Transformace konceptuálního modelu na relační

Michal Valenta

Katedra softwarového inženýrství
Fakulta informačních technologií
České vysoké učení technické v Praze

BI-DBS, LS 2021/2022

https://courses.fit.cvut.cz/BI-DBS/

CREATE TABLE

- CREATE TABLE

```
CREATE TABLE tabulka (
sloupec datovy_typ [io_sloupce [, io_sloupce...]],
...
[io_tabulky [, io_tabulky ...]] );
```

```
CREATE TABLE VYPUJCKY (
c_kopie CHAR (3) NOT NULL,
c_zak   CHARACTER (6) NOT NULL,
cena    DECIMAL(5,2),
rod_c   CHARACTER (10) NOT NULL,
datum_v DATE);
```

ALTER TABLE, DROP TABLE

- ALTER TABLE

```
ADD sloupec, DROP sloupec, ALTER sloupec,
ADD CONSTRAINT io, DROP CONSTRAINT io
```

```
ALTER TABLE KINA ADD pocet_mist INTEGER;
```

- DROP TABLE

```
DROP TABLE tabulka [CASCADE]
```

```
DROP TABLE KINA CASCADE;
```

Integritní omezení v SQL

- Integritní omezení sloupce:
  - NOT NULL
  - DEFAULT
  - UNIQUE
  - PRIMARY KEY
  - REFERENCES
  - CHECK
- Integritní omezení tabulky – stejné jako IO sloupce
  (NOT NULL je speciálním případem CHECK)
  složené IO vždy na úrovni tabulky
- Pojmenování IO
  není syntakticky nutné, ale vřele doporučované

Integritní omezení v SQL

```
DROP TABLE KINA CASCADE CONSTRAINTS;
CREATE TABLE KINA ...
...
CREATE TABLE PREDSTAVENI
(NAZEV_K Char Varying(20) NOT NULL,
NAZEV_F Character Varying(20) NOT NULL,
DATUM  date  NOT NULL,
CONSTRAINT PREDSTAVENI_PK
PRIMARY KEY (NAZEV_K, NAZEV_F),
CONSTRAINT PREDSTVENI_KINA_FK
FOREIGN KEY (NAZEV_K) REFERENCES KINA,
CONSTRAINT PREDSTAVENI_FILMY_FK
FOREIGN KEY (NAZEV_F) REFERENCES FILMY);
```

Referenční integrita, kaskádní reakce

Referenční integrita (cizí klíč) – v SQL čtyři možné způsoby reakce:

```
[ CONSTRAINT constraint_name ]
 FOREIGN KEY ( column_name [, ... ] )
 REFERENCES reftable [ ( refcolumn [, ... ] ) ]
 [ ON DELETE action ] [ ON UPDATE action ]
action ::= [NO ACTION | RESTRICT |
CASCADE | SET NULL | SET DEFAULT]
```

```
CREATE TABLE order_items (
product_no integer REFERENCES products
 ON DELETE RESTRICT,
order_id integer REFERENCES orders
 ON DELETE CASCADE,
quantity integer,
PRIMARY KEY (product_no, order_id));
```

Poznámka: implementace tohoto rysu nebývá kompletní.

Datové typy v SQL

- numerické
- textové
- datum a čas
- ... a mnoho dalších závisí na konkrétním RDBMS

poznámka

NULL je prvkem každého datového typu (pokud nespecifikujeme NOT NULL).
Tříhodnotová logika: TRUE, FALSE, UNKNOWN.
Konverze: implicitní, explicitní (pomocí funkce CAST).

Datové typy v PostgreSQL (přehledově)

Transformace schématu

- Algoritmus převodu konceptuálního schématu na relační bývá součástí modelovacích nástrojů (Oracle Data Modeller, Enterprise Architect,...).
- Pomocí nastavení je obvykle možné chování vestavěných generátorů výrazně upravit.
- Převod některých konstrukcí konceptuálního modelu (například ISA hierarchie) má několik možných variant, z nichž žádná není úplně přesná; optimální varianta pro konkrétní situaci závisí na dalších okolnostech (často používané operace, způsoby uložení dat, počet atributů v podtypech a nadtypu,...).
- V některých případech (povinnost nedeterminantu ve vztahu) nemáme na úrovni relačního modelu dostatečně efektivní mechanismus, který by kontrolu zajistil, proto se na kontrolu některých IO rezignuje.

Postup transformace

Entity

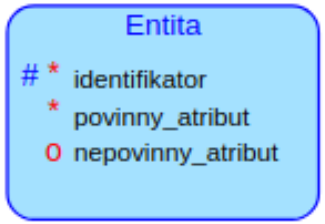
- název entity $\rightarrow$ název relace
- atributy entity $\rightarrow$ atributy relace
- domény atributů entity se “namapují“ na domény relačních atributů
- povinnost atributů entity $\rightarrow$ NOT NULL na relační úrovni
- atributy identifikátoru entity $\rightarrow$ PRIMARY KEY
- alternativní klíče $\rightarrow$ UNIQUE + NOT NULL
- u slabých entit je třeba do klíče přibrat identifikátory identifikačních vlastníků

Vztahy

- jediná možnost provázání dat ze dvou relací je referenční integrita (FOREIGN KEY)
- podle kardinality a parciality je třeba použít vztahové tabulky a integritní omezení NOT NULL a UNIQUE

Poznámka: na jeden atribut lze „uvalit“ i více než jedno IO (např. NOT NULL a UNIQUE nebo PRIMARY a FOREIGN key).

Transformace silné entity



Relační zápis (zjednodušené, používané i v testech)

entita(identifikator, *povinny_atribut, nepovinny_atribut)

Poznámka: atributy, které jsou součástí identifikátoru, jsou povinné, není potřeba před ně psát *.

SQL

```
CREATE TABLE entita (
identifikator INTEGER,
povinny_atribut VARCHAR(20) NOT NULL,
nepovinny_atribut VARCHAR(40),
CONSTRAINT entita_pk PRIMARY KEY (identifikator) );
```

Poznámka: Domény atributů na konceptuální úrovni nástroje obvykle vyžadují, obvykle se ale nezobrazují, aby schéma bylo přehlednější.

Vztah 1:1, obě entity povinná účast

Relační zápis

zamestnanec_vuz(cislo_z, *jmeno_z, adresa, *spz, *vyrobce, model)

SQL

```
CREATE TABLE zamestnanec_vuz (
cislo_z integer CONSTRAINT zamestnantec_vuz_pk PRIMARY KEY,
jneno_z varchar(20) NOT NULL,
adresa varchar(40),
spz varchar(20) NOT NULL CONSTRAINT zamestnantec_vuz_uk UNIQUE,
vyrobce varchar (40) NOT NULL,
model varchar(40));
```

Poznámka: vztah 1:1 z obou stran povinný není v praxi příliš častý. Vzhledem k IO je jedna tabulka relevantní. Atribut spz je alternativním klíčem (v relačním zápise není vyznačeno).

Vztah 1:1, jedna entita povinná účast

Relační schéma

zamestnanec(cislo_z, *jmeno_z, adresa)
vuz(spz, *vyrobce, model, *cislo_z)

vuz[cislo_z] $\subseteq$ zamestnanec[cislo_z]

SQL:

```
CREATE TABLE zamestnanec (
cislo_z integer CONSTRAINT zamestnantec_pk PRIMARY KEY,
jneno_z varchar(20) NOT NULL
adresa varchar(40));

CREATE TABLE vuz (
spz varchar(20) CONSTRAINT vuz_pk PRIMARY KEY,
vyrobce(40) NOT NULL,
model varchar(40),
cislo_z integer NOT NULL UNIQUE,
CONSTRAINT zamestnanec_vuz_fk
        FOREIGN KEY (cislo_z) REFERENCES zamestnantec(cislo_z));
```

Poznámka: alter. klíč cislo_z v relaci vuz opět není ve zjednodušeném relačním zápisu uveden.

Vztah 1:1, nepovinná účast

První možnost

Jako v předchozím případě. Jediný rozdíl je ten, že atribut cislo_z v tabulce vuz bude nepovinný.

Poznámka: atribut cislo_z již není alernativním klíčem (UNIQUE + NOT NULL), ale stále (vzhledem ke kardinalitě 1:1) by měl být UNIQUE.
Poznámka 2: implementace relačních DB jsou v tomto opět nejednotné (PostgreSQL: pouze jedna NULL hodnota, Oracle: může být více NULL hodnot, ikdyž je atribut UNIQUE).

Vztah 1:1, nepovinná účast

Druhá možnost

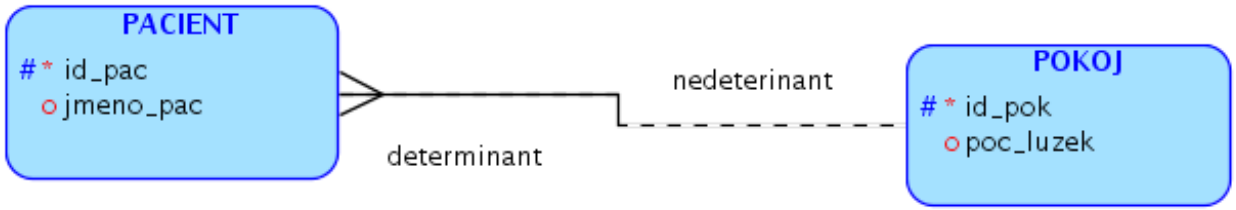
zamestnanec(cislo_z, *jmeno_z, adresa)
vuz(spz, *vyrobce, model)
zamestnanec_vuz (cislo_z, *spz)

zamestnanec_vuz[spz] $\subseteq$ vuz[spz]
zamestnanec_vuz[cislo_z] $\subseteq$ zamestnanec[cislo_z]
spz v relaci zamestnanec_vuz je NOT NULL a UNIQUE

SQL:
CREATE TABLE vuz (....); CREATE TABLE zamestnanec (...);
CREATE TABLE zamestnanec_vuz (
cislo_z integer PRIMARY KEY REFERENCES zamestnanec (cislo_z),
spz varchar(20) NOT NULL UNIQUE REFERENCES vuz (spz));

Poznámka: Stejně jako v předchozích příkladech implementace vztahu 1:1 je i zde atribut spz alternativním klíčem v tabulce zamestnanec_vuz.

Vztah 1:N, povinná účast determinantu



Relační zápis

pacient(id_pac, jmeno_pac, *id_pok)
pokoj(id_pok, poc_luzek)

pacient[id_pok] $\subseteq$ pokoj[id_pok]
id_pok v relaci pacient NOT NULL

SQL

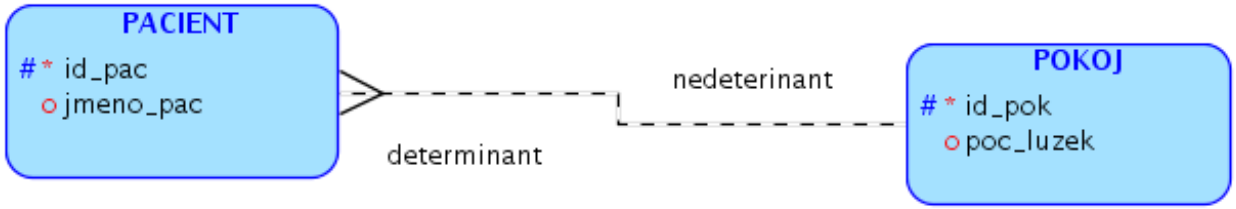
```
CREATE TABLE pacient (
id_pac integer CONSTRAINT pacient_pk PRIMARY KEY,
jmeno_pac varchar(20), id_pokoj integer NOT NULL);

CREATE TABLE pokoj (
id_pok integer CONSTRAINT pokoj_pk PRIMARY KEY, poc_luzek integer);

ALTER TABLE pacient ADD CONSTRAINT pacient_pokoj_fk
            FOREIGN KEY (id_pok) REFERENCES pokoj(id_pok));
```

Poznámka: Informaci o parcialitě nedeterminantu ztrácíme.

Vztah 1:N, nepovinná účast determinantu



Varianta 1 (2 tabulky)

Jako v předchozím případě. Rozdíl je ten, že atribut id_pok v tabulce pacient bude nepovinný.

Varianta 2 (3 tabulky)

pacient(id_pac, jmeno_pac)
pokoj(id_pok, poc_luzek)
umisteni (id_pac, *id_pok)

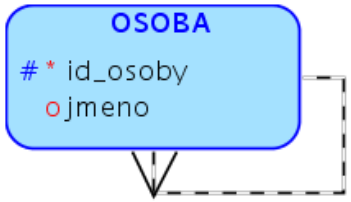
umisteni[id_pac] $\subseteq$ pacient[id_pac]
umisteni[id_pok] $\subseteq$ pokoj[id_pok]
id_pok v relaci umisteni NOT NULL

SQL:

```
CREATE TABLE pacient (...); CREATE TABLE pokoj (...);
CREATE TABLE umisteni (
id_pac integer PRIMARY KEY REFERENCES pacient (id_pac),
id_pokoj integer NOT NULL REFERENCES pokoj (id_pok));
```

Poznámka: Informaci o parcialitě nedeterminantu opět ztrácíme.
Poznámka 2: Ve variantě 2 je v tabulce umístění atribut id_pok povinný správně, jinak nemá smysl n-tici (řádek) vytvářet. Atribut id_pok zde však už nemůže být (alternativním) klíčem!

Rekurzivní vztah



Relační zápis

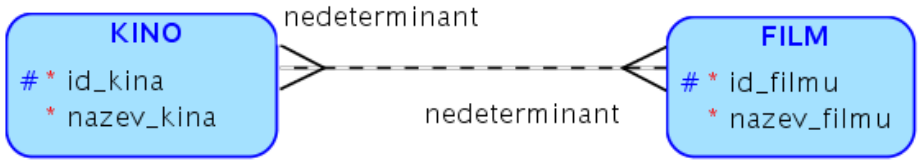
osoba(id_osoby, jmeno, manager_id)
osoba[manager_id] $\subseteq$ osoba[id_osoby]

SQL

```
CREATE TABLE osoba (
id_osoby integer PRIMARY KEY,
jmeno varchar(30),
manager_id integer);

ALTER TABLE osoba
ADD CONSTRAINT osoba_manager_fk
        FOREIGN KEY (manager_id) REFERENCES osoba (id_osoby));
```

Vztah M:N



Relační zápis

kino(id_kina, *nazev_kina)
film(id_filmu, *nazev_filmu)
predstaveni(id_kina, id_filmu)

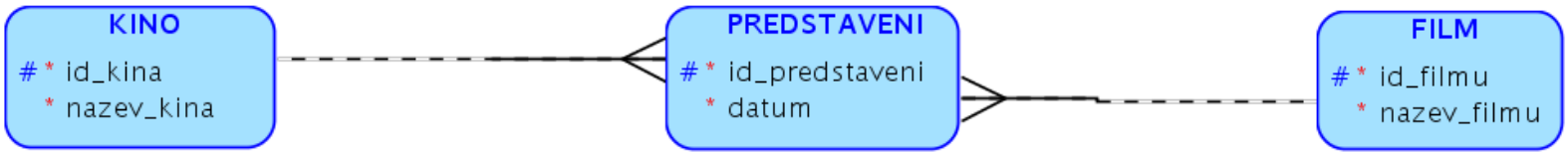
predstaveni[id_kina] $\subseteq$ kino[id_kina]
predstaveni[id_filmu] $\subseteq$ film[id_filmu]
M:N vždy pomocí vztahové tabulky.

SQL

```
CREATE TABLE kino (....); CREATE TABLE film (...);
CREATE TABLE predstaveni (
id_kina integer REFERENCES kino (id_kina),
id_filmu integer REFERENCES film (id_filmu),
CONSTRAINT predstaveni_pk PRIMARY KEY (id_kina,id_filmu));
```

Poznámka 1: Informaci o parcialitě nedeterminantů opět ztrácíme.
Poznámka 2: Jeden film v jenom kině nejvýše jednou? Zřejmě vede na dekompozici.

Dekompozice vztahu M:N – silná entita



Relační zápis

kino(id_kina, *nazev_kina)
film(id_filmu, *nazev_filmu)
predstaveni(id_predstaveni, *datum, *id_kina, *id_filmu)

predstaveni[id_kina] $\subseteq$ kino[id_kina]
predstaveni[id_filmu] $\subseteq$ film[id_filmu]

SQL

```
CREATE TABLE kino (....); CREATE TABLE film (...);
CREATE TABLE predstaveni (
id_predstavení integer PRIMARY KEY,
datum date NOT NULL,
id_kina integer NOT NULL REFERENCES kino (id_kina),
id_filmu integer NOT NULL REFERENCES film (id_filmu));
```

Poznámka: atributy id_kina a id_filmu jsou v tabulce predstaveni opět povinné, protože bez nich by záznam neměl smysl.

Dekompozice vztahu M:N – slabá entita

Relační zápis

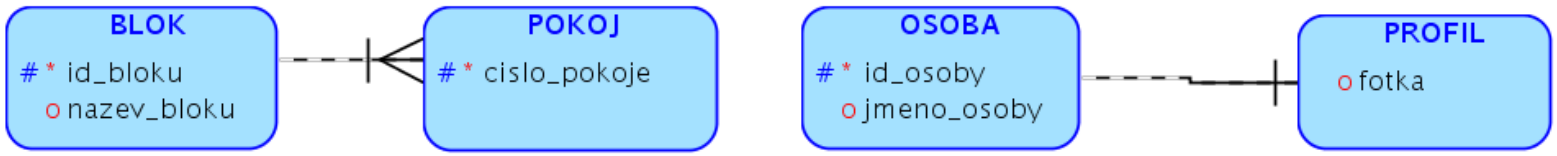
kino(id_kina, *nazev_kina)
film(id_filmu, *nazev_filmu)
predstaveni(datum, id_kina, id_filmu)

predstaveni[id_kina] $\subseteq$ kino[id_kina]
predstaveni[id_filmu] $\subseteq$ film[id_filmu]

SQL

```
CREATE TABLE kino (....); CREATE TABLE film (...);
CREATE TABLE predstaveni (
datum date,
id_kina integer NOT NULL REFERENCES kino (id_kina),
id_filmu integer NOT NULL REFERENCES film (id_filmu),
CONSTRAINT PRIMARY KEY predstaveni_pk (datum, id_kina, id_filmu));
```

Slabá entita, identifikační závislost



Relační zápis

blok(id_bloku, nazev_bloku)
pokoj(cislo_pokoje, id_bloku)
pokoj[id_bloku] $\subseteq$ blok[id_bloku]

osoba(id_osoby, jmeno_osoby)
profil(id_osoby, fotka)
profil[id_osoby] $\subseteq$ osoba[id_osoby]

SQL

```
CREATE TABLE blok (...);
CREATE TABLE pokoj (
cislo_pokoje integer, id_bloku integer
        REFERENCES blok (id_bloku),
PRIMARY KEY (id_bloku, cislo_pokoje));
```

```
CREATE TABLE osoba (...);
CREATE TABLE profil (
fotka blob,
id_osoby integer
        REFERENCES osoba (id_osoby),
PRIMARY KEY (id_osoby));
```

ISA hierarchie

Poznámka 1: ISA v BI-DBS chápeme jako „disjoint, complete“! Kontrola této vlastnosti není na slide uvedena. Jak na to?
Varianta 1: Lze použít deklarativní IO `CHECK`.
Varianty 2 a 3: Nutno řešit procedurálním IO.
Poznámka 2: Nástroje pro konc. modelování a generování SQL kódu požadavky typu „disjoint, complete"“ obvykle neřeší.
Poznámka 3: Varianta 2 (bez dalších IO) je vlastně implementace „rolí“. Viz příklad OSOBA, STUDENT, UCITEL v přednášce o E-R modelování.

varianta 1

osoba(id_osoby, *email, telefon, jmeno, ico, dic, *typ)
Je vhodné zavést rozlišovací atribut (typ), abychom rozlišili právnickou a fyzickou osobu.

varianta 2

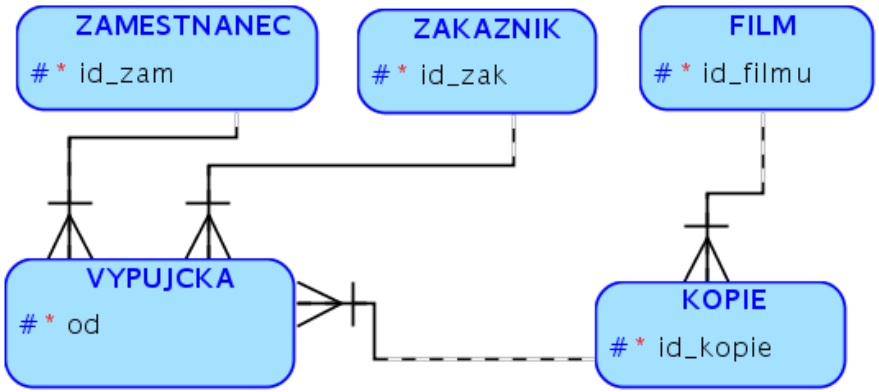
osoba(id_osoby, *email, telefon)
fyzicka(id_osoby, *jmeno)
pravnicka(id_osoby, *ico, *dic)

fyzicka[id_osoby] $\subseteq$ osoba[id_osoby]
pravnicka[id_osoby] $\subseteq$ osoba[id_osoby]

varianta 3

fyzicka(id_osoby, *email, telefon, *jmeno)
pravnicka(id_osoby,*email, telefon *ico, *dic)

Migrace klíčů, složený cizí klíč



Relační zápis

zakaznik(id_zak)
zamestnanec(id_zam)
film(id_filmu)
kopie(id_filmu, id_kopie)
vypujcka(od, id_zak, id_zam, id_filmu,id_kopie)
kopie[id_filmu] $\subseteq$ film[id_filmu]
vypujcka[id_zak] $\subseteq$ zakaznik[id_zak]
vypujcka[id_zam] $\subseteq$ zamestanec[id_zam]
vypujcka[id_filmu, id_kopie] $\subseteq$ kopie[id_filmu, id_kopie]

Poznámka: podobnou konstrukci najdete skoro v každé zkouškové písemce.

Smyčky

Relační zápis

zamestnanec(id_zam)
zvire(id_zvir, *krmi_id_zam, sponzoruje_id_zam)
zvire[krmi_id_zam] $\subseteq$ zamestnanec[id_zam]
zvire[sponzoruje_id_zam] $\subseteq$ zamestnanec[id_zam]

Poznámka 1: Diskuse smyček (ve vaší semestrálce). Může být sponzorem zvířete i zaměstnanec, který ho nekrmí? Pokud ano, je smyčka OK, protože každá „cesta“ má jiný význam.
Poznámka 2: Prefixy zavedeny, aby se dva atributy v jedné relaci nejmenovaly stejně. Takto to (preventivně) řeší SQLDeveloper. Důsledek: nemůžete pak používat `JOIN USING` nebo `NATURAL JOIN`. Při transformaci na SQL lze na úrovni relačního (žlutého) modelu změnit.

K zapamatování

- Konstrukty E-R (entity (silné, slabé) a vztahy(obecné, identifikační, výlučné, ISA) a jejich IO (identifikátory, povinnosti atributů, kardinality, parciality) vyjadřujeme v relačním modelu (potažmo v SQL) pomocí relací (tabulek) a kombinací jeho IO (`PRIMARY`, `UNIQUE`, `FOREIGN`, `NOT NULL`, `CHECK`).
- Syntaxe SQL umožňuje IO pojmenovat. V praxi je to užitečné.
- Ikdyž je (binární) E-R jednoduchá notace, máme při transformaci někdy více možností a někdy nejsme schopni transformovat úplně věrně (nějakou informaci ztratíme).
- Nástroje pro konc. modelování a gen. SQL jsou konfigurovatelné. Pro správné nastavení je nutné transformaci rozumět. Optimální varianta závisí i na způsobu dotazování DB. Je to „skill“(věda) :-).
- Smyčky (cykly) v konceptuálním schématu nemusí být špatně. Někdy se jim nelze vyhnout. Je třeba umět smyčku interpretovat a rozhodnout, jestli je opodstatněná. Pokud hrozí nekonsistence, lze doplnit další IO (v sem. pr. formou komentáře v E-R sch.).