Otázka 19 – A7B36DBS

Zadání

Návrh relačního schématu. Normalizace schématu formou dekompozice. Kritéria kvality dekompozice. Návrh schématu relační databáze přímou transformací z konceptuálního schématu. (A7B36DBS)

Slovníček pojmů

- **relace** množina prvků, též množina n-tic
- **atribut** jeden prvek z dané relace
- **doména** množina hodnot, kterých může atribut nabývat
- **stupeň relace** je počet atributů relace
- **relační schéma** výraz tvaru R(A, f), kde R je jméno schématu, A = {A1, A2,…, An} je konečná množina jmen atributů, f je zobrazení přiřazující každému jménu atributu Ai neprázdnou množinu (obor hodnot atributu), kterou nazýváme doménou atributu Di, tedy f(Ai) = Di. Často se tímto pojmem rozumí název relace a v závorce uvedené její atributy, tedy například Kino(Název, Adresa,Rok_založení)
- **relační model** sdružení dat do tzv. relací (tabulek), které obsahují n-tice (řádky). Tabulky (relace) tvoří základ relační databáze. Tabulka je struktura záznamů s pevně stanovenými položkami (sloupci - atributy). Každý sloupec má definován jednoznačný název, typ a rozsah, neboli doménu. Záznam se stává n-ticí (řádkem) tabulky. Pokud jsou v různých tabulkách sloupce stejného typu, pak tyto sloupce mohou vytvářet vazby mezi jednotlivými tabulkami. Tabulky se poté naplňují vlastním obsahem - konkrétními daty
- **relační databáze** databáze založená na relačním modelu
- **tabulka** - reprezentace instance relačního schématu
- **primární klíč** - sloupec (nebo sloupce), který jednoznačně určuje řádky v tabulce
- **cizí klíč** - slouží pro vyjádření vztahů, relací, mezi databázovými tabulkami. Jedná se o pole či skupinu polí, která nám umožní identifikovat, které záznamy z různých tabulek spolu navzájem souvisí.
- **normalizace** - proces rozkladu údajů do množin dat, která jsou spojeny pomocí společného prvku (jinak: Normalizace je proces rozhodování jaký sloupec umístíme v které tabulce.)
- **funkční závislost** - sloupec A je funkčně závislý na B právě tehdy když, pro každou hodnotu ve sloupci A existuje nejvýše jedna hodnota ve sloupci B.
- **konceptuální model databáze**
    - databázový model umožňující zobrazit a popsat objekty v databázi a vztahy mezi nimi z hlediska jejich významu a chování

- výsledkem konceptuálního modelování je implementačně nezávislé databázové schéma, tj. schéma obecně aplikovatelné v jakémkoli technicko-programovém prostředí.

Návrh relačního schématu

Při návrhu relačního schématu se obvykle používá transformace z konceptuálního schématu.
Konceptuální (někdy také sémantické) modely jsou pokusem umožnit vytvoření popisu dat v databázi nezávisle na jejich uložení. Konceptuální model představuje formální popis modelované reality. Hlavními úkoly je nalezení entit, vtahů a atributů. Slouží obvykle k vytvoření schémat s následnou transformací na databázové schéma. Spojíme-li sémantickou a databázovou úroveň, dostáváme se k tzv. objektově orientovaným SŘBD. Konceptuální modely používají pojmy:
entita (objekt) - student,předmět
vztah (relationship) - studuje
atributy (vlastnost) - věk, rč

Činnosti při tvorbě E-R modelu:

E-R model je množina pojmů, které nám pomáhají, na konceptuální úrovni abstrakce popsat uživatelskou aplikaci za účelem specifikovat následně strukturu databáze. Každá entita musí být jednoznačně identifikovatelná. Atribut (skupina atributů), jehož hodnota slouží k indentifikaci konkrétní entity se nazývá identifikačním klíčem.

- identifikace typů entit jako množiny objektů stejného typu. Např. KNIHA, ABONENT_KNIHOVNY, ZAMESTNANEC označují typy entit.
- identifikace typů vztahů, do kterých entity identifikovaných typů mohou vstupovat. Např. ABONENT(entita) MA_PUJCEN (vztah) daný EXEMPLAR (entita).
- na základě přiměřené úrovně abstrakce přiřazení jednotlivým typům entit a vztahů popisné atributy. Např. PRIJMENI (popisný atribut) daného ZAMESTNANCE (entita), DATUM (popisný atribut), do kdy si daný ABONENT (entita) VYPUJCIL (údaj typu vztah) daný EXEMPLAR (entita).
- formulace integritních omezení (IO) vyjadřujících s větší či menší přesností soulad schématu s modelovanou realitou.

Entita je objekt reálného světa, který je schopen nezávislé existence a je jednoznačně odlišitelný od ostatních objektů. Vztah je vazba mezi dvěma entitami (obecně i více entitami). Hodnota popisného typu popisným typem budeme rozumět jednoduchý datový typ. Atributem budeme rozumět funkci přiřazující entitám či vztahům hodnotu popisného typu, určující některou podstatnou vlastnost entity nebo vztahu.

Příklad:

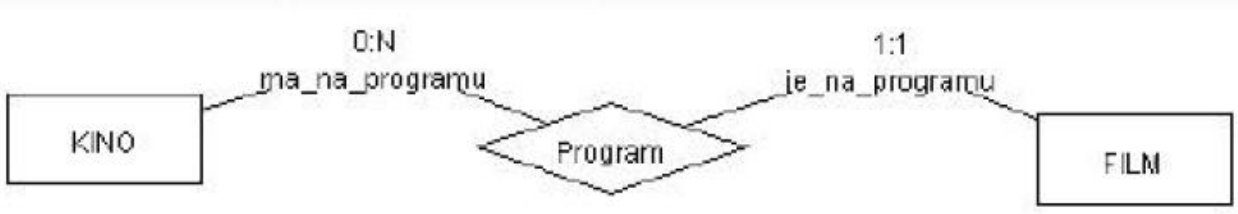
lineární zápis:
E: Zaměstnanec, Oddělení
R: Je zaměstnán na(Zaměstnanec, Oddělení)

Do E-R modelu dále značíme:

Kardinalita vztahů = násobnost účasti ve vztahu (1:1, 1:N, M:N)
Parcialita = povinnost účasti ve vztahu:
povinná účast = všechny výskyty účastníka musí být zapojeny do příslušného vztahu
nepovinná účast = jednotlivé výskyty členské entity mohou být zapojeny do vztahu daného typu

Příklad:

Kino může mnohokrát. Film musí právě jednou.



Slabé entitní typy

Součástí klíče některých entitních typů nemusí pouze být jejich vlastní atributy. V takovém případě nemusíme být schopni rozlišit mezi dvěma instancemi jednoho entitního typu na základě hodnot jeho vlastních atributů. V takovém případě mluvíme o slabém entitním typu. Jeho různé instance jsou identifikovatelné tím, že jsou v povinném vztahu k instanci entity jiného typu. Tento druhý entitní typ nazýváme identifikační vlastník a vztahu, který je spojuje identifikační vztah. Slabý entitní typ má vždy povinné členství ve vztahu k tzv. identifikačnímu vlastníku (jedná se tedy o existenční závislost). Opačné tvrzení neplatí. Nelze o každé existeční závislosti mluvit jako o závislosti identifikační. Tedy existenčně závislá entita ještě není slabá entita.

ISA hierarchie, podtypy entit

Speciální atributy představují v abstraktním modelování takové atributy, které danému typu entity přiřazují jeho nadtyp. Atribut je pak podtypem svého nadtypu. Jde o tzv. ISA-hierarchii.

Transformace ER modelu na relační schéma

Z ER modelu tedy můžeme podle několika pravidel vytvořit relační schéma. Tento proces bude podrobněji popsán v poslední části otázky. Jde především o tyto transformace:

- Entita se transformuje 1. na tabulku
- Každý atribut entity se změní na sloupec tabulky
- Primární klíč entity bude primárním klíčem tabulky
- Provedou se potřebné úpravy, aby bylo relační schéma validní.

Relační model

Relační databázový model důsledně odděluje data, která jsou chápána jako relace, od jejich implementace. Přístup k datům je symetrický, tj. při manipulaci s daty se nezajímáme o přístupové mechanizmy k datům. Pro manipulaci s daty jsou k dispozici dva silné prostředky - relační kalkul a relační algebra. Pro omezení redundance dat v relační databázi jsou navrženy pojmy umožňující normalizovat relace. Od matematické relace se relační model liší v několika aspektech:

- relace je vybavena pomocnou strukturou, které se říká schéma relace. Schéma relace se skládá ze jména relace, jmen atributů a domén
- prvky domén, ze kterých se berou jednotlivé komponenty prvků relace, jsou atomické (dále nedělitelné) hodnoty.

Relační model dat se tedy skládá z těchto částí, které definují relaci:

- jména atributů
- domény atributů
- n-tice atributů
- relace
- schémata relací
- jména schémat relací

Inuitivně (pracovně), ale nepřesně: relace = tabulka, schéma = záhlaví tabulky.

Příklad tabulka kino:

název	adresa
Blaník	Václavské nám.
Mír	Strašnická
Domovina	V Dvorcích

Schéma relací: Kino (název, adresa)

Relační algebra

Relační algebra je nejzákladnějším prostředkem pro práci s relacemi. Jedná se o dotazovací jazyk mezi jehož základní operace patří projekce, selekce a spojení.

- Projekce relace R s položkami A na množinu položek B vytvoří relaci s položkami B a záznamy, které vzniknou z původní tabulky R odstraněním položek A-B. Odstraněny jsou i eventuelně opakující se záznamy. Značí se : R[B] (laicky: jde o výcuc určitých sloupců/položek B z A)
- Selekce relace R s položkami A podle logické podmínky F vytvoří tabulku s týmiž položkami a ponechá ty záznamy z původní tabulky, které splňují logickou podmínku F. Značí se: R(Q) (laicky: jde o výcuc určitých řádek z A, které splňují podmínku F)
- Spojení relací R a S s položkami A a B vytvoří minimalizovanou tabulku se záznamy, jejichž projekce na A je z tabulky R a projekce na B je z tabulky S. Značí se: R * S (laicky: jde o spojení tabulek R a S podle určitého sloupce)

Relační kalkul

Relační kalkul je dotazovací jazyk, který vychází z predikátové logiky 1.řádu a v relačních databázích se vyskytuje ve dvou formách. Jedná se o n-ticový (řádkový) a doménový relační kalkul. Doménový relační kalkul oproti řádkovému pracuje s proměnnými, které nemají za hodnoty n-tice, ale jednotlivé prvky z domén, tj. jednoduché hodnoty atributů.

Integritní omezení

Je nutné zajistit, aby se do relací dostala pouze „správná“ data - přípustné n-tice. Úplná definice relačního schématu je:

(R,I) … schéma relační databáze

R = { R1 ,R2 ,… ,Rk}

I … množina integritních omezení

Přípustná relační databáze se schématem (R,I) je množina relací R1*, R2*, … , Rk* takových, že jejich n-tice vyhovují tvrzením v I. Integritním omezením jsou např. klíče.

Příklad:

KINO(NÁZEV_K, ADRESA),
FILM(JMÉNO_F, HEREC, ROK)
MÁ_NA_PROGRAMU(NÁZEV_K, JMÉNO_F, DATUM)

Integritní omezení:

IO1: primární klíče

IO2: Cizí klíče

IO3: V kinech se nehraje více, něž dvakrát týdně

IO4: Jeden film se nedává více, než ve třech kinech

Podmínky, které musí splňovat relační tabulka:

- všechny hodnoty v tabulce musí být elementární - tzv. Dále nedělitelné - podmínka 1.NF
- sloupce mohou být v libovolném pořadí
- řádky mohou být v libovolném pořadí
- sloupce musí být homogenní = ve sloupci musí být údaje stejného typu
- každému sloupci musí být přiřazeno jednoznačné jméno (tzv. atribut)
- v relační tabulce nesmí být dva zcela stejné řádky. Tzn., že každý řádek je jednoznačně rozlišitelný.

Normalizace schématu formou dekompozice

Normalizace je odstranění redundantních(opakujících) se dat, omezení složitosti (rozložení složité relace na dvojrozměrné tabulky) a zabránění tzv. aktualizačním anomáliím (např. abychom smazáním všech knih autora nepřišli o data o autorovi). Což by mělo vést k databázi přehlednější, rozšiřitelnější a výkonnější.

Normalizace by měla vést k vzniku tabulek, které lze snadno udržovat a efektivně se na ně dotazovat. Normalizované schéma musí zachovat všechny závislosti původního schémat a relace musí zachovat původní data, což znamená, že se musíme pomocí přirozeného spojení dostat k původním datům.

Normální formy:

- 1.NF – První normální forma
- 2.NF – Druhá normální forma
- 3.NF – Třetí normální forma
- BCNF – Boyce Coddova normální forma
- 4.NF – Čtvrtá normální forma
- 5.NF – Pátá normální forma

1. normální forma (1.NF)
Relace je v první normální formě, pokud každý její atribut obsahuje jen atomické hodnoty. Tedy hodnoty z pohledu databáze již dále nedělitelné. Například v relaci obsahující data o nějaké osobě budeme chtít mít více telefonních čísel:

Osoba

Jméno	Přijmení	Adresa	Telefony
Jan	Novák	Havlíčkova 2 Praha 3	125789654;601258987;789456123
Petr	Kovář	Svatoplukova 15 Brno	369852147;357951456;963852741
Pavel	Pavel	Papalášova 25 Kocourkov	546789123;123456789;987456123

S takovouto tabulkou by byla spousta problémů, například by se dost špatně prováděly změny čísel, případně vyhledávání podle telefonního čísla.

Aby tabulka byla v 1NF musíme buďto rozdělit atribut telefon do více atributů (pouze za předpokladu, že jsme si jisti, že se množství telefonních čísel nezvýší), nebo oddělit telefoní čísla do samostatné tabulky, což já osobně preferuji, protože je to podstatně flexibilnější řešení:

Osoba

ID	Jméno	Příjmení	Adresa
1	Jan	Novák	Havlíčkova 2 Praha 3
2	Petr	Kovář	Svatoplukova 15 Brno
3	Pavel	Pavel	Papalášova 25 Kocourkov

Telefon

3.4. Metoda dekompozice a syntézy

Tyto metody jsou vhodné pouze pro jednoduché aplikace. Umožňují přímý návrh logického schématu relační databáze (relačních schémat) na základě znalostí funkčních vztahů. Cílem je již známé kritérium - získání relací ve třetí normální formě.

3.4.1. Metoda dekompozice

Principem je postupná dekompozice relačního schématu až do okamžiku, kdy jsou všechna schémata ve třetí normální formě. Dekompozice se řídí tímto pravidlem:

Mějme schémata R(A, B, C), kde A, B, C jsou množiny atributů, a funkční závislost B $\rightarrow$ C. Rozložíme-li R na schémata R1(B, C) a R2 (A, B), je takto provedená dekompozice bezeztrátová.

Bezeztrátovost zde znamená, že nedochází ke ztrátě informace z původní relace. Spojením nově vytvořených relací vznikne původní relace.

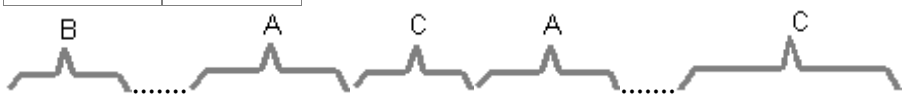
Příklad: Báze dat zachycuje zájmovou realitu tak, jak byla popsána v kapitole **Normalizace a její význam.**

1. Označíme atributy A, B, C a provedeme rozklad dle uvedeného pravidla.

Č.stud.	**Č.před.**	Jméno	Adresa	Hodnocení	Č.oboru	Fakulta

2. V získaných tabulkách hledáme nové atributy A, B, C a provedeme další rozklad.

Č.oboru	Fakulta



Č.stud.	**Č.před.**	Jméno	Adresa	Hodnocení	Č.oboru

3. V takto získaných tabulkách se již další závislosti nenacházejí.

Č.stud.	Jméno	Adresa	Katedra

Č.stud.	**Č.před.**	Hodnocení

3.4.2. Metoda syntézy
Vychází ze zadané množiny atributů a funkčních závislostí:

1.Nalezneme neredundandní pokrytí I' původní funkční množiny I.

2.Rozdělíme I' do skupin takových, že všechny závislosti ve skupině mají stejnou levou stranu a žádné dvě skupiny nemají závislosti se stejnou levou stranou.

3.Pro každé dvě skupiny Si a Sj testujeme, zdali levé strany na sobě vzájemně závisí (jsou-li v uzávěru množiny I'). Jestliže ano, spojíme obě skupiny v jednu a obě závislosti dáme do J. Z vytvořené skupiny odstraníme závislosti tvaru L $\rightarrow$ B, kde L je levá strana závislosti jedné ze dvou skupin a B je levá strana závislosti druhé skupiny.

4.Nalezneme minimální množinu I >1 z I' takovou, že uzávěr IčJ je roven I'č J, odstraníme ze skupin ty závislosti, které ubyly z I' a dodáme nutné ekvivalentí klíče z J.

5.Pro každou skupinu sestrojíme schéma relace s atributy odpovídajícími atributům v závislostech obsažených v té skupině s klíčem rovným levé straně pravidel (resp. s ekvivalentními klíči).

Příklad :

Mějme schéma R ((A, B, C, D, E, F), I) kde I={EF $\rightarrow$ AD, CD $\rightarrow$ EF, AE $\rightarrow$ B, BF $\rightarrow$ C, C $\rightarrow$ A}.

ad 1. Neredundandní pokrytí je stejné jako I, tedy:

I´={EF $\rightarrow$ AD, CD $\rightarrow$ EF, AE $\rightarrow$ B, BF $\rightarrow$ C, C $\rightarrow$ A}

ad 2. V daném případě každý prvek množiny I´ tvoří samostatnou skupinu:

S1:	$EF \rightarrow AD$
S2:	$CD \rightarrow EF$
S3:	$AE \rightarrow B$
S4:	$BF \rightarrow C$
S5:	$C \rightarrow A$

ad 3. Provedeme test na závislost levých stran jednotlivých skupin. Je zřejmé, že závisí pouze skupina S1 a S2, dále postupuje dle bodu 3. Nové skupiny jsou

S1:	$EF \rightarrow AD$
	~~$CD \rightarrow EF$~~
S2:	$AE \rightarrow B$
S3:	$BF \rightarrow C$
S4:	$C \rightarrow A$

$J=\{EF \rightarrow AD, CD \rightarrow EF\}$

ad 4. Dle bodu 4 nalezneme množinu I, upravíme I´ a ze skupin zrušíme stanovené prvky.

$I=\{EF \rightarrow AD, CD \rightarrow EF\}$

z I' tady ubyly $EF \rightarrow AD$, $CD \rightarrow EF$

S1:	$AE \rightarrow B$
S2:	$BF \rightarrow C$
S3:	$C \rightarrow A$

Dodáme ekvivalentní klíče z J, tj. $CD \leftrightarrow EF$

ad 5. Sestavíme relace:

$R1=\{R (E, F, C, D), \{CD \leftrightarrow EF\}\}$

$R2=\{R (A, E, B), \{AE \rightarrow B\}\}$

R3={R (B, F, C), {BF $\rightarrow$ C}}

R4={R (C, A), {C $\rightarrow$ A}}

ID_osoby	Cislo
1	125789654
1	601258987
1	789456123
2	369852147
2	357951456
2	963852741
3	546789123
3	123456789
3	987456123

2.normální forma (2.NF)

Relace se nachází v druhé normální formě, jestliže je v první normální formě a každý neklíčový atribut je plně závislý na primárním klíči, a to na celém klíči a nejen na nějaké jeho podmnožině. Z čehož vyplívá, že druhou normální formu musíme řešit pouze v případě, že máme vícehodnotový primární klíč. Zní to poněkud složitě, ale nic na tom není, opět pomůže příklad:

V tabulce zboží v obchodě bude název zboží, výrobce, telefon na výrobce, cena zboží a množství na skladě.

Sklad

Název	Výrobce	Telefon Výrobce	Cena	Množství
Mléčná čokoláda	Milka	+420123456789	30Kč	2500
Oříšková čokoláda	Milka	+420123456789	30Kč	2800
Tyčinka milkyway	Milka	+420123456789	10Kč	7000
Mléčná čokoláda	Orion	+420987654321	25Kč	5800
Oříšková horalka	Horalka	+420897654321	7Kč	4560

Klíčem této relace je kombinace atributů Název a Výrobce. Telefon výrobce ovšem není závislí na celém klíči, ale pouze na atributu výrobce. To by vedlo k aktualizační anomálii a to k té, že pokud by se vymazaly veškeré výrobky od výrobce Milka, ztratilo by se telefoní číslo na výrobce Milka, což není zrovna žádané. Řešením je opět rozpad na dvě tabulky:

Výrobek

Název	Výrobce_ID	Cena	Množství
Mléčná čokoláda	1	30Kč	2500
Oříšková čokoláda	1	30Kč	2800
Tyčinka milkyway	1	10Kč	7000
Mléčná čokoláda	2	25Kč	5800
Oříšková horalka	3	7Kč	4560

Výrobce

Vyrobce_ID	Vyrobce	Telefon
1	Milka	+420123456789
2	Orion	+420987654321

Vyrobce_ID	Vyrobce	Telefon
3	Horalka	+420897654321

3.normální forma (3.NF)
V této formě se nachází tabulka, splňuje-li předchází dvě formy a žádný z jejich atributů není tranzitivně závislý na klíči. Jiné vyjádření téhož říká, že relace je v 3.NF, pokud je ve 2.NF a všechny neklíčové atributy jsou navzájem nezávislé.

Opět definice, která zní nesrozumitelně, ale její použití je vlastně jednoduché. Tranzitivní závislost je taková závislost, mezi minimálně dvěma atributy a klíčem, kde jeden atribut je funkčně závislý na klíči a druhý atribut je funkčně závislý na prvním.

Koukám, že jsem tomu opět moc nepomohl, takže nejlepší bude příklad:

Řekněme, že firma chce uchovávat informace o zaměstnancích, takže vytvoříme relaci Zaměstnanec s atributy r.č. (primární klíč), Jméno, Příjmení, Město, PSČ, Funkce a Plat, zbytek adresy vynecháme, protože pro příklad není důležitý.

Zaměstnanec

r.č	Jméno	Příjmení	Město	PSČ	Funkce	Plat
1	Jack	Smith	Jihlava	58601	CEO	150000
2	Franta	Vomáčka	Praha10	10000	Senior Software Architect	80000
3	Pepa	František	Plzeň	10000	Senior Software Architect	80000
4	Pavel	Novák	Kocourkov	99999	Junior Developer	30000
5	Petr	Koukal	Praha10	12345	Database Designer	75000
6	Honza	Novák	Plzeň	12345	Junior Developer	30000

Z této tabulky je vidět kromě závislosti všech atributů na klíči ještě závislost PSČ a Města a závislost Platu na Funkci. Aby jsme si to ukázali pomocí obou vyjádření definic. Závislost r.č -> Město -> PSČ je tranzitivní závislost PSČ na klíči, stejně tak závislost r.č. -> Funkce ->Plat. Pochopitelnější je asi druhé vyjádření, podle něj jsou závislosti Město -> PSČ a Funkce ->Plat přesně ty, které porušují sousloví: "všechny neklíčové atributy jsou navzájem nezávislé". Řešením problému je opět rozpad na více relací, v tomto případě dokonce na 3, protože jsme 3.NF porušily rovnou dvakrát.

Zaměstnanec

r.č	Jméno	Příjmení	Město_ID	Funkce_ID
1	Jack	Smith	1	1
2	Franta	Vomáčka	2	2
3	Pepa	František	4	2
4	Pavel	Novák	3	4
5	Petr	Koukal	2	3
6	Honza	Novák	4	4

Město

Město_ID	Město	PSČ
1	Jihlava	58601
2	Praha10	10000
3	Kocourkov	99999
4	Plzeň	12345

Funkce

Funkce_ID	Funkce	Plat
1	CEO	150000
2	Senior Software Architect	80000
3	Database Designer	75000
4	Junior Developer	30000

Boyce Coddova normální forma (BCNF)
Boyce/Coddova normální forma se pokládá za variaci třetí normální formy a dokonce je původní definicí 3.NF tak jak byla publikována v 70 letech. Je vymezena stejnými pravidli jako 3.NF forma, říká, že musí platit i mezi hodnotami uvnitř složeného primárního klíče.

Relace se nachází v BCNF, jestliže pro každou netriviální závislost X -> Y platí, že X je nadmnožinou nějakého klíče schématu R.

Zní to poněkud šíleně, ale ničeho se nebojte, k tomu, aby byla porušena BCNF musí být splněno několik podmínek a to poměrně specifických:

- Relace musí mít více kandidátních klíčů
- Minimálně 2 kandidátní klíče musí být složené z více atributů
- Některé složené kandidátní klíče musí mít společný atribut.

Nejsnáze Boyce/Coddovu normální formu pochopíme s pomocí funkčních závislostí.
Boyce/Coddova normální forma v podstatě říká, že mezi kandidátními klíči nesmí být žádná funkční závislost. Jak známo, nejlépe se definice chápou na příkladech, takže mějme relaci adresář:

Původní příklad byl odstraněn, byl chybný, tento jsem si vypůjčil ze script Databázové systémy, Prof. RNDr. Jaroslav Pokorný CSc., Ing Ivan Halška

Adresa

Město	Ulice	PSČ
Praha 10	Černokostelecká	100 00
Jihlava	Žižkova	58601
Praha 10	Vrátkovská	100 00
Brno	Dvořákova	589 74
Praha 6	Chaloupeckého	160 00

V této relaci platí dvě netriviální funkční závislosti:
{Město,Ulice} -> PSČ a PSČ -> Město
Protože neplatí Ulice -> PSČ ani Město -> PSČ, tvoří dvojice {Město, Ulice} klíč schématu. Klíčem je ale i {Ulice, PSČ} platí totiž PSČ -> Město, nikoliv však PSČ -> Ulice. Tudíž je {PSČ, Ulice} kandidátním klíčem schématu. Schéma má všechny atributy atomické a nemá žádný neklíčový atribut a tudíž je v 3.NF, ale není v BCNF. Tento fakt vede k tomu, že nelze evidovat města s PSČ bez znalosti Ulice a krom toho jsou v relaci redundantní data, pokud by se evidovalo velké množství ulic v jednom městě, začal by to být problém.

Klasické řešení, rozpad na dvě tabulky. Vzhledem k tomu, že neplatí PSČ -> Ulice, musíme spojit PSČ a Ulice. Výsledkem tudíž budou relace Města(PSČ, Město) a Ulice(PSČ, Ulice)

Město

PSČ	Město
100 00	Praha 10
160 00	Praha 6
586 01	Jihlava
Brno	589 74

Adresa

Ulice	PSČ
Černokostelecká	100 00
Vrátkovská	100 00
Dvořákova	586 01
Chaloupeckého	160 00
Dvořákova	589 74

Čtvrtá normální forma (4.NF)

Tabulka je ve čtvrté normální formě, je-li v BCNF a popisuje pouze příčinnou souvislost (jeden fakt). Sice jednoduché vyjádření bez složitých definic, ale poněkud nicneříkající, takže zkusíme jinou definici: " Relace je ve čtvrté normální formě, pokud je v Boyce/Coddově normální formě, a navíc všechny vícehodnotové závislosti jsou zároveň funkčními závislostmi z kandidátních klíčů. " Mno koukám, že jsem tomu moc nepomohl, tak zkusíme definici a příklad ze skript Tvorba datového modelu v prostředí strategických informačních systému, Prof. Ing. Jindřich Kaluža, CSc. : "ve čtvrté normální formě je relace tehdy, je-li v BCNF a všechny vícehodnotové závislosti obsažené v relaci jsou zároveň funkčními závislostmi. Vícehodnotovou závislost atributů lze definovat následovně: V relaci R, která je v BCNF, s atributy A, B, C nastává vícehodnotová závislost atributu B na atributu A právě tehdy, jestliže množina hodnot B přiřazená dvojici hodnot A, C závisí jen na hodnotě atributu A a je nezávislá na hodnotě atributu C."

Tak teď už je to definice přesná a všeříkající, ale bez perfektní znalosti všech použitých pojmů je opět špatně pochopitelná, tudíž příklad si vypůjčím vysvětlení a příklad ze skript Databázové systémy, Vostrovský, Merunka:

Čtvrtá normální forma se zabývá vztahy uvnitř složeného primárního klíč. Pokud je v tabulce složený primární klíč, může se stát, že některé hodnoty tohoto klíče jsou na sobě nezávislé, ale tím, že spolu tvoří klíč, vzniká falešná souvislost mezi těmito hodnotami a nemohou existovat nezávisle na sobě, což není v souladu s modelovanou realitou. 4.NF proto vyžaduje, aby klíč tvořily jen ty hodnoty, které mají skutečnou vzájemnou souvislost.

Mějme relaci zachycující vztah zaměstnance, kvalifikace a úkolu: Pracovní zařazení(Zaměstnanec, Úkol, Kvalifikace)

Pracovní zařazení

Zaměstnanec	Úkol	Kvalifikace
Ing Petr Pastyňák	Tvorba webu	Webdeveloper
Ing PetrPastyňák	Návrh databáze podnikového IS	Database Specialist
Eva Petrželová	Asistentka Ing Pastyňáka	Psaní na stroji
Eva Petrželová	Asistentka Pastyňáka	ECDL
Pavel Mrkvička	Analytik podnikového IS	Aanalyst
Pavel Mrkvička	Analytik podnikového IS	UML

Všechny atributy dohromady tvoří klíč schématu a neexistuje mezi nimi žádná funkční závislost, tudíž je v BCNF a všechno vypadá ideálně, ale není tomu tak. I když se dá předpokládat, že atributy Kvalifikace a Úkol jsou na sobě nezávislé, tak tabulka neumožňuje zachytit kvalifikaci zaměstnance, který nemá přiřazen žádný úkol (a úkolujte někoho o kom netušíte co umí) a nelze ani úkolovat zaměstnance bez kvalifikace. Krom ztráty informací se rozkladem vyvarujeme i redundance dat. Tudíž je opět nutno tabulku rozdělit a to na dvojici: Kvalifikace (Zaměstnanec, Kvalifikace), Úkol (Zaměstnanec, Úkol).

Kvalifikace

Zaměstnanec	Kvalifikace
Ing Petr Pastyňák	Webdeveloper
Ing Petr Pastyňák	Database Specialist
Eva Petrželová	Psaní na stroji
Eva Petrželová	ECDL
Pavel Mrkvička	Aanalyst
Pavel Mrkvička	UML
Ing Petr Cibula	Project manager
Ing Petr Cibula	RUP Specialist

Úkol

Zaměstnanec	Úkol
Ing Petr Pastyňák	Tvorba webu
Ing Petr Pastyňák	Návrh databáze podnikového IS
Eva Petrželová	Asistentka Ing Pastyňáka
Pavel Mrkvička	Analytik podnikového IS
Jan Celer	Kopání odvodňovacího kanálu

Do rozložených relací jsem záměrně přidal data, která v původní relaci nebyla, ale měla by být. Krásně se tím ukazuje, jak snadné je teď najít project m,anagera na tvorbu podnikového IS, ale zkuste si to v nenormalizované tabulce, když pan Cibula zrovna nemá přidělen žádný úkol.

Pátá normální forma (5.NF)
Relace je v páté normální formě, pokud je ve čtvrté a není možné do ní přidat další atribut (skupinu atributů) tak, aby se vlivem skrytých závislostí rozpadla na několik dílčích relací.

A už je to tu zase, poměrně normálně znějící definice, ale opět docela naprd. Takže zkusíme jinou:

Relace je v páté normální formě jestliže je ve 4NF a nemůže-li být dále bezeztrátově rozložena. Jinými slovy relace, která má n klíčových atributů (n >= 3) a která se rozloží na relace o n-1 klíčových atributech, nemůže být opětovně spojena operací přirozeného spojení do jedné relace, aniž by došlo ke ztrátě informace.

To už začíná být trošku lepší, ale zkusme to ještě jednou trošku jinak:

Pátá normální forma se týká primárních klíčů, které jsou tvořeny nejméně třemi atributy. V případě, že mezi těmito hodnotami v klíči existují párové cyklické závislosti, tak je třeba tyto závislosti extrahovat do samostatných tabulek, ale původní tabulku je v některých případech třeba zachovat!

To byli definice, teď zkusím trošku jiný popis. K porušení 5NF musí opět být splněno několik podmínek a to dost specifických. Relace musí být ve 4NF a musí mít klíč složený z třech nebo více atributů a mezi nimi musí být párové cyklické závislosti, ale nikoliv funkční, ani multizávislosti, to by nebyla ve 4NF. Typicky se jedná o vztah třech a více tabulek, kde platí vztahy M:N:O:M a tento vztah je vytvořen jednou relací. 5NF řeší redundanci dat a možnou ztrátu závislostí.

Myslím, že příklad opět pomůže. Mějme firmu, která provozuje síť obchodních zástupců strojírenských firem pro celou Evropu. Ta potřebuje vědět, který zástupce zastupuje kterou firmu a v jakých státech a ve kterých státech působí firmy. Předpokládejme, že o Zástupcích, Firmách i Státech máme vytvořeny informační relace a použité hodnoty jsou pouze cizí klíče, kterými řešíme vztahy mezi těmito relacemi. Zdánlivě jednoduché:

Obchodní zastoupení

Zástupce	Firma	Stát
Antonín Bahel	Siemens	Německo

Zástupce	Firma	Stát
Antonín Bahel	Siemens	Rakousko
Ctirad Drba	Siemens	Francie
Ctirad Drba	Škoda Plzeň	Rakousko
Antonín Bahel	Škoda Plzeň	Norsko

Problém vypadá na první pohled vyřešeně, ale dle naší definice páté normální formy tomu tak není, neboť zde existují závislosti Zástupce-> Firma -> Sát -> Zástupce a to jsou párové cyklické závislosti. Mohlo by se stát, že s vymazáním obchodního zástupce, by se mohlo ztratit informace o tom, že firma prodává v zemi, kde jí zastupoval pouze ten smazaný zástupce a to je pochopitelně nežádoucí. Stejně tak odebrání firmy může způsobit ztrátu informace o působení obchodního zástupce v některé zemi a to je taktéž nežádoucí. Takže musíme provést rozpad tři relace, které nám pokryjí všechny vztahy.

Pusobi

Zástupce	Stát
Antonín Bahel	Německo
Antonín Bahel	Rakousko
Ctirad Drba	Francie
Ctirad Drba	Rakousko
Antonín Bahel	Norsko

Zastupuje

Zástupce	Firma
Antonín Bahel	Siemens
Ctirad Drba	Siemens
Ctirad Drba	Škoda Plzeň
Antonín Bahel	Škoda Plzeň

Zastoupeni

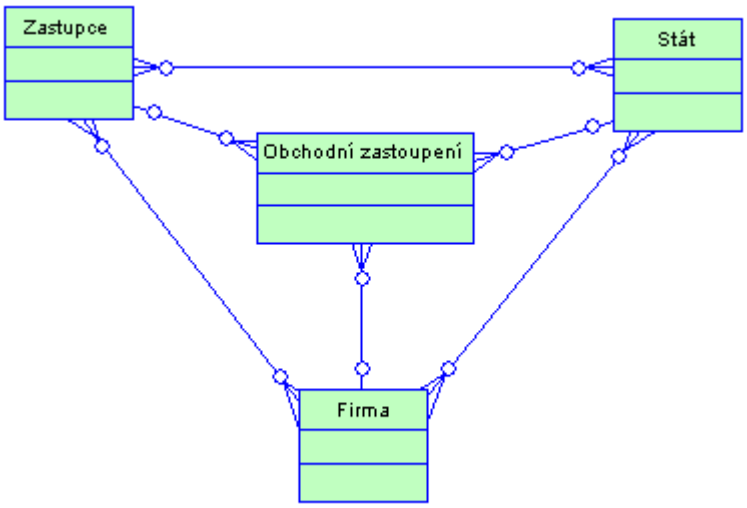
Firma	Stát
Siemens	Německo
Siemens	Rakousko
Siemens	Francie
Škoda Plzeň	Rakousko
Škoda Plzeň	Norsko

Zdá se, že problém je vyřešen, nicméně není. Jedna z definic říká, že relace je v páté normální formě pokud již nelze bezeztrátově rozdělit a menší relace. Důležité je slovíčko bezeztrátově. Protože pokud si spojíme výsledné tabulky pomocí přirozeného spojení, nedostaneme původní výsledek. Dostaneme úplně jiné informace.

Takže jak z toho, tříatributová relace není dobře, tři dvouatributové jsou taky špatně. A co takhle nechat oboje? Ve své podstatě udržuje každá relace jinou informaci. Zastupuje nám říká, které firmy kdo zastupuje, Pusobi říká, kde nám pracuji zástupci a Zastoupeni říká, kam prodávají firmy a ObchodniZastoupeni, říká kdo koho kde.

Pátá normální forma v tomto příkladu nebyla ani tak o špatném převodu konceptu do fyzického modelu databáze, jako spíš o neuvědomění si skutečných vztahů. Ve své podstatě jsem se snažili vymodelovat tuto situaci:

Schéma databázového modelu



Normalizovat je určitě potřeba a čím složitější databáze a čím více dat, tím více je potřeba normalizovat. Ale i tady platí všeho s mírou. Například u příkladu u 3.NF by firma s několika desítkami zaměstnanců asi neměla potřebu dávat PSČ do další tabulky a bylo by to zbytečné. Ale v tabulce zákazníků některého z mobilních operátorů s milióny zákazníků to už význam určitě má.

Kritéria kvality dekompozice

- bezztrátovost
- odstranění renundancí
- rychlost
- úspora datového prostoru

Kvalita dekompozice je určena typen normální formy databáze. Čím vyšší NF tím kvalitnější dekompozice. Neplatí vždy. Někrerá kritéria jdou proti sobě.

Ještě jednou:

- předpokládejme, že máme relaci, kerá není v požadované normální formě
- je potřeba tuto relaci restrukturalizovat na množinu normalizovaných relací
- tato restrukturalizace obvykle zahrnuje rozdělení původní relace do několika menších normalizovaných relací
- tento proces je nazýván dekompozice
- dekompozici je potřeba provádět pečlivě:
    - dekompozice by neměla způsobit ztrátu informace
    - mělo by být možné zkontrolovat integritní omezení v dekomponované verzi stejně snadno jako v původní verzi

Příklad: Dostaneme relaci REZERVACE = (ID_NAMORNIKA, JMENO_NAMORNIKA, ID_LODE, DEN)
Víme, že existuje funkční závislost ID_NAMORNIKA → JMENO_NAMORNIKA. Je tato relace ve 3 normální formě?
Není, protože existuje tranzitivní závislost způsobená závislostí ID_NAMORNIKA →

JMENO_NAMORNIKA.

Předpokládejme, že bychom původní relaci REZERVACE dekomponovali do relací R1 = (ID_NAMORNIKA, JMENO_NAMORNIKA) a R2 = (ID_LODE, DEN).

Jsou tyto relace ve 3. normální formě?

Ano (všechny neklíčové atributy jsou závislé jen a jen na klíči, nejsou závislé vzájemně; předtím to neplatilo, protože ID_LODE A DEN jsou tranzitivně závislé na ID_NAMORNIKA - přes JMENO_NAMORNIKA), ale nyní nezjistíme, kdo si zarezervoval kterou loď… došlo ke ztrátě informace.

Pozn.: v příkladu se patrně předpokládá, že jméno námořníka je jedinečné a tedy také jednineč ně určuje id lodě.

Vlastnost bezztrátového spojení

Definice: Pokud je relace R dekomponovaná do dvou částí X a Y takových, že

$X \subseteq R$ a $Y \subseteq R$

dekompozice má vlastnost bezztrátového spojení, pokud pro každou platnou instanci r relace R platí:

$$\pi_X(r) \bowtie \pi_Y(r) = r$$

- jinými slovy, dekompozice má vlastnost bezztrátového spojení pouze v případě, že jsme schopni zpětně rekonstruovat původní relaci prostřednictvím operace join
- všimněte si, že instance r musí splňovat všechny funkční závislosti, které platí pro relaci R

Poznámka: Ve výše uvedené definici se vyskytuje vzorec, který si jistě zaslouží drobný komentář. Symbolem „Pí“ s dolním indexem X označujeme instanci, která vznikla z původní instance projekcí na množinu atributů relace X (při této operaci může dojít k odstranění duplicitních řádků). Analogicky toto platí pro druhé „Pí“ s indexem Y. Symbol mezi těmito znaky je operátor označující přirozené spojení. Celá definice vlastně řeší, co se stane, když znovu spojíme nově vzniklé instance přirozeným spojením (přes společné atributy). Pokud je spojení bezztrátové, získáme původní instanci . Pokud byla dekompozice provedena špatně, můžeme získat více nebo méně záznamů (v obou případech se jedná o jev, způsobený ztrátou informace - i když se zdá, že máme řádky navíc, jsou to řádky, o kterých nemáme dostatečnou informaci).

Příklad: Mějme nějakou instanci r z R (reprezentovanou tabulkou):

ID_NAMORNIKA	JMENO_NAMORNIKA	ID_LODI	DEN
101	Lubby	103	03/05/96
101	Lubby	102	06/05/96
102	Dennis	102	17/05/96
103	Rusty	104	13/05/96

Jak můžeme relaci R dekomponovat do dvou relací, které budou ve 3. normální formě a neztratit přitom žádnou informaci?

Provedeme dekompozici relace R na relace:

R_1 = (ID_NAMORNIKA, JMENO_NAMORNIKA)

R_2 = (ID_NAMORNIKA, ID_LODI, DEN)

Pro instanci r získáme pro každou z nově vzniklých relací R_1, R_2 po jedné nové instanci:

Instance $\pi_{R_1}(r)$:

ID_NAMORNIKA	JMENO_NAMORNIKA
101	Lubby
102	Dennis
103	Rusty

Instance $\pi_{R_2}(r)$:

ID_NAMORNIKA	ID_LODE	DEN
101	103	03/05/96
102	102	06/05/96
102	102	17/05/96
103	104	13/05/96

Spojením $\pi_{R_1}(r) \bowtie \pi_{R_2}(r)$ získáme zpět instanci **r**.

Je zřejmé, že takto to bude fungovat pro libovolnou instanci r relace R.

Věta: Nechť F je množina funkčních závislostí platných pro relaci R. Dekompozice relace R na relace s množinami atributů $R_1 \subseteq R$ a $R_2 \subseteq R$

má vlastnost bezztrátového spojení právě tehdy když uzávěr funkčních závislostí F+ zahrnuje alespoň jednu z následujících funkčních závislostí:

$R_1 \cap R_2 \rightarrow R_1$,

$R_1 \cap R_2 \rightarrow R_2$.

Jinými slovy, atributy společné v R_1 a R_2 musí být klíčem buď v R_1 nebo R_2.

Příklad: V předchozím příkladě je $R_1 \cap R_2$ = {ID_NAMORNIKA}.

Existuje funkční závislost ID_NAMORNIKA → {ID_NAMORNIKA, JMENO_NAMORNIKA} a současně R_2= (ID_NAMORNIKA, JMENO_NAMORNIKA).

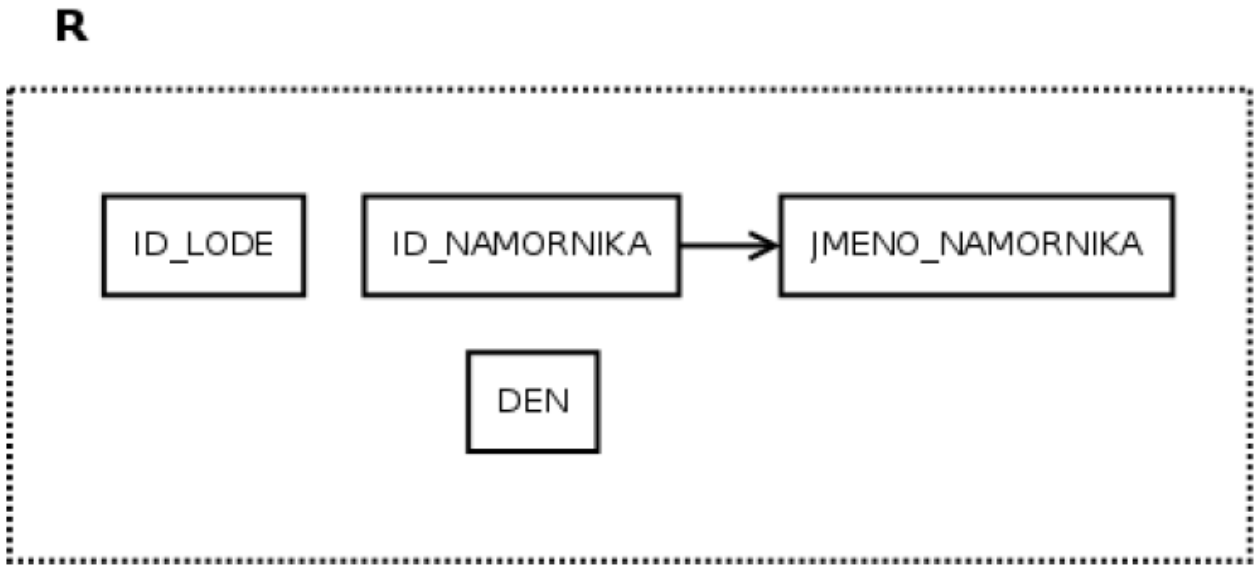
Platí tedy výše uvedená věta (všimněme si, že $R_1 \cap R_2 \rightarrow R_2$) a dekompozice má tedy vlastnost bezztrátového spojení.

Poznámka: Proč platí předchozí věta? Jednoduše řečeno, podmínky věty zajišťují, že atributy účastnící se v přirozeném spojení (R1∩ R2) jsou kandidátním klíčem pro alespoň jednu z uvedených dvou relací. Tím je zajištěno, že se nikdy nemůžeme dostat do situace, kdy by se nám vygenerovaly

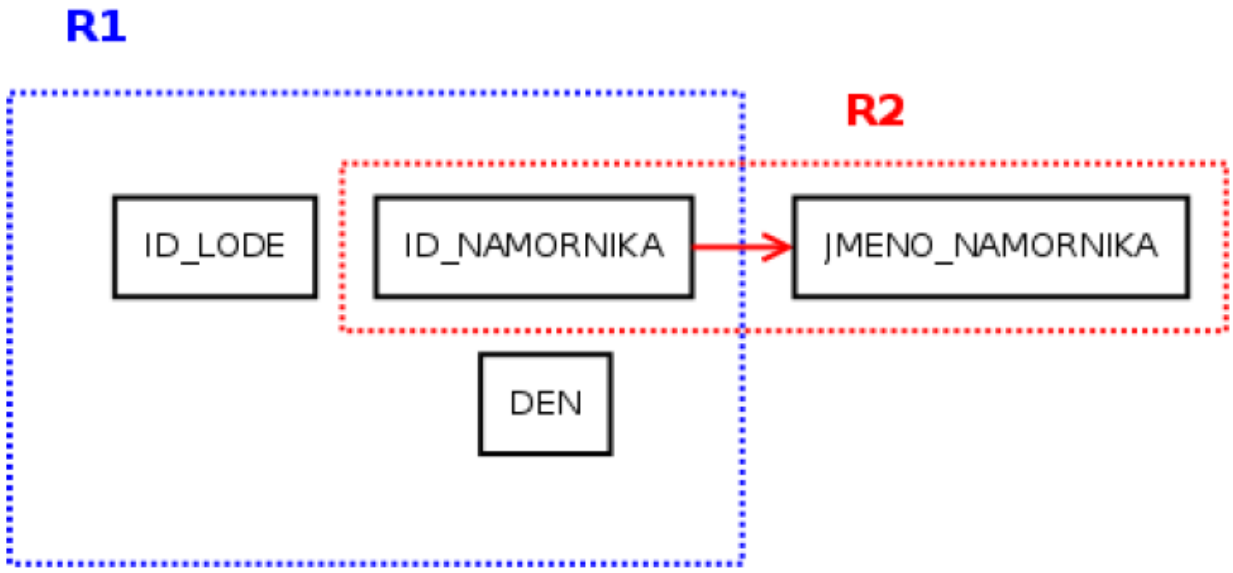
„falešné“ n-tice, protože pro každou hodnotu atributů přes které se provádí spojení, zde bude jedinečná n-tice v alespoň jedné z relací.

Grafické znázornění dekompozice

Výše zmíněný příklad můžeme graficky znázornit. Situace před dekompozicí vypadala takto:

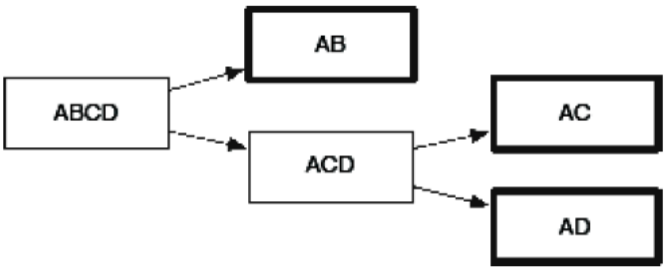


Situace po dekompozici:



Jak dlouho lze provádět dekompozici

Příklad: Dostaneme relaci R = (A, B, C, D) a množinu funkčních závislostí F = {A → BCD}. Jak dalece můžeme provádět dekompozici relace R, tak aby tato dekompozice měla vlastnost bezztrátového spojení?



Poznámky:

- všech normálních forem včetně BNCF lze dosáhnout prostřednictvím dekompozice mající vlastnost bezztrátového spojení
- pokud tedy již nemůže být relace dále dekomponována tak, aby nedošlo ke ztrátě informace (nelze provést dekompozici s vlastností bezztrátového spojení), je zaručeně v BCNF
- ale pozor: to, že lze dosáhnout BNCF úplnou dekompozicí neznamená, že je to vždy potřeba, BCNF může být dosaženo již v některé dřívější fázi dekompozice

Algoritmus pro provádění dekompozice do 3NF/BCNF

def decompose(R, F^+):

- nechť A je některý atribut z R
- nechť X R
- if v množině F+ existuje funkční závislost (X → A), která porušuje 3NF/BCNF:

R_1= R - A

R_2= XA

decompose(R_1, F^+)

decompose(R_2, F^+)

else:

done() # hotovo

Poznámka: R označuje relaci, kterou dekomponujeme. F^+ značí uzávěr funkčních závislostí (viz následující příklad).

Příklad: Dostali jsme relaci R = (A, B, C, D, E) a F = {A → B, B → AE, AC → D}.

Klíče jsou AC, BC.

První průchod (R):

A → B porušuje BCNF v R.

Dekomponujeme R na R_1= (A, C, D, E) a R_2= (A, B)

R_2 je nyní v BCNF, takže zde už nemáme co na práci. Zato R_1 vyžaduje další dekompozici.

Druhý průchod (R_1):

A → E (tranzitivně přes B) porušuje BCNF v R_1.

Dekomponujeme na R_{11}= (A, C, D) a R_{12}= (A, E)

Jak R_{11}, tak R_{12} jsou nyní v BCNF. Jsme tedy hotovi.

Finální dekompozice je tedy: R_{11}= (A, C, D), R_{12}= (A, E), R_2= (A, B)

Kritéria kvality dekompozice

Naše požadavky na kvalitu jsou:

- výsledná schémata by měla mít stejnou sémantiku (význam) (podprobněji viz a))
- nová relace by měla obsahovat stejná data data, jako obsahovala původní relace (podrobněji viz b))

a) pokrytí závislostí

Tomuto požadavku se také někdy říká „pokrytí původní množiny vlastností“. Cílem je aby původní schéma a schémata získaná dekompozicí nějak odrážela stejné vlastnosti. Důsledkem porušení je chudší sémantika. Vlastnosti to jsou vlastně funkční závislosti (označeny F). Musí tedy platit:

$F^+ = \cup F_i^+$

F jsou funkční závislosti v R

$^+$ značí uzávěr (Matematicky: Uzávěr je nejmenší uzavřená množina, která danou množinu obsahuje. Uzavřená množina je taková množina, jejíž doplněk je otevřená množina. Množina je otevřená, pokud s každým bodem X, který do ní patří, patří do této množiny i jeho okolí.).

Pokud toto platí, říkáme, že R „má vlastnost pokrytí závislostí“. To znamená, že vezmeme-li funkční závislosti v jednotlivých R_i (schématech vzniklých dekompozicí) a uděláme jejich uzávěr, měli bychom dostat totéž jako když uděláme uzávěr z F (tedy funkční závislosti v původním schématu).

Příklad: Máme-li například tabulku ADRESAR(MESTO, ULICE, DUM, PSC), jsou zde dvě netriviální závislosti: {město, ulice} → psc a psc → mesto.

Schéma chceme normalizovat, proto uděláme dekompozici: tabulka_ulic(ulice,dům,psc) a tabulka_měst(město,psc).

V dekomponovaném schématu můžeme opět najít závislost psc → mesto, ale už nemůžeme ověřit {město,ulice} → psc, což je špatně, protože jsme tak ztratili část vlastností v původním schématu.

b) bezztrátové spojení

Nové spojení by měly obsahovat stejná data jako obsahovala původní relace. Dekompozici lze považovat za několik projekcí původní relace na množiny atributů nových schémat. Pro každou přípustnou relaci S* by tedy mělo platit:

$S^* = * S_i^*[A_i]$

* na pravé straně výrazu značí spojeni

S^* je relace podle schématu S

To znamená, že (Poučka:) S^* se dá rekonstruovat pomocí přirozeného spojení projekcí na atributy jednotlivých relací dekompozice. Lidsky řečeno: původní relace (tj. S^*) můžeme získat tak, že spojíme (tj. *) projekce na atributy (tj. to $S_i^*[A_i]$, což si můžeme představit jako výsledek JOIN), které vznikly v jednotlivých „pod-schématech“ vzniklých dekompozicí („jednotlivé relace dekompozice“). Pokud toto platí říkáme, že dekompozice „má vlastnost bezztrátového spojení“

Příklad: Máme tabulku ZNAMKY(predmet,student,znamka), tedy to je to S^* a tu dekomponujeme na:

ZAPIS(předmět,student) a ZNAMKOVANI(předmět,znamka)

Na první pohled je vidět, že ztratíme informaci o tom, jakou který student dostal známku. Přestože možných kombinací máme více, nevíme která platí, takže informací máme méně. Důsledkem, že není daná dekompozice bezztrátová je, že ztratíme závislost mezi data – musíme tedy provést dekompozici dostatečně smysluplně.

Poučky:

Máme schéma R(A,B,C) a A,B,C jsou disjunktní (tj. různé, výlučné..) množiny atributů a funkční závislost B→C. Rozložíme-li R na schémata R1(B,C) a R2(A,B) je tato dekompozice bezeztrátová.

Z toho plyne:

Je-li dekompozice R1(B,C) a R2(A,B) bezztrátová, musí platit buď B→C nebo B→A.

Normální formy

Proces normalizace se při návrhu databáze dělá proto, abychom dosáhli co nejvyšší výkonnosti při použití co nejúspornějších zdrojů. Normalizace databáze organizuje data podle jejich významu, je méně náchylná k problémům, snadněji upravitelná, redukuje přebytečná a opakovaně zadávaná data.

Máme pět normálních forem (a BCNF, což je varianta 3.). Obvykle normalizujeme do 3. normální normy (případně BCNF), normalizace probíhá postupně od 1. normální formy k vyšší tzn. chceme-li 3. musíme mít 1. a 2. splněnu!!! Ještě existuje 4. a 5. normální forma, ale v praxi se nepoužívají a často se ani neuvádí.

1. normální forma (1NF)

Každý atribut obsahuje pouze atomické hodnoty.

Atomické = dále nedělitelné (z pohledu databáze). Typickým příkladem je adresa: namísto sloupce adresa „Nerudova 123, Praha 4“ musíme mít sloupce ulice, čp, město, psč, atd.. Má to hned několik praktických důvodu – nelze například vyhledávat nebo seřadit výsledek dotazu podle jednotlivých částí adresy.

Důležitá je i poznámka, že hodnoty mají být atomické z pohledu databáze viz. například datum nemusíme rozdělit na den, měsíc, rok atd, jelikož datum je v databázi atomická hodnota (existuje datový typ datum) – lidsky řečeno, tedy jeden atribut (sloupec) by měl obsahovat právě jednu hodnotu databázového typu. To je zároveň nejjednodušší i nejobtížnější myšlenka datového modelování.

Tato tabulka není 1NF

Jména	Příjmení	Adresa
Jan	Novák	Ostravská 16, Praha 16000
Petr	Nový	Svitavská 8, Brno 61400
Pepa	Vojtak	Znojemská 1, Beroun

Tato tabulka už je v 1NF

Jména	Příjmení	Ulice	č.p.	Město	PSČ
Jan	Novák	Ostravská	16	Praha	16000
Petr	Nový	Svitavská	8	Brno	61400
Pepa	Vojtak	Znojemská	1	Beroun	26601

2. normální forma (2NF)

Každý neklíčový atribut je plně závislý na primárním klíči

Toto znamená, že se nesmí v řádku tabulky objevit položka, která by byla závislá jen na části primárního klíče. Z definice vyplývá, že problém 2NF se týká jenom tabulek, kde volíme za primární klíč více položek než jednu. Jinými slovy, pokud má tabulka jako primární klíč jenom jeden sloupec, pak 2NF je splněna triviálně.

Tato tabulka není v 2NF

číslo_zamestance	Jméno	Příjmení	číslo_pracovny	název pracoviště
1	Jan	Novák	10	studovna
2	Petr	Nový	15	posluchárna
3	Pepa	Vojtak	10	studovna

Jako primární klíč v této tabulce nemůže zvolit pouze číslo_zamestance, protože název pracoviště není závislý na ID zaměstnance, nemůžeme zvolit ani dvojci (číslo_zamestance,číslo_pracovny), protože jméno, příjmení a název pracovny nejsou plně závislé na dvojci zvoleného klíče. Lidsky řečeno: jméno, příjmení jsou závislé na číslo_zamestanace, zatímco název pracovny je závislý pouze na čísle pracovny a nezávisí vůbec na jménu zaměstnance. Není tedy možné docílit 2NF v jedné tabulce – musíme je rozdělit – odborně se tomu říká „dekompozice relačního schématu“.

Toto schéma už je v 2NF

číslo_zamestance	Jméno	Příjmení	číslo_pracovny
1	Jan	Novák	10
2	Petr	Nový	15
3	Pepa	Vojtak	10

číslo_pracovny	název pracoviště
10	studovna
15	posluchárna

10	studovna

3. normální forma (3NF)

Žádný atribut není tranzitivně závislý na klíči.

Jiná definice (stejný význam): Všechny neklíčové atributy musí být vzájemně nezávislé. Myslim, že na škole se spíš preferuje první definice, takže vysvětlení: Tranzitivní závislost je taková závislost, mezi minimálně dvěma atributy a klíčem, kde jeden atribut je funkčně závislý na klíči a druhý atribut je funkčně závislý na prvním.

Tato tabulka není v 3NF

id	Jména	Příjmení	funkce	plat
1	Jan	Novák	programátor	35 000
2	Petr	Nový	technik	25 000
3	Pepa	Vojtak	vedoucí	75 000

Předpokládejme, že podle funkce je daný plat. V této tabulce je tedy na id závislá funkce, a plat je závislý na funkci, takže plat je transitivně závislý na id. Řešením je stejně jako v 2NF dekompozice.

Toto schéma je v 3NF

id	Jména	Příjmení	funkce
1	Jan	Novák	programátor
2	Petr	Nový	technik
3	Pepa	Vojtak	vedoucí

funkce	plat
programátor	35 000
technik	25 000
vedoucí	75 000

Rozdíl mezi 2NF a 3NF je v tom, že 2NF řeší situaci, kdy je primární klíč složený z více atributů zatímco 3NF řeší i to je-li primární klíč jeden atribut. Radši ještě jeden příklad: následující tabulka vyhovuje 2NF, ale ne 3NF.

Tato tabulka splňuje 2NF, ale není v 3NF

Turnaj	Rok	Vítěz	Datum narození
Indiana Invitational	1998	Al Fredrickson	21. června 1975
Cleveland Open	1999	Bob Albertson	28. září 1968
Des Moines Masters	1999	Al Fredrickson	21. července 1975
Indiana Invitational	1999	Chip Masterson	14. března 1977

Máme složený klíč (název_turnaje,rok_turnaje), který unikátně identifikuje řádku. 3NF je narušena tím, že neklíčový atribut datum_narozeni vítěze je přes neklíčovy atribut vítěz závislý na klíči (turnaji). Náprava:

Toto schéma je v 3NF

Turnaj	Rok	Vítěz
Indiana Invitational	1998	Al Fredrickson
Cleveland Open	1999	Bob Albertson
Des Moines Masters	1999	Al Fredrickson
Indiana Invitational	1999	Chip Masterson

Vítěz	Datum narození
Al Fredrickson	21. června 1975
Bob Albertson	28. září 1968
Al Fredrickson	21. července 1975
Chip Masterson	14. března 1977

Ještě se hodí poznamenat, že pokud bychom měli závislost klíč → klíč → neklíč nebo byly všechny atributy součástí nějakého klíče je schéma také v 3NF. Jinak 3NF je v praxi často dostačující tj. nejsou zde (většinou) aktualizační anomálie ani redundance (nemusí platit vždy viz. BCNF).

Boyce-Coddova normální forma (BCNF)

Školní definice: Schéma R je v BCNF, jestliže pro každou netriviální závislost A→B platí, že A obsahuje klíč schématu R.

Trochu upravená: Tabulka splňuje BCNF, právě když pro dvě množiny atributů A a B; A→B a současně B není podmnožinou A platí, že množina A obsahuje primární klíč tabulky.

Boyce/Coddova normální forma se pokládá za variaci třetí normální formy. V podstatě je vymezena stejnými pravidly jako 3NF forma, říká, že musí platit i mezi hodnotami uvnitř složeného primárního klíče. Tj. aby byla porušena musí platit tyto podmínky:

- Relace musí mít více kandidátních klíčů
- Minimálně 2 kandidátní klíče musí být složené z více atributů
- Některé složené kandidátní klíče musí mít společný atribut.

Pro vysvětlení se podívejme na příklad:

Toto schéma není v BCNF

Přednáška	Učitel	Místnost	Čas
Systémy	Vomáčka	M1	Ut3
Programování	Kryl	M3	St2
Programování	Kryl	M2	Pa4

Možné klíče jsou tady hodina-učitel, hodina-místnost a hodina přednáška - všechny atributy jsou součástí nějakého klíče tudíž je to schéma v 3NF, ale není v BCNF, protože učitel je závislý na přednášce (závislost mezi podklíči). Také je vidět, že přesto, že je schéma v 3NF, je zde redundance – informace, kdo učí kterou přednášku (předpokládejme, že to jeden předmět – jeden přednášející). Řešením je opět dekompozice:

Toto schéma už je v BCNF

Přednáška	Místnost	Čas
Systémy	M1	Ut3
Programování	M3	St2
Programování	M2	Pa4

Přednáška	Učitel
Systémy	Vomáčka
Programování	Kryl

Touto úpravou jsme odstranili redundanci přednáška – učitel a zároveň jsme neztratili informaci kdo co učí a kdy.

Každé schéma, které je v BCNF je také v 3NF, obráceně to ale neplatí. Má-li ale schéma jediný klíč nebo jednoduché klíče, potom je-li v 3NF je i v BCNF.

Návrh schématu relační databáze přímou transformací z konceptuálního schématu

Postup transformace je dnes již natolik rutinní, že je zabudován do téměř všech CASE nástrojů. Nicméně je dobré vědět, co se přitom děje, abychom rozuměli, a abychom eventuálně mohli zvolit jinou možnost, pokud se to pro naši konkrétní aplikační oblast lépe hodí. Mnohé CASE nástroje pro některé prvky modelu nabízejí možnost volby, co s nimi při generování relačního schématu udělat, jiné tu možnost nenabízejí.

Postup transformace lze popsat v následujících krocích.

1. Každý složený atribut rozložte do složek, opakujte tak dlouho, až není dalších složených atributů.
2. Vícehodnotové atributy převeďte na vztah k "hodnotovému" entitnímu typu představujícímu doménu atributu.

3. Pro každý atribut vyberte nejvhodnější datový typ.
4. Rozhodněte o primárních klíčích pro entitní typy.
   Volba primárních klíčů úzce souvisí s efektivitou ukládání dat a provozu relační databáze. To proto, že v relačních databázích primární klíče tabulek slouží k provazování záznamů cizími klíči, a k podpoře efektivity realizace těchto vazeb se používá technologie indexování. Její efektivita je závislá na datové velikosti klíče, a na jeho stabilitě.
   Protože jiné sledovatelné účely, jako je podpora vyhledávání na základě sémantických identifikátorů, lze naplnit nezávislými prostředky, je v současné době tendence navrhovat pro primární klíče tabulek nevýznamové umělé identifikátory (často pojmenovávané ID), a ostatní alternativní klíče definovat jako další unikátní sloupce. Završením této tendence je možnost v moderních objektově-relačních databázích přiřazovat záznamům skryté identifikátory, jejichž hodnota není dostupná nikomu kromě databázového systému.
5. Rozhodněte o všech ISA vztazích (tj. o dědičnosti), co se s nimi má udělat. Pro každou typovou hierarchii připadají do úvahy 3 možnosti:
   - Absorpce do nadtypu. Bude *jedna tabulka, ve které bude vše*. Specifické atributy podtypů vytvoří nepovinné sloupce v této tabulce. – Tato volba je vhodná v případě, když nemáme žádný důvod mít pro podtypy zvláštní tabulky. Nevýhodou je, že vznikají sloupce s významným množstvím NULL hodnot, a je nutno eventuálně definovat složité integritní podmínky pro řádky tabulky.
   - Rozdělení do podtypů. *Každý podtyp bude tvořit jednu tabulku*, ve které bude všechno pro tento podtyp, včetně zděděných vlastností. Takže nebude žádná tabulka pro nadtyp. – Tato volba je vhodná v případě, že nepotřebujeme tabulku pro nadtyp, a podtypy tvoří členění, tj. nepřekrývají se a vyčerpávají všechny případy z nadtypu.
   - Separace vlastností. Bude *jedna tabulka pro nadtyp, a pro každý podtyp další tabulka* se sloupci specifickými pro tento podtyp a s cizím klíčem ukazujícím na "mateřský" záznam v tabulce nadtypu. Tyto cizí klíče budou zároveň unikátní v tabulkách podtypu. Pokud měl některý podtyp jiný identifikátor, než nadtyp, definujte v tabulce tohoto podtypu alternativní klíče. – Tato volba je vhodná v případě, pokud potřebujeme jak tabulku pro nadtyp (například pro nějakou kontrolu), a pro podtypy máme specifická pravidla či specifické vztahy.
6. Rozhodněte o všech vztazích 1:1, co s nimi. (Ryzí 1:1 jsou vzácné, častější je případ, že takový vztah může někdy nabýt kardinality 1:n.). Opět jsou 3 možnosti:
   - Dominantní role. Vztah bude mapován k jednomu entitnímu typu v tabulce tohoto entitního typu, jako cizí klíč odkazující do tabulky příslušející k tomu druhému entitnímu typu. – Tato volba je nejčastější, protože jedna z rolí bývá tzv. *dominantní*: Například vedoucí oddělení hraje dominantní roli vůči oddělení: v běžné situaci má každé oddělení vedoucího (má jednoho a ne více vedoucích), zato většina pracovníků nejsou vedoucími oddělení. Takže vztah "vede" mezi oddělením a pracovníkem bude mapován jako cizí klíč do tabulky pro oddělení. Na tomto příkladu si můžeme uvědomit, že ve výjimečných případech může jeden pracovník vést více oddělení, takže tento vztah není ryzí 1:1, ačkoli typicky ano. – Pokud chceme zajistit ryzí kardinalitu 1:1, musíme požadovat unikátnost pro sloupec s příslušným cizím klíčem (v tomto příkladu v tabulce oddělení unikátnost pro sloupec odkazující na vedoucího).
     Pokud má vztah atributy, mapujeme je do tabulky odpovídající ne-dominantní roli jako další sloupce. Například datum, odkdy je tento pracovník vedoucím toho oddělení, bude jako další sloupec v tabulce oddělení.
   - Další tabulka. Bude vytvořena *další tabulka pro ten 1:1 vztah*. Ta bude tedy mít dva cizí klíče, jeden odkazující do tabulky pro první entitní typ, druhý do tabulky pro druhý entitní typ. Pokud chceme zajistit skutečně 1:1, musí každý z těchto cizích

klíčů být v tabulce unikátní. – Tato volba je vhodná, pokud výskyt vztahu je vzácný jak pro entity prvního typu tak i pro entity druhého typu.
Například pokud chceme evidovat manželské vztahy mezi zaměstnanci, budou výskyty vzácné, a hodí se takováto varianta.
Pokud má vztah atributy, mapujeme je do tabulky vztahu jako další sloupce.

- Společná tabulka. Oba entitní typy *mapujeme do společné tabulky*, se dvěma množinami sloupců, v první budou sloupce příslušné k prvnímu entitnímu typu, ve druhé sloupce příslušné k druhému entitnímu typu. Tato volba je vhodná, pokud je vztah povinný pro oba entitní typy, a navíc je stabilní.
  Například evidujeme taneční páry, žádný tanečník či tanečnice nejsou sólo. Ale i tento vztah nemusí být v delším časovém horizontu stabilní, tato logická organizace dat by se hodila jen pro průběh jediné soutěže.
  Pokud má vztah atributy, mapujeme je do společné tabulky vztahu jako další sloupce. Například odkdy daný pár spolu tančí.

7. Každý samostatný entitní typ mapujte do samostatné tabulky. Každý jednoduchý atribut entitního typu mapujte do samostatného sloupce příslušné tabulky.
   Zvažme, zda pro "hodnotový" entitní typ podle bodu 2. budeme vůbec nějakou tabulku definovat. Pokud pro daný atribut nevytvoříme číselník, nejspíš takovou tabulku nepotřebujeme. V této fázi si ji však ještě mysleme, na konci celého procesu transformace ji odstraníme.
8. Každý vztah n:m nebo vztah s aritou vyšší než 2 mapujte do samostatné tabulky. Pro každou roli vztahu bude v této tabulce jeden sloupec obsahující cizí klíč odkazující do tabulky odpovídajícího entitního typu.
   Například vztah "nazpíval" mezi písní a zpěvákem mapujeme do tabulky se sloupci nazvanými například "pisen", "zpevak", v nichž budou cizí klíče odkazující do tabulky "PISEN" resp. "ZPEVAK".
   Primární klíč v tabulce mapující takový vztah *je složený z* množiny všech *těchto zmíněných cizích klíčů*.
   Případné atributy vztahu mapujte do dalších sloupců tabulky vztahu.
   - V některých případech může být vhodné, jako primární klíč tabulky vztahu navrhnout umělý klíč, a klíč složený z kombinace cizích klíčů odkazujících na role ve vztahu definovat jako alternativní. Důvodem k takové volbě může být potřeba se na řádky tabulky vztahu odněkud odkazovat.
9. Mapujte vztahy 1:n:
   - *Běžné je mapování do sloupce s cizím klíčem v tabulce na straně n.* Například pro vztah "kdo podal" mezi objednávkou a zákazníkem vytvoříme v tabulce objednávek sloupec odkazující do tabulky zákazníků. Případné atributy vztahu mapujeme do dalších sloupců tabulky na straně n. Pokud vztah "kdo podal" má atribut "kdy", vznikne z něj další sloupec v tabulce objednávek.

Méně běžné je mapování do samostatné tabulky. Tato volba je vhodná pro případy, kdy role na straně n je nepovinná a vzácná. Například některé dokumenty jsou součástí jiného dokumentu. Takový vztah "je součástí" můžeme mapovat do samostatné tabulky. Od mapování vztahu m:m se tato volba odlišuje tím, že v tabulce vztahu bude sloupec odpovídající podřízené roli primárním klíčem. Pro vztah "je součástí" by byl v příslušné tabulce primárním klíčem sloupec s odkazem na podřízený dokument.
Rozhodně ne všechny předchozí kroky lze svěřit automatice CASE nástroje (např. Power Designeru používanému při výuce na VŠE) , co se má rozhodnout, musíte rozhodnout sami. Někdy musíte CASE nástroji pomoci nějakým trikem.

Normalizovaná databáze

Databáze, ve které se žádný údaj či fakt zbytečně neopakuje, je normalizovaná. Uvažte například údaje o kontaktních adresách na dodavatelské firmy, nebo záznamy o skutečnostech, že daný uživatel navštívil danou stránku. Pokud si budeme zapisovat kontaktní adresu na dodavatele ke každému zboží zvlášť, bude naše databáze nenormalizovaná. Pokud při každé návštěvě každého uživatele budeme zaznamenávat, které stránky navštívil (a nikoli třeba ještě časový údaj), pak bude naše databáze nenormalizovaná.

Formální otázky normalizace relační databáze odložme stranou. Panuje však pověra, že použijeme-li standardní transformaci konceptuálního modelu do relačního schématu, například uvedenou v této kapitole, pak získáme normalizovanou databázi. Není to nutně pravda, výsledek záleží na kvalitě konceptuální informační analýzy. Například dvoutvárné entity jsou typickým případem, kdy je snadné se dopustit chyby. Jsou i jiné případy. Takže na normalizaci musíme myslet už při konceptuální analýze: aby každý typ faktu byl modelován jen jednou, aby typy fakt byly nerozložitelné.

Normalizace je dobrá k tomu, abychom

- usnadnili zapisování nových dat, protože nebude nutno je zapisovat více než jednou
- usnadnili aktualizaci dat, protože nebude nutno přepisovat na více místech
- zabránili nekonzistenci v datech (pokud by se informace o zákazníkovi zapisovaly do každé objednávky znova, mohlo by v každé jeho objednávce o něm být zapsáno něco jiného)
- zabránili ztrátě dat (například kdybychom smazali všechny objednávky nějakého zákazníka, nemuseli bychom již o tom zákazníkovi mít žádnou informaci)
- usnadnili výpočty relevantních statistik z dat (například kolik procent z našich stránek který návštěvník viděl)
- výrazně omezili nutnost budoucích radikálních změn ve schématu a navázaných aplikacích

Denormalizace
Postup transformace popsaný v této kapitole vede v bezchybných případech k normalizovanému schématu. To ale nemusí být vždy žádoucí, normalizované schéma je typicky "rozlámáno" do mnoha tabulek, které je třeba při práci s daty propojovat. Pokud se tomu z nějakého dobrého důvodu chceme vyhnout, tzv. denormalizujeme. Nebo pokud se chceme vyhnout opětovným vyhodnocováním stejných výrazů, a místo toho ukládáme výsledky těchto výpočtů do databáze.

Například do položky faktury zaznamenáme i vypočítaný údaj "množství*jednotková cena". Nebo kromě rodného čísla zapíšeme i datum narození nebo pohlaví. Nebo do záznamu o studentovi budeme zapisovat i počet získaných kreditů, i když se dá vypočítat ze záznamů o jeho zkouškách. Při zápisech studentů pak nemusí být systém zatěžován opakovaným ověřováním, zda si může ještě něco zapsat. Nebo do záznamu o knihovní jednotce zapíšeme, zda je vypůjčená, i když se to dá zjistit ze záznamů o výpůjčkách. Usnadní to vyhledávání volných jednotek.

Jak je vidět, denormalizujeme proto, aby

- se usnadnilo vyhledávání potřebných dat
- omezilo vypočítávání výsledků, které jsou stále stejné.

Co denormalizovat
Je vidět, že při normalizaci a denormalizaci jde o jakýsi kompromis mezi usnadněním aktualizací dat a usnadněním jejich vyhledávání. Při rozhodování pomůže, když rozdělíme modelovaná fakta na ta, která odrážejí aktuální stav, který se může měnit, a na fakta archivní, jež jednou zaznamenána se měnit nebudou. U archívních dat se denormalizace bát nemusíme, dokud nenarazíme na problém s objemem dat.

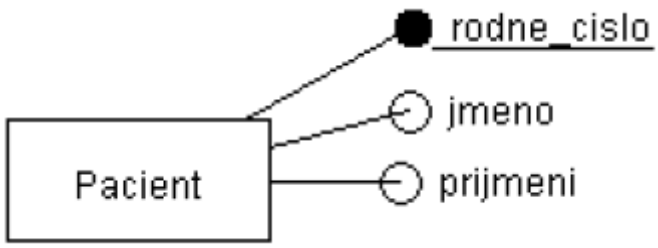
U dat, jež podléhají aktualizacím, záleží rozhodování na provozu databáze, co se má spíše podpořit, jak často který požadavek nastává.

Jinak:
Zadání mluví o návrhu relačního schématu databáze a ve skriptech (a tedy pravděpodobně i u státnic) si potrpí na důsledné odlišování toho co je možná v relačním schématu a co je možné v reálně používaných relačních databázích. Rozdíl je především v tom, že relační model jako takový nepovoluje NULL sloupce, zatímco ve všech běžně používaných databázích toto není problém. Na rozdíly z toho vyplývající upozorním v dalším textu.

Reprezentace silného entitního typu

- reprezentace silného entitního typu není problém
- výsledné schéma bude mít všechny atributy převáděné entity
- identifikační klíč budou tvořit všechny atributy, které se podílejí na klíči

Příklad:



Výsledné schéma relace:
Pacient(__rodne_cislo__, jmeno, prijmeni)

V SQL:

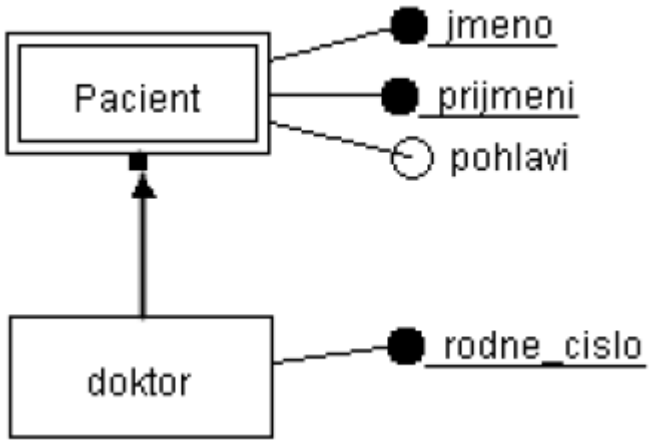
```
CREATE TABLE Pacient (
rodne_cislo Integer NOT NULL,
jmeno VarChar2(255) NULL,
prijmeni VarChar2(255) NULL,
Constraint PK_Pacient PRIMARY KEY (rodne_cislo)
)
```

Reprezentace slabého entitního typu

- nejprve do schématu relace dáme všechny atributy slabého typu, v tento moment schéma obsahuje jen část identifikačního klíče
- následně přidáme všechny identifikační atributy identifikačního vlastníka (tj. všechny atributy které se podílejí na klíči u entity na kterém je daná slabá entita závislá)

Příklad:

- předpokládáme, že nějaká instituce nesmí uchovávat rodná čísla pacientů, jen doktorů a spoléhá se, že jeden doktor nemá víc pacientů se stejným jménem a příjmením



Výsledná schémata relací:
Doktor(__rodne_cislo__)
Pacient(__rodne_cislo__, __jmeno__, __prijmeni__, pohlavi)

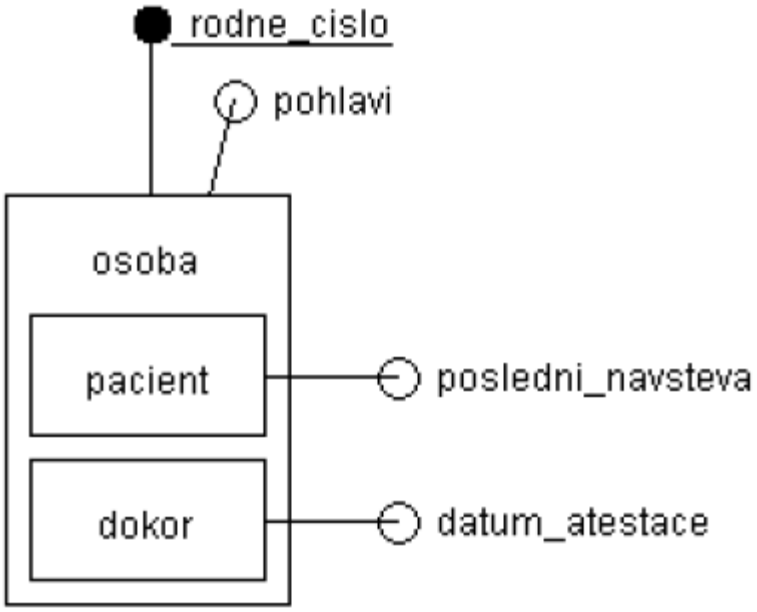
V SQL:
CREATE TABLE doktor (
rodne_cislo Integer NOT NULL,
Constraint PK_doktor PRIMARY KEY (rodne_cislo)
)

CREATE TABLE Pacient (
jmeno VarChar2(255) NOT NULL,
prijmeni VarChar2(255) NOT NULL,
pohlavi Char(1) NULL,
d_rodne_cislo Integer NOT NULL,
Constraint PK_Pacient PRIMARY KEY (jmeno, prijmeni, d_rodne_cislo)
)

Reprezentace ISA hierarchie

- při převodu předpokládám postup „shora dolů“, tj. od zdroje ISA hierarchie (u jednoduché závislosti, kdy je do zdroje ISA hierarchie vnořena jenom jedna úroveň entit je to jedno, ale při komplikovanějším vnoření to jedno není)
- v každém kroku vezmeme jednu entitu, vytvoříme k ní relaci jakoby se jednalo o klasický silný entitní typ a nakonec k ní přilepíme identifikační atributy zdroje ISA hierarchie (tj. pokud jedeme „shora dolů“ tak přilepíme identifikační atributy z nejbližšího nadtypu)

Příklad:



Výsledná schémata relací:
Osoba(__rodne_cislo__, pohlavi)
Pacient(__rodne_cislo__, posledni_navsteva)
Doktor(__rodne_cislo__, datum_atestace)

V SQL:
CREATE TABLE osoba (
rodne_cislo Integer NOT NULL,
pohlavi Char(1) NULL,
Constraint PK_osoba PRIMARY KEY (rodne_cislo)
)

CREATE TABLE pacient (
posledni_navsteva Date NULL,
o_rodne_cislo Integer NOT NULL,
Constraint PK_pacient PRIMARY KEY (o_rodne_cislo)
)

CREATE TABLE dokor (
datum_atestace Date NULL,
o_rodne_cislo Integer NOT NULL,
Constraint PK_dokor PRIMARY KEY (o_rodne_cislo)
)

Reprezentace vztahů
Vztah 1:1
Můžou nastat tři případy.
Oba entitní typy mají nepovinné členství ve vztahu

- pokud nemůžeme povolit pro sloupec NULL hodnoty, nemáme jinou možnost než vytvořit schémata tří relací, pro každý entitní typ jedno, třetí bude vztahové a bude obsahovat klíče obou předchozích jako cizí klíče

- jako primární klíč schématu vztahové relace může sloužit klíč kterékoliv ze dvou ostatních schémat relací
- případné atributy vztahu bude také obsahovat vztahová relace
- pokud bychom mohli použít NULL hodnoty, mohli bychom použít jen dvě relace a na konec jedné z nich přilepit klíčové atributy schématu druhé relace

Příklad:

Výsledná schémata relací:

Zamestanec(__osobni_cislo__, jmeno, ...)

Auto(__spz__, vyrobce, ...)

Pouziva(__osobni_cislo__, __spz__)

Integritní omezení:

Pouziva[osobni_cislo] $\subseteq$ Zamestnanec[osobni_cislo]

Pouziva[spz] $\subseteq$ Auto[spz]

Jeden entitní typ má nepovinné členství ve vztahu, druhý povinné

- entitní typ, který má povinné členství ve vztahu je závislý na druhém entitním typu (který je nezávislý, protože členství ve vztahu pro něj není povinné)
- definujeme schémata dvou relací
- ke schématu relace pro závislý entitní typ přidáme atributy odpovídající identifikačnímu klíči
- nezávislého entitního typu, stejně tak k témuto schématu přidáme připadné atributy relace, v tomto schématu můžou být primárním klíčem opět jak klíčové atributy závislého typu, tak nezávislého
- pokud bychom mohli použít NULL hodnoty, mohli bychom použít jen jedno „slepené“ schéma relaci, přičemž všechny sloupce závislého entitního typu by mohly nabývat NULL hodnoty

Příklad:

Výsledná schémata relací:

Zamestanec(__osobni_cislo__, jmeno, ...)

Auto(__spz__, vyrobce, ..., osobni_cislo)

Integritní omezení:

Auto[osobni_cislo] $\subseteq$ Zamestnanec[osobni_cislo]

Oba entitní typy mají povinné členství ve vztahu

- definujeme jedno schéma relaci: vznikne slepením schémat relací pro původní entitní typy
- oba původní klíče můžou být primárními klíči
- případně přidáme atributy odpovídající atributům vztahu

Příklad:

Výsledné schéma relace:

Zamestanec(__osobni_cislo__, jmeno, ..., __spz__, vyrobce, ...)

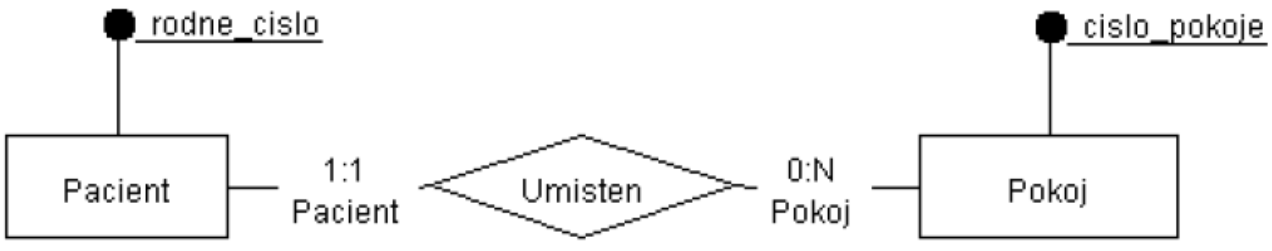
Vztah 1:N

- Entita jednoho typu je determinantem entity druhého typu, pokud entitu druhého typu jednoznačně určuje. Tedy pokud pro N záznamů entity druhého typu existuje jenom 1 entita prvního typu, je entita prvního typu determinantem.
- přidat příklad ze skript

Povinné členství determinantu ve vztahu

- v takovém případě definujeme dvě relační schémata, pro každý entitní typ jedno
- k relačnímu schématu determinantu přilepíme cizí klíč odkazující k identifikačnímu klíčí druhého entitního typu
- primárním klíčem bude klíč determinantu

Příklad:



Výsledná schémata relací:

Pacient(__rodne_cislo__, ..., cislo_pokoje, ...)

Pokoj(__cislo_pokoje__, pocet_luzek, ...)

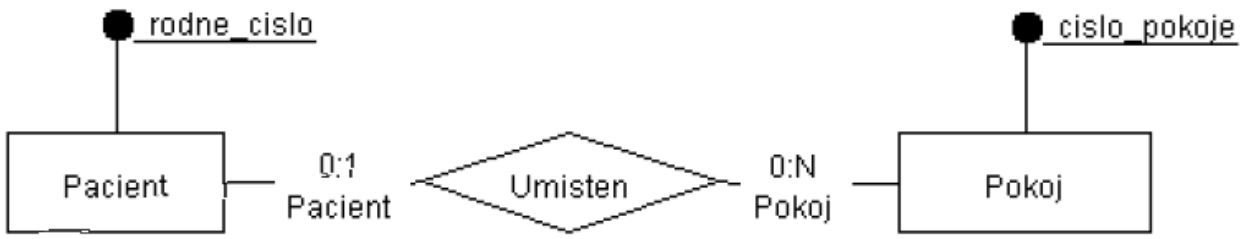
Integritní omezení:

Pacient[cislo_pokoje] ⊆ Pokoj[cislo_pokoje]

Nepovinné členství determinantu ve vztahu

- definujeme tři schémata relace, pro každý entitní typ jedno, třetí vztahové
- vztahové schéma bude obsahovat cizí klíče odkazující na primární klíče obou entitních typů, případně atributy odpovídající atributům relace, primární klíč vztahového relačního schématu bude ten z cizích klíčů, který odkazuje na primární klíč determinantu
- pokud bychom měli povolené NULL hodnoty, můžeme použít stejné řešení jako v předchozím bodě, s tím že cizí klíč by mohl být i NULL

Příklad:



Výsledná schémata relací:

Pacient(__rodne_cislo__, ...)

Pokoj(__cislo_pokoje__, pocet_luzek, ...)

Umisten(__rodne_cislo__, cislo_pokoje)

Integritní omezení:

Umisten[cislo_pokoje] ⊆ Pokoj[cislo_pokoje]

Umisten[rodne_cislo] ⊆ Pacient[rodne_cislo]

Vztah M:N

- Vždy definujeme tři schémata relace, jedno pro každý entitní typ, a třetí vztahové, které bude obsahovat všechny primární klíče účastníků vztahu a případně další vztahové atributy.