DATOVÉ MODELOVÁNÍ METODOU HIT

(Minikurs pro analytiky informačních systémů)

Zdenko Staníček
SHINE studio s.r.o., organizační a informační inženýrství, řízení projektů
Minská 52, 616 00 Brno
tel.+fax: 05-41233458
e-mail: sta@shine.cz

OBSAH:

Poděkování:

Děkuji všem, kteří stáli u zrodu datového modelu HIT a metody HIT. Z nich jmenovitě chci poděkovat Pavlovi Maternovi, Jiřímu Zlatuškovi a Františkovi Krejčímu. Děkuji za to, že HIT existuje. Dobře se na tom vydělává. Tu možnost měli a mají všichni. Já jsem ji využil.

Většina příkladů uvedených v kapitole 3 tohoto minikursu je převzata z pracovních materiálů, které vytvořil František Krejčí.

Z.S.

0. ÚVOD

Ve všech úspěšných a uznávaných technologiích používaných ve světě pro budování řídících a informačních systémů se pro modelování datové základny na konceptuální/logické úrovni používá entitně-vztahový model. V podstatě vždy jde o nějakou modifikaci či rozšíření Chenova E-R modelu. S příslušnou modifikací entitně-vztahového modelu se pojí většinou metodika popisující konkrétní postup datového modelování.

V následujících odstavcích je vyložena metoda HIT, která z praktického hlediska rovněž patří mezi modifikace entitně-vztahového přístupu k modelování datové základny.

Opodstatnění modifikace představované metodou HIT lze stručně shrnout do následujících faktů:

- výsledný logický model datové základny, představovaný v HIT metodě tzv. C-schematem, jež obdržíme po transformaci schématu funkčních závislostí, je skutečně entitně-vztahovým modelem, poněvadž ke konstrukci popisu datové základny využívá objekty, vztahy a charakteristiky objektů resp. vztahů (atributy);
- výsledný logický model je v normalizovaném tvaru (minimálně čtvrtá normální forma je zajištěna);
- k normalizovanému entitně-vztahovému modelu datové základny dospějeme algoritmem transformace ze schématu funkčních závislostí;
- schéma funkčních závislostí popisuje nejpřirozenějším způsobem stav našeho poznání skutečnosti: zachycuje sémantiku přiřazení jedněch objektů jiným objektům resp. charakteristik (atributů) objektům;

Klasický postup modelování databáze prostředky entitně-vztahového modelu je následující:

- projektant rozliší na základě intuice a zkušenosti "objekty" ("entity"), přesněji řečeno jejich typy, "vztahy" mezi těmito objekty a "atributy" (charakteristiky) objektů resp. vztahů
- pomocí těchto tří základních stavebních kamenů popíše realitu
- nasadí některou z normalizačních procedur, aby obdržel logický model v normalizovaném tvaru; poznamenejme, že použití jakékoli normalizační procedury představuje detailní studium funkčních závislostí mezi klíči (tj. identifikacemi objektů resp. vztahů) a neklíčovými položkami;

Postup v modifikované verzi entitně-vztahového přístupu představované metodou HIT je následující:

- projektant začne studiem funkčních závislostí mezi objekty navzájem a mezi objekty a jejich charakteristikami
- rozpoznané funkční závisloti explicitně zaznamená včetně jejich sémantiky
- po vyčištění schématu funkčních závislostí od závislostí odvoditelných a rozložitelných provede algoritmus transformace, který je automatizovaný; výsledkem je logický model dat popsaný v normalizovaném tvaru.

Vzhledem k výrazné vypovídací schopnosti explicitně zaznamenaných funkčních závislostí, které umožňují řešitelskému týmu detailní a kompetentní rozmýšlení nad danou problematikou, a vzhledem k faktu, že studiu funkčních závislostí se nelze při normalizačním procesu vyhnout, dáváme přednost dále popsanému postupu, který vychází z metody HIT.

1. PŘIROZENÝ POSTUP POZNÁVÁNÍ

Chceme-li cokoliv poznat a porozumět tomu, musíme si do mysli nejprve vetknout **objekty**, ze kterých se poznávaná skutečnost skládá, potom pochopit **souvislosti** mezi těmito objekty a nakonec porozumět **chování** toho, co poznáváme.

Jiný postup poznání komplikuje, zkresluje a činí neefektivním.

Objektem je to, co dokážeme v mysli jednoznačně odlišit od všeho ostatního a čemu má smysl přiřadit nějaké charakteristiky (vlastnostií, atributy). Je-li něco objektem, pak je to v přirozeném jazyce vždy označeno podstatným jménem. Svět, který poznáváme, se skládá z objektů. V mysli si vytváříme rovněž objekty, z nichž některé v reálném světě neexistují. Příklad: kulatý čtverec. Přesněji řečeno odrazem objektu v mysli je pojem. Abychom se navzájem domluvili, musíme pojmům rozumět stejně. Proto je vhodné každý pojem definovat. Definice pojmu je zároveň i definicí pojmem označeného objektu.

Každý objekt (pojem) má tu vlastnost, že v daném stavu světa k němu existuje buď žádná, jedna nebo více instancí. Příklad: kulatý čtverec - žádná instance, pavilon C brněnského výstaviště - jedna instance, člověk - miliardy instancí, vše v daném stavu světa. Pojmy (objekty), ke kterým v libovolném stavu světa existuje nejvýše jedna instance jsou jednotliviny (individua). Všechny ostatní pojmy označují třídy (množiny) individuí. Objekty takovými pojmy označené jsou právě tyto třídy neboli množiny.

Abychom věci poznali, soustřeďujeme se na jejich vlastnosti čili atributy. Atribut je funkce neboli přiřazení: dosadím-li do takové funkce určitou instanci objektu, vrátí mi jako hodnotu nějakou vlastnost. Speciální vlastností objektů je, že nějak souvisejí s jinými objekty. To že dva objekty spolu souvisí, tj. že je mezi nimi vztah, lze tedy vyjádřit atributem: dosadíme-li do takového atributu konkrétní instanci objektu, vrátí nám jako hodnotu jednu nebo více instancí jiného nebo téhož objektu, které s touto instancí určitým konkrétním způsobem souvisí. Pro poznání a porozumění nějaké oblasti jsou právě tyto atributy nejdůležitější.

Poznat něco znamená vidět souvislosti a rozumět jim. Znalost souvislostí nám většinou napovídá o možném chování toho, čeho souvislosti známe. Proto popis chování děláme až nakonec, na dokreslení celého obrazu.

2. PŘEHLED POJMŮ METODY HIT PRO ETAPU MAPOVÁNÍ

2.1 Základní typy, sortalizace, funkční závislosti

Objekty našeho zájmu, o kterých bude budovaný IS vypovídat, resp. s kterými bude manipulovat, jsou např. zaměstnanci, organizace, výrobky, územní jednotky,... ale i formuláře, úkoly, etapy řešení, ukazatelé, apod. Každý objekt je popsán řadou nějakých charakteristik, které jsou u každého konkrétního objektu vyjádřeny nějakými hodnotami (numerickými resp. textovými), např. číslo zaměstnance, cena výrobku, termín etapy, plošný výměr územní jednotky apod. Základním principem metody HIT je funkcionální přístup a tedy základním pojmem je pojem funkce, čili přiřazení. Objektům přiřazujeme jiné objekty (které jsou s nimi v nějakém vztahu) nebo jejich charakteristiky, charakteristikám můžeme přiřazovat objekty. Přiřazovat charakteristikám jiné charakteristiky nemá z pragmatického hlediska smysl. To, čemu jsou přiřazeny alespoň dvě různé charakteristiky, je v našem pojetí vždy objektem.

DEF 1. Množiny, které se vyskytují jako definiční obory nebo obory hodnot ve funkcích, se kterými budeme dále pracovat, nazýváme základní typy. Množiny, jejichž prvky jsou hodnoty nějakých charakteristik, tj. množiny rekursivně vyčíslitelné, jejichž hodnoty jsou vyjádřeny alfanumerickými řetězci, nazýváme popisné typy.. Množiny, jejichž prvky jsou nějaké objekty ("stejného druhu"), tj. množiny vymezené nějakou vlastností a tedy rekursivně nevyčíslitelné, nazýváme objektové typy. Popisné a objektové typy nazýváme společným názvem uzlové typy. Kartézský součin (tj. množina všech možných n-tic) utvořený z uzlových typů se nazývá n-ticový typ. N-ticový typ, který je kartézským součinem výhradně popisných typů, nazýváme popisný n-ticový typ.

Pozn.: Ekvivalentní termín k "popisný typ" je deskriptivní sorta, a k "objektový typ" - entitní sorta.

Problém správné volby základních typů, tj. tzv. **problém sortalizace**, patří právě do etapy mapování reality.

Množinu BOOL = (TRUE, FALSE), kde TRUE značí "pravda" a FALSE značí "nepravda", budeme v dalším pokládat za popisný typ. Typ BOOL nemůže být součástí n-ticového typu.

Funkce, se kterými v HITu pracujeme, vyjadřují empirická fakta rozpoznaná v realitě. Proto jsou tyto funkce vždy závislé na stavu světa (možném světě). Tato skutečnost je v etapě mapování reality podstatná. Poněvadž rozlišení funkcí vyjadřujících empirická fakta od matematických / logických funkcí bude vždy zřejmé, nebudeme explicitní odkaz na závislost funkcí na stavu světa uvádět.

Ze třídy všech možných funkcí, které lze definovat nad základními typy, se v HITu z pragmatických důvodů omezujeme na určitou podtřídu, která je postačující pro popis jakýchkoli (empirických) faktů z reality.

Než přistoupíme k definici, zaveďme následující konvence. Základní typy budeme označovat písmeny T, T1,..., atd., pokud nebudeme rozlišovat mezi jednotlivými druhy základních typů. Písmeny E, E1,..., atd. budeme obecně označovat objektové typy a písmeny D,D1,... budeme značit popisné (deskriptivní) typy.

Jestliže z uzlových typů T1,...,Tn vytvoříme n-ticový typ, pak jej značíme symbolem (T1,...,Tn). Ve funkcích, které budeme v dalším používat, vystupují proměnné. Za proměnnou je vždy možné dosazovat prvky, ale vždy jen prvky určitého typu - právě toho typu, jakého je proměnná. Proměnné značíme x,y,...,atd. a prvky ze základních typů, které můžeme za proměnné dosazovat, značíme x, y, a, b, ... Fakt, že proměnná x resp. prvek x je typu T označujeme x/T resp. x/T.

Jestliže funkce **f** přiřazuje prvkům typu T1 prvky typu T2 říkáme, že **f** je funkce typu (T1 ---> T2). Tak jako základní typy jsou množiny, tak i typ (T1 --->T2) je množina, a to právě množina všech možných funkcí (přiřazení) z T1 do T2. Poznamenejme, že nepožadujeme, aby funkce byly totální, tj. aby každému prvku z T1 něco přiřazovaly. Je-li funkce **f** typu (T1 ----> T2), značíme tuto skutečnost opět
f/(T1 ⟶ T2).

Funkce **f**/(T1 ⟶ T2) přiřazuje prvkům z T1 prvky z T2 (tzv. jednoznačná funkce). Pro adekvátní popis reality je však často vhodné pracovat i s funkcemi, které prvkům jednoho typu T1 přiřazují množiny prvků druhého typu T2 (tzv. mnohoznačné funkce); je-li **g** taková funkce, pak píšeme **g**/(T1 ---> (T2 ---> BOOL)). (Poznamenejme, že (T2 ---> BOOL) je typ tzv. charakteristických funkcí podmnožin množiny T2; charakteristická funkce podmnožiny je funkce, která nabývá hodnoty TRUE na všech prvcích této podmnožiny a hodnoty FALSE na všech ostatních prvcích.
Typ (T1→ (T2 → BOOL)) tedy skutečně znamená přiřazování podmnožin prvků z T2 prvkům z T1.)

Podtřída funkcí, se kterými v HITu pracujeme je tvořena tzv. funkčními závislostmi (hitovskými atributy, HIT-atributy).

DEF 2. Funkční závislostí (stručně funkcí) A budeme nadále rozumět funkci A, která je typu (T1 →T2) resp. typu T1 → (T2 →BOOL)), přičemž platí:

(1) T1 je buď uzlový typ, který není BOOL, nebo n-ticový typ

(2) T2 je libovolný základní typ; v případě, že A je mnohoznačná funkce, pak T2 nesmí být BOOL

(3) alespoň jeden z uzlových typů obsažených v T1 nebo T2 je objektový typ.

2.2 Definovatelnost (odvoditelnost)

Nejdůležitější prostředek pro práci s funkčními závislostmi při návrhu databáze je jejich tzv. definování (odvozování).

DEF 3. Budeme říkat, že funkční závislost A je definovatelná (odvoditelná) z funkčních závislostí A1, ...,An, jestliže lze napsat algoritmus, který vypočte pouze za pomoci logických a matematických operací každou hodnotu funkce A právě z hodnot funkcí A1,...,An a to v libovolném stavu světa (či databáze).

Příklad 1.

Mějme funkce

A1 = (PLAT) daného (#ZAMESTNANEC)/(#ZAMESTNANEC → PLAT)

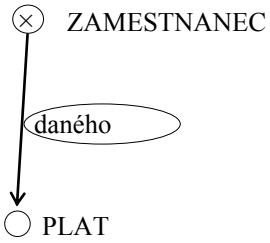
A2 = (#ZAMESTNANEC)-s daneho(#ZAVOD)/
(#ZAVOD ⟶ (#ZAMESTNANEC ⟶ BOOL))

a funkci

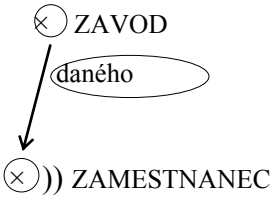
A = (PRUMERNY-PLAT) zaměstnanců daného (#ZAVOD).

V grafickém zápisu:

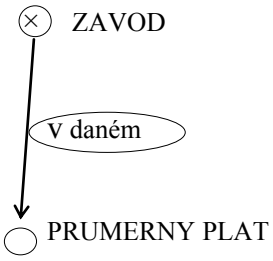
fce A1:



fce A2:



fce A:



Hodnotu funkce A můžeme evidentně pro každý závod vypočíst z hodnot funkcí A1 a A2 algoritmem, který bude obsahovat matematické operace sečítání a dělení (resp. jednoduše matematickou funkci pro výpočet průměru.

Pozn.: Jestliže k dané množině funkčních závislostí přidáme nějakou funkci, která je z nich odvoditelná, pak nám to nepomůže k získání žádné nové informace.

V příkladu jsme zavedli následující konvence:

- jména zákl. typů uzavíráme do závorek
- jména objektových typů jsou vždy vyznačena prefixem #
- na levé straně rovnítka stojí tzv. zkrácený název funkční závislosti, na jeho pravé straně úplný název této funkční závislosti, - úplný název funkční závislosti vyjadřuje její význam, tj. sémantiku
- je-li funkční závislost funkcí mnohoznačnou, vyjadřujeme to příponou "-s", připojenou k první závorce, tj. tvorba množného čísla jakoby anglicky.

Je zřejmé, že to, zda nějakou funkční závislost A je možno považovat za odvoditelnou z funkčních závislostí A1,...,An, závisí na tom, jaké logické resp. matematické operace (funkce) máme pro sestavení algoritmu k dispozici. V dalším textu budeme vždy předpokládat, že máme k dispozici běžné logické (matematické) operace, které lze zapsat standardními prostředky např. programovacího jazyka COBOL.

2.3 Rotace, singularizace, poměr a triviální odvození

Nejdůležitější způsoby odvozování jsou tzv. rotace funkce a triviální odvození funkce.

DEF 4. Rotace funkce A je taková funkce A', pro kterou platí:

(i) množiny výskytů uzlových typů v A' a v A jsou totožné

(ii) funkce A' je odvozena z funkce A nejvýše za použití operace singularizace.

POZN.: Matematická operace singularizace I je operace nad množinami, definovaná takto:

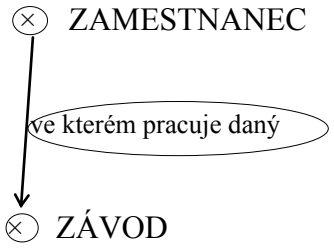
I(M) = x, když M ={x} /jednoprvková množina/
I(M) je nedefinováno v ostatních případech

Příklad 2. Singularizace přináší jisté problémy: Vezměmě funkci A2 z příkladu 1. Její další rotace mohou mít tvar

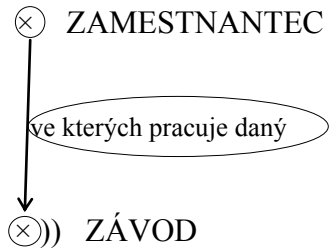
A2' = (#ZAVOD) ve kterém pracuje daný (#ZAMESTNANEC)
A2" = (#ZAVOD)-s ve kterých pracuje daný (#ZAMESTNANEC)

V grafickém zápisu:

fce A2':



fce A2":



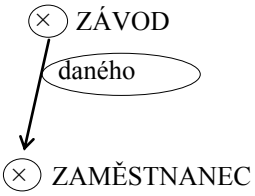
tedy
A2'/(#ZAMĚSTNANEC → #ZÁVOD) a
A2"/(#ZAMĚSTNANEC → (#ZÁVOD → BOOL))

Je zřejmé, že realitu lépe vystihuje A2' než A2". V tomto případě je tedy použití operace singularizace oprávněné. Vyjděme nyní obráceně z funkce A2"; její další možné rotace jsou funkce A2 a funkce

A2"' = (#ZAMESTNANEC) daného (#ZAVOD).

v grafickém zápisu:

fce A2"':



Zde
A2/(#ZÁVOD⟶ (#ZAMĚSTNANEC ⟶ BOOL)) a
A2"'/(#ZÁVOD⟶ #ZAMĚSTNANEC).

Uvědomme si nyní, že A2''' představuje funkci, která je na všech závodech majících více než jednoho zaměstnance nedefinovaná - viz def. singularizátoru. V tomto případě je tedy evidentně použití operace singularizace neoprávněné.

DEF 5. Použití singularizátoru při definování rotace nějaké funkce je oprávněné, jestliže z výsledné singularizované rotace lze definovat (odvodit) zpět původní funkci. Rotace definované bez použití singularizátoru (tzv. plurální rotace) a rotace definované s oprávněným použitím singularizátoru (tzv. singulární rotace) společně nazýváme přípustné rotace. Přípustnou rotaci funkce A značíme rotA. Přípustnou singulární rotaci funkce A nazýváme též singulárním omezením funkční závislosti A.

Pozn.: Jestliže A má obor hodnot opatřený angl. koncovkou "-s", říkáme, že A je dán v plurální rotaci; jestliže A nemá obor hodnot opatřený angl. koncovkou "-s", pak A je dán v singulární rotaci.

Singulární omezení funkční závislosti A jsou speciální případy integritních omezení (nebo-li tvrzení konzistence), vyjadřujících podmínky pro správnou aktualizaci databáze.

Singulární omezení souvisejí s tzv. poměrem funkční závislosti:

DEF 6. Obrácená funkce k funkci A, která je typu (T1→ T2) resp. (T1 →(T2 →BOOL), je funkce A' typu (T2 → T1) resp. (T2 → (T1→ BOOL)), která je přípustnou rotací funkce A.

Schématicky např.

A = (T2) txt_1 (#T11) ... txt_n (T1n)/((T11, ..., T1n) → T2)
A' = (T1)-s txt (T2)/(T2 → (T1 → BOOL)
kde T1 = (#T11, ... , T1n).

DEF 7. Poměrem funkční závislosti A rozumíme zápis

p,m : q,n

kde p udává, zda funkce A je totální (p = 1) nebo parciální (p = 0), q udává totéž pro funkci obrácenou;
m udává, zda funkce A je jednoznačná (m = 1) nebo mnohoznačná (m = M) a n udává totéž pro obrácenou funkci.
Zápis p : q nazýváme dolní poměr;
zápis m : n nazýváme horní poměr.

Pozn.: Známe-li horní poměr funkce A, nelze žádným formálním způsobem odvodit horní poměr libovolné jiné rotace A (vyjma obrácené funkce). Horní poměr každé rotace A představuje netriviální empirický fakt. Z toho vyplývá následující důležité pravidlo:

PRAVIDLO 1: Při návrhu databáze metodou HIT je třeba mít na mysli s každou funkční závislostí i všechny její rotace a zejména všechna její singulární omezení.

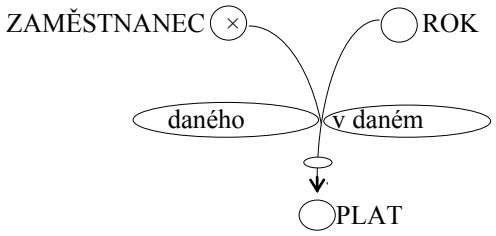
DEF 8. Řekneme, že funkční závislost A' vznikla z funkční závislosti A triviálním odvozením, jestliže hodnota funkce A' je vypočtena (v každém stavu světa) z hodnot funkce A algoritmem, který obsahuje nejvýše singularizaci a/nebo existenční kvantifikátor.

Příklad 3. Mějme funkci

A3 = (PLAT) daného (#ZAMESTNANEC) v daném (ROK)

v grafickém zápisu:

fce A3:



která popisuje "historii" vývoje platu zaměstnance v jednotlivých letech.

Z funkce A3 lze odvodit funkci

A3' = (#ZAMESTNANEC)-s kteří někdy měli daný (PLAT)

v grafickém zápisu:

fce A3':

Hodnoty funkce A3' vypočteme z A3 následujícím algoritmem:

```
p := p/ PLAT
for all z/ZAMĚSTNANEC
        if EX r/ROK (A3(z,r) = p)
        then write z
        else endif
endfor
```

Poněvadž A3 je funkce, znamená zápis A3(z,r) = p aplikaci funkce A3 na argumenty z,r dávající výsledek p. Klíčovým slovem write v zápisu algoritmu označujeme výstup hodnot atributu A3'. Poněvadž algoritmus výpočtu hodnot A3' z A3 neobsahuje jiné logické /matematické funkce než existenční kvantifikátor EX, jedná se o triviální odvození.

2.4 Podfunkce a rozložitelnost

Funkci vzniklou triviálním odvozením z funkce A budeme někdy nazývat podfunkcí funkce A.

Specielně A je podfunkcí A.

Každá podfunkce A' funkce A, která není totožná s A, ani s žádnou její přípustnou rotací, se nazývá vlastní podfunkce funkce A.

Pro usnadnění celého postupu návrhu DB, zejména ve fázi mapování a transformace, je rozumné požadovat, aby funkce se kterými pracujeme byly co nejjednodušší.

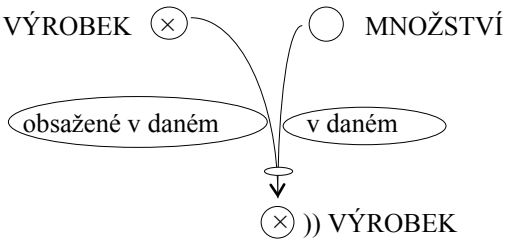
DEF 9. Počet prvků množiny všech výskytů uzlových typů ve funkci A se nazývá složitost funkce A.

Příklad 4. Složitost funkcí A1, A2, A z příkladu 1 je 2; složitost funkce A3 z příkladu 3 je 3. Složitost funkce A4

A4 = (#VYROBEK)-s obsažené v daném (#VYROBEK) v daném (MNOZSTVI)

v grafickém zápisu:

fce A4:



(která popisuje kusovník) je rovněž 3. Množina typů obsažených v A4 je sice dvouprvková, ale množina výskytů typů v A4 je tří prvková.

DEF 10. Řekneme, že funkce A je rozložitelná na funkce A1, A2 jestliže platí

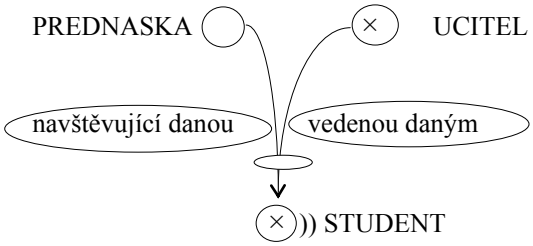
(i) A1 i A2 jsou triviálně odvoditelné z A
(ii) složitost A1 i složitost A2 je menší než složitost A, tj. A1, A2 jsou vlastní podfunkce funkce A
(iii) funkce A je odvoditelná z A1 a A2 za použití logické operace konjunkce a případně operace singularizace.

Příklad 5. Mějme funkci

A5 = (#STUDENT)-s navštěvující danou (PREDNASKA) vedenou daným (#UCITEL) /M:M

v grafickém zápisu:

fce A5:



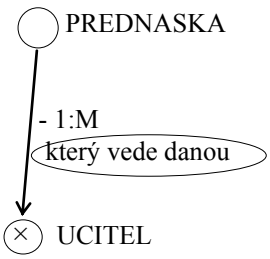
(zde předpokládíme, že přednáška je zajímavá pouze jako jméno / název, tj. nic dalšího o ni neevidujeme, a tedy typ PREDNASKA reprezentuje pouze jakýsi číselník, nebo-li je to vyjmenovaný typ)

a uvažme, že problémová oblast je popsána ještě další funkcí

A6 = (#UCITEL) který vede danou (PREDNASKA) /1:M

v grafickém zápisu:

fce A6:

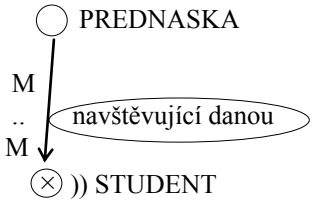


Potom zřejmě z funkce A5 lze triviálně odvodit A6 a další funkci

A7 = (#STUDENT)-s navštěvující danou (PREDNASKA) /M:M

v grafickém zápisu:

fce A7:



Algoritmus pro výpočet hodnot A6 z A5:

```
p: = p / PŘEDNÁŠKA
for all u /UČITEL
        if EX s / STUDENT (s IN A5(p,u))
        then write u to U
        else endif
endfor
if count U NE 1
        then cancel-undef
        else endif
```

Použití operace singularizace v odvození A6 je vyjádřeno v algoritmu příkazem if , který ukončí algoritmus s nedefinovaným výstupem, existuje-li více učitelů splňujících podmínku "s IN A5(p,u)", resp. neexistuje-li žádný.

Algoritmus pro výpočet hodnot A7 z A5:

```
p: = p /PREDNASKA
for all s /STUDENT
        if EX u /UCITEL (s IN A5(p,u))
        then write s
        else endif
endfor
```

Nyní obráceně sestavme algoritmus:

```
 p := p /PREDNASKA
 u := u /UCITEL
for all s /STUDENT
 if s IN A7(p) AND A6(p) = u
          then write s
          else endif
 endfor
```

Snadno vidíme, že algoritmus vrací právě ty studenty, kteří navštěvují přednášku p vedenou učitelem u, tj. vypočítává hodnoty funkce A5 pomocí funkcí A6 a A7. Algoritmus obsahuje pouze logickou operaci konjunkce a tedy A5 je podle def. 1 rozložitelná na A6 a A7.

Pro zkoumání rozložitelnosti funkcí lze použít následující dvě tvrzení.

VĚTA 1. Jestliže existuje rotace rotA funkce A taková, že rotA/ (T $\longrightarrow$ (T1,...,Tk)), kde T je základní typ a T1,...,Tk jsou uzlové typy, pak A je rozložitelná na funkce
A1 / (T $\longrightarrow$ T1)
...
Ak / (T $\longrightarrow$ Tk).

Poznamenejme, že z praktických důvodů zacházíme s některými n-ticovými popisnými typy jako s uzlovými.

Příkladem jsou
DATUM = (DEN,MĚSÍC,ROK),
ADRESA = (ULICE, ČÍSLO_DOMU, PSČ, MĚSTO, STÁT) apod.

Potom samozřejmě ani jednoznačnou funkci, která má takový n-ticový popisný typ v oboru hodnot, nebudeme pokládat za rozložitelnou.

VĚTA 2. Nechť funkce A není rozložitelná podle věty 1. Jestliže existuje vlastní podfunkce A' funkce A, která je přípustnou singulární rotací, potom A je rozložitelná na A' a na funkci A" pro kterou platí:

(i) definiční obor A" je tentýž jako definiční obor A'

(ii) obor hodnot A" obsahuje zbylé uzlové typy funkce A, které nejsou v podfunkci A'.

POZN.: Funkce A5 z příkladu 5 je rozložitelná právě podle věty 2.

2.5 Jádro, HIT-schéma

Pojem jádra je zaveden jako množina nerozložitelných a vzájemně neodvoditelných funkcí. HIT-schéma někdy /v užším smyslu/ chápeme přímo jako (datové) jádro; přesněji je HIT schéma definováno následovně:

DEF 11. HIT-schématem dané problémové oblasti (resp. sémantickým modelem této oblasti) rozumíme trojici následujících seznamů:

B - seznam všech uzlových typů

K - seznam všech funkčních závislostí jádra zapsaný diagramy funkcí nebo ekvivalentně (tj. vždy včetně sémantiky)

C - seznam všech integritních omezení (tvrzení konzistence). Při tom seznam C povinně obsahuje všechna singulární omezení funkcí z K.

2.6 Identifikační funkce a nadtypy, podtypy

Poslední z pojmů metody HIT (mapování reality) jsou pojem identifikační funkce objektového typu a pojmy nadtyp a podtyp u objektových typů.

DEF 12. Nechť E je libovolný objektový typ a T nějaký základní typ, T not= BOOL, a nechť existuje funkce, pro kterou platí

(i) A/(E $\longrightarrow$ T),

(ii) A je (v každém stavu databáze) totální funkce (tj. v každém stavu databáze je každému prvku z E přiřazena funkcí A nějaká hodnota z T).

Potom typ T nazveme kandidátem identifikace objektového typu E a příslušnou funkci A kandidátkou identifikační funkce.

K danému objektovému typu může existovat jeden nebo více kandidátů identifikace. Je-li jich více, pak často nastává případ, že k jednoznačnému určení konkrétního prvku z E stačí pouze nějaká podmnožina vybraná z množiny všech kandidátů identifikace. Pro snadnější implementaci databáze v dalších krocích metody požadujeme, aby mezi kandidáty identifikace, kteří jednoznačně určují konkrétní prvky z E, byl vždy alespoň jeden popisný typ.

DEF 13. Nechť Ai/(E$\rightarrow$ Ti) i=1,...,k, Ti uzlové typy, jsou kandidátky identifikačních funkcí objektového typu E, z nichž alespoň jedna je popisná funkce. Nechť A je funkce typu (E $\longrightarrow$ (T1,...,Tk)) definovaná z funkcí A1,...,Ak tak, že A je rozložitelná na A1,...,Ak podle věty 1. Nechť dále platí

(i) obrácená funkce A' k funkci A je typu ((T1,...,Tk)$\longrightarrow$ E,

(ii) množina (T1,...,Tk) je minimální množina, pro kterou platí (i).

Potom množinu typů (T1,...,Tk) nazýváme identifikace objektového typu E a značíme idE. Funkci A nazýváme identifikační funkcí objektového typu E.

Množina idE = (T1,...,Tk) může být i jednoprvková; potom identifikační funkce A je totožná s jednou kandidátkou identifikační funkce a je nerozložitelná. Poznamenejme, že podmínka (ii) v def. 13 znamená, že kdybychom vypustili libovolný z typů T1,...,Tk, např. Ti, pak obrácená funkce B' k funkci B/(E $\rightarrow$ (T1,...,Ti-1, Ti+1,...,Tk)) by byla typu ((T1,...,Ti-1, Ti+1, ...,Tk) $\rightarrow$ (E $\longrightarrow$BOOL)).

Platí následující tvrzení:

VĚTA 3. Je-li A/(E$\longrightarrow$ T), kde T=(T1,...,Tk), identifikační funkce objektového typu E, pak

(i) A je (v každém stavu databáze) totální funkce

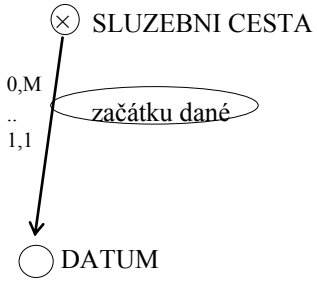
(ii) obrácená funkce A' k funkci A nemusí být totální funkcí.

(iii) hodnoty funkce A jednoznačně určují jednotlivé prvky objektového typu E.

Příklad 6. Vezměme funkce
A8 = (#ZAMESTNANEC) který je účastníkem dané (#SLUZEBNI-CESTA)
/1,1 : 0,M

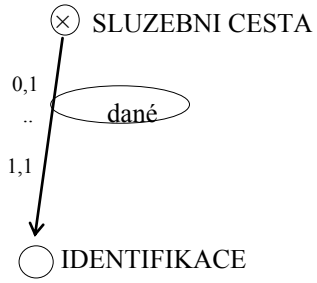
A9 = (DATUM) začátku dané (#SLUZEBNI-CESTA) /1,1 : 0,M
v grafickém zápisu:
fce A8:

fce A9:



Vytvořme n-ticový typ IDENTIFIKACE = (ZAMĚSTNANEC, DATUM) a funkce A8, A9 nahraďme funkcí

A10 = (IDENTIFIKACE) dané (#SLUZEBNI-CESTA) /1,1 : 0,1
v grafickém zápisu:
fce A10:

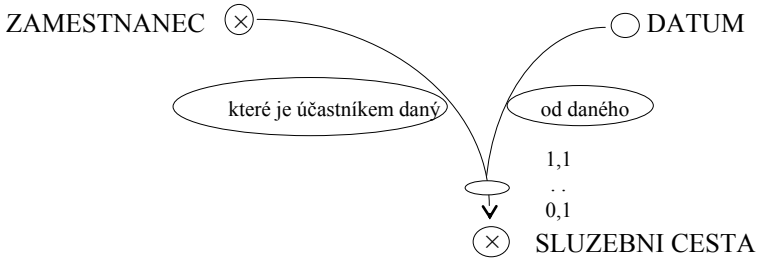


Atribut A10 je totální funkce a funkce A'10 k ní obrácená

A10' = (#SLUZEBNI-CESTA) které je účastníkem daný (#ZAMESTNANEC)
od daného (DATUM) /0,1 : 1,1

v grafickém zápisu:

fce A10' :



je skutečně jednoznačná, ale není totální, jak vidíme z dolního poměru.

DEF 14. Jsou-li E1 a E2 objektové typy, a jestliže pro množiny E1, E2 platí v každém stavu světa (databáze), že E1 je podmnožinou E2, pak říkáme, že E1 je podtypem E2, resp. E2 je nadtypem E1. Jinak též říkáme, že objektové typy E1, E2 jsou ve vztahu podtyp - nadtyp.

Příkladem vztahu podtyp - nadtyp je třeba dvojice UCITEL - ZAMESTNANEC (každý učitel je zároveň zaměstnancem), nebo dvojice PODNIK - ORGANIZACE (každý podnik je zároveň organizací).

PRAVIDLO 2: Jestliže objektové typy E1,...,Ek jsou podtypy objektového typu E, potom má-li E identifikaci, dědí všechny jeho podtypy E1,...,Ek tuto identifikaci a žádná další identifikace se u nich neurčuje. Jestliže obráceně, každý z podtypů E1,...,Ek má svoji identifikaci (tyto identifkace mohou být navzájem různé), podtypy jsou po dvou disjunktní a sjednocením těchto podtypů dostaneme celý nadtyp E, pak u nadtypu E se žádná další identifikace neurčuje.

DEF 15. Jestliže E1,...Ek jsou podtypy typu E a E má identifikaci, říkáme, že objektový typ E (nadtyp E) je explicitně identifikován. Dále říkáme, že podtypy E1,...Ek jsou v tomto případě identifikovány implicitně. Jestliže každý z podtypů E1,...Ek má svoji vlastní identifikaci, podtypy jsou po dvou disjunktní a sjednocením těchto podtypů dostaneme celý nadtyp E, říkáme, že (podtypy) E1,...,Ek jsou explicitně identifikovány a (nadtyp) E je identifikován implicitně.

PRAVIDLO 3: Každý objektový typ, který se vyskytuje v seznamu uzlových typů HIT-schématu /viz. def.11/ musí mít explicitní nebo implicitní identifikaci!

3. VYBRANÁ METODICKÁ DOPORUČENÍ

V této části jsou shrnuty poznatky jejichž využití může projektantovi / analytikovi usnadnit návrh "dobrého" schématu z hlediska srozumitelnosti a jednoznačnosti použitých pojmů (přirozeného jazyka). Pravidla pro testování rozložitelnosti jsou zobecněna do jednoho návodu (viz 3.4) a pro zkoumání rozložitelnosti a definovatelnosti jsou doporučeny v praxi užitečné techniky (viz 3.4 a 3.5). Domníváme se, že tyto techniky mohou projektantovi usnadnit tvůrčí využití HIT metody v praxi při práci s automatizovanou podporou projektování v rámci centrálního katalogu komplexního informačního systému, představovaného dnes většinou datovým slovníkem nějakého CASE prostředku.

Poznamenejme, že v dalším textu budeme často používat alternativních termínů pro některé základní pojmy: objektový typ = entitní sorta, popisný typ = deskriptivní sorta, odvoditelnost = definovatelnost.

3.1 Role přirozeného jazyka v HIT metodě

Přirozený jazyk hraje při aplikaci HIT metody v praxi velmi důležitou úlohu. Jednak jako prostředek k dorozumění projektanta s budoucím uživatelem (případně jej zastupujícím expertem) a jednak ve standardizované podobě jako prostředek pro zápis navrženého schématu. Domníváme se, že to je nejlepší způsob, jak docílit toho, aby navržené schéma bylo srozumitelné a jednoznačně chápané jak všemi zúčastněnými osobami v průběhu návrhu, tak i kýmkoliv jiným někdy později, a to bez ohledu na to, v jaké oblasti je HIT metoda aplikována.

3.2 Pojmenování základních typů

Ve většině případů se při vybírání jmen pro základní sorty vystačí s podstatnými jmény, které se objevují ve větách formulovaných expertem. Dá se říci, že každé podstatné jméno je kandidátem na název základního typu.

Například expert řekne: "Chtěli bychom evidovat všechny nehody našich aut, přičemž nás zajímá, komu se staly, kde se staly a kdy." Podtržená podstatná jména jsou kandidáti na názvy základních typů. Je ovšem zřejmé, že z věty vyplývají další kandidáti - nepřímo podtrženými zájmeny vyjádřená podstatná jména: RIDIC, MISTO, CAS.

Řidčeji se pro název základního typu použije složitějšího gramatického útvaru jako např. VEDLEJSÍ PRACOVNI POMER, ZPUSOB LECBY, TYP AUTA.

Doporučení k používání n-ticových typů: Tyto typy by se měly používat co nejméně. Převážně je používáme pro stručné a přehledné vyjádření expertova pohledu na konkrétní hodnoty aktuálních naplnění nějaké funkční závislosti pokud to usnadní porozumění mezi projektantem a expertem. K funkcím s n-ticovými typy by se projektant měl později vrátit a analyzovat je bez použití n-ticových typů. Jinak by měl dát raději přednost entitním sortám vycházejícím z vlastností n-tic. Název vybraný pro n-ticový typ můžeme alternativně chápat jako název entitní sorty definované vlastností příslušných n-tic. Např. typ OBDOBI = (DATUM, DATUM) : vlastnost "být obdobím" je vlastnost dvojic typu (DATUM, DATUM); tedy OBDOBI můžeme alternativně chápat jako název entitní sorty.

PŘÍKLAD: Popis s použitím n-ticového typu: název funkční závislosti nechť je:

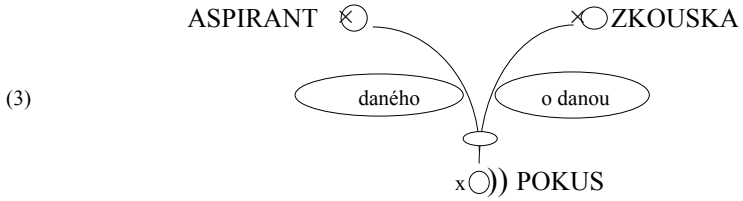
(POKUS)-s daného (#ASPIRANT) o danou (#ZKOUSKA),

Typ POKUS je kartézským součinem sort DATUM a VÝSLEDEK: jakákoliv dvojice prvků těchto typů je prvkem typu POKUS. Jestliže tedy dva aspiranti učiní v tentýž den každý jeden pokus o nějakou zkoušku s tímtéž výsledkem, jsou záznamy o těchto dvou pokusech nerozlišitelné. Půjde o tutéž dvojici typu POKUS. Důsledek tohoto chápání je ten, že výše uvedená funkce je nerozložitelná, ale přitom je zbytečně složitá.

Jestliže se rozhodneme chápat POKUS jako entitní sortu, dostaneme:

(#POKUS)-s daného (#ASPIRANT) o danou (#ZKOUSKA) (3)

v grafickém zápisu:



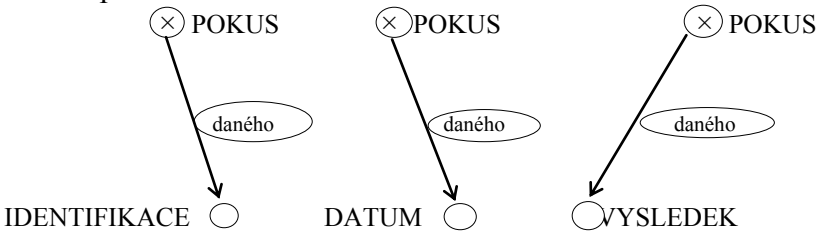
a funkce

(IDENTIFIKACE) daného (#POKUS)

(DATUM) daného (#POKUS)

(VYSLEDEK) daného (#POKUS)

v grafickém zápisu:

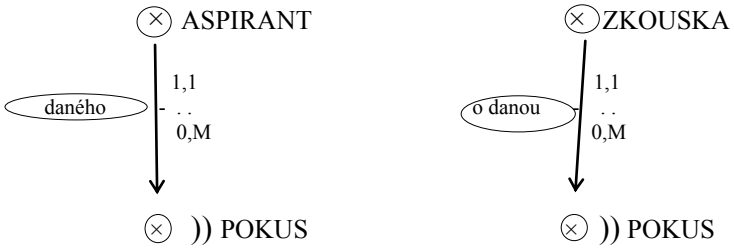


Funkce (3) je rozložitelná, protože každé dva pokusy jsou různé prvky sorty POKUS (každý musí mít jinou identifikaci) bez ohledu na to, zda popisy obou pokusů pomocí typů DATUM a VÝSLEDEK jsou stejné nebo ne. Dostáváme tedy místo funkční závislosti (3):

(#POKUS)-s daného (#ASPIRANT) /0,M : 1,1

(#POKUS)-s o danou (#ZKOUSKA) /0,M : 1,1

v grafickém zápisu:



3.3 Pojmenování atributů (atributových funkcí)

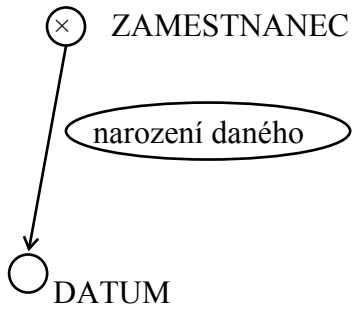
Názvem funkční závislosti musí být nějaký složený výraz přirozeného jazyka, tj. několik slov přirozeného jazyka v gramaticky správném spojení. Z názvu musí být patrno, o jakou charakteristiku jde a také musí být patrno, o charakteristiku čeho se jedná. Inspiraci pro navrhování názvů jednotlivých funkčních závislostí projektant hledá ve větách, které se dovídá od experta. Musí umět rozpoznat ty výrazy přirozeného jazyka, které mají funkční charakter - tj. **označují přiřazení něčeho něčemu**.

V jednoduchých případech je to zřejmé. Tak např.

DATUM narození daného ZAMĚSTNANCE,

tj. (DATUM) narození daného (#ZAMESTNANEC)

graficky



je funkce zaměstnanců a je přirozené, že ji chápeme jako funkci, která (když mluvíme o aktuálním světě a určitém okamžiku) pro každého zaměstnance dá jeho datum narození. Můžeme totiž přirozeně říci

"DATUM narození ZAMĚSTNANCE Josefa Nováka",

tj. **(**(DATUM) narození daného (#ZAMESTNANEC)**)** (Josef Novák)

tj. uvedný výraz lze aplikovat na argument (Josef Novák).

Další příklady tohoto druhu:

VÁHA daného VÝROBKU, tj. (VAHA) daného (#VYROBEK)

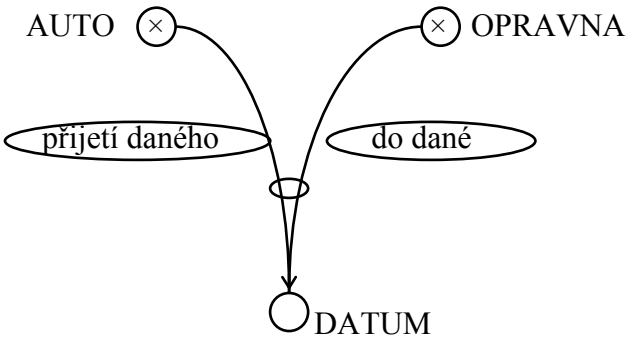
ZNAČKA daného AUTA, tj. (ZNACKA) daného (#AUTO).

Uvědomit si, že takovéto výrazy přirozeného jazyka označují funkční závislosti, nedělá žádné potíže. To proto, že jde o funkce jednotlivých věcí - ZAMĚSTNANCŮ, VÝROBKŮ, AUT atd. a navíc jde o funkce, které na ničem jiném než na příslušnosti k té věci, kterou popisují, nezávisí a které se příliš nemění v čase.

Jde-li o funkce více věcí (obecně n-tic nějakých věcí), které jsou spolu v nějakém vztahu, je sestavení názvu takové funkční závislosti obtížnější. Tak, např.

DATUM přijetí daného AUTA do dané OPRAVNY,
tj.: (DATUM) přijetí daného (#AUTO) do dané (#OPRAVNA)

graficky:



DATUM ukončení opravy daného AUTA od dané OPRAVNY,
tj.: (DATUM) ukončení opravy daného (#AUTO) od dané (#OPRAVNA)

CENA opravy daného AUTA v dané OPRAVNĚ,
tj.: (CENA) opravy daného (#AUTO) v dané (#OPRAVNA)

DATUM odvozu daného AUTA z dané OPRAVNY.
tj.: (DATUM) odvozu daného (#AUTO) z dané (#OPRAVNA)

To všechno jsou funkce dvojic typů složených z aut a opraven, které spolu souvisí tak, že auto má být nebo bylo opraveno v opravně. Zavedení entitní sorty OPRAVA může popis zjednodušit. Je ale nutné uvědomit si, že každá oprava je teď pro nás entitou, tj. objektem zájmu. Pak bychom mohli vyprojektovat:

DATUM přijetí auta do dané OPRAVY,
tj.: (DATUM) přijetí auta do dané (#OPRAVA)

DATUM ukončení dané OPRAVY,
tj.: (DATUM) ukončení dané (#OPRAVA)

CENA dané OPRAVY,
tj.: (CENA) dané (#OPRAVA)

DATUM odvozu auta z dané OPRAVY
tj.: (DATUM) odvozu auta z dané (#OPRAVA) a dále

OPRAVY v dané PROVOZOVNĚ (=OPRAVNĚ)
tj.: (#OPRAVA)-s v dané (#OPRAVNA)

(rotace: PROVOZOVNA, ve které se provedla daná OPRAVA),
tj.: (#OPRAVNA) ve které se provedla daná (#OPRAVA)

OPRAVY daného AUTA
tj.: (#OPRAVA)-s v daného (#AUTO)

(rotace: AUTO, u kterého byla provedena daná OPRAVA),
tj.: (#AUTO) u kterého byla provedena daná (#OPRAVA)

a samozřejmě identifikační funkce:

identifikační CISLO dané OPRAVY,
tj.: identifikační (CISLO) dané (#OPRAVA).

V případech, kdy se funkce-přiřazení toho, co popisujeme, mohou měnit, se projektant musí experta dotazovat, zda ho evidence těchto změn zajímá a názvy projektovaných funkčních závislostí tomu přizpůsobovat.

Tak může vyprojektovat: CENA daného VÝROBKU

nebo

CENA daného VÝROBKU v daném ROCE

Diskusi o tom, jak rozeznávat, o jaké funkční závislosti expertovi jde a jak je zapsat zakončíme doporučením:

DOPORUČENÍ:

Při mapování, se sice snažíme rozeznat funkční závislosti jádra od ostatních funkčních závislostí, avšak soustředíme se hlavně na vlastní proces mapování. Oddělení funkcí jádra od ostatních můžeme nechat na pozdější dobu.

3.4 Jak zkoumat, zda funkce není rozložitelná

Otázka zní, jak má projektant poznat, že funkční závislost, kterou navrhl, je rozložitelná. Vpodstatě má k dispozici větu o rozkladu uvedenou přesně v odstavci 2.4. v kapitole 2. jako věta 2. Podstata použití této věty je následující:

Je-li funkce X podezřelá z rozložitelnosti snažíme se najít takovou její rotaci X', pro kterou by platilo, že její hodnoty (přesněji hodnoty každého jejího aktuálního naplnění) závisí pouze na prvních k argumentech z definičního oboru a nezávisí na zbylých m-k argumentech - tj. závisí na nich konstantně. To pak znamená, že hodnoty (k+1)-ího až m-tého argumentu závisí na hodnotách prvních k-argumentů. Je-li tomu tak, pak rotace X' a tedy i funkce X je rozložitelná na funkce Y a Z - viz následující schémata funkcí:

(A1,...,Ak,Ak+1,...,Am) $\rightarrow$ Am+1 funkce X'

(kde k je menší než m), je rozložena na

(A1,...,Ak) $\longrightarrow$ Am+1 funkci Y

a na

(A1,...,Ak,Aik+1,...,Aim-1)⟶ Aim (⟶ BOOL) funkce Z

kde (ik+1,...,im) je nějaká vhodná permutace indexů (k+1,...,m).

Dále pro funkci

(A1,...,Ak,Ak+1,...,Am) ⟶ (Am+1 ⟶ BOOL) funkce X'

bude mít rozklad tvar

(A1,...,Ak) ⟶ (Am+1 ⟶ BOOL) funkce Y

a

(A1,...,Ak,Aik+1,...,Aim-1)⟶ Aim (⟶ BOOL) funkce Z

kde pro indexy platí totéž, co nahoře.

Mnohoznačnost funkce Z je v obou případech uvedena v závorce, protože v konkrétních případech tam může být i nemusí.

Poznamenejme, že Y a Z jsou samostatné, na sobě nezávislé podfunkce funkce X i její rotace X'.

Znovu vyvstává otázka: Jak má projektant poznat, že hodnoty dané funkce závisí pouze na některých jeho argumentech? Zkušenost ukazuje, že standardizované názvy jednotlivých rotací mohou úvahy o rozložitelnosti usnadnit.

Příklad na rozklad pomocí jednoznačné funkce:

NEHODY které měl daný ŘIDIČ s daným AUTEM daného DATA,

tj.: (#NEHODA)-s které měl daný (#RIDIC) s daným (#AUTO) daného (DATUM)

Uděláme rotaci směrem k řidiči a pojmenujeme ji takto:

ŘIDIČ který měl danou NEHODU s daným AUTEM daného DATA,

tj.: (#ŘIDIČ) který měl danou (#NEHODA) s daným (#AUTO) daného (DATUM)

Název by měl pomoci k zjištění, že podtržená část, tj. "ŘIDIČ, který měl danou NEHODU", je samostatná a nezávisí na hodnotách v aktuálním naplnění typů AUTO a DATUM, a že tedy původní funkci můžeme rozložit na tuto část a na

DATUM dané NEHODY daného AUTA
tj.: (DATUM) dané (#NEHODA) daného (#AUTO).

Na tuto funkci ovšem uvedený postup můžeme aplikovat ještě jednou a dostaneme:

DATUM dané NEHODY
tj.: (DATUM) dané (#NEHODA)

a

NEHODY daného AUTA / n:1
tj.: (#NEHODA)-s daného (#AUTO) /M:1

Poměrem jsme vyjádřili, že rotace uvedené funkce bude jednoznačnou funkcí).

Příklad na rozklad pomocí mnohoznačné funkce:

ČÁSTKA na daném ÚČTU daného VLASTNÍKA k danému DATU,
tj.: (CASTKA) na daném (#UCET) daného (#VLASTNIK) k danému (DATUM)

Uděláme rotaci k vlastníkovi (bude to mnohoznačná funkce, poněvadž dotazem v bance zjistíme, že jeden účet může mít více vlastníků) a pojmenujeme ji takto:

VLASTNÍCI daného ÚČTU na kterém je daná ČÁSTKA k danému DATU,

tj.: (#VLASTNIK)-s daného (#UCET) na kterém je daná (CASTKA)
k danému (DATUM) /M:M

Poněvadž VLASTNÍCI daného ÚČTU zjevně nezávisí na ČÁSTCE a DATUMu, tj. podtržená část je samostatná funkce, opět rozkládáme, a to na tuto funkci a na funkci:

ČÁSTKA na daném ÚČTU k danému DATU,
tj.: (CASTKA) na daném (#UCET) k danému (DATUM)

Poznamenejme na okraj, že úvahy o rozkladu nelze považovat za formální záležitost. Třeba při naposled uvedeném rozkladu se může přijít na to, že ÚČET může mít víc VLASTNÍKŮ proto, že jsou vlastníci a spoluvlastníci účtů a že dohromady se jim říká disponenti. Funkční závislost "VLASTNÍCI daného ÚČTU" by se pak musela přejmenovat na "DISPONENTI daného ÚCTU" nebo nahradit dvěma funkčními závislostmi "VLASTNÍK daného ÚCTU" a "SPOLUVLASTNÍCI daného ÚČTU", jestliže bychom chtěli odlišit vlastníky od spoluvlastníků.

Aby nevznikl dojem že všechny funkce lze tímto způsobem rozkládat, uvedeme aspoň dva příklady funkčních závislostí, které jsou složitější, ale nerozložitelné:

KURS dané MĚNY k dané MĚNĚ v dané BANCE k danému DATU,
tj.: (KURS) dané (MENA) k dané (MENA) v dané (#BANKA) k danému (DATUM)

nebo

VÝROBKY dodané daným DODAVATELEM danému ODBĚRATELI,
tj.: (#VYROBEK)-s dodané daným (#DODAVATEL) danému (#ODBERATEL).

Další příklady na úvahy o rozložitelnosti:

Příklad 1: Funkce

PLAT daného ZAMĚSTNANCE, který pracuje na daném PRACOVIŠTI

je zřejmě rozložitelná. Dejme tomu ale, že byl vyprojektován název

PLAT daného ZAMĚSTNANCE na daném PRACOVIŠTI.

Předpokládejme, že expertovi jde o to, aby se evidoval plat jednotlivých zaměstnanců a pracoviště, na kterých (kterém?) pracuje. Pak je ale první název výstižnější a druhý název zavádějící. Druhý totiž vyjadřuje, že máme na mysli jednotlivé platy každého zaměstnance na jeho jednotlivých pracovištích. Kdyby měl expert na mysli toto, pak by byl druhý název v pořádku a funkce by byla nerozložitelná.

Zajímavé je, že první název, který byl použit, napovídá, že funkce, kterou tím názvem označujeme, je rozložitelná bez ohledu na to, zda zaměstnanec pracoval na více než jednom pracovišti, tj. i v případě, že druhý podatribut je

PRACOVISTE na kterých pracuje daný ZAMESTNANEC

Příklad 2: Jde o rozklad funkce

vlastní DĚTI daného OTCE a dané MATKY.

To co bychom chtěli vyjádřit (evidovat) pomocí takto nazvané funkce, můžeme rozdělit do dvou funkcí

vlastní MATKA daného DÍTEte (1)

a

vlastní OTEC daného DÍTEte (2)

a původní funkci jsme vždycky schopni z těchto dvou složit.

I toto řešení však má háček. V těchto dvou funkcích (1) a (2) můžeme evidovat víc informací než v původním funkci složitosti 3, pomocí které nemůžeme zaznamenat to, že sice víme, která je vlastní matka nějakého dítěte, ale není znám jeho vlastní otec. Jestliže tedy vyprojektujeme poslední dvě uvedené funkce, nemůžeme říci, že vznikly rozkladem původní funkce složitosti 3; prostě jsme původní funkci nahradili novými dvěma.

3.5 Jak zkoumat, zda funkce není nad jinou definovatelná

Funkce, definovatelné nad jinými funkcemi jsou dvojího druhu. Buďto jsou definovatelné nad jinými funkcemi proto, že jsou na tyto funkce rozložitelné. Tomu případu je věnována předchozí podkapitola. Anebo jde o funkce, které jsou sice nerozložitelné, ale přesto definovatelné nad jinými funkcemi. (Může nastat i kombinace obou případů.) Nyní se budeme věnovat druhému případu.

Jestliže chceme přijít na to, že nějaká funkce je definovatelná nad jinými funkcemi, pak musíme najít jakým algoritmem a z jakých funkcí je vypočitatelná. Tuto otázku lze v praxi většinou lehce zodpovědět, pokud jde o agregované funkce, které slouží řízení na vyšším než základním stupni. Tak je tomu v případě, že se jedná o různé přehledy, výkazy a ukazatele. Avšak i tyto, na první pohled definovatelné funkce mohou být považovány za nedefinovatelné, jestliže se v systému, který projektujeme, nebudou evidovat základní funkční závislosti, ze kterých se předchozí dají vypočítat.

Co doporučit, jako pomůcku k úvahám zda daná funkce je definovatelná z jiných či nikoliv?

Situace je podobná, jako při určování primitivní funkce (neurčitého integrálu) k dané funkci v integrálním počtu. Žádný univerzální a formální postup neexistuje. Dá se říci, že ve většině případů jsou funkce, které projektant navrhuje pomocí HIT metody, nedefinovatelné nad

ostatními. Jsou to většinou funkce, které reprezentují bezprostřední charakteristiky jednotlivých věcí (nebo více věcí najednou).

Na příklad:

ADRESA dané PROVOZOVNY,

VEDOUCÍ dané PROVOZOVNY,

VĚDNÍ OBOR, ve kterém daný ZAMĚSTNANEC získal vědeckou hodnost,

MNOŽSTVÍ daného HNOJIVA použitého na daném POLI v daném ROCE. atd.

U funkcí, které se budou projektantovi zdát podezřelé z definovatelnosti nad jinými funkcemi, je jediná možnost: vymyslet funkce, nad kterými je podezřelá funkce definována a formulovat algoritmus, kterým je aktuální naplnění podezřelé funkce vypočítáváno z aktuálních naplnění těchto funkcí. Vůbec nezáleží na tom jestli funkce, které se vymyslí, jsou již vyprojektovány nebo ne. Nechť funkce B1,...,Bn jsou ty nad kterými je A definována; doporučujeme přemýšlet nad množinou funkcí (A, B1,..., Bn) a spolu s expertem rozhodnout, které z nich budeme považovat za základní, a které za definovatelné. Je jasné, že základní nemusí být zrovna funkce B1,...,Bn. Se všemi funkcemi A,B1,...,Bn, které jsou přitom ve hře, pak naložíme podle toho jak výše naznačené úvahy dopadnou.

Uvedeme jednoduchý ilustrační příklad: Představme si, že projektujeme skladové hospodářství obchodu. Jedna z funkcí, kterou navrhneme, bude

TYP daného VÝROBKU.

Dejme tomu, že už jsme před tím navrhli funkci

MODEL daného VÝROBKU,

a že máme představu, jaká je souvislost mezi MODELEM a TYPEM. To nás může přivést k podezření, že první funkce - TYP daného VÝROBKU je definovatelná. Pokusíme se tedy vymyslet nad čím a dojdeme k závěru, že je definován nad funkcemi

MODEL daného VÝROBKU

a

MODELY, na které je rozdělen daný TYP

(s horním poměrem M:1, vyjadřujícím, že každý model patří do jediného typu).

Uvažujeme-li však nad všemi třemi funkcemi dohromady, můžeme dojít k závěru, že

TYP daného VÝROBKU

a

MODEL daného VÝROBKU

budeme považovat za základní i přesto, že horní poměr definovatelné třetí funkce budeme muset zařadit do konceptuálního schématu jako tvrzení konzistence.

Poznamenejme, že podobná úvaha může proběhnout i když začneme od funkce

MODELY, na které je rozdělen daný VÝROBEK,

jestliže jej začneme podezřívat z definovatelnosti.

Upozorněme znovu, že funkci je možno považovat za definovatelnou, jestliže na výpočet jakéhokoliv jejího aktuálního naplnění nemá vliv nic jiného, než aktuální hodnoty funkcí, ze kterých se vychází. I slovo algoritmus, použité expertem, může být ošidné. Při projektování ukazatelové databanky pro vrcholové řízení v resortu jsme se setkali s následujícím případem: Každý ODVOZENÝ UKAZATEL se počítá ze ZÁKLADNÍCH UKAZATELŮ podle zadaného algoritmu. Vycházeje z této informace, je funkce

HODNOTA daného ODVOZENÉHO UKAZATELE v daném ROCE

definovatelná nad funkcí

HODNOTA daného ZÁKLADNÍHO UKAZATELE v daném ROCE.

Problém je v tom, že algoritmy se rok od roku mohou měnit, takže je nutné je chápat jako objekt a funkce, které je popisují, vzít v úvahu při zkoumání definovatelnosti funkcí popisujících odvozené ukazatele.

4. TRANSFORMACE SCHÉMATU FUNKČNÍCH ZÁVISLOSTÍ DO C-SCHÉMATU

Schéma funkčních závislostí představuje jistý **informační potenciál - informační schopnost**, tj. objem informace, který je možno uložit do databáze popsané tímto schématem. Schéma funkčních závislostí popisuje daný informační potenciál na sémantické úrovni, tj. tak, aby bylo podkladem pro komunikace lidí - řešitelů, uživatelů, manažerů.

Cílem informační analýzy komponenty je však vytvoření konceptuálního (logického) schématu, C-schématu, které daný informační potenciál popisuje jako strukturu logických souborů a vazeb mezi nimi, tj. ve tvaru ER diagramu.

ER-diagram se získá z HIT-schématu následujícím algoritmem:

(1) Ke každé funkci složitosti n větší než 2 zavedeme konkatenovaný typ, který bude složený právě ze všech uzlových typů obsažených v dané funkci; danou funkci nahradíme konfigurací n funkcí (z nichž každá je složitosti 2), jejichž společným definičním oborem je zavedený konkatenovaný typ a jejichž obory hodnot jsou po řadě všechny uzlové typy původní funkce.

POZNÁMKA: Krokem 1 jsme vlastně zavedli vztahové množiny známé z Chenova E-R modelu.

(2) Všechna singulární omezení platná pro danou (původní) funkci přeformulujeme jako tvrzení konzistence vyjádřená pomocí vzniklých funkcí.

(3) Každý objektový typ, který je explicitně identifikován a každý konkatenovaný typ reprezentujeme jedním logickým souborem.

(4) Popisné funkce obsahující explicitně identifikovaný objektový typ, které neobsahují význačný popisný typ, reprezentujeme pomocí struktury (tj. pomocí položek) logického souboru, který dle kroku 3 reprezentuje příslušný objektový typ.

POZNÁMKA: Význačný popisný typ D je takový, pro který platí: pro každou popisnou funkci, ve které je typ D obsažen platí, že singulární rotace této funkce s typem D v oboru hodnot není přípustná. Jinak řečeno: každá funkce, která má typ D v oboru hodnot je mnohoznačná.

(5) Popisné funkce, které obsahují implicitně identifikovaný objektový typ, nahradíme popisnými funkcemi, které z původních získáme záměnou implicitně identifikovaného objektového typu za explicitně identifikovaný objektový typ z dvojice nadtyp-podtyp. Jedná se o zúžení resp. rozšíření definičního oboru funkce. Dále pokračujeme krokem 4.

(6) Popisné funkce obsahující konkatenovaný typ, které neobsahují význačný popisný typ, reprezentujeme pomocí struktury (tj. pomocí položek) logického souboru, který dle kroku 3 reprezentuje příslušný konkatenovaný typ.

(7) Význačné popisné typy reprezentujeme samostatnými logickými soubory, jejichž struktura obsahuje kromě identifikace souboru jedinou položku reprezentující daný popisný typ.

(8) Vztahové funkce, v nichž oba výskyty objektových typů patří k explicitně identifikovaným objektovým typům, resp. jeden z nich je konkatenovaný typ, reprezentujeme spojeními, jejichž typy jsou určeny horními poměry odpovídajících vztahových funkcí dle tabulky (m, n větší než 1):

Poměr	typ spojení
1:1	1-1
1:n	1-M
m:1	M-1
m:n	M-M

(9) Jestliže ve vztahové funkci je některý z výskytů objektových typů příslušný k implicitně identifikovanému objektovému typu, nahradíme jej funkcí, která z původní vznikne záměnou výskytu implicitně identifikovaného objektového typu za výskyt explicitně identifikovaného objektového typu (z dvojice nadtyp-podtyp). Opět se jedná o rozšíření resp. zúžení definičního oboru funkce. Dále pokračujeme krokem 8.

(10) Výsledkem popsaného algoritmu je C-schéma popsané prakticky v entitním modelu, tj. ER schéma. Logické soubory, které C-schéma tvoří, můžeme pokládat za tabulky (relace) ve 4NF (čtvrté normální formě).

KONEC ALGORITMU