Rychlík

Zkouška je test, není to na ABCD, test má 8 – 10 otázek, plný počet je 20 bodů, nad 18 je jednička, min. je 10 bodů

# Úvod – postavení databází v návrhu informačního systému

1. Definice cílů (proč to chci dělat) – záleží na tom, zda je to komerční firma, či univerzita – proč to chce uživatel dělat, dobré je říct jak se pozná, že informační systém splnil vše, co se od něj očekávalo – pozn. Mělo by to být měřitelné
2. Specifikace požadavků (co chci dělat)
3. Návrh informačního systému (jak to udělat)
   1. Dělají se modely reality – vždy datový model, obvykle funkční model, a pak nějaké další (např. diagram datových toků nebo stavový model)
4. Realizace
   1. Zrealizovat databázi
   2. Programování aplikace
5. Testování
6. Předání do provozu
   1. Informační systém vždy někomu přidá práci, ale jako celek by to práci mělo zmenšit

Budeme se zabývat datovým modelem

# Martin Zima

Na příštím cvičení zadání semestrální práce

3. – 5. Praktická cvičení

6. cvičení – problematika datového modelování – E-R-A model

7. – 8. Cvičení – ověřování a prezentace zadání sam. Prac. – u tabule předvést své zadání – 5-10 minut

9. – 10. Cvičení – relační model dat, relační algebra

11. cvičení – funkční závislost, normální formy

12. – 13. Prostor pro odevzdávání a předvádění samostatných prací

Praktická cvičení

cv. 3, cv. 4 – MS Access (8.10., 15.10. vše v UL409)

cv. 5 – MySQL (phpMyAdmin – pozn. Nainstalovat podporu php pro iis) (22.10.)

podmínky pro zápočet (nutné podmínky):

1. Včas odevzdaný vyplněný formulář zadání semestrální práce
2. Ověření a prezentace zadání semestrální práce
3. Osobně předvedená a odevzdaná semestrální práce (opět předvedení tak 5-10 minut)
4. Dokumentace – v papírové vytištěné podobě

Žádné body ze cvičení – o známce rozhoduje až výsledek zkoušky

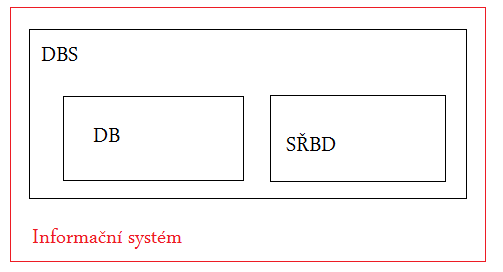
## Systémy řízení báze dat

SŘBD

### Souborově orientovaný přístup k datům

* Organizace dat: soubor, záznam, položka
* Položka je z pohledu databázového zpracování dále nedělitelná
* V žádném případě se neuvažovaly vazby mezi soubory
* Např. student a jeho známky – student má nějaké známky
* Data nebylo možné sdílet (více aplikací nemohlo sdílet stejná data)
* Duplicita dat
* Problém konzistence
* V každé aplikaci musel být naprogramován základní přístup k datům
* Každý tvůrce aplikace si musel algoritmy programovat sám

## Databázový přístup k datům

* Je založen na:
* 
  + Báze dat - je množina souborů (a popisu jejich dat), které jsou navzájem v logickém vztahu a které jsou spravovány systémem řízení báze dat
  + Systém řízení báze dat je programový systém, který v bázi dat zajišťuje
    - 1) definování dat (CREATE TABLE apod.)
    - 2) manipulaci s daty – aktualizaci, změnu, výběr, mazání
    - 3) ochranu dat
      * Ochrana dat proti zneužití (systém přidělování přístupových práv apod.)
      * Ochrana proti ztrátě dat – zajištění dat proti ztrátě – když odejde disk, data musí být ještě někde jinde, aby nedošlo ke ztrátě dat (transakční zpracování)
    - 4) komunikaci mezi systémem a uživatelem
  + Databáze
    - Vlastnosti:
      * neredundantnost – bez duplicity dat, databáze by měla jít navrhnout tak, aby tam nemusela být neopodstatněná duplicita – opodstatněná duplicita je jen v případě, že to s duplicitou poběží rychleji
      * Současný přístup více uživatelů – vícenásobné využití – a to jak uživateli, tak i aplikacemi – SŘBD (Oracle, Firebird, Access…)
      * Integrita dat – např. chceme smazat studenta co má výpůjčky, ono to nedovolí
      * Nezávislost dat na aplikaci
      * Databáze musí být schopna zrealizovat jakýkoli datový model
    - DDL - Očekáváme nástroj pro definování dat – Data Definition Language (jazyky 4GL – vezmu myš a naklikám to myší :D)
    - DML – jazyk pro manipulaci s daty – Data Manipulation Language
      * Např. SQL
      * V principu nabízejí dva typy těchto jazyků
        + Procedurální jazyk

Jakým způsobem ta data ze systému dostat

* + - * + Neprocedurální jazyk

Umožňují popsat, jaká data chci ze systému dostat

* + - * Hostovací jazyky – daly k dispozici sadu funkcí, pomocí kterých se dalo s daty manipulovat
    - DCL – Data Control Language
      * Část jazyka, která řídí přístup k datům (např. GRANT a REMOVE z SQL)
      * Např. Označení transakce, označení indexu…
      * Zajišťuje i vícenásobný přístup
      * Měl by umět obnovu systému po chybě
    - Okolí databázového systému – obecně jsou to uživatelé – nejvyšším uživatelem je administrátor systému – řeší konflikty mezi ostatními uživateli a systémem – ví, jak nastavit SŘBD, měl by se vyznat v parametrech…
      * Administrátor dat = **Gestor dat** = člověk, který umí vysvětlit význam dat v databázi a umí vysvětlit jejich souvislost
    - Uživatelé – programátoři, tvůrci těch aplikací
      * Analytici – měl by poznat reálný svět a popsat ho
      * Vývojáři – podle analýzy udělá vývoj té aplikace
      * Neprogramátoři

## Konceptuální modelování

ANSI – SPARC (1975)

Na data se pohlíží ve 3 úrovních

1. External level – pohledy uživatelů – jinej pohled má studijní referentka, jiný pohled má učitel…
2. Conceptual level – vše výše uvedené se smrskne do této úrovně – říká to, jaká data budu ukládat = logický pohled na data
3. Internal level – výše uvedené vyústí v tuto úroveň – úroveň, která říká, jak ta data budu ukládat

* Konceptuální modelování vychází z toho, že se na aplikaci nedíváme jako na informační systém, ale přesto musíme požadovat, aby byl v aplikaci soulad mezi daty a funkcemi
* Udělá se návrh funkcí, návrh dat, ale důležité je udělat konfrontaci funkcí a dat
* ERA model = Entita Relace Atribut = konceptuální model
* Analýza zdola nahoru znamená, že sesbírám požadavky všech rolí uživatelů (role = rozvrhář, učitel…) a z toho vytvořím datový model – použiju tenhle postup tehdy, když budu chtít, aby ten informační systém pokryl požadavky uživatelů
* Metoda shora dolů znamená, že nejdřív navrhnu a pak určím, co koncový uživatel bude dělat

# Semestrální práce

Vyplnit formulář <https://portal.zcu.cz/wps/PA_Courseware/DownloadDokumentu?id=25736>

Téma je na každém zvlášť

Včasné odevzdání formuláře:

Nejpozději do začátku 7. Cvičení což je taky první prezentační cvičení (5. Listopadu 2009)

Formulář vyplnit elektronicky – na druhou stranu dát ERA model, vytvořen pomocí nějakého počítačového nástroje

## Výběr ze dvou variant

1. MS Access
   1. Fyzický datový model
   2. Naplněno testovacími daty
   3. Formulářová aplikace
   4. Dotazy (pohledy)
   5. Vše uloženo v jednom souboru
2. MySQL
   1. Fyzický datový model
   2. Naplněno testovacími daty
   3. Dotazy (pohledy)
   4. Referenční integrita

## Fyzický datový model

* Po tzv. rozkladu a normalizaci min. 5 tabulek (bez tzv. číselníků)
* Číselník = např. tabulka Barva – 2 sloupce – číslo, barva
* Z datového modelu musí být patrné, že před rozkladem obsahoval relaci M:N
* Datový model by neměl obsahovat relaci typu 1:1
* Integrita databáze

## Konkrétní příklady co se dá řešit

* Půjčovny všeho druhu
* Různá sportovní evidence
* Cestovní kancelář
* Ordinace praktického lékaře
* Všechna témata jsou vyjmenována na courseware

## MS Access – formulářová aplikace

### Nutné

* Pro „každou“ tabulku bude vytvořen jeden odpovídající formulář, který bude nabízet základní operace, které se nad uloženými daty v tabulce dají realizovat: I(nsert), U(pdate), D(elete), S(elect)(výpis obsahu tabulky)
* Jeden dotaz nad jednou tabulkou
* Jeden dotaz nad větším počtem tabulek
* Pro dotazy vytvořit odpovídající formuláře – zakázat změnu dat (pouze pro čtení)

### Doporučené (nepovinné)

* Formulář s podformulářem
* Sestava
* !!! vůbec neřešit konzistenci databáze – např. při prodeji nějakého kusu zboží není nutné zmenšit odpovídající počet kusů na skladu

## MySQL – požadavky na dotazy/pohledy

### Nutné

* Jeden dotaz nad jednou tabulkou
* Jeden dotaz nad větším počtem tabulek
* !!! vůbec neřešit konzistenci databáze – např. při prodeji nějakého kusu zboží není nutné zmenšit odpovídající počet kusů na skladu

## Požadavky na dokumentaci

* Vytištěné podobě A4
* Jednostranný tisk
* Obsah přibližně 4 strany
* Titulek, jméno autora, email, datum
* Zadání (specifikace)
* Nová verze obrázku datového modelu, která byla použita pro implementaci
* Access – stránka nebo dvě směsice funkční analýzy a uživatelského manuálu – popis toho, jak program ovládat, občas doplněné screenshotem
* MySQL – SQL kód dotazů
* Závěr, zhodnocení – informace, která říká, co nám to při tvorbě práce dalo, nedalo, jak by šel program doplnit, rozšířit pro reálné nasazení…
* Stránek může být i více

## Předvedení a odevzdání semestrální práce

* Předvedení bude s autorem (osobní)
* 5 – 10 minut
* Případné dotazy
* Kde: UL409 (dříve v UK430 v úředních hodinách)
* Kdy: 12. Nebo 13. Týden
* Lze předvádět i na vlastním notebooku

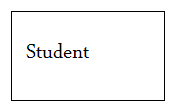
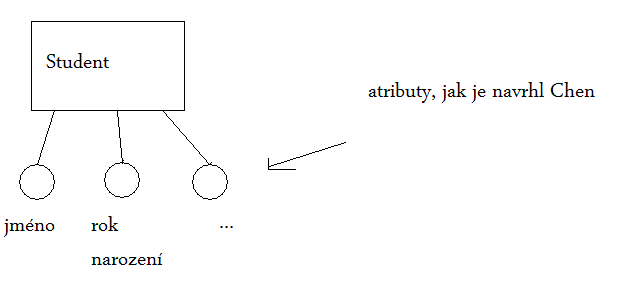
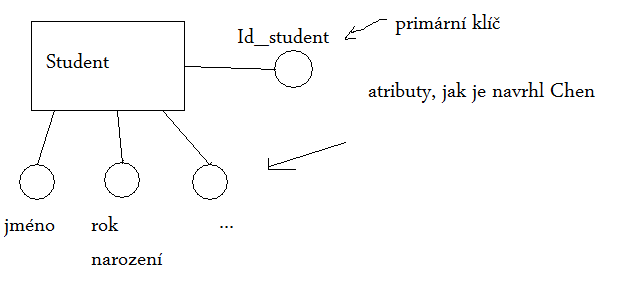
### Odevzdání

Access – 1 soubor, dokumentace v tištěné i elektronické podobě

MySQL – exportovaná databáze v kódu jazyka SQL

# ERA modely

1. Návrhář se musí umět podívat na reálný svět a musí tam vidět to podstatné
   1. Proč je úloha, kterou jsem navrhl nepoužitelná? – protože my si vytváříme tu představu, kterou pak realizujeme
2. Jak to, co vypozorujeme, popsat? Jak to popsat abychom tomu rozuměli my nebo i někdo jiný třeba za rok? A jak to popsat, aby se to dalo v tom SQL naprogramovat?
   1. USTÁLIL SE ZPŮSOB ZÁPISU MODELU, TZV. ERA MODELY
   2. ERA – podle třech základních komponent – entita, relace, atribut
   3. ER model – entita, relace – modifikace ERA modelu, bez atributů, bude u zkoušky
   4. EER model – enhanced entity relation…

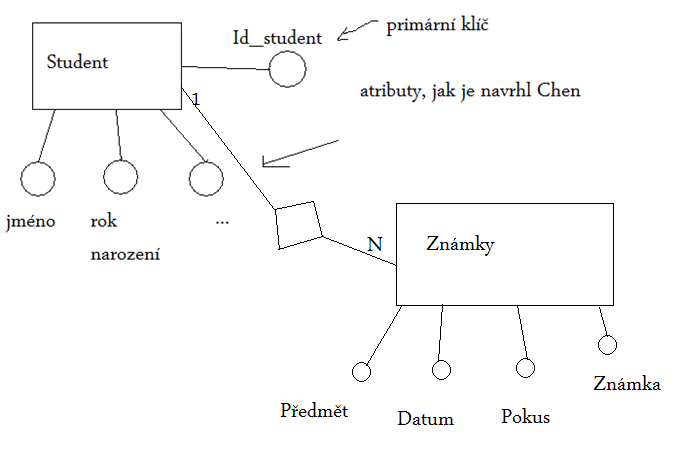
* S návrhem takhle to kreslit vyrukoval už v roce 1976 pán jménem Chen
* Entita
  + Entitou se rozumí model objektu z reálného světa
  + 
  + Správně by to mělo být „entitní množina“, ale říká se tomu jen „entita“
  + Atribut popisuje entitu
  + Atributy jsou určitého datového typu
  + 
  + Pokud se zajímám o to, jak to schéma implementovat, tak u atributu určuji, jakého je typu
  + Atributy jsou různého typu – text, datum, čas, objekt, memo…
  + Primární klíč
    - Pod pojmem klíč je množina atributů, jejichž hodnoty jednoznačně určují entitu
    - Primární klíč je minimální množina atributů, jejichž znalost jednoznačně určí tu množinu
    - Mezi klíčem a primárním klíčem je i kandidát primárního klíče, z těch kandidátů, které jednoznačně určují entitu, pak vybírám primární klíč
    - 
    - Primární klíč musí být unikátní
    - Speciální typ automatické číslo (sekvence) – drahá režie, čísluje furt do pryč, k vymazaným se nevrací
    - Není číslo jako číslo
    - V databázích je standardně číslo ve tvaru tzv. dekadické – číslo může být libovolně dlouhé s libovolnou přesností

U atributu se vždy zamyslet, jestli je to opravdu typu číslo (jestli s tím někde počítám, sčítám to apod.) – např. u osobního čísla použít VŽDY TYP TEXT

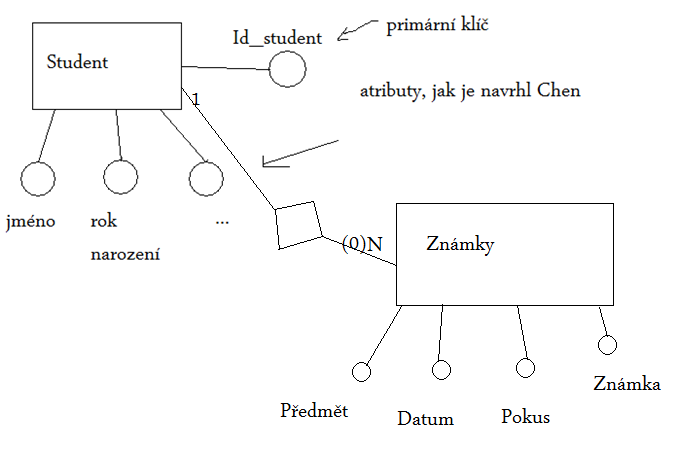
Access je relační databázový systém – neznamená to, že umožňuje vytvářet relace, ale že je odvozen od matematického modelu relace

Relace – vztah mezi entitními množinami

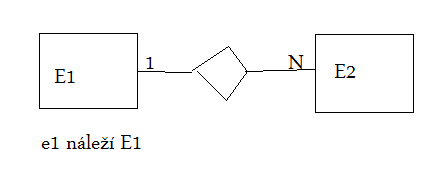
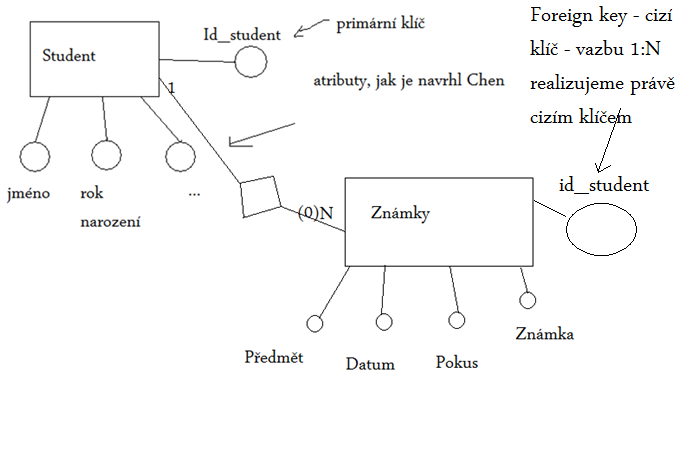
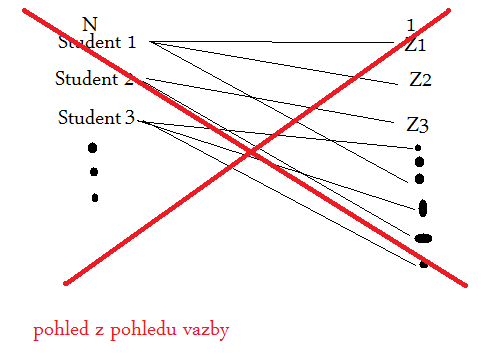
V úvaze se určuje tzv. kardinalita, povinnost výskytu

* Kardinalita – vybíráme jeden ze tří typů, jakého ta vazba je – ty typy se symbolicky značí: 1:1, 1:N, M:N
  + Říká, kolik entit na jedné straně vazby a kolik entit na druhé straně vazby se na té vazbě podílí – ale říká to typově – jestli jedna, nebo víc
  + 

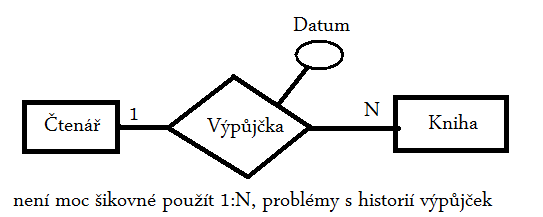
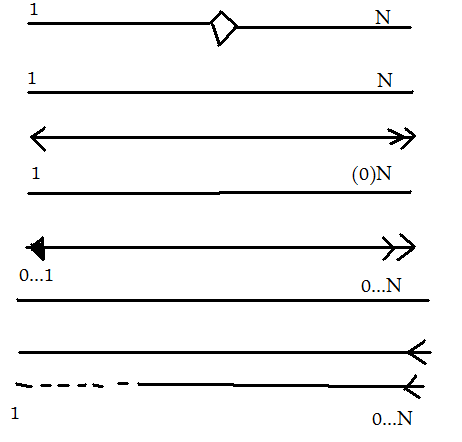
## 1:N



1. N = ta vazba nemusí existovat

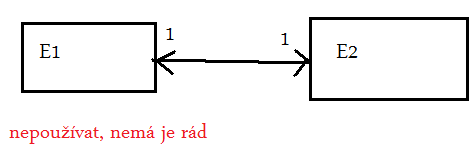
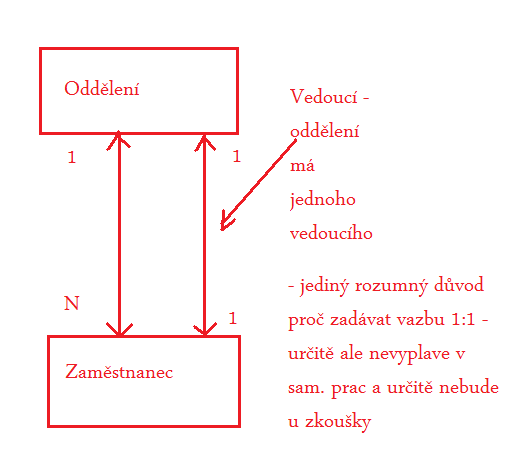
* Když můžeme, nechat ten konec volný – používat to (0)N – je tam méně zámků, transakce pak jede rychleji
* 
* Např. studenti na předmětu
* Vlastně skoro vše je vazby 1:N
* Cizí klíč se v modelu na úrovni tvorby modelu nemusí uvádět
* Cizí klíč není jednoznačný – nesmí být
* V Accessu bacha na „povolit duplicitu“
* 
* Databázisti ale používají pohled z pohledu entit

## Značení

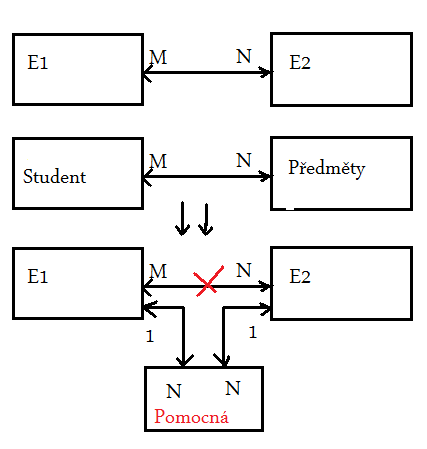
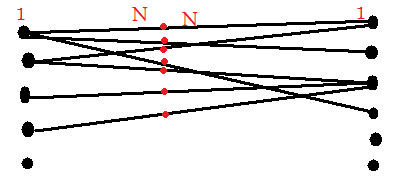
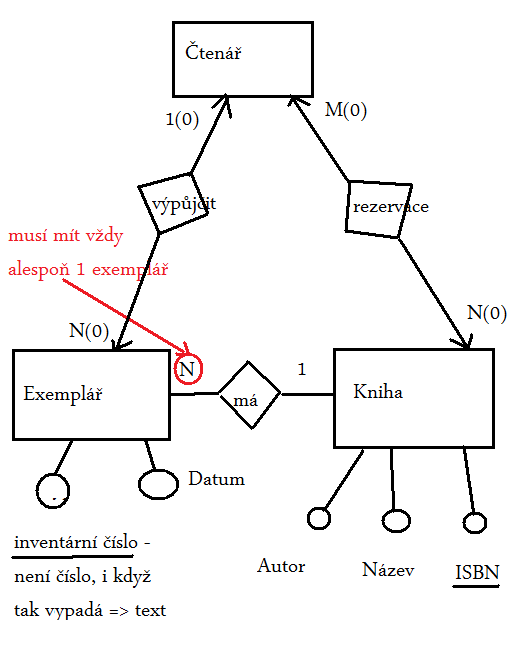
* 
* Takhle to navrhl Chen původně, připouštěl, i že relace má atribut – to se ale hodně rychle opustilo, jak realizovat relaci s atributem??? – jedině další tabulkou…
* Možnost zavedená Chenem dávat atribut vazbě se ukázala jako nepraktická, protože stejně se taková relace musela realizovat jako tabulka, tj. stejně jako jiná entitní množina
* 

|  |  |  |  |
| --- | --- | --- | --- |
| Souborově orientovaný přístup | ERA model | Datová struktura | Relační pohled |
| Soubor | Entitní množina | Tabulka | Relace |
| Záznam (dříve věta - špatně) | Entita | Řádek | n-tice |
| Položka | Název atributu (slangově atribut) | Název sloupce, „sloupec“ | Doména |

## 1:1

* Entitě e1 z entitní množiny E1 přísluší nejvíce jedna entita e2 z entitní množiny E2 a jedné entitě e2 z entitní množiny E2 přísluší nejvíce jedna entita e1 z entitní množiny E2.
* 
* Základní otázka – proč to nespojit do jedné entity???
  + Vazba je neúpovinná
  + Vymlouvat se na to že ušetřím místo tak neušetřím – vše se ukládá s proměnnou délkou
  + 
  + Rychlík nemá rád vazby 1:1

## M:N

* Nelze jí zadat
* Vazbu M:N realizujeme rozkladem vazby
* 
* 
* Jednoduchý model:
* 
* Pozn. Pomocná množina ve vazbě M:N student – předměty je v podstatě tabulka Zápis

## Cvičení

### Zástupné symboly

0 – číslice (0, 1, 2, 3, 4, 5, 6, 7, 8, 9), povinný výskyt

9 – číslice (-||-) nepovinný výskyt

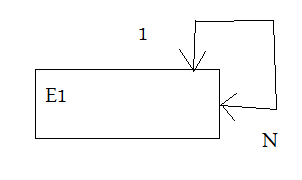
L – písmeno (‚a‘ – ‚z‘, ‚A‘ – ‚Z‘), povinný výskyt

? – písmeno (-||-), nepovinný výskyt

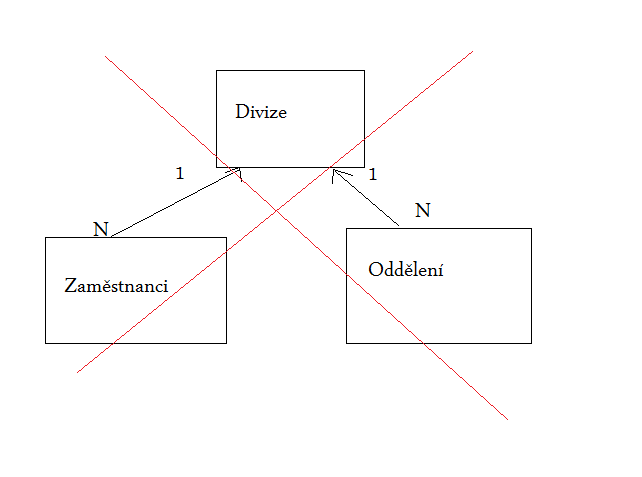
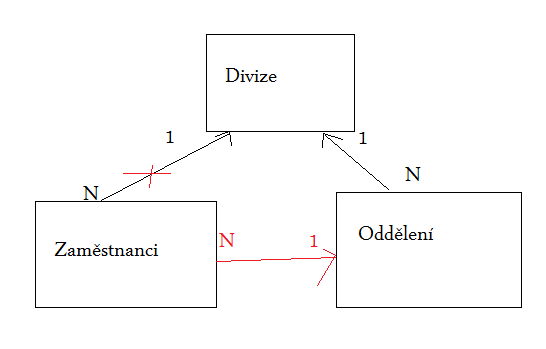
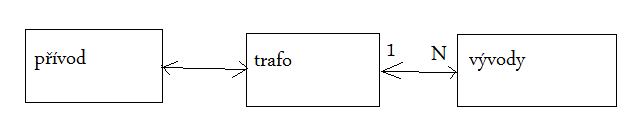
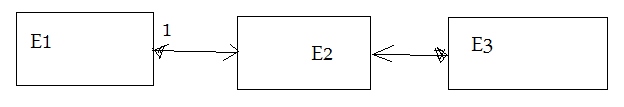
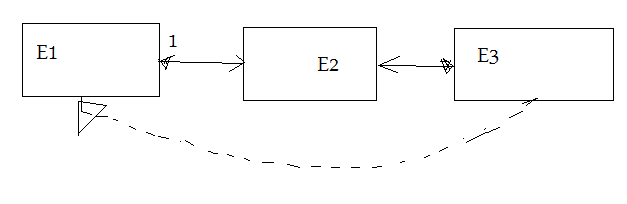
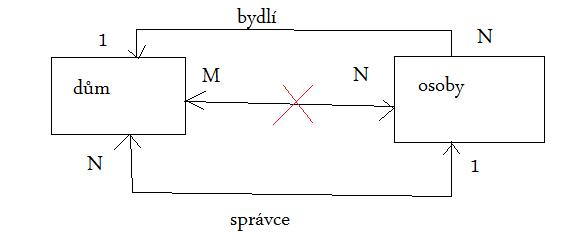
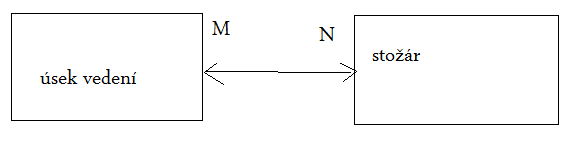
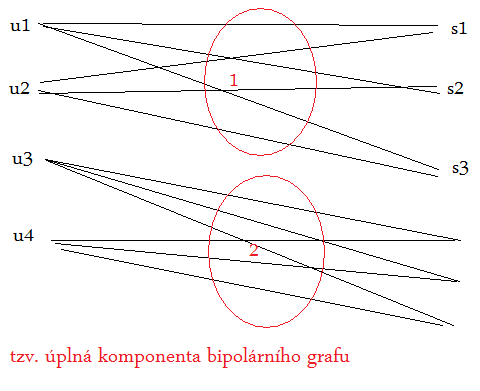
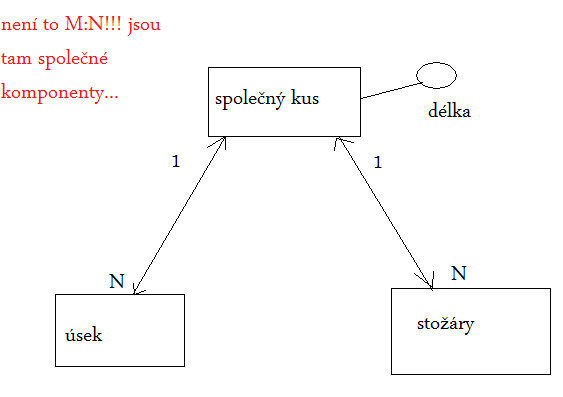
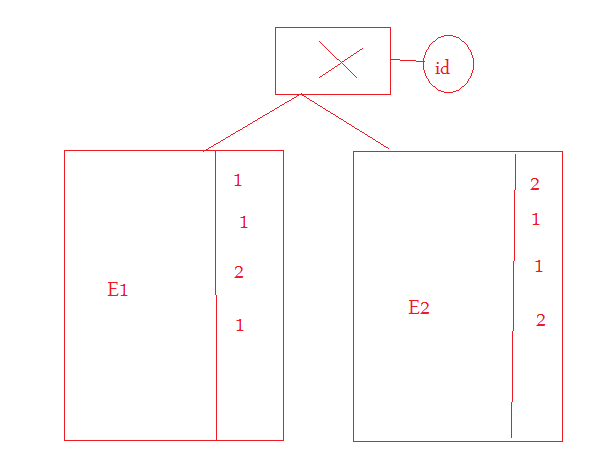
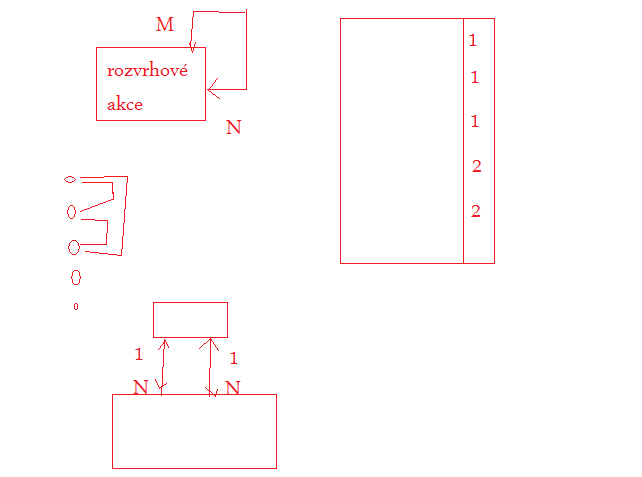
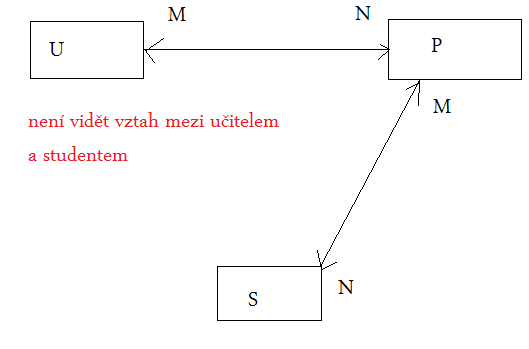
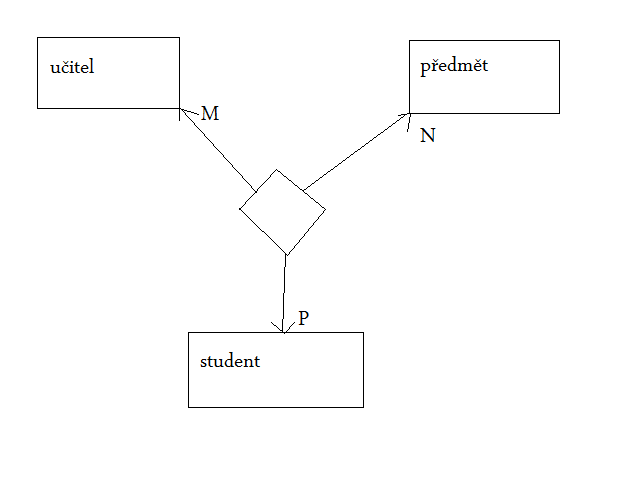
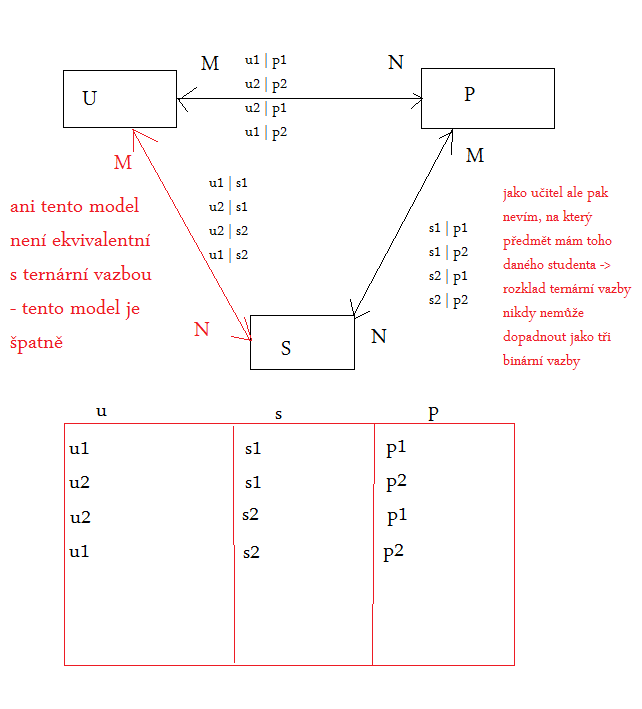
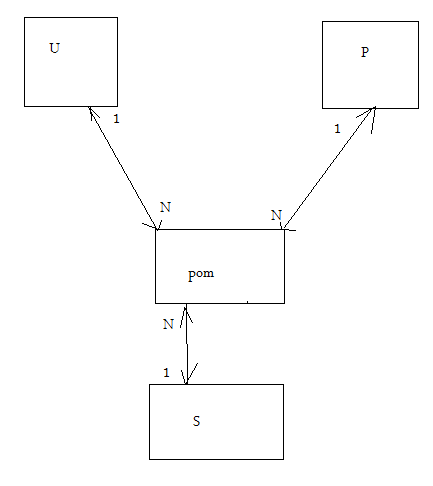
A – číslice nebo písmeno, nepovinný výskyt

a – (-||-), nepovinný výskyt

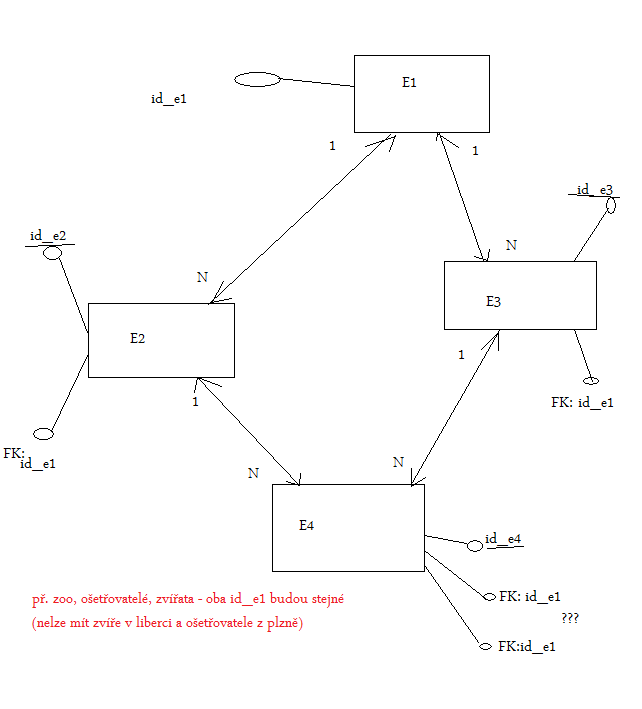
\ - nemá funkci zástupného symbolu, ale vypíná funkci zástupného symbolu – přinutí access zobrazit následující znak

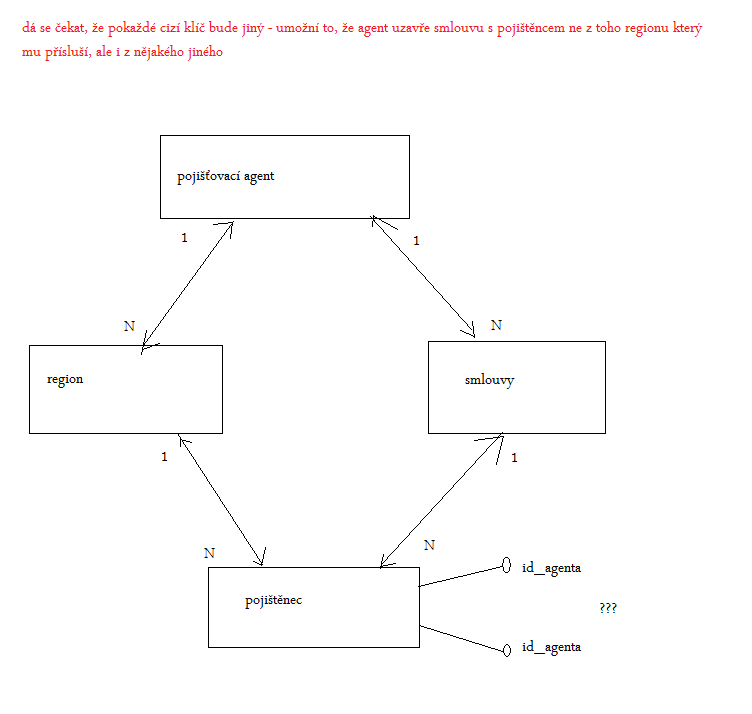
* Mezi dvěma entitami může existovat více vazeb
* Např. entita oddělení a entita zaměstnanci – jedna entita ukazuje, že zaměstnanec je zaměstnán v daném oddělení, druhá je, že ten vedoucí je také zaměstnanec
* Obvykle je vazba binární, ale obecně může být n-ární – může např. spojovat tři množiny
* Výjimkou je unární vazba – ta jde nakreslit v casech
* 
* Např. prerekvizity v předmětech…
* Mohou být i dvě unární vazby – matka, otec… - chci vytvořit model nějakého rodokmenu např.
* Pozn. Slabá entitní množina (opak je regulární entitní množina – to jsou všechny ostatní…)
  + slabá je proto, že k tomu vede jeden ze dvou následujících úkolů – nemůže existovat sama o sobě, aniž by existovala nadřazená množina
  + Nebo ji nemohu identifikovat, když nemám nadřazenou množinu
  + Do klíče musím zabudovat id ze zaměstnance
  + Ano, ale je to archaismus – teoreticky je to možné, ale v praxi se to nepoužívá

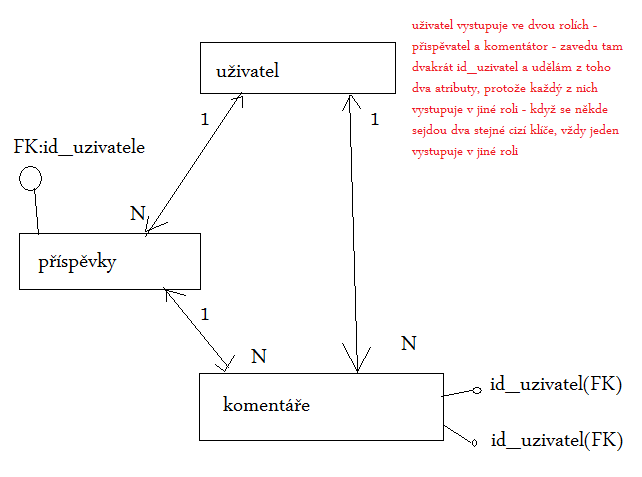
## Na co je třeba si dát pozor

1. Fantrap – např. mám podnik, ten má nějaká oddělení, větší oddělení (divize), pak má zaměstnance, najednou se každá divize začne členit na oddělení – nejčastější, bacha na ní!!! U zkoušky ani v záverečné práci se objevit nesmí!!!
   1. Tento datový model je chybný, protože nelze poznat, v kterém oddělení zaměstnanec pracuje
      1. 
   2. 
2. Chasm trap
   1. 
   2. Tahle past se může projevit v případě, že vazby nejsou povinné
   3. 
   4. Projevuje se , když vazby jsou nepovinné, musím si uvědomit, když vazba není obsazená, jestli potřebuju mít vztah přes tu entitu která zmizela – pokud ano, jsem v té pasti a nezbývá nic jiného, než si udělat novou vazbu mezi E1 a E3 -> dostanu se do problému s cyklem, ale to až dále
   5. 
3. Pozor na vazby M:N
   1. tyto dvě vazby se mne jako pozorovateli mohou jevit jako M:N, ale ve skutečnosti se jedná o dvě vazby 1:N – vazbu M:N nikdy nelze rozložit jako 1:N
   2. 
   3. 
   4. 
   5. 
   6. tento model není náhradou té původní ternární vazby
   7. Tento obrázek není náhradou té původní ternární vazby
   8. 
   9. 

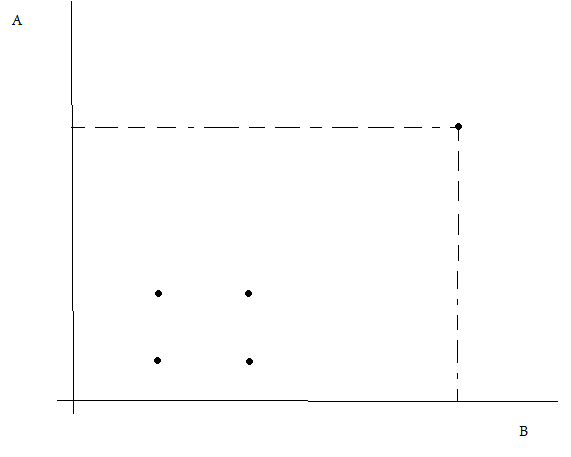
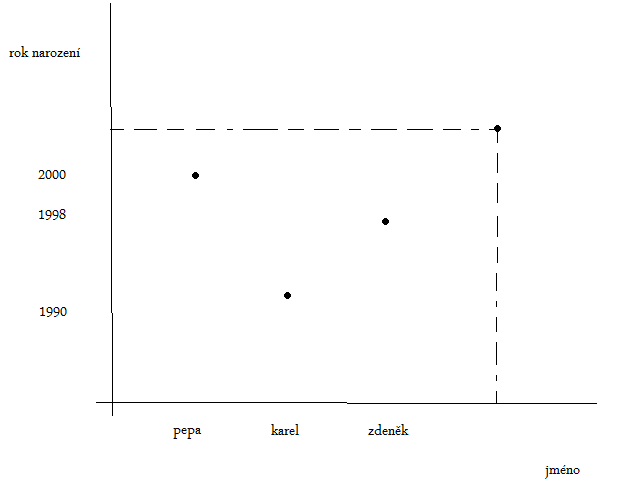
## Cykly v ERA diagramu







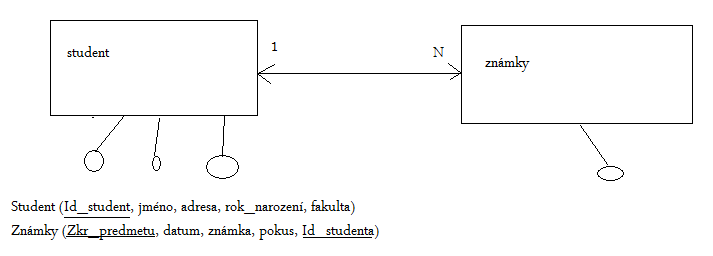
# Relační model dat

* „relační“ ve smyslu matematiky (podmnožina kartézského součinu)
* 
* Osoba
  + Jméno
  + Rok narození
  + 
  + Pro reprezentaci relace bychom zvolili strukturu Tabulka
  + Pokud nás SŘBD nutí, abychom pracovali s tabulkou, jedná se o relační SŘBD
* Pevně daný systém
* {Di, i=1, …}
* R je podmnožinou D1 x D2 x … x Dn
* Prvky relace R jsou uspořádané n-tice [d1, d2, …, dn[ takové, že di náleží Di
* Relace je tabulka s m řádkami a n sloupečky (n říká že kolik je domén, tolik bude sloupečků)
* Doménou rozumíme množinu hodnot, které se mohou vyskytnout jako konkrétní prvek v pozici i (jinými slovy které hodnoty se mohou objevit ve sloupečku i)
* Zavádí se velmi významný pojem – relační schéma – je to jeden z možných popisů, jak popsat data jinak než ERA modelem

## Relační schéma

R(A1, A2, … , An)

Př.



Var b)

Známky (zkr\_predmetu, datum, známka, pokus, Id\_student)

* Eviduju ještě pokusy (vím, že známky bude 4, ale zajímá mne ještě pokus)
* Pozn. Volbou primárního klíče určuji můj pohled na realitu
* U var. A) mám jen poslední pokus (vím, že 5. Byl poslední, ale nevím, jaké známky byly z předchozích pokusů

## Relační schéma databáze

(R, I)

R – seznam relačních schémat, de facto báze dat

I- Integritní omezení – v našem případě jen jedno – nesmí existovat známka bez vazby na studenta

* Již umíme ERA model a relační schéma databáze
* Na ERA model máme nástroje, ale na relační schéma je nutné být velmi opatrný

Z vymezení dále plyne: homogenita sloupců, hodnoty ve sloupcích musí být stejného typu, protože jsou ze stejné domény

V relaci nemohu mít dva stejné prvky -> tedy nemohou být dva stejné řádky v relaci -> musím mít primární klíč

Pořadí řádků v tabulce je nevýznamné

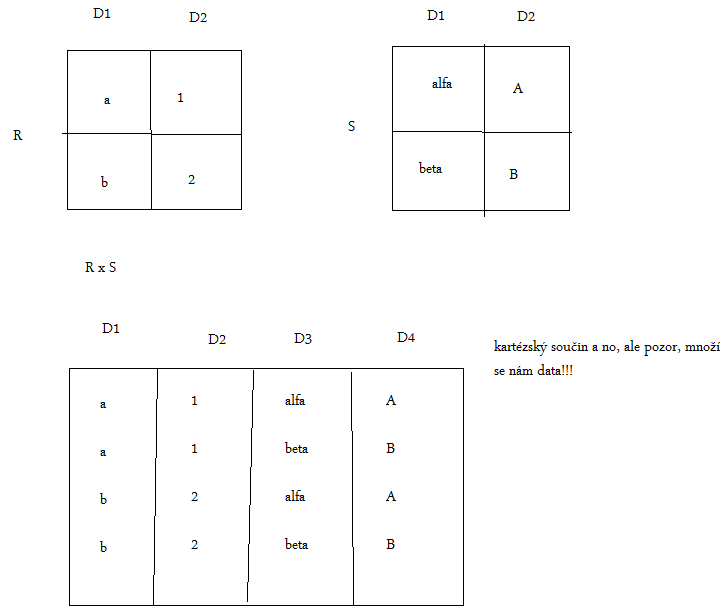
Pokud přehodíme sloupce i názvy, pak je pořadí sloupců také nevýznamné

Důvod, proč se na data díváme jako na relaci, je ten, že existují operace s množinami…

Množinový rozdíl

* Chci tabulku, ze které vyhážu nějaké sloupečky… - není to úplně to pravé

## Operace relační algebry (používané při manipulaci s daty)

1. Kartézský součin
   1. 
2. Pro operaci s daty jsou definovány další tři operace: projekce, selekce, spojení
   1. Projekce
      1. R(**A**) pozn. (**A** = A1, …, An)
      2. Vytvoří relaci se schématem **B** a prvky, které vzniknou z původní relace R, odstranění hodnot atributů A \ B, odstraněny jsou i případné duplicity prvků

|  |  |  |  |
| --- | --- | --- | --- |
| **A1** | **A2** | **A3** | **A4** |
|  | **x** |  | **x** |
|  | **x** |  | **x** |
|  | **x** |  | **x** |

* + 1. **R[B]**
    2. **Př. Student[Jméno] – viz předchozí příklad**
    3. Když budu mít dva studenty se stejným jménem (v původní tabulce student), tak to jméno bude ve výsledku jen jednou!!!
  1. Selekce
     1. Mějme relaci R se schématem A R(**A**)
     2. Operace selekce podle logické podmínky velké fí vytvoří relaci se stejným schématem, a ponechá ty prvky, které splňují logickou podmínku velké fí
     3. „vytvoří relaci se stejným schématem“ = sloupečky zůstanou stejný
     4. Formule (podmínka) fí je bool výraz, kde t1 op t2, kde ti je buď konstanta, nebo jméno atributu a „op“ = { <, <=, =, >=, >, != }
     5. Značení je R(velké fí)
     6. Př.Student(adresa = ‚Plzeň‘)
  2. Operace spojení
     1. Operace spojení relací se schématy R(**A**) a S(**B**) vytvoří relaci se schématem A u B tak, že projekce na atributy A je z relace R a projekce na atributy B je z relace S
     2. [A] … R B…S
     3. R\* S
     4. R[A op B] S dostanu z toho {(r || s); r náleží R, s náleží S a současně (r[A] op s[B])}
     5. **Zavedení podmínky théta něco mi vyloučí, abych pracoval s celým kartézským součinem**
        1. **Tzv. théta (op) spojení – to nejobecnější**
           1. **Př. R(M, N, O, P) S(A, B)**
           2. **Konkrétní hodnoty:**

**x, 101, 5, a 5, a**

**Y, 106, 3, A 10, c**

**Y, 111, 9, B 8, e**

**… …**

* + - * 1. **R[o >= A] S**
        2. **M N O P A B**
        3. **X, 101, 5, a, 5, a**
        4. **…**
        5. **Y, 111, 9, b, 5, a**
        6. **…**
        7. **Y, 111, 9, b, 8, c**
        8. **…**
        9. **Pozn. Srovnáváme atributy a chceme, aby byly stejného typu**
        10. **– umožní porovnat atributy z různých domén (musí být stejného typu), ale mohou být různého významu**
      1. **Přirozené spojení**
         1. **Je odvozeno od théta spojení, kde se použije znaménko =**
         2. **Jeden z porovnávaných atributů se vyloučí ze schématu**
         3. **Př. Výrobek(č\_výrobku, cena)**
         4. **TX10, 30**
         5. **EL82, 110**
         6. **SP40, 40**
         7. **ZY162, 500**
         8. **Relace Zásoby(č\_skladu, č\_výrobku, množství)**
         9. **100, EL82, 5000**
         10. **100, SP40, 120**
         11. **110, EL82, 600**
         12. **Hledáme relaci Výrobek \* Zásoby**
         13. **(č\_výrobku, cena, č\_skladu, množství) – všimnout si: číslo výrobku je tam jen jednou !!!**
         14. **EL82, 110, 100, 5000**
         15. **EL82, 110, 110, 600**
         16. **SP40, 40, 100, 120**
         17. **– vytipuju množinu stejných atributů, v této množině stejných atributů udělám kartézský součin všech stejných hodnot – číslo výrobku tam bude jen jednou – množina stejných atributů se tak ve výsledku objeví jen jednou**
      2. **Kompozice – pod pojmem kompozice si představíme to samé co u předchozího, ale vyhodíme oba atributy**
         1. **POZOR!!! ČASTÁ CHYBA – ŘÁDKA S EL82 SE NAPÍŠE JEDNOU A MNOŽSTVÍ SE SEČTE – TAKHLE NE!!! (různé sklady)**
      3. **Některé operace lze vyjádřit pomocí jiných operací**
         1. **Např. průnik lze vyjádřit pomocí množinového rozdílu,…**
         2. **Ale dá se vytipovat, že nemusíme mít ty operace všechny, ale stačí jen některé, abychom s tím mohli udělat prakticky vše – minimální množina operací – lze z ní sestavit libovolná operace**

**Pozn. Množina min. operací je např.: STAČÍ NÁM SJEDNOCENÍ, KARTÉZSKÝ SOUČIN, MNOŽINOVÝ ROZDÍL, SELEKCI A PROJEKCI**

Př.

EXEMPLÁŘ (ISBN, inv\_číslo, datum\_nákupu, cena, země\_vydání)

Výpůjčka(inv\_číslo, číslo\_čtenáře, datum\_zpět)

* Výsledná tabulka po aplikování selekce se jmenuje T
* T(isbn) -> tabulka o jednom řádku a jednom sloupci
* Tak jako výsledkem jakékoli operace v relační algebře je opět operace, tak i zde je výsledkem jakékoli operace s daty zase tabulka
* Např. měli jsme tam dvě relace jako zadání příkladu, pak netriviální poznámku že výsledkem je tabulka, Dotaz: v těchto tabulkách nalezni ISBN všech vypůjčených knih – bude u zkoušky!!!
* (EXEMPLÁŘ(Země\_vydání=‘GB‘)[Isbn, Inv\_číslo]\*VÝPŮJČKA)[Isbn]
* tahle konstrukce nemusí být jediná
* Formule operací relační algebry jsou užitečné ve chvíli, kdy chceme optimalizovat dotazy
* Než použiju operaci spojení, tak vyházet vše co jde

Př. Student – Známka

* Navazuje na příklad student – známka když jsme si ukazovali, jak se píše relační schéma
* Vypiš jména studentů, kteří mají z db1 jedničku
* Známka(Zkr\_předmětu=‘DB1‘ a současně známka = 1)\*Student)[jméno]
* … spojit relací Student (\*Student) a z toho projekce na atribut jméno ([jméno])
* Lze začít mi jinak, např. nejdřív známka spojeno se studentem a pak teprve známka, tak je to neoptimalizované, je to špatně
* U zkoušky je také příklad, abychom z nějakého zadání, které bude formou relačního příkladu sestavit dotaz (někdy v relační algebře, někdy SELECT…)

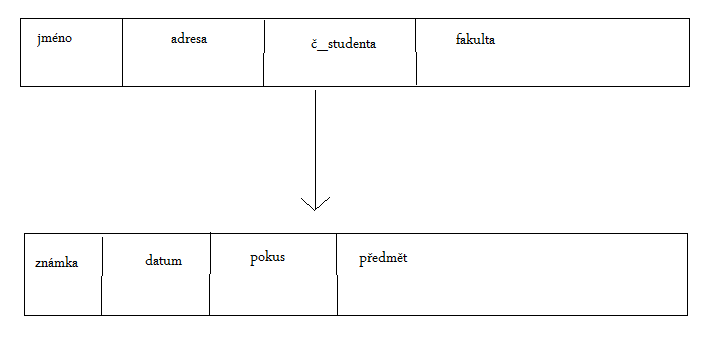
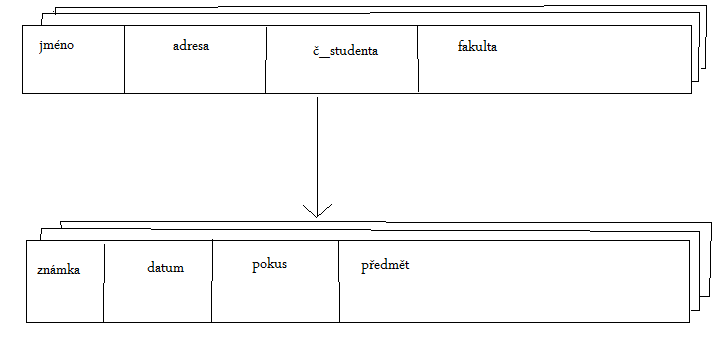
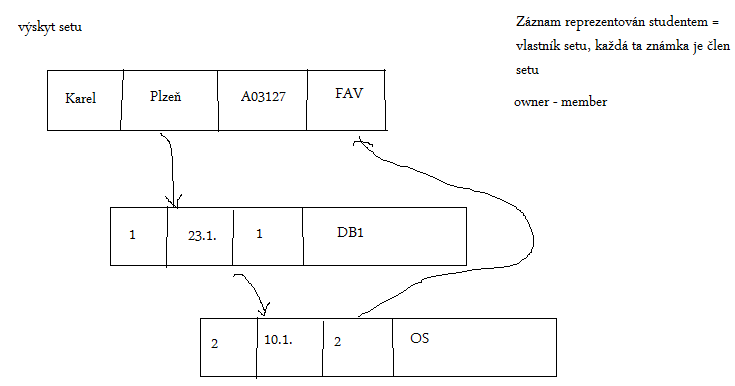
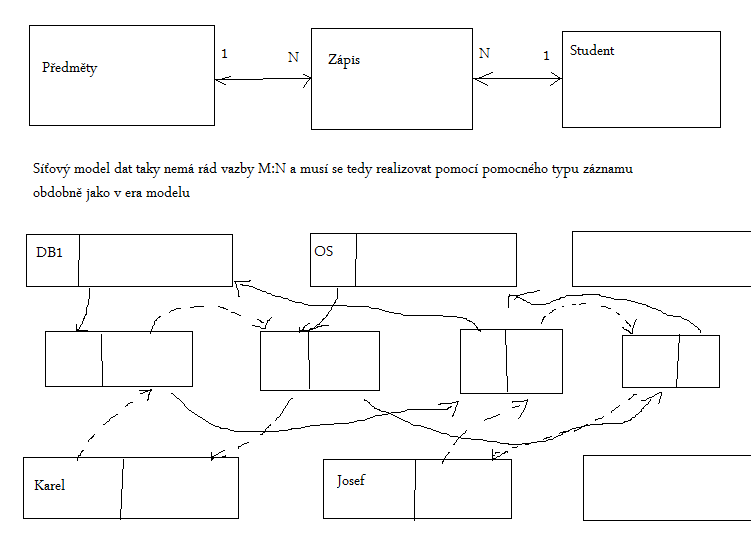
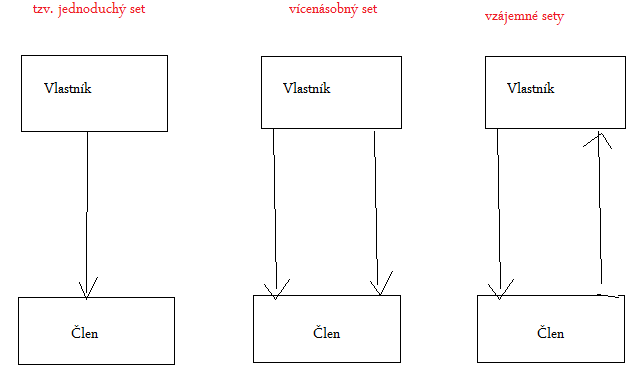
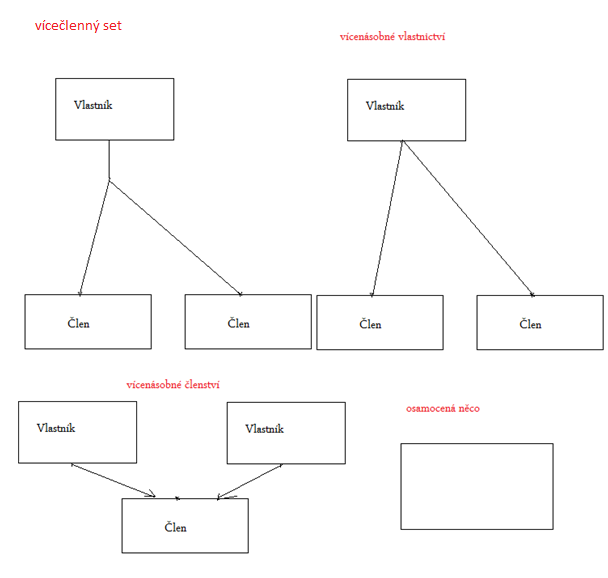
# Síťový model dat

* Je spojen převážně s programovacím jazykem COBOL
* Nikdo to už neumí, přesto je to zapotřebí
* Pracuje se s typem záznamu
* Vyšlo to ze souborově orientovaného pohledu na zpracování dat
* Typ záznamu obsahuje názvy položek
* Obsahuje spojky – spojka je pointer co někam ukazuje
* Spojka v tomhle pojetí ukazuje na souvislost entit (ukazuje, že je student a ten má známky) – ukazuje na související data (související entity)

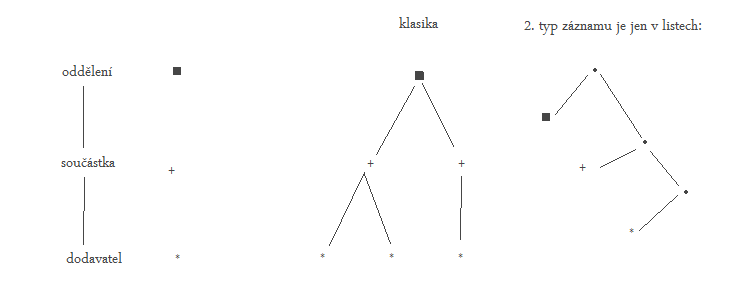
Typ záznamu

Spojka

Výskyt setu

* 
* 
* 
* 
* Na tomto obrázku jsou namalovány konkrétně 4 výskyty setu – výskyt setu je s konkrétníma datama
* 
* 
* 1971 – DBTG CODASTL
* 4 principy určující co je síťový model dat:

1. existuje množina typů záznamů (typ záznamu předměty, známky, učitel)
2. existuje množina pojmenovaných spojek
3. každá spojka je funkcí alespoň v jednom směru (vychází z matematického modelu)
4. spojka typu „sama na sebe“ („Lii“) se nepřipouští

* reprezentantem je programovací jazyk COBOL, používá se dodnes
* dalšími reprezentacemi síťového modelu z historie – velmi často se nabízelo programátorům, aby se využívalo hostujícího jazyka (databáze, kterou jsme neviděli a sada procedur, která k té databázi přistupovala)
* manipuluje se také s pojmy povinné/nepovinné a automatické/manuální členství
* speciálním typem síťového datového modelu je hierarchický model
  + struktura dá dohromady strom
  + při realizaci hierarchických systémů se dospělo ke dvou typům stromů
  + 
  + Uzamykání záznamů z pohledu zpracování transakcí strom výrazně napomůže
  + Pozn. Jsou dva typy reprezentace – každý uzel reprezentuje typ záznamu/typ záznamu je jen v listech

# Normální formy

* Snaží se odstranit neopodstatněnou duplicitu dat v tabulce

## Závislost atributů

* Populace – populací nazýváme hodnoty atributů relace R v daném čase
* Závislost atributů nás zajímá, pokud platí pro všechny populace
* Z vyplněné tabulky nemohu usoudit, že tam ta závislost je, ale mohu vyloučit, že tam není

## Funkční závislost

* Popisuje vztah mezi atributy téže relace
* Definice: Nechť A a B jsou atributy relace R. Budeme říkat, že atribut B funkčně závisí na atributu A, jestliže pro všechny populace relace R platí, že pro libovolnou n-tici u,v je prvkem relace R, že u.A (hodnota atributu A v řádku U) == v.A => u.B == v.B (tvrdá implikace jestliže, pak – obráceně neplatí)
* Značíme: A->B, atributu A říkáme determinant

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| u | Hodnota1 |  | Hodnota2 |  |
|  |  |  |  |  |
|  |  |  |  |  |
| v | Hodnota1 |  | Hodnota2 |  |
|  |  |  |  |  |

Pozn. Jiné vymezení: Ke každé hodnotě atributu A existuje nejvýše jedna hodnota atributu B. (pozn. Rozdíl oproti definici je pouze v „null“ (připouštíme prázdnou hodnotu).

* Př. Relace Výuka (**P**ředmět, **U**čitel, **M**ístnost, **H**odina, **S**tudent, **Z**námka)
  + Pod pojmem předmět chápeme přednášku
  + Hodinu chápeme jako určení času
  + P->U ? opatrně – možná jo, ale jen pokud předmět neučí někdo jiný
  + HM->P ? ANO, platí
  + HU->M ? ANO
  + PS->Z ? ANO
  + HS->M ? ANO, pokud se to nekryje s jiným předmětem
  + Je vidět, že závislostí lze najít větší množství – musím hledat všechny? Když ne, tak které?
  + Pozn. Relace nahoře není hezká, nesplňuje tu normální formu
  + Pokud uvažuju relaci a v relaci mám vytipováno co je primárním klíčem, tak atributy relace jsou závislé na primárním klíči – každý atribut relace na primárním klíči závisí
  + Pomocí závislosti lze definovat primární klíč
    - R(Ω) k je ostrou podmnožinou omega
    - Potom k nazvu primárním klíčem relace R, jestliže z k plyne omega
    - Neexistuje k‘ které je podmnožinou k takové, že z k‘ plyne omega
  + Jak předchozí příklad rozmotat?
    - Armstrongova pravidla (1974)
      * Cílem je vnést pořádek do toho, že mám množinu funkčních závislostí, která vznikla bez ladu a skladu.

1. **reflexivnost** (triviální funkční závislost) – mějme množinu atributů X a Y jako podmnožinu menší nebo rovnou množině atributů X. Potom X->X nebo X->Y
2. **Tranzitivita** – jestliže platí, že z X plyne Y a současně z Y plyne (je determinantem pro) Z, tak X->Z
3. **Spojování** (kompozice) – X->Y a současně X->Z pak X->YZ
   * + - Tato pravidla lze dokázat, ale už je pan Armstrong dokázal před námi
       - Soubor pravidel nemusí být vždy v tomto tvaru, ale vždy musí být korektní (platí), úplný (na jejich základě lze odvodit všechny ostatní závislosti) a nezávislý (žádné z těch tří pravidel nelze odvodit pomocí těch dvou zbývajících).
4. Projekce (dekompozice)
   * X->YZ => X->Y a X->Z
   * Předpoklad je, že z X->YZ
   * Na YZ podle reflexivnosti závisí Y i Z, protože je to vlastní podmnožina toho Z
   * Z jedničky: YZ->Y a YZ->Z
   * Z dvojky: X->YZ a YZ->Y => X->Y
5. Doplnitelnost
   * X->Y a X je podmnožina W => W->Y
   * Důkaz za dcv.
6. Spojování II
   * X->Y a W->Z => XW->YZ
7. Pseudotranzitivnost
   * X->Y a YW->Z => XW->Z

Př.

A->B a B->CD pak AE->D

Předpoklad: E určitě nebude podmnožinou atributu A

Nabízí se tranzitivnost

Z dvojky: A->B a B->CD => A->CD

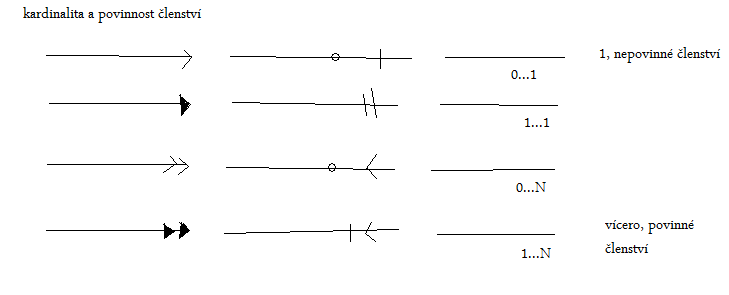
Z malého áčka: A->C a A->D

Z malého béčka: AE->D

## Použití Armstrongových pravidel

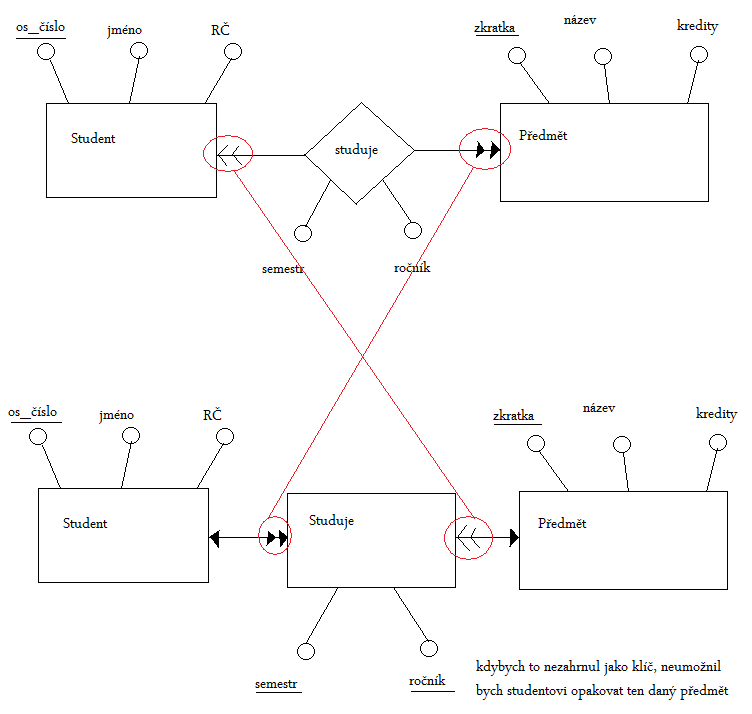
* Použiju je, pokud mám divokou relaci (tzv. úplnou relaci)
* Mám množinu funkčních závislostí
* Výsledkem bude rozklad úplného relačního schématu – chci, aby výsledek měl tu vlastnost, že tam nebudou nadbytečné závislosti = výsledek je rozklad univerzálního schématu
* F – množina všech funkčních závislostí k danému problému – k této množině já určím to, čemu matematici říkají uzávěr – množina všech funkčních závislostí, které lze z F odvodit – je to NP problém
* Elementární funkční závislost = funkční závislost, která má na pravé straně jen jeden atribut
* Hledáme takové pokrytí G, aby platilo, že G+ = F+
  + Tento krok říká, že najdu jinou množinu funkčních závislostí, ze které lze vygenerovat stejnou množinu závislostí jako z F
* F‘+ = F+ (F‘ je množina elementárních funkčních závislostí)
* Redundantní závislost - „f“ je redundantní (nadbytečné) v F pokud (F-{f])+ = F+
* Hledám neredundantní pokrytí
* Tímto postupem v praxi ale nejdeme, volíme postup mnohem jednodušší
* Pozn. Výsledek není jednoznačný – to, co tu nejednoznačnost dělá, je postup odebírání redundantních závislostí

Co si z dneška pamatovat? Jak je definovaná funkční závislost, vědět, že je dobré ty závislosti najít, ale neznamená to, že to stačí…



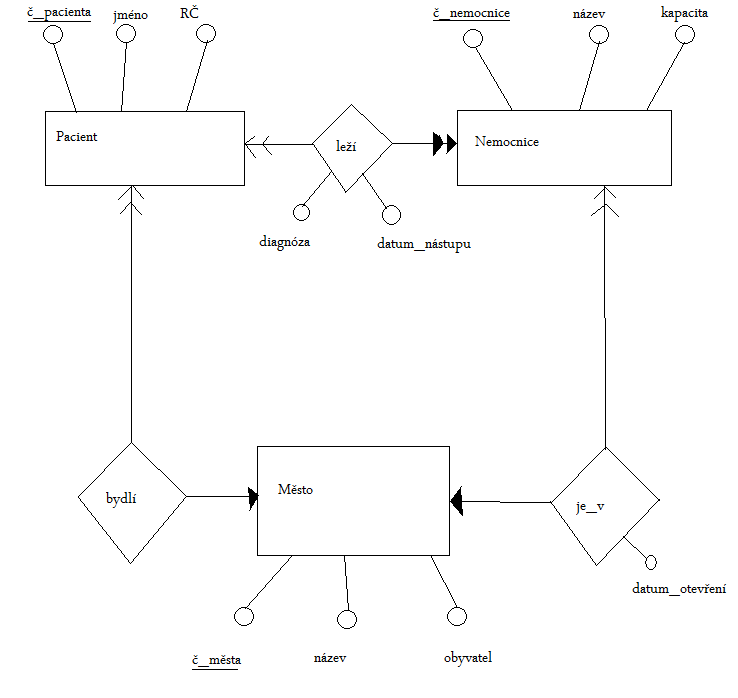
## Rozklad relace M:N

* Student studuje předměty
* Předmět studuje více studentů
* Os\_číslo, jméno, RČ



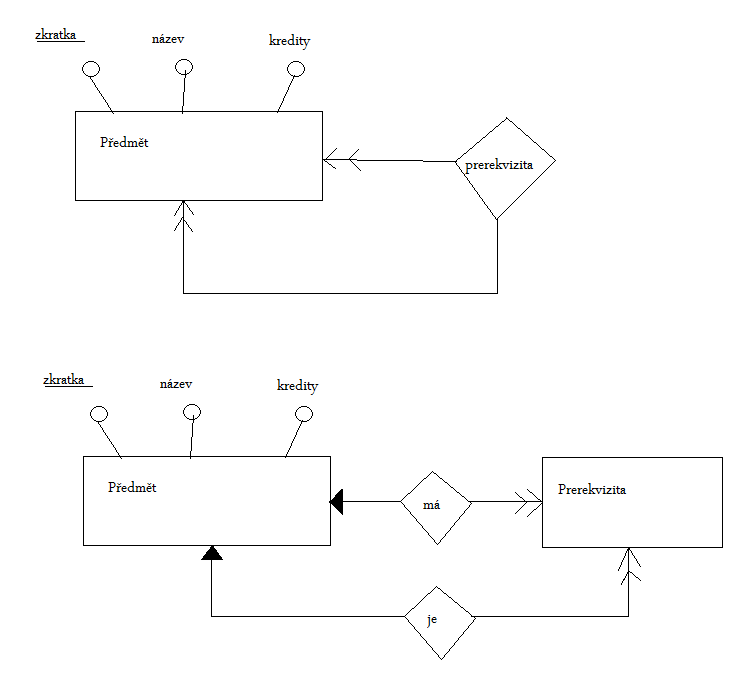
Př.

* V nemocnici leží několik pacientů
* Ve městě je několik nemocnic
* Každý pacient bydlí ve městě
* Předpokládá se globální číslování přes všechny nemocnice (pacienti, kteří tam budou figurovat, budou mít všichni unikátní číslo)



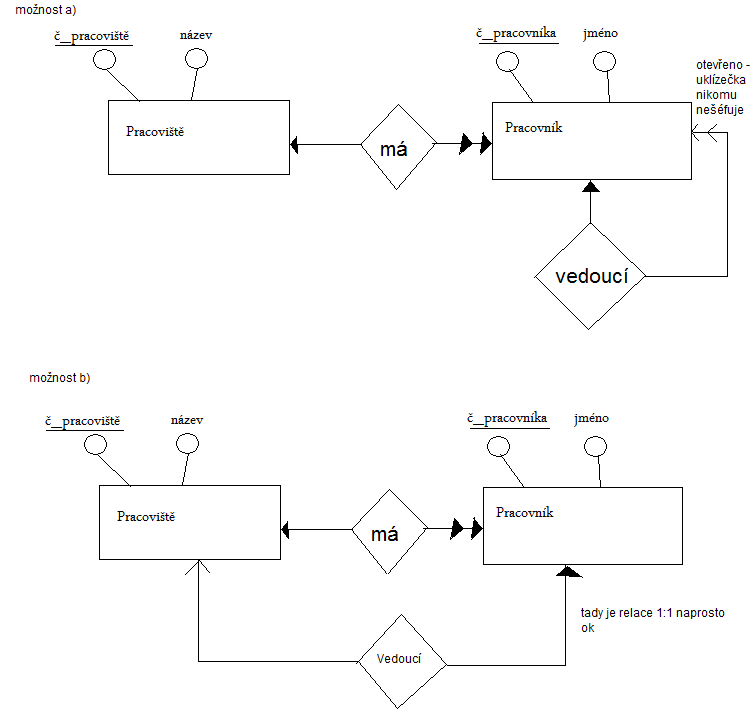
Př.

* Prerekvizity (podmiňující předmět):
  + Každý předmět může mít své prerekvizity
  + Každý předmět může být prerekvizitou jiného předmětu

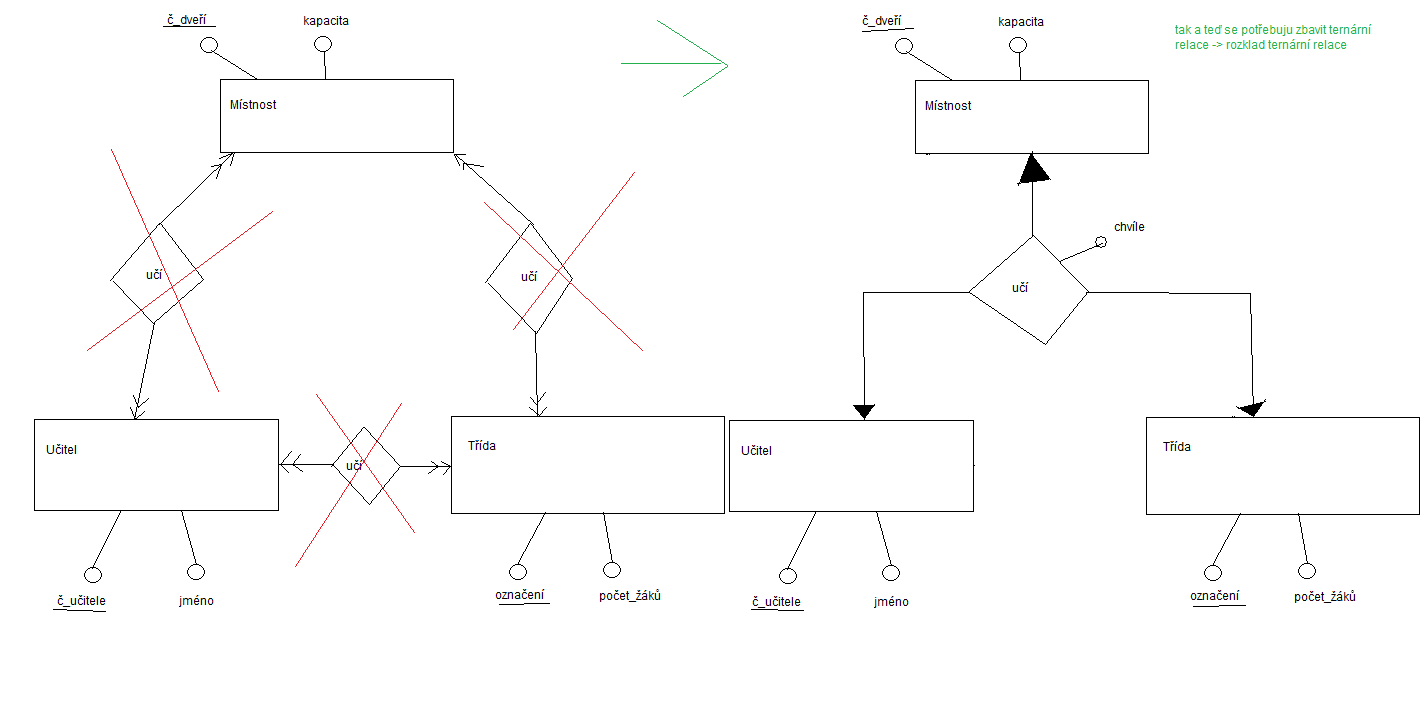


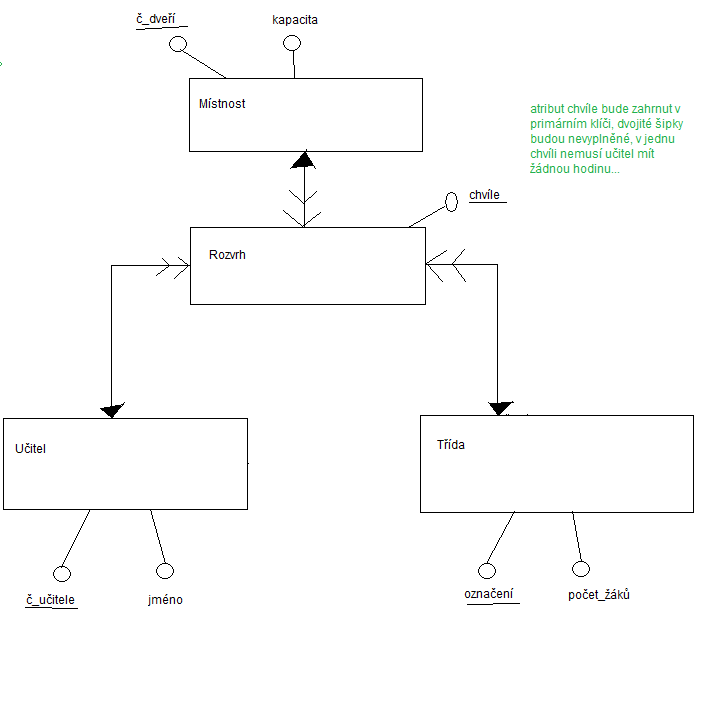
Př.

* Pracoviště
* Každé pracoviště má své pracovníky
* Každý pracovník má svého nadřízeného pracovníka

  
Př.

* „specialitka“
* Rozvrh na ZŠ
* V jednu chvíli v jedné místnosti učí jeden učitel jednu třídu žáků.





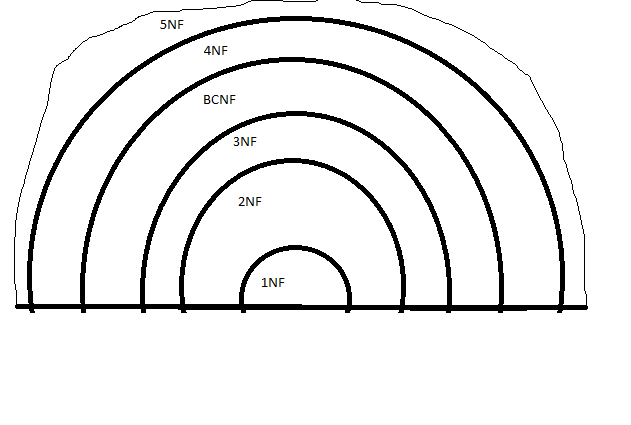
# Využití Armstrongových pravidel

* Uzávěr množiny atributů X vzhledem k F (množině všech funkčních závislostí které jsme si v tom problému nadefinovali) je množina všech atributů funkčně závislých na X (množina všech atributů které z X mohu odvodit)
* Př
  + R(W, X, Y, Z), F: W -> Z, YZ -> X, WZ -> Y
  + Budeme hledat klíč (primární klíč) z atributů W, X, Y, Z podle těch závislostí
  + (WZ)+={W,X,Y,Z} „WZ a jejich uzávěr je množina všech atributů které množina obsahuje – pokud to dokážu, tak množina WZ je primární klíč“
    1. Hledám: WZ->W, WZ -> X, WZ -> Y, WZ -> Z – pokud to dokážu, pak WZ je určitě klíčem a mohlo by být primárním klíčem
    2. WZ->W – to umíme, viz armstrongovo pravidlo číslo jedna
    3. WZ -> Y – to je dáno, je to v té množině F
    4. WZ -> Z – opět armstrongovo pravidlo jedna – reflexivnost
    5. Zbývá dokázat WZ -> X, zkusím že X je na něčem závislé, schválně jestli se k něčemu dopracuju přes to X: WZ -> Y(dáno) a WZ -> Z(reflexivnost) – pravidlo číslo tři – WZ -> y a současně WZ -> Z => WZ->YZ, z tranzitivnosti WZ -> YZ a současně YZ->X => WZ->X
    6. Čili dokázal jsem, že ten primární klíč bude schován někde v tom WZ
  + W+={W,X,Y,Z} – opět zkoumám zda, platí: W->W (platí, reflexivnost), W->X, W->Y, W->Z (je dáno z F)
    1. Chybí mi dokázat ty dva chybějící vztahy
    2. W->W, W->Z – nabízí se vzít kompozici (3) W->WZ
    3. Z F: WZ->Y z (2) pak plyne, že W->Y
    4. Zbývá už jen tedy, zda z W->X: z F: W->Z a právě bylo dokázáno: W->Y a z (3) pak plyne W->YZ a z F: YZ->X pak z (2) => W->X
    5. Závěr je, že W musí obsahovat klíč a to proto, že jsme dokázali, že uzávěrem W jsou všechny atributy v relaci
* Čili ukázali jsme si, že mám-li dáno nějaké relační schéma a množinu funkčních závislostí, které v tom schématu platí, tak na základě odvozování mohu určit klíč původní relace – nepřímo se tedy dotýkáme poznámky z minula (použití univerzálního schématu a funkčních závislostí a na základě toho s tím mohu pracovat, ale nepoužívá se to)

# Normální formy

* Ukládání do tabulky není jen uložení struktury, ale i sémantiky (ukládaná data mají i sémantiku) – ta je dána vztahy mezi atributy

## 1. Normální forma

* Nevhodné uložení může způsobit nepříjemné duplicity a jejich odstranění se věnuje problematika normálních fo
* Relační model předpokládá, že komponenty relace nemohou být opět relace – pokud toto relace splňuje, pak je v první normální formě
* Jakmile z toho uděláme tabulku, pak máme jistotu, že je to v první normální formě
* Př.
  + LET(číslo\_trasy, Odkud, Kam, Rozvrh)
  + ROZVRH(Den, Čas\_odletu)

## 2. normální forma

* už musím zkoumat závislost atributů
* musíme dodefinovat tzv. silnou funkční závislost
  + Nechť v relaci R platí A->B
  + Říkáme, že B silně funkčně závisí na A, jestliže neexistuje žádná vlastní podmnožina A‘ je podmnožina A taková, že by platilo A‘ -> B
  + Říká to, že A nebude jednoduchý atribut, ale bude složen z více atributů – jestliže se stane, že z té složeniny nemohu nic vyhodit, pak je to silná fční závislost
* Relace R je ve 2NF (2. Normální formě) jestliže je v 1NF a jestliže každý atribut který nepatří ke klíči relace R silně závisí na klíči relace R.
* Praktický návod: Vemu relaci r, u relace R určím klíč a pokud je klíč nějaká složenina (více než jeden atribut) budu zkoumat, jestli každý atribut té relace závisí na celém tom složeném klíči – jestliže zjistím že nějaký atribut závisí jen na tom jednom atributu z klíče, pak to není splněno
* Př
  + DODÁVKA(Č\_dodavatele, Č\_součástky, Adresa\_dodavatele, Množství)
  + Budu zkoumat, zda z čísla dodavatele plyne množství, nebo z Č\_součástky plyne množství…
  + Protože platí fční závislost Z čísla dodavatele plyne adresa dodavatele, je porušena 2NF.
  + Provede se tedy dekompozice – převod do druhé normální formy provedeme tak, že rozložíme původní relaci na dvě (případně více) a to podle funkční závislosti která 2NF kazí.
  + Výsledek:
    1. DODÁVKA\_1(Č\_dodavatele, č\_součástky, množství)
    2. DODAVATEL(Č\_dodavatele, Adresa\_dodavatele)

## 3. normální forma

* relace je ve 3. Normální formě (3NF), jestliže je v 2NF a žádný atribut, který není součástí klíče, není tranzitivně závislý na klíči.
* Vemu relaci, určím u ní klíč, vím že je v 2NF, např. klíáčem je jen jeden atribut – tranzitivita je schovaná v tom, že každý atribut je závislý na klíči – jestliže hovořím o tranzitivní závislosti na klíči – ZKOUMÁM, ZDA NEEXISTUJE FUNKČNÍ ZÁVISLOST MEZI NEKLÍČOVÝMI ATRIBUTY
* Př.
  + ZAMĚSTNANCI(Č\_zaměstnance, oddělení, budova) – předpoklad, že oddělení je v jedné budově
  + Předpoklad v podstatě říká, že když znám oddělení, tak znám i budovu
  + Opět je zde neopodstatněná duplicita, problémy při změně budovy
  + Vadí vztah Oddělení->Budova
  + Jak se to spraví?
  + To, co tu 3NF kazí, vytáhneme do speciální relace a v té původní zbyde zbytek
  + ZAM(Č\_zaměstnance, oddělení)
  + ODD(Oddělení, Budova)

|  |  |  |  |
| --- | --- | --- | --- |
| A1 | A2 | A3 | A4 |
| a | 1 | A | Alfa |
| b | 1 | C | Beta |
| c | 2 | B | Beta |
| a | 3 | A | Gama |
| b | 1 | C | Delta |

* Pro tento příklad předpokládejme, že závislosti, které lze odvodit z těchto dat, platí pro všechny populace relace (populace = hodnoty atributů v daném čase) – tohle v praxi nikdy nemůže platit, je to hodně odvážné tvrzení
* Je ta tabulka v 1NF, 2NF, nebo 3NF? – příklady tohoto typu jsou v písemce
* 1) určit klíč
  + Může být klíč A1? Nemůže, opakuje se, A2? A3? A4? Nemůžou
  + Může být klíčem A1 a A2? Nemůžou – dva stejné řádky
  + Vybereme A1, A4 například – nejednoznačné řešení, ale A1, A4 může být klíčem
  + 2NF – budu zkoumat 4 fční závislosti – A1->A2?, A1->A3?, A4->A2?, A4->A3? – tyto čtyři závislosti potenciálně mohou porušit 2NF pokud platí – zda neklíčový atribut závisí na podmnožině klíče
  + A1->A2? „a-1“, „a-3“ -> neplatí; A4->A2 -> neplatí (1-beta, 2-beta); A1->A3 – platí!!!
  + Není v 2NF protože A1->A3
  + Není v 3NF protože není v 2NF
  + Ale to není konec – mohl by být klíč A2, A4?
    1. 2NF: A2->A1, A4->A1, A2->A3, A4->A3
    2. A2->A1? Neplatí, A4->A1? Neplatí, A2->A3? Neplatí, A4->A3? Neplatí
    3. Závěr: relace je ve 2NF protože A2 neplyne A1, A4 neplyne A1, A2 neplyne A3, A4 neplyne A3
    4. To je správná odpověď na tento příklad!!!
  + 3NF: A1->A3?, A3->A1? => není v 3NF, A1->A3 platí
  + Vždy se normy vztahují ke klíči – proto může někomu vyjít, že je v NF, někomu jinému že není
  + 1) to, jestli je splněna 2Nf je určeno tím, jak zvolím klíč. Jednou vyjde, že ano, jednou že ne – to nevadí, protože zkoumáme až do 3NF a vždy to spravíme tak, že:
  + R1(A1,A3)
  + Vytáhnu fční vztah a zbude:
  + Ad1) R2(A1,A2, A4); ad2) R2(A1, A2, A4)
  + Dotáhnout tento příklad do konce znamená zkoumat, zda R2 je v 3NF
* Při návrhu databáze dbáme, aby všechny relace splňovaly 3NF
* Ale i to musíme opatrně – kdyby vyšlo, že celá databáze je složena z tabulek o 2 sloupcích, tak 2 sloupcová tabulka splňuje vždy 3NF – ale rovnou si řekněme, že návrh stagu kde by byly všechny tabulky 2 sloupečkové je na nic, protože by to trvalo týdny a týdny než by to spočetlo – zkušený návrhář pozná, kdy se má na tu 3NF vykašlat a pořešit to nějak jinak

## Boyce-Coddova Normální forma

* Relace R se nachází v Boyce-Coddově NF (BCNF) jestliže pro každou funkční závislost tvaru X->Y, kde Y není v X, platí, že X je nadmnožina nějakého klíče nebo je X v R klíčem.
* V podstatě nerozlišuje v závislosti, jestli do toho zahrnu neklíčové nebo klíčové atributy – tady neříkám, že X musí být tím klíčem, tady jen říkám, že to X by mohlo být klíčem
* Tady pojem klíč NENÍ PRIMÁRNÍ KLÍČ, ALE NĚCO, CO MI URČÍ, ŽE NA ZÁKLADĚ TĚCH HODNOT UMÍM ŘÍCT, O JAKOU SE JEDNÁ ENTITU
* Žádný příklad, který ukazuje, že relace je ve 3NF a není v BCNF není korektní
* Rychlík samozřejmě vymyslel příklad, který je v pořádku a ten si tu uvedeme:
* Př.
  + Zkoušení(Student, datum, čas, učitel, místnost)
  + Předpoklady:
    1. Čas je chápán jako kvantová (8:00, 8:15…)
    2. Student jde maximálně na jednu zkoušku v jeden den
    3. Daný učitel si v daný den objedná místnost jen jednu
  + Zkusíme najít klíč
    1. Primární klíč zvolíme: Student a Datum
  + Př. By měl ukázat, že tato relace je v 3NF a není v BCNF
  + Je ve 2NF? Zkoumám, zda nějaký atribut S->{Č, U, M}, D->{Č, UM, M}
    1. Ano, je v 2NF – datum i student se tam objevují vícekrát v různých kombinacích
  + 3NF?
    1. Hledám Č->U, Č->M, U-.Č, U->M, M->Č, M->U? Je v 3NF
  + BCNF?
    1. Jdeme hledat funkční závislosti – např. Datum, čas a učitel může určit místnost
    2. Ano, fční závislost platí a D, č, u může být klíčem
    3. DČM->S – ano, může být klíčem – určí jednu řádku – nebavím se ale o primárním klíči
    4. DČS->M – významem totéž
    5. DU->M – tato fční závislost platí
       1. Nalezli jsme fční závislost, kde determinant nemůže být klíčem
       2. Máme tabulku, která nesplňuje BCNF
       3. Jak to spravit?
          1. Co s tím? Rozdělit…
          2. VĚTŠINOU SE DOPORUČUJE (pro profíky) NF DOTÁHNOUT AŽ DO TÉ BCNF – TAM SE MI VYLOUČÍ, ŽE MÁM ŠPATNĚ NAVRŽENÝ KLÍČ
          3. BCNF bohužel nepoznáme na první pohled…
* Př. Který nesplňuje BCNF

|  |  |  |  |
| --- | --- | --- | --- |
| A1 | A2 | A3 | A4 |
| a | 1 | A | alfa |
| b | 1 | A | beta |
| c | 2 | B | gama |
| a | 1 | C | Delta |
| b | 3 | D | Alfa |

A2A4 může být primárním klíčem, ale pak nalezneme závislost A3->A2. Tato závislost nebyla zkoumána pro druhou ani třetí NF, protože A2 je součástí klíče. Tahle závislost nám to kazí…

Př. (J. Pokorný)

* Adresář(Město, Ulice, PSČ)
* Tvrdí, že tento příklad je v 3NF a není v BCNF, protože město,ulice->PSČ
* Ale tvrdí, že PSČ->město
* Příklad do pranice – musím určit co to je město, co to je ulice, neexistuje ulice co má tři PSČ?
* Příklad je v pořádku, ale je třeba dohledat tyto tři pojmy
* Co to je ulice? Nějaký registr, každá ulice má své id – to bychom rádi chápali jako pojem „ulice“ – ale většinou je klíš jméno ulice, např. „plzeňská“ a ta se pak může opakovat ve více městech – kdyby ulice byla jednotná v rámci celé republiky, pak jen ze znalosti ulice umím určit město
  + A) id: U->M (nemůže platit)
  + B) název U-/>M (rozumíme tedy pod pojmem ulice její název)
* No a pak ještě nějaké další nesrovnalosti, bylo toho moc

## Dekompozice

* To, abychom převedli relaci do normální formy tak vždy na závěr vyžadovalo nějakou dekompozici původní relace na více jiných
* Dekompozici provedeme na základě závislostí, které kazí tu formu
* Očekáváme od správně provedené dekompozice dvě vlastnosti:
  + 1) výsledná schémata by měla obsahovat stejná data jako původní relace
    1. Tzv. Bezztrátovost
  + 2) výsledná schémata by měla mít stejnou sémantiku (stejné funkční závislosti)
* Věta:
  + Mějme schéma R(A,B,C) „A, B, C jsou množiny atributů“, a s funkční závislostí B->C
  + Rozložíme-li R na R1(A,B) a R2(B, C) je takto provedená dekompozice bezztrátová.
  + Pozn.
    1. Důkaz, že data jsou stejná – pro R musí platit, že R1 \* R2 = R
  + Př.
    1. Exemplář(ISBN, inv\_číslo, datum\_nákupu, cena, země\_vydání)
    2. ISBN->{Cena, země\_vydání}
    3. R1(ISBN, cena, Země vydání) a R2(Inv\_číslo, cena) – dostanu úplně špatný rozklad – závěr\_ odporuje to té větě -> nesmysl

# 4. Normální forma

* zabývá se bohatším vztahem než je funkční závislost – tzv. multizávislostí
* Řekneme, že atribut Y multizávisí na atributu X (značí se: X ->-> Y), jestliže každé hodnotě atributu z množiny X přísluší množina hodnot atributu Y
* Pozn. Pokud by množina měla jeden prvek, jednalo by se o klasickou závislost
* To, co nám vadí, je pokud se vyskytují dvě multizávislosti
* Př.
  + Značka auta Země výroby
  + Škoda ČR
  + Škoda SR
  + Ford USA
  + Ford Kanada
  + Ford Německo
  + Výše uvedené je ukázka toho, že je to multizávislost
  + Značka auta Země výroby Motor
  + Škoda ČR Benzín
  + Škoda SR Benzín
  + Škoda ČR Nafta
  + Škoda SR Nafta
  + Ford USA
  + Ford Kanada
  + Ford Německo
  + Příklad ukazuje, že když jsou v jedné relaci dvě multifunkční závislosti, tak je to špatně – pozn. To se ale změní, pokud např. tu škodovku by na Slovensku nedělali s nafťákem
  + Multizávislost A ->->B v R(omega) se nazývá triviální,
  + Jestliže a) B je podmnožinou A
  + B) A jsednoceno s B je to omega
  + A nazývá se netriviální, jestliže ani ad a) ani ad b) není splněno.
* Relace je v 4NF, jestliže je v BCNF a neobsahuje žádnou netriviální multizávislost.
* Pozn. Obdoba pravidel pro funkční závislosti:
  + 1) doplnění
    1. Když X->->Y v R(omega), pak X->->omega\XY
  + 2) tranzitivnost
    1. Když X->->Y a Y->->Z, pak X->->Z\Y

# 5. Normální forma

* 5NF operuje s pojmem závislost podle spojení
* Mám relaci R, provedu její dekompozici na nějaké R1, R2, R3 – pokud bychom to udělali správně a ta dekompozice bude bezztrátová, pak bude platit: R = R1\*R2\*R3
* Závislost podle spojení je jev, že spojení relací získaných nějakou dekompozicí z původní relace R, dostanu opět výchozí relaci R.
* Relace se nachází v 5NF, jestliže je ve 4NF a neobsahuje závislost podle spojení.
* Př.
  + Relace, která není v 5NF
  + R1(A, B, C)

|  |  |  |  |
| --- | --- | --- | --- |
| P1 | D1 | A |  |
| P1 | D1 | B |  |
| P2 | D2 | A |  |
| P2 | D2 | B |  |
| P3 | D1 | A |  |
| P3 | D1 | B |  |
| P3 | D2 | A |  |
| P3 | D2 | B |  |

Projekce

* R2(A, B) R3(B, C) R4(A, C)
* P1 D1 D1A P1 A
* P2 D1 D1B P1 B
* P3 D1 D2A P2 A
* P3 D2 D2B P2 B
  + - * 1. P3 A
        2. P3 B
* R1 = R2 + R3 + R4
* Závěr: chceme relace v 5NF – nechceme, aby tam byla schována závislost podle spojení

# Integritní omezení

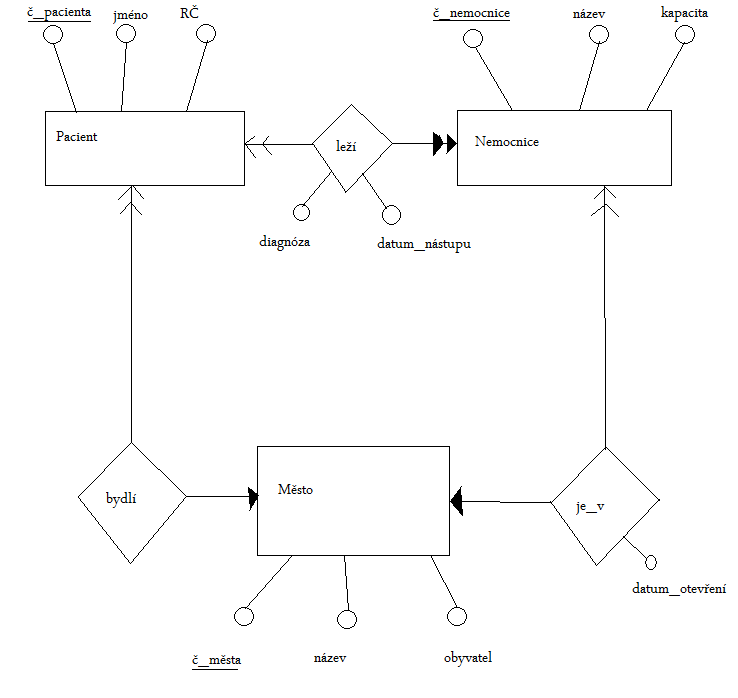
* Omezuje hodnoty dat v databázi
* Jsou trojího typu
  + Entitní integritní omezení – říká, že každá entita je v té databázi jen jednou (nejsou dvě stejné řádky v tabulce) – zajistí se určením primárního klíče
    1. Jaké vlastnosti má primární klíč? – musí být unikátní a musí být vyplněn (nesmí být null)
  + Doménové integritní omezení – doména je v podstatě obor hodnot, které může nabývat daný atribut – doménová omezení pak kladou nějaká omezení na tu množinu hodnot – např. plat musí být větší než nula, známky (1, 2, 3, 4)
  + Referenční integritní omezení – popisuje vztah mezi entitami – v podstatě navazuje na to, co jsme navrhli v ERA modelu – zejména to, co hlídá, je povinnost výskytu a možnost toho přiřazení
    1. Př. Čtenář a jeho výpůjčky – nemohu půjčit knihu někomu, kdo neexistuje
    2. Co mi může porušit tu referenční integritu? Které operace se musí hlídat, aby byla referenční integrita zachována? Jestliže zapisuji entitu s cizím klíčem, musí být jeho hodnota z primární tabulky. Jestliže mažu čtenáře, který má nějaké výpůjčky, pak by mi vznikly výpůjčky, které by patřili čtenáři, který již neexistuje.
    3. Tyto dvě operace se musejí hlídat a řešit – 1) restriktivní způsob – hláška, akce se neprovede 2) změna se projeví kaskádně – např. smažu čtenáře, který má výpůjčky, pak se smažou i všechny jeho výpůjčky (související záznamy) 3) dosazení null – podstatné je to, že dneska se všechna tato integritní omezení se naprogramují na úrovni Data definitiv Language

# SQL

# Relační model dat = RMD

* Relační schéma (relace)
  + R(atr1, atr2, … , atrn)
* Transformace E-RA -> RMD:

## Př. Nemocnice (z minula)



### RMD:

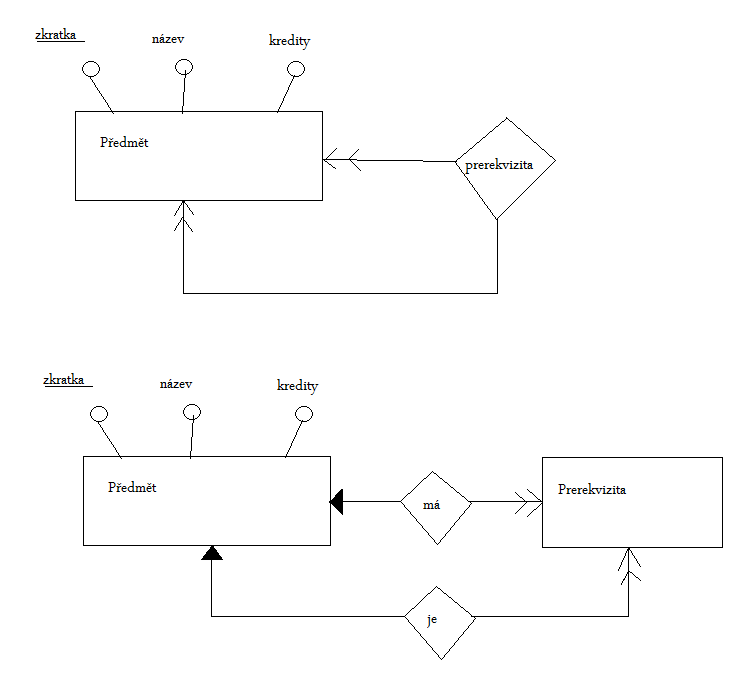
Pacienti(č\_pacienta, jméno, RČ, č\_města)

Nemocnice(č\_nemocnice, název, kapacita, č\_města, datum\_otevření)

Město(č\_města, název, obyvatel)

Leží(diagnóza, d\_nástupu, č\_pacienta, č\_nemocnice)

## Př. Prerekvizity



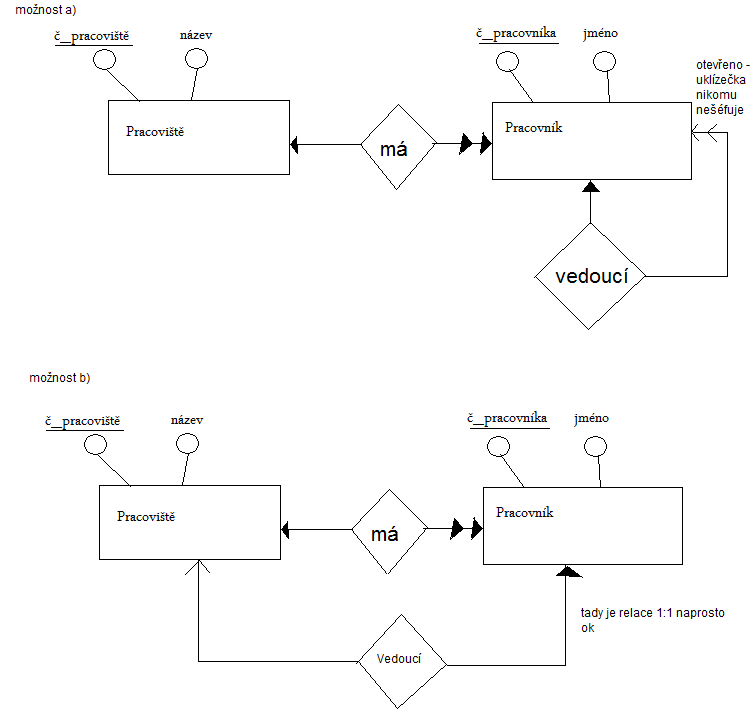
* Prerekvizita je slabá entitní množina, nemá žádné atributy, ale to je u slabé entitní množiny povoleno

### RMD:

Předměty(zkratka, název, kredity)

Prerekvizity(zkratka, zkratka\_prerekvizity)

## Př. Pracoviště



## Varianta A

### RMD

Pracoviště(č\_pracoviště, název)

Pracovníci(č\_pracovníka, jméno, č\_pracoviště, č\_vedoucího)

## Varianta B

### RMD

1)

Pracoviště(č\_pracoviště, název, č\_vedoucího)

Pracovníci(č\_pracovníka, jméno, č\_pracoviště)

2)

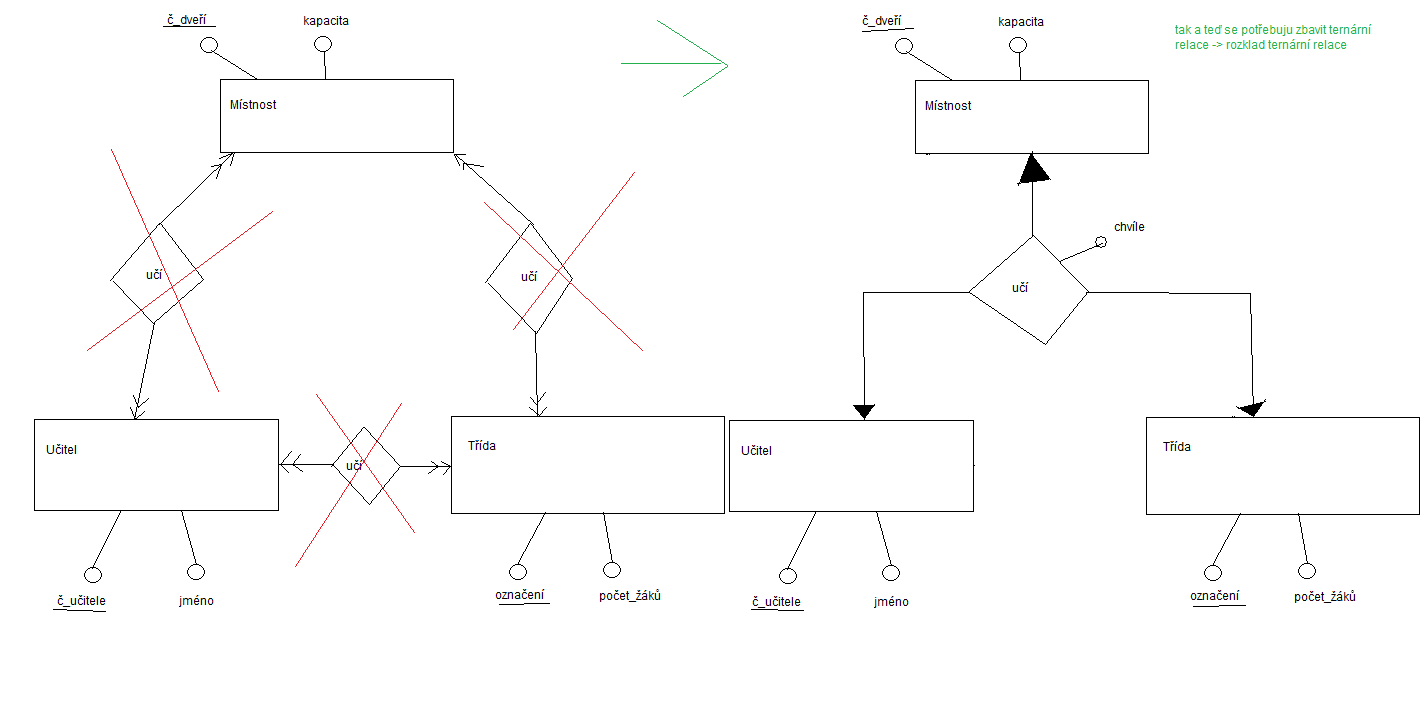
~~Pracoviště(č\_pracoviště, název)~~

~~Pracovníci(č\_pracovníka, jméno, č\_pracoviště, č\_vedeného\_pracoviště)~~

Jak toho docílit obecně? Ten příslušný cizí klíč se připíše do entitní množiny příslušného relačního schématu tam, kam směřuje ta otevřená šipka.

Ve verzi dva by to bylo moc řídké, spoust pracovníků by ten atribut neměla vyplněný.

## Př. Rozvrh na ZŠ



## RMD:

Místnosti(č\_dveří, kapacita)

Učitelé(č\_učitele, jméno)

Třídy(označení, počet\_žáků)

Výuka(chvíle, č\_dveří, č\_učitele, označení)

# SŘBD

# Relační algebra

* Operace (unární):
  + Projekce R[atributy]
  + Selekce R(podmínka)
* Spojení (binární):
  + Přiřazení R\*S
  + Théta-spojení R[t1 théta t2]S – théta je relační operátor (=, <, >,…)
* Kartézský součin RxS
* Množinové:
  + Průnik
  + Sjednocení

Př.

## Skladové hospodářství

Výrobky(č\_výrobku, cena)

Sklady(č\_skladu, č\_výrobku, množství)

|  |  |  |
| --- | --- | --- |
| V | Č\_výrobku | Cena |
|  | 0205 | 105 |
|  | 3429 | 200 |
|  | 1234 | 300 |
|  | 0100 | 150 |

|  |  |  |  |
| --- | --- | --- | --- |
| S | Č\_skladu | č\_výrobku | množství |
|  | S2 | 1234 | 50 |
|  | S1 | 0100 | 25 |
|  | S1 | 1234 | 10 |
|  | S3 | 0205 | 5 |
|  | S2 | 5603 | 20 |

1. Seznam čísel uskladněných výrobků

S[č\_výrobku]

1234

0100

0205

5603

1. V jakém skladu a v jakém množství je uskladněn výrobek č. 1234?

(S(č\_výrobku = 1234))[č\_skladu, množství]

* „projekce ze selekce“

S2 50

S1 10

1. V\*S

Č\_výrobku cena č\_skladu množství

0205 105 S3 5

~~3429 200~~

1234 300 S2 50

1234 300 S1 10

0100 150 S1 25

- výrovek „5603“ nám ruší integritu databáze, ale 3429 je ok

Př. Nemocnice

P č\_pacienta RČ jméno č\_města

1 100 Jan 1

2 150 Václav 4  
 3 200 Petr 2

4 300 Martin 5

M č\_města název\_m obyvatel

1 Plzeň 300 000

2 Praha 1 000 000

3 Brno 400 000

4 Ostrava 500 000

5 Lhota 1 000

L č\_pacienta datum\_n č\_nemocnice

1 duben 2001 1

2 září 2000 2

3 Leden 1999 1

1 duben 2002 3

4 únor 1995 2

N č\_nemocnice název č\_města

2 1.ZN 2

2 2.FN 3

3 VN 3

1. Názvy měst obsahující více než 400 000 obyvatel.

M(obyvatel > 400 000)[název\_m]

Praha, Ostrava

1. Ke každému pacientovi zjistěte název města, ve kterém bydlí

(P\*M)[jméno, RČ, název\_m]

1. Kteří pacienti bydlí ve městě, ve kterém je alespoň jedna nemocnice?

Buď ((M\*N)[č\_města] \* P)[jméno, RČ]

Petr 200

A nebo (P\*N)[jméno, RČ]

1. Kteří pacienti leží v nemocnici, která je postavena ve stejném městě, jako ve kterém pacient bydlí.

(((P[č\_pacienta, č\_města] průnik (L\*N)[č\_pacienta, č\_města])[č\_pacienta])\*P)[jméno, RČ]

1 1 1 2

2 4 2 2

3 2 3 2

4 5 1 3

4 2

((L\*N)[č\_p, č\_m]\*p)[jméno, RČ)

# Transakční zpracování

* Slouží k zachování integrity báze dat
* Integrita databáze se může porušit: operátor zadá špatné hodnoty, selhání hardwaru…
* Obrana: zálohování
* Jediné, co zbylo z ochrany na úrovni systému je pojem – tzv. kontrolní bod – kompletní záloha databáze v daném časovém okamžiku
* Pojem transakce je převzat z účetnictví – jedná se o jednu logickou operaci, která je realizována dvěma či více elementárními operacemi
* Transakce je logická jednotka práce, základní jednotka pro zotavení systému po chybě
* Během provádění transakce je databáze v nekonzistentním stavu
* Transakce se vždy nachází v jednom z následujících stavů:
  + PC – částečně potvrzena – stav, kdy proběhly všechny elementární operace, transakce je již hotova, ale ještě nebyla potvrzena
  + C- potvrzena (commit)
  + Přerušení obsluhou
  + F - chyba
  + Ab – abort
* Ochrana transakcí
  + Žurnál je soubor, do kterého se ukládá popis transakcí – s jakým objektem pracuje, nová hodnota, někdy i stará hodnota, kdo to pustil, kdy to pustil…
* Dopředné čtení
  + Vychází z kontrolního bodu – zapisuju žurnál dopředu a zapisuju hodnoty, které mám v žurnálu uloženy až do okamžiku té chyby
* Zpětné čtení
  + „Rollback“, „Undo“ – vychází se v okamžiku chyby a ukládají se staré hodnoty
  + Vyžaduje staré hodnoty
  + Hotová transakce mi nezajímá, zajímají mne jen ty rozpracované

Příští a ten další týden odevzdání semestrálních prací!!!

# Funkční závislost atributů

* Existuje i vztah mezi atributy dané entitní množiny
* R(A, B) A->B
* Př.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Předmět | Učitel | Kroužek | Místnost | Čas |
| Matematika | Novák | A4-01 | PC406 | Út 3 |
| Matematika | Novák | A4-01 | UL411 | Po 9 |
| Matematika | Novák | E3-15 | PC406 | Út 3 |
| Matematika | Novák | E3-15 | UL411 | Po 9 |
| Dějepis | Suková | S1-07 | AM701 | St 5 |
| Dějepis | Suková | A4-01 | AM701 | St 5 |
| Dějepis | Suková | S1-87 | AM701 | St 5 |
| Dějepis | Suková | K3-05 | AM701 | St 5 |

* Jeden učitel učí jen jeden předmět

P->U

U->P

{M, U}->U a z tranzitivity (Armstrongova pravidla) {M, U}->P

{K, Č}->M

{K, Č}->U

{K, Č}->P

* Aplikuji Armstrongova pravidla: {K, Č}->{M, U, P}

# Normální formy

VÝUKA(č\_učitele, student, předmět, zařazení\_učitele, pracoviště\_učitele, budova\_pracoviště, ROZVRH)

ROZVRH(den, čas\_od)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **č\_učitele** | **Předmět** | **Student** | Zařazeni | Pracoviště | Budova | ROZVRH |
| U1 | DB1 | A97222 | Asistent | KIV | UK | Út | 9:00 |
| U1 | Cpp | E99444 | Asistent | KIV | UK | Pá | 15:00 |
| U1 | Cpp | A97222 | Asistent | KIV | UK | Pá | 15:00 |
| U1 | DB1 | E99444 | Asistent | KIV | UK | Út | 9:00 |
| U555 | GRA | H96777 | Docent | KEE | PC | Čt | 9:00  Pá | 15:00 |

* Jedná se jen o fragment tabulky…
* Učitel učí více předmětů
* Jestliže student navštěvuje jeden učitelův předmět, musí navštěvovat všechny učitelovo další předměty

1. Dostat to do 1NF

VÝUKA(č\_učitele, student, předmět, zařazení\_učitele, pracoviště\_učitele, budova\_pracoviště, den, čas)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **č\_učitele** | **Předmět** | **Student** | Zařazeni | Pracoviště | Budova | **Den** | **Čas** |
| U1 | DB1 | A97222 | Asistent | KIV | UK | Út | 9:00 |
| U1 | Cpp | E99444 | Asistent | KIV | UK | Pá | 15:00 |
| U1 | Cpp | A97222 | Asistent | KIV | UK | Pá | 15:00 |
| U1 | DB1 | E99444 | Asistent | KIV | UK | Út | 9:00 |
| U555 | GRA | H96777 | Docent | KEE | PC | Čt | 9:00 |
| U555 | GRA | H96777 | Docent | KEE | PC | Pá | 15:00 |

* Nefungoval primární klíč, je nutné k primárnímu klíči přidat den i čas

1. Dostat do 2NF

VÝUKA(č\_učitele, student, předmět, den, čas)

UČITEL(č\_učitele, zařazení\_učitele, pracoviště\_učitele, budova\_pracoviště)

{č\_učitele}->{zařazení\_učitele, pracoviště\_učitele, budova\_pracoviště}

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **č\_učitele** | **Předmět** | **Student** | **Den** | **Čas** |
| U1 | DB1 | A97222 | Út | 9:00 |
| U1 | Cpp | E99444 | Pá | 15:00 |
| U1 | Cpp | A97222 | Pá | 15:00 |
| U1 | DB1 | E99444 | Út | 9:00 |
| U555 | GRA | H96777 | Čt | 9:00 |
| U555 | GRA | H96777 | Pá | 15:00 |

|  |  |  |  |
| --- | --- | --- | --- |
| **č\_učitele** | Zařazeni | Pracoviště | Budova |
| U1 | Asistent | KIV | UK |
| U555 | Docent | KEE | PC |

VÝUKA N:1 UČITEL

1. Dostat do 3NF

{pracoviště\_učitele}->{budova\_pracoviště}

„Každý neklíčový atribut nesmí tranzitivně záviset na jiném neklíčovém atributu relace“

VÝUKA(č\_učitele, student, předmět, den, čas)

UČITEL(č\_učitele, zařazení\_učitele, pracoviště\_učitele)

PRACOVIŠTĚ(pracoviště\_učitele, budova\_pracoviště)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **č\_učitele** | **Předmět** | **Student** | **Den** | **Čas** |
| U1 | DB1 | A97222 | Út | 9:00 |
| U1 | Cpp | E99444 | Pá | 15:00 |
| U1 | Cpp | A97222 | Pá | 15:00 |
| U1 | DB1 | E99444 | Út | 9:00 |
| U555 | GRA | H96777 | Čt | 9:00 |
| U555 | GRA | H96777 | Pá | 15:00 |

|  |  |  |
| --- | --- | --- |
| **č\_učitele** | Zařazeni | Pracoviště |
| U1 | Asistent | KIV |
| U555 | Docent | KEE |

|  |  |
| --- | --- |
| **Pracoviště** | Budova |
| KIV | UK |
| KEE | PC |

UČITEL N:1 PRACOVIŠTĚ

1. 4. Normální forma – jen pro zajímavost

X->->Y – tzv. multizávislost

„pro jakoukoli hodnotu množiny atributů X platí u funkční závislosti několik hodnot“

{č\_učitele}->->{student} – tato multizávislost vyplývá z podmínky, že když už student navštěvuje jeden učitelův předmět, musí navštěvovat všechny ostatní předměty

* Každý učitel učí stejnou množinu studentů v každý den a každý čas
* Student není podmnožinou učitele, po sjednocení nezískáme všechny atributy, takže tohle má tvořit klíč relace R, což netvoří => do 4Nf to provedeme tak, že provedeme rozklad

UČITEL\_učí\_PŘEDMĚTY(č\_učitele, předmět, den, čas\_od)

UČITEL\_učí\_STUDENTY(č\_učitele, student)

|  |  |  |  |
| --- | --- | --- | --- |
| **č\_učitele** | **Předmět** | **Den** | **Čas** |
| U1 | DB1 | Út | 9:00 |
| U1 | Cpp | Pá | 15:00 |
| U555 | GRA | Čt | 9:00 |
| U555 | GRA | Pá | 15:00 |

|  |  |
| --- | --- |
| **č\_učitele** | **Student** |
| U1 | A97222 |
| U1 | E99444 |
| U555 | H96777 |

|  |  |  |
| --- | --- | --- |
| **č\_učitele** | Zařazeni | Pracoviště |
| U1 | Asistent | KIV |
| U555 | Docent | KEE |

|  |  |
| --- | --- |
| **Pracoviště** | Budova |
| KIV | UK |
| KEE | PC |

UČITEL\_učí\_STUDENTY N:1 UČITEL

UČITEL\_učí\_PŘEDMĚTY N:1 UČITEL

1. BCNF: „Pro každou funkční závislost, kterou v množině atributů najdeme, platí: množina atributů X musí být nadmnožinou primárního klíče nebo primárním klíčem relace.“
   1. BCNF=>3NF
   2. 4NF=>BCNF
   3. Tzn. My jsme ve 4NF, takže splňujeme i BCNF

KONEC

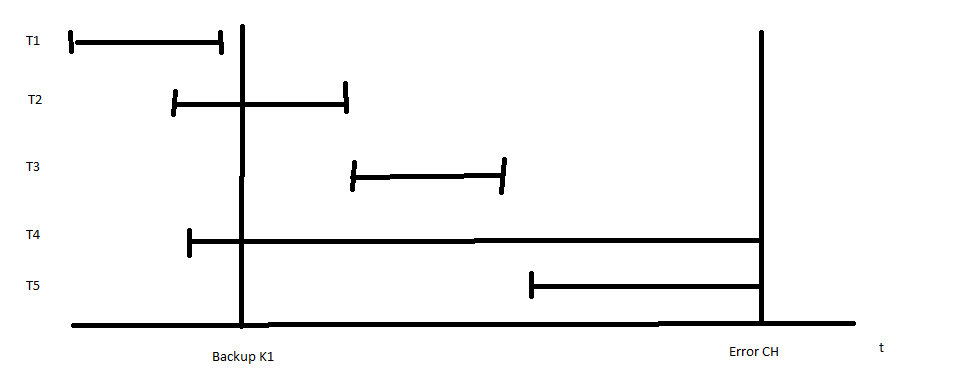
# Dvoufázové potvrzování

## Tvorba žurnálu s odloženou realizací změn

* Změny do databáze se zapisují, až když je transakce částečně potvrzena
* Transakce může být částečně potvrzena, až když jsou všechny změny (elementární operace) uloženy v žurnálu
* Musí zamknout objekty a může je odemknout, až ta transakce skončí

## Přímý zápis do databáze

* Zapisuje se současně do databáze i do žurnálu – do žurnálu musím psát staré hodnoty, protože v případě chyby použiju rollback.
* Může to zamykat a uvolňovat dle potřeby
* Pro běh je to rychlejší, ale pak musíme vrátit všechny transakce, které v té době (od špatné hodnoty) proběhly – v případě chyby nebo nedokončení transakce – tzv. dominový efekt
* Zde se stávají chybnými i ty transakce, které jen čtou – přečtou špatná data, je nutné je tedy ukládat, protože se pak stávají chybnými



## „Přímý zápis do databáze“

T1 – je ok, neděláme s ní nic

T2, T3 – když mám žurnál, tak se použije a dopředným čtením se může navodit, že ta transakce proběhla – vezmu nové hodnoty a to co bylo v kontrolním bodě, to přepíšu tím ze žurnálu

T4, T5 – použiju žurnál na ten rollback, tzn. Vrátím provedené změny

## Rollback algoritmus

1. Vytvoříme 2 seznamy – Redo, Undo
2. Do seznamu Undo zapíšu všechny transakce, které jsou rozpracovány v kontrolním bodě
   1. v příkladu jsou to T2 a T4
3. procházím časem, jakmile narazím na začátek transakce, zapíšu jí do seznamu undo
4. jakmile narazím na konec transakce, převedu transakci ze seznamu Undo do seznamu Redo
5. Transakce v seznamu Redo převedu do konzistentního stavu systémem redo a transakce v seznamu Undo převedu do konzistentního stavu systémem undo

# Paralelní běh transakcí

* Jak přijdeme na to, že paralelní běh transakcí dopadl dobře? => Uspořádatelnost – Paralelní běh transakcí je uspořádatelný, jestliže výsledek paralelního běhu je stejný, jako jakýkoli sériový běh

T1, T2: A <- read(x)

A <- A+5

Write(X, A)

* X je společné všem transakcím – objekt v databázi, A je vlastní pouze dané transakci



Plány 1 a 3 jsou chybné, protože nezajišťují uspořádatelnost

Cestou, jak uspořádatelnost zajistit, je používání zámků.

# Uzamykací protokoly

* Operace LOCK(x)
* Objekt může být uzamčen pouze jednou transakcí
* Pokud chce s uzamčeným objektem pracovat jiná transakce, je pozastavena
* Objekt může být odemknut jen tou transakcí, která ho zamkla
* Pokud existují pozastavené transakce, mohou se spustit

Př. Dvě transakce- každá bude mít jiný obsah

X=1, Y=1

T1:

LOCK(x)

LOCK(y)

A <- read(x)

B <- read(y)

B <- A+B

A <- A-B

Write(Y, B)

UNLOCK(Y)

Write(X, A)

UNLOCK(X)

T2:

LOCK(Y)

B <- read(Y)

B <- 2\*B

Write(Y, B)

UNLOCK(Y)

LOCK(X)

A <- read(X)

A <- A+1

Write(X, A)

UNLOCK(X)

Plán 1: T1, T2 -> po T1: x =-1, y=2, po T2: x=0, y=4

Plán 2: T2, T1 -> po T2: x=2, y=2, po T1: x=-2, y=4

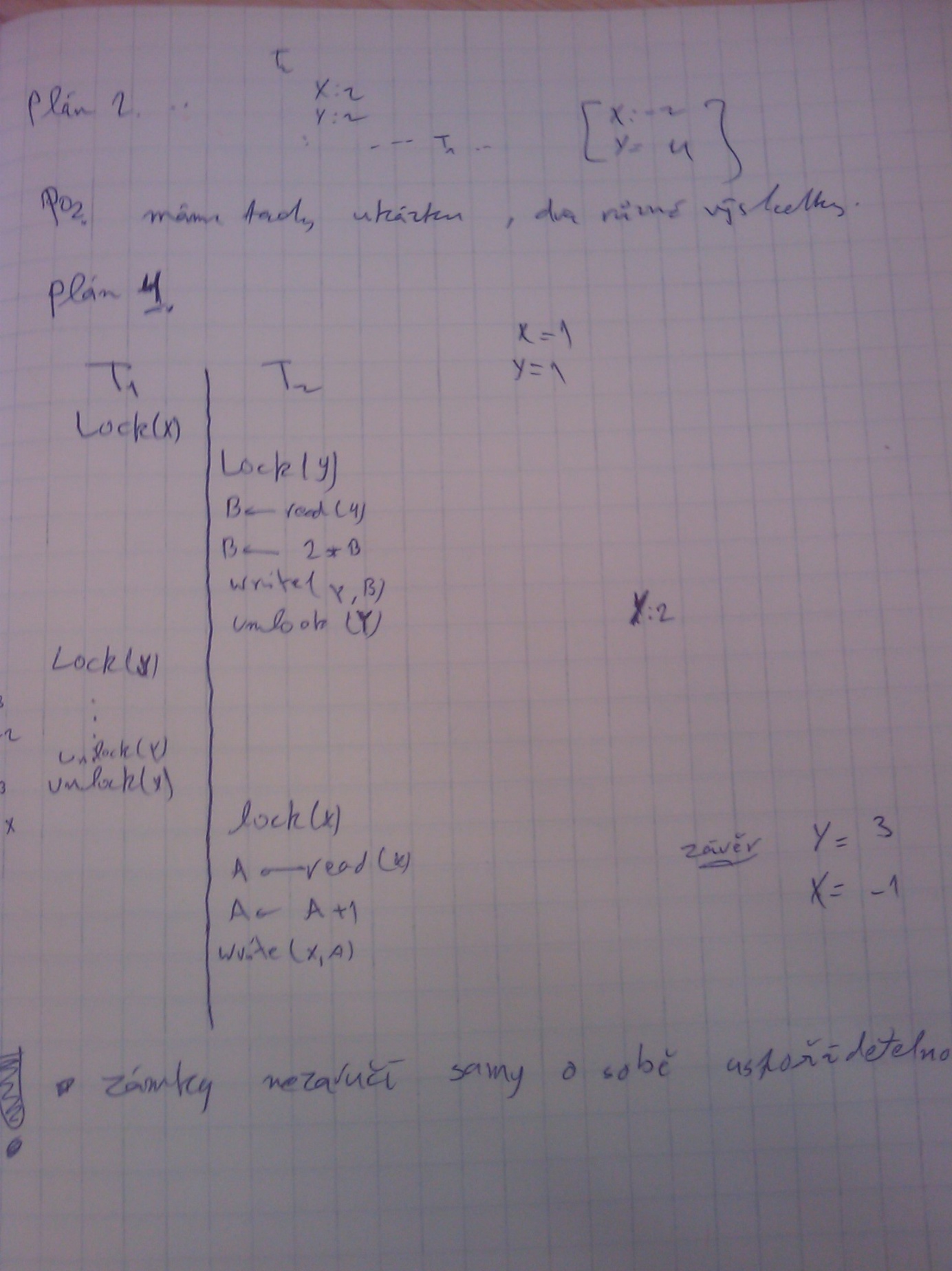
* Mimo jiné máme ukázku, že máme
* dva různé výsledky – to je v pořádku . prostě jak to pouští, tak to je

Plán 4:

Zámky samy o sobě nezaručí uspořádatelnost.

Objekt může být v každém okamžiku uzamčen pouze pro jednu transakci.

JAKMILE TRANSAKCE NĚJAKÝ OBJEKT ODEMKNE, NESMÍ UŽ NIC ZAMYKAT.



Transakce T2 je napsaná špatně, „UNLOCK“, „LOCK“

While(něco)

{

LOCK(x)

.

.

.

UNLOCK(x)

}

* toto vykazuje náhodné chyby – pak nám tam v tom cyklu někdo šáhne a bude to náhodně padat

správně:

LOCK(X)

While(něco)

{

.

.

.

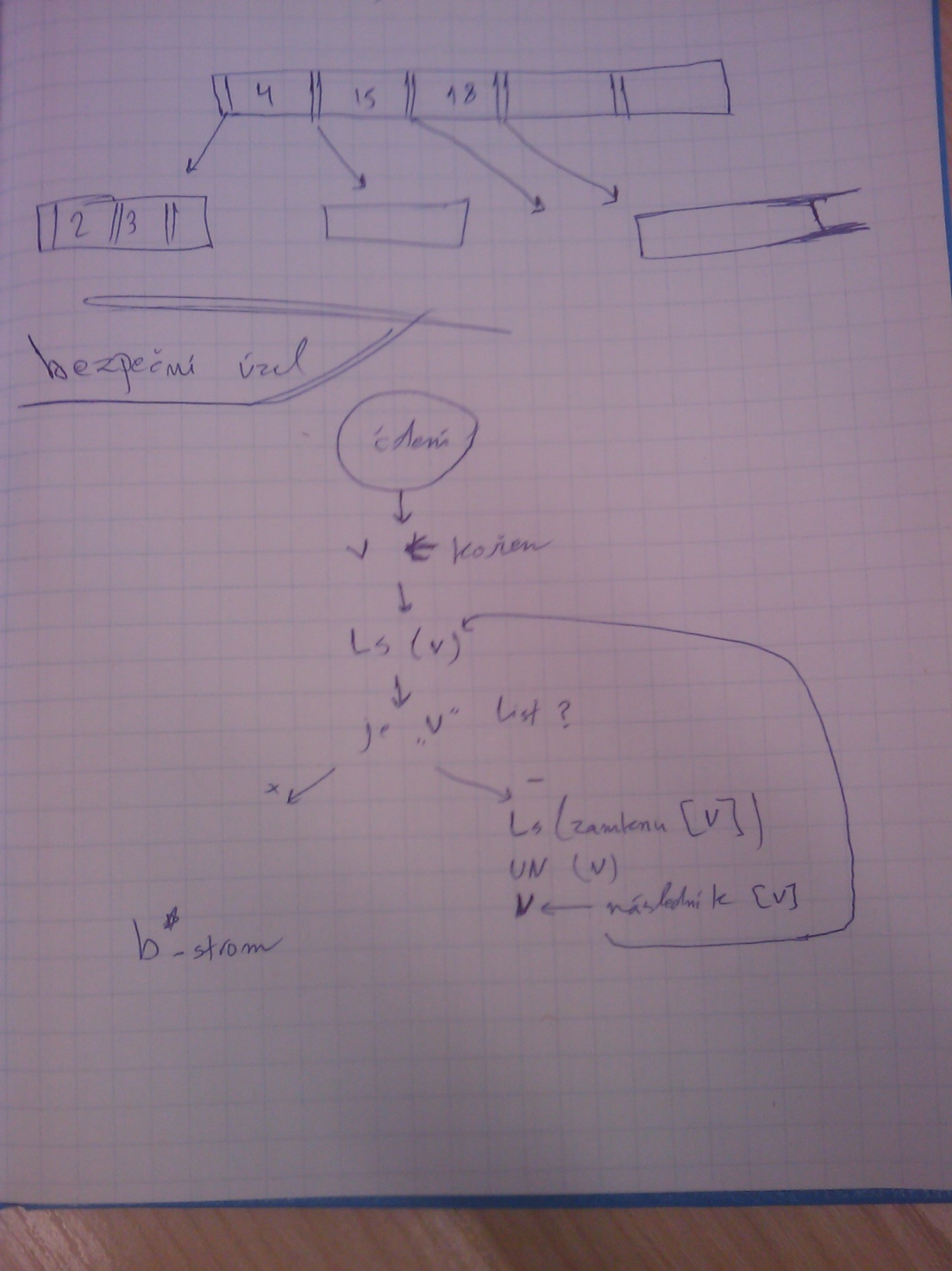
}

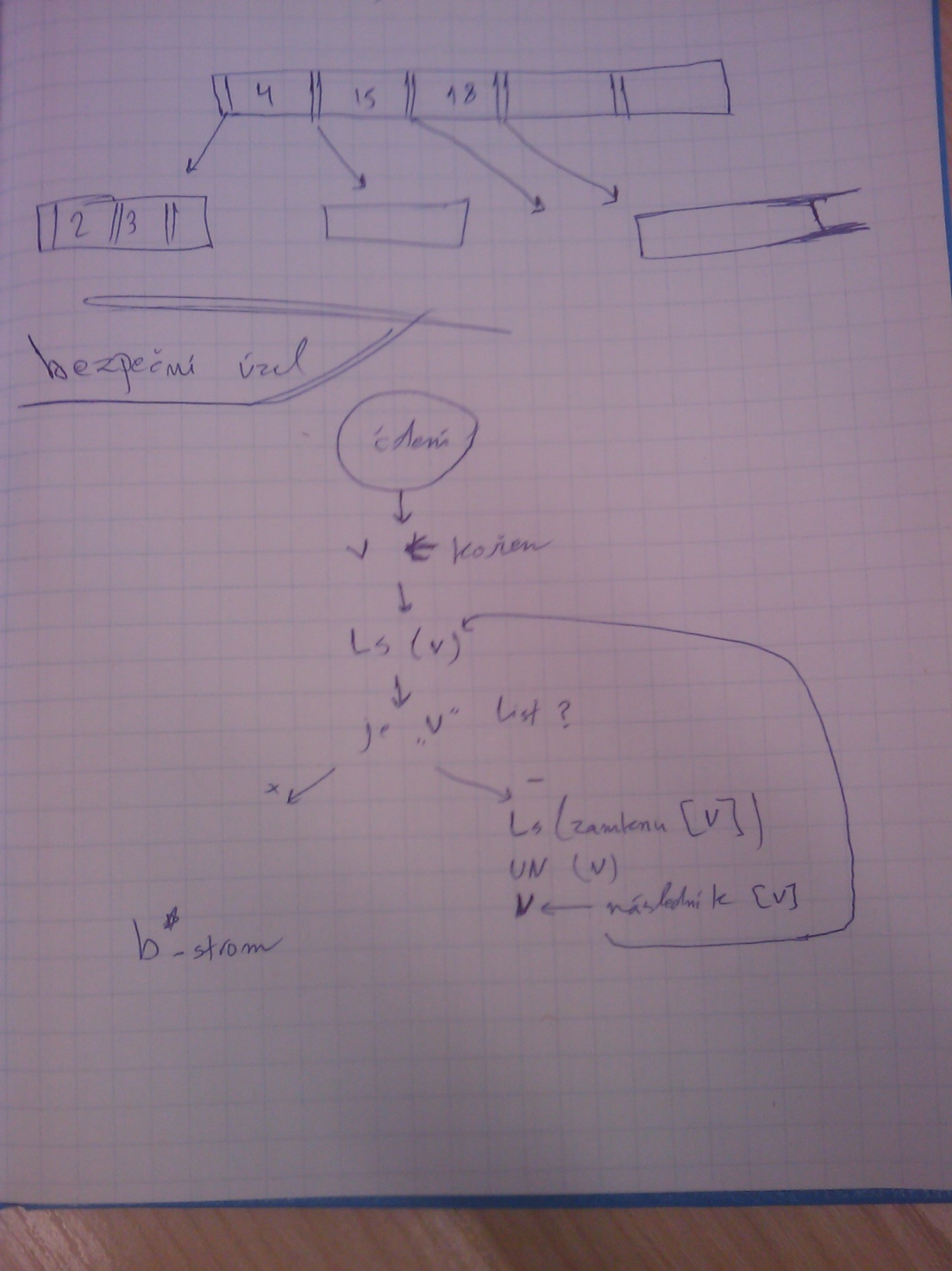
UNLOCK(x)

* toto sice má dlouho aktivní zámek, ale nedá se nic dělat
* lze prokázat, že dodržení dvoufázového protokolu zaručí uspořádatelnost
* může ale dojít k další nepříjemnosti, kterou je „Uváznutí“ – kdybychom přehodili Unlock(y) s lock(x) v T2, tak T4 uvízne -> Deadlock
* díky žurnálu můžeme transakce seknout v případě dreadlocku a pustit je znovu – když se znovu seknou, opět je seknu a pustím znovu – to seknutí je náhodný jev – v SŘBD lze deadlock vyřešit tím, že jednu transakci zrušíme
* díky žurnálu to můžem udělat
* někdy se zavádí pojem „výlučný zámek“ a „sdílený zámek“ – sdílený zámek dovolí ostatním transakcím číst

### Jiné uzamykací protokoly

* nemusíme vždy používat dvoufázový protokol, ale musíme znát interní uspořádání dat
* to v praxi ale neznáme, takže dvoufázový protokol dodržovat musíme – ale v případě, kdy víme, že databáze je hierarchická – uspořádaná do stromu – to dodržovat nemusíme
* Protokol pro hierarchickou databázi
  + 1. Výlučný zámek transakce T lze použít na jakýkoli objekt
  + Další objekt může být uzamčen transakcí T pouze když byl v t uzamčen jeho předchůdce
  + Objekty mohou být odemknuty kdykoli
  + Objekt, který byl transakcí T zamknut a odemknut, nesmí být znovu v T zamknut
  + Všechny plány podle tohoto protokolu jsou uspořádatelné a navíc nemůže vzniknout dreadlock
* V uspořádání není klasická hierarchie, ale je to B-strom
  + Interní organizace dat je ve všech sřbd podle b-stromů
  + B-strom má tu výhodu, že nemůže zdegenerovat – má stejnou výšku
  + Když ale vkládám novou hodnotu a ten uzel je plnej a on se mi rozpadne na dva, pak musím zařídit to rozpadnutí na dva pomocí nějakého algoritmu . když se mi anopak zmenší počet prvků na polovinu, musím naopak dva uzly sloučit
  + „bezpečný uzel“ – uzel, se kterým nemusím dělat takové harakiri že ho rozpojuju a spojuju
  + Pozn. V databázích to není čistý b-strom, ale b\*-strom, kde jsou data až v listech





Srovnání uzamykacích protokolů – hodnotím počet zámků v danou dobu a na jak dlouhou dobu se zamyká.¨

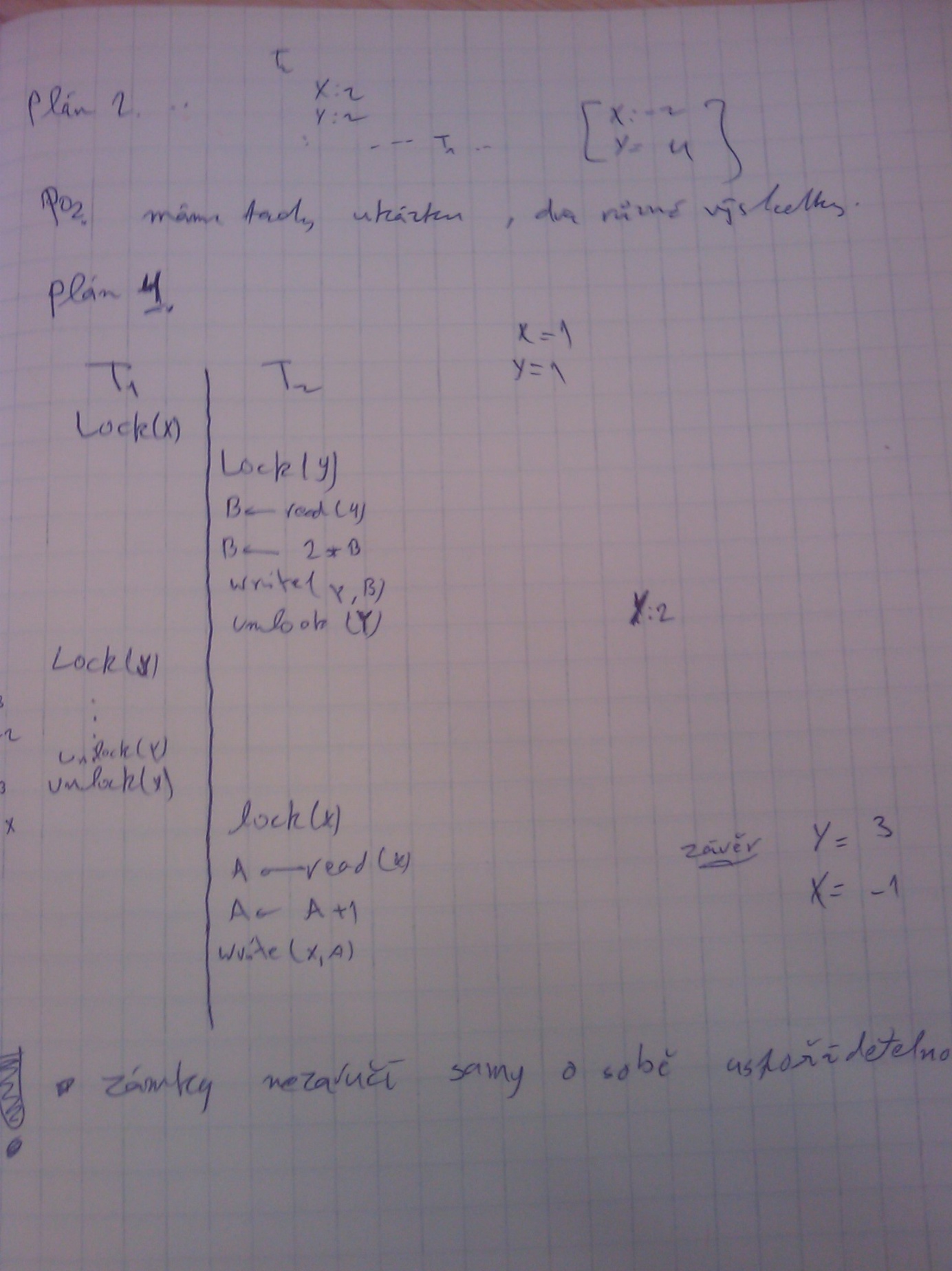
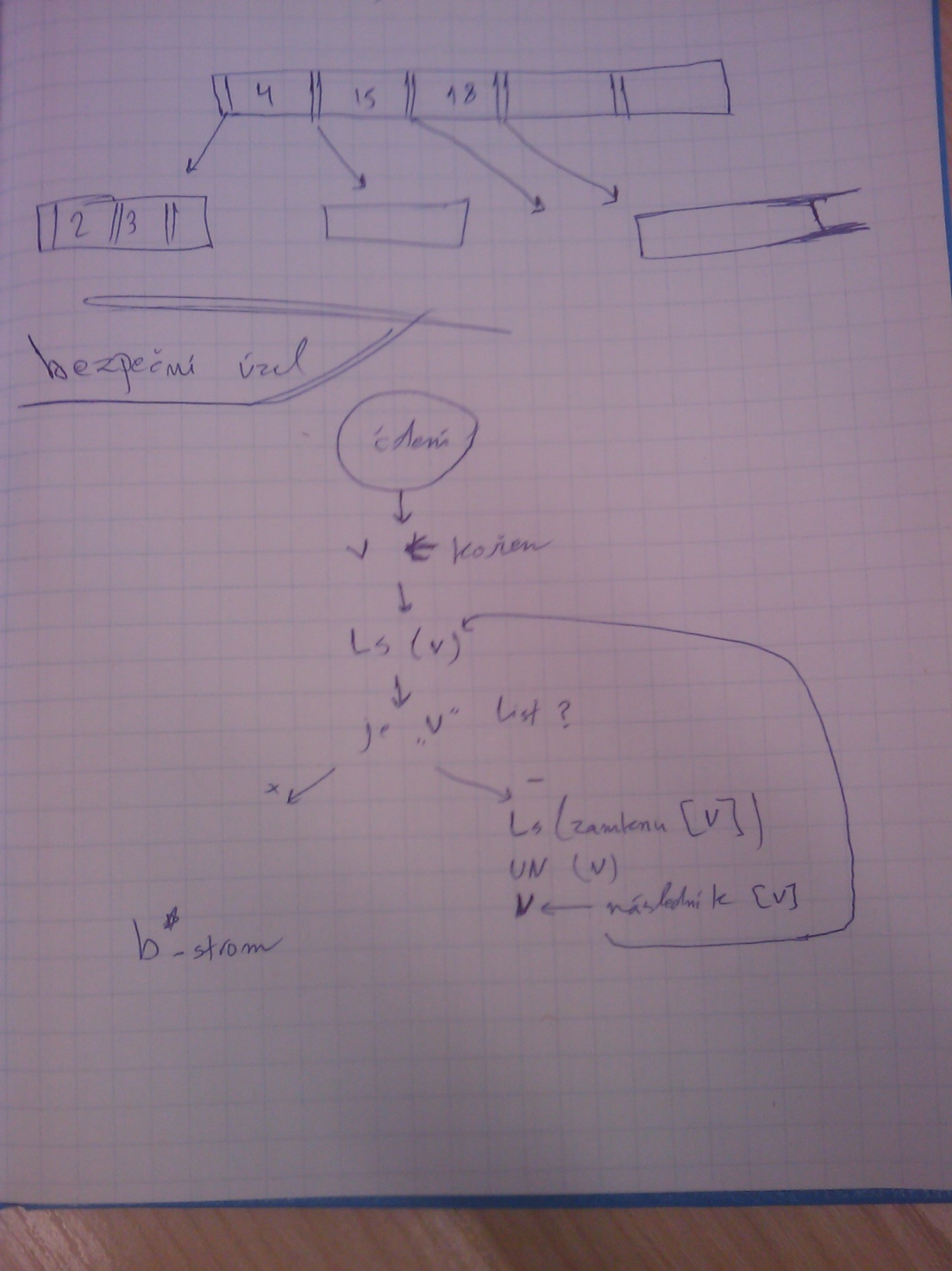
Stromový protokol uzamyká více zámků na výrazně kratší dobu. Co je rychlejší to nikdo neví. Poslední důležitá věc v tomto semestru:

# Časové značky

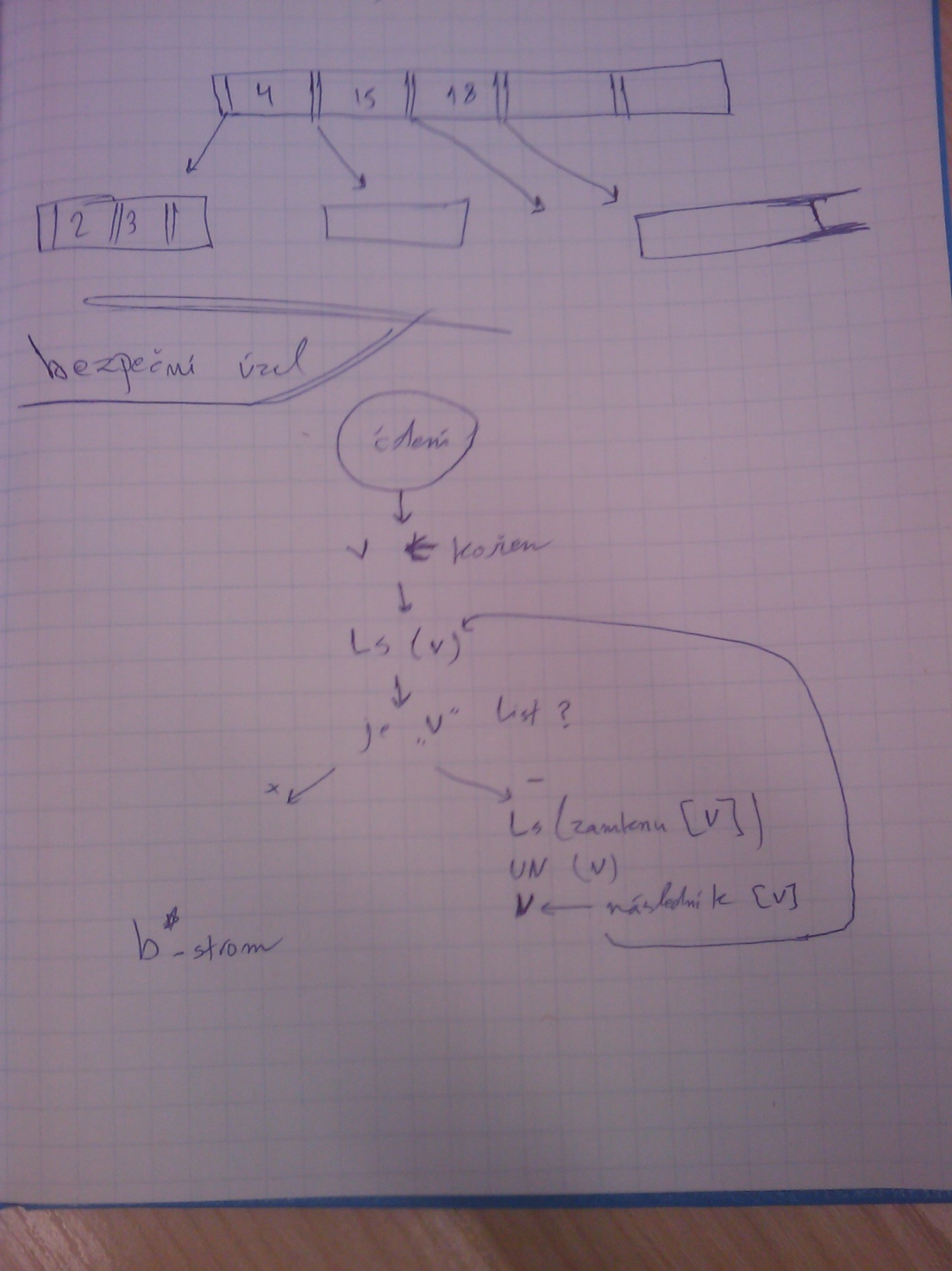
* Existují protokoly, které nepoužívají zámky a přesto zajistí uspořádatelnost – používají tzv. časové značky (např. sřbd Interbase)
* Princip: u každého objektu zapisujeme dvě časové značky – čas posledního čtení a čas posledního zápisu

## Základní plánovač

Tyhle obrázky umět ke zkoušce včetně dvoufázového protokolu!!!



## Thomasův plánovač – zajímavost

* V tomhle případě se plánovač tváří, že operace proběhla, ale ve skutečnosti se nezapíše nic
* 

Příště databáze nejsou!!!