DBI025
1. Cvičení: E-R modelování

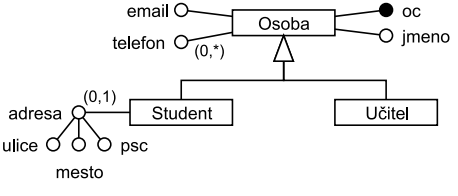
Martin Nečaský, KSI

24. února 2008

Příklady v tomto cvičení se týkají konceptuálního modelování pomocí E-R (Entity-Relationship) modelu. Pro každý příklad navrhněte E-R schéma nebo rozšiřte E-R schéma z předchozího příkladu. Zadání jsou záměrně mírně zamlžena. U reálných ne-IT zadavatelů mohou být ještě podstatně mlžnější a neúplnější. Schémata vždy minimalizujte, tzn. modelujte vždy právě to, co je požadováno v zadání (nic víc, nic míň).

Příklad 1:
Na fakultě jsou učitelé a studenti. Abychom mohli každou osobu jednoznačně identifikovat, přiřazujeme jí při registraci do systému osobní kód. Např. *Martin Nečaský* má osobní kód *necasky*. Email musí systém evidovat na každou osobu. Telefon musí mít každý učitel, pro studenty není úplně nutný. Pokud je student v posledním ročníku, musíme mu poštou posílat pozvánku na statní zkoušku a vyrozumění o výsledku státní zkoušky. Co se týče kontaktních telefonů, na studenta stačí uchovávat jeden, na učitele potom občas potřebujeme i více telefonů.

Řešení příkladu 1:
Situaci modeluje schéma na obrázku 1. Atribut telefon je přiřazen k entitnímu typu Osoba, protože telefon můžeme uchovávat jak pro učitele tak pro studenty. Schéma neobsahuje integritní omezení, že pro daného studenta udržujeme maximálně 1 telefonní číslo.

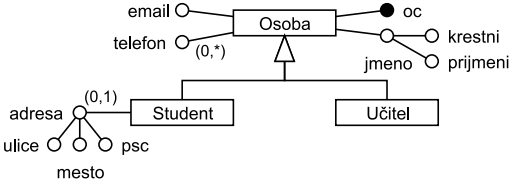


Obrázek 1: Řešení Příkladu 1

Příklad 2:
Kromě samotného popisu domény mají na konceptuální model vliv také požadavky funkční. Stručně řečeno, funkční požadavky nepopisují, jak mají data vypadat, ale co chceme s daty dělat. Zkontrolujte schéma z předchozího příkladu, zda splňuje následující jednoduché funkční požadavky (pokud ne, doplňte schéma):

1. vypsat seznam studentů/učitelů
2. seřadit daný seznam uživatelů podle křestního jména nebo podle příjmení
3. vypsat seznam uživatelů z daného města

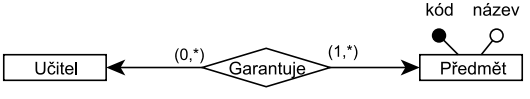
Řešení příkladu 2:
Schéma z předchozího příkladu je již vhodné pro požadavky 1) a 3). Pro splnění požadavku 2) musíme dále strukturovat atribut jmeno na krestni a prijmeni. Řešení je na obrázku 2.



Obrázek 2: Řešení Příkladu 2

Příklad 3:
Rozšiřte schéma na obrázku 1 o předměty vyučované na fakultě. Pro každý předmět evidujeme jeho název a identifikační kód. Každý předmět je garantován 1 či více učiteli.

Řešení příkladu 3:
Přidáme entitní typ Předmět a binární vztahový typ Garantuje modelující vztah "učitel garantuje předmět". Řešení je na obrázku 3.

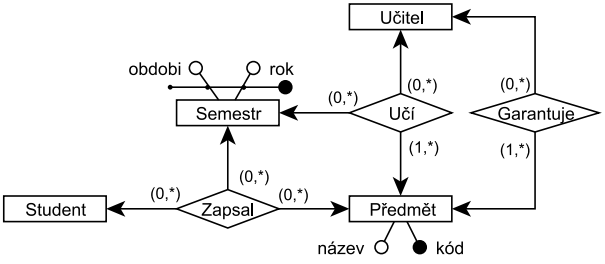


Obrázek 3: Řešení Příkladu 3

Příklad 4:
Kromě garantů daného předmětu potřebujeme evidovat také učitele, kteří vyčují daný předmět. Není nutné, aby garant také předmět vyučoval. V každém semestru

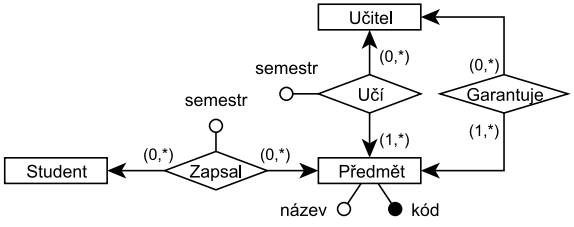
mohou daný předmět vyučovat jiní učitelé. Také budeme evidovat zápisy studentů do předmětů. I pro zápis evidujeme semestr, pro který je zápis platný. Rozšiřte schéma na obrázku 3 tak, aby byly tyto nové požadavky splněny.

Řešení příkladu 4:
Přidáme entitní typ Semestr, který je identifikován klíčem složeným z atributů Rok a Období (tj. letní nebo zimní). Dále pak dva ternární vztahové typy Učí a Zapsal modelující vztahy "učitel učí předmět v semestru" a "student má zapsán předmět v semestru". Řešení je na obrázku 4.



Obrázek 4: Řešení Příkladu 4

Příklad 5:
Řeší schéma na obrázku 5 požadavky předchozího příkladu?



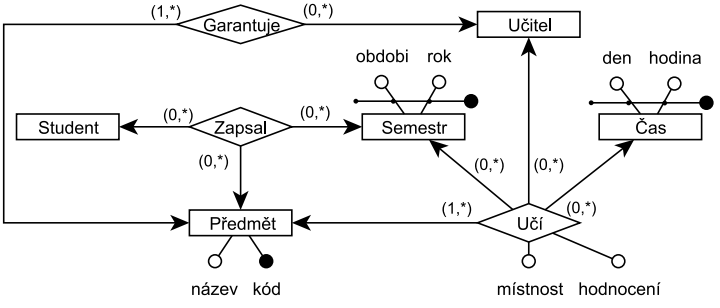
Obrázek 5: Příklad 5

Řešení příkladu 5:
Neřeší. Problém je ve významu vztahového typu. Instance vztahového typu je jednoznačně určena klíči entitních typů ve vztahu. Např. daná dvojice předmět-učitel může být ve vztahového typu Učí nejvýše jednou, dvě instance Učí se stejným předmětem a učitelem už jsou vzájemně nerozlišitelné. My však potřebujeme, aby učitel mohl vyučovat daný předmět ve více semestrech a stejně tak aby si student mohl zapsat stejný předmět ve více semestrech. Atribut semestr nám však díky vlastnostem vztahových typů nepomůže a potřebujeme ho nahradit přímo entitním typem. Potom je daná instance Učí jednoznačně určena učitelem, předmětem

a navíc semestrem. Učitel může tedy učit daný předmět vícekrát, v různých semetrech. V daném semestru však nejvíše jednou. Stejně tak pro vztahový typ Zapsal.

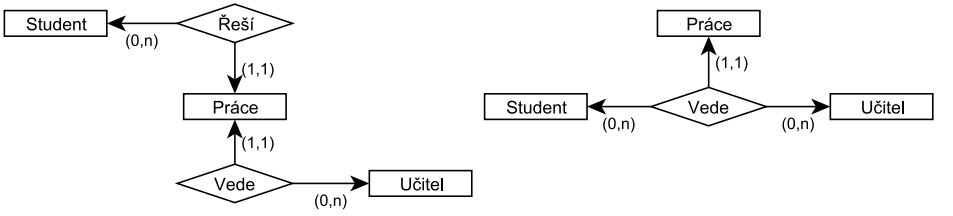
Příklad 6:
Rozšiřte dále schéma na obrázku 4 podle následujících upřesňujících požadavků. Předmět učí v jednom semestru více učitelů (máme přednášku a cvičení k předmětu v jedné a více paralelkách). Učitel může v jednom semestru vést více paralelních přednášek a cvičení k danému předmětu. Potřebujeme také vědět, kde daná výuka probíhá (tj. v jaké místnosti). Učitelova výuka je hodnocena (bodová stupnice 1-10).

Řešení příkladu 6:
Aby mohl daný učitel vést pro daný předmět více paralelek v jednom semestru, přidáme ještě ke vztahovému typu Učí identifikaci podle času. Dále přidáme ještě atributy místnost a hodnocení. Řešení je na obrázku 6.

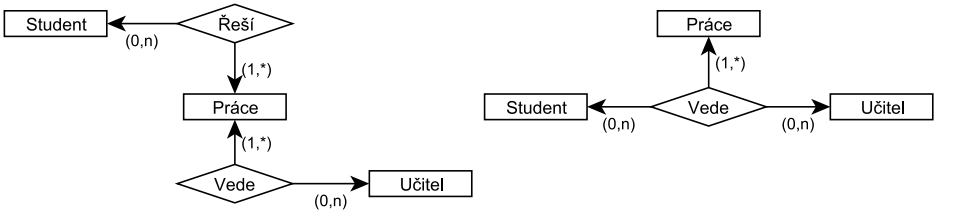


Obrázek 6: Řešení Příkladu 6

Příklad 7:
Vysvětlete rozdíl v sémantice mezi dvojicemi schémat na následujících obrázcích.



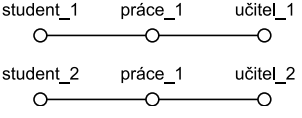
Obrázek 7: Příklad 7a



Obrázek 8: Příklad 7b

Řešení příkladu 7:
Schémata na obrázku 7 mají stejný význam. Pro daného studenta v obou případech zjistíme stejné dvojice (práce, na které student pracuje a k ní učitel, který práci vede). Pro daného učitele vždy zjistíme stejné dvojice (práce, kterou učitel vede a k ní student, který práci řeší). A konečně pro danou práci vždy zjistíme stejnou dvojici (student, který práci řeší a učitel, který jí vede).

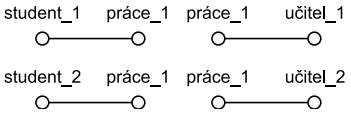
Schéma na obrázku 8 nemají stejný význam bez specifikace dalších omezení. Uvažujme schéma instancí schématu vpravo:



Obrázek 9: Schéma Instancí k Příkladu 7

Dekompozicí do schématu vlevo dojde k dekompozici v instancích zobrazené na obrázku 10.

Uvažujeme-li sémantiku schématu vpravo, získáme na otázku "Kteří učitelé vedou studenta student_2 v práci práce_1?" odpověď "učitel_2". Uvažujeme-li

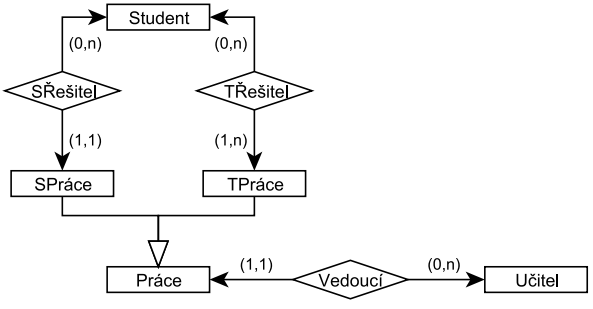


Obrázek 10: Schéma Instancí k Příkladu 7

však sémantiku schématu vlevo, získáme na tu samou otázku odpověď "učitel_1 a učitel_2". Aby bylo možné dekompozici provést, musí platit omezení, že pro danou práci existuje právě jeden řešitel nebo že pro danou práci existuje právě jeden vedoucí. To však zaručeno není. Je důležité si uvědomit, že dekompozice nemusí vést na ekvivalentní schéma, i když se to na první pohled může zdát.

Příklad 8:
Upravte schéma na obrázku 7 vlevo tak, aby bylo možné rozlišit samostatné práce (tj. práce, které mají jednoho řešitele) a týmové práce (tj. práce, které mohou mít více řešitelů).

Řešení příkladu 8:
Pomocí IS-A vytvoříme dva typy prací - samostatné a týmové a rozlišíme i vztah Řeší jako na obrázku 11.

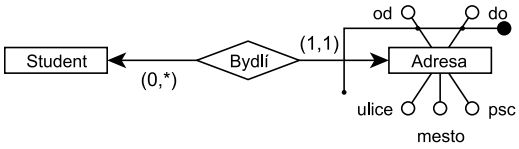


Obrázek 11: Řešení Příkladu 8

Příklad 9:
Upravte schéma pro adresy studentů tak, aby bylo možné sledovat historii adres studenta. Každá adresa musí být identifikovatelná studentem, ke kterému patří a datumem od-do, ve kterém byla platná.

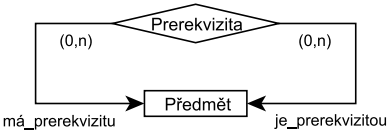
Řešení příkladu 9:
Modelujeme adresy jako slabý entitní typ. Řešení je na obrázku 12.

Příklad 10:
Navrhněte schéma modelující prerekvizity předmětů.



Obrázek 12: Řešení Příkladu 9

Řešení příkladu 10:
Potřebujeme vztahový typ, ve kterém participuje jeden entitní typ dvakrát, tj. tzv. rekurzivní vztah. Musíme použít tzv. rolí k jejich odlišení. Řešení je na obrázku 12.



Obrázek 13: Řešení Příkladu 10

DBI025
2. Cvičení: Převod E-R Modelu do Relačního Modelu

Martin Nečaský, KSI

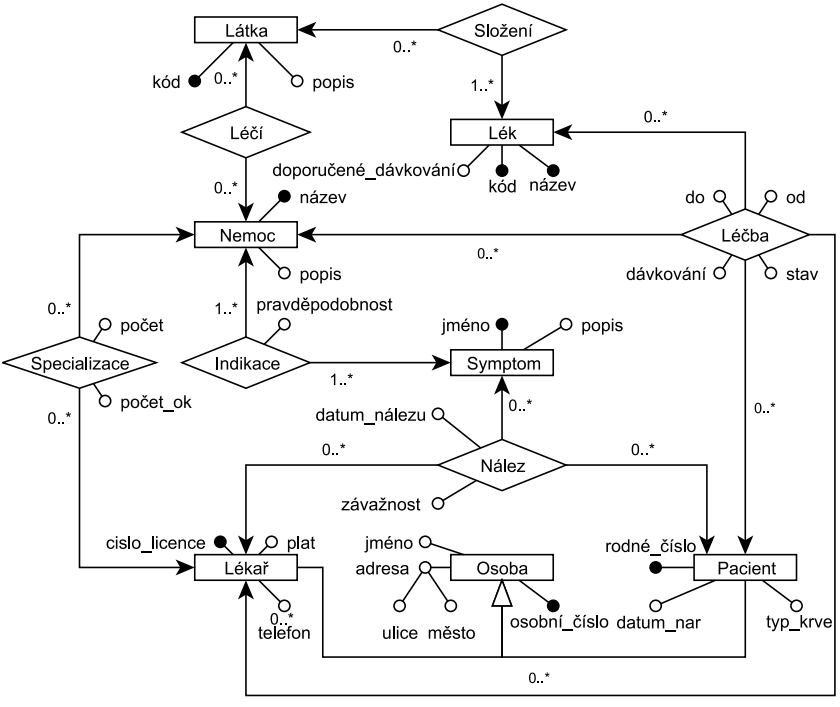
3. března 2008

Příklad 1:
Navrhněte E-R schéma pro nemocniční systém. V systému uchováváme údaje o různých osobách. Potřebujeme pro ně jméno, adresu (ulici a město) a osobní číslo. Rozlišujeme dva speciální typy osob, pacienty a lékaře. Lékař má svojí licenci a také uchováváme jeho aktuální plat a seznam telefonních čísel na něj (nemusí být žádné). Pacient má své rodné číslo, datum narození a krevní skupinu. Každý lékař má několik specializací, na různé nemoci. Potřebujeme vědět, kolikrát již lékař danou nemoc ze své specializace léčil a kolikrát byla tato léčba úspěšná. Pacienti chodí k lékařům na vyšetření. Během vyšetření lékař hledá u pacienta symptomy. Symptomy potom s určitou pravděpodobností indikují různé nemoci. Každý nález symptomu je zaznamenán do systému. Také je zaznamenán datum nálezu a jeho závažnost. Lékař potom odhaduje nemoc a může nařídit pacientovi léčbu této nemoci nějakým lékem. Informace o léčbě jsou ukládány do systému. V systému uložíme datumy od-do, kdy má pacient lék brát a dávkování. Také ukládáme stav léčby. Lék má svůj jedinečný název a také kód. Dále má své doporučené dávkování. U léků evidujeme jejich složení. Lék se skládá z různých látek. Každá látka má svůj identifikační kód a popis. Evidujeme také, jaké látky jsou vhodné k léčbě jakých nemocí.

Řešení příkladu 1:
Výsledné schéma je na obrázku 1.

Příklad 2:
Odvoďte relační schéma z E-R schématu na obrázku 1. Uvědomte si rozdíl mezi pojmy *schéma relace (tabulky)* a *schéma relační databáze*.

Řešení příkladu 2:



Obrázek 1: Řešení Příkladu 1

Osoba(osobni_cislo, jmeno, ulice, mesto)
Lekar(osobni_cislo, cislo_licence, plat)
Telefon(telefon, cislo_licence_lekare)
Pacient(osobni_cislo, rodne_cislo, datum_narozeni, typ_krve)
Symptom(jmeno, popis)
Nemoc(nazev, popis)
Latka(kod, popis)
Lek(kod, nazev, doporucene_davkovani)
Specializace(nazev_nemoci, cislo_licence_lekare, pocet, pocet_ok)
Indikace(nazev_nemoci, jmeno_symptomu, pravdepodobnost)
Nalez(cislo_licence_lekare, jmeno_symptomu, rodne_cislo_pacienta, datum_nalezu, zavaznost)
Lecba(rodne_cislo_pacienta, cislo_licence_lekare, nazev_nemoci, kod_leku, od, do, davkovani, stav)
Slozeni(kod_leku, kod_latky)
Leci(kod_latky, nazev_nemoci)

Referenční integritu, tj. správné hodnoty referencí na klíče, můžeme zajistit následujícími omezeními (tzv. cizí klíče):

$Lekar[osobni\_cislo] \subseteq Osoba[osobni\_cislo]$
$Pacient[osobni\_cislo] \subseteq Osoba[osobni\_cislo]$
$Telefon[cislo\_licence\_lekare] \subseteq Lekar[cislo\_licence]$
$Specializace[nazev\_nemoci] \subseteq Nemoc[nazev]$
$Specializace[cislo\_licence\_lekare] \subseteq Lekar[cislo\_licence]$
...atd.

Nevýhodou tohoto relačního schématu jsou komplikované klíče. Např. v případě schématu relace Lecba *máme klíč složený ze čtyř atributů. Dokonce i v případě jednoduchých klíčů, jako např.* rodne_cislo *ve schématu relace* Pacient *je problém, protože hodnotami klíče budou řetězce. S takovými klíči se ale špatně pracuje a je lepší přidat umělé klíče, jejichž hodnoty jsou jednoduché (typicky celá čísla, navíc automaticky generovaná od 1). Pomocí těchto umělých klíčů efektivně vyřešíme naši interní identifikaci záznamů v databázi a reference mezi nimi. Tyto umělé klíče jsou čistě věcí implementace v relační databázi (resp. relačním modelu dat) a nepatří tedy do E-R schématu, který je určen spíše pro zadavatele a implementační detaily ho nezajímají. Následující relační schéma ukazuje předchozí schéma doplněné o umělé klíče. Ty jsou zde používány i pro reference. Název umělého klíče lze vytvořit např. z názvu relace doplněného o '_id'. Často se také používá jenom 'oid' (jako zkratka za Object IDentification). Na názvech celkem nezáleží - pojmenovávací pravidla si lze zvolit libovolně - jen je dobré nějaká mít*

dohodnutá předem.

Osoba(osoba_id, osobni_cislo,jmeno,ulice,mesto)
Lekar(lekar_id,osoba_id,cislo_licence,plat)
Telefon(telefon_id,telefon,lekar_id)
Pacient(pacient_id,osoba_id,rodne_cislo,datum_narozeni,typ_krve)
Symptom(symptom_id,jmeno,popis)
Nemoc(nemoc_id,nazev,popis)
Latka(latka_id,kod,popis)
Lek(lek_id,kod,nazev,doporucene_davkovani)
Specializace(specializace_id,nemoc_id,lekar_id,pocet,pocet_ok)
Indikace(indikace_id,nemoc_id,symptom_id,pravdepodobnost)
Nalez(nalez_id,lekar_id,symptom_id,pacient_id,datum_nalezu,zavaznost)
Lecba(lecba_id,pacient_id,lekar_id,nemoc_id,lek_id,od,do,davkovani,stav)
Slozeni(slozeni_id,lek_id,latka_id)
Leci(leci_id,latka_id,nemoc_id)

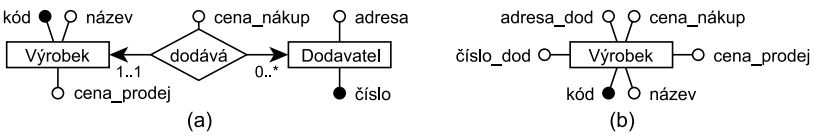
Příklad 3:
Vysvětlete pojmy klíč a cizí klíč na relačním schématu z předchozího příkladu.

Řešení příkladu 3:
Každá relace musí mít alespoň jeden klíč ve svém schématu. Každý prvek relace (n-tice, řádek) pak musí mít hodnotu klíče jednoznačnou v rámci všech ostatních prvků (n-tic, řádků). Klíče odvodíme z E-R schématu a můžeme přidat i umělé klíče.

Cizí klíče používáme pro popis odkazů mezi prvky dvou různých relací. Jako cizí klíč můžeme označit jeden nebo více atributů schématu relace. Tento cizí klíč se odkazuje na klíč ve schématu jiné nebo stejné relace. Tak např. v našem předchozím relačním schématu (bez umělých klíčů) je atribut cislo_licence_lekare *ve schématu relace* Telefon *cizím klíčem, který se odkazuje na klíč* cislo_licence *ve schématu relace* Lekar*. Odkazy nejsou v relacích realizovány jako přímé pointery. Jsou zajištěny speciálními omezeními, např.* $Telefon[cislo\_licence\_lekare] \subseteq Lekar[cislo\_licence]$*. To popisuje, že množina všech hodnot atributu* cislo_licence_lekare *v relaci* Telefon *je podmožinou všech hodnot atributu* cislo_licence *v relaci* Lekar*.*

Příklad 4:
Uvažujte dvě E-R schémata na obrázku 2. Popište mezi nimi rozdíl na konceptuální úrovni.

Řešení příkladu 4:
Schéma (a) popisuje dodavatele jako samostatný entitní typ a vztah dodávat výrobek jako explicitní vztahový typ. To je na konceptuální úrovni zřejmě věcně správnější. Nicméně (b) může v určitých případech stačit a navíc je jednodušší (méně entitních a vztahových typů). V (b) ale nemáme explicitně modelované dodavatele. Ztrácíme tak možnost identifikace dodavatelů a jejich účasti v případných



Obrázek 2: Příklad 5

dalších vztazích. Navíc pokud nejsme opatrní, máme problémy při převodu do relačního modelu - viz. další příklad.

Příklad 5:
Převodem jak E-R schématu (b) z obrázku 2 do relačního modelu můžeme získat následující jednoduché schéma relační databáze:

$$Vyrobek(\underline{kod}, nazev, cena\_prodej, cen\_nakup, adresa\_dod, cislo\_dod)$$

Je s ním vše v pořádku?

Řešení příkladu 5:
Není. Uvažujme následující relaci Vyrobek*:*

$$\begin{array}{lllllll} \{ & ('vyr1', & 'vyrobek1', & 500, & 400, & 'Prazska1, Praha', & 'dod1'), \\ & ('vyr2', & 'vyrobek2', & 900, & 700, & 'Prazska2, Praha', & 'dod2'), \\ & ('vyr3', & 'vyrobek3', & 700, & 300, & 'Prazska1, Praha', & 'dod1'), \\ & ('vyr4', & 'vyrobek4', & 800, & 200, & 'Prazska3, Praha', & 'dod3'), \\ & ('vyr5', & 'vyrobek5', & 800, & 100, & 'Prazska1, Praha', & 'dod1')\} \end{array}$$

Problém je, že pro každého dodavatele jsou údaje o něm zkopírované tolikrát, kolik výrobků dodává. Např. pro dodavatele dod1 *to je celkem třikrát (pro výrobky* vyr1*,* vyr3 *a* vyr5*. Tomu se říká redundance (redundantní data ¡-¿ opakování stejných dat na více místech v databázi). Ty mohou způsobovat řadu problémů - viz. další příklad.*

Příklad 6:
Zkuste na předchozím příkladě popsat, co mohou způsobovat redundance za problémy.

Řešení příkladu 6:

- *problém s vložením: při vkládání výrobku musíme zároveň dodat i data o dodavateli (i když už je máme třeba u jiného výrobku)*
- *problém s aktualizací: nutnost aktualizace adresy dodavatele na více místech zároveň*
- *problém s mazáním: vymazání posledního výrobku, který dodavatel dodává znamená i vymazání údajů o samotném dodavateli*

Řešením problémů je odstranění redundance. Toho dosáhneme buď správným návrhem E-R schématu nebo pomocí tzv. normalizací, které budou tématem příštích cvičení.

Příklad 7:
Najdete podobný problém i v E-R schématu z obrázku 1?

Řešení příkladu 7:
Problém může být ve vztahovém typu Léčba*. Je potřeba blíže specifikovat požadavky. Problém nastane, pokud během léčby nemoci pacienta podává lékař více léků. Uvažujme následující relaci* Lecba*:*

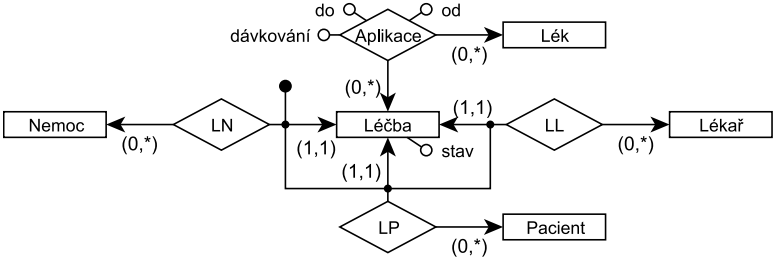
$$\begin{array}{lllll} \{ & (\textit{'pacient1'}, & \textit{'lekar1'}, & \textit{'nemoc1'}, & \textit{'lek1'}, \quad ...), \\ & (\textit{'pacient1'}, & \textit{'lekar1'}, & \textit{'nemoc1'}, & \textit{'lek2'}, \quad ...), \\ & (\textit{'pacient1'}, & \textit{'lekar1'}, & \textit{'nemoc1'}, & \textit{'lek3'}, \quad ...)\} \end{array}$$

Redundance je skryta v opakování trojice 'pacient1', 'lekar1', 'nemoc1'. Ta říká, že lékař 'lekar1' léčí pacienta 'pacient1' na nemoc 'nemoc1'. K tomu aplikuje léky 'lek1', 'lek2', 'lek3'. Pro každý aplikovaný lék je trojice zopakována. Protože je cizí klíč kod_leku *součástí klíče relace* Lecba*, musí mít každý prvek této relace nějakou hodnotu tohoto atributu. To znamená, že nemůže mít tato relace prvek*

$$(\textit{'pacient1'}, \quad \textit{'lekar1'}, \quad \textit{'nemoc1'}, \quad , \quad ...)$$

(navíc ani prázdné hodnoty v relačním modelu dat nemůžeme používat - to půjde až s využitím SQL). Jinými slovy nemůžeme v naší relaci reprezentovat situaci, kdy lékař léčí nemoc pacienta, ale ještě neaplikoval žádný lék.

Problém můžeme vyřešit na úrovni E-R modelu např. následovně:



Obrázek 3: Příklad 7

tj. převedeme vztahový typ Lecba *na slaby entitni typ identifikovany kombinaci klicu* Lekar, Pacient, Nemoc. Lek *v identifikaci zahrnut není, což nám umožňuje nastavit kardinalitu tak, abychom pro jednu léčbu mohli mít více léků. Zbavíme se i redundance. V závislosti na dalších upřesněních zadavatele ale ani toto schéma nemusí být ještě správně. Například teď není možné, aby jeden lékař léčil daného pacienta na danou nemoc více než jednou. Aby to bylo možné, je potřeba rozšířit klíč slabého entitního typu* Lecba *o nějaký doplňující atribut.*

Příklad 8:
Uvažujte různé možnosti reprezentace IS-A hierarchie Osoba - Lékař—Pacient. Liší se nějak? Najděte výhody/nevýhody těchto řešení.

Řešení příkladu 8:

$$Osoba(\underline{oc}, ulice, mesto, jmeno)$$
$$Lekar(\underline{oc}, \underline{licence}, plat)$$
$$Pacient(\underline{oc}, \underline{cp}, datum\_nar, typ\_krve)$$

Údaje o jedné osobě ve více různých relacích, což zpomalí přístup k datům o osobě. Není redudnance. Můžeme mít i osobu, která není lékařem ani pacientem.

$$Osoba(\underline{oc}, ulice, mesto, jmeno)$$
$$Lekar(\underline{oc}, \underline{licence}, plat, ulice, mesto, jmeno)$$
$$Pacient(\underline{oc}, \underline{cp}, datum\_nar, typ\_krve, ulice, mesto, jmeno)$$

Údaje o jedné osobě v jedné relaci. Redundance. Můžeme mít i osobu, která není lékařem ani pacientem. Nutná opatrnost při změně schématu všech osob (např. odstranění atributu jmeno *nebo pridani atributu* email *nutné propagovat do více schémat relací).*

$$Lekar(\underline{oc}, \underline{licence}, plat, ulice, mesto, jmeno)$$
$$Pacient(\underline{oc}, \underline{cp}, datum\_nar, typ\_krve, ulice, mesto, jmeno)$$

Údaje o jedné osobě v jedné relaci. Není redundance. Nemůžeme ale mít osobu, která není ani lékařem a ani pacientem. Nutná opatrnost při změně schématu všech osob.

$$Osoba(\underline{oc}, ulice, mesto, jmeno, \underline{licence}, plat, \underline{cp}, datum\_nar, typ\_krve, typ\_osoby)$$

Údaje o jedné osobě v jedné relaci. Není redundance. Můžeme ale mít osobu, která není ani lékařem a ani pacientem. Musíme přidat umělý atribut typ_osoby*, který určuje, o jaký typ osoby se jedná. Potřebujeme ale tzv. nedefinovanou hodnotu* NULL*, kterou však relační model nemá (SQL však ano).*

DBI025
3. Cvičení: Funkční Závislosti a Normální Formy

Martin Nečaský, KSI

12. března 2007

Příklad 1:
Vysvětlete a formálně popište pojem *funkční závislost.*

Řešení příkladu 1:
Funkční závislost $X \to Y$ nad schématem relace R s množinou atributů A, tj. $R(A)$ je parciální zobrazení $f : X \to Y$ kde:

- $X \subseteq A$ *a* $Y \subseteq A$ *a*
- *jsou-li* $r1$ *a* $r2$ *dva prvky relace* R *takové, že* $r1(X) = r2(X)$*, potom* $r1(Y) = r2(Y)$

Představujeme-li si relaci jako tabulku, potom $r1$ a $r2$ jsou dva její řádky. Pokud se tyto dva řádky shodují na hodnotách ve sloupečcích pro atributy z X, potom se shodují i na hodnotách ve sloupečcích pro atributy z Y. Funkční závislost je speciálním typem obecnějšího pojmu, tzv. integritní omezení. Integritní omezení popisují povolená data v relacích. Jiným typem integritních omezení jsou např. cizí klíče, zmiňované v předchozích cvičeních. Klíče jsou také typem integritních omezení, konkrétně speciálním typem funkčních závislostí, viz. dále.

Příklad 2:
Uvažujte následující schéma relace:

$$Vyuka(kod\_predmetu, cas, mistnost, rc\_ucitele, telefon\_ucitele, hodnoceni, budova, areal)$$

Dále uvažujte následující požadavky:

1. každý učitel učí daný předmět pouze v jedné místnosti
2. učitelé mohou sdílet telefony, každý má ale jenom jeden
3. učitelé jsou hodnoceni pro každý předmět zvlášť a jenom jednou

Odvoďte funkční závislosti, které pro $Vyuka$ platí. Pokud vyjde, že daná množina atributů již určuje všechny ostatní atributy, označte množinu těchto ostatních atributů jen pomocí *all*, ať to nemusíte vypisovat.

Řešení příkladu 2:

$$
\begin{array}{lll}
cas, mistnost & \rightarrow all & \\
cas, rc\_ucitele & \rightarrow all & \\
rc\_ucitele & \rightarrow telefon\_ucitele & (2) \\
kod\_predmetu, rc\_ucitele & \rightarrow mistnost, hodnoceni & (1), (3) \\
mistnost & \rightarrow budova, areal & \\
budova & \rightarrow areal &
\end{array}
$$

Příklad 3:
Vysvětlete pojem klíče jako speciálního případu funkční závislosti. Opět jej popište formálně.

Řešení příkladu 3:
Klíčem nad schématem relace R s množinou atributů A, tj. $R(A)$, je každá podmnožina K množiny A taková, že:

- $K \rightarrow A$
- *neexistuje K', která je podmnožinou (ostře) K a $K' \rightarrow A$*

Klíčem je tedy každá množina atributů, která již funkčně určuje všechny ostatní atributy a navíc je minimální vzhledem k inkluzi, tj. neexistuje žádná vlastní podmnožina s touto vlastností. Klíčů může být i více.

Příklad 4:
Odvoďte klíče pro Příklad 2. Poté uvažujte požadavek:

- každý předmět je vyučován právě jedním učitelem

Změní nějak tento požadavek odvozené klíče?

Řešení příkladu 4:
Klíče bez dodatečného požadavku jsou množiny $\{cas, mistnost\}$ a $\{cas, rc\_ucitele\}$. S požadavkem navíc získáváme funkční závislost $kod\_predmetu \rightarrow rc\_ucitele$ a je tedy možné každý výskyt atributu $rc\_ucitele$ na levé straně nějaké funkční závislosti nahradit atributem $kod\_predmetu$. Získáme tedy navíc ještě klíč $\{cas, kod\_predmetu\}$.

Příklad 5:
Co to jsou Armstrongova pravidla?

Řešení příkladu 5:
Mějme $R(A, F)$. Nechť X, Y a Z jsou podmnožinou A a F je množina funkčních závislostí pro F. Armstrongova pravidla jsou:

- *jestliže Y je podmnožinou X, potom $X \rightarrow Y$ (triviální FZ, axiom)*
- *jestliže $X \rightarrow Y$ a $Y \rightarrow Z$, potom $X \rightarrow Z$ (tranzitivita, pravidlo)*
- *jestliže $X \rightarrow Y$ a $X \rightarrow Z$, pak $X \rightarrow YZ$ (kompozice, pravidlo)*
- *jestliže $X \rightarrow YZ$, pak $X \rightarrow Y$ a $X \rightarrow Z$ (dekompozice, pravidlo)*

Platí

- *korektnost* co z množiny F odvodíme, platí pro libovolnou instanci R
- *úplnost* lze jimi odvodit všechny FZ platné ve všech instancích R
- *nezávislost* odstraněním jakéhokoliv z nich porušíme úplnost

Pro příklady 6 až 10 uvažujte schéma relace $R(A, B, C, D, E, F)$ a množiny funkčních závislostí $F = \{A \rightarrow BEF,\ BC \rightarrow DE,\ BDE \rightarrow F,\ ADF \rightarrow CE,\ E \rightarrow CBD\}$ a $G = \{A \rightarrow B,\ AB \rightarrow E,\ AD \rightarrow C,\ BC \rightarrow E,\ BEC \rightarrow FD, E \rightarrow C,\ EC \rightarrow B\}$.

Příklad 6:
Vysvětlete pojem pokrytí množiny funkčních závislostí. Je F pokrytím G?

Řešení příkladu 6:
Pokrytí množiny funkčních závislostí F je každá množina funkčních závislostí G taková, že $F^+ = G^+$, kde F^+ (nazývaný uzávěr množiny F) značí množinu všech funkčních závislostí, které lze z F odvodit pomocí Armstrongových pravidel. F je pokrytím G. Stačí odvodit G z F a naopak.

Příklad 7:
Vysvětlete pojem uzávěru množiny atributů. Najděte A^+, B^+ a BC^+.

Řešení příkladu 7:
Uzávěr X^+ množiny atributů X vzhledem F je množina všech atributů funkčně závislých na F.

$$\begin{array}{lcl} A+ & = & \{A, B, C, D, E, F\} \\ B+ & = & \{B\} \\ BC+ & = & \{B, C, D, E, F\} \end{array}$$

Příklad 8:
Vysvětlete pojem redundantního atributu a najděte všechny z předchozího příkladu. Výslednou množinu FZ označte jako F'

Řešení příkladu 8:
Atribut a je redundantní v $X \rightarrow Y$, pokud $Y \subseteq (X \setminus \{a\})^+$. Výsledná $F' = \{A \rightarrow BCEF,\ BC \rightarrow DE,\ E \rightarrow CBDF\}$

Příklad 9:
Co je to redundantní funkční závislost? Najděte redundantní funkční závislosti v F'.

Řešení příkladu 9:
FZ f je redundantní v F, pokud $(F - \{f\})^+ = F^+$. Výsledná $F'' = \{A \rightarrow E,\ BC \rightarrow E,\ E \rightarrow BCDF\}$.

Příklad 10:
Najděte minimální pokrytí G.

Řešení příkladu 10:
Minimální pokrytí G je každé neredundantní (bez redundantních FZ) kanonické (maximálně dekomponované) pokrytí, které se skládá pouze z redukovaných FZ

(tj. bez redundantních atributů na levé straně). F'', které dekomponujeme na kanonické pokrytí je minimální pro G, protože je i pro F. Je důležité nejprve odstranit redundantní atributy a potom teprve závislosti. Pokud bychom hledali minimální pokrytí přímo z G, způsobilo by nedodržení tohoto pořadí, že FZ $A \rightarrow B$ bychom neshledali redundantní, i když je.

Příklad 11:
Vysvětlete pojem 1. normální forma (1NF).

Řešení příkladu 11:
Každý atribut schématu relace je elementárního typu a je nestrukturovaný. To umožňuje pracovat s daty v podobě plochých relací.

Příklad 12:
Vysvětlete pojem redundance. Najděte ve schématu z Příkladu 2 místo, které může způsobit redundance. Proč nám redundance vadí?

Řešení příkladu 12:
Redundance je ukládání stejné informace víckrát na různých místech. Ve schématu z Příkladu 2 jsou např. hodnoty atributu areal redundantní. Tolikrát, kolikrát se v relaci $Vyuka$ daná budova objeví, tolikrát je uveden i její areál. Jinou redundanci způsobuje atribut $telefon\_ucitele$. Redundance nám vadí z různých důvodů:

1. *ukládání stejné informace na více místech, což zvyšuje potřebný prostor*
2. *při vložení dat příslušejících jedné entitě je potřeba zároveň vložit data i o jiné entitě*
3. *při vymazání dat příslušejících jedné entitě je potřeba vymazat data patřící jiné entitě*
4. *pokud se změní jedna kopie redundantních dat, je třeba změnit i ostatní kopie, jinak se databáze stane nekonzistentní*

2-4 se někdy souhrně nazývají aktualizační anomálie.

Příklad 13:
Vysvětlete pojem 2. normální forma (2NF). Upravte schéma z Příkladu 2 tak, aby splňovalo podmínky 2NF.

Řešení příkladu 13:
2NF předpokládá schéma v 1NF a navíc říká, že ve schématu relace neexistují částečné závislosti neklíčových atributů na klíči, tj. že pro žádný neklíčový atribut neexistuje vlastní podmnožina klíče, na které bý závisel. Klíčem relačního schématu $Vyuka$ je $rc_u citele, cas$, příp. $mistnost, cas$ a existuje FZ $rc_u citele \rightarrow telefon_u citele$, čili neklíčový atribut $telefon\_ucitele$ závisí částečně na klíči. Dalším porušením je $mistnost \rightarrow budova$. Řešením je dekompozice podle těchto FZ:

$$Vyuka2(kod\_predmetu, cas, mistnost, rc\_ucitele, hodnoceni)$$
$$Mistnost2(mistnost, budova, areal)$$
$$Kontakt2(rc\_ucitele, telefon\_ucitele)$$

Příklad 14:
Vysvětlete pojem 3. normální forma (3NF). Upravte schéma z předchozího příkladu tak, aby splňovalo podmínky 3NF.

Řešení příkladu 14:
3NF předpokládá schéma ve 2NF a navíc říká, že žádný neklíčový atribut není tranzitivně závislý na žádném klíči. Přesněji, v daném schématu $R(A, F)$ platí alespoň jedna z podmínek pro každou závislost $X \to a$ (kde X je podmnožinou A, a je z A):

- *závislost je triviální*
- *X je nadklíč*
- *a je částí klíče*

Ve schématu $Mistnost2$ je klíčem $mistnost$ a platí FZ $budova \to areal$. Atribut $areal$ je tedy tranzitivně závislý na klíči a schéma není ve 3NF. Přesněji řečeno závislost není triviální, $budova$ není nadklíč a $areal$ není částí klíče. FZ $kod\_predmetu, rc\_ucitele \to hodnoceni$ také porušuje 3NF. Řešením je opět dekompozice:

$$Vyuka3(kod\_predmetu, cas, mistnost, rc\_ucitele)$$
$$Hodnoceni3(kod\_predmetu, rc\_ucitele, hodnoceni)$$
$$Mistnost3(mistnost, budova)$$
$$Budova3(budova, areal)$$
$$Kontakt3(rc\_ucitele, telefon\_ucitele)$$

Příklad 15:
Vysvětlete pojem Boyce-Coddovy normální formy (BCNF). Upravte schéma z předchozího příkladu tak, aby splňovalo podmínky BCNF, pokud uvažujete navíc i FZ $kod\_predmetu \to rc\_ucitele$.

Řešení příkladu 15:
BCNF zpřísňuje 3NF. Požaduje, aby každý atribut byl závislý na klíči. Přesněji, zakazuje poslední možnost z podmínek 3NF, tj. v daném schématu $R(A, F)$ platí alespoň jedna z podmínek pro každou závislost $X \to a$ (kde X je podmnožinou A, a je z A):

- *závislost je triviální*
- *X je nadklíč*

Pro funkční závislost $kod\_predmetu \to rc\_ucitele$ nejsou podmínky BCNF splněny, protože se nejedná o triviální závislost a $kod\_predmetu$ není nadklíč (podle specifikovaných funkčních závislostí). Řešením je znovu dekompozice:

$$Vyuka4(kod\_predmetu, cas, mistnost)$$
$$Hodnoceni4(kod\_predmetu, hodnoceni)$$
$$Uci4(kod\_predmetu, rc\_ucitele)$$
$$Mistnost4(mistnost, budova)$$
$$Budova4(budova, areal)$$
$$Kontakt4(rc\_ucitele, telefon\_ucitele)$$

Je navíc možné spojit schémata $Hodnoceni4$ a $Uci4$ do jednoho.

DBI025
4. Cvičení: Funkční Závislosti a Normální Formy - Algoritmy

Martin Nečaský, KSI

19. března 2007

Pro toto cvičení uvažujte následující schéma relace a funkční závislosti:

$Vyuka(kod\_predmetu, cas, mistnost, rc\_ucitele, telefon\_ucitele, hodnoceni, budova, areal)$

a následující funkční závislosti:

$$
\begin{array}{ll}
mistnost, cas & \rightarrow kod\_predmetu \\
kod\_predmetu & \rightarrow mistnost \\
mistnost & \rightarrow budova, areal \\
budova & \rightarrow areal \\
kod\_predmetu, mistnost & \rightarrow rc\_ucitele \\
rc\_ucitele & \rightarrow telefon\_ucitele \\
kod\_predmetu, rc\_ucitele & \rightarrow mistnost, budova, areal \\
kod\_predmetu, rc\_ucitele, cas & \rightarrow hodnoceni \\
rc\_ucitele, cas & \rightarrow mistnost
\end{array}
$$

Příklad 1:
Pomocí algoritmu pro určení atributového uzávěru z přednášky určete atributové uzávěry následujících množin atributů

$$
\begin{array}{l}
\{kod\_predmetu\} \\
\{kod\_predmetu, mistnost\} \\
\{kod\_predmetu, mistnost, cas\}
\end{array}
$$

Popište, jak algoritmus pracuje.

Řešení příkladu 1:
Uzávěr budujeme po krocích. Dokud se rozšiřuje, pokračujeme. Krok, ve kterém se uzávěr již nerozšíří, je poslední. Uzávěr C množiny atributů X je iniciován jako X. Krokem se rozumí aplikace všech funkčních závislostí na množinu C. Čili pro $\{kod\_predmetu\}$:

$$
\begin{array}{l}
\{kod\_predmetu\}^+ = \{kod\_predmetu\} \\
\{kod\_predmetu\}^+ = \{kod\_predmetu, mistnost\} \\
\{kod\_predmetu\}^+ = \{kod\_predmetu, mistnost, budova, areal, rc\_ucitele\} \\
\{kod\_predmetu\}^+ = \{kod\_predmetu, mistnost, budova, areal, rc\_ucitele, \\
\qquad telefon\_ucitele\}
\end{array}
$$

a pro další

$$\{kod\_predmetu, mistnost\}^+ = \{kod\_predmetu, mistnost, budova, areal, rc\_ucitele, telefon\_ucitele\}$$
$$\{kod\_predmetu, mistnost, cas\}^+ = \{kod\_predmetu, mistnost, cas, budova, areal, rc\_ucitele, telefon\_ucitele, hodnoceni\}$$

Příklad 2:
Vysvětlete k čemu slouží algoritmus příslušnosti funkční závislosti do množiny funkčních závislostí. Jak pro jeho realizaci můžete využít algoritmus atributového uzávěru? Použijte algoritmus ke zjištění, zda funkční závislosti

$$kod\_predmetu \rightarrow rc\_ucitele \quad (1)$$
$$rc\_ucitele \rightarrow kod\_predmetu \quad (2)$$
$$rc\_ucitele, cas \rightarrow kod\_predmetu \quad (3)$$

patří do původní množiny FZ.

Řešení příkladu 2:
Algoritmus umožňuje určit, zda daná funkční závislost $X \rightarrow Y$ patří do uzávěru množiny závislostí F. Pokud je odpověď kladná, přidáním této fukční závislosti do F nepřidáváme žádnou novou informaci. Pro realizaci tohoto algoritmu stačí určit, zda $Y \subseteq X+$ vzhledem k F. Ani nemusíme počítat celý $X+$, stačí jen dokud nezjistíme podmínku. Tedy:

$$\{kod\_predmetu\}^+ \supseteq \{kod\_predmetu, mistnost, budova, areal, rc\_ucitele\}$$
$$\{rc\_ucitele\}^+ = \{rc\_ucitele, telefon\_ucitele\}$$
$$\{rc\_ucitele, cas\}^+ \supseteq \{rc\_ucitele, cas, telefon\_ucitele, mistnost, kod\_predmetu\}$$

Tedy (1) *a* (3) *patří do původní množiny FZ a* (2) *nepatří.*

Příklad 3:
Popište, jak funguje algoritmus pro testování redundantních funkčních závislostí. Určete, zda

$$kod\_predmetu, rc\_ucitele \rightarrow mistnost, budova, areal$$

je redundantní v původní množině funkčních závislostí.

Řešení příkladu 3:
Funkční závislost $X \rightarrow Y$ je redundantní v F, pokud

$$(F \setminus \{X \rightarrow Y\})^+ = F^+$$

Stačí tedy určit, zda $X \rightarrow Y$ patří do $(F \setminus \{X \rightarrow Y\})^+$. To znamená určit, zda $Y \subseteq X^+$ vzhledem k $F \setminus \{X \rightarrow Y\}$. Máme zjistit, zda funkční závislost

$$kod\_predmetu, rc\_ucitele \rightarrow mistnost, budova, areal$$

(označme ji f) je redundantní v původní množině FZ, označme ji F. Stačí tedy určit, zda

$$\{mistnost, budova, adresa_budovy\} \subseteq \{kod\_predmetu, rc\_ucitele\}^+$$

vzhledem k $F \setminus \{f\}$. Platí

$$\{kod\_predmetu, rc\_ucitele\}^+ \supseteq \{kod\_predmetu, rc\_ucitele, mistnost, telefon\_ucitele, budova, areal\}$$

což znamená, že testovaná FZ je redundantní v F.

Příklad 4:
Popište, jak funguje algoritmus pro testování redundantních atributů. Určete, zda je některý z atributů na levé straně funkční závislosti

$$kod\_predmetu, rc\_ucitele, cas \rightarrow hodnoceni$$

redundantní.

Řešení příkladu 4:
Nechť $X \rightarrow Y$ je funkční závislost v F. Atribut $a \in X$ je redundantní, pokud platí

$$(X \setminus \{a\}) \rightarrow Y \in F$$

což znamená určit, zda $Y \subseteq (X \setminus \{a\})^+$ vzhledem k F. Pokud se nám podaří určit, že $a \in X \setminus \{a\})^+$ vzhledem k F, máme podmínku ověřenu také, protože potom $(X \setminus \{a\}) \rightarrow X$ a k tomu $X \rightarrow Y$ a tedy z tranzitivity $(X \setminus \{a\}) \rightarrow Y$ patří do uzávěru F. Při ověřování podmínky můžeme použít i původní funkční závislost $X \rightarrow Y$. Jedná se totiž o tvrzení, které v uzávěru stále platí (i po odstranění redundantního atributu musí být $X \rightarrow Y$ odvoditelná). Máme zjistit, zda je některý z atributů v

$$kod\_predmetu, rc\_ucitele, cas \rightarrow hodnoceni$$

redundantní. Platí

$$\{kod\_predmetu, rc\_ucitele, cas\}^+ = \{kod\_predmetu, rc\_ucitele, cas, mistnost, budova, areal, telefon\_ucitele, hodnoceni\}$$

Potřebujeme otestovat, zda funkční závislosti s levými stranami redukovanými o testované atributy patří do uzávěru původní množiny funkčních závislostí, tj. zda z redukované levé strany odvodíme původní pravou stranu.

- *test redundance atributu kod_predmetu:*

  $$\{rc\_ucitele, cas\}^+ \supseteq \{rc\_ucitele, cas, telefon\_ucitele, mistnost, kod\_predmetu\}$$

  Odvodili jsme testovaný atribut a je tedy možné odvodit i celou pravou stranu, tedy kod_predmetu je redundantní v dané FZ.

- *test redundance atributu rc_ucitele:*

  $$\{kod\_predmetu, cas\}^+ \supseteq \{kod\_predmetu, cas, mistnost, budova, areal, rc\_ucitele\}$$

  Opět jsme odvodili testovaný atribut, tedy $rc_ucitele$ je redundantní v dané FZ.

- *test redundance atributu cas:*

  $$\{kod\_predmetu, rc\_ucitele\}^+ \supseteq \{kod\_predmetu, rc\_ucitele, mistnost, budova, areal, telefon\_ucitele\}$$

  Pravou stranu nelze odvodit, tedy čas není redundantní v dané FZ.

Vyšlo tedy, že kod_predmetu a rc_ucitele jsou redundantní (ne oba najednou, jen za předpokladu, že druhý z nich je na levé straně přítomen) a cas není.

Příklad 5:
Popište, jak lze využít předchozí algoritmy ke konstrukci minimálního pokrytí množiny FZ. Určete tímto způsobem minimální pokrytí původní množiny FZ.

Řešení příkladu 5:
Minimální pokrytí lze zkonstruovat následujícím způsobem:

1. *dekomponovat každou funkční závislost na elementární funkční závislosti*
2. *z každé funkční závislosti odstranit redundantní atributy na levé straně*
3. *odstranit redundantní závislosti*

Tedy:

1.
$$
\begin{aligned}
&mistnost, cas \rightarrow kod\_predmetu\\
&kod\_predmetu \rightarrow mistnost\\
&mistnost \rightarrow budova\\
&mistnost \rightarrow areal\\
&budova \rightarrow areal\\
&kod\_predmetu, mistnost \rightarrow rc\_ucitele\\
&rc\_ucitele \rightarrow telefon\_ucitele\\
&kod\_predmetu, rc\_ucitele \rightarrow mistnost\\
&kod\_predmetu, rc\_ucitele \rightarrow budova\\
&kod\_predmetu, rc\_ucitele \rightarrow areal\\
&kod\_predmetu, rc\_ucitele, cas \rightarrow hodnoceni\\
&rc\_ucitele, cas \rightarrow mistnost
\end{aligned}
$$

2.
$$
\begin{aligned}
&mistnost, cas \rightarrow kod\_predmetu\\
&kod\_predmetu \rightarrow mistnost\\
&mistnost \rightarrow budova\\
&mistnost \rightarrow areal\\
&budova \rightarrow areal\\
&kod\_predmetu \rightarrow rc\_ucitele\\
&rc\_ucitele \rightarrow telefon\_ucitele\\
&kod\_predmetu \rightarrow mistnost\\
&kod\_predmetu \rightarrow budova\\
&kod\_predmetu \rightarrow areal\\
&kod\_predmetu, cas \rightarrow hodnoceni\\
&rc\_ucitele, cas \rightarrow mistnost
\end{aligned}
$$

3.
$$
\begin{aligned}
&mistnost, cas \rightarrow kod\_predmetu\\
&mistnost \rightarrow budova\\
&budova \rightarrow areal\\
&kod\_predmetu \rightarrow rc\_ucitele\\
&rc\_ucitele \rightarrow telefon\_ucitele\\
&kod\_predmetu \rightarrow mistnost\\
&kod\_predmetu, cas \rightarrow hodnoceni\\
&rc\_ucitele, cas \rightarrow mistnost
\end{aligned}
$$

Příklad 6:
Popište algoritmus nalezení prvního klíče. Použijte ho k nalezení prvního klíče ve vašem minimálním pokrytí.

Řešení příkladu 6:
Je-li A množina všech atributů daného schématu relace, stačí nalézt redundantní

atributy v $A \rightarrow A$. Tedy pro A rovno množině všech atributů ve schématu naší relace postupujeme následovně:

(1) $kod\_predmetu, cas, mistnost, rc\_ucitele, telefon\_ucitele, hodnoceni, budova, areal \rightarrow A$
(2) $cas, mistnost, rc\_ucitele, telefon\_ucitele, hodnoceni, budova, areal \rightarrow A$ ($kod\_predmetu$ *je redundantni*)
(3) $cas, mistnost, rc\_ucitele, telefon\_ucitele, hodnoceni, budova, areal \rightarrow A$ (cas *neni redundantni*)
(4) $cas, rc\_ucitele, telefon\_ucitele, hodnoceni, budova, areal \rightarrow A$ ($mistnost$ *je redundantni*)
(5) $cas, rc\_ucitele, telefon\_ucitele, hodnoceni, budova, areal \rightarrow A$ ($rc\_ucitele$ *neni redundantni*)
(6) $cas, rc\_ucitele \rightarrow A$ ($telefon\_ucitele, hodnoceni, budova, areal$ *jsou redundantni*)

Ověření:

$\{cas, rc\_ucitele\}^+ = \{cas, rc\_ucitele, telefon\_ucitele, mistnost, kod\_predmetu, budova, areal, hodnoceni\}$

Příklad 7:
Popište algoritmus nalezení všech klíčů (Lucchesi-Osborn z přednášky). Použijte ho k nalezení klíčů ve vašem minimálním pokrytí.

Řešení příkladu 7:
Algoritmus předpokládá znalost jednoho klíče. Dále předpokládá, že všechny FZ jsou netriviální, tj. pro každou $X \rightarrow Y$ platí, že X a Y jsou disjunktní. Jde vlastně o obecnější hledání ekvivalentních množin atributů. Dvě množiny atributů jsou ekvivalentní (vzhledem k dané množině FZ), pokud určují stejné množiny atributů. Algoritmus pracuje v krocích. Udržuje aktuální množinu klíčů. V prvním kroce tato množina obsahuje pouze první klíč, nalezený např. předchozím algoritmem. V každém kroku je zjištěno 0 a více nových klíčů. Pokud v daném kroce již žádný nový klíč není zjištěn algoritmus končí. Krok algoritmu prochází všechny funkční závislosti a pro každou FZ prochází všechny nové klíče v aktuální množině klíčů. Pro danou FZ $X \rightarrow Y$ a klíč K testuje, zda:

- *množiny Y a K mají neprázdný průnik*
- *neexistuje klíč K' v aktuální množině klíčů takový, že K' je podmnožinou $(K \cup X) \setminus Y$, jinak řečeno, testuje, zda pro každý klíč K' v aktuální množině klíčů platí, že K' není podmnožinou $(K \cup X) \setminus Y$*

Potom pravá strana bez redundantních atributů z $(K \cup X) \setminus Y \rightarrow A$ je nový klíč, kde A je množina všech atributů schématu relace. Tedy

(1) $Keys = \{(cas, rc\_ucitele)\}$
(2) $Keys = \{(cas, rc\_ucitele), (cas, kod\_predmetu)\}$
$z\ kod\_predmetu \rightarrow rc\_ucitele$
(3) $Keys = \{(cas, rc\_ucitele), (cas, kod\_predmetu), (cas, mistnost)\}$
$z\ mistnost, cas \rightarrow kod\_predmetu$
(4) $Keys = \{(cas, rc\_ucitele), (cas, kod\_predmetu), (cas, mistnost)\}$
$zrc\_ucitele, cas \rightarrow mistnost$

Příklad 8:
Rozeberte možnosti, kterými můžete zkonstruovat relační schéma tak, aby splňovalo podmínky normálních forem. Jaké jsou výhody/nevýhody těchto přístupů?

Řešení příkladu 8:
V zásadě jsou dvě krajní možnosti. První možností je vytvoření množiny schémat relací, např. převodem z ER diagramu a množiny funkčních závislostí, na jejichž základě se provede normalizace. Nevýhodou je možnost přílišného rozdrobení dat. Druhou možností je vytvoření jednoho globálního schématu a jeho rozdělení až na základě funkčních závislostí. Je to však méně intuitivní. Vhodné je volit postup někde mezi těmito přístupy.

Příklad 9:
Vysvětlete pojem "bezestrátovost". Uvažujte atributy $\{kod\_predmetu, rc\_ucitele, mistnost\}$ a funkční závislost $kod\_predmetu \rightarrow mistnost$. Která z dekompozic

(a) $\{kod\_predmetu, rc\_ucitele\}, \{kod\_predmetu, mistnost\}$
(b) $\{kod\_predmetu, rc\_ucitele\}, \{rc\_ucitele, mistnost\}$
(c) $\{kod\_predmetu, mistnost\}, \{rc\_ucitele, mistnost\}$

je bezestrátová?

Řešení příkladu 9:
Jedná se o vlastnost dekompozice, která zaručuje korektní rekonstrukci univerzální relace z dekomponovaných relací (rekonstrukcí získáme opět původní data ve stavu, v jakém byla před dekompozicí). Bezestrátovost lze popsat podmínkami:

- *Nechť $R(XYZ, F)$ je univerzální schéma, kde $Y \rightarrow Z$ je v F. Potom dekompozice $R1(YZ, F1)$, $R2(YX, F2)$ je bezztrátová.*
- *Dekompozice $R(A, F)$ do $R1(A1, F1)$, $R2(A2, F2)$ je bezztrátová, jestliže platí $(A1 \cap A2) \rightarrow A1$ nebo $(A2 \cap A1) \rightarrow A2$*

Podle takových definic je případ (a) bezestrátová dekompozice. V případech (b) a (c) není bezestrátovost zaručena. V případě (b) jsou ke každému předmětu přiřazeny všechny místnosti, ve kterých učí učitelé daného předmětu (i jiné předměty). V případě (c) jsou k danému předmětu přiřazení všichni učitelé, kteří něco učí v místnosti, kde se učí tento předmět (i učitelé, kteří tu učí jiné předměty).

Příklad 10:
Vysvětlete pojem "zachování pokrytí závislostí" v případě dekompozice.

Řešení příkladu 10:
Jedná se o vlastnost dekompozice, která zaručuje zachování všech funkčních závislostí. Pokud dekomponujeme $R(A, F)$ na $R1(A1, F1)$, ..., $Rn(An, Fn)$, potom dekompozice zachovává pokrytí závislostí, pokud sjednocení množin $F1^+$, ..., Fn^+ se rovná množině F^+. Pokrytí závislostí může být narušeno dvěma způsoby

- *při dekompozici F neodvodíme všechny FZ platné v Fi, tj. ztratíme FZ, která má přímo platit v jednou dílčím schématu*

- *i když odvodíme všechny platné (tj. provedeme projekci F^+), můžeme v důsledku ztratit FZ, která platí napříč schématy*

Příklad 11:
Popište algoritmus dekompozice relačních schémat a jeho vlastnosti. Dekomponujte původní schéma relace s vaším minimálním pokrytím.

Řešení příkladu 11:
Algoritmus dekompozice umožňuje dekomponovat relační schéma do BCNF a zachovává bezeztrátovost. Nezachovává pokrytí závislostí. Pracuje v krocích. V každém kroku vybere schéma nějaké relace, které není v BCNF (pokud takové již neexistuje, končí). Pro vybrané schéma vybere některou z neklíčových FZ a provede dekompozici schématu podle této funkční závislosti. To může být zdrojem případného porušení zachování pokrytí původní množiny FZ. Pro náš příklad může dekompozice proběhnout takto:

1. *Původní schéma s minimálním pokrytím FZ a s určenými klíči.*

   $Vyuka(kod\_predmetu, cas, mistnost, rc\_ucitele, telefon\_ucitele, hodnoceni, budova, areal)$
   $cas, rc\_ucitele \rightarrow vse$
   $cas, kod\_predmetu \rightarrow vse$
   $cas, mistnost \rightarrow vse$
   $mistnost \rightarrow budova$
   $budova \rightarrow areal$
   $kod\_predmetu \rightarrow rc\_ucitele$
   $rc\_ucitele \rightarrow telefon\_ucitele$
   $kod\_predmetu \rightarrow mistnost$

2. *$mistnost \rightarrow budova$ porušuje 2NF.*

   $Vyuka1(kod\_predmetu, cas, mistnost, rc\_ucitele, telefon\_ucitele, hodnoceni, areal)$
   $cas, rc\_ucitele \rightarrow vse$
   $cas, kod\_predmetu \rightarrow vse$
   $cas, mistnost \rightarrow vse$
   $kod\_predmetu \rightarrow rc\_ucitele$
   $rc\_ucitele \rightarrow telefon\_ucitele$
   $kod\_predmetu \rightarrow mistnost$

   $Vyuka2(mistnost, budova) \quad (BCNF)$
   $mistnost \rightarrow budova$

3. $rc\_ucitele \rightarrow telefon\_ucitele$ *porušuje 2NF.*

$Vyuka1(kod\_predmetu, cas, mistnost, rc\_ucitele, hodnoceni, areal)$
$cas, rc\_ucitele \rightarrow vse$
$cas, kod\_predmetu \rightarrow vse$
$cas, mistnost \rightarrow vse$
$kod\_predmetu \rightarrow rc\_ucitele$
$kod\_predmetu \rightarrow mistnost$

$Vyuka2(mistnost, budova) \quad (BCNF)$
$mistnost \rightarrow budova$

$Vyuka3(rc\_ucitele, telefon\_ucitele) \quad (BCNF)$
$rc\_ucitele \rightarrow telefon\_ucitele$

4. $kod\_predmetu \rightarrow rc\_ucitele$ *porušuje BCNF.*

$Vyuka1(kod\_predmetu, cas, mistnost, hodnoceni, areal)$
$cas, kod\_predmetu \rightarrow vse$
$cas, mistnost \rightarrow vse$
$kod\_predmetu \rightarrow mistnost$

$Vyuka2(mistnost, budova) \quad (BCNF)$
$mistnost \rightarrow budova$

$Vyuka3(rc\_ucitele, telefon\_ucitele) \quad (BCNF)$
$rc\_ucitele \rightarrow telefon\_ucitele$

$Vyuka4(kod\_predmetu, rc\_ucitele) \quad (BCNF)$
$kod\_predmetu \rightarrow rc\_ucitele$

5. $kod\_predmetu \rightarrow mistnost$ *porušuje BCNF.*

$Vyuka1(kod\_predmetu, cas, hodnoceni, areal)$
$cas, kod\_predmetu \rightarrow vse$

$Vyuka2(mistnost, budova) \quad (BCNF)$
$mistnost \rightarrow budova$

$Vyuka3(rc\_ucitele, telefon\_ucitele) \quad (BCNF)$
$rc\_ucitele \rightarrow telefon\_ucitele$

$Vyuka4(kod\_predmetu, rc\_ucitele) \quad (BCNF)$
$kod\_predmetu \rightarrow rc\_ucitele$

$Vyuka5(kod\_predmetu, mistnost) \quad (BCNF)$
$kod\_predmetu \rightarrow mistnost$

Dekompozice tu nevypadá nejšíkovněji, ztratili jsme např. poměrně důležitou FZ $budova \rightarrow areal$. *Lze přidat schéma* $Vyuka6(budova, areal)$. *Jiný a lepší výsledek by byl, kdybychom jako první provedli dekompozici podle* $budova \rightarrow areal$. *Je proto nutné dát pozor a pokud jsme ztratily nějakou funkční závislost, je potřeba doplnit odpovídající schémata relací.*

Příklad 12:
Popište algoritmus "syntéza" relačních schémat a jeho vlastnosti. Dekomponujte podle něj původní schéma relace s vaším minimálním pokrytím.

Řešení příkladu 12:
Jde o algoritmus pro dekompozici do 3NF, zachovávající pokrytí závislostí. Základní verze nezachovává bezztrátovost. Bezztrátovost lze zajistit přidáním dalšího schématu do dekompozice, které obsahuje univerzální klíč (tj. nějaký klíč původního univerzálního schématu). Schéma v dekompozici, které je podmnožinou jiného můžeme vypustit. Můžeme také sloučit schémata s funkčně ekvivalentními klíči, ale tato operace může obecně porušit 3NF (nebo BCNF pokud jí bylo dosaženo). Pro náš příklad proběhne syntéza takto:

1. *Minimální pokrytí už máme:*

   $Vyuka(kod\_predmetu, cas, mistnost, rc\_ucitele, telefon\_ucitele, hodnoceni, budova, areal)$
   $mistnost, cas \rightarrow kod\_predmetu$
   $mistnost \rightarrow budova$
   $budova \rightarrow areal$
   $kod\_predmetu \rightarrow rc\_ucitele$
   $rc\_ucitele \rightarrow telefon\_ucitele$
   $kod\_predmetu \rightarrow mistnost$
   $kod\_predmetu, cas \rightarrow hodnoceni$
   $rc\_ucitele, cas \rightarrow mistnost$

2. *Kompozice levych stran:*

   $mistnost, cas \rightarrow kod\_predmetu$
   $mistnost \rightarrow budova$
   $budova \rightarrow areal$
   $kod\_predmetu \rightarrow rc\_ucitele, mistnost$
   $rc\_ucitele \rightarrow telefon\_ucitele$
   $kod\_predmetu, cas \rightarrow hodnoceni$
   $rc\_ucitele, cas \rightarrow mistnost$

3. *Vytvoreni schemat:*

   $Vyuka1(mistnost, cas, kod\_predmetu)$
   $mistnost, cas \rightarrow kod\_predmetu$

   $Vyuka2(mistnost, budova)$
   $mistnost \rightarrow budova$

   $Vyuka3(budova, areal)$
   $budova \rightarrow areal$

$Vyuka4(kod\_predmetu, rc\_ucitele, mistnost)$
$kod\_predmetu \rightarrow rc\_ucitele, mistnost$

$Vyuka5(rc\_ucitele, telefon\_ucitele)$
$rc\_ucitele \rightarrow telefon\_ucitele$

$Vyuka6(kod\_predmetu, cas, hodnoceni)$
$kod\_predmetu, cas \rightarrow hodnoceni$

$Vyuka7(rc\_ucitele, cas, mistnost)$
$rc\_ucitele, cas \rightarrow mistnost$

4. *Mozna spojeni schemat (ekvivalentni klice):*

$Vyuka1(mistnost, cas, kod\_predmetu, hodnoceni, rc\_ucitele)$
$mistnost, cas \rightarrow kod\_predmetu$
$kod\_predmetu, cas \rightarrow hodnoceni$
$rc\_ucitele, cas \rightarrow mistnost$

$Vyuka2(mistnost, budova)$
$mistnost \rightarrow budova$

$Vyuka3(budova, areal)$
$budova \rightarrow areal$

$Vyuka4(kod\_predmetu, rc\_ucitele, mistnost)$
$kod\_predmetu \rightarrow rc\_ucitele, mistnost$

$Vyuka5(rc\_ucitele, telefon\_ucitele)$
$rc\_ucitele \rightarrow telefon\_ucitele$

5. *Mozna spojeni schemat (podmnoziny atributu) - porusujeme BCNF:*

$Vyuka1(mistnost, cas, kod\_predmetu, hodnoceni, rc\_ucitele)$
$mistnost, cas \rightarrow kod\_predmetu$
$kod\_predmetu, cas \rightarrow hodnoceni$
$rc\_ucitele, cas \rightarrow mistnost$
$kod\_predmetu \rightarrow rc\_ucitele, mistnost$

$Vyuka2(mistnost, budova)$
$mistnost \rightarrow budova$

$Vyuka3(budova, areal)$
$budova \rightarrow areal$

$Vyuka5(rc\_ucitele, telefon\_ucitele)$
$rc\_ucitele \rightarrow telefon\_uitele$

DBI025
5. Cvičení: Relační Algebra

Martin Nečaský, KSI

2. dubna 2007

Pro toto cvičení uvažujte následující schéma relační databáze:

$Student(rc\_studenta, jmeno, adresa)$
$Ucitel(rc\_ucitele, jmeno, telefon, nazev\_katedry)$
$Katedra(nazev\_katedry, rc\_vedouciho)$
$Predmet(kod\_predmetu, nazev, anotace)$
$Mistnost(cislo\_mistnosti, kapacita, cislo\_budovy)$
$Budova(cislo\_budovy, adresa)$
$Prerekvizita(kod\_predmetu, kod\_prerekvizity)$
$Vyuka(kod\_predmetu, rc\_ucitele, typ, semestr, den, cas\_zacatku,$
$cislo\_mistnosti, hodnoceni)$
$Zapis(kod\_predmetu, rc\_studenta, vysledek, semestr)$

Každý příklad je dotazem v přirozeném jazyce položeným "přirozeným" uživatelem. Přepište tento dotaz do relační algebry.

Příklad 1:
Ukaž mi jména všech učitelů.

Řešení příkladu 1:

$$R_1 = Ucitel[jmeno]$$

Příklad 2:
Ukaž mi jména všech učitelů z katedry s názvem "KSI".

Řešení příkladu 2:

$$R_2 = Ucitel(nazev\_katedry = "KSI")$$

Příklad 3:
Ukaž mi učitele z katedry, jejímž vedoucím je "Novák".

Řešení příkladu 3:

$R_3^1 = (Katedra \times Ucitel)(rc\_vedouciho = rc\_ucitele \wedge jmeno = "Novak")[nazev\_katedr$
$R_3^1 = (Katedra[rc\_vedouciho = rc\_ucitele]Ucitel)(jmeno = "Novak")[nazev\_katedry]$
$R_3^1 = (Katedra * Ucitel < rc\_ucitele \rightarrow rc\_vedouciho >)(jmeno = "Novak")[nazev\_kate$
$R_3 = Ucitel * R_3^1$

Příklad 4:
Vypiš seznam předmětů, které se učí v pondělí nebo v pátek.

Řešení příkladu 4:

$R_4 = (Predmet * Vyuka)(den = "Po" \vee den = "Pa")$
$R_4 = Predmet * Vyuka(den = "Po" \vee den = "Pa")$
$R_4 = (Predmet * Vyuka)(den = "Po") \cup (Predmet * Vyuka)(den = "Pa")$

Příklad 5:
Vypiš předměty, které se neučí ani v pondělí ani v pátek.

Řešení příkladu 5:

$$R_5^1 = (Predmet * Vyuka)(den \neq "Po" \wedge den \neq "Pa")$$

Dotaz R_5^1 je špatně. Pokud jsou povoleny i předměty bez výuky, potom takové předměty nejsou ve výsledku zařazeny. Navíc pokud je předmět učen i jiné dny, objeví se ve výsledku také. Lze však využít množinový rozdíl, kterým v relační algebře modelujeme negaci.

$$R_5 = Predmet \setminus R_4[kod\_predmetu, nazev, anotace]$$

Příklad 6:
Ukaž učitele, kteří mají výuku s hodnocením alespoň 3.

Řešení příkladu 6:

$$R_6 = (Ucitel * Vyuka)(hodnoceni > 3)$$

Příklad 7:
Kteří učitelé mají nestíhatelný rozvrh? Rozvrh je nestíhatelný, pokud je čas na přejezd mezi budovami, ve kterých se výuka koná, menší než 1 hodina. Trvání výuky je 1,5 hodiny. Předpokládejte pro zjednodušení, že operace '-' (aritmetický rozdíl) s operandy typu čas vrací číslo udávající časový rozdíl v minutách.

Řešení příkladu 7:

$$R_7^1 = (Vyuka * Mistnost * Budova)[rc\_ucitele, semestr, den, hodina, cislo\_budovy]$$
$$R_7^2 = R_7^1 * R_7^1 < hodina \rightarrow hodina1, cislo\_budovy \rightarrow cislo\_budovy1 >$$
$$R_7^3 = R_7^2(hodina <= hodina1 \wedge cislo\_budovy \neq cislo\_budovy1)$$
$$R_7 = R_7^3((hodina1 - (hodina - 90)) <= 60)$$

Příklad 8:
Kteří studenti nemají zapsaný žádný předmět?

Řešení příkladu 8:

$$R_8 = Student \setminus (Zapis * Student)[rc\_studenta, jmeno, adresa]$$

Příklad 9:
Předměty, které učí někdo z KSI.

Řešení příkladu 9:

$$R_9 = (Predmet * Vyuka * Ucitel)(nazev\_katedry = "KSI")$$
$$[kod\_predmetu, nazev, anotace]$$

Pro další příklady, nechť $PredmetKSI = R_9$.

Příklad 10:
Kteří studenti mají zapsán nějaký předmět, který učí někdo z KSI?

Řešení příkladu 10:

$$R_{10} = (Student * Zapis * PredmetKSI)[rc\_studenta, jmeno, adresa]$$

Příklad 11:
Kteří studenti mají zapsány všechny předměty, které učí někdo z KSI? (Nápověda: nemáme sice přímo něco jako všeobecný kvantifikátor, ale máme implicitně něco jako existenční kvantifikátor (viz např. zadání předchozího příkladu, tj. spojení Student a Zapis a zpět projekce na atributy relace Student) a máme také množinovou operaci 'minus', která funguje jako negace. Pokud vám nápověda nedává smysl, podívejte se na základní pojmy predikátové logiky)

Řešení příkladu 11:

$$R_{11}^1 = Zapis[rc\_studenta] \times PredmetKSI[kod\_predmetu]$$
$$R_{11}^2 = R_{11}^1 \setminus Zapis[rc\_studenta, kod\_predmetu]$$
$$R_{11}^3 = R_{11}^2[rc\_studenta]$$
$$R_{11}^4 = Zapis[rc\_studenta] \setminus R_{11}^3$$
$$R_{11} = R_{11}^4 * Student$$

- R_{11}^1 *jsou všechny možné kombinace studentů a předmětů z KSI (všechny možné zápisy studentů do předmětů KSI),*
- R_{11}^2 *obsahuje pro každého studenta ty předměty z KSI, které nemá zapsány,*
- R_{11}^3 *obsahuje studenty, kteří nemají zapsán nějaký predmet z KSI,*
- R_{11}^4 *obsahuje studenty, kteří mají zapsány všechny předměty z KSI (pryč ty studenty, kterým něco chybí)*

$$
\begin{aligned}
R_{11} = (& \\
& Zapis[rc\_studenta] \\
& \setminus \\
& (\\
& \quad (Zapis[rc\_studenta] \times PredmetKSI[kod\_predmetu]) \\
& \quad \setminus \\
& \quad Zapis[rc\_studenta, kod\_predmetu] \\
&)[rc\_studenta] \\
) & * Student \\
= & (Zapis[rc\_studenta, kod\_predmetu] \div PredmetKSI[kod\_predmetu]) * Student
\end{aligned}
$$

Jedná se tedy o operaci dělení. Vybíráme taková rodná čísla, kde pro každý předmět z KSI je dvojice $rc\_studenta, kod\_predmetu$ *v tabulce*

$$Zapis[rc\_studenta, kod\_predmetu]$$

Příklad 12:
Kteří studenti mají zapsány pouze předměty, které učí někdo z KSI? Uvažujte pouze studenty, kteří mají něco zapsáno.

Řešení příkladu 12:

$$
\begin{aligned}
R_{12}^1 &= (Predmet \setminus PredmetKSI)[kod\_predmetu] \\
R_{12}^2 &= Zapis * R_{12}^1 \\
R_{12}^3 &= R_{12}^2[rc\_studenta] \\
R_{12}^4 &= Zapis[rc\_studenta] \setminus R_{12}^3 \\
R_{12} &= Student * R_{12}^4
\end{aligned}
$$

- R_{12}^1 *jsou předměty, které neučí nikdo z KSI,*
- R_{12}^2 *obsahuje zápisy předmětů, které neučí nikdo z KSI,*
- R_{12}^3 *obsahuje rodná čísla studentů, kteří mají zápis předmetu, který neuči nikdo z KSI,*

- R_{12}^4 *obsahuje rodná čísla studentů, kteří nemají zápis předmětů, které neučí nikdo z KSI (ale rozdíl děláme proti* $Zapis$*, tj. objeví se tam pouze studenti, kteří mají alespoň něco zapsáno, studenti, kteří nic zapsáno nemají, nejsou uvažováni)*

Příklad 13:
Studenti, kteří mají zapsány právě předměty, které učí někdo z KSI.

Řešení příkladu 13:

$$R_{13}^1 = (Predmet \setminus PredmetKSI)[kod\_predmetu]$$
$$R_{13}^2 = (Zapis * R_{12}^1)[rc\_studenta]$$
$$R_{13}^3 = (Zapis[rc\_studenta, kod\_predmetu] \div PredmetKSI[kod\_predmetu]) \setminus R_{13}^2$$
$$R_{13} = Student * R_{13}^3$$

- R_{13}^1 *jsou predměty, které neučí nikdo z KSI,*
- R_{13}^2 *obsahuje rodná čísla studentů, kteří mají nějaké zápisy předmětů, které neučí nikdo z KSI,*
- R_{13}^3 *od rodných čísel studentů, kteří mají zapsány všechny předmety z KSI odebereme rodná čísla těch studentů, kteří mají také zápis předmětu mimo KSI*

Příklad 14:
Studenti, kteří nemají zapsán žádný předmět, který učí někdo z KSI. Uvažujte i studenty, kteří nemají žádný zápis.

Řešení příkladu 14:

$$R_{14} = Student \setminus R_{10}[rc\_studenta, jmeno, adresa]$$

Příklad 15:
Zkuste přijít na nějaká omezení relační algebry

Řešení příkladu 15:
Rekurzivita, agregace.

DBI025
6. Cvičení: Relační Kalkuly

Martin Nečaský, KSI

2. dubna 2007

Pro toto cvičení uvažujte následující schéma relační databáze:

$Student(rc\_studenta, jmeno, adresa)$
$Ucitel(rc\_ucitele, jmeno, telefon, nazev\_katedry)$
$Katedra(nazev\_katedry, rc\_vedouciho)$
$Predmet(kod\_predmetu, nazev, anotace)$
$Mistnost(cislo\_mistnosti, kapacita, cislo\_budovy)$
$Budova(cislo\_budovy, adresa)$
$Prerekvizita(kod\_predmetu, kod\_prerekvizity)$
$Vyuka(kod\_predmetu, rc\_ucitele, typ, semestr, den, cas\_zacatku,$
$cislo\_mistnosti, hodnoceni)$
$Zapis(kod\_predmetu, rc\_studenta, vysledek, semestr)$

V následujícím textu jsou dotazy vyjádřené v DRK a NRK. V případě DRK, pokud by byl zápis příliš dlouhý, nevypisujeme všechny atributy, které vybíráme, ale jen klíče. Tj. např. pokud chceme učitele, tak vybíráme jen rodné číslo. Jinak bychom museli všechny atributy vypisovat a zápis by byl příliš dlouhý a nepřehledný.

Každý příklad je dotazem v přirozeném jazyce položeným "přirozeným" uživatelem. Vždy ho vyjádřete v DRK i NRK.

Příklad 1:
Ukaž mi jména všech učitelů.

Řešení příkladu 1:

$$DRK: \{(jmeno) : Ucitel(jmeno)\}$$
$$NRK: \{u[jmeno] : Ucitel(u)\}$$

Příklad 2:
Ukaž mi jména všech učitelů z katedry s názvem "KSI".

Řešení příkladu 2:

$$DRK: \{(rc\_ucitele) : Ucitel(rc\_ucitele, "KSI")\}$$
$$NRK: \{u : Ucitel(u) \wedge u.nazev\_katedry = "KSI"\}$$

Příklad 3:
Ukaž mi učitele z katedry, jejímž vedoucím je "Novák".

Řešení příkladu 3:

$$\begin{aligned} DRK: \{(rc\_ucitele) : \exists nazev\_katedry(\exists rc\_vedouciho(& \\ & Katedra(nazev\_katedry, rc\_vedouciho) \wedge \\ & Ucitel(rc\_vedouciho, "Novak") \wedge \\ & Ucitel(rc\_ucitele, nazev\_katedry)\ ))\} \\ NRK: \{u : Ucitel(u) \wedge \exists Katedra(k)\, \exists Ucitel(v)(& \\ & v.jmeno = "Novak" \wedge \\ & k.rc\_vedouciho = v.rc\_ucitele \wedge \\ & u.nazev\_katedry = k.nazev\_katedry)\} \end{aligned}$$

Příklad 4:
Vypiš seznam předmětů, které se učí v pondělí nebo v pátek.

Řešení příkladu 4:

$$\begin{aligned} DRK: \{(kod\_predmetu) : Predmet(kod\_predmetu) \wedge & \\ (Vyuka(kod\_predmetu, "Po") \vee Vyuka(kod\_predmetu, "Pa"))\} & \\ NRK: \{p : Predmet(p) \wedge \exists Vyuka(v)(v.kod\_predmetu = p.kod\_predmetu \wedge & \\ (v.den = "Po" \vee v.den = "Pa"))\} & \end{aligned}$$

Příklad 5:
Vypiš předměty, které se neučí ani v pondělí ani v pátek.

Řešení příkladu 5:

$$\begin{aligned} DRK: \{(kod\_predmetu) : Predmet(kod\_predmetu) \wedge & \\ \neg(Vyuka(kod\_predmetu, "Po") \vee Vyuka(kod\_predmetu, "Pa"))\} & \\ NRK: \{p : Predmet(p) \wedge \neg\exists Vyuka(v)(v.kod\_predmetu = p.kod\_predmetu \wedge & \\ (v.den = "Po" \vee v.den = "Pa"))\} & \end{aligned}$$

Příklad 6:
Ukaž učitele, kteří mají výuku s hodnocením alespoň 3.

Řešení příkladu 6:

$$DRK: \{(rc\_ucitele) : Ucitel(rc\_ucitele) \wedge \exists hodnoceni(Vyuka(rc\_ucitele, hodnoceni) \wedge 3 \leq hodnoceni)\}$$
$$NRK: \{u : Ucitel(u) \wedge \exists Vyuka(v)(v.rc\_ucitele = u.rc\_ucitele \wedge 3 \leq v.hodnoceni)\}$$

Příklad 7:
Kteří učitelé mají nestíhatelný rozvrh? Rozvrh je nestíhatelný, pokud je čas na přejezd mezi budovami, ve kterých se výuka koná, menší než 1 hodina. Trvání výuky je 1,5 hodiny. Předpokládejte pro zjednodušení, že operace '-' (aritmetický rozdíl) s operandy typu čas vrací číslo udávající časový rozdíl v minutách.

Řešení příkladu 7:

$$\begin{aligned} DRK: \{&(rc\_ucitele) : Ucitel(rc\_ucitele) \wedge \\ &\exists semestr, den, hodina1, cislo\_budovy1, hodina2, cislo\_budovy2(\\ &Vyuka(rc\_ucitele, semestr, den, hodina1, cislo\_budovy1) \wedge \\ &Vyuka(rc\_ucitele, semestr, den, hodina2, cislo\_budovy2) \wedge \\ &hodina1 \leq hodina2 \wedge cislo\_budovy1 \neq cislo\_budovy2 \wedge \\ &hodina2 - (hodina1 + 90) \leq 60)\} \end{aligned}$$

$$\begin{aligned} NRK: \{&u : Ucitel(u) \wedge \\ &\exists Vyuka(v1) \exists Vyuka(v2)(\\ &v1.rc\_ucitele = u.rc\_ucitele \wedge v2.rc\_ucitele = u.rc\_ucitele \wedge \\ &v1.semestr = v2.semestr \wedge v1.den = v2.den \wedge \\ &v1.hodina \leq v2.hodina \wedge v1.cislo\_budovy \neq v2.cislo\_budovy \wedge \\ &v2.hodina - (v1.hodina + 90) \leq 60)\} \end{aligned}$$

Příklad 8:
Kteří studenti nemají zapsaný žádný předmět?

Řešení příkladu 8:

$$DRK: \{(rc\_studenta) : Student(rc\_studenta) \wedge \nexists kod\_predmetu(Zapis(rc\_studenta, kod\_predmetu))\}$$
$$NRK: \{s : Student(s) \wedge \nexists Zapis(z)(z.rc\_studenta = s.rc\_studenta)\}$$

Příklad 9:
Předměty, které učí někdo z KSI.

Řešení příkladu 9:

$$
\begin{aligned}
DRK : \{&(kod\_predmetu) : Predmet(kod\_predmetu) \wedge \exists rc\_ucitele(\\
&Vyuka(rc\_ucitele, kod\_predmetu) \wedge \\
&Ucitel(rc\_ucitele, nazev\_katedry : "KSI")\} \\
NRK : \{&p : Predmet(p) \wedge \exists Vyuka(v) \exists Ucitel(u)(\\
&v.kod\_predmetu = p.kod\_predmetu \wedge \\
&v.rc\_ucitele = u.rc\_ucitele \wedge \\
&u.nazev\_katedry = "KSI")\}
\end{aligned}
$$

Příklad 10:
Kteří studenti mají zapsán nějaký předmět, který učí někdo z KSI?

Řešení příkladu 10:

$$
\begin{aligned}
DRK : \{&(rc\_studenta) : Student(rc\_studenta) \wedge \exists kod\_predmetu(\\
&Predmet(kod\_predmetu) \wedge \\
&Zapis(rc\_studenta, kod\_predmetu) \wedge \\
&\exists rc\_ucitele(\\
&\quad Vyuka(rc\_ucitele, kod\_predmetu) \wedge \\
&\quad Ucitel(rc\_ucitele, nazev\_katedry : "KSI"))\} \\
NRK : \{&s : Student(s) \wedge \\
&\exists Predmet(p) \exists Zapis(z) \exists Vyuka(v) \exists Ucitel(u)(\\
&\quad s.rc\_studenta = z.rc\_studenta \wedge \\
&\quad p.kod\_predmetu = z.kod\_predmetu \wedge \\
&\quad v.kod\_predmetu = predmet.kod\_predmetu \wedge \\
&\quad v.rc\_ucitele = u.rc\_ucitele \wedge \\
&\quad u.nazev\_katedry = "KSI")\}
\end{aligned}
$$

Příklad 11:
Kteří studenti mají zapsány všechny předměty, které učí někdo z KSI?

Řešení příkladu 11:

$$
\begin{aligned}
DRK : \{&(rc\_studenta) : Student(rc\_studenta) \wedge \forall kod\_predmetu(\\
&(Predmet(kod\_predmetu) \wedge \exists rc\_ucitele(\\
&\quad Vyuka(rc\_ucitele, kod\_predmetu) \wedge \\
&\quad Ucitel(rc\_ucitele, nazev\_katedry : "KSI"))) \\
&\Rightarrow Zapis(rc\_studenta, kod\_predmetu))\} \\
NRK : \{&s : Student(s) \wedge \forall Predmet(p)(\\
&\exists Vyuka(v) \exists Ucitel(u)(\\
&\quad v.kod\_predmetu = p.kod\_predmetu \wedge \\
&\quad v.rc\_ucitele = u.rc\_ucitele \wedge \\
&\quad u.nazev\_katedry = "KSI") \\
&\Rightarrow \exists Zapis(z)(\\
&\quad s.rc\_studenta = z.rc\_studenta \wedge \\
&\quad p.kod\_predmetu = z.kod\_predmetu)\}
\end{aligned}
$$

Příklad 12:
Kteří studenti mají zapsány pouze předměty, které učí někdo z KSI? Uvažujte pouze studenty, kteří mají něco zapsáno.

Řešení příkladu 12:

$$
\begin{aligned}
DRK : \{&(rc\_studenta) : Student(rc\_studenta) \wedge \forall kod\_predmetu(\\
&(Predmet(kod\_predmetu) \wedge Zapis(rc\_studenta, kod\_predmetu)) \\
&\Rightarrow \exists rc\_ucitele(\\
&\quad Vyuka(rc\_ucitele, kod\_predmetu) \wedge \\
&\quad Ucitel(rc\_ucitele, nazev\_katedry : "KSI"))\} \\
NRK : \{&s : Student(s) \wedge \forall Predmet(p)(\\
&\exists Zapis(z)(\\
&\quad s.rc\_studenta = z.rc\_studenta \wedge \\
&\quad p.kod\_predmetu = z.kod\_predmetu) \\
&\Rightarrow \exists Vyuka(v) \exists Ucitel(u)(\\
&\quad v.kod\_predmetu = p.kod\_predmetu \wedge \\
&\quad v.rc\_ucitele = u.rc\_ucitele \wedge \\
&\quad u.nazev\_katedry = "KSI")\}
\end{aligned}
$$

Příklad 13:
Studenti, kteří mají zapsány právě předměty, které učí někdo z KSI.

Řešení příkladu 13:

$$
\begin{aligned}
DRK:\ \{&(rc\_studenta) : Student(rc\_studenta) \wedge \forall kod\_predmetu(\\
&\quad Predmet(kod\_predmetu) \Rightarrow \\
&\quad (\exists rc\_ucitele(Vyuka(rc\_ucitele, kod\_predmetu) \wedge \\
&\qquad Ucitel(rc\_ucitele, nazev\_katedry : "KSI"))) \\
&\quad \Leftrightarrow Zapis(rc\_studenta, kod\_predmetu))\}
\end{aligned}
$$

$$
\begin{aligned}
NRK:\ \{&s : Student(s) \wedge \forall Predmet(p)(\\
&\quad \exists Vyuka(v) \exists Ucitel(u)(\\
&\qquad v.kod\_predmetu = p.kod\_predmetu \wedge \\
&\qquad v.rc\_ucitele = u.rc\_ucitele \wedge \\
&\qquad u.nazev\_katedry = "KSI") \\
&\quad \Leftrightarrow \exists Zapis(z)(\\
&\qquad s.rc\_studenta = z.rc\_studenta \wedge \\
&\qquad p.kod\_predmetu = z.kod\_predmetu)\}
\end{aligned}
$$

Příklad 14:
Studenti, kteří nemají zapsán žádný předmět, který učí někdo z KSI. Uvažujte i studenty, kteří nemají žádný zápis.

Řešení příkladu 14:

$$
\begin{aligned}
DRK:\ \{&(rc\_studenta) : Student(rc\_studenta) \wedge \forall kod\_predmetu(\\
&\quad (Predmet(kod\_predmetu) \wedge Zapis(rc\_studenta, kod\_predmetu)) \\
&\quad \Rightarrow \nexists rc\_ucitele(\\
&\qquad Vyuka(rc\_ucitele, kod\_predmetu) \wedge \\
&\qquad Ucitel(rc\_ucitele, nazev\_katedry : "KSI"))\}
\end{aligned}
$$

$$
\begin{aligned}
NRK:\ \{&s : Student(s) \wedge \forall Predmet(p)(\\
&\quad \exists Zapis(z)(\\
&\qquad s.rc\_studenta = z.rc\_studenta \wedge \\
&\qquad p.kod\_predmetu = z.kod\_predmetu) \\
&\quad \Rightarrow \nexists Vyuka(v) \exists Ucitel(u)(\\
&\qquad v.kod\_predmetu = p.kod\_predmetu \wedge \\
&\qquad v.rc\_ucitele = u.rc\_ucitele \wedge \\
&\qquad u.nazev\_katedry = "KSI")\}
\end{aligned}
$$

Příklad 15:
Dvojice studentů, kteří mají zapsány stejné předměty.

Řešení příkladu 15:

$$
\begin{aligned}
DRK : \{&(rc\_studenta1, rc\_studenta2) : \\
&Student(rc\_studenta1) \wedge Student(rc\_studenta2) \wedge \\
&rc\_studenta1 < rc\_studenta2 \wedge \\
&\forall kod\_predmetu(Predmet(kod\_predmetu) \Rightarrow \\
&\quad (Zapis(rc\_studenta1, kod\_predmetu) \\
&\quad \Leftrightarrow \\
&\quad Zapis(rc\_studenta2, kod\_predmetu)\ ) \\
&)\}
\end{aligned}
$$

$$
\begin{aligned}
NRK : \{&s1, s2 : Student(s1) \wedge Student(s2) \wedge s1.rc\_studenta < s2.rc\_studenta \wedge \\
&\forall Predmet(p)(\\
&\quad \exists Zapis(z)(\\
&\quad\quad s1.rc\_studenta = z.rc\_studenta \wedge \\
&\quad\quad p.kod\_predmetu = z.kod\_predmetu) \\
&\quad \Leftrightarrow \\
&\quad \exists Zapis(z)(\\
&\quad\quad s2.rc\_studenta = z.rc\_studenta \wedge \\
&\quad\quad p.kod\_predmetu = z.kod\_predmetu) \\
&)\}
\end{aligned}
$$

Příklad 16:
Co to jsou bezpečné výrazy v RK?

DBI025
7. Cvičení: SQL

Martin Nečaský, KSI

14. dubna 2007

Pro toto cvičení uvažujte následující schéma relační databáze:

$Student(rc\_studenta, jmeno, adresa)$
$Ucitel(rc\_ucitele, jmeno, telefon, nazev\_katedry)$
$Katedra(nazev\_katedry, rc\_vedouciho)$
$Predmet(kod\_predmetu, nazev, anotace)$
$Mistnost(cislo\_mistnosti, kapacita, cislo\_budovy)$
$Budova(cislo\_budovy, adresa)$
$Prerekvizita(kod\_predmetu, kod\_prerekvizity)$
$Vyuka(kod\_predmetu, rc\_ucitele, typ, semestr, den, cas\_zacatku,$
$cislo\_mistnosti, hodnoceni)$
$Zapis(kod\_predmetu, rc\_studenta, vysledek, semestr)$

Příklad 1:
Ukaž mi jména všech učitelů.

Řešení příkladu 1:
```
1 SELECT *
2 FROM   Ucitel;
```

Příklad 2:
Ukaž mi jména všech učitelů z katedry s názvem "KSI".

Řešení příkladu 2:
```
1 SELECT *
2 FROM   Ucitel
3 WHERE  nazev_katedry = "KSI";
```

Příklad 3:
Ukaž mi učitele z katedry, jejímž vedoucím je "Novák".

Řešení příkladu 3:
```
1 SELECT U1.*
2 FROM   Ucitel AS U1, Katedra AS K, Ucitel AS U2
3 WHERE  U1.nazev_katedry = K.nazev_katedry AND
4        U2.rc_ucitele = K.rc_vedouciho AND
5        U2.jmeno_ucitele = "Novak";
```

Příklad 4:
Vypiš seznam předmětů, které se učí v pondělí nebo v pátek.

Řešení příkladu 4:
```
1 SELECT Predmet.*
2 FROM   Predmet JOIN Vyuka ON (Predmet.kod_predmetu=Vyuka.kod_predmetu)
3 WHERE  Vyuka.den = "Po" OR
4        Vyuka.den = "Pa"
```

```
1 SELECT Predmet.*
2 FROM   Predmet NATURAL JOIN Vyuka
3 WHERE  Vyuka.den = "Po" OR
4        Vyuka.den = "Pa"
```

Příklad 5:
Vypiš předměty, které se neučí ani v pondělí ani v pátek.

Řešení příkladu 5:
```
1 SELECT *
2 FROM   Predmet
3 WHERE  kod_predmetu NOT IN (
4          SELECT kod_predmetu
5          FROM   Vyuka
6          WHERE  Vyuka.den = "Po" OR
7                 Vyuka.den = "Pa"
8        );
```

Využíváme konstrukce nazev_sloupce IN poddodaz, která udává podmínku, že hodnota daná jménem sloupce je prvkem množiny definované poddotazem.

Poddotaz na řádcích 4-7 vrátí kódy předmětu které se učí v pondělí nebo v pátek. Na řádku 3 potom zajistíme výběr těch prředmětů, jejichž kód se nevyskytuje v množině vrácené poddotazem. Tj. kódy takových předmětů, které se neučí ani v pondělí ani v pátek.

```
 1 SELECT *
2 FROM   Predmet
3 WHERE  NOT EXISTS (
4          SELECT *
5          FROM   Vyuka
6          WHERE  Vyuka.kod_predmetu = Predmet.kod_predmetu AND
7                 (Vyuka.den = "Po" OR
8                 Vyuka.den = "Pa")
9        );
```

Druhá možnost využívá predikát $EXISTS$ umožňující test neprázdnosti množiny vrácené poddotazem v predikátu.

Poddotaz na řádcích 4-8 vrátí množinu výuk daného předmětu probíhající v pondělí a v pátek. Predikát pak zajistí zařazení do výsledku pouze těch předmětů, pro které žádná taková výuka neexistuje.

Příklad 6:
Ukaž učitele, kteří mají výuku s hodnocením alespoň 3.

Řešení příkladu 6:

```
1 SELECT Ucitel.*
2 FROM   Ucitel, Vyuka
3 WHERE  Ucitel.rc_ucitele = Vyuka.rc_ucitele AND
4        Vyuka.hodnoceni ≥ 3;
```

```
1 SELECT Ucitel.*
2 FROM   Ucitel NATURAL JOIN Vyuka
3 WHERE  Vyuka.hodnoceni ≥ 3;
```

```
1 SELECT *
2 FROM   Ucitel
3 WHERE  rc_ucitele IN (
4          SELECT rc_ucitele
5          FROM   Vyuka
6          WHERE  hodnoceni ≥ 3
7        );
```

```
1 SELECT *
2 FROM   Ucitel
3 WHERE  EXISTS (
4          SELECT *
5          FROM   Vyuka
6          WHERE  Ucitel.rc_ucitele = Vyuka.rc_ucitele AND
7                 hodnoceni ≥ 3
8        );
```

Je tedy opět několik možností, jak dotaz v SQL vyjádřit. Je však potřeba dát pozor na to, že IN a $EXISTS$ obecne NEJSOU EKVIVALENTNÍ. Souvisí to s krajními případy v hodnotách $NULL$. Ty se ale v přikladech výše neobjeví, protože pracujeme s klíči a ty nemůžou mít hodnotu $NULL$.

Příklad 7:
Kteří učitelé mají nestíhatelný rozvrh? Rozvrh je nestíhatelný, pokud je čas na přejezd mezi budovami, ve kterých se výuka koná, menší než 1 hodina. Trvání výuky je 1,5 hodiny. Předpokládejte pro zjednodušení, že operace '-' (aritmetický rozdíl) s operandy typu čas vrací číslo udávající časový rozdíl v minutách.

Řešení příkladu 7:
```
1 SELECT DISTINCT Ucitel.*
2 FROM   Vyuka AS V1, Vyuka AS V2, Ucitel AS U
3 WHERE  V1.rc_ucitele = U.rc_ucitele AND
4        V2.rc_ucitele = U.rc_ucitele AND
5        V1.semestr = V2.semestr AND
6        V1.den = V2.den AND
7        V1.cas_zacatku < V2.cas_zacatku AND
8        V1.budova <> V2.budova AND
9        V2.cas_zacatku-(V1.cas_zacatku+90)<60;
```

Příklad 8:
Kteří studenti nemají zapsaný žádný předmět?

Řešení příkladu 8:
```
1 SELECT *
2 FROM   Student
3 WHERE  NOT EXISTS (
4          SELECT *
5          FROM   Zapis
6          WHERE  Zapis.rc_studenta = Student.rc_studenta
7        );
```

Příklad 9:
Předměty, které učí někdo z KSI.

Řešení příkladu 9:
```
1 SELECT DISTINCT Predmet.*
2 FROM   Predmet NATURAL JOIN (Vyuka NATURAL JOIN Ucitel)
3 WHERE  Ucitel.nazev_katedry = "KSI";
```

Příklad 10:
Kteří studenti mají zapsán nějaký předmět, který učí někdo z KSI?

Řešení příkladu 10:

```
01 SELECT *
02 FROM   Student
03 WHERE  EXISTS (
04           SELECT *
05           FROM   Zapis
06           WHERE  Zapis.rc_studenta = Student.rc_studenta AND
07                  EXISTS (
08                    SELECT *
09                    FROM   Vyuka NATURAL JOIN Ucitel
10                    WHERE  Vyuka.kod_predmetu =
11                             Zapis.kod_predmetu AND
12                           Ucitel.nazev_katedry = "KSI"
13                  )
14        );
```

```
01 SELECT DISTINCT Student.*
02 FROM   Student JOIN Zapis ON (Student.rc_studenta = Zapis.rc_studenta)
03           JOIN Vyuka ON (Zapis.kod_predmetu=Vyuka.kod_predmetu)
04              JOIN Ucitel ON (Vyuka.rc_ucitele=Ucitel.rc_ucitele)
05           )
06 WHERE  Ucitel.nazev_katedry = "KSI";
```

Příklad 11:
Kteří studenti mají zapsány všechny předměty, které učí někdo z KSI?

Řešení příkladu 11:

```
01 SELECT *
02 FROM   Student
03 WHERE  NOT EXISTS (
04           SELECT Predmet.*
05           FROM   Predmet NATURAL JOIN (Vyuka NATURAL JOIN Ucitel)
06           WHERE  Ucitel.nazev_katedry = "KSI" AND
07                  NOT EXISTS (
08                    SELECT *
09                    FROM   Zapis
10                    WHERE  Zapis.rc_studenta = Student.rc_studenta
AND
11                           Zapis.kod_predmetu = Predmet.kod_predmetu
12                  )
13        )
```

```
   01 SELECT *
02 FROM   Student
03 WHERE  NOT EXISTS (
04           SELECT Vyuka.*
05           FROM   Vyuka NATURAL JOIN Ucitel
06           WHERE  Ucitel.nazev_katedry = "KSI" AND
07                  NOT EXISTS (
08                    SELECT *
09                    FROM   Zapis
10                    WHERE  Zapis.rc_studenta = Student.rc_studenta
AND
11                           Zapis.kod_predmetu = Vyuka.kod_predmetu
12                  )
13        )
```

Je vhodné si dotaz v přirozeném jazyce přeformulovat do "SQL češtiny" (tj. pro kvantifikace máme pouze existenční kvantifikátor):

- *Ve výsledku bude každý student, který má zapsán **k**aždý předmět z KSI.*
- *Ve výsledku bude každý student takový, že pro **k**aždý předmet z KSI, **e**xistuje studentův zápis tohoto předmětu.*
- *Ve výsledku bude každý student takový, že **n**eexistuje předmět z KSI takový, že **n**eexistuje studentův zápis tohoto předmětu.*

Příklad 12:
Kteří studenti mají zapsány pouze předměty, které učí někdo z KSI? Uvažujte pouze studenty, kteří mají něco zapsáno.

Řešení příkladu 12:
```
01 SELECT *
02 FROM   Student
03 WHERE  NOT EXISTS (
04           SELECT *
05           FROM   Zapis
06           WHERE  Zapis.rc_studenta = Student.rc_studenta AND
07                  NOT EXISTS (
08                    SELECT *
09                    FROM   Vyuka NATURAL JOIN Ucitel
10                    WHERE  Zapis.kod_predmetu=Vyuka.kod_predmetu AND
11                           Ucitel.nazev_katedry = "KSI"
12        )
```

Opět je vhodné si dotaz přeformulovat:

- *Ve výsledku bude každý student, který má zapsány **p**ouze předměty z KSI.*
- *Ve výsledku bude každý student takový, že pro **k**aždý jeho zápis existuje vyučující zapsaného předmětu z KSI.*
- *Ve výsledku bude každý student takový, že **n**eexistuje žádný jeho zápis, pro který **n**eexistuje vyučující zapsaného předmětu z KSI.*

Příklad 13:
Studenti, kteří nemají zapsán žádný předmět, který učí někdo z KSI.

Řešení příkladu 13:
```
01 SELECT *
02 FROM   Student
03 WHERE  NOT EXISTS (
04           SELECT *
05           FROM   Zapis
06           WHERE  Zapis.rc_studenta = Student.rc_studenta AND
07                  EXISTS (
08                    SELECT *
09                    FROM   Vyuka NATURAL JOIN Ucitel
10                    WHERE  Zapis.kod_predmetu=Vyuka.kod_predmetu AND
11                           Ucitel.nazev_katedry = "KSI"
12        )
```

Opět je vhodné si dotaz přeformulovat:

- *Ve výsledku bude každý student, který nemá zapsány **ž**ádné předměty z KSI.*
- *Ve výsledku bude každý student takový, že **n**eexistuje žádný jeho zápis předmětu z KSI.*
- *Ve výsledku bude každý student takový, že **n**eexistuje žádný jeho zápis, pro který **e**xistuje vyučující zapsaného předmětu z KSI.*

Příklad 14:
Jaká je kapacita všech místností dohromady?

Řešení příkladu 14:
```
01 SELECT SUM(kapacita)
02 FROM   Mistnost
```

Příklad 15:
Jaká je největší kapacita místnosti?

Řešení příkladu 15:
```
01 SELECT MAX(kapacita)
02 FROM   Mistnost
```

Příklad 16:
Jaká je kapacita všech mísností v každé budově?

Řešení příkladu 16:
```
01 SELECT   SUM(kapacita)
02 FROM     Mistnost NATURAL JOIN Budova
03 GROUP BY cislo_budovy
```

Příklad 17:
Jaké jsou budovy s celkovou kapacitou místností větší než 100?

Řešení příkladu 17:
```
01 SELECT   Budova.cislo_budovy, SUM(kapacita)
02 FROM     Mistnost NATURAL JOIN Budova
03 GROUP BY Budova.cislo_budovy
04 HAVING   SUM(kapacita) ≥ 100
```

```
01 SELECT   *
02 FROM     Budova
02 WHERE    100 ≤ (
02            SELECT SUM(Mistnost.kapacita)
02            FROM   Mistnost
02            WHERE  Mistnost.cislo_budovy = Budova.cislo_budovy
02          )
```

Příklad 18:
Jaké jsou místnosti s největší kapacitou?

Řešení příkladu 18:
```
01 SELECT *
02 FROM   Mistnost
03 WHERE  kapacita >= ALL (
04          SELECT kapacita FROM Mistnost
05        )
```

```
01 SELECT *
02 FROM   Mistnost
03 WHERE  kapacita = (SELECT MAX(kapacita) FROM Mistnost)
```

Příklad 19:
Učitelé, kteří učí alespoň 5 různých předmětů s více než 100 zapsanými studenty v jednom semestru.

Řešení příkladu 19:
```
01 SELECT   Ucitel.rc_ucitele
02 FROM     Ucitel NATURAL JOIN Vyuka
03 WHERE    100 ≤ (
04            SELECT   COUNT(*)
05            FROM     Zapis
06            WHERE    Zapis.kod_predmetu=Vyuka.kod_predmetu AND
07                     Zapis.semestr=Vyuka.semestr
08          )
09 GROUP BY Ucitel.rc_ucitele
10 HAVING   COUNT(DISTINCT kod_predmetu) ≥ 5
```

$Zam(c\_zam, j\_zam, plat, vek)$
$Odd(j\_odd, rozpocet, c\_ved)$
$Pracuje(c\_zam, j\_odd, uvazek)$

1 OUTER JOIN

1. Zamestnanci, kteri nejsou vedouci. Vyjadrete dotaz nejprve pomoci mnozinovych operaci a potom zkuste dotaz vyjadrit pomoci outer join.

```
01 SELECT *
02 FROM   Zam
03 EXCEPT
04 SELECT Zam.*
05 FROM   Zam JOIN Odd ON (c_zam=c_ved)
```

```
01 SELECT *
02 FROM   Zam LEFT OUTER JOIN Odd ON (c_zam=c_ved)
03 WHERE  j_odd IS NULL;
```

2. Vysvetlete rozdil mezi nasledujicimi dotazy:

```
01 SELECT *
02 FROM   Zam LEFT OUTER JOIN Odd ON (c_zam=c_ved);
```

```
01 SELECT *
02 FROM   Zam RIGHT OUTER JOIN Odd ON (c_zam=c_ved);
```

```
01 SELECT *
02 FROM   Zam FULL OUTER JOIN Odd ON (c_zam=c_ved);
```

Vsechny dotazy nejprve provedou INNER JOIN na podmince c_zam=c_ved. Potom se ale lisi. Prvni dotaz k vysledku prida jeste vsechny radky z LEFT tabulky (Zam), ktere se v INNER JOIN nespojily a doplni atributy z Odd hodnotami NULL. Druhy dotaz dela to same, ale symetricky vzhledem k RIGHT tabulce (Odd). Treti dotaz dela obe doplneni zaroven.

3. Jmena oddeleni, ve kterych pracuji pouze zamestnanci mladsi 30 let.

```
01 SELECT  Odd.*
02 FROM    Odd LEFT OUTER JOIN
03         (Pracuje JOIN Zam
04           ON (
05             Pracuje.c_zam=Zam.c_zam AND
06             Zam.vek>30
07           )
08         ) ON (Odd.j_odd = Pracuje.j_odd)
09 WHERE Pracuje.c_zam IS NULL;
```

2 Vlastnosti agregacnich funkci, ALL, ANY, EXISTS, IN

(prevzato z J. Pokorny: Dotazovaci jazyky. Skripta UK, Vydavatelstvi Karolinum, 2002, 255 s. - zde naleznete i vice informaci o tomto tematu)

SQL zavadi NULL znacici nedefinovanou hodnotu. Muze se objevovat ve sloupeccich libovolneho datoveho typu, pokud tento sloupecek neni definovan jako NOT NULL. Porovnavani s NULL se ridi trochu jinymi pravidly. Mejme tabulku T se sloupeckem A datoveho typu INTEGER a nasledujici dotaz:

```
01 SELECT *
02 FROM   T
03 WHERE  A > 100
```

Podminka A > 100 se pro vsechny radky s hodnotou A mensi nebo rovnou 100 vyhodnoti jako FALSE a s hodnotou vetsi nez 100 jako TRUE. Pokud ale ma radek v A hodnotu NULL, tj. hodnotu nezname, nelze podminku vyhodnotit ani jako FALSE ani jako TRUE. Je potreba tedy doplnit dalsi logickou hodnotu - UNKNOWN. SQL tedy nepracuje s 2-hodnotovou logikou TRUE, FALSE ale s 3-hodnotovou logikou TRUE, FALSE, UNKNOWN. Radek se dostane do tabulky vysledku pouze v pripade, pokud je podminka za WHERE vyhodnocena jako TRUE. Kazde porovnani s NULL vede na UNKNOWN. Dokonce i v obracene variante, kdyz testujeme, zda je na radku v A hodnota NULL:

```
01 SELECT *
02 FROM   T
03 WHERE  A = NULL
```

Vysledek nevrati nic, protoze se podminka vyhodnoti vzdy jako UNKNOWN (NULL=NULL je UNKNOWN). Pokud chceme radky, ktere maji v A NULL, musime pouzit predikat IS:

```
01 SELECT *
02 FROM   T
03 WHERE  A IS NULL
```

Lze pouzit i IS NOT, ktery pri porovnani s NULL vrati TRUE, pokud je hodnota znama, tj. neni NULL:

```
01 SELECT *
02 FROM   T
03 WHERE  A IS NOT NULL
```

Necht $\emptyset$ znaci prazdnou tabulku a $\aleph$ znaci tabulku, jejiz radky obsahuji jenom NULL hodnoty. Takove tabulky asi explicitne v databazi mit nebudete, ale muzete je dostat jako vysledky poddotazu, takze pozor.

- COUNT
  - COUNT(*) zapocitava vse (duplicity i radky obsahujici same NULL)

  - COUNT(sloupec) ignoruje hodnoty NULL, implicitne ma ALL
  - COUNT na prazdne tabulce dava 0

- SUM, AVG, MIN, MAX
  - ignoruji NULL
  - na prazdne tabulce davaji NULL

- IN
  - A IN (∅) vraci FALSE (hodnota A tam urcite neni, protoze tam neni nic)
  - A IN (א) vraci UNKNOWN (jsou tam NULL a o nich nic nevime, tj. nevime jestli dany NULL je hodnota A nebo neni); obdobne pro NOT IN
  - A IN (T), kde T obsahuje radky se samymi NULL (mohou tam byt i "normalni" radky), potom, pokud A neni v "normalnich" radcich, vraci UNKNOWN; obdobne pro NOT IN

- ALL, ANY
  - A op ALL(∅) vraci TRUE
  - A op ANY(∅) vraci FALSE
  - A op ALL(א) vraci UNKNOWN
  - A op ANY(א) vraci UNKNOWN

- EXISTS
  - EXISTS vraci TRUE na neprazdne mnozine
  - jinak vraci FALSE

V nasledujicim dotazu vrati vnitrni SELECT tabulku, ktera obsahuje cisla vedoucich a NULL hodnoty pro nevedouci. Ve vnejsim SELECT bloku se pro kazdeho vedouciho NOT IN vyhodnoti jako FALSE (protoze ve vnitrnim je) a pro kazdeho nevedouciho se vyhodnoti jako UNKNOWN (sice tam neni, ale jsou tam NULL a nevime, co pod nimi je). Do vysledku tedy neni zahrnut zadny radek a vysledna tabulka je prazdna. Je to dost umely priklad, takhle to asi nikdo nenapise, ve slozitejsich a neprehlednejsich dotazech se ale neco podobneho muze objevit.

```
01 SELECT *
02 FROM   Zam AS Z
03 WHERE  Z.c_zam NOT IN (
04         SELECT c_ved
05         FROM Zam LEFT OUTER JOIN Odd ON (Zam.c_zam=Odd.c_ved)
06        );
```

Zkusme dotaz, ktery vrati zamestnance, kteri maji jiny plat nez vsichni zamestnanci starsi 50 let. Prvni varianta pomoci IN, druha varianta pomoci EXISTS.

```
01 SELECT *
02 FROM   Zam AS Z1
03 WHERE  Z1.plat NOT IN (
04     SELECT Z2.plat
05     FROM Zam AS Z2
06     WHERE Z2.vek>=50
07 );
```

```
01 SELECT *
02 FROM   Zam AS Z1
03 WHERE  NOT EXISTS (
04     SELECT Z2.plat
05     FROM Zam AS Z2
06     WHERE Z2.vek>=50 AND Z2.plat = Z1.plat
07 );
```

Pokud dojde k situaci, kdy zrovna u zamestnancu starsich 50 let mame nezname platy, tj. NULL, vysledky se lisi. V prvnim pripade vraci vnoreny SELECT tabulku, kde jsou same NULL platy a pro kazdeho zamestnance ve vnejsim SELECT je tedy NOT IN vyhodnoceno jako UNKNOWN a vysledek je prazdna tabulka. Ve druhem pripade vraci vnoreny SELECT vzdy prazdnou tabulku (podminka za WHERE je vzdy UNKNOWN protoze porovnavame s NULL). Pro kazdeho zamestnance ve vnejsim SELECT je tedy NOT EXISTS vyhodnoceno jako TRUE a vysledkem dotazu jsou vsichni zamestnanci.

Dale zkusme dotaz, ktery vrati vsechny zamestnance, kteri maji plat vetsi, nez vsichni zamestnanci starsi 50 let. Prvni varianta je pomoci ALL, druha pomoci MAX.

```
01 SELECT *
02 FROM   Zam AS Z1
03 WHERE  Z1.plat > ALL(
04     SELECT Z2.plat
05     FROM Zam AS Z2
06     WHERE Z2.vek>=50
07 );
```

```
01 SELECT *
02 FROM   Zam AS Z1
03 WHERE  Z1.plat > (
04     SELECT MAX(Z2.plat)
05     FROM Zam AS Z2
06     WHERE Z2.vek>=50
07 );
```

Pokud dojde k situaci, ze neexistuji zamestnanci starsi 50 let, vysledky se lisi. V prvnim pripade je vysledkem vnitrniho SELECT prazdna tabulka a ALL na prazdne tabulce vraci TRUE. Tedy pro kazdeho zamestnance ve vnejsim SELECT bude podminka vyhodnocena jako TRUE. Vysledkem prvni varianty

jsou tedy vsichni zamestnanci. Ve druhem pripade pocitame MAX pres prazdnou tabulkua vysledkem je tedy NULL hodnota. Porovnani na NULL vraci UNKNOWN. Pro kazdeho zamestnance ve vnejsim SELECT tedy dotavame hodnotu podminky UNKNOWN a vysledek dotazu je tedy prazdna tabulka.

DBI025
8. Cvičení: Komplexní příklad návrhu struktury databáze a dotazů

Martin Nečaský, KSI

16. dubna 2007

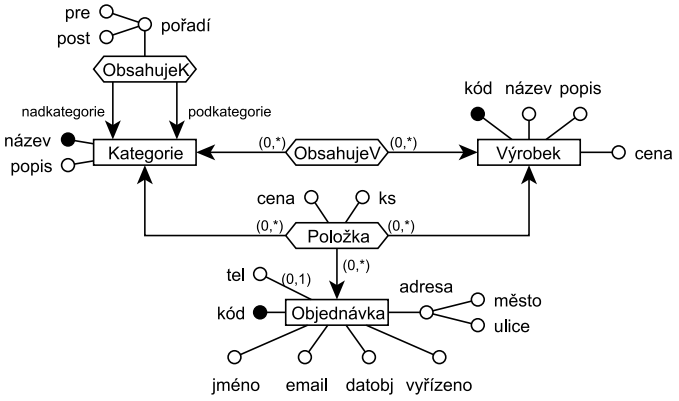
V tomto cvičení projdeme základy procesu tvorby databáze od zadání přes návrh struktury až po návrh dotazů. Uživatelské zadání se týká internetového obchodu a požadavky jsou následující:

"Základem internetového obchodu bude katalog výrobků. Bude obsahovat kategorie, ve kterých budou rozmístěny výrobky. Pro kategorii potřebujeme název a popis. Kategorie budou vytvářet stromovou strukturu. Chceme si upravovat pořadí kategorií v nadřazené kategorii. Pro výrobek potřebujeme kód, název, popis a cenu. Jeden výrobek může být v různých kategoriích ale nemusí být v žádné, pokud ho chceme schovat. Zákazníci budou kategorie procházet a vybrané výrobky si budou objednávat do košíku. Zajímá nás, z jaké kategorie si zákazník výrobek objednal. Na konci nákupu potom košík potvrdí, zadají své kontaktní údaje a odešlou objednávku. Musí být zadána adresa pro doručení, jméno zákazníka a email. Zákazník také může zadat svůj telefon, pokud chce abychom ho kontaktovali telefonicky v případě nějakých problémů. Interně budeme k objednávkám přiřazovat svoje identifikační kódy. Správce obchodu bude spravovat kategorie, výrobky i objednávky. U objednávek bude potřebovat rozlišit, zda je objednávka vyřízena či nikoliv a bude chtít filtrovat a řadit objednávky podle datumu objednání. Občas budeme odstraňovat výrobky, které si už nikdo neobjednal od zadaného datumu. Budeme také potřebovat následující statistiky prodeje:

- *pro danou kategorii seznam výrobků které byly objednány z této kategorie a nebo nějaké její podkategorie včetně počtu takových objednávek pro každý z výrobků*
- *pro daný výrobek a zadané období seznam kategorií, ze kterých byl výrobek objednán a počet objednávek tohoto výrobku z této kategorie za toto období*
- *celková suma objednávek pro dané období podle města z adresy pro doručení (abychom mohli optimalizovat naší dodavatelskou síť)*
- *pro dané období seznam výrobků, které nebyly v daném období ani jednou objednány"*

Příklad 1:
Navrhněte konceptuální E-R schéma podle uživatelského zadání.

Řešení příkladu 1:
E-R schéma pro naše zadání může vypadat např. takto:



kde atribut poradi u vztahového typu ObsahujeK budeme používat pro zajištění uživatelského pořadí podkategorií a zjišťovaní celého stromu podkategorií s daným vrcholem. Atribut má složku pre a post, kde první je pořadí v preorder a druhý v postorder průchodu stromem kategorií.

Příklad 2:
Sepište si v přirozeném jazyce seznam databázových dotazů, které budete potřebovat a v případě potřeby podle toho upravte E-R schéma. Takový seznam je většinou součástí analýzy uživatelského zadání, většinou je ale součástí komplexnějšího procesu vycházejícího z tzv. případů užití (use cases, viz. UML v pozdějších přednáškách).

Řešení příkladu 2:
Dotazy typu "SELECT":

1. *Název a popis dané kategorie.*
2. *Seznam názvů podkategorií dané kategorie seřazené v pořadí daným administrátorem.*
3. *Seznam názvů a cen výrobků v dané kategorii.*
4. *Adresa pro doručení, jméno zákazníka, email a telefon dané objednávky.*
5. *Seznam položek dané objednávky.*
6. *Suma cen z dané objednávky.*
7. *Seznam objednávek podle datumu vytvoření a příznaku uzavřeno/neuzavřeno.*
8. *Pro danou kategorii seznam výrobků které byly objednány z této kategorie a nebo nějaké její podkategorie včetně počtu takových objednávek pro každý z výrobků.*

9. *Pro daný výrobek a zadané období seznam kategorií, ze kterých byl výrobek objednán a počet objednávek tohoto výrobku z této kategorie za toto období.*

10. *Celková suma objednávek pro dané období podle města z adresy pro doručení.*

11. *Pro dané období seznam výrobků, které nebyly v daném období ani jednou objednány.*

12. *Seznam výrobků, které nejsou v žádné kategorii.*

Příklad 3:
Odvoďte z E-R schématu relační databázové schéma v co nejvyšší normální formě.

Řešení příkladu 3:
Z E-R schématu odvodíme např. následující schéma relační databáze:

$Kategorie(katid, nazev, popis, nadkatid, pre, post)$
$Vyrobek(vyrid, kod, nazev, popis, cena)$
$ObsahujeV(katid, vyrid)$
$Objednavka(objid, kod, jmeno, email, datobj, vyrizeno, mesto, ulice, tel)$
$Polozka(katid, vyrid, objid, cena, ks)$

s funkčními závislostmi pro jednotlivá schémata relací:

$$
\begin{array}{ll}
Kategorie: & \\
katid & \rightarrow all \\
pre & \rightarrow all \\
post & \rightarrow all \\
Vyrobek: & \\
vyrid & \rightarrow all \\
kod & \rightarrow all \\
ObsahujeV: & \\
katid, vyrid & \rightarrow all \\
Objednavka: & \\
objid & \rightarrow all \\
kod & \rightarrow all \\
Polozka: & \\
vyrid, objid & \rightarrow all
\end{array}
$$

Všimněte si, že jsme pro schémata relací přidali svoje umělé klíče. Ty budou číselné, neměnné a uměle generované, což jsou pro nás významné výhody (číselné jsou efektivnější než např. znakový kód výrobku, neměnné a uměle generované znamená efektivnější správu databáze). Pokud dokážeme zaručit, že klíč odvozený přímo z E-R schématu bude neměnný a bude podobně efektivní jako číselný, nemusíme umělý klíč zavádět (např. kód výrobku může být vždy řetězec o pevné délce nebo dokážeme zaručit nějakou rozumnou maximální délku - pak je efektivita podobná jako u číselného klíče). Musíme ale počítat, že např. kód výrobku bude zadávat administrátor a ten se může např. překlepnout ... . Zde volíme např. strategii umělý klíč pro každý entitní typ (i když, pokud bychom se zeptali zadavatele zjistíme, že chce aby systém generoval automaticky identifikační čísla objednávek jako řetězce tvaru číslo dne v měsíci + tříznakový kód

měsíce + poslední dvojčíslí roku + čtyřciferné pořadové číslo objednávky v daném dni, což by v pohodě mohl být i primární klíč). Někdy i u schémat relací vzniklých ze vztahových typů E-R schématu zavádíme jejich vlastní umělé klíče. Zde se to zdá výhodné u Polozka (budeme pracovat s položkou jako s entitou a tedy k ní budeme přistupovat přes nejaký klíč a dvojhodnotový není příliš efektivní na logické databázové úrovni - teď je otázka zda jsme tedy položky neměli modelovat jako slabé entitní typy, jenže pak se zesložití E-R schéma což není dobrý nápad, pokud chceme dobře vycházet se zadavatelem ;-)), čili

$$Polozka(polid, katid, vyrid, objid, cena, ks)$$

$$\begin{array}{ll} Polozka: & \\ vyrid, objid & \rightarrow all \\ polid & \rightarrow all \end{array}$$

Z E-R schématu se nám podařilo odvodit již schéma relační databáze v BCNF.

Příklad 4:
Odvoďte z relačního databázového schématu SQL skript pro vytvoření relační databáze včetně vložení počátečních dat.

Řešení příkladu 4:
Tabulka pro relaci "Kategorie" bude obsahovat sloupečky "katid" (číselný primární klíč), "nazev" (řetězec o max. 64 znacích, klíč), "popis" (řetězec o max. 256 znacích), "nadkatid" (číselný cizí klíč do tabulky "Kategorie" na primární klíč), "pre" (číselný klíč), "post" (číselný klíč). Pokud je smazána kategorie, budou smazány i všechny její podkategorie.

```
01 CREATE TABLE Kategorie (
02   katid    INTEGER      NOT NULL,
03   nazev    VARCHAR(64)  UNIQUE,
04   popis    VARCHAR(256),
05   nadkatid INTEGER,
06   pre      INTEGER      UNIQUE NOT NULL,
07   post     INTEGER      UNIQUE NOT NULL,
08   PRIMARY KEY (katid),
09   CONSTRAINT fk_kategorie_nadkatid
10         FOREIGN KEY (nadkatid) REFERENCES Kategorie (katid)
11         ON DELETE CASCADE
12 );
```

Pokud chceme mít primární klíč automaticky generovaný, využijeme některý z mechanismů nabízený konkrétním databázovým systémem, např. v IBM DB2:

```
02   katid    INTEGER      NOT NULL
02                           GENERATED ALWAYS AS IDENTITY
02                             (START WITH 1, INCREMENT BY 1,
```

V oracle můžeme využít např. tzv. sekvencí.

Tabulka pro relaci "Vyrobek".

```
01 CREATE TABLE Vyrobek (
02   vyrid    INTEGER      NOT NULL,
03   kod      CHAR(8)      UNIQUE NOT NULL,
04   nazev    VARCHAR(64)  UNIQUE NOT NULL,
05   popis    VARCHAR(256),
06   cena     NUMBER(9,2),
07   PRIMARY KEY (vyrid)
08 );
```

Tabulka pro relaci "ObsahujeV".

```
01 CREATE TABLE ObsahujeV (
03   katid    INTEGER      NOT NULL,
04   vyrid    INTEGER      NOT NULL,
05   PRIMARY KEY (katid, vyrid),
06   CONSTRAINT fk_obsahujev_katid
07         FOREIGN KEY (katid) REFERENCES Kategorie (katid)
08         ON DELETE CASCADE,
09   CONSTRAINT fk_obsahujev_vyrid
10         FOREIGN KEY (vyrid) REFERENCES Vyrobek (vyrid)
11         ON DELETE CASCADE
12 );
```

Tabulka pro relaci "Objednavka".

```
01 CREATE TABLE Objednavka (
02   objid    INTEGER      NOT NULL,
03   kod      CHAR(11)     UNIQUE,
04   jmeno    VARCHAR(64)  NOT NULL,
05   email    VARCHAR(64)  NOT NULL,
06   datobj   DATE,
07   vyrizeno SMALLINT     CHECK (vyrizeno IN (0,1)) DEFAULT 0,
08   mesto    VARCHAR(64)  NOT NULL,
09   ulice    VARCHAR(64)  NOT NULL,
10   tel      VARCHAR(64),
11   PRIMARY KEY (objid)
12 );
```

Tabulka pro relaci "Polozka".

```
01 CREATE TABLE Polozka (
02   polid    INTEGER      NOT NULL,
03   katid    INTEGER      NOT NULL DEFAULT 0,
04   vyrid    INTEGER      NOT NULL DEFAULT 0,
05   objid    INTEGER      NOT NULL,
06   cena     NUMBER(9,2),
07   ks       SMALLINT     NOT NULL,
08   PRIMARY KEY (polid),
09   UNIQUE (vyrid, objid),
10   CONSTRAINT fk_polozka_katid
11         FOREIGN KEY (katid) REFERENCES Kategorie (katid)
12         ON DELETE SET DEFAULT,
13   CONSTRAINT fk_polozka_vyrid
14         FOREIGN KEY (vyrid) REFERENCES Vyrobek (vyrid)
15         ON DELETE SET DEFAULT
16   CONSTRAINT fk_polozka_objid
17         FOREIGN KEY (objid) REFERENCES Objednavka (objid)
18         ON DELETE CASCADE
19 );
```

Příklad 5:
Vyjádřete dotazy v přirozeném jazyce pomocí SQL.

Řešení příkladu 5:
Přepis dotazů typu "SELECT" v přirozeném jazyce:

1. *Název a popis dané kategorie.*
2. *Seznam názvů podkategorií dané kategorie seřazené v pořadí daným administrátorem.*

   ```
   1 SELECT   nazev
   2 FROM     Kategorie
   3 WHERE    katid = ?
   4 ORDER BY pre;
   ```

   kde za otazník doplníme katid požadované kategorie.
3. *Seznam názvů a cen výrobků v dané kategorii.*
4. *Adresa pro doručení, jméno zákazníka, email a telefon dané objednávky.*
5. *Seznam položek dané objednávky.*
6. *Suma cen z dané objednávky.*
7. *Seznam objednávek podle datumu vytvoření a příznaku uzavřeno/neuzavřeno.*

8. *Pro danou kategorii seznam výrobků které byly objednány z této kategorie a nebo nějaké její podkategorie včetně počtu takových objednávek pro každý z výrobků.*

```
1 SELECT   V.vyrid, COUNT(P.polid) AS pocetobj
2 FROM     ((Kategorie AS NK
3             JOIN Kategorie AS K ON
4                   (NK.pre <= K.pre AND NK.post >= K.post))
5               JOIN Polozka AS P ON (K.katid = P.katid))
6                 JOIN Vyrobek AS V ON (V.vyrid = P.vyrid)
7 WHERE    K.katid = ?
8 GROUP BY V.vyrid
```

9. *Pro daný výrobek a zadané období seznam kategorií, ze kterých byl výrobek objednán a počet objednávek tohoto výrobku z této kategorie za toto období.*

```
1 SELECT   K.katid, COUNT(P.polid) AS pocetobj
2 FROM     (Kategorie AS K JOIN Polozka AS P
3             ON (K.katid = P.katid))
4               JOIN Objednavka AS O ON (P.objid = O.objid)
5 WHERE    P.vyrid = ? AND
6          CAST(O.datobj AS VARCHAR(32)) BETWEEN ? AND ?
7 GROUP BY K.katid
```

kde za první otazník doplníme vyrid požadovaného výrobku a za druhý a třetí doplníme požadované období (tj. datum od a datum do).

10. *Celková suma objednávek pro dané období podle města z adresy pro doručení.*

```
1 SELECT   mesto, SUM(P.cena*P.ks) AS celkovacena
2 FROM     Objednavka O JOIN Polozka P ON (O.objid = P.objid)
3 WHERE    CAST(O.datobj AS VARCHAR(32)) BETWEEN ? AND ?
4 GROUP BY mesto
```

kde za první a druhý otazník doplníme požadované období (tj. datum od a datum do).

11. *Pro dané období seznam výrobků, které nebyly v daném období ani jednou objednány.*

```
1 SELECT   V.*
2 FROM     (Polozka AS P JOIN Objednavka AS O ON (P.objid = O.objid))
3            RIGHT OUTER JOIN Vyrobek V ON (P.vyrid = V.vyrid)
4 WHERE    P.vyrid IS NULL AND
5          CAST(O.datobj AS VARCHAR(32)) BETWEEN ? AND ?
7 GROUP BY mesto
```

kde za první a druhý otazník doplníme požadované období (tj. datum od a datum do)

12. *Seznam výrobků, které nejsou v žádné kategorii.*

```
1 SELECT   V.*
2 FROM     Vyrobek V
4 WHERE    NOT EXISTS (
5            SELECT * FROM ObsahujeV OV
6            WHERE OV.vyrid = V.vyrid
7          )
```

Dále také budeme potřebovat aktualizační dotazy, např.

- *Vytvoř kategorii s názvem 'Prsteny' a prázdným popisem v kategorii s katid 123 na konci seznamu podkategorií. Pro to ale nejdříve musíme zjistit ... pomocí*

```
1 SELECT pre AS npre, post AS npost
2 FROM   Kategorie
2 WHERE  katid = 123
```

```
1 SELECT MAX(pre) AS mpre
2 FROM   Kategorie
2 WHERE  pre > npre AND post < npost
```

kde npre a npost je preorder a postorder pořadí kategorie s katid 123 a mpre je preorder pořadí poslední kategorie v podstromu s vrcholem 123. Nová kategorie bude mít pořadí mpre+1 a npost. Před samotným vložením ale musíme udělat místo pro novou kategorii, tj. posunout vhodně pořadí dotčených uzlů.

```
1 UPDATE Kategorie
2 SET    pre = pre + 1
2 WHERE  pre >= ?
```

kde za první otazník dosadíme hodnotu mpre+1, čímž si uděláme místo pro preorder pořadí nové kategorie.

```
1 UPDATE Kategorie
2 SET    post = post + 1
2 WHERE  pre >= ?
```

kde za první otazník dosadíme hodnotu npost, čímž si uděláme místo pro postorder pořadí nové kategorie. Teď už můžeme vložit i novou podkategorii

```
1 INSERT INTO Kategorie
2  VALUES (NULL, 'Prsteny', '', 123, ?, ?)
```

kde za první otazník dosadíme hodnotu mpre+1 a za druhý otazník dosadíme hodnotu npost.

- *Vlož výrobek s vyrid 4353 do kategorie s katid 123.*

```
1 INSERT INTO ObsahujeV
2  VALUES (123, 4353)
```

- *Odstraň výrobky, které si už nikdo neobjednal od zadaného datumu.*

```
1 DELETE FROM Vyrobek V
2 WHERE vyrid NOT IN (
3   SELECT vyrid
4   FROM   Polozka P JOIN Objednavka O ON (P.objid = O.objid)
5   WHERE  CAST(O.datobj AS VARCHAR(32)) >= ?
6 )
```

kde za otazník dosadíme zadaný datum.

1. Vysvetlete pojem transakce.

Jedna se o predpis sekvence akci, ktere maji byt vykonany ve stanovenem poradi. Muze se jednat o databazove operace (cteni/zapis) i o nedatabazove operace (napr. aritmeticke vypocty).

Formalne lze transakci popsat jako sekvenci operaci read/write ukoncenou operaci commit/abort. Tato sekvence se nazyva rozvrh.

2. Na nasledujici transakci vysvetlete pozadovane vlastnosti transakcniho zpracovani (ACID).

Algorithm 1 $PriobjednejVyrobek1(pKosikID, pVyrobekID, pKusu)$

1: SELECT $kusu$ FROM $Sklad$ WHERE $vyrobekID = pVyrobekID$;
2: **if** $kusu < pKusu$ **then**
3: ZrusTransakci("Zbozi neni na sklade!")
4: **end if**
5: UPDATE $PolozkaKosiku$ SET $kusu+ = pKusu$ WHERE $kosikID = pKosikID$ AND $vyrobekID = pVyrobekID$;
6: UPDATE $Sklad$ SET $kusu- = pKusu$ WHERE $vyrobekID = pVyrobekID$;

A (= Atomicity) Znamena, ze transakce je provedena bud cela nebo vubec. Pocet kusu daneho vyrobku je tedy pripocten k zakaznikove kosiku prave tehdy, kdyz je take odecten stejny pocet kusu ze skladu.

C (= Consistency) Znamena, ze transakce musi zachovat databazi v konzistentnim stavu (predevsim spleni integritnich omezeni pred a po spusteni transakce). U jednotlive transakce to musi zajistit jeji programator. Napr. zajistujeme, ze priobjednavka neni provedena, pokud nemame dostatek zbozi na sklade.

I (= Isolation) Znamena, ze transakce je izolovana od ostatnich soucasne bezicich transakci. Transakce "ma pocit", ze ma databazovy stroj "jen pro sebe". Napr. musi byt zajisteno, ze pokud zjistim, ze mam dostatek zbozi na sklade, pak mohu provest bezpecne upravu kosiku a skladu. Tj. nestane se, ze po zjisteni dostatecneho mnozstvi mi jina transakce zbozi "nevyfoukne", cimz by se pocet kusu na sklade mohl dostat do zapornych hodnot.

D (= Durability) Znamena, ze po uspenem skonceni transakce jsou jeji vysledky trvale ulozeny. To neni samozrejme. Typycky jsou data ulozena v bufferech a je potreba je zapsat na disk, coz se z duvodu efektivity nedeje ihned po vykonani operace. Tj. po skonceni priobjednavky jsou aktualizovane informace o polozce kosiku a stavu zbozi na sklade trvale ulozeny.

V dalsim textu uz budeme uvazovat jenom formalizovane transakce, tj. rozvrhy operaci read/write/abort/commit.

Pro otazky 3. - 11. uvazujte nasledujici transakce: Uvazujte nasledujici tri transakce T_1, T_2 a T_3.

$T_1 = (read_1(vyrobek) \quad write_1(kosik_1) \quad write_1(vyrobek) \quad commit_1)$
$T_2 = (read_2(vyrobek) \quad write_2(vyrobek) \quad write_2(kosik_2) \quad commit_2)$
$T_3 = (read_3(kosik_1) \quad read_3(kosik_2) \quad write_3(kosik_1) \quad write_3(kosik_2) \quad commit_3)$

3. Navrhnete nejaky seriovy rozvrh transakci a popiste jeho vysledek.

4. Navrhnete nejaky usporadatelny rozvrh transakci, ktery neni seriovy.

Def. z prednasky: Rozvrh je usporadatelny (presneji receno serializovatelny, z angl. serializable), pokud jeho vykonani vede ke konzistentnimu stavu DB, tj. k temuz stavu databaze, ktery obdrzime libovolnym seriovym rozvrhem na stejnych transakcich.

T_1	T_2	T_3
	$read_2(vyrobek)$	
		$read_3(kosik_1)$
	$write_2(vyrobek)$	
$read_1(vyrobek)$		
		$read_3(kosik_2)$
		$write_3(kosik_1)$
$write_1(kosik_1)$		
$write_1(vyrobek)$		
		$write_3(kosik_2)$
	$write_2(kosik_2)$	

5. Uvazujte transakci $T_4 = (write_4(kosik_2) \quad commit_4)$ a rozvrh, ktery vznikne tak, ze k rozvrhu z reseni predchoziho prikladu pridame operace T_4 na konec. Porusi potom prehozeni operaci $write_3(kosik_2)$ a $write_2(kosik_2)$ podminku usporadatelnosti rozvrhu?

V takovem pripade se podminka neporusi, protoze nezavisle na tom, co se provede, je nakonec zapsana do $kosik_2$ nezavisla hodnota transakci T_4.

Priklady 6. - 8. se tykaji tzv. konfliktu operaci read a write. Sami o sobe tyto konflikty vadit nemusi. Jsou to ale mista potencialnich problemu pro splneni podminky usporadatelnosti rozvrhu.

6. Vysvetlete problem konfliktu W-R (dirty read). Objevuje se tento konflikt v reseni predchoziho prikladu?

Problem tohoto konfliktu je, ze cteme data, ktera jina transakce zmenila, ale jeste nepotvrdila. Mohou nastat dva problemy. Bud tato transakce data bude jeste dale menit a my jsme je tedy nemeli cist a nebo je jiz menit nebude, ale dojde k jejimu abortu a nase transakce se tedy stava neplatnou.

V reseni predchoziho prikladu se konflikt vyskytuje, napr.: $write_2(vyrobek) \rightarrow read_1(vyrobek)$.

7. Vysvetlete problem konfliktu R-W (unrepeatable read). Objevuje se tento konflikt v reseni predchoziho prikladu?

Problem tohoto konfliktu je, ze prepisujeme objekt, ktery jina transakce jiz precetla. Pokud tato transakce bude chtit tento objekt cist znovu, precte jinou hodnotu.

V reseni predchoziho prikladu se konflikt vyskytuje, napr.: $read_3(kosik_1) \rightarrow write_1(kosik_1)$.

8. Vysvetlete problem konfliktu W-W. Objevuje se tento konflikt v reseni predchoziho prikladu?

Problem tohoto konfliktu je, ze prepisujeme objekt, ktery jina transakce jiz take prepsala, ale jeste neskoncila. Tim dochazi ke ztrate hodnoty zapsane touto transakci.

V reseni predchoziho prikladu se konflikt vyskytuje, napr.: $write_3(kosik_1) \rightarrow write_1(kosik_1)$.

9. Vysvetlete pojem konfliktove usporadatelnosti. Je kazdy konfliktove usporadatelny rozvrh usporadatelny? Je kazdy usporadatelny rozvrh konfliktove usporadatelny?

Rozvrh je konfliktove usporadatelny, pokud je konfliktove ekvivalentni nejakemu seriovemu rozvrhu (na stejne mnozine transakci). Dva rozvrhy jsou konfliktove ekvivalentni, kdyz vsechny "konfliktni" pary operaci v jednom rozvrhu existuji (stejne konfliktni) i ve druhem rozvrhu.

V otazce 5. je navrzen rozvrh, ktery je usporadatelny, ale neni konfliktove usporadatelny.

10. Uvazujte nasledujici rozvrh. Overte pomoci precedencniho grafu jeho konfliktovou usporadatelnost. Popiste pripadne nalezene problemy.

T_1	T_2	T_3
	$read_2(vyrobek)$	
		$read_3(kosik_1)$
	$write_2(vyrobek)$	
$read_1(vyrobek)$		
$write_1(kosik_1)$		
$write_1(vyrobek)$		
		$read_3(kosik_2)$
		$write_3(kosik_1)$
		$write_3(kosik_2)$
	$write_2(kosik_2)$	

T_3 cte objekt $kosik_1$ a nez do nej zapise, zapise do nej transakce T_1. To ale neodpovida zadnemu seriovemu rozvrhu.

11. Modifikujte rozvrh z reseni prikladu 4. pridanim nove transakce tak, aby konflikt W/R, resp. R/W, resp. W/W porusil podminku na konfliktovou usporadatelnost.

12. Je nasledujici rozvrh zotavitelny? Zduvodnete. Upravte ho tak, aby byl zotavitelny. Nejprve uvazujte situaci, kdy nam nevadi kaskadovy abort. Potom uvazujte situaci, kdy nam vadi i kaskadovy abort. Pozorujte, jaky ma zprisnovani podminek vliv na paralelizaci zpracovani.

T_1	T_2	T_3
	$read_2(vyrobek)$	
		$read_3(kosik_1)$
	$write_2(vyrobek)$	
$read_1(vyrobek)$		
		$read_3(kosik_2)$
		$write_3(kosik_1)$
$write_1(kosik_1)$		
$write_1(vyrobek)$		
		$write_3(kosik_2)$
		$commit_3$
$commit_1$		
	$write_2(kosik_2)$	
	$commit_2$	

Rozvrh je zotavitelny, pokud kdykoliv cte transakce T_i hodnoty zapsane transakci T_j a T_i commituje, musi T_j commitovat drive. Kdykoliv tedy T_i cte objekt x, do ktereho pred tim zapsala T_j, musi T_i cekat, az T_j commituje a potom teprve muze commitovat i T_i. Nam vadi, ze transakce T_1 cte objekt *vyrobek* zapsany transakci T_2 a pritom T_1 commituje, aniz by commitovala T_2. Pokud by T_2 abortovala po commitu T_1, hodnota objektu *vyrobek* prectena transakci T_1 by byla neplatna a T_1 by musela byt abortovana take. Ta vsak jiz commitovala. Zotavitelna varianta (muze nastat kaskadovy abort):

T_1	T_2	T_3
	$read_2(vyrobek)$	
		$read_3(kosik_1)$
	$write_2(vyrobek)$	
$read_1(vyrobek)$		
		$read_3(kosik_2)$
		$write_3(kosik_1)$
$write_1(kosik_1)$		
$write_1(vyrobek)$		
		$write_3(kosik_2)$
		$commit_3$
	$write_2(kosik_2)$	
	$commit_2$	
$commit_1$		

Pokud nechceme kaskadove aborty, musime cist pouze commitovana data:

T_1	T_2	T_3
	$read_2(vyrobek)$	
		$read_3(kosik_1)$
	$write_2(vyrobek)$	
	$write_2(kosik_2)$	
	$commit_2$	
$read_1(vyrobek)$		
		$read_3(kosik_2)$
		$write_3(kosik_1)$
$write_1(kosik_1)$		
$write_1(vyrobek)$		
		$write_3(kosik_2)$
		$commit_3$
$commit_1$		

12. Vysvetlete pojem dvoufazoveho uzamykaciho protokolu (2PL). Popiste, jak pracuje 2PL planovac. Jak musi pracovat, aby obdrzel zotavitelne rozvrhy a predchazel kaskadovym abortum?

2PL uplatnuje tri pravidla pro sestaveni rozvrhu:

a) transakce musi spravne zamykat objekty, ke kterym pristupuje (dobre formovana transakce)

b) transakce nemuze zamykat objekty potom, co jiz nejaky objekt odemkla (dvoufazova transakce, 1. faze - zamykani, 2. faze - odemykani)

c) v rozvrhu nedochazi ke konfliktum na zamcich mezi transakcemi (legalni rozvrh)

Zamky umistuje do rozvrhu primo 2PL planovac. 2PL planovac pracuje tak, ze kdykoliv obdrzi operaci r/w(e), musi ziskat zamek s/x(e). Pokud ho nelze ziskat, transakce musi cekat, dokud nedojde k uvolneni blokujiciho zamku. K uvolneni zamku muze dojit az ve chvili, kdy je potvrzeno uspesne provedeni operace, kvuli ktere byl zamek nastaven. Aby byl dodrzen dvoufazovy protokol, nesmi uz pro danou transakci vystavovat zamky, pokud byl nejaky zamek odemcen. Tim nastava problem, kdy tedy odemykat. Bez blizsich informaci je nejlepsi pouze zamykat a odemykani provest az pri commit/abort transakce. Tak pracuji tzv. striktni 2PL planovace. Tim take ziskavame rozvrhy, ktere jsou zotavitelne a predchazeji kaskadovym abortum (cteme pouze potvrzena, tj. commitovana data).

13. Je nasledujici rozvrh legalni? Jsou transakce dvoufazove? Je rozvrh konfliktove usporadatelny?

T_1	T_2	T_3
$X(vyrobek_1)$		
$read_1(vyrobek_1)$		
		$S(vyrobek_2)$
		$read_3(vyrobek_2)$
$write_1(vyrobek_1)$		
$U(vyrobek_1)$		
		$U(vyrobek_2)$
	$S(vyrobek_1)$	
		$S(vyrobek_1)$
		$read_3(vyrobek_1)$
		$U(vyrobek_1)$
		$commit_3$
$X(vyrobek_2)$		
	$read_2(vyrobek_1)$	
	$U(vyrobek_1)$	
	$commit_2$	
$read_1(vyrobek_2)$		
$write_1(vyrobek_2)$		
$U(vyrobek_2)$		
$commit_1$		

Rozvrh je legalni, protoze zadna z dvojic zamku neni v konfliktu. T_1 a T_3 nejsou dvoufazove. Rozvrh neni konfliktove usporadatelny, protoze jeho precedencni graf obsahuje cyklus zpusobeny nasledujicimi konfliktnimi dvojicemi operaci: $read_3(vyrobek_2) \rightarrow write_1(vyrobek_2)$ a $write_1(vyrobek_1) \rightarrow read_3(vyrobek_1)$.

14. Upravte transakce tak, aby byly dvoufazove a navrhnete rozvrh. Je konfliktove usporadatelny?

T_1	T_2	T_3
		$S(vyrobek_2)$
		$read_3(vyrobek_2)$
		$S(vyrobek_1)$
		$U(vyrobek_2)$
		$read_3(vyrobek_1)$
		$U(vyrobek_1)$
		$commit_3$
$X(vyrobek_1)$		
$read_1(vyrobek_1)$		
$write_1(vyrobek_1)$		
$X(vyrobek_2)$		
$U(vyrobek_1)$		
	$S(vyrobek_1)$	
	$read_2(vyrobek_1)$	
	$U(vyrobek_1)$	
	$commit_2$	
$read_1(vyrobek_2)$		
$write_1(vyrobek_2)$		
$U(vyrobek_2)$		
$commit_1$		

Tento rozvrh jiz je konfliktove usporadatelny. Pokud by T_3 cetla $vyrobek_1$ a $vyrobek_2$ v obracenem poradi, bylo by mozne zlepsit konkurenci zpracovani:

T_1	T_2	T_3
		$S(vyrobek_1)$
		$read_3(vyrobek_1)$
		$S(vyrobek_2)$
		$U(vyrobek_1)$
$X(vyrobek_1)$		
$read_1(vyrobek_1)$		
$write_1(vyrobek_1)$		
		$read_3(vyrobek_2)$
		$U(vyrobek_2)$
		$commit_3$
$X(vyrobek_2)$		
$U(vyrobek_1)$		
	$S(vyrobek_1)$	
	$read_2(vyrobek_1)$	
	$U(vyrobek_1)$	
	$commit_2$	
$read_1(vyrobek_2)$		
$write_1(vyrobek_2)$		
$U(vyrobek_2)$		
$commit_1$		

15. Navrhnete rozvrh, ktery by mohl byt vysledkem striktniho 2PL planovace pro transakce v predchozim rozvrhu.(U operace neni nutne psat i kdyz ja je v reseni mam, aby bylo jasno co se deje. Striktni 2PL planovac uvolni vsechny zamky pri commitu automaticky)

T_1	T_2	T_3
		$S(vyrobek_1)$
		$read_3(vyrobek_1)$
	$S(vyrobek_1)$	
	$read_2(vyrobek_1)$	
	$U(vyrobek_1)$	
	$commit_2$	
		$S(vyrobek_2)$
		$read_3(vyrobek_2)$
		$U(vyrobek_1)$
		$U(vyrobek_2)$
		$commit_3$
$X(vyrobek_1)$		
$read_1(vyrobek_1)$		
$write_1(vyrobek_1)$		
$X(vyrobek_2)$		
$read_1(vyrobek_2)$		
$write_1(vyrobek_2)$		
$U(vyrobek_1)$		
$U(vyrobek_2)$		
$commit_1$		

16. Je v nasledujicim rozvrhu nejaky problem? Pokud ano, popiste ho. Jakym zpusobem ho muzem planovac resit?

T_1	T_2	T_3
$X(vyrobek_1)$		
$read_1(vyrobek_1)$		
	$X(vyrobek_2)$	
$write_1(vyrobek_1)$		
	$write_2(vyrobek_2)$	
		$S(vyrobek_3)$
$S(vyrobek_2)$		
		$read_3(vyrobek_3)$
	$X(vyrobek_3)$	
		$S(vyrobek_1)$
...	...	...

Planovac zablokuje T_1, ktera musi cekat na uvolneni exkluzivniho zamku na $vyrobek_2$. Pote zablokuje T_2, ktera musi cekat na uvolneni sdileneho zamku na $vyrobek_3$. Nakonec zablokuje T_3, ktera musi cekat na uvolneni exklusivniho zamku na $vyrobek_1$. Vsechny transakce jsou tedy zablokovany, zadna nemuze potrebny zamek uvolnit a dochazi k deadlocku.

Prvni moznosti reseni deadlocku je jejich detekce. Muzeme sledovat dobu, po kterou transakce ceka a pokud ceka moc dlouho, usoudime, ze je v deadlocku a abortujeme ji. Preciznim zpusobem detekce deadlocku je hledani cyklu ve wait-for grafu (uzly jsou transakce a hrana vede z T_A do T_B, pokud T_A ceka na nejaky zamek prideleny transakci T_B). Pokud cyklus najdeme, vybereme nejakou transakci z cyklu a tu zrusime (nejcasteji tu s nejkratsim logem - udelala zatim nejmene prace, abort je nejjednodussi). V priklade bychom napr. abortovali T_3.

Druhou moznosti je prevence deadlocku, tj. udelovani zamku tak, aby cyklus ve wait-for grafu vubec nevznikal. Kazde transakci priradime prioritu danou casovym razitkem jejiho vzniku. Cim starsi transakce tim vyssi priorita. Predpokladejme, ze T_A pozaduje zamek, ktery jiz drzi T_B. Muzeme pouzit jedno z nasledujicich reseni:

- **wait-die** pokud ma T_A vyssi prioritu, ceka, pokud ne, je zrusena
- **wound-wait** pokud ma T_A vyssi prioritu, je zrusena T_2, jinak T_A ceka

V priklade by reseni wait-die znamenalo, ze T_1 ma vyssi prioritu nez T_2 a muze cekat, T_2 ma vyssi prioritu nez T_3 a muze cekat, ale T_3 ma nizsi prioritu nez T_1 a proto je zrusena a spustena znovu. Je dulezite ji ale priradit puvodni prioritu, jinak by mohla byt opakovane rusena (tzv. livelock). Pokud ma stale puvodni prioritu, nebude dochazet k dlouhemu opakovanemu ruseni, protoze

postupne "starne" oproti ostatnim.

Reseni wound-wait by znamenalo, ze T_1 ma vyssi prioritu nez T_2 a proto je T_2 zrusena a spustena znovu.

17. Predpokladejme, ze vyrobky maji atribut *cena* a *kategorie*. Uvazujme konkretni instanci tabulku vyrobku: $\{vyrobek_1(20, k_1), vyrobek_2(40, k_1), vyrobek_3(10, k_2), vyrobek_4(70, k_2)\}$. Dale uvazujme transakci T_1, ktera pro kazdou kategorii spocita soucet cen vyrobku v teto kategorii a T_2, ktera do kategorie k_1 vklada novy vyrobek a z kategorie k_2 odstranuje vyrobek. Je v nasledujicim rozvrhu nejaky problem?

T_1	T_2	T_3
$S(vyrobek_1)$		
$S(vyrobek_2)$		
$sum_1 =$		
$sum(vyrobek_1.cena,$		
$vyrobek_2.cena)$		
	$INSERT(vyrobek_5(40, k_1))$	
	$X(vyrobek_3)$	
	$DELETE(vyrobek_3)$	
	$commit_2$	
$S(vyrobek_4)$		
$sum_2 =$		
$sum(vyrobek_4.cena)$		
$unlock, commit_1$		

Vysledek rozvrhu je $sum_1 = 60$ a $sum_2 = 70$. V pripade serioveho rozvrhu T_1, T_2 je vsak vysledek $sum_1 = 60$ a $sum_2 = 80$ a v pripade serioveho rozvrhu T_2, T_1 je vysledek $sum_1 = 100$ a $sum_2 = 70$. Rozvrh tedy neodpovida zadnemu seriovemu rozvrhu. Problem je, ze transakci T_1 transakce T_2 pridava/odstranuje radky tabulky, nad kterou pracuje. Teto transakci se rika fantom.

T_1 vzdy uzamyka cast tabulky vyrobku, ktera odpovida dane logicke podmince. Tato cast tabulky by nemela byt zmenena, dokud ji opet neodemkne. Klasicke zamky na to vsak nestaci, protoze uzamykaji uz jen konkretni zaznamy bez explicitni znalosti teto logicke podminky. Pokud existuje index nad vlastnosti kategorie (B+-strom), muzeme provest uzamceni vyrobku s dane kategorie uzamcenim casti indexu. Pokud potom chce druha transakce pridavat/mazat z tabulky vyrobku, postupuje od korene indexu a hleda, zda je potrebna cast tabulky uzamcena nebo volna. V priklade by napr. T_1 uzamkla cast indexu pro atribut kategorie odpovidajici hodnote k_1 a T_2 by pred vkladanim musela projit index od korene a zjistit, zda k_1 je uzamcena. V tomto pripade by zjistila, ze ano a proto by musela na zamek cekat (prip. by byla zrusena, zalezi jak resime deadlocky). V tomto pripade by pravdepodobne potrebny index existoval, protoze atribut kategorie je cizi klic. Obecne ale index existovat nemusi. Potom bychom museli uzamknout celou tabulku vyrobku. Komplikovanejsi situaci je vztah M:N mezi kategoriemi a vyrobky.

17. Jake jsou urovne izolace transakci v SQL?

a) **READ UNCOMMITED**: muze nastat spinave cteni, neopakovatelne cteni i fantom

b) **READ COMMITED**: (striktni 2PL na X(), 2PL na S()) nemuze nastat spinave cteni, muze nastat neopakovatelne cteni i fantom

c) **REPEATABLE READ**: (striktni 2PL na X() i S()) nemuze nastat spinave cteni ani neopakovatelne cteni, muze se objevit fantom

d) **SERIALIZIBLE**: (striktni 2PL na X() i S() + prevence fantoma) nemuze nastat spinave cteni, neopakovatelne cteni ani fantom

Pozn.: Zajimave cteni o urovnich izolace v Oracle najdete u Toma:
http://www.oracle.com/technology/oramag/oracle/05-nov/o65asktom.html

18.Uvazujte transakce $T_1, \ldots, T_5$ na nasledujicim obrazku. Uvazujte cas posledniho kontrolniho bodu (checkpointu) t_c a cas havarie t_f. Popiste jak muze probihat zotaveni systemu.

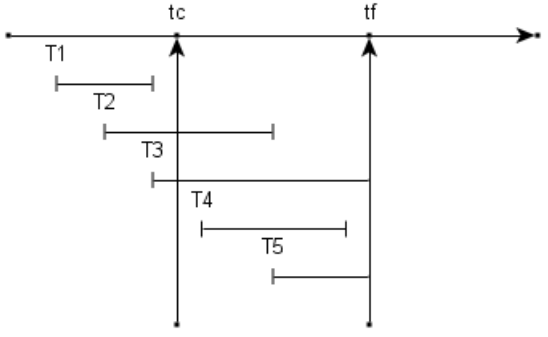


Figure 1: Zotaveni transakci

Logujeme kazdou zmenu databaze provedenou transakcemi. Typicky nejsou zaznamy logu primo zapisovany na disk, ale do bufferu. Pri jeho zaplneni je buffer odeslan na disk. Samotne transakce take neprovadeji zapisy primo na disk, ale do cache. Databazovy system provadi tzv. checkpointy, tj. odeslani obsahu cache na disk. Odpovidajici zaznamy v logu museji byt odeslany na disk pred samotnym odeslanim obsahu cache. Nakonec je do logu (primo na disk) zapsan zaznam o provedeni checkpoint vcetne seznamu aktivnich transakci (T_2 a T_3. Transakce muze prejit do stavu *commited* jen pokud je nejprve na disk zapsan log zaznam o jejim commitu. Popsanemu principu se rika *write-ahead logging* nebo emphWAL.

Po padu systemu probehne zotaveni nasledujicim zpusobem:

a) inicializuje prazdne $undo - list$ a $redo - list$

b) jde od konce logu a hleda prvni zaznam $checkpoint\, L$, kde L je seznam aktivnich transakci pri danem checkpointu

- pokud najde pri pruchodu zaznam $commit\, T_i$, da T_i do $redo - list$
- pokud najde pri pruchodu zaznam $start\, T_i$ a T_i neni v $redo - list$, da T_i do $undo - list$

c) kazdou $T_i \in L$, ktera neni v $redo - list$ da do $undo - list$ (v tomto okamziku je $undo - list$ seznam nepotvrzenych transakci, ktere museji byt rollbackovany a $redo - list$ je seznam potvrzenych transakci, ktere museji byt provedeny znovu)

d) kazda T_i v $undo - list$ je rollbackovana

e) kazda T_i v $redo - list$ je provedena znovu

$$Zam(c\_zam, j\_zam, plat, vek)$$
$$Odd(j\_odd, rozpocet, c\_ved)$$
$$Pracuje(c\_zam, j\_odd, uvazek)$$

V nekterych dotazech vyjadrenych v relacni algebre budu misto projekce a nasledneho prejmenovani $R[A] < A \to B >$ pouzivat zkratku $R[A \to B]$.

1. Jmena oddeleni, ve kterych pracuji vsichni zamestnanci mladsi nez 30 let.

$$\begin{aligned} R_1 &= Pracuje[j\_odd, c\_zam] \div Zam(vek <= 30)[c\_zam] \\ &= \quad (Pracuje[j\_odd] \setminus (\\ &\qquad ((Pracuje[j\_odd] \times Zam(vek <= 30)[c\_zam]) \\ &\qquad \setminus Pracuje[j\_odd, c\_zam])[j\_odd]) \end{aligned}$$

$$\begin{aligned} DRK : \{&j\_odd : Odd(j\_odd) \wedge \forall c\_zam \, \forall vek(\\ &Zam(c\_zam, vek) \wedge vek <= 30 \\ &\Rightarrow \\ &Pracuje(c\_zam, j\_odd) \\ &)\} \end{aligned}$$

$$\begin{aligned} NRK : \{&o.j\_odd : Odd(o) \wedge \forall Zam(z)(\\ &Z.vek <= 30 \\ &\Rightarrow \\ &\exists Pracuje(p)(p.j\_odd = o.j\_odd \wedge p.c\_zam = z.c\_zam) \\ &)\} \end{aligned}$$

```
01 SELECT j_odd
02 FROM   Odd
03 WHERE  NOT EXISTS (
04           SELECT *
05           FROM   Zam
06           WHERE  vek<=30 AND NOT EXISTS (
07             SELECT *
08             FROM   Pracuje
09             WHERE  Pracuje.c_zam = Zam.c_zam AND
10                    Pracuje.j_odd = Odd.j_odd
11           )
12         );
```

```
01 SELECT j_odd
```

```
02 FROM   Odd
03 WHERE  (SELECT COUNT(DISTINCT c_zam)
04         FROM   Zam NATURAL JOIN Pracuje
05         WHERE  Pracuje.j_odd = Odd.j_odd AND
06                Zam.vek<=30)
07        =
08        (SELECT COUNT(c_zam)
09         FROM   Zam
10         WHERE  Zam.vek<=30);
```

2. Jmena oddeleni, ve kterych pracuji pouze zamestnanci mladsi nez 30 let.

$$R_2 = Odd[j\_odd] \setminus (Pracuje * Zam(vek > 30)[c\_zam])[j\_odd]$$

$$\begin{aligned}
DRK : \{&j_odd : Odd(j_odd) \wedge \forall c_zam(\\
&Pracuje(c_zam, j_odd) \\
&\Rightarrow \\
&\exists vek(Zam(c_zam, vek) \wedge vek <= 30) \\
&)\}
\end{aligned}$$

$$\begin{aligned}
NRK : \{&o.j_odd : Odd(o) \wedge \forall Zam(z)(\\
&\exists Pracuje(p)(p.j_odd = o.j_odd \wedge p.c_zam = z.c_zam) \\
&\Rightarrow \\
&Z.vek <= 30 \\
&)\}
\end{aligned}$$

3. Jmena oddeleni, ve kterych maji vsichni zamestnanci plat mensi nez je plat vedouciho oddeleni.

$$\begin{aligned}
R_3^1 &= (Odd[c_ved = c_zam]Zam)[j_odd, plat, c_ved] < plat \rightarrow plat_ved > \\
R_3^2 &= (Pracuje * R_3^1)(c_ved <> c_zam)[c_zam, j_odd, plat_ved] \\
R_3^3 &= (Zam * R_3^2)(plat > plat_ved)[j_odd] \\
R_3 &= Odd[j_odd] \setminus R_3^3
\end{aligned}$$

$$
\begin{aligned}
DRK : \{ & j\_odd : \exists c\_ved, plat\_ved(\\
& Odd(j\_odd, c\_ved) \wedge Zam(c\_ved, plat\_ved) \wedge \forall c\_zam(\\
& Pracuje(c\_zam, j\_odd) \\
& \Rightarrow \\
& \exists plat(Zam(c\_zam, plat) \wedge plat < plat\_ved) \\
&))\}
\end{aligned}
$$

$$
\begin{aligned}
NRK : \{ & o.j\_odd : Odd(o) \wedge \exists Zam(v)(\\
& o.c\_ved = v.c\_zam \wedge \forall Zam(z)(\\
& \exists Pracuje(p)(p.j\_odd = o.j\_odd \wedge p.c\_zam = z.c\_zam) \\
& \Rightarrow \\
& z.plat < v.plat) \\
&))\}
\end{aligned}
$$

```
01 SELECT j_odd
02 FROM   Odd JOIN (Zam AS Ved) ON Odd.c_ved=Ved.c_zam
03 WHERE  Ved.plat > ALL(
04           SELECT plat
05           FROM   Zam NATURAL JOIN Pracuje
06           WHERE  Pracuje.j_odd = Odd.j_odd AND
07                  Zam.c_zam <> Ved.c_zam
08        );
```

4. Zamestnanci, kteri pracuji ve trech ruznych oddelenich (chceme cislo zamestnance a jmena techto 3 oddeleni).

$$
\begin{aligned}
R_4 = & Zam \\
& * Pracuje[c\_zam, j\_odd] < j\_odd \rightarrow j1 > \\
& * Pracuje[c\_zam, j\_odd] < j\_odd \rightarrow j2 > \\
& * Pracuje[c\_zam, j\_odd] < j\_odd \rightarrow j3 > \\
& (j1 < j2 \wedge j2 < j3) \\
& [c\_zam, j1, j2, j3]
\end{aligned}
$$

$$
\begin{aligned}
DRK : \{ & c\_zam, j1, j2, j3 : Zam(c\_zam) \wedge \\
& Pracuje(c\_zam, j1) \wedge Pracuje(c\_zam, j2) \wedge Pracuje(c\_zam, j2) \wedge \\
& j1 < j2 \wedge j2 < j3 \\
&)\}
\end{aligned}
$$

$$
\begin{aligned}
NRK : \{ & z.c\_zam, p1.j\_odd, p2.j\_odd, p3.j\_odd : Zam(z) \wedge \\
& Pracuje(p1) \wedge Pracuje(p2) \wedge Pracuje(p3) \wedge \\
& p1.j\_odd < p2.j\_odd \wedge p2.j\_odd < p3.j\_odd \\
&)\}
\end{aligned}
$$

5. Dvojice zamestnancu (staci jejich cisla), kteri pracuji ve stejnych oddelenich (nestaci, aby existovalo oddeleni, ve kterem oba pracuji, musi platit, ze pro danou dvojici z1,z2 ve vysledku pracuje z1 ve vsech oddelenich, ve kterych pracuje z2 a naopak).

$$
\begin{aligned}
R_5^1 &= Odd[c\_odd] \times Zam[c\_zam] \\
R_5^2 &= (R_5^1 \setminus Pracuje[c\_odd, c\_zam]) < c\_zam \rightarrow c\_zam2 > \\
R_5^3 &= Pracuje[c\_odd, c\_zam \rightarrow c\_zam1] \\
R_5^4 &= (R_5^3 * R_5^2)[c\_zam1, c\_zam2] \\
R_5^5 &= (Zam[c\_zam \rightarrow c\_zam1])[c\_zam1 < c\_zam2](Zam[c\_zam \rightarrow c\_zam2]) \\
R_5^6 &= R_5^5 \setminus (R_5^4 \cup R_5^4[c\_zam1 \rightarrow c\_zam2, c\_zam2 \rightarrow c\_zam1])
\end{aligned}
$$

Jadro reseni je ve vyrazu R_5^4, ktery da dvojice c_zam1, c_zam2 takove, ze existuje oddeleni, ve kterem pracuje c_zam1 a c_zam2 tam nepracuje.

$$
\begin{aligned}
DRK : \{ & c\_zam1, c\_zam2 : \\
& Zam(c\_zam1) \wedge Zam(c\_zam2) \wedge c\_zam1 < c\_zam2 \wedge \\
& \forall j\_odd(Odd(j\_odd) \Rightarrow \\
& (Pracuje(c\_zam1, j\_odd) \\
& \Leftrightarrow \\
& Pracuje(c\_zam2, j\_odd)) \\
&)\}
\end{aligned}
$$

$$
\begin{aligned}
NRK : \{ & z1.c\_zam, z2.c\_zam : \\
& Zam(z1) \wedge Zam(z2) \wedge z1.c\_zam < z2.c\_zam \wedge \\
& \forall Odd(o)(\\
& \exists Pracuje(p)(p.j\_odd = o.j\_odd \wedge p.c\_zam = z1.c\_zam) \\
& \Leftrightarrow \\
& \exists Pracuje(p)(p.j\_odd = o.j\_odd \wedge p.c\_zam = z2.c\_zam) \\
&)\}
\end{aligned}
$$

6. Zamestnanci se sumou uvazku, kteri nejsou ve spolecnosti zamestnani na plny uvazek.

Nasledujici reseni neumoznuje pridat sumu uvazku, protoze z vnoreneho dotazu nic nedostaneme.

```
01 SELECT *
02 FROM   Zam
04 WHERE  Zam.c_zam IN (SELECT   P.c_zam
05                      FROM     Pracuje AS P
06                      GROUP BY P.c_zam
07                      HAVING   SUM(P.uvazek)<100
```

Nasledujici reseni je spatne (vraci atributy, podle kterych nebylo GROUP BY).

```
01 SELECT    Zam.*, SUM(Pracuje.uvazek)
02 FROM      Zam NATURAL JOIN Pracuje
03 GROUP BY  Zam.c_zam
04 HAVING    SUM(Pracuje.uvazek)<100
```

Nasledujici reseni jsou O.K (Zam.c_zam je klic).

```
01 SELECT    Zam.*, SUM(Pracuje.uvazek)
02 FROM      Zam NATURAL JOIN Pracuje
03 GROUP BY  Zam.c_zam, Zam.j_zam, Zam.plat, Zam.vek
04 HAVING    SUM(Pracuje.uvazek)<100
```

```
01 SELECT Zam.*, ZamUvazek.cel_uvazek
02 FROM   Zam JOIN (SELECT    Pracuje.c_zam,
03                            SUM(Pracuje.uvazek)
04                              AS cel_uvazek
05                  FROM      Pracuje
06                  GROUP BY  Pracuje.c_zam
07                  HAVING    SUM(Pracuje.uvazek)<100)
08                              AS ZamUvazek
09                  ON (Zam.c_zam = ZamUvazek.c_zam)
```

7. Prumer minimalnich platu zamestnancu starsich 30 let v jednotlivych oddelenich. Nepocitame celkovy plat, ale plat zamestnance v danem oddeleni, tj. podle uvazku.

```
01 SELECT AVG(min_plat)
02 FROM   (SELECT   MIN(plat*uvazek/100) AS min_plat
03          FROM    Pracuje JOIN Zam USING(c_zam)
04          WHERE   vek>30
05          GROUP BY j_odd) AS T
```

8. Jmena zamestnancu, kteri jsou zamestnani v nejvetsim poctu oddeleni.

```
01 SELECT Zam.j_zam
02 FROM   Zam
03 WHERE  Zam.c_zam IN (SELECT   P1.c_zam
04                      FROM     Pracuje AS P1
05                      GROUP BY P1.c_zam
06                      HAVING   COUNT(*)>=ALL(
07                                 SELECT   COUNT(*)
08                                 FROM     Pracuje AS P2
09                                 GROUP BY P2.c_zam))
```