVATION ↔ Ticket#

AIRPLANE ↔ Plane#

Airportcode → Name, City, State

Flt# → Airline, Dtime, Atime, Miles, Price, (From) Airportcode, (To) Airportcode

(Flt#, Date) → Flt#, Date, Plane#

(Cust#, Flt#, Date) →Cust#, Flt#, Date, Ticket#, Seat#, CheckInStatus

Ticket# → Cust#, Flt#, Date, Ticket#, Seat#, CheckInStatus

Cust# → CustomerName, CustomerAddress, Phone#