Vysoká škola polytechnická Jihlava

Databázové systémy 1

Studijní opora

Ing. Zbyněk Bureš. Ph.D.

Zbyněk Bureš

DATABÁZOVÉ SYSTÉMY 1

1. vydání
ISBN 978-80-87035-88-7

Vydala Vysoká škola polytechnická Jihlava, Tolstého 16, Jihlava, 2014

Tisk Ediční oddělení VŠPJ, Tolstého 16, Jihlava

Úvod

Předmět je úvodem do databázových systémů. Zabývá se metodikou návrhu databáze a to na úrovni konceptuální, logické (relační) a implementační. Věnuje se jazyku SQL, návrhu a normalizaci relačního schématu. Předmět částečně souvisí s předmětem Objektové modelování. Na předmět přímo navazuje předmět Databázové systémy 2. Pro studium předmětu Databázové systémy 1 nejsou nutné žádné specifické znalosti, vhodné jsou však základy teorie množin a základy programování.

Studijní opora je členěna do kapitol, v nichž se student postupně seznámí s celou problematikou. V kapitolách je obsažen jednak teoretický a praktický výklad, jednak řada řešených a neřešených příkladů, které umožní studentovi procvičit si zevrubně všechny potřebné dovednosti.

Oblasti, které by měl student zvládnout:

- konceptuální analýza a modelování, ER diagram
- relační model dat a relační algebra
- transformace konceptuálního modelu do relačního modelu
- základy normalizace relačního modelu
- jazyk SQL - jazyk pro definici dat, jazyk pro manipulaci s daty, dotazování

1 Základy softwarového inženýrství

Cíl kapitoly

Kapitola přináší úvod do softwarového (SW) inženýrství, jehož je tvorba databázových systémů podmnožinou. Z hlediska SW inženýrství student lépe pochopí motivaci pro správný postup při tvorbě databází. Přednáška o SW inženýrství představuje také propojovací článek s dalšími předměty. Cílem je naučit se:

- proč se vůbec zavádí disciplína SW inženýrství
- jaký je správný metodický přístup při tvorbě softwaru
- jaký je životní cyklus softwaru
- jaké je místo databází v softwarových produktech, zejména v informačních systémech

Klíčové pojmy

Softwarové inženýrství, software, informační systém, životní cyklus softwaru.

Proč softwarové inženýrství

V začátcích počítačů byly programy tvořeny malými týmy či jednotlivci

- úzce specializované programy, obvykle vědecké a technické výpočty
- programovány v jazyce s malou úrovní abstrakce (strojový kód, assembler, nanejvýš Fortran)

S příchodem integrovaných obvodů se možnosti počítačů zvětšily (rychlejší, větší paměť) – potenciál pro rozvoj rozsáhlejších SW produktů (bankovních IS atp.)

Aplikace tradičních programovacích metod (víceméně ad hoc návrh a implementace) na větší systémy vedla k problémům:

- velké zpoždění při vývoji
- mnohonásobné zvýšení ceny oproti původním odhadům
- chybovost nového SW
- tzv. softwarová krise

Příčina: rozsáhlé projekty jsou zcela odlišné od malých, nelze přímo přenést zkušenosti. Mnoho práce totiž zabere

- naplánovat, jak rozdělit projekt do dílčích celků (modulů)
- specifikace činnosti a rozhraní modulů
- naplánovat interakce modulů

Až potom přichází vlastní „kódování“, tj. napsání a odladění samostatných modulů, sestavení modulů do výsledého celku a testování celku (moduly obvykle hned nespolupracují správně, ačkoli samostatně jsou v pořádku).

Odhad náročnosti prací

- 1/3 celkové práce je plánování
- 1/6 kódování
- 1/4 testování modulů
- 1/4 testování systému

Kódování je tedy vlastně to nejjednodušší, nejvíc práce dá správně vše naplánovat, rozdělit projekt na moduly, a navrhnout a zajistit bezchybnou interakci modulů.

Co je software (SW)

Často se míní, že software je pouze program, či kód. Software je ale nutno chápat komplexněji:

- programy
- dokumentace
    - systémová
    - uživatelská
- konfigurační data

Rozlišujeme dva typy produktů

- generický (vyvíjen na základě potřeb trhu, koupí si ho, kdo chce – textové editory, OS, apod)
- na zakázku

Co je softwarové inženýrství

SW inženýrství je aplikace inženýrských metod na vývoj SW.

- specifikace
- vývoj
- testování
- údržba
- management

Jiná definice: aplikace disciplinovaného, systematického, měřitelného přístupu na vývoj a údržbu SW.

Informační systém (IS)

IS je systém vzájemně propojených informací a procesů, které s informacemi pracují. Procesy jsou přitom nějaké funkce, transformující vstupy na výstupy. IS je jedním z hlavních faktorů efektivního řízení a konkurenceschopnosti podniku - závislost podniku na kvalitních a včasných informacích.

Životní cyklus IS

Životní cyklus IS lze definovat různě, každý autor uvádí různý počet fází, principielně jde o tyto fáze:

- předběžná analýza, tj. specifikace cílů
- analýza systému, tj. specifikace požadavků
- projektová studie, tj. návrh
- implementace
- testování
- zavádění systému
- zkušební provoz
- rutinní provoz a údržba
- reengineering

Platí, že většina chyb systému je způsobena špatným návrhem, nikoli špatnou implementací:

- programátoři naprogramují přesně to, co se jim řeklo, i když je to špatně
- čím později (ve vyšší fázi vývoje) se přijde na chybu, tím dražší je její náprava

Předběžná analýza, specifikace cílů

Cílem je pouze sestavit základní rámec požadavků, cílů a funkcí, ne je podrobněji rozebírat (to je úkolem další etapy).

- shromáždění požadavků uživatelů, cílů organizace
- odhad doby realizace a nákladů

Analýza systému, specifikace požadavků

Jde o rozbor předchozí části. Tato etapa má klíčovou důležitost: veškeré chyby ve struktuře dat i systému, které se zde neodhalí, jsou později velice obtížně odstranitelné.

Projektová studie, návrh

Jedná se o výsledek analýzy systému. Projektová studie obsahuje obvykle tyto podklady

- obsah smlouvy o návrhu a realizaci IS
- časový harmonogram
- cenu vyvíjeného projektu
- konkrétní implementaci systému (logický datový model, fyzický datový model…)
- podmínky zavádění do praxe (HW, SW, atd.)
- podmínky celkového předání IS
- záruční servis

Implementace

- vlastní programování
    - programátoři
    - zodpovědný analytik
- definice vstupů a výstupů
- naprogramování funkcí
- vyladění interakce funkcí
- ověření funkcí
- příprava co možná nejreprezentativnějších testovacích dat

Testování

Provádějí se připravené testy na hotovém IS. Je nutné vyzkoušet reakce systému na všechny možné vstupy a opravit případné chyby. Testování se často provádí mimo reálné prostředí (letectví, zdravotnictví atd.).

Zavádění systému

- instalace, uvedení do provozu
- předání dokumentace
- školení uživatelů
    - nejprve vedoucí a pak podřízené
    - nesmí se podcenit

Zkušební provoz

- povinnost okamžitého servisu
- odstraňování chyb
- dořešení dodatečných požadavků

Rutinní provoz a údržba

- závěrečná fáze projektu, provoz a používání systému
- zajištění optimálního provozu
- zabezpečení systému
- záloha dat

Reengineering

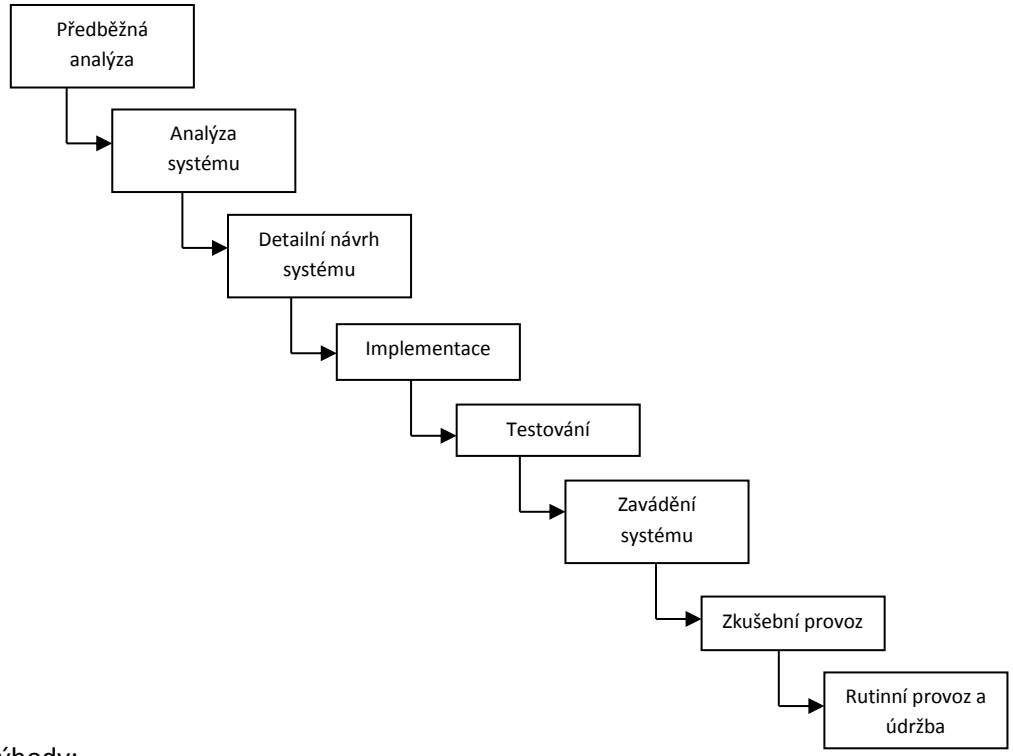
Postupem doby dochází k přehodnocení požadavků na systém. Pokud nelze nové požadavky uspokojit úpravou stávajícího systému, jdeme zase na začátek životního cyklu.

Modely životního cyklu IS

Etapy životního cyklu IS lze provádět různě, existuje několik modelů, které lze při tvorbě SW použít.

Model vodopád

Při návrhu se provádějí postupně jednotlivé etapy životního cyklu, které na sebe navazují a neprolínají se. Etapy se provádějí podle stanoveného plánu a nevracíme se k nim, dokončená etapa je vstupem etapy následující.



Výhody:

- rychlý
- levný, pokud se nevyskytnou problémy
- pevná struktura vývoje IS, šetření zdrojů

Nevýhody:

- skutečné projekty se v krocích definovaných vodopádem obvykle nedají řešit
- výsledek máme až po skončení poslední fáze vývoje, tj. chyba na začátku vede k nutnosti přepracovat vše
    - nesmírně drahé…

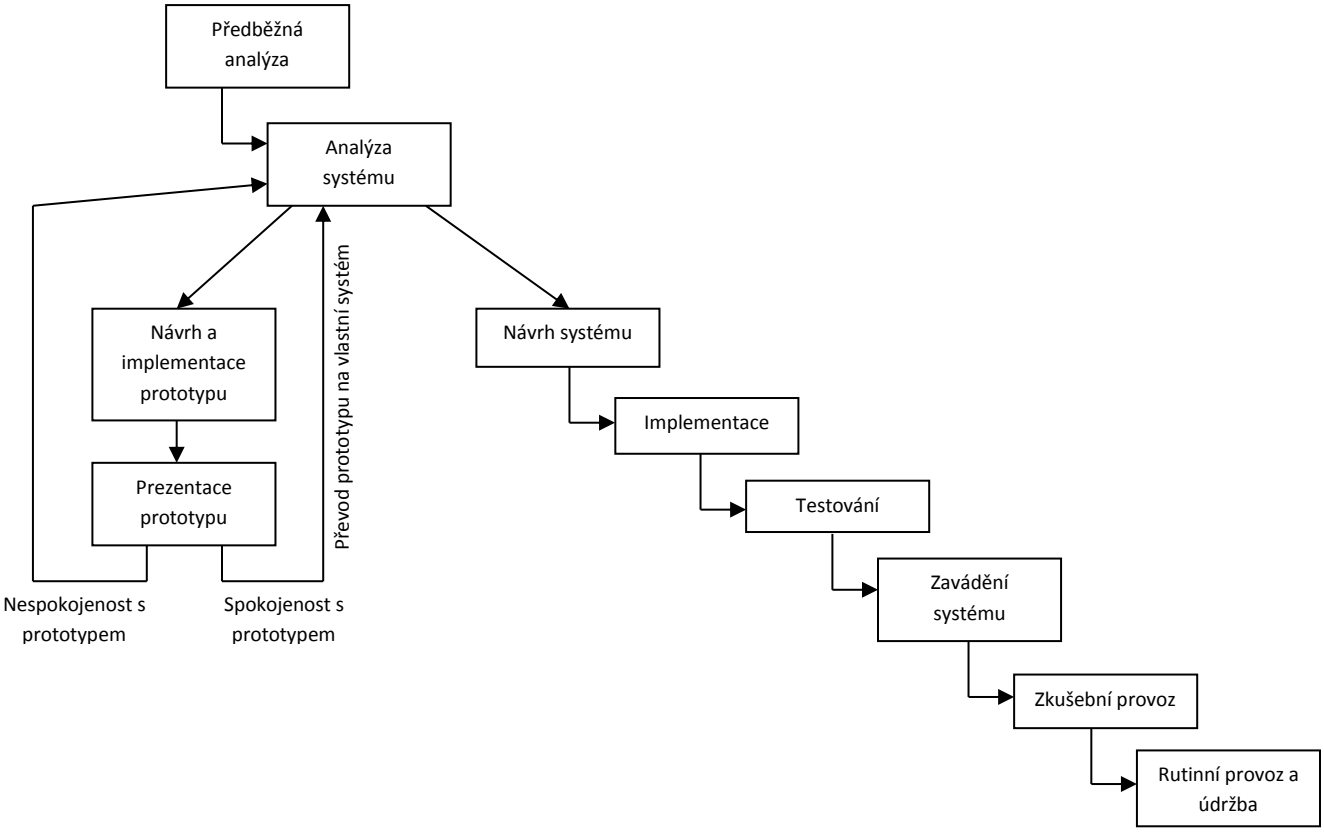
I když má model vodopád nevýhody, je to pořád lepší než ad hoc (náhodný) vývoj bez řízení.

Prototypový model

Tento model počítá se změnami postoje zákazníka v průběhu vývoje SW a dává prostor pro změny. Vytvářejí se postupně tzv. prototypy, které slouží k seznámení zákazníků s prvními verzemi systému co nejdříve

- zjednodušená implementace celku nebo plná implementace části systému
- prezentace všech GUI, základní funkčnost

Dále následuje iterativní upřesňování požadavků a jejich implementace do prototypu, dokud zákazník není spokojen, nakonec proběhne implementace.



Výhody

- umožňuje přesně definovat požadavky uživatelů
- reakce na změny zadání či požadavků

Nevýhody

- u větších systémů poměrně náročné
- vlivem iterací se může „rozpliznout“ struktura systému
- obtížné řízení
    - nevíme, v jaké fázi vývoje se nacházíme
    - nutno omezit, kolik prototypů budeme vytvářet
    - přesně stanovit termíny předání prototypů

Model spirála

Jde o kombinaci prototypového modelu a průběžné analýzy rizik, tj. opakování vývojových fází tak, že v každém dalším kroku se na již ověřenou část nabalí části na vyšší úrovni. Postup vývoje je podobný jako ve vodopádu:

- specifikace cílů a určení plánu řešení
- vyhodnocení alternativ řešení a analýza rizik
- vývoj prototypu dané úrovně, předvedení, vyhodnocení
- revize požadavků - validace (pracuje prototyp jak má?)
- verifikace – ověření: je celkový výstup daného kroku v souladu se zjištěnými požadavky?

Výhody

- využívá ověřené kroky vývoje a analýzou rizik předchází chybám
- umožňuje konzultovat požadavky zákazníků v jednotlivých krocích a modifikovat systém podle upřesněných požadavků
- prvotní verze systému lze sledovat během vzniku

Nevýhody

- vyžaduje neustálou spolupráci zákazníků
- neumožňuje přesné naplánování termínů, cen a jednotlivých výstupů
- závislost na správě provedené analýze rizik

Databáze v IS

Databáze (DB) je důležitou součástí každého IS. DB musí umožňovat

- uchovávání dat
- manipulaci s daty (vkládání, aktualizaci, mazání)
- efektivní vyhledávání dat
- propojování dat, komplexní informace

Chybný návrh databáze může způsobovat chybné dotazy nebo i nemožnost dosáhnout určitých informací. Ve strukturované analýze obvykle návrh databáze spadá do datové analýzy.

Kontrolní otázky

- Co zabírá při vývoji velkého IT systému relativně nejvíce času?
- Jaká fáze životního cyklu přichází bezprostředně po implementaci?
- Co kromě vlastního programu obsahuje software?

Úkoly k procvičení

Vymyslete zadání pro jednoduchý informační systém. Příklady zadání pro inspiraci:

- evidence železničních přejezdů
- evidence geologických jednotek
- lékárna
- sklad elektro
- knihovna
- hřbitov
- letiště
- evidence vozů ve spediční firmě
- půjčovna nářadí
- „Pragokoncert“
- emailový klient

Domluvte se se spolužákem a uskutečněte řízený rozhovor, který bude představovat první fázi životního cyklu SW, tj. předběžnou analýzu požadavků a specifikace cílů. Vystupujte jednou v roli zákazníka, jednou v roli zpracovatele. Zpracovatel má za úkol identifikovat všechny požadavky na IS zadavatele a sepíše výsledek své investigace. Bude se ptát na to, jaká data je třeba uchovávat, jaké jsou mezi nimi vztahy a co se s daty bude asi dělat (jaké jsou procesy). Zadavatel by měl mít vymyšleny aspoň tři netriviální dotazy na data.

2 Data a informace, modelování reality, databázový přístup

Cíl kapitoly

Kapitola seznamuje čtenáře s modelováním reality v počítači a s principy hromadného zpracování dat, dále poskytuje úvod k databázovému přístupu zpracování dat a k tvorbě databází. Cílem je naučit se:

- co to jsou data a co jsou informace
- proč a jak se zobrazují objekty reálného světa do počítačové reprezentace
- jaké jsou principy hromadného zpracování dat
- základní princip databázového přístupu, konstrukce databázového systému
- princip databázového modelování

Klíčové pojmy

Data versus informace, modelování reality, hromadné zpracování dat, databázový přístup, databázový systém, systém řízení báze dat, úrovně modelování databáze.

Data a informace

Tradiční úloha počítačů bylo vždy (hromadné) zpracování dat. Rozlišujme mezi daty a informacemi:

Data = naměřené údaje, výsledky pozorování, množiny znaků, množiny čísel…
Informace = interpretace dat a vztahů mezi nimi

Surová data je třeba je nějak uspořádat a zpracovat. Informace jsou výsledkem zpracování dat, jsou to tedy v podstatě interpretovaná data.

Příklad na rozdíl data vs. informace

data:

- 8110067613 je jakési číslo

informace:

- 8110067613 je rodné číslo osoby jménem Jindřich Málek (… interpretace daného čísla, co kdyby to bylo třeba telefonní číslo?)

Jako další příklad lze uvést nutnost správné interpretace fontu – znakům abecedy jsou přiřazena čísla ve znakové sadě, neznáme-li použitou znakovou sadu, nepřečteme text (anebo máme ø místo ř).

Modelování reality

Objektu reálného světa přísluší nějaký objekt v informačním systému. Musí mít vhodnou reprezentaci:

- abstrakce
- zanedbání nepodstatných vlastností
- reprezentace
    - analogová – např. v analogovém počítači, provádějícím simulaci newtonovské mechaniky, může zrychlení odpovídat napětí na nějakém prvku, elektrickou integrací pak dostaneme rychlost atd.
    - digitální – vše se nějak převede na čísla a jejich posloupnosti či struktury

Nejjednodušší údaje

- čísla, řetězce znaků
- ani zde není často reprezentace jednoduchá, např. konečným dekadickým zápisem čísla nelze dokonale reprezentovat číslo 1/3

Komplexnější objekty

- mají množství elementárních atributů, reprezentovatelných např. číslem, textovým popisem apod.
- relativně jednoduché jsou ještě např. fyzikální objekty
    - zvuk, elektrický signál – reprezentace časovým průběhem nebo spektrem
    - barva – reprezentace třeba souřadnicemi RGB
- složitější jsou např.
    - osoby, zvířata v zoo, budovy, zboží, faktury…
    - jejich reprezentace závisí na úhlu pohledu

Při modelování komplexních objektů je třeba ze všech jejich atributů vybrat jen ty, které nás zajímají a mají pro nás smysl. I taková databáze osob může být podle použití velmi různá, například v evidenci zaměstnanců továrny nebudeme uchovávat informaci o jejich porodní váze, ale o počtu jejich dětí ano; v evidenci novorozenců budeme uchovávat porodní váhu, ale nikoli informaci o počtu jejich dětí (je to vždy nula). Jiný příklad:

- v evidenci nádraží pro účely správy železniční cesty budeme uchovávat počet dopravních kolejí, ale nikoli stavební sloh výpravní budovy
- v evidenci nádraží pro účely národního památkového ústavu nebudeme uchovávat počet kolejí, ale stavební sloh výpravní budovy ano

Objekty mezi sebou mají vztahy, například místnosti se nalézají v budově, nadřízený má pod sebou zaměstnance, apod. Někdy je obtížné odlišit vlastnost objektu od vztahu s jiným objektem (podrobně viz přednáška o konceptuálním modelování).

Hromadné zpracování dat

Máme reprezentace objektů a jejich vztahů, co s nimi ale dál?

Systémy pro zpracování dat

- sběr, uchování, vyhledání a zpracování dat za účelem poskytnutí informací
    - výběr informace
    - prognózy vývoje
    - plánování
    - rozhodování
    - použití pro automatizaci inženýrských prací
    - použití pro zpracování ekonomických agend

Prudký nárůst množství dat, která je nutno spravovat vede k databázovému přístupu. Existují různé způsoby pořádání dat:

- papírové kartotéky, diář, šanon
- množiny souborů (souborový přístup) a tabulky
- databáze

Nevýhody klasických přístupů

Papírové kartotéky:

- uspořádání podle jediného kriteria (například jméno)
- vyhledání podle jiného kriteria (např. rodného čísla) je obtížné a zahrnuje často procházení všech záznamů
- komplexnější vyhledávání podle více kriterií je ještě obtížnější

Množiny souborů:

- soubory obsahují surová data; jejich popis, struktura a interpretace je „skryta“ v aplikačních programech, které se soubory pracují
- redundance a nekonzistence
    - opakování dat v různých souborech (duplicity, redundance)
    - nekonzistence = dvě kopie týchž dat nejsou stejné
- obtížný přístup
    - každý nový typ požadavku (dotazu) vyžaduje zásah programátora – nepružné
- izolace dat
    - data nejsou provázána, navíc mohou být uložena v různých formátech
- problémy s víceuživatelským přístupem
- problémy s ochranou dat
- problémy s integritou dat
    - data podléhají přirozeným integritním omezením (např. jméno je řetězec znaků), jejichž splnění je obtížné zaručit
    - data pak neodpovídají situaci z reálného světa

- kontrola vstupu musí být součástí aplikačních SW
- programy a data jsou vzájemně závislé
    - pokud je nutné změnit organizaci dat, je třeba tyto změny promítnout do všech programů, které s daty pracují.
- nízké prostředky pro tvorbu vazeb mezi daty
    - např. odkazování pomocí ukazatelů je plně v rukou programátora – technicky složité, zdlouhavé, náchylné k chybám

Databázový přístup řeší výše uvedené nevýhody:

- snaha umožnit efektivní přístup k libovolné podmnožině uložených dat
    - obtížné
- odtržení definic dat, struktury dat a manipulace s daty od uživatelských programů
- struktura dat je předem dána a je uložena spolu s vlastními daty
    - data nejsou uložena v separátních souborech, ale v organizované, centrálně řízené struktuře – databázi (DB)

Výhody databázového přístupu

- data je možné vyhodnocovat různými způsoby
- zamezení redundance dat
    - snížení celkového objemu dat
- snadnější údržba a zachování konzistence (data v databázi se musejí nacházet v konzistentním, tj. bezesporném stavu)
- možnost zabezpečit integritu dat
    - možnost kontroly vstupujících údajů též na úrovni DB, nejen na úrovni aplikace
- sdílení dat
    - možnost paralelního přístupu k datům
- ochrana dat před zneužitím
    - rozvinutá bezpečnostní politika
- nezávislost dat na aplikaci
    - datové soubory jsou striktně odděleny od aplikační části a spravují se společně
    - neizolovanost dat – data jsou uložena na stejném místě ve stejném formátu
    - aplikační programy se při vhodném návrhu DB nemusí vůbec starat o detaily uložení dat, vzniká tak vícevrstvá architektura
    - přístup k datům je možný pouze prostřednictvím databázových programů

Databáze

Databáze obsahuje principiálně 4 komponenty

- datové prvky
    - elementární hodnoty, např. jméno, rodné číslo, obrázek
- vztahy mezi datovými prvky
    - např. ten a ten vědecký článek napsal ten a ten autor

- integritní omezení (IO)
    - podmínky, které musejí data splňovat (např. věk je číslo z intervalu 0 .. 120)
- schéma
    - popis dat a jejich struktury
    - často zahrnuje i IO

Databázový systém

Databáze je definovaná schematem a je nezávislá na aplikačních programech. Správa databáze je zajišťována centrálním programem nazývaným Systém řízení báze dat (SŘBD), anglicky *database management system* (DBMS). Databázový systém (DBS) je tvořen databází a SŘBD:

DBS = DB + SŘBD

Existují různé SŘBD: Oracle, MySQL, PostgreSQL, Teradata, MS SQL Server, Firebird, SyBase…

SŘBD poskytuje

- jazykové prostředky
    - jazyk pro definici dat (data definition language, DDL) – prostředky pro popis dat, definici schematu DB, definici integritních omezení, apod.
    - jazyk pro manipulaci s daty (data manipulation language, DML) – prostředky pro vkládání, aktualizaci a mazání dat a především pro výběr dat z DB na základě nějakých kriterií.
    - data control language, DCL – prostředky pro zajištění ochrany dat, příkazy umožňující definovat práva uživatelů.
    - transaction control language, TCL – prostředky pro řízení transakcí.
- řízení víceuživatelského přístupu
- řízení transakčního zpracování
    - řešení „paralelního“ přístupu k datům – chce-li například jedna transakce číst data, nemůže je jiná v tu chvíli měnit
    - transakce je atomická jednotka práce s databází, převádí DB z jednoho konzistentního stavu do druhého, musí být vždy provedena celá, nebo vůbec
    - příklad: provádíme-li bankovní převod, peníze se musí odečíst ze zdrojového účtu, ale musí se také přičíst na účet cílový. Dojde-li uprostřed zpracování k výpadku, zdrojový účet musí být ve stejném stavu jako před započetím transakce
- zotavení z chyb
    - souvisí mj. s transakčním zpracováním, prováděné elementární operace jsou žurnálovány (log file), pokud se transakce nezdaří, je na základě žurnálu obnoven původní stav databáze (rollback)
- utajení dat
- řízení katalogu dat a paměti
- optimalizace zpracování požadavků
    - daný dotaz lze vyhodnocovat různě, SŘBD má vybrat ten nejefektivnější způsob

Je třeba se na tomto místě zmínit, že ne všechny SŘBD poskytují všechny výše uvedené prostředky.

Charakteristika dat v DB

Data jsou perzistentní, přetrvávají nezávisle na aplikacích, na tom, zda se s nimi pracuje či ne.
Data musejí být spolehlivá

- integrita databáze
    - stav, kdy jsou data správná, konzistentní a aktuální
    - data musejí musejí splňovat určitá omezení: například v databázi učeben v areálu VŠ musí být každé učebně přiřazena budova, kde sídlí, jinak jde o nekompletní nebo osiřelá data
    - může být narušena chybami SW, HW či špatným návrhem databázového schematu
- bezpečnost dat – data musejí být chráněna před neoprávněným přístupem

Data by měla být neredundantní

- jedna informace není zbytečně uložena vícekrát
- např. chceme-li evidovat, že ve Stavovském divadle se hraje Don Giovanni, Rigoletto a Prodaná nevěsta, je obvykle nutné mít prvek „Stavovské divadlo“ uložen třikrát
- obtížně dosažitelné, ne vždy výhodné
- obvyklá analogie principu neurčitosti: buď data zabírají mnoho prostoru a jsou rychle dosažitelná nebo naopak, oboje současně nelze splnit
- redundance je klíčový pojem v konceptuálním modelování, více viz další přednášky

Data jsou sdílená

- manipuluje s nimi více uživatelů současně, je nutné řešit otázku současného přístupu k týmž datům, nesmí dojít k porušení integrity a konzistence DB
    - transakční zpracování

Data jsou nezávislá

- lze změnit fyzickou reprezentaci dat, aniž by se muselo měnit databázové schema či přepisovat aplikační SW
- lze změnit část databázového schematu bez nutnosti změn celého aplikačního SW

Databázové modelování

Při modelování databáze lze rozlišit tři úrovně abstrakce, tři úrovně pohledu na databázi:

1. konceptuální model
    - nejvyšší úroveň abstrakce, modelujeme datové entity jako objekty reálného světa, nestaráme se o implementaci a architekturu
    - např. ER modelování, UML modelování
2. logický model
    - střední úroveň abstrakce, modelujeme entity jako struktury v konkrétním logickém modelu (třeba jako tabulky), pojmy konceptuálního modelování dostávají konkrétní podobu (stále se nezajímáme o efektivní implementaci)
    - např. relační, objektově-relační, objektový model

3. fyzický model
    - nízká úroveň abstrakce, implementace logického modelu v daných technických podmínkách.
    - optimalizace výkonu SŘBD a uložení dat tak, aby manipulace s nimi byla rychlá, zabezpečená a škálovatelná
    - např. management datových souborů, indexování, B-stromy, transakční zpracování

Kontrolní otázky

- Jakými prostředky dosahuje databázový přístup výhod nad klasickým souborovým přístupem?
- Co je úlohou SŘBD?
- Jak se liší jazyk pro definici dat a jazyk pro manipulaci s daty?
- Co to je integrita databáze?
- Co to je redundance?
- Jak se liší konceptuální, logický a fyzický model databáze?
- Co to je transakce?

3 Konceptuální modelování

Cíl kapitoly

Kapitola seznamuje čtenáře s konceptuálním modelem databáze, konkrétně s E-R modelem. Probírají se základní konstrukty E-R modelu a jejich význam při modelování. Dále jsou představena kriteria pro návrh korektního konceptuálního modelu, je uvedena metodika konceptuální analýzy a příklad. Cílem je naučit se:

- co to je konceptuální model
- použití E-R modelu pro databázové modelování
    - entity, vztahy, atributy
    - identifikační klíč
    - integritní omezení
    - kardinalita a parcialita vztahů
    - dekompozice M:N vztahu
    - další konstrukty E-R modelu
- jak má vypadat korektní konceptuální model (schema)
- jak se postupuje při konceptuální analýze

Klíčové pojmy

Konceptuální model, E-R model, entita, vztah, atribut, identifikační klíč, integritní omezení, kardinalita, parcialita – povinné resp. nepovinné členství ve vztahu, ISA hierarchie.

Motivace

Konceptuální model umožňuje popis dat v DB nezávisle na fyzické a logické implementaci, má co nejvěrněji vystihnout lidský konceptuální pohled na danou oblast. Konceptuální model představuje vlastně datovou analýzu

- modelování datové reality
- jaká budeme mít v informačním systému data
- pohled uživatele vs. pohled analytika

Konceptuální modely jsou různé

- síťový
    - datové prvky vzájemně propojeny ukazateli
- hierarchický
    - předpokládá stromovou strukturu datových prvků

- objektový
    - data modelována třídami, instancemi jsou objekty
- entitně-vztahový (**E-R model**, entity-relationship)
    - datové prvky (entity) a vztahy mezi nimi
    - v podstatě standard pro datové modelování
    - pro DB je ER model obdobou UML v OO
    - definuje **konceptuální schéma** databáze

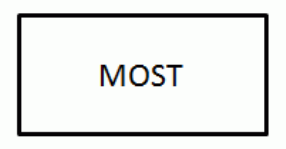
E-R model

- množina pojmů, umožňujících popis struktury DB na uživatelsky srozumitelné úrovni
- nejlépe se vyjadřuje graficky

Základní konstrukty

entita (entity)

- objekt reálného světa, který je schopen nezávislé existence a je jednoznačně identifikovatelný
- např. mostní objekt číslo 221-0608
- je obvykle odpovědí na otázku „co“, je vystižena podstatným jménem

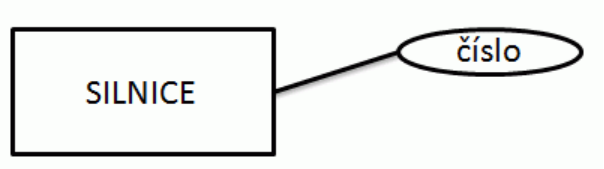


vztah (relationship)

- vazba mezi dvěma či více entitami
- např. mostní objekt č. 221-0608 leží na silnici třetí třídy č. 221, tj. obecně entita „most“ je ve vztahu „leží na“ s entitou „silnice“
- často je vystižen slovesem

atribut (attribute)

- nějaká popisná hodnota, přiřazená entitě či vztahu; např. entita „SILNICE“ má atribut „číslo“



Postup modelování

Při modelování projektant na základě podrobného seznámení s modelovanou realitou

- identifikuje typy entit jako třídy objektů téhož typu, například „SILNICE“, „MOST“ atd.
- identifikuje typy vztahů, do nichž mohou entity vstupovat, např. MOST (entita) LEŽÍ NA (vztah) SILNICI (entita)
- přiřadí typům entit a typům vztahů vhodné atributy, které popisují jejich vlastnosti
- formuluje **integritní omezení**, vyjadřující soulad schématu s modelovanou realitou, např. MOST musí ležet na nějaké SILNICI; SILNICE je identifikována číslem

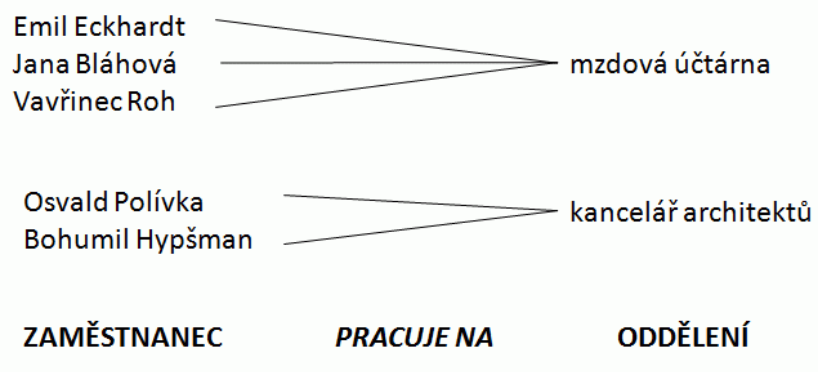
Měli bychom ještě upozornit na jistou nejednoznačnost v terminologii

- jako označení abstraktního objektu by měl být používán pojem „typ entity“: analogie třídy v OOP
- pojem „entita“ by měl být použit pro označení konkrétního výskytu objektu: analogie instance třídy v OOP

Typ entity, typ vztahu

Příklad

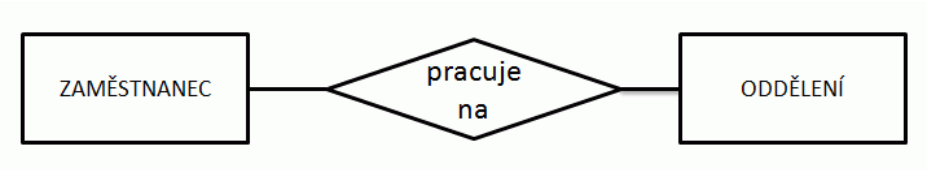
„Zaměstnanec Emil Eckhardt, rč. 105516246“ je konkrétní osoba, charakterizovaná množinou vlastností, které nás zajímají z hlediska zaměstnavatele. Osob s touto rolí může být evidentně více, říkáme, že pan Eckhardt je entitou typu ZAMĚSTNANEC. Každý zaměstnanec je zaměstnán na právě jednom oddělení, např. pan Eckhardt pracuje ve mzdové účtárně – potřebujeme tedy také entitní typ ODDĚLENÍ a vztahový typ PRACUJE NA. Tuto konkrétní situaci můžeme znázornit v tzv. **diagramu výskytů**



Máme zde

- dva entitní typy a jeden vztahový typ
- pět výskytů ent. typu ZAMĚSTNANEC
- dva výskyty ent. typu ODDĚLENÍ
- pět výskytů vztahového typu PRACUJE NA

Obecněji můžeme tutéž situaci znázornit pomocí diagramu entit a vztahů, tj. **E-R diagramu**



Identifikační klíč

Každá entita musí být jednoznačně identifikovatelná. Atribut nebo skupina atributů, jejichž hodnota slouží k identifikaci konkrétní entity se nazývá **identifikační klíč** (v E-R schematu značíme podtrženě). Příkladem může být atribut „rodné číslo“ entity ZAMĚSTNANEC.

Entitní typ může mít několik kandidátů na klíč, například ZAMĚSTNANEC může být identifikován

- rodným číslem
- číslem zaměstnance
- trojicí „jméno, příjmení, datum narození“

Volba klíče se děje zejména podle efektivity. Pokud má entita jednoduchý, nejlépe celočíselný atribut, který ji může identifikovat, zvolíme právě tento atribut (např. „číslo místnosti“ nebo „číslo faktury“ apod.), není třeba přidávat žádný umělý identifikátor. Tento způsob ovšem poněkud narazí, pokud se daný objekt může během své existence „přečíslovat“ – u faktury se to patrně nestane, ale u místnosti se to stát může. V případě pravděpodobnosti změny hodnoty klíče je tedy lepší zavést pomocný umělý celočíselný identifikátor (id_místnosti) a číslo místnosti ponechat jako neklíčový atribut. Klíčům složeným z více atributů se také spíše vyhýbáme, raději zavedeme pomocný celočíselný identifikátor: složené klíče činí potíže při indexování, při navazování referenční integrity (viz níže), apod. Klíč používající atribut složitějšího datového typu (např. neomezený text) rovněž není vhodný, raději opět zavedeme pomocný celočíselný identifikátor.

Pomocné umělé identifikátory se tedy mohou jevit jako dobré univerzální řešení, z hlediska teorie však mají své nedostatky. Předně nedokážou zajistit, aby se jeden a tentýž objekt nevyskytnul v databázi dvakrát: pokud budou mít dvě instance entity zcela totožné hodnoty všech atributů a lišit se budou pouze hodnotou umělého identifikátoru, zjevně se jedná o dva stejné objekty zadané dvakrát, pouze pod jiným číslem. Tato situace je téměř ekvivalentní stavu, kdy by entitní typ neměl vůbec žádný identifikační klíč – dvě či více instancí entity pak může mít stejné hodnoty všech atributů. Pohledem databázové teorie, zejména relačního modelu (viz níže), je tato situace nežádoucí, neboť eliminuje některé jinak velmi užitečné principy (omezení redundance, omezení nekonzistence, jednoznačnost entit). V praxi je tedy třeba nalézt kompromis a nevýhody použití umělého identifikátoru omezit např. preciznější definicí a kontrolou integritních omezení pomocí uložených procedur, triggerů, a též na úrovni aplikační vrstvy.

Přiřazení atributů

Typy entit a vztahů mají přiřazeny atributy, které je charakterizují. V klasickém pojetí jsou jednotlivé atributy atomické, tj. dále nedělitelné (tzv. 1. normální forma). Atributy také podléhají integritním omezením

- datový typ (množina hodnot, množina operací, paměťový prostor...)
- příznak, zda je atribut klíčový (PRIMARY KEY)
- příznak, zda atribut může být prázdný (NULL)
- příznak, že atribut musí mít unikátní hodnotu (UNIQUE)
- zjevně klíčový atribut musí být UNIQUE a NOT NULL

Kardinalita (poměr) vztahu

Kardinalita je integritní omezení pro vztahový typ. Existují tři druhy kardinality

- 1:1
- 1:N
- M:N

Kardinalita každého binárního vztahu je popsána dvěma tvrzeními, která vyjadřují zapojení každého ent. typu do vztahu. Mějme entitní typy KINO a FILM a vztahový typ HRAJE.

Poměr 1:1

1. dané kino dává maximálně jeden film
2. daný film je na programu maximálně v jednom kině

- může zahrnovat vztahy 0:1 a 1:0, tj. kino nic nedává nebo film se nikde nehraje, což je vlastně povinnost resp. nepovinnost členství (viz dále)

Poměr 1:N

1. dané kino může hrát více než jeden film
2. daný film se dává maximálně v jednom kině

- může zahrnovat vztahy 0:1, 1:0, 1:1
- důležitý je směr („jedno kino – víc filmů“ versus „jeden film – víc kin“)

Poměr M:N

1. dané kino může hrát více než jeden film
2. daný film se může dávat ve více než v jednom kině

- může zahrnovat vztahy 0:1, 1:0, 1:1, 1:N, N:1 – některé z nich mohou být vyloučeny přísnějšími pravidly

Kardinalita také odpovídá tvrzení, že jedna entita jednoznačně určuje druhou entitu, resp. je determinantem entity druhého typu:

- pro vztah 1:1 lze např. říct, že název kina determinuje název filmu a také název filmu determinuje název kina
- pro vztah 1:N lze např. říct, že název filmu determinuje název kina, ale název kina nedeterminuje název filmu (tj. známe-li název kina, nemůžeme jednoznačně říct, jaký film se v něm hraje, neboť jich může být více)
- u vztahu M:N není determinující ani jedna entita

Parcialita (členství) vztahu

Účastní-li se entita vztahu, říkáme, že je členem vztahu. Někdy má entita předepsanou účast ve vztahu, tj. členství je povinné, např. zaměstnanec musí být zaměstnán na nějakém oddělení. Jindy se entita do vztahu zapojovat nemusí, členství je nepovinné, např. oddělení může existovat i bez zaměstnanců. Povinné členství ve vztahu je velmi důležité integritní omezení: vyjadřuje, že entita **nemůže existovat** bez zapojení do vztahu s druhou entitou.

V grafickém konceptuálním modelu se parcialita značí různě, například

- v kině se může dávat víc filmů anebo žádný
- film se musí dávat právě v jednom kině

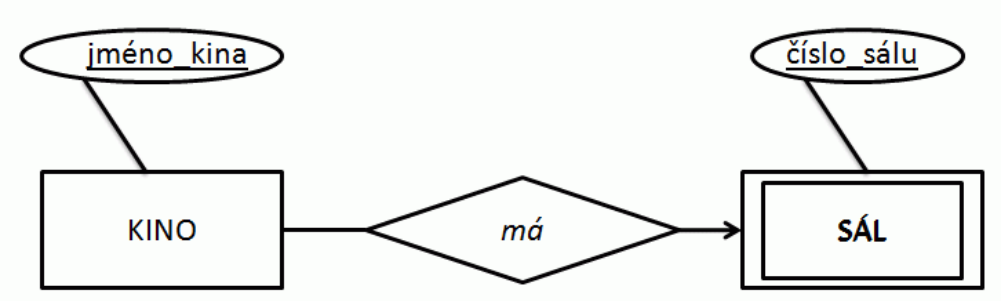
Slabé entitní typy

Součástí klíče některých entit nemusí být pouze jejich vlastní atributy. **Slabá entita** je taková entita, která není jednoznačně identifikovatelná pouze pomocí svých atributů - mohou existovat instance, které mají

stejné hodnoty atributů. Např. identifikace kinosálu je možná pouze ve spojení s identifikací multikina, v němž se nalézá:

- kino – identifikační vlastník
- kinosál – slabý entitní typ

Vztah kino-kinosál se pak nazývá identifikační vztah. Slabý entitní typ má vždy povinné členství v identifikačním vztahu, jinak by nemohl existovat. Výše uvedený příklad vyjadřuje stav, kdy v databázi může existovat více sálů s týmž číslem, které se však nalézají v různých kinech. Identifikační klíč sálu je pak dvojice (jméno_kina, číslo_sálu), vlastní atribut „číslo_sálu“ je jen částečný klíč.

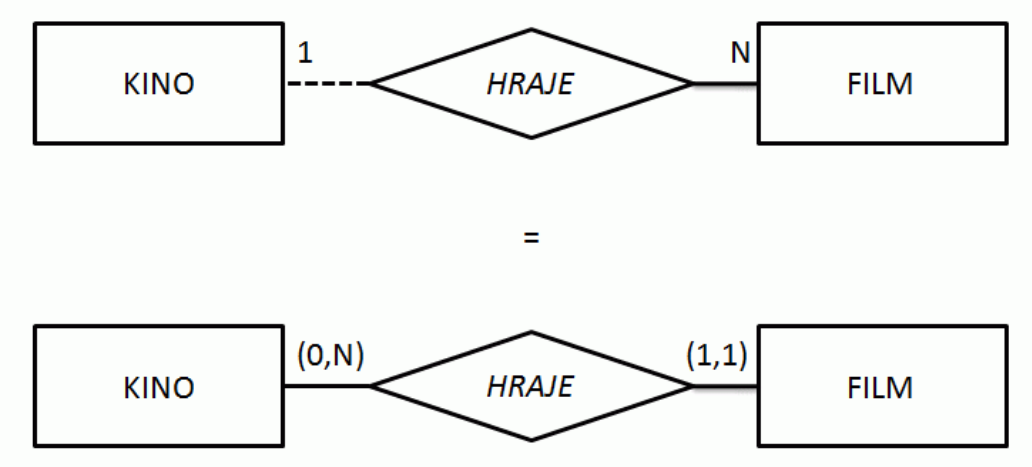


Integritní omezení pro vztahy, souhrn

Kardinalita a parcialita jsou důležitá integritní omezení, na jejich volbě závisí efektivita implementace DB:

- čím přesněji je databázové schema definováno při návrhu, tím méně chyb se může vloudit na úrovni aplikačních programů
- není vhodné zjednodušovat si práci na návrhu přílišnou benevolencí (např. nastavovat všem entitám nepovinná členství)

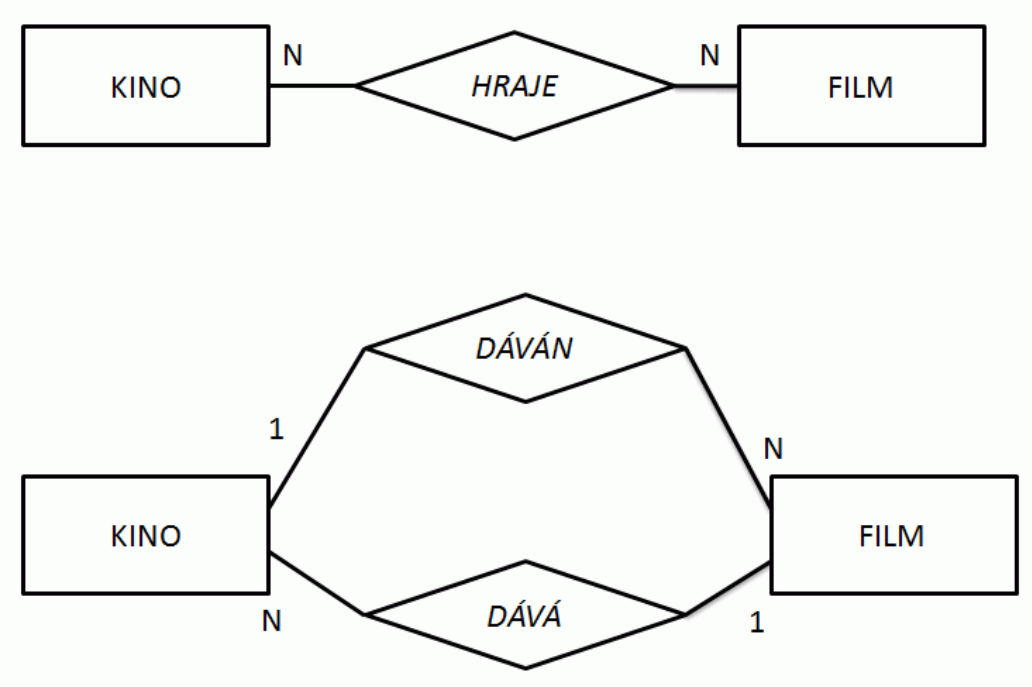
Účast entity ve vztahu lze alternativně vyjadřovat dvojicí (min,max), kde min a max udávají minimální resp. maximální počet výskytů dané entity ve vztahu, nula jako minimum označuje nepovinné členství, >0 jako minimum označuje existenční závislost:



Dekompozice M:N vztahu

Návrh konceptuálního schematu je sice nezávislý na logickém modelu, nicméně SŘBD obvykle neumějí reprezentovat vztahy M:N přímo. Vztahy M:N je určitě vhodné používat při konceptuálním modelování, avšak musíme být schopni je následně rozdělit do dvou vztahů typu 1:N – tzv. dekompozice.

Chybná dekompozice



Proč je tato dekompozice chybná:

- původní schema říká, že ani v jednom směru neexistuje funkční závislost mezi účastníky vztahu, ani jedna entita není determinantem vztahu

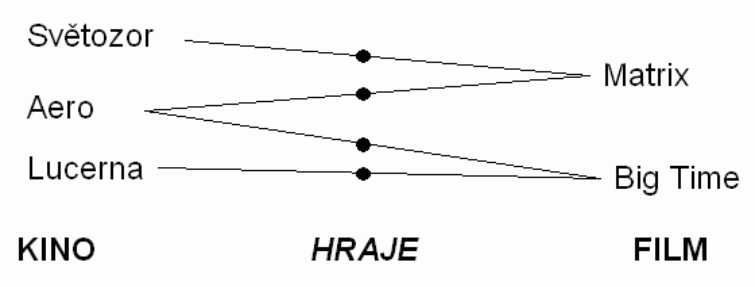
Nové schema vyjadřuje, že

- ve vztahu DÁVÁ je entita KINO determinantem vztahu, určuje tedy jednoznačně promítaný film
- ve vztahu DÁVÁN je entita FILM determinantem vztahu, určuje tedy jednoznačně kino, kde se hraje

Výše uvedené podmínky jsou ovšem současně splněny pouze pro vztah 1:1, což je jen podmnožina vztahu M:N.

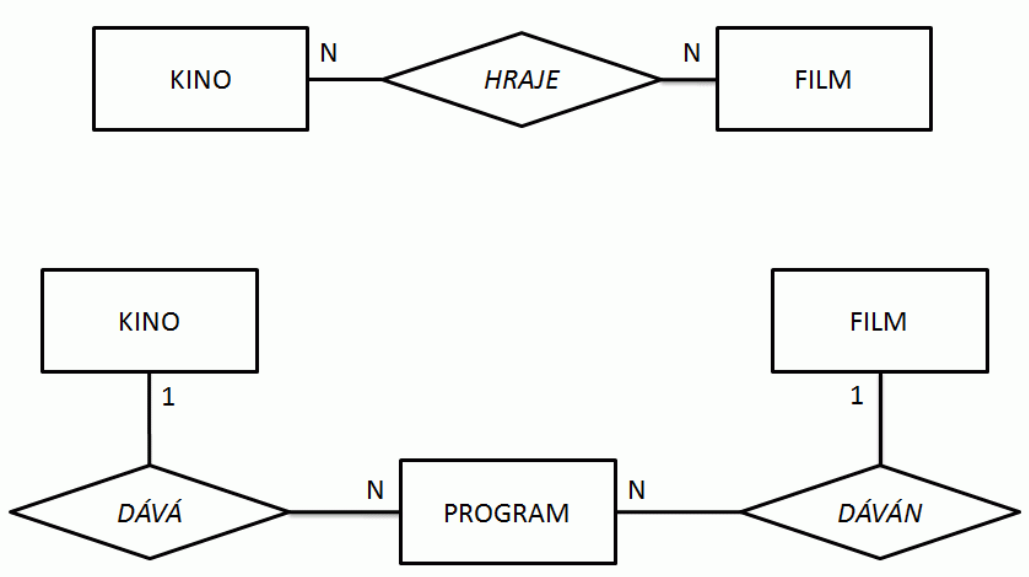
Správná dekompozice M:N vztahu

Podívejme se na diagram výskytů

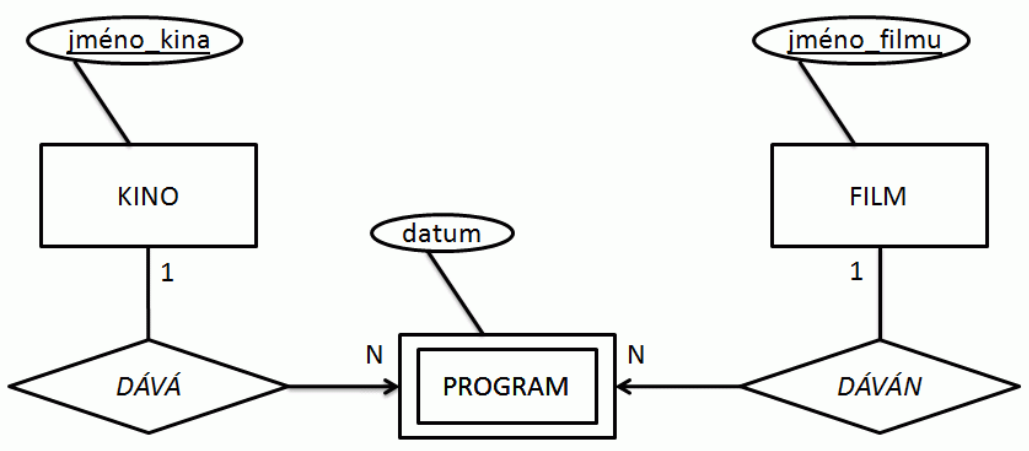


Každý výskyt vztahu odpovídá promítání konkrétního filmu v konkrétním kině. Budeme-li nyní považovat výskyt vztahu za výskyt entitního typu, máme vyhráno.

- vztah lze v podstatě vždy považovat za entitu, vzniká tzv. průnikový entitní typ



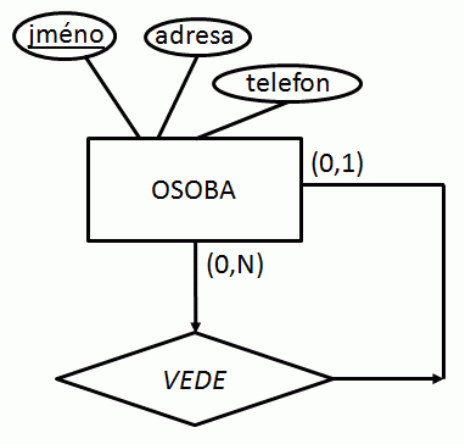
Lze též využít identifikační závislosti



- entita PROGRAM má identifikační klíč (jméno_kina,jméno_filmu)
- pokud není atribut „datum“ klíčový, kino nemůže promítat jeden film v různých dnech

Rekurzivní typ vztahu

Popisuje situaci, kdy typ entity vstupuje do vztahu sám se sebou. Vzniká tak stromová hierarchie – osoba může vést několik osob, ale je vedena pouze jednou osobou. Aby se odlišila úloha entitního typu na jedné a druhé straně vztahu, je vhodné zavést tzv. role (vede, je_veden).

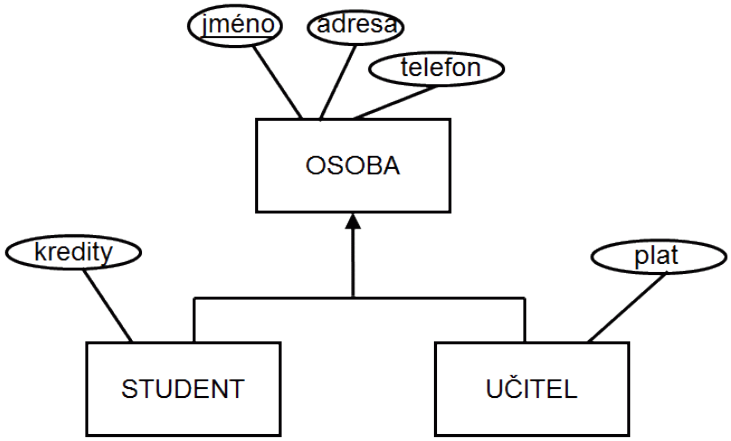


ISA hierarchie

ISA hierarchie je v podstatně analogie dědičnosti v objektově orientovaném programování, např. OSOBA může být buď STUDENT nebo UČITEL – entitní nadtyp má entitní podtypy. Atributy nadtypu se dědí, k nim může mít každý podtyp sadu vlastních atributů.

- STUDENT IS A OSOBA, UČITEL IS A OSOBA
- společné atributy: jméno, RČ, adresa
- atributy vázané pouze na podtyp
    - STUDENT: počet kreditů, ...
    - UČITEL: platová třída, ...

ISA vztah je reflexivní a tranzitivní, zavádí do schematu obvykle stromové struktury.



Důležité je si uvědomit, že entitní podtypy musí tvořit úplné pokrytí entitního nadtypu, tj. v našem případě každá osoba je buď student nebo učitel, ale nemůže být nic jiného. Naopak, platí, že prvek entitního nadtypu může patřit pouze do jednoho ent. podtypu, tj. entitní podtypy jsou vzájemně disjunktní. Tedy když je nějaká osoba studentem, nemůže být zároveň učitelem a naopak. Příslušnost k entitním podtypům je tedy výlučná.

Speciální typy atributů

Konceptuální model se nemusí omezovat pouze na atomické atributy (logický model se na ně obvykle již omezuje). Můžeme zavést speciální typy atributů: skupinový nebo vícehodnotový (multiatribut).

Skupinový atribut

- např. adresa (ulice, číslo, město, PSČ)
- heterogenní strukturovaný datový typ, podobný např. datovému typu *struct* v jazyce C
- může tvořit hierarchie
- užitečný, přistupujeme-li někdy ke složkám a někdy k celku

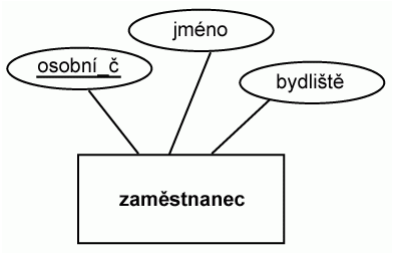
Vícehodnotový atribut

- např. atribut telefon ent. typu ZAMĚSTNANEC
- homogenní datový typ s více prvky stejného typu, podobný např. poli
- předem nevíme, kolik prvků bude mít

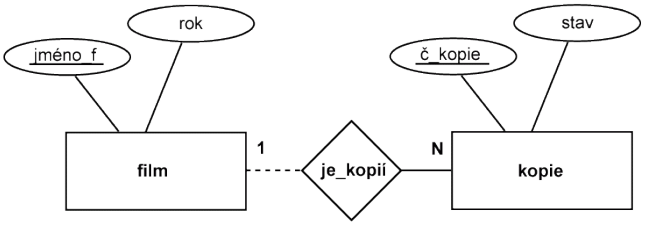
Alternativní notace E-R diagramu

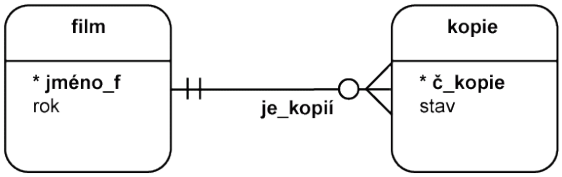
Ve většině této studijní opory je použita tzv. Chenova notace E-R diagramů. V databázové praxi se však častěji používá tzv. Martinova notace (Crow's foot notation). Tato alternativní notace je čitelnější a přehlednější, zejména při větším počtu entitních typů a jejich atributů. Proto mohu rozhodně doporučit její používání. Níže je uvedena „převodní tabulka" mezi Chenovou notací (vlevo) a Martinovou notací (vpravo), specifikující nejběžnější podobu hlavních konstruktů E-R diagramu. Existují ale i různé přechodové typy notací, které kombinují prvky obou uvedených. Poněkud matoucím rozdílem obou notací je specifikace parciality, Chenova notace uvádí povinné/nepovinné členství u entity, jíž se toto omezení týká, Martinova notace uvádí symbol nuly na opačné straně vztahu.

Entitní typ s identifikačním klíčem a atributy

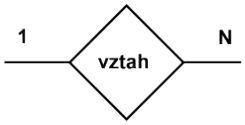


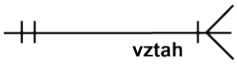
Dva entitní typy a vztah

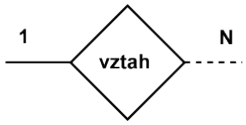


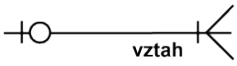


Další typy vztahů

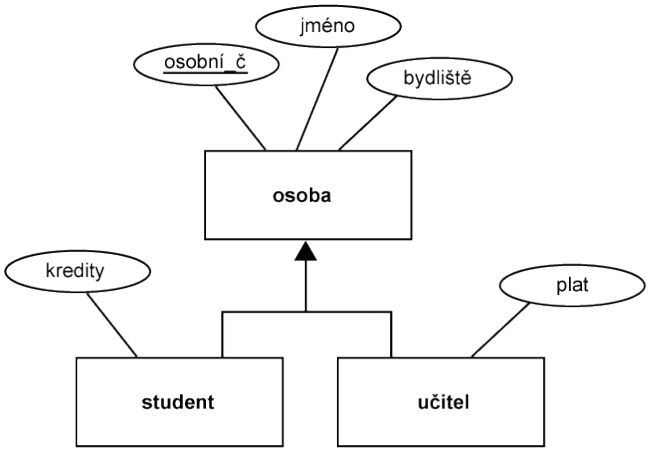


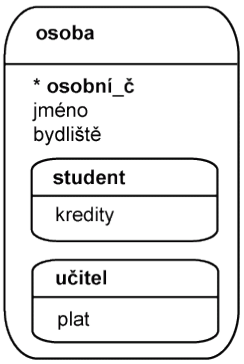






ISA hierarchie





Korektní konceptuální schema

Korektní schema má smysluplnou a jednoznačnou sémantiku. Obvykle ISA hierarchie a identifikační vztahy přinášejí potíže, např. pokud má entitní podtyp dva předchůdce (vícenásobná dědičnost), nevznikne strom, ale složitější graf, který je obtížné reprezentovat: jak identifikovat podtyp, který má dva rodiče? Zavádí se tedy pravidlo, že v ISA hierarchii musí být na vrcholové úrovni pouze jeden uzel (entita) – zdroj ISA hierarchie, pak lze snadno dědit identifikační klíč zdroje.

Korektní konceptuální schema

- každý entitní typ má max. jeden zdroj ISA hierarchie

- ISA vztahy netvoří orientovaný cyklus
    - cyklus by popisoval tytéž množiny => redundance
- identifikační vztahy netvoří orientovaný cyklus
    - entita by identifikovala sebe sama
- typ entity v ISA hierarchii, který není zdrojem, není identifikačně závislý na jiném ent. typu
    - je již identifikován zdrojem své ISA hierarchie
- jména typů entit a vztahů jsou v rámci schematu jednoznačná, jména atributů jednoznačná v rámci objektu
- vystupuje-li entita ve vztahu víckrát, každé členství je popsáno dalším identifikátorem – rolí

Analogií zdroje ISA hierarchie je identifikační zdroj

- vstupuje-li slabá entita do dvou identifikačních vztahů, pak
    - buď pro ni existují dva identifikační zdroje
    - nebo týž id. zdroj vystupuje ve dvou rolích, jež se musí odlišit označením
    - např. máme entitu ŽÁDOST_O_MANŽELSKOU_PŮJČKU a entitu OSOBA, entita OSOBA vystupuje ve dvou rolích

Splňuje-li konceptuální schema výše uvedené body, jedná se o **dobře definované schema**.

Konceptuální analýza

Z reality je třeba do modelu zobrazit podstatné jevy a to pokud možno optimálním způsobem vzhledem k plánovanému použití systému. Zvolená reprezentace reality v modelu závisí na schopnostech a vidění analytika, stejné jevy lze mnohdy modelovat různým způsobem

- viděli jsme, že rozlišení entita – vztah není vždy jasné
- také rozlišení atribut - vztah je často nejasné, v obou případech jde o funkci

Například evidujeme osoby, které sedí v místnostech. OSOBA má řadu atributů, jde určitě o samostatný entitní typ, souvislost osoby s místností může být ale vyjádřena různě

- číslo místnosti přidáme jako další atribut OSOBY
- nebo zavedeme entitní typ MÍSTNOST a typ vztahu SEDÍ

První způsob lze akceptovat, pokud například skoro v každé místnosti sedí jedna osoba a pokud nás nezajímají další atributy místnosti. Jakmile se u místnosti evidují další vlastnosti, jako třeba telefonní linka, kapacita a možnost vstupu na čip, určitě použijeme druhý způsob, protože „kapacita“ jistě není vlastností OSOBY.

Vhodná modelová reprezentace není samoúčelná, špatný návrh nám totiž může znemožnit určité typy dotazů. Např. zvolíme-li v tomto příkladu způsob 1, nejsme schopni evidovat místnosti, ve kterých zrovna nikdo nesedí. Na úrovni logického modelu (relačního) se tyto anomálie řeší tzv. **normalizací** (viz další kapitoly).

Metodika návrhu databáze

První krok

- shromažďování a formulace uživatelských požadavků (diskuse se zaměstnanci)
- identifikace a dokumentace dat, která je třeba reprezentovat
- zde se vynořují entity a vztahy, hrubá datová a funkční analýza

Základní funkční analýza

Přestože se zde funkční analýzou nezabýváme, je nutné se o ní zmínit, neboť povědomí o procesech, které budou databázi využívat, je klíčové pro optimální návrh schematu

- rozpracujeme pracovní postupy, které provádějí typičtí pracovníci, rozhodnout, zda dostanou počítačovou podporu
- z typických pracovníků se později stanou uživatelské role (třídy uživatelů)
- identifikace událostí, které spouštějí pracovní postupy

Vlastní E-R modelování

- definice typů nezávislých entit, pro začátek stačí navrhnout identifikátory, další atributy počkají
- definice typů vztahů
- definice typů závislých entit
- přiřazení atributů entitním a vztahovým typům

Proces by měl probíhat tak dlouho, dokud nedojde k souhlasu modelu s představou zadavatele.

Příklad konceptuálního modelu

Zadání – databáze psího útulku

Chceme evidovat psy, kteří jsou chováni v kotcích v útulku. Útulek je dělen na pavilony, v každém pavilonu se může nalézat více kotců. V jednom kotci může být víc psů. Každý evidovaný pes musí mít přiřazen kotec, určitý kotec ale může být prázdný. Dále evidujeme ošetřovatele, každý ošetřovatel má na starost několik kotců, každý kotec má jednoho ošetřovatele. U psa chceme evidovat: číslo známky (pokud ji má), přibližnou rasu, přibližný věk, stručně popis. U kotce chceme evidovat: číslo (unikátní v rámci pavilonu), kapacitu, typ. U pavilonu chceme evidovat: id. kód, umístění, popis. U ošetřovatele evidujeme: jméno, rodné číslo, pohlaví. Zajímá nás i historie, kde kdy byl který pes.

První krok – identifikace entitních typů a klíčů

PES – *evidenční číslo* (číslo známky by mohlo být kandidátem na klíč, ale co např. se psy, kteří nemají známku?)
KOTEC – *číslo* (můžeme si dovolit, neočekáváme přečíslovávání kotců)
PAVILON – *id. kód* (můžeme si dovolit, neočekáváme přečíslovávání pavilonů)
OŠETŘOVATEL – *osobní číslo* (rodné číslo by mohlo být kandidátem na klíč, má ale nevhodný datový typ – řetězec – a navíc u cizinců se každý půlrok mění a nemusí být zcela unikátní)

Druhý krok – identifikace vztahů

- „bydlí“ mezi entitami PES a KOTEC: M:N
- „umístěn“ mezi entitami PAVILON a KOTEC: 1:N
- „stará se“ mezi ent. OŠETŘOVATEL a KOTEC: 1:N

Třetí krok – identifikace slabých entitních typů

KOTEC – závisí na entitě PAVILON

Čtvrtý krok – přiřazení atributů

PES

- evidenční číslo
- číslo známky (celé číslo větší než 0)
- rasa (text max 255 znaků),
- rok narození (celé číslo větší než 1900)
- popis (text neomezený)

KOTEC

- číslo (celé číslo > 0)
- kapacita (celé číslo > 0)
- typ (text neomezený)

PAVILON

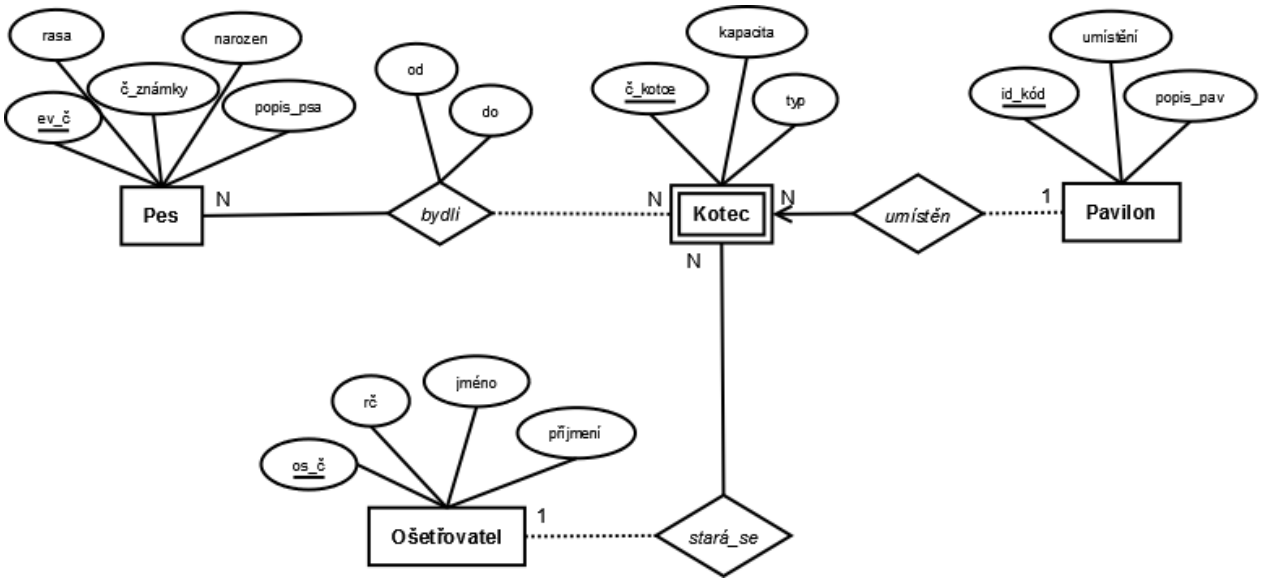
- id. kód (řetězec max. 3 znaky)
- umístění (text max 255 znaků)
- popis (text neomezený)

OŠETŘOVATEL

- osobní číslo
- rodné číslo (řetězec 9 nebo 10 znaků, dělitelný 11)
- jméno (text max 30 znaků)
- příjmení (text max 50 znaků)

vztah bydlí

- od (datum)
- do (datum)



Kontrolní otázky

- Co to je silný a slabý entitní typ?
- Může být entitní typ zapojen do více vztahů?
- Může být vztah zapojen do dalších vztahů?
- Co to je integritní omezení?
- Co to je ISA hierarchie?
- Jaké jsou postupné kroky při tvorbě konceptuálního modelu?

Řešené příklady

Příklad 1

Namodelujeme jednoduchou databázi žáků ve třídách. Jeden žák může chodit pouze do jedné třídy, ale v jedné třídě může být více žáků. U třídy evidujeme její kód (např. 1A) a jméno třídního, identifikátorem třídy bude její kód. U žáka evidujeme rodné číslo, jméno a do jaké třídy chodí, identifikátorem žáka bude umělý atribut - evidenční číslo. Nezajímá nás historie, evidujeme jen aktuální stav. Uvědomte si determinanty vztahu/ů. Varianta A: povolujeme evidenci prázdných tříd, varianta B: každá třída musí mít přiřazeného aspoň jednoho žáka.

Řešení:

- Identifikace entitních typů a jejich klíčů:

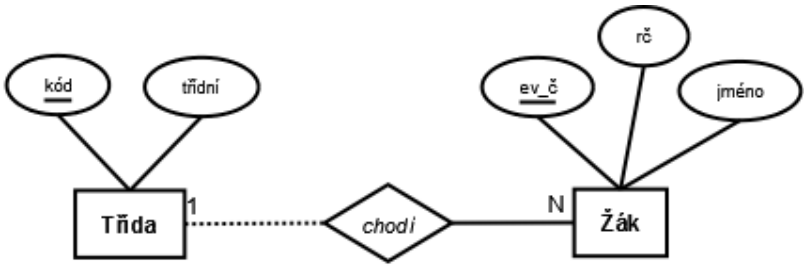
  ŽÁK – evidenční číslo (rodné číslo by mohlo být kandidátem na klíč, má ale nevhodný datový typ a navíc u cizinců se každý půlrok mění a nemusí být zcela unikátní
  TŘÍDA – kód

- Identifikace vztahů a jejich integritních omezení:

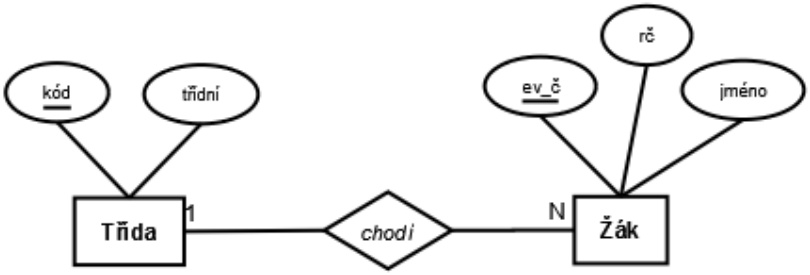
  Budeme mít jediný vztah pojmenovaný „chodí“, bude typu 1:N (jednu třídu může navštěvovat víc žáků, jeden žák chodí jen do jedné třídy). Entitní typ ŽÁK bude mít v tomto vztahu povinné členství (každý žák musí chodit do nějaké třídy), entitní typ TŘÍDA bude mít ve vztahu členství nepovinné (varianta A, třída může být prázdná, nemusí do ní chodit ani jeden žák), resp. povinné (varianta B, třída nesmí být prázdná, musí do ní chodit aspoň jeden žák).

- Determinantem vztahu *chodí* je entitní typ Žák, neboť u žáka lze vždy jednoznačně říci, kterou třídu navštěvuje. Entitní typ Třída není determinantem vztahu, neboť u třídy není jednoznačně dán žák, který ji navštěvuje; takovýchto žáků může být obecně více.

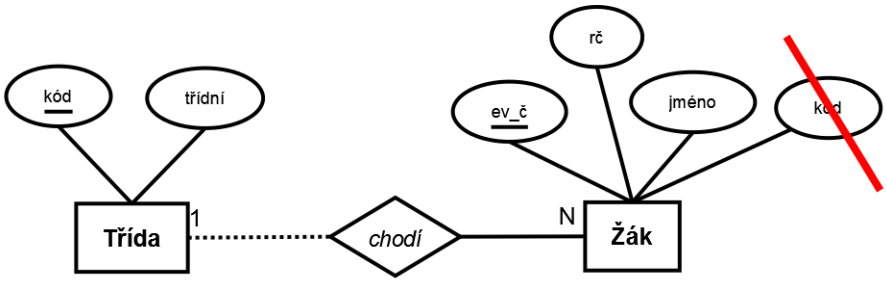
  A.



  B.



Následující obrázek zachycuje chybné řešení (varianta A):



Proč je tento diagram chybně? Příslušnost žáka ke třídě je jednoznačně dána vztahem *chodí*, atribut *kód* u entitního typu Žák je tedy nadbytečný. Poznámka: po transformaci ER diagramu do relačního modelu (popř. do tabulek v SQL) bude v relaci Žák skutečně atribut *kód* přítomen, nicméně se tam dostane až dodatečně transformací vztahu *chodí*, viz příklady na transformaci ER modelu do relačního modelu.

Příklad 2

Evidujeme studenty a vedoucí BP. Každý student si může zvolit pouze jednoho vedoucího, jeden vedoucí však může vést více studentů. Evidujeme i studenty, kteří zatím nemají žádného vedoucího. Naopak každý učitel, vedený v evidenci vedoucích, musí mít přiřazeného aspoň jednoho studenta. U studenta i vedoucího evidujeme: jméno, příjmení, rodné číslo. U vedoucího evidujeme navíc tituly; případ, kdy by měl nějaké tituly i student, explicitně neřešíme – případné tituly studenta nás nezajímají. Podobně explicitně neřešíme situaci, kdy by student byl zároveň vedoucím – pak se bude jeho záznam vyskytovat v obou entitách, pokaždé v jiné roli.

Řešení:

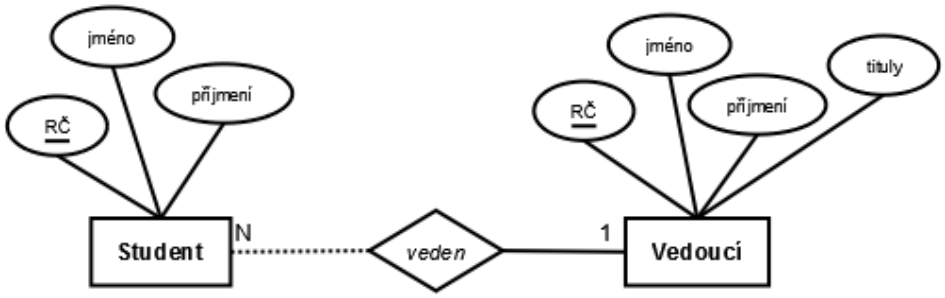
- Identifikace entitních typů a jejich klíčů:

  STUDENT – rodné číslo (pro jednoduchost, nevýhody viz příklad 1)
  VEDOUCÍ – rodné číslo (pro jednoduchost, nevýhody viz příklad 1)

- Identifikace vztahů a jejich integritních omezení:

  Budeme mít jediný vztah pojmenovaný *veden*, bude typu 1:N (jeden vedoucí může vést více studentů, jeden student má jen jednoho vedoucího), entitní typ Student bude mít v tomto vztahu nepovinné členství (evidujeme i studenty, kteří nemají zatím vedoucího BP), entitní typ Vedoucí bude mít ve vztahu členství povinné (každý vedoucí musí mít pod sebou aspoň jednoho studenta).

- Determinantem vztahu *veden* je entitní typ Student, neboť u studenta lze vždy jednoznačně říci, jakého má vedoucího. Entitní typ Vedoucí není determinantem vztahu, neboť vedoucí může vést více studentů, student není tedy pro daného vedoucího jednoznačně dán.



Příklad 3

Chceme evidovat piloty a jejich letadla. U pilota evidujeme: jméno, příjmení, rodné číslo, datum poslední lékařské prohlídky. U letadla evidujeme: typ, rok výroby, datum poslední revize, evidenční číslo. Víme, že každý pilot může vlastnit více letadel, ale jedno letadlo patří vždy jen jednomu pilotovi. Chceme evidovat i piloty, kteří žádné letadlo zatím nevlastní. U letadel však vždy musí být uveden vlastník. Nezajímá nás historie, evidujeme jen aktuální stav.

Řešení:

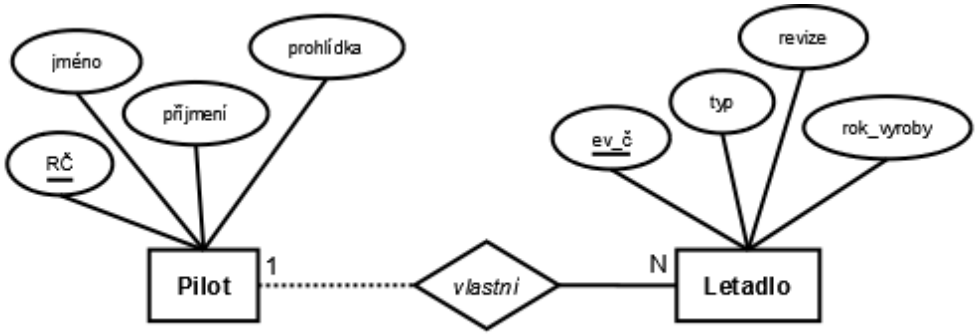
- Identifikace entitních typů a jejich klíčů:

  PILOT – rodné číslo (pro jednoduchost, nevýhody viz příklad 1)
  LETADLO – evidenční číslo

- Identifikace vztahů a jejich integritních omezení:

  Budeme mít jediný vztah pojmenovaný „vlastní“, bude typu 1:N (jeden pilot může vlastnit víc letadel, jedno letadlo má jen jednoho vlastníka), entita PILOT bude mít v tomto vztahu nepovinné členství (evidujeme i piloty, kteří nevlastní žádná letadla), entita LETADLO bude mít ve vztahu členství povinné (u každého letadla musíme znát vlastníka).

- Determinantem vztahu *vlastní* je entitní typ Letadlo, neboť u letadla lze vždy jednoznačně říci, kdo je jeho majitel. Entitní typ Pilot není determinantem vztahu, neboť pilot může vlastnit více letadel – určení vlastněného letadla pro daného pilota tedy není jednoznačné.



Příklad 4

Chceme evidovat psy, kteří jsou chováni v kotcích v útulku. Útulek je dělen na pavilony, v každém pavilonu se může nalézat více kotců. V jednom kotci může být víc psů. Každý evidovaný pes musí mít přiřazen kotec, určitý kotec ale může být prázdný. U psa chceme evidovat: evidenční číslo, přibližnou rasu, přibližný věk, stručně popis. U kotce chceme evidovat: číslo, kapacitu, typ, nepředpokládáme přečíslovávání kotců. U pavilonu chceme evidovat: číslo, umístění, popis, nepředpokládáme přečíslovávání pavilonů. Nezajímá nás historie, evidujeme jen aktuální stav.

Řešení:

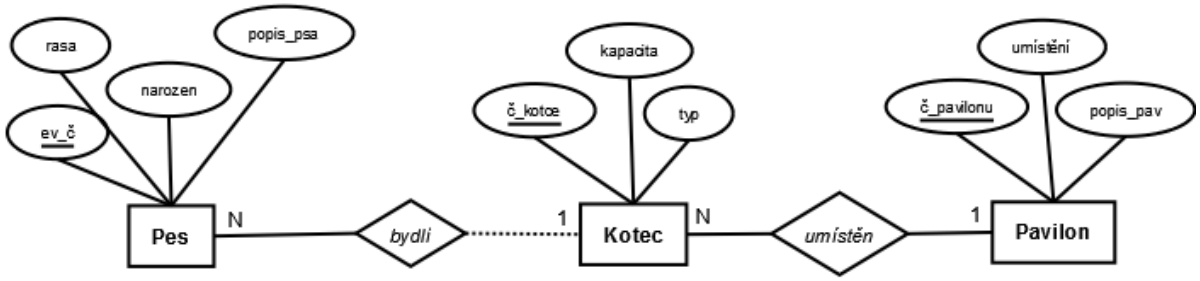
- Identifikace entitních typů a jejich klíčů:

  PES – evidenční číslo (umělý identifikátor)
  KOTEC – číslo (pro jednoduchost, nepředpokládáme přečíslovávání kotců)
  PAVILON – číslo (pro jednoduchost, nepředpokládáme přečíslovávání pavilonů)

- Identifikace vztahů a jejich integritních omezení:

  Budeme mít dva vztahy: „bydlí“ a „umístěn“, oba budou typu 1:N. Integritní omezení viz ER diagram.

- Determinantem vztahu *bydlí* je entitní typ Pes (vždy víme, v kterém kotci bydlí), determinantem vztahu *umístěn* je entitní typ Kotec (vždy víme, v kterém pavilonu se nalézá).



Příklad 5

Máme studenty, kteří navštěvují předměty. Každý student si může zapsat více předmětů, v každém předmětu může studovat více studentů. Předmět může být prázdný. Student nemusí studovat nic. U studenta evidujeme: jméno, příjmení, rodné číslo. U předmětu evidujeme: kód, název, syllabus.

Řešení:

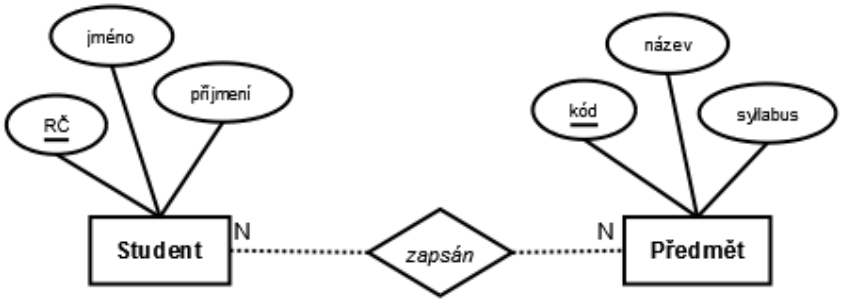
- Identifikace entitních typů a jejich klíčů:

  STUDENT – rodné číslo (pro jednoduchost, nevýhody viz příklad 1)
  PŘEDMĚT – kód

- Identifikace vztahů a jejich integritních omezení:

  Budeme mít jeden vztah „zapsán“ a to typu N:N – každý student si může zapsat víc předmětů, každý předmět může studovat více studentů. Oba entitní typy mají nepovinné členství ve vztahu.

- Determinantem vztahu *zapsán* není ani jeden entitní typ – pro daného studenta nemůžeme jednoznačně určit zapsaný předmět (může jich být více); pro daný předmět nemůžeme jednoznačně určit zapsaného studenta (může jich být více).



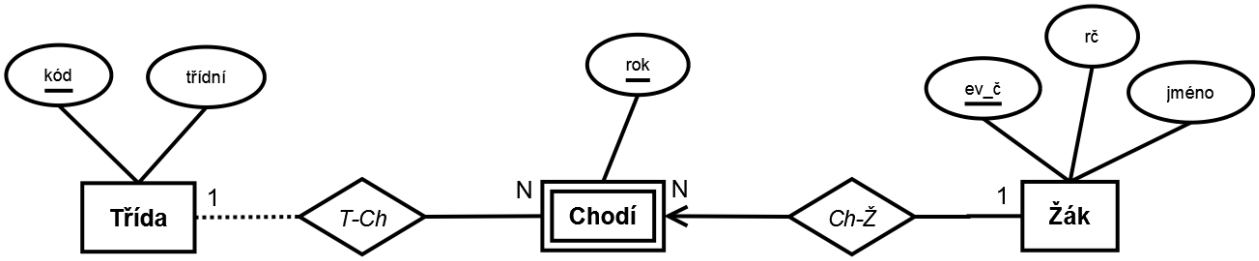
Příklad 6

Použijeme mírně upravené zadání z příkladu 1 a namodelujeme jednoduchou databázi žáků ve třídách. Jeden žák může chodit pouze do jedné třídy, ale v jedné třídě může být více žáků. U třídy evidujeme její kód (např. 1A) a jméno třídního. U žáka evidujeme rodné číslo, jméno a do jaké třídy chodí. Tentokrát budeme ale chtít evidovat i historii, tj. do jaké třídy kdy chodil jaký žák, rozlišovaná období jsou školní roky. Povolujeme evidenci prázdných tříd.

Řešení:

Základní entitní typy, tj. Žák a Třída, budou shodné jako v řešení bez evidence historie, mezi nimi bude rovněž jeden vztah. Navíc je ale nyní nutné nějak rozlišit, kdy kam který žák chodil. Tedy poněkud se změní pohled na vztah *chodí*: jeden žák bude nyní moci chodit do více tříd, pochopitelně ale nikoli současně – toto tvrzení připomíná vztah N:N. Vztah *chodí* bude vhodné reprezentovat novým slabým entitním typem, který bude de facto dekompozicí vztahu N:N. V tomto entitním typu budeme uchovávat, ve kterém školním roce chodil žák do jaké třídy. Slabý entitní typ bude závislý na entitním typu Žák a dále bude mít parciální klíč „rok“. Klíčem entitního typu *Chodí* tedy bude dvojice {ev_č, rok}, a to z toho důvodu, že jeden žák smí v jeden rok chodit pouze do jedné třídy, tedy kombinace {ev_č, rok} musí být unikátní. Naproti tomu do jedné třídy smí v daném roce chodit více žáků, proto nebude kód třídy součástí

primárního klíče. Zavedením vztahového entitního typu vzniknou dva vztahy kardinality 1:N, determinantem je v obou případech entitní typ *Chodí*.



Příklad 7

Navrhněte ER model evidence služebních cest. Máme seznam zaměstnanců, kteří vyjíždějí na služební cesty. Na jednu služební cestu může jet více zaměstnanců současně. Ke služební cestě používají zaměstnanci vždy služební vozy. Eviduje se, který zaměstnanec řídí a který je jen spolujezdec. Ke každé služební cestě musí být přiřazen právě jeden řidič a nula či více spolujezdců. Služební vozy mají přiřazenou kategorii podle řidičských oprávnění (např. B, C, D, apod.). Řidičská oprávnění evidujeme i u zaměstnanců, abychom mohli správně přiřadit danému služebnímu vozu řidiče.

Řešení:

- Identifikace entitních typů a jejich klíčů:

  ZAMĚSTNANEC – osobní číslo (rodné číslo má již dříve zmíněné nevýhody)
  CESTA – číslo
  VŮZ – evidenční číslo (kandidátem by mohlo být i SPZ, ale ta se při přeregistrování vozu mění)
  OPRÁVNĚNÍ – kód

- Identifikace vztahů a jejich integritních omezení:

  Budeme mít dva vztahy mezi Zaměstnancem a Cestou: „řídí“ a „jede“. Ke každé služební cestě musí být přiřazen právě jeden řidič a nula či více spolujezdců. Řešením by mohlo být i zavedení jediného vztahu mezi Zaměstnance a Cestou, který by obsahoval příznak, zda je daný zaměstnanec na dané cestě řidičem. Toto řešení ale samo o sobě nezaručí, aby na služební cestě byl vždy právě jeden řidič, tato podmínka by se tedy musela kontrolovat nějakým dalším způsobem (trigger, uložená procedura, aplikace).

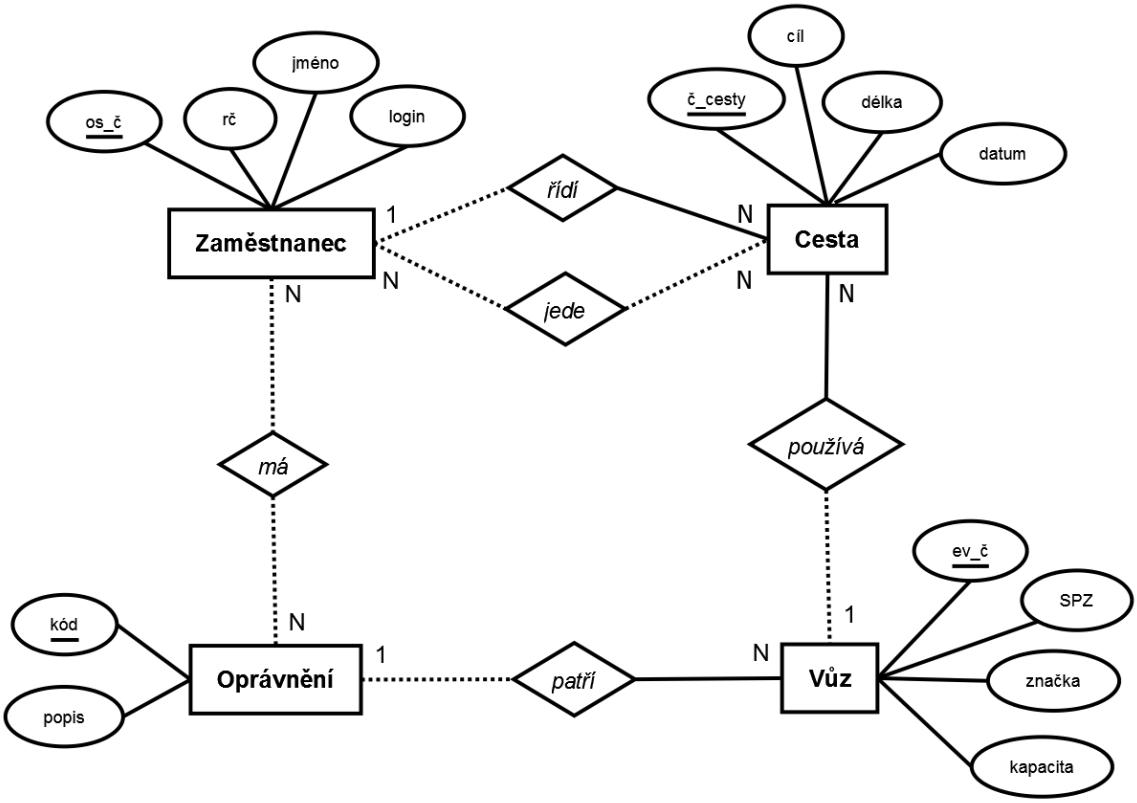
  Dále bude N:N vztah „má“ mezi Zaměstnancem a Oprávněním – daný zaměstnanec může mít žádné či několik řidičských oprávnění, určité oprávnění může (a nemusí) mít více zaměstnanců.

  Dále bude 1:N vztah „patří“ mezi Vozem a Oprávněním – daný vůz odpovídá právě jednomu oprávnění, jedno oprávnění ale může (a nemusí) mít samozřejmě víc vozů.

Dále bude 1:N vztah „používá“ mezi Cestou a Vozem – daný vůz se může účastnit více cest, daná cesta se koná vždy právě jedním vozem.

- Identifikace dalších integritních omezení:
    - na dané cestě nemůže být tentýž zaměstnanec zároveň řidičem a spolujezdcem
    - řidič určité služ. cesty musí mít oprávnění k použitému vozu
    - počet cestujících nesmí překračovat kapacitu vozu

  Uvedená fakta nelze do diagramu nijak zakreslit, proto je nutné je uvést zvlášť. Při implementaci se pak musí požadavky ošetřit např. triggerem na úrovni serveru nebo jiným způsobem na úrovni aplikace.



Příklad 8

Navrhněte ER model mateřské školy. Databáze musí ve všech vztazích evidovat současný i minulý (a potencionálně samozřejmě i budoucí) stav, rozlišovanými obdobími jsou školní roky. Evidujeme děti, které patří do tříd (identifikované symboly jako jsou zvířátka apod., symboly mají dvoupísmenné zkratky), dítě může chodit každý rok do jiné třídy. Každou třídu má na starosti jedna učitelka, jedna učitelka má v daném roce na starosti vždy jen jednu třídu. Evidujeme také místnosti (číslované) ve školce, místnosti se dělí na třídy (herny), jídelny a kanceláře; každá herna může náležet více třídám, jedna třída má jen jednu hernu; každá jídelna může náležet několika třídám, ale každá třída má jen jednu jídelnu; každá kancelář náleží několika učitelkám, každá učitelka sedí v jedné kanceláři.

Řešení:

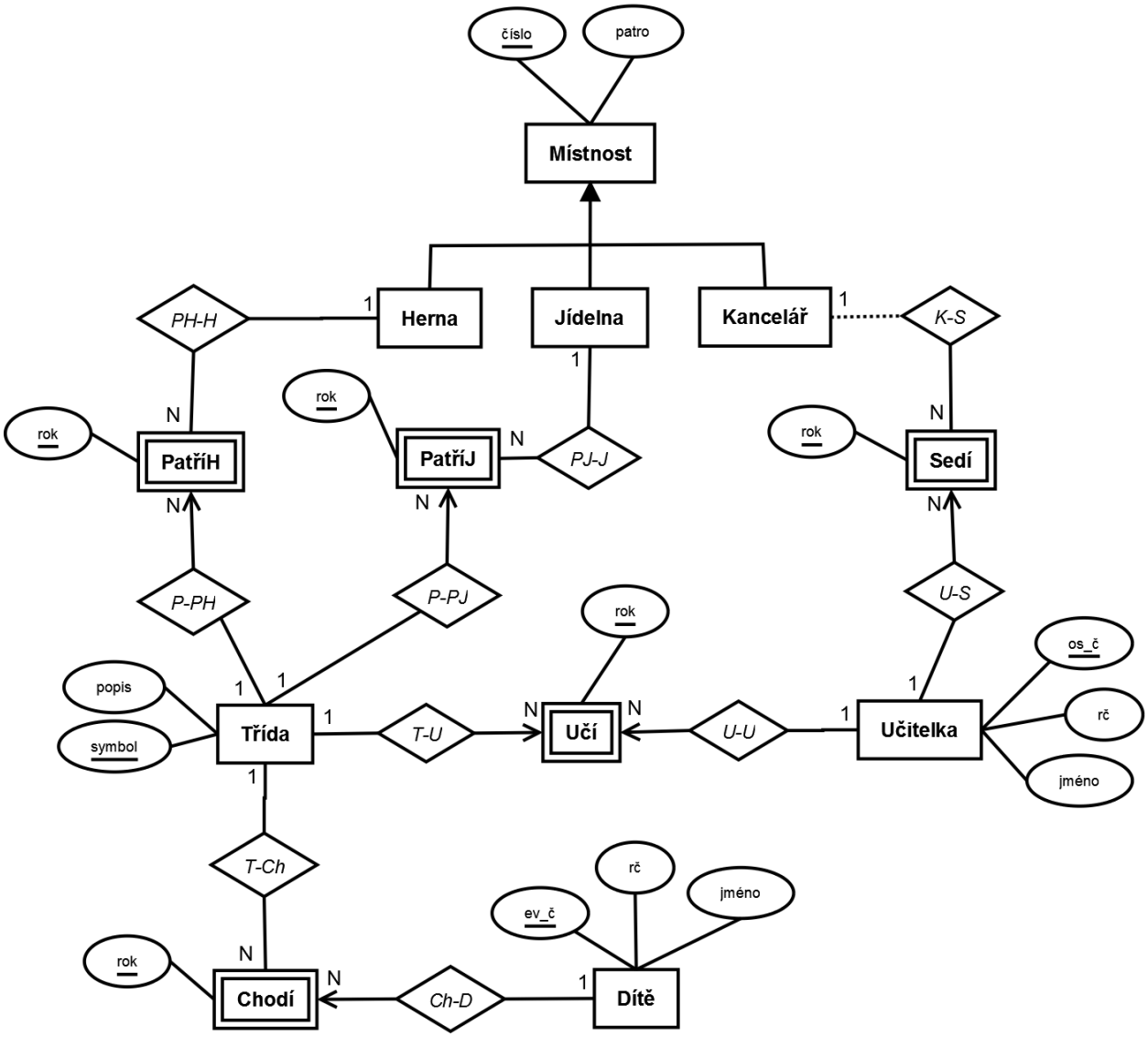
- Identifikace hlavních entitních typů a jejich klíčů:

  DÍTĚ – evidenční číslo (nevýhody rodného čísla již byly zmíněny)
  TŘÍDA – symbol
  UČITELKA – osobní číslo (nevýhody rodného čísla již byly zmíněny)
  MÍSTNOST – číslo (nepředpokládáme přečíslovávání místností)

  MÍSTNOST bude kořen ISA hierarchie s podtypy HERNA, JÍDELNA a KANCELÁŘ.

- Identifikace principiálních vztahů a jejich integritních omezení:
  - Dítě-Třída (*Chodí*), N:1, povinné členství na obou stranách
  - Učitelka-Třída (*Učí*), 1:1, povinné členství na obou stranách
  - Třída-Herna (*PatříH*), N:1, povinné členství na obou stranách
  - Třída-Jídelna (*PatříJ*), N:1, povinné členství na obou stranách
  - Kancelář-Učitelka (*Sedí*), 1:N, povinné členství u Učitelky

Všechny vztahy mají evidovat historii, proto pro každý vztah zavedeme slabý entitní typ, který bude mít parciální klíč „rok“. V případě vztahů 1:N bude tento slabý typ závislý na determinantu vztahu, tím se zajistí požadovaný poměr. Zdůvodnění: potřebujeme, aby klíčem slabého entitního typu *Chodí* byla dvojice ev_č dítěte a školní rok, neboť v daném školní roce smí dítě chodit jen do jedné třídy, tudíž dvojice {ev_č,rok} musí být vždy unikátní. Naproti tomu do jedné třídy smí chodit v daném roce více dětí, proto se dvojice {rok,symbol} může v relaci opakovat vícekrát (tolikrát, kolik dětí do třídy daný rok chodí) a tedy symbol nebude součástí klíče. V případě vztahu 1:1 se nabízí více řešení závislostí, žádné z nich však nemůže samo o sobě zajistit poměr 1:1 (zdůvodnění ponecháváme na čtenáři). Zvolíme např. závislost slabého ent. typu na obou hlavních entitních typech. Zajistit poměr 1:1 je nutné např. triggerem na úrovni serveru nebo jiným způsobem na úrovni aplikace.

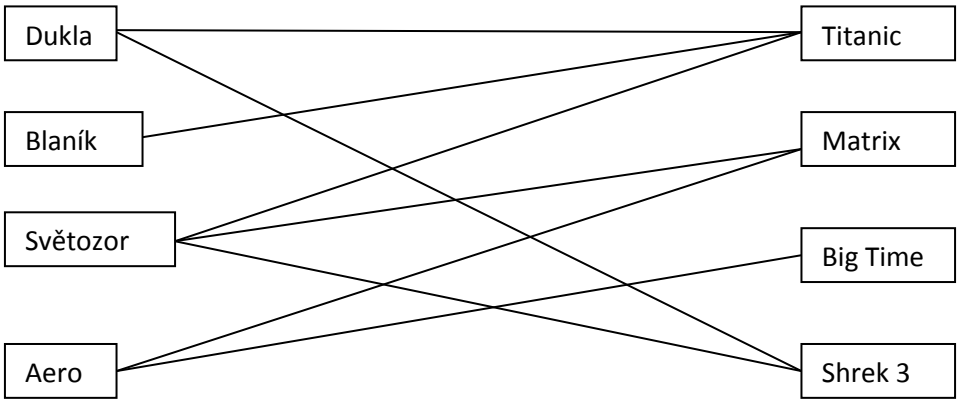

číslo
patro
Místnost
PH-H
Herna
Jídelna
Kancelář
K-S
rok
PatříH
PatříJ
PJ-J
Sedí
P-PH
P-PJ
U-S
popis
symbol
Třída
T-U
Učí
U-U
Učitelka
os_č
rč
jméno
T-Ch
ev_č
Chodí
Ch-D
Dítě
1
N

Úlohy k procvičení

1. Mějme seznam kin a seznam filmů, v každém kině se může dávat víc filmů a každý film se může dávat ve více kinech.

   kino má atributy: název, adresa, počet míst, Dolby ano/ne
   film má atributy: název, rok vzniku, země vzniku, dabing ano/ne

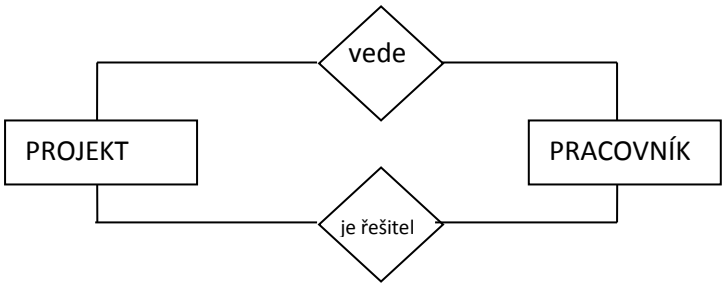
   Příklad výskytového diagramu:



   Navrhněte E-R model. Řešte dvě varianty:
   - chceme evidovat jen filmy, které jsou někde dávány
   - chceme evidovat všechny filmy

---

2. Mějme ER diagram na obrázku. Nakreslete výskytový ER diagram, klíčovým atributem typu entity PROJEKT je NázevProjektu, klíčovým atributem entitního typu PRACOVNÍK je ČísloPrac. Pro vyjádření jednoho typu vztahu použijte plnou spojnici, pro druhý čárkovanou.



3. Konceptuální schema má obsahovat atributy adresa domu, počet bytů v domě, jméno majitele domu, jméno nájemníka. V modelovaném světě se má vyjádřit:
   která osoba je majitelem domu
   která osoba (které osoby) bydlí v daném domě
   osoba, která je majitelem nějakého domu, nemusí bydlet ve vlastním domě
   Uvažujte dva typy entit a dva typy vztahů. Nakreslete typový ER diagram.

---

4. Je možné ztotožnit entitu (instanci entitního typu) se souhrnem hodnot jejích atributů?

---

5. Zapište dvojici studijních předpisů, vyjadřující 1:N vztah JE UČEN ve směru od PŘEDMĚTU k UČITELi. Vyjádřete tvrzení o determinaci, jestliže identifikačním klíčem UČITELe je JMÉNO UČITELE a identifikačním klíčem PŘEDMĚTU je KÓD PŘEDMĚTU.

---

6. Pro každou dvojici verbálních pravidel identifikujte dva entitní typy a jeden typ vztahu. Ve všech případech stanovte poměr a parcialitu vztahu.
   a) Oddělení zaměstnává libovolné množství osob.
      Osoba je zaměstnána maximálně v jednom oddělení.
   b) Vedoucí řídí maximálně jedno oddělení,
      Oddělení má nanejvýš jednoho vedoucího.
   c) Každý autor může napsat různé množství knih,
      Kniha může být napsána více autory.
   d) Družstvo se skládá z hráčů,
      Hráč hraje pouze za jedno družstvo.
   e) Učitel vyučuje maximálně jednomu předmětu,
      Předmět je vyučován právě jedním učitelem.
   g) Objednávka zboží může být na více výrobků,
      Výrobek se může objevit na více objednávkách,
   h) Zákazník může předložit řadu objednávek,
      Každá objednávka je právě od jednoho zákazníka.

---

7. Praktický lékař má ve své kartotéce mnoho pacientů, ale konkrétní pacient se může zaregistrovat vždy pouze u jednoho praktického lékaře. Jestliže konceptuální schéma zahrnuje pouze aktuální registrace pacientů, jaký je stupeň vztahu LÉKAŘ-PACIENT mezi entitními typy LÉKAŘ a PACIENT? Nakreslete E-R diagram a příklad E-R výskytového diagramu za předpokladu, že identifikační klíče jsou JMÉNO_LÉKAŘE a ČÍSLO_PACIENTA.

8. Jak se změní odpověď na otázku 7, jestliže schéma rozšíříme o možnost registrovat vedle současných i dřívější registrace?

---

9. Jak se změní odpověď na otázku 7, jestliže schéma rozšíříme a dáme pacientovi možnost současně se registrovat u více lékařů?

---

10. Stanovte vhodné typy členství pro entitní typy v těchto případech:

Entitní typy	vztah
a) DŮM, OSOBA	VLASTNICTVÍ
b) DŮM, NÁJEMNÍK	OBÝVÁ
c) DŮM, OSOBA	OBÝVÁ
d) OBJEDNÁVKA, POLOŽKA-OBJEDNÁVKY	OBSAHUJE
Poznámka: objednávka může sestávat z více položek	
e) ZÁKAZNÍK-BANKY, BANKOVNÍ-ÚČET	MÁ-PŘIDĚLEN
f) ZAMĚSTNANEC, KVALIFIKAČNÍ-STUPEŇ	MÁ

---

11. Nakreslete E-R diagram a jeden výskytový E-R diagram pro všechny případy z cvičení 10. Není třeba volit identifikační klíče.

---

12. Tabulka zobrazuje momentální situaci, které stavební prvky jsou začleněny do kterých stavebních modulů

id . prvku	id. modulu
P1	M15
P1	M29
P1	M32
P2	M12
P2	M15
P3	M12
P3	M32

a) Nakreslete výskytový E-R diagram popisující konkrétní situaci z tabulky.

b) Nakreslete E-R diagram vyjadřující jednoduchý vztah mezi entitními typy PRVEK a MODUL.

c) Proveďte dekompozici vašeho diagramu na ekvivalentní diagram, který bude obsahovat pouze 1:N vztahy.

---

13. Frekventanti rekvalifikačních kursů jsou rozděleni do studijních skupin. Každou skupinu může učit několik učitelů. Každý učitel může učit několik skupin. Jedna skupina vždy používá stejnou učebnu (např. skupina S1 vždy užívá místnost M12). Vzhledem k tomu, že skupiny mohou chodit na fakultu v jiných dobách, může více skupin mít přidělenou stejnou místnost.
    a) Nakreslete E-R diagram zobrazující entitní typy UČITEL, SKUPINA, MÍSTNOST a vztahové typy UČITEL-SKUPINA a MÍSTNOST-SKUPINA.
    b) Překreslete váš diagram tak, aby výsledek obsahoval pouze vztahy typu 1:N

---

14. V kontextu s právě řečeným nám diagram připomíná možnost, že vznikl dekompozicí vztahu M:N mezi mezi entitními typy KATEDRA a FAKULTA. Je to pravda?

---

15. Které z těchto tvrzení je pravdivé?
    a) Každý M:N vztah může být rozložen na dva vztahy 1:N,
    b) struktura X(1:N)Y(N:1)Z znamená, že existuje vztah M:N vztah mezi X a Z.

---

16. Porovnejte vztah ISA mezi podtypem a nadtypem s identifikačním vztahem mezi slabou entitou a jejím identifikačním vlastníkem.

---

17. Údaje sledované o zaměstnancích zahrnují číslo zaměstnance, jméno, příjmení, adresu, datum narození, pracovní zařazení, datum zařazení, roční příjem, měsíční plat, kvalifikační stupeň. Požaduje se sledování historie pracovního zařazení včetně data uvedení do funkce. U zaměstnance se sleduje jeden roční plat, ale až 12 měsíčních výplat, které představují částky vyplacené v posledních dvanácti měsících po daňových srážkách. Zaměstnanec mohl získat několik kvalifikačních stupňů. Definujte entitní typ ZAMĚSTNANEC
    a) připouští-li model vícehodnotové a skupinové atributy.
    b) nepřipouští-li model vícehodnotové ani skupinové atributy.

---

18. JEDNODUCHÁ UNIVERSITA

    Uvažujme jednu universitu s několika fakultami. Každý student studuje právě na jedné fakultě. Má jméno, rodné číslo, studentské číslo. Zaměstnanci fakulty jsou organizováni na katedrách daného názvu a čísla. Mají kromě jména a rodného čísla i zaměstnanecké číslo a funkci (na fakultě). Zaměstnanci vypisují přednášky. Ne každý však musí vypsat v daném roce přednášku. Přednášky jsou dány v rámci fakulty kódem (tj. mohou mít stejné názvy), konají se v daný den, hodinu v dané místnosti. Studenti se zapisují na přednášky a vykonávají z nich zkoušku s daným hodnocením.

    a) Navrhněte E-R diagram s odpovídajícími IO. Další (explicitní) IO zapište v přirozeném jazyce.
    b) Diskutujte případy, kdy uvažujeme katedru jako entitní typ nebo jako atribut. V jakých případech se použije jedno nebo druhé řešení ?
    c) Uvažujte funkce DOCENT, PROFESOR, ODBORNÝ_ASISTENT. Profesoři mohou zaměstnávat pomocné vědecké síly (studenty) na řešení projektů, Docent nebo profesor může být vedoucím projektu (je dán číslem a názvem). Vytvořte odpovídající kategorie zaměstnanců jako ISA-hierarchii.
    d) Navrhněte pro příklad JEDNODUCHÁ UNIVERSITA alespoň jeden unární a ternární typ vztahu.

---

19. Namodelujte zadání, které jste navrhli ve cvičení ke kapitole 1, případně též zadání Vašeho spolužáka na základě vykonané předběžné analýzy.

Příklady 2. – 18. byly převzaty z [1].

4 Relační model dat

Cíl kapitoly

Kapitola seznamuje čtenáře s relačním modelem dat jako představitelem logického modelu databáze. Na relačním modelu je postavena většina současných databázových strojů, z relačního modelu vychází též jazyk SQL, proto je důležité se s ním dobře seznámit. Cílem je naučit se:

- co to je relace
- jaký je vztah relace k tabulce
- souvislost relace a entitního typu E-R modelu
- jak se zajišťují základní integritní omezení, jako je klíč či referenční integrita

Klíčové pojmy

logický model model, relace, n-tice, integritní omezení, klíč, nadklíč, referenční integrita.

Motivace

Relační model dat (RMD) je zástupcem druhé vrstvy databázového modelování, tj. logického modelu:

- nižší úroveň abstrakce než konceptuální model
- vyšší úroveň abstrakce než fyzický model

RMD řeší logické struktury, v nichž jsou data uložena, vlastní implementaci těchto struktur ponechává na fyzickém modelu. Řeší rovněž manipulaci s daty ve strukturách, tj. operace nad daty.

RMD se objevil v roce 1970 v článku E.F. Codda. Odděluje data, chápaná jako relace, od jejich implementace, při manipulaci s daty se nezajímáme o mechanismy přístupu k datovým prvkům. Existují dva prostředky manipulace s daty:

- relační algebra
- relační kalkul

Zároveň existují pojmy vedoucí k tzv. normalizaci relací.

Relace

Matematicky:

Mějme systém neprázdných množin (domén)

$D_i, 1 \leq i \leq n$

Podmnožina R kartézského součinu

$R \subseteq D1 \times D2 \times D3 \times ... \times Dn$

se nazývá n-ární relace (n – řád relace), relace je tedy množina. Prvky této relace jsou uspořádané n-tice

$(d1 , d2 , d3 , ... , dn)$ kde $di \in Di$

V databázích nemá smysl uvažovat nekonečné relace, proto se omezíme na konečné podmnožiny kart. součinu.

Jednodušeji
Relaci si představíme jako tabulku s konečným počtem řádků a s *n* sloupci, kde každému prvku relace odpovídá jeden řádek. Buňky ve sloupci *k* mohou nabývat pouze hodnot z domény Dk. Sloupcům přiřadíme (různá) jména Ai.

Schema relace R odpovídá záhlaví tabulky můžeme znázornit lineárním zápisem

R(A1:D1, A2:D2, ..., An:Dn) anebo bez domén
R(A1, A2, ..., An) anebo
R(**A**), kde **A** = {A1, A2, ..., An}

Rozdíly mezi tabulkou a relací
- tabulka může obsahovat stejné řádky, relace nikoli (je to množina, jeden prvek je v ní jednou)
- v relaci nejsou prvky uspořádané, tabulka má ale vždy nějak uspořádané řádky

Je tedy nutná jistá obezřetnost při zacházení s pojmy relace a tabulka, neboť nejsou ekvivalentní.

Nástin souvislosti relace s entitou v E-R modelu
- detailně bude popsáno v dalším

schema relace ~ entitní typ
prvek relace (n-tice) ~ výskyt entity
prvek n-tice ~ atribut dané entity

Souvislost tabulky s entitou
záhlaví tabulky ~ entitní typ
řádek tabulky ~ výskyt entity
sloupec tabulky ~ atribut

Relace odpovídá entitnímu typu lépe než tabulka:
- unikátnost prvků (evidence osob: dvojníci?)
- neuspořádanost prvků

1. normální forma
Normální forma (NF) je nějaké omezení struktury a závislostí mezi jednotlivými atributy.
1. NF:
- každý datový prvek (hodnota buňky v tabulce) je atomický, tj. dále nedělitelný
  - srovnej speciální typy atributů (vícehodnotový a skupinový) v konceptuálním modelu
- základní předpoklad RMD, bohužel někdy je omezující
  - řešením jsou např. objektové nebo objektově-relační databáze

Integritní omezení (IO)

Schematem relace popisujeme pouze datovou strukturu, musíme tedy zajistit, aby se do relací dostala pouze správná data. K tomu slouží integritní omezení, představující obvykle logické podmínky na data. Relace vyhovující všem IO se nazývá přípustná.

Základní IO

- unikátnost n-tic (u množinového pojetí není třeba zdůrazňovat – je implicitní, ale u tabulek ne)
- rozsah hodnot atributu je dán typem (doménou)
- existence klíče schematu

Klíč schematu relace R(A)

Klíč schematu relace R(**A**) je definován jako minimální množina atributů z **A**, jejichž hodnoty jednoznačně určují prvky relace R. Minimalita znamená, že nelze odebrat žádný atribut, aniž by to narušilo identifikační schopnost. Jinými slovy, máme-li dvě různé n-tice z přípustné relace R, pak existuje alespoň jeden klíčový atribut, jehož hodnota se pro tyto dvě n-tice liší. Klíčů může být potencionálně víc, vybíráme jeden, který označujeme jako primární; v nejhorším případě jsou klíčem všechny atributy relace R. Je-li klíčem jediný atribut, nazývá se jednoduchý, ostatní jsou složené. Atribut, který je součástí nějakého klíče, se nazývá klíčový, jinak je neklíčový. To, že množina nemůže obsahovat duplicitní prvky, je přímo důsledkem definice klíče. SQL podporuje tzv. kolekci n-tic, kde jsou povoleny duplicitní řádky, to ale není relace, na relaci to převedeme klauzulí DISTINCT, viz další přednášky. O problematice volby klíče pojednává detailně též komentář v kapitole 3.

Připojíme-li ke klíči libovolný další atribut, naruší se minimalita, ale nikoli identifikační schopnost, takové množině atributů říkáme **nadklíč**. Nadklíč je tedy buď sám klíč anebo množina atributů, která klíč obsahuje.

Definice IO

Mnohá IO lze definovat na úrovni DB, není třeba se spoléhat na aplikační programy. Převod IO konceptuálního schematu (např. kardinalita, parcialita) do logického modelu bude popsán v dalších přednáškách. Jedním z IO v RMD, které tento převod podporuje je **referenční integrita**, která popisuje vztah mezi daty ve dvou relacích, např. most musí ležet na nějaké silnici. Relace MOST má atribut (skupinu atributů) – cizí klíč, jehož hodnota musí být přítomna v relaci SILNICE jako primární klíč.

Příklad referenční integrity

Most

číslo_m	nosnost	šířka	technologie	číslo_s
2469	55	11	beton	21
8963	40	8	beton	21
8965	42	9	beton	602
500489	3,5	4	kámen	1314

klíč

cizí klíč

Silnice

číslo_s	třída	start	cíl	vlastník	délka	oprava
21	1	D5	Německo	stát	62	12.6.2009
53	1	Znojmo	Pohořelice	stát	38	30.5.2007
230	2	Nepomuk	Bečov nad Teplou	Plzeňský kraj	88	14.2.1999
276	2	Bělá pod Bezdězem	Kněžmost	Liberecký kraj	21	21.8.2003
403	2	Kouty	Telč	kraj Vysočina	36	2.10.2001
602	2	Pelhřimov	Starý Lískovec	kraj Vysočina	108	5.8.2008
0395	3	Kostelec	Cejle	kraj Vysočina	2,5	28.9.1992
1314	3	Štoky	Smrčná	kraj Vysočina	6,3	10.7.2000

klíč

Ošetření IO

- deklarativní na straně databáze – ideální, avšak pro koncového uživatele nestačí, nepohodlné
- procedurální na straně klienta
  - kontrola probíhá na úrovni aplikace, potencionální zdroj chyb
  - doplněk k prvnímu způsobu
- procedurální na straně serveru
  - moduly, uložené v databázi, které provádí server
  - triggery – procedury vázané na jisté události v DB

Kontrolní otázky

- Může být v relaci více klíčů?
- Musí mít relace klíč?
- Může relace obsahovat více stejných prvků?
- Může mít každý prvek relace jinou strukturu?
- Co to je integritní omezení?
- Co to je referenční integrita?

5 Relační algebra

Cíl kapitoly

Kapitola pracuje s relačním modelem, zavedeným v předchozí kapitole. Nad relacemi jsou zavedeny operace relační algebry, jejichž smyslem je získávat z relací data. Snahou je, aby bylo možné pomocí relační algebry zodpovědět jakýkoli dotaz na data, uložená v soustavě relací. Cílem je naučit se:

- co to je relační algebra obecně, jaké jsou základní ideje
- množinové operace a jejich význam pro dotazování
- základní operace rel. algebry a jejich význam pro dotazování, zejména
    - projekce
    - selekce
    - přirozené spojení a další varianty spojení
- rozšíření relační algebry – vnější spojení

Klíčové pojmy

kartézský součin, sjednocení, průnik, množinový rozdíl, projekce, selekce, přirozené spojení, Θ-spojení, polospojení, vnější spojení.

Manipulace s relacemi

Zatím máme struktury (relace), ale nemáme operace nad nimi. Jsou nutné prostředky především pro aktualizaci relací a dotazování. Aplikujeme množinový přístup, kdy vstupem operace jsou celé relace a výstupem operace je opět relace.

Aktualizace relace

- přidání prvku do množiny (INSERT)
- odebrání prvku z množiny (DELETE)
- změna prvku v množině (UPDATE)

Operace DELETE a UPDATE vyžadují identifikaci prvku (prvků), na něž se vztahují. Identifikace probíhá výlučně na základě hodnot atributů v relacích (žádné jiné informace o prvcích v množině totiž nemáme). Operace INSERT a UPDATE sledují unikátnost prvků, zejména aby nedošlo k porušení unikátnosti primárního klíče.

Základní ideje RMD z hlediska operace s daty

- 1. NF – komponenty n-tic jsou atomické
- přístup k prvkům relace výhradně podle jejich obsahu, tj. nelze třeba říct, že odstraníme třetí řádek tabulky (SŘBD sice mají identifikátory řádků tabulek, ale ty by se neměly v běžné praxi používat, pouze při správě DB)
- jedinečnost n-tic
- množinový přístup

Relační algebra

Relační algebra je množina operací, jejichž aplikace na nějaké relace vrací opět relaci, výsledek by tudíž měl mít definováno schéma. Relace jsou množiny, máme proto k dispozici běžné množinové operace:

- součin (×)
- sjednocení ($\cup$)
- průnik ($\cap$)
- rozdíl (–)

Na součin se neklade žádné omezení, ostatní operace jsou možné jen s kompatibilními operandy, tj. obě relace musí mít stejný počet atributů a rovnost odpovídajících si domén.

Daná operace je popsána nějakým výrazem, např. $(R \cap S) \times W$, je třeba rozlišit pojmy dotaz a výraz. Dotaz je funkce na stavech databáze, výraz je pouze syntaktický zápis dotazu. Týž dotaz může být tedy popsán více výrazy. Jestliže dva různé výrazy označují tentýž dotaz, jsou **ekvivalentní**. Např. v algebře je ekvivalentní $(x * x)$ a (x^2). Ekvivalentní výrazy jsou také $(R \cap S) \times W$ a $(R - (R - S)) \times W$.

Další operace

- přejmenování atributů (například při kartézském součinu relace sama se sebou)
- projekce R[C] relace se schematem R(**A**) na množinu atributů C, kde $C \subseteq \mathbf{A}$
- selekce $R(\varphi)$ relace se schematem R(**A**) podle logické podmínky φ
- spojení relací R a S se schematy R(**A**) a S(**B**) R * S

Projekce

Projekci R[C] relace se schematem R(**A**) na množinu atributů C, kde $C \subseteq A$, lze popsat pomocí projekce n-tice u[C], což je operace, která z n-tice u vybere pouze atributy, patřící do množiny C. Potom je-li R relace, pak $R[C] = \{ u[C] \mid u \in R \}$. Tato projekce tedy vytvoří relaci se schematem C a n-ticemi, které vzniknou z původní relace odstraněním hodnot atributů z $A - C$. Odstraněny jsou i event. duplicitní n-tice. Přijmeme-li na chvilku analogii relace a tabulky, jde o „svislý řez“ tabulkou.

Příklad

R(X, Y, Z)

X	Y	Z
1	2	3
1	2	4
2	5	9

R[C], C = {X, Y}

X	Y
1	2
2	5

Selekce

Selekci R(φ) relace se schematem R(**A**) podle logické podmínky φ lze definovat takto: je-li R relace, pak R(φ) = { u | u ∈ R ∧ φ(u)}. Operace vytvoří relaci s týmž schematem a pouze těmi n-ticemi, které splňují podmínku φ. Podmínka je zadána boolským výrazem, používáme obvyklé spojky and, or, not. Přijmeme-li na chvilku analogii relace a tabulky, jde o „vodorovný řez" tabulkou.

Příklad

R(X, Y, Z)

X	Y	Z
1	2	3
1	2	4
2	5	9

R(φ), φ: X < 2

X	Y	Z
1	2	3
1	2	4

Spojení

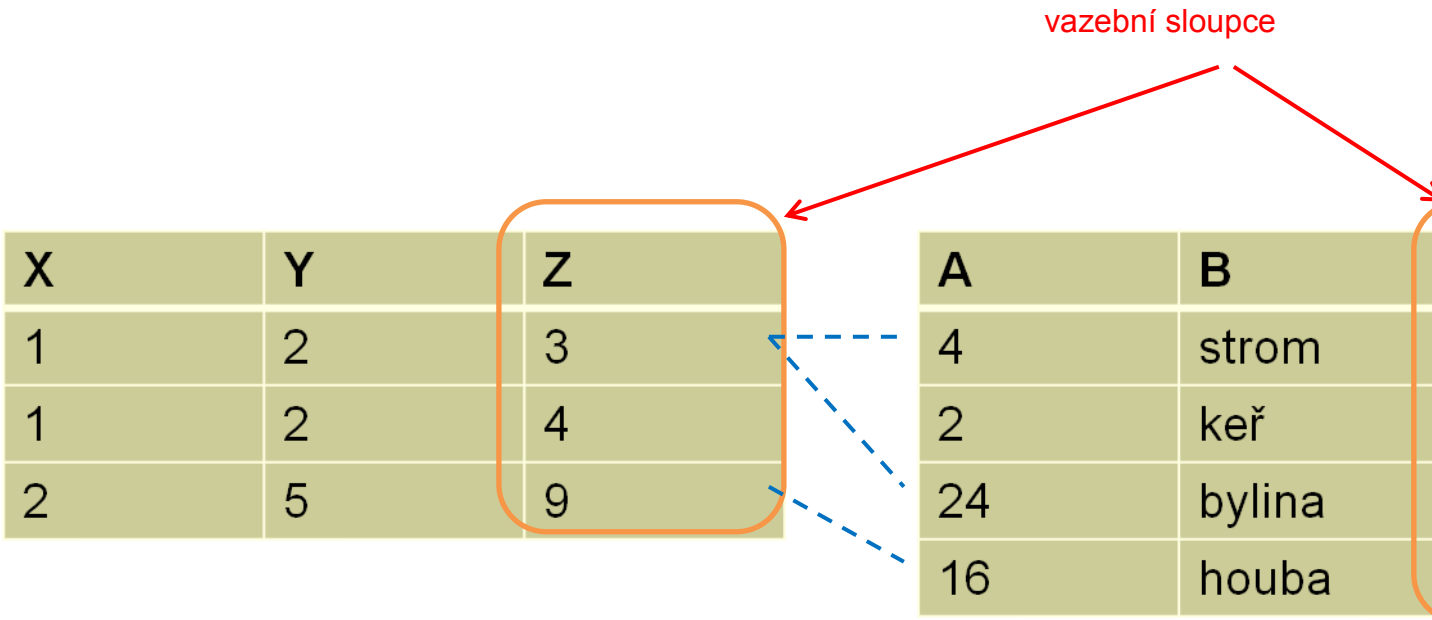
Spojení relací R, S se schematy R(**A**), S(**B**) je definováno takto: jsou-li R a S relace, pak spojení relací

$$R * S = \{ u \mid u[\mathbf{A}] \in R \ \wedge \ u[B] \in S \}$$

Operace vytvoří největší relaci se schematem **A** ∪ **B** a n-ticemi, jejichž projekce na **A** je z relace R a projekce na **B** je z relace S. Tato operace se nazývá přirozené spojení. Společné atributy (**A** ∩ **B**) jsou zastoupeny pouze jednou – tzv. **vazební sloupce**. Do výsledku spojení jdou pouze ty n-tice z R, k nimž se v S najde n-tice se stejnými hodnotami společných atributů.

Je-li (**A** ∩ **B**) prázdná množina (tedy neexistují žádné vazební sloupce), vytvoří se kartézský součin, kdy se každá n-tice z R spojí z každou n-ticí z S, ve výsledku je tedy |R|*|S| prvků, kde |M| značí mohutnost množiny M.

Příklad
R(X, Y, Z), S(A, B, Z)



X	Y	Z
1	2	3
1	2	4
2	5	9

A	B
4	strom
2	keř
24	bylina
16	houba

R * S – spojíme ty řádky z tabulek, které mají stejný obsah vazebního sloupce

X	Y	Z	A	B
1	2	3	4	strom
1	2	3	24	bylina
2	5	9	16	houba

Příklad 2 – přirozené spojení R a S
R(X, Y), S(A, B)

X	Y
1	2
3	4

A	B
hrušeň	strom
tavolník	keř

R * S = R × S

X	Y	A	B
1	2	hrušeň	strom
1	2	tavolník	keř
3	4	hrušeň	strom
3	4	tavolník	keř

Příklad na dotaz

Mějme relace

KINO(název_k, adresa)

FILM(jméno_f, herec, rok)

MÁ_NA_PROGRAMU(název_k, jméno_f, datum)

Dotaz: najdi herce, kteří hrají ve filmech, dávaných v kině Dukla.

Řešení:

(MÁ_NA_PROGRAMU(název_k='Dukla')[jméno_f] * FILM) [herec]

Další užitečné relační operace

Θ-spojení (spojení přes podmínku) relací R(**A**), S(**B**), kde $\Theta \in \{<, >, =, \leq, \geq, \neq\}$, je definováno jako

$$R\ [t1\ \Theta\ t2]\ S \quad = \quad \{ u \mid \quad u[\mathbf{A}] \in R \quad \wedge \quad u[\mathbf{B}] \in S \quad \wedge \quad u.t1\ \Theta\ u.t2\}$$

Výsledná relace bude mít schema obsahující atributy z R i S (vč. duplicitních atributů), na jejich prvcích bude splněna podmínka daná v závorkách []. Kolize jmen atributů řešíme např. přejmenováním.

Příklad

R(X, Y, Z), S(A, B, Z)

X	Y	Z
1	2	3
1	2	4
2	5	9

A	B	Z
4	strom	3
2	keř	8
24	bylina	3
16	houba	9

R [Y ≥ A] S

X	Y	R.Z	A	B	S.Z
2	5	9	4	strom	3
1	2	3	2	keř	8
1	2	4	2	keř	8
2	5	9	2	keř	8

Θ -spojení je ekvivalentní kartézskému součinu R a S s následnou selekcí podle podmínky v závorkách. Často používané je **spojení přes rovnost**, jehož speciálním případem je přirozené spojení (tam se vyžaduje rovnost všech odpovídajících si atributů). Spojení přes rovnost umožňuje spojovat přes atributy se stejnou doménou, ale různým jménem, umožňuje též spojovat přes jinou množinu atributů než v přirozeném spojení.

Levé Θ -polospojení (semijoin) relací R(**A**), S(**B**) je definováno jako

$$R <t1\ \Theta\ t2]\ S = (R\ [t1\ \Theta\ t2]S)[A]$$

Definice nemá naznačit způsob implementace. Použití je např. v distribuovaných DB, aby se nemusely přenášet celé relace. Polospojení je vlastně omezení relace R na ty prvky, které jsou spojitelné s prvky relace S podle zadané podmínky. Nejčastější je polospojení přes rovnost, lze definovat též přirozené polospojení. Analogicky k levému zavedeme pravé polospojení. Lze definovat také **přirozené polospojení**, kde podmínka je rovnost všech odpovídajících si atributů, značíme R <* S resp. R *> S.

Příklad

R(X, Y, Z), S(A, B, Z)

X	Y	Z
1	2	3
1	2	4
2	5	9

A	B	Z
4	strom	3
2	keř	8
24	bylina	3
16	houba	9

R <Y = A] S

X	Y	Z
1	2	3
1	2	4

Rozšíření relační algebry

Operace relační algebry lze rozšířit o další operace, které do rel. algebry nepatří, ale implementačně dávají smysl. Hlavním rozšířením je **vnější spojení** (outer join), které uvažuje prázdné hodnoty buněk (proto není v RMD, neboť tam prázdné hodnoty neexistují). Prázdná hodnota NULL patří do všech domén (!) a značí, že hodnota atributu je nedefinovaná, neznámá. Prázdná hodnota vede k tříhodnotové logice a k různým obtížím. Vnější spojení si můžeme představit jako běžné (vnitřní) spojení, které navíc k výsledku přidá i řádky nespojitelné s ničím. Rozlišujeme levé, pravé a úplné vnější spojení.

Levé vnější spojení

Levé vnější spojení relací R(**B**), S(**B**):

- projekce na atributy **A** je relace R, projekce na atributy **B** je podmnožina relace S a řádek prázdných hodnot

  $$R *_L S = (R * S) \cup (R' \times (NULL, \ldots, NULL))$$

  kde R‘ značí n-tice z R nespojitelné s S

Pravé vnější spojení

Pravé vnější spojení relací R(**B**), S(**B**):

- projekce na atributy **A** je podmnožina relace R a řádek prázdných hodnot, projekce na atributy **B** je relace S

  $R *_R S = (R * S) \cup ((NULL, \dots, NULL) \times S')$

  kde S' značí n-tice z S nespojitelné s R

Úplné vnější spojení

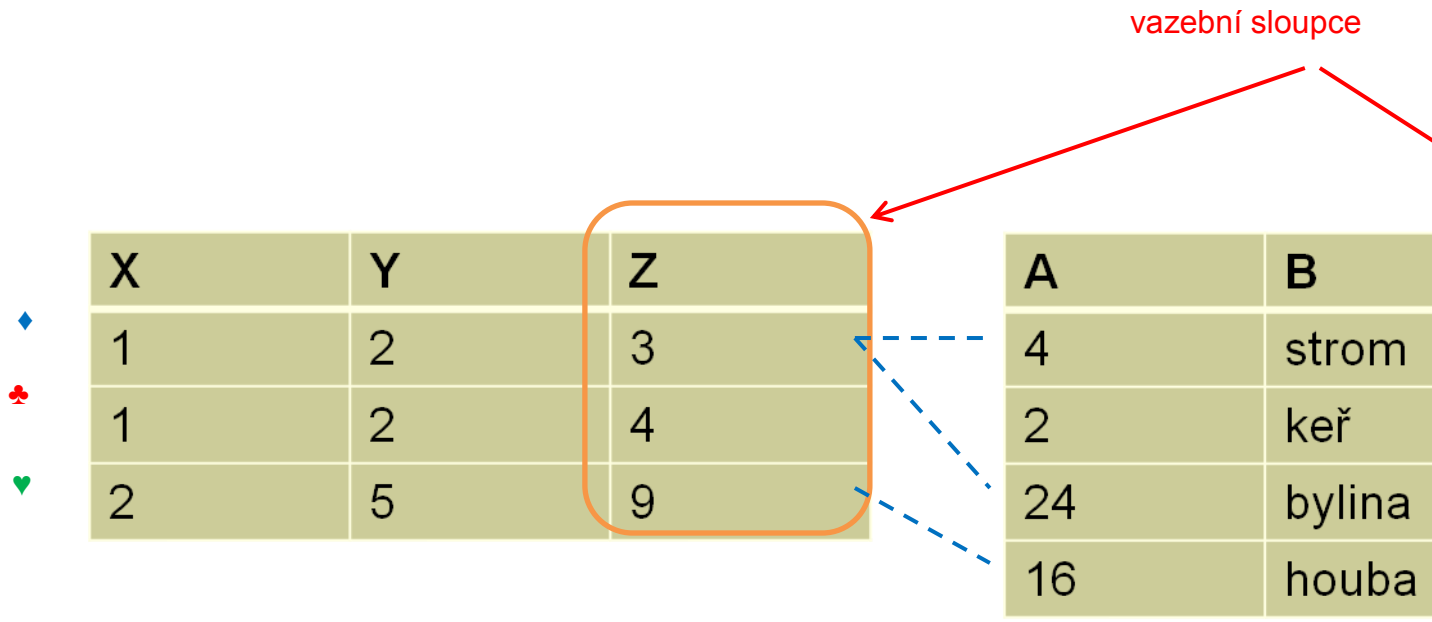
Úplné vnější spojení relací R(**B**), S(**B**):

- projekce na atributy **A** je podmnožina R a řádek prázdných hodnot, projekce na atributy **B** je podmnožina S a řádek prázdných hodnot

  $R *_F S = (R *_L S) \cup (R *_R S)$

Příklad

R(X, Y, Z), S(A, B, Z)



X	Y	Z
1	2	3
1	2	4
2	5	9

A	B
4	strom
2	keř
24	bylina
16	houba

$R *_L S$

X	Y	Z	A	B
1	2	3	4	strom
1	2	3	24	bylina
2	5	9	16	houba
1	2	4	NULL	NULL

Závěrem

Při zápisu výrazů relační algebry je vždy nutné dávat pozor na pořadí operací a prioritu operátorů.
Příklad: najdi herce, kteří hrají ve filmech, dávaných v kině Dukla.
Možné řešení:

(MÁ_NA_PROGRAMU(název_k='Dukla') * FILM) [herec]

Chybné řešení (projekce na atribut „herec" se v relaci film provede před spojením, tj. vypadne vazební sloupec):

MÁ_NA_PROGRAMU(název_k='Dukla') * FILM [herec]

Kontrolní otázky

- Patří zduplikování prvku relace mezi operace relační algebry?
- Jaký je rozdíl mezi dotazem a výrazem?
- Jakým způsobem vybíráme ke zpracování požadované prvky relace?
- Přijmeme-li na chvilku analogii relace a tabulky, jak lze intuitivně popsat projekci resp. selekci relace?
- Jaký je rozdíl mezi vnitřním a vnějším spojením?

Řešené příklady

Příklad 1

Mějme relace se schematy R(a,b,c) a S(d,e,f). Co bude výsledkem přirozeného spojení R * S? Jaké bude mít výsledná relace T schema?

Řešení:

V tomto případě bude výsledkem kartézský součin S × R, neboť relace R a S nemají žádné společné vazební atributy. Výsledné schema bude T(a,b,c,d,e,f).

Příklad 2
Mějme relace R(a,b,c) a S(b,c,d). Jaké schema bude mít relace T vzniklá přirozeným spojením R * S?

Řešení:
Výsledek přirozeného spojení má schema dané sjednocením množin atributů spojovaných relací. Tedy vznikne relace se schematem T(a,b,c,d).

Příklad 3
Mějme relace R(a,b,c) a S(b,c,d). Jaké schema bude mít relace T vzniklá pravým přirozeným polospojením R *> S? Definujte pravé přirozené polospojení pomocí přirozeného spojení a projekce.

Řešení:
Výsledek přirozeného polospojení má schema dané množinou atributů pravé (pro pravé polospojení) nebo levé relace (pro levé polospojení). Zde tedy vznikne relace se schematem T(b,c,d). Polospojení

R *> S = (R * S) [b,c,d].

Příklad 4
Mějme relace R(a,b,c) a S(b,c,d). Můžeme použít operaci R – S (tj. množinový rozdíl)? Můžeme použít operaci R $\cap$ S (průnik)? Můžeme použít operaci R $\cup$ S (sjednocení)? Můžeme použít operaci R $\times$ S (kartézský součin)?

Řešení:
Můžeme použít pouze kartézský součin. Ostatní jmenované operace nelze použít, neboť relace nejsou kompatibilní – mají rozdílné množiny atributů.

Příklad 5
Mějme relace R(a,b,c), S(b,c,d) a T(d,e,f). Liší se nějak relace dané následujícími operacemi (uvažujeme pořadí vyhodnocování operandů zleva)?

U1 = R * S * T
U2 = R * T * S
U3 = R * (T * S)

Řešení:
U1 = R * S * T = (R * S) * T = R * (S * T) = R * (T * S) = U3
U2 = R * T * S = (R * T) * S = (R $\times$ T) * S $\neq$ U1 $\neq$ U2

Relace U1 a U3 jsou identické, relace U2 se obecně může lišit.

Příklad 6 (*Převzato z [1]*)
Uvažujme tři schémata relací:

KINO(NÁZEV K, ADRESA),
FILM(JMÉNO_F, HEREC, ROK)
PROGRAM(NÁZEV K, JMÉNO F, DATUM).

První schéma popisuje kina v městě, ve druhé relaci je nabídka filmů spolu s herci, kteří v nich hrají, a rokem, kdy film vznikl. Třetí relace popisuje program kin (např. v rámci jednoho měsíce). Jako IO by mohla být použita např. tato tvrzení:

IO1: V kinech se nehraje více něž dvakrát týdně.
IO2: Jeden film se nedává více než ve třech kinech v městě.

Některá IO plynou z definice klíče. Patří k nim např.

IO3: Jeden film nemohou v jednom kině dávat vícekrát. Např. hodnota klíče (Blaník, Top Gun) určuje pouze jedno datum.

KINO	NÁZEV_K	ADRESA
	Blaník	Václ.n. 4
	Vesna	V olšinách 6
	Mír	Starostrašnická 3
	Domovina	V dvorcích

FILM	JMÉNO_F	HEREC	ROK
	Černí baroni	Vetchý	94
	Černí baroni	Landovský	94
	Top gun	Cruise	86
	Top gun	McGillis	86
	Kmotr	Brando	72
	Nováček	Brando	90
	Vzorec	Brando	80

PROGRAM	NÁZEV_K	JMÉNO_F	DATUM
	Blaník	Top gun	29.03.94
	Blaník	Kmotr	08.03.94
	Mír	Nováček	10.03.94
	Mír	Top gun	09.03.94
	Mír	Kmotr	08.03.94

Začneme selekcí PROGRAM(NÁZEV_K = Mír). Výsledek operace je relace R1. Relace R2 tvoří výsledek projekce R1[JMÉNO_F, DATUM].

R1	NÁZEV_K	JMÉNO_F	DATUM
	Mír	Nováček	10.03.94
	Mír	Top gun	09.03.94
	Mír	Kmotr	08.03.94

R2	JMÉNO_F	DATUM
	Nováček	10.03.94
	Top gun	09.03.94
	Kmotr	08.03.94

Dále provedeme spojení FILM * R2. Ve výsledku R3 se uplatní pouze označené n-tice (spojí se vždy ty se stejným označením).

FILM	JMÉNO_F	HEREC	ROK
	Černí baroni	Vetchý	94
	Černí baroni	Landovský	94
*	Top gun	Cruise	86
*	Top gun	McGillis	86
-	Kmotr	Brando	72
+	Nováček	Brando	90
	Vzorec	Brando	80

R2	JMÉNO_F	DATUM
+	Nováček	10.03.94
*	Top gun	09.03.94
-	Kmotr	08.03.94

R3	JMÉNO_F	HEREC	ROK	DATUM
	Top gun	Cruise	86	09.03.94
	Kmotr	Brando	72	08.03.94
	Nováček	Brando	90	10.03.94
	Top gun	McGillis	86	09.03.9

V relaci R4 jsou data po operaci R3[HEREC].

R4	HEREC
	Cruise
	Brando
	McGillis

Relační algebru lze použít jako dotazovací jazyk. Řešili jsme dotaz "Nalezni herce, kteří hrají ve filmech v kinu Mír". Bez použití schémat pro mezivýsledky lze posloupnost operací zapsat pomocí jednoho výrazu

(PROGRAM(NÁZEV_K = 'Mír') [JMÉNO_F, DATUM] * FILM) [HEREC]

Všimněte si, že spojení je zapsáno v obráceném pořadí operandů než ve vlastním příkladu. Výsledek operace je týž (operace je totiž komutativní), ovšem čas potřebný k provedení operace může být výrazně odlišný. Implementace operace spojení může totiž preferovat menší relaci jako první operand. Dále atribut DATUM v projekci není pro výsledek podstatný a bylo by možné ho ve výrazu vynechat. Jde ojev s

důsledky ovlivňujícími celou koncepci SŘBD. Interpretuje-li SŘBD dotazy bez jakékoli optimalizace, závisí efektivnost vyhodnocení vlastně na uživateli, což jistě odporuje základním idejím databázových systému.

Příklad 7: (*Převzato z [1]*)
Uvažujme schémata R(A, B, C) a S(B, C, D, E) a relace na obrázku. Pod schématem T je výsledek operace R [A < B] S. Znak * označuje spojované n-tice. Dále je zobrazen výsledek operace R <A < B] S a R <* S.

R	A	B	C
	8	2	3
*	1	2	3
*	1	1	4
	3	6	7
	3	8	9

S	B	C	D	E
*	2	4	2	3
*	2	3	2	3
	1	4	5	6
*	2	3	4	7

R [A < B] S	A	R.B	R.C	S.B	S.C	D	E
	1	2	3	2	4	2	3
	1	2	3	2	3	2	3
	1	2	3	2	3	4	7
	1	1	4	2	4	2	3
	1	1	4	2	3	2	3
	1	1	4	2	3	4	7

R <A < B] S	A	B	C
	1	2	3
	1	1	4

R <* S	A	B	C
	8	2	3
	1	2	3
	1	1	4

Příklad 8
Mějme databázi studentů a předmětů, na které se zapisují a které absolvují. Databáze je realizována třemi relacemi se schematy:

STUDENT(EV_Č, RČ, JMÉNO, SPECIALIZACE)
PŘEDMĚT(KÓD, NÁZEV, SYLLABUS, GARANT)
ZÁPIS(EV_Č, KÓD, SEMESTR, ZNÁMKA)

Napište v relační algebře následující dotazy.

a. Seznam všech studentů.

Řešení: vypíšeme obsah celé relace STUDENT.

```
STUDENT
```

b. Seznam všech specializací studentů.

Řešení: použijeme projekci na relaci STUDENT, tj. výběr pouze některých atributů, v našem případě atributu specializace. Výstupem bude v případě opakujících se specializací pouze jedna specializace za všechny, které se opakují – relace eliminují duplicity. Toto chování je odlišné od chování SQL, viz další příklady.

```
STUDENT[specializace]
```

c. Seznam všech studentů zaměřených na SW inženýrství.

Řešení: použijeme selekci na relaci STUDENT, tedy výběr pouze některých n-tic relace podle logické podmínky. V našem případě bude podmínkou, že specializace je SWI.

```
STUDENT(specializace=SWI)
```

d. Seznam všech předmětů, které negarantuje Bláha.

Řešení: použijeme opět selekci na relaci PŘEDMĚT, podmínkou bude, že garant se nerovná Bláha.

```
PŘEDMĚT(garant<>Bláha)
```

e. Jména všech všech studentů zaměřených na SW inženýrství.

Řešení: použijeme selekci s podmínkou, že specializace je SWI, a následnou projekci na atribut jméno. Pozor, musíme dodržet uvedené pořadí – kdybychom provedli nejdříve projekci, pak bychom již neměli k dispozici atribut specializace, podle kterého potřebujeme vybírat n-tice.

```
STUDENT(specializace=SWI)[jméno]
```

f. Kódy všech předmětů, které si zapsal student s ev_č. 12345.

Řešení: použijeme selekci s podmínkou, že ev_č='12345', a následnou projekci na atribut kód. Stačí používat pouze relaci ZÁPIS, neboť obsahuje všechny požadované informace.

```
ZÁPIS(ev_č='12345')[kód]
```

g. Kódy všech předmětů, které si zapsal student Jan Novák.

Řešení: narozdíl od předchozího příkladu musíme nyní využít spojení relací ZÁPIS a STUDENT, protože jméno studenta již nemáme v relaci ZÁPIS k dispozici. Vazebním atributem bude atribut ev_č, je dán automaticky použitím přirozeného spojení. Další postup je obdobný – vybereme selekcí pouze studenty s daným jménem a nakonec použijeme projekci na atribut kód.

```
( STUDENT * ZÁPIS ) (jméno='Jan Novák') [kód]
```

Jiné řešení. Selekci můžeme předřadit před spojení, neboť nemá vliv na množinu atributů a spojení tedy neovlivní:

```
( STUDENT(jméno='Jan Novák') * ZÁPIS ) [kód]
```

Nesprávné řešení. Důvod: priorita projekce je vyšší než priorita spojení, projekce se tedy provede před spojením, čímž přijdeme o vazební atribut ev_č a výsledkem spojení bude kartézský součin:

```
STUDENT(jméno='Jan Novák') * ZÁPIS[kód]
```

h. Garanti všech předmětů, z nichž byla udělena známka 3.

Řešení: analogicky k předchozímu příkladu.

```
( ZÁPIS(známka=3) * PŘEDMĚT ) [garant]
```

i. Seznam všech předmětů, které si někdo zapsal.

Řešení: jsou to takové předměty, jejichž kód se vyskytuje v relaci ZÁPIS, jinými slovy, jsou to ty předměty, které lze spojit s nějakou n-ticí v relaci ZÁPIS. Jelikož chceme vypsat pouze atributy relace PŘEDMĚT, použijeme buď operaci přirozené polospojení nebo přirozené spojení s následnou projekcí na všechny atributy relace PŘEDMĚT.

```
ZÁPIS *> PŘEDMĚT

( ZÁPIS * PŘEDMĚT ) [kód, název, syllabus, garant]
```

j. Seznam předmětů, které si zatím nikdo nezapsal.

Řešení: předměty, které si nikdo nezapsal jsou takové, které nemají svůj kód obsažen v relaci ZÁPIS. Tato negativní informace se ale relační algebrou nedá zjistit. Musíme tedy použít jistý trik – od množiny všech předmětů odebereme předměty, které si někdo zapsal. Zbydou nám pak právě takové, které si nezapsal nikdo. Můžeme využít výsledku předchozího příkladu.

```
PŘEDMĚT – ( ZÁPIS *> PŘEDMĚT )
```

Pozor, při množinových operacích (vyjma kartézského součinu) je nutná kompatibilita operandů, tj. shodná množina atributů. Není tedy možné psát např.

```
PŘEDMĚT – ( ZÁPIS * PŘEDMĚT )
```

k. Jména všech studentů, kteří si zapsali nějaký předmět garantovaný Bláhou.

Řešení: jména studentů máme v relaci STUDENT, informace o zápisech jsou v relaci ZÁPIS, garanty předmětů máme v relaci PŘEDMĚT. Musíme tedy použít spojení všech tří relací. Následně musíme pochopitelně vybrat jen předměty s požadovaným garantem a zobrazit pouze atribut jméno.

```
( STUDENT * ZÁPIS * PŘEDMĚT )(garant=Bláha)[jméno]
```

Pozor, při spojování více relací musíme dbát na správné pořadí – aby vše fungovalo tak, jak chceme, vždy musí být vedle sebe ty relace, které mají nějaký vazební atribut. Jinak by mohlo dojít namísto spojení ke kart. součinu. Nelze tedy psát např.

```
( STUDENT * PŘEDMĚT * ZÁPIS )(garant=Bláha)[jméno]
```

neboť relace STUDENT a PŘEDMĚT nemají žádný společný atribut.

l. Jména a specializace studentů, kteří si zapsali **pouze** předměty garantované Bláhou.

Řešení: potřebujeme vypsat studenty, kteří si zapsali předměty garantované Bláhou a zároveň si nezapsali žádné jiné předměty. Tedy nejprve zkonstruujeme množinu studentů, kteří si zapsali předměty garantované Bláhou (viz předchozí příklad) a z této množiny odebereme studenty, kteří si zapsali předměty garantované někým jiným než je Bláha.

```
( ( STUDENT * ZÁPIS * PŘEDMĚT(garant= Bláha) ) -
  ( STUDENT * ZÁPIS * PŘEDMĚT(garant<>Bláha) ) ) [jméno, specializace]
```

m. Jména studentů, kteří si nic nezapsali.

Řešení: analogicky k předchozím příkladům. Od množiny všech studentů odebereme ty studenty, kteří si něco zapsali. Zbydou nám pak právě takoví, kteří si nezapsali nic.

```
( STUDENT – ( STUDENT <* ZÁPIS ) ) [jméno]
```

nebo s jiným pořadím operací

```
STUDENT[jméno] – ( STUDENT * ZÁPIS )[jméno]
```

n. Garanti předmětů, které si nezapsal student Jan Bok.

Řešení: analogicky k předchozím příkladům.

```
PŘEDMĚT[garant] – ( STUDENT(jméno='Jan Bok') * ZÁPIS * PŘEDMĚT )[garant]
```

Úlohy k procvičení

1. Vyjádřete přirozené spojení a theta spojení pomocí operací projekce, selekce a součin

---

2. Za jaké podmínky pro relace R(**A**) a S(**B**) platí, že R * S = R ∩ S?

---

3. Jsou následující výrazy ekvivalentní?

   $R(\varphi_1)(\varphi_2)$

   $R(\varphi_2)(\varphi_1)$

   $R(\varphi_1$ **and** $\varphi_2)$

---

4. Nechť R(**A**) a S(**B**) jsou schémata relací. Ukažte, že označují následující výrazy stejný dotaz.

   (R * S)[A]

   R[A] * S[A $\cap$ B]

   R * S[A $\cap$ B]

   Použití jaké jiné operace jsou tyto výrazy ekvivalentní?

Úlohy 1. – 4. převzaty z [1]. Další úlohy na relační algebru najdete na konci následující kapitoly.

6 Transformace E-R schematu do RMD

Cíl kapitoly

Kapitola poskytuje propojení mezi konceptuálním E-R modelem a relačním modelem dat. Cílem je naučit se převést libovolný E-R model do relačního modelu a to pokud možno optimálně vzhledem k plánovanému využití a se zobrazením všech integritních omezení.

Klíčové pojmy

Silný a slabý entitní typ, determinant vztahu, identifikační závislost, cizí klíč.

Motivace

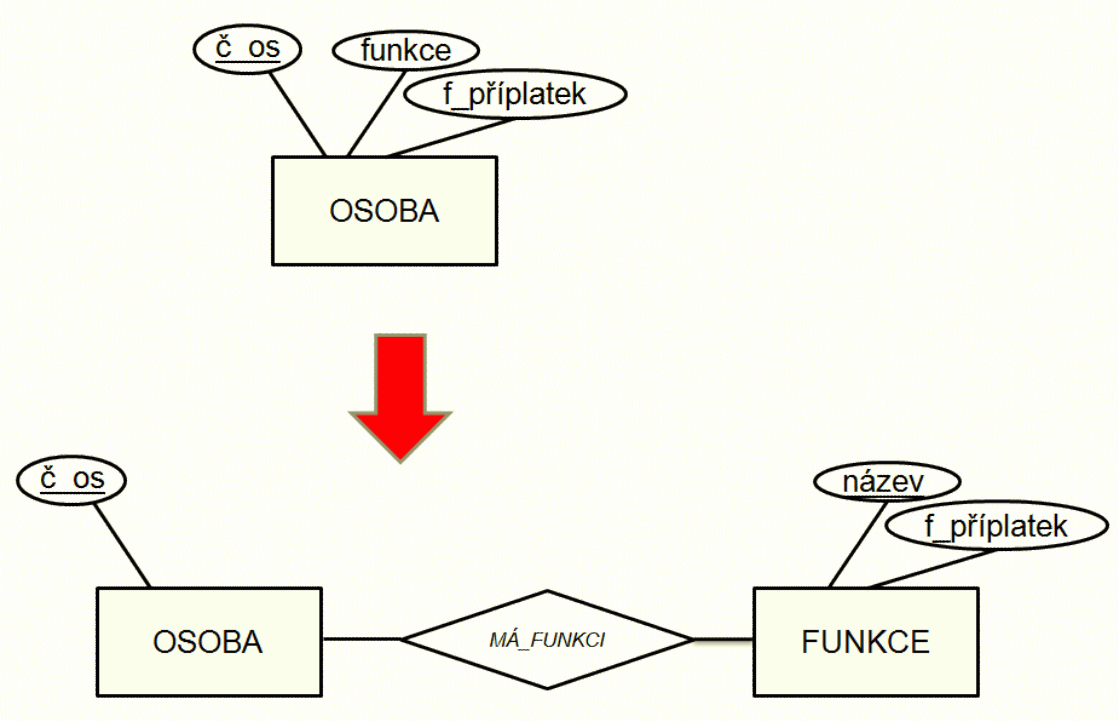
Výsledkem konceptuálního modelování je E-R model, který může být dost vzdálený logickému modelu, je nutné převést E-R model do relačního modelu. Transformace zajišťuje reprezentaci základních konstruktů bez ohledu na další vlastnosti (např. NF). RMD reprezentuje entitní a vztahové typy stejně – schematem relace, rekonstrukce E-R modelu z RMD tudíž není intuitivní.

Silný entitní typ

- přímočaré – silnému ent. typu odpovídá schema relace se stejnou množinou atributů
- primární klíč odpovídá identifikačnímu klíči entit. typu
- popisným typům atributů se přiřadí domény
- cíl – normalizovaná tabulka
    - závisí na citu analytika a množství funkčních závislostí (FZ), které uvažujeme
    - mělo by platit, že jediné netriviální funkční závislosti odvoditelné z ent. typu jsou závislosti atributů na identifikačním klíči, odhalení dalších FZ může indikovat další ent. typ

- například mějme
  OSOBA(<u>č_os</u>, funkce, f_příplatek)
    - pokud platí, že funkce→ f_příplatek, zavedeme nový entitní typ FUNKCE(název, příplatek) a vztahový typ MÁ_FUNKCI
    - tyto problémy je nutné řešit již na úrovni konceptuálního schematu, tzv. normalizace

Normalizace



Možnosti transformace vícehodnotových atributů

- některé SŘBD je přímo umožňují
- jestliže víme max. počet výskytů atributu, pak pro ně „rezervujeme“ místo v relaci, nevyužité budou mít například hodnotu NULL
    - zabírá místo
- zavedeme další relaci, odpovídající vícehodnotovému atributu

Možnosti transformace skupinových atributů

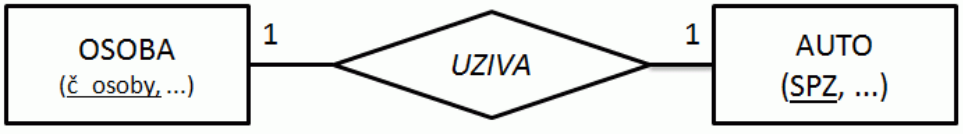
- některé SŘBD je přímo umožňují
- jinak je nutno oželet hierarchickou strukturu atributu, uložíme pouze atomické složky a opět využijeme nové relace

Vztahový typ

Vztah 1:1

Mějme služební vozidla v podniku

- žádný vůz není využíván více zaměstnanci
- žádný zaměstnanec nevyužívá víc než jeden vůz



Reprezentace zde závisí na tom, zda je členství ve vztahu povinné či nikoli (parcialita).

Povinné členství pro oba ent. typy

- každý zaměstnec má právě jedno auto
- každé auto je přiděleno právě jednomu zaměstnanci

- **atributy obou entitních typů zařadíme do jediné relace** – slévání, přilepení atributů, vztah reprezentován implicitně
- klíčem bude buď č_osoby nebo SPZ

OSOBA(č_osoby, ... , SPZ, ...)

Povinné členství pouze pro jeden ent. typ

- každé auto je přiděleno právě jednomu zaměstnanci
- každý zaměstnec má žádné nebo jedno auto

- **dvě relace (VŮZ a OSOBA)**, do relace VŮZ přidáme atribut č_osoby
- klíčem by mohlo být i č_osoby

OSOBA(č_osoby, ...)
VŮZ(SPZ, ..., č_osoby)

Nepovinné členství pro oba ent. typy

- každé auto přiděleno žádné nebo jedné osobě
- každý zaměstnec má žádné nebo jedno auto

- nelze přilepit atributy ani k vozu, ani k osobě, anebo bychom museli jedině připustit prázdné hodnoty (v SQL toto lze)
- **vytvoříme třetí (vztahovou) relaci UŽÍVÁ**
    - atributy odpovídající identifikačním klíčům obou e. typů
    - klíčem nové relace může být č_osoby nebo SPZ

OSOBA(č_osoby, ...)
VŮZ(SPZ, ...)
UŽÍVÁ(č_osoby, SPZ)

Co když má vztah atributy?

- v prvních dvou případech se přilepí tam, kde jsou klíče obou relací (č_osoby i SPZ)
- ve třetím případě se přidají do nové vztahové relace
- optimální reprezentace by měla také respektovat funkcionalitu
    - například když se agenda autoparku příliš nevyužívá, můžeme si dovolit více menších relací (pružnější odezva např. při dotazech na zaměstnance), dotaz na autopark bude pomalejší (náročná operace spojení)

Vztah 1:N

- ent. typ PACIENT je determinantem ent. typu POKOJ, opačně to neplatí
- hraje roli pouze parcialita determinantu (PACIENT)

Povinné členství determinantu vztahu

- evidujeme pouze hospitalizované pacienty

- **přilepíme atribut č_pokoje k relaci PACIENT** (tj. k determinantu)

PACIENT(rč, ..., č_pokoje)
POKOJ(č_pokoje, ...)

Nepovinné členství determinantu vztahu

- evidujeme i ambulantní pacienty

- přilepíme-li atribut č_pokoje k relaci PACIENT jako v předchozím případě, musíme připustit prázdnou hodnotu (v SQL možné)
- **zavedení třetí (vztahové) relace UMÍSTĚN**
    - atributy odpovídající identifikačním klíčům obou e. typů
    - klíčem nové relace je klíč determinantu

PACIENT(rč, ...)
POKOJ(č_pokoje, ...)
UMÍSTĚN(rč, č_pokoje)

Vztah M:N

- např. vůz může náležet více zaměstnancům, jeden zaměstnanec může mít přiděleno víc vozů

- **v každém případě budou tři relace** (dvě pro entity, jedna pro vztah)
- primárním klíčem vztahové relace bude dvojice příslušných cizích klíčů

OSOBA(č_osoby, ...)
VŮZ(č_vozu, ...)
POUŽÍVÁ(č_osoby, č_vozu)

Slabý entitní typ

- identifikační závislost na id. vlastníkovi, slabý ent. typ má pouze parciální identifikační klíč
- identifikační vztah je speciálním případem vztahu 1:N, kde slabý entitní typ má povinné členství, tedy máme vyřešeno
- k relaci slabého ent. typu přilepíme atributy odpovídající identifikačním klíčům id. vlastníků jako cizí klíče

Entitní podtyp (ISA hierarchie)

- narozdíl od slabého typu nemá žádný parciální klíč, je identifikován zdrojem ISA hierarchie
- více možností transformace, podrobněji viz předmět DS2
- obvykle definujeme schema obsahující atributy odpovídající vlastním atributům ent. podtypu a k nim přilepíme atributy odpovídající identifikačnímu klíči zdroje ISA hierarchie

Problémy transformace

Mějme schema

Přidejme ke vztahu atribut „do kdy je hospitalizován“.

První možnost:

PACIENT(rč, ..., č_pokoje, do_dne)
POKOJ(č_pokoje, ...)

Druhá možnost

PACIENT(rč, ...)
POKOJ(č_pokoje, ...)
UMÍSTĚN(rč, č_pokoje, do_dne)

První možnost neumožňuje korektně evidovat pacienty nezávisle na pokoji (ambulantní). Můžeme sice povolit prázdnou hodnotu atributu č_pokoje v relaci PACIENT, ale to vede k potížím:

- musíme zavést další IO, že „má-li číslo pokoje prázdnou hodnotu, pak má prázdnou hodnotu i atribut do_dne“ a případně „má-li číslo pokoje neprázdnou hodnotu, pak má neprázdnou hodnotu i atribut do_dne“
- pokud bude identifikátor nedeterminantu víceatributový, pak musíme hlídat současné prázdné či neprázdné hodnoty všech identifikačních atributů

I druhá možnost ale potřebuje dodefinovat referenční integritu.

Referenční integrita
Týká se téměř všech transformačních operací.

Transformace vztahů:

- musí vždy platit, že hodnota cizího klíče (v samostatné vztahové entitě nebo klíče přilepeného k determinantu) je obsažena jako hodnota primárního klíče ve vztahované entitě
    - např. hodnota atributu č_pokoje přilepeného do relace PACIENT musí existovat jako primární klíč nějaké n-tice v relaci POKOJ

Transformace slabých entitních typů

- musí vždy platit, že hodnota cizího klíče slabé entity je obsažena jako hodnota primárního klíče v relaci identifikačního vlastníka, tj. nesmí se stát, že slabá entita odkazuje na neexistující záznam v relaci identifikačního vlastníka

Transformace entitních podtypů

- musí vždy platit, že hodnota primárního klíče entitního podtypu je obsažena jako hodnota primárního klíče v relaci zdroje ISA hierarchie, tj. nesmí se stát, že entitní podtyp odkazuje na neexistující záznam v relaci zdroje ISA hierarchie

Problémy s parcialitou

- v reprezentaci vztahu 1:N nejsme schopni odlišit povinné a nepovinné členství nedeterminantu.
- při povinném členství všech účastníků vztahu může nastat problém s aktualizacemi relací
    - deadlock

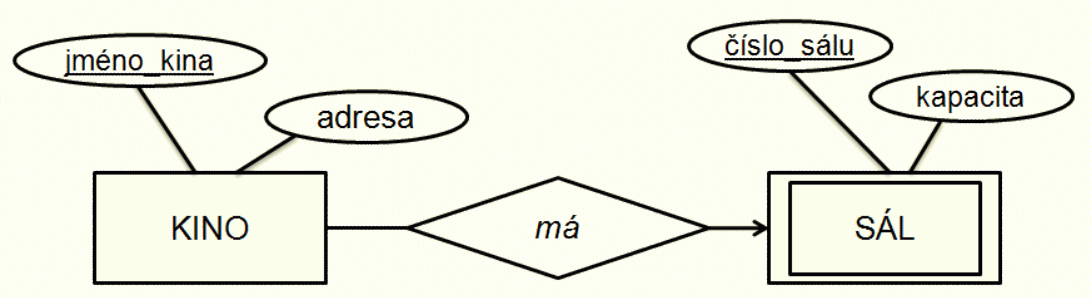
Problémy s kardinalitou

- jak poznat vztah 1:N s povinným členstvím determinantu od vztahu 1:1 s nepovinným členstvím jedné entity?
    - v obou případech řešíme transformaci přilepením cizího klíče do druhé relace, schemata relací tedy vypadají stejně
    - u vztahu 1:1 musíme zajistit, aby hodnoty cizího klíče byly pro každou n-tici unikátní; to lze provést buď tak, že cizí klíč bude zároveň i primární klíč (což není u vztahu 1:1 principielně problém), nebo to lze např. v SQL nařídit (constraint UNIQUE)

Častá chyba

Záměna pojmů konceptuálního a logického modelu:

- entitě v *ER modelu* přiřadíme atribut, který slouží k reprezentaci vztahu v *relačním* modelu:

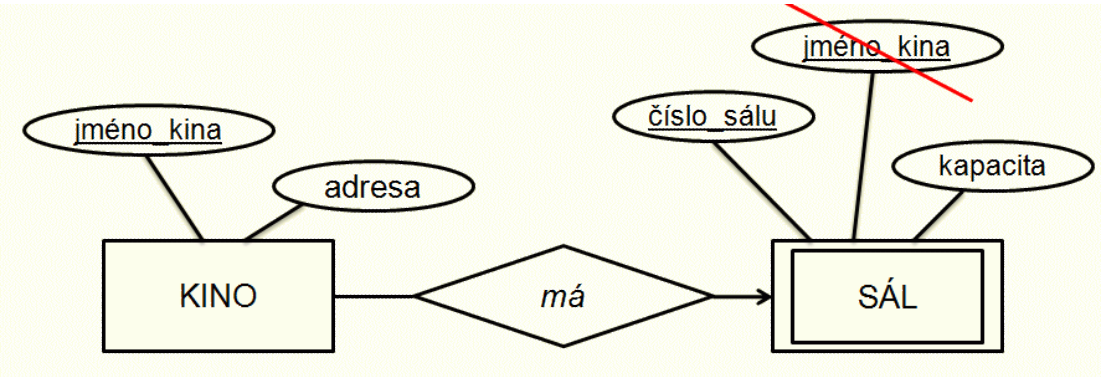


Toto schema bude tranformováno do dvou relací:

KINO(jméno_kina, adresa)

SÁL(číslo_sálu, jméno_kina, kapacita)

To ovšem neznamená, že by ER diagram měl vypadat takto:



- atribut „jméno_kina“ v entitě SÁL je na konceptuální úrovni nesmyslný
- to, že daný sál patří do daného kina je **jednoznačně řečeno vztahem má**

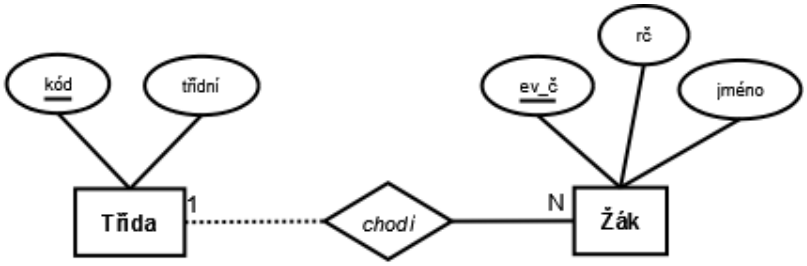
Kontrolní otázky

- Co to je cizí klíč? Jak souvisí s referenční integritou?
- Jaká je souvislost dekompozice M:N vztahu a převodu M:N vztahu do RMD?
- Jsou nějaká integritní omezení E-R modelu, která se v RMD nedají explicitně vyjádřit?

Řešené příklady

Aby byla zřejmá kontinuita tvorby databáze, budeme transformovat především ER modely, vytvořené v kapitole 3. Primární klíče značíme podtrženě, cizí klíče značíme italikou.

Příklad 1

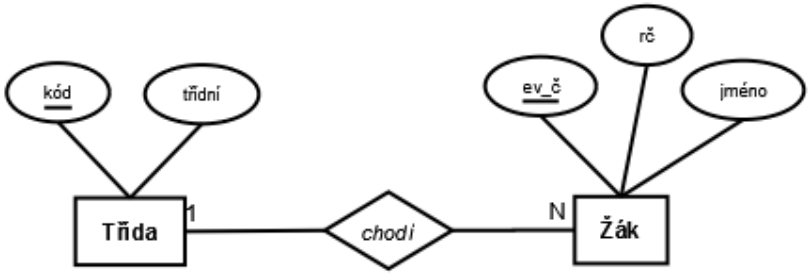


Řešení: Silné entitní typy Třída a Žák transformujeme do samostatných relací s odpovídajícími atributy. Vztah 1:N má povinné členství determinantu a proto se transformuje přilepením cizího klíče do relace determinantu. Díky tomu, že v relačním modelu neexistují prázdné hodnoty atributů, je takto zajištěno, že každý žák musí mít přiřazenu třídu, do které chodí. Také nepovinné členství Třídy ve vztahu je transformací zachyceno: pokud do určité třídy nechodí žádný žák, potom se její kód nevyskytuje v žádném prvku relace Žák, ale existence třídy tím není ovlivněna.

TŘÍDA(<u>kód</u>, třídní)
ŽÁK(<u>ev_č</u>, rč, jméno, *kód*)

Příklad 2

Podívejme se na velmi podobný případ jako v předešlém příkladu, rozdílem je pouze fakt, že do každé třídy musí nyní chodit aspoň jeden žák, tj. entitní typ Třída má povinné členství ve vztahu.



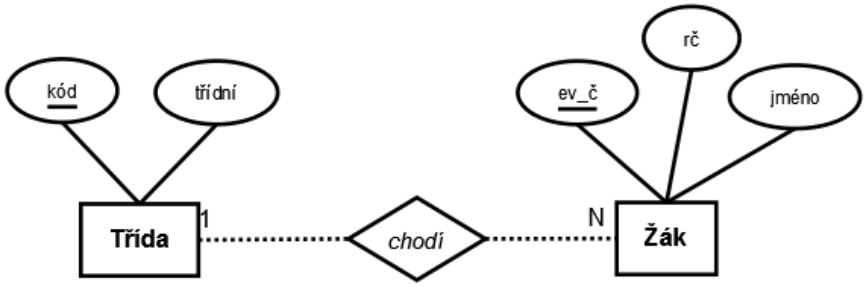
Řešení: Postupujeme analogicky jako v předchozím příkladu. Silné entitní typy Třída a Žák transformujeme do samostatných relací s odpovídajícími atributy. Vztah 1:N má povinné členství determinantu a proto se transformuje přilepením cizího klíče do relace determinantu. Je tedy zajištěno, že každý žák musí mít přiřazenu třídu, do které chodí. Povinné členství Třídy ve vztahu ovšem relační model nezajistí: podle ER modelu by neměla existovat žádná třída, jejíž kód se nevyskytuje v relaci Žák. To ale relační model nikterak neříká, vidíme ostatně, že transformací ER diagramu jsme obdrželi stejná

schemata relací jako v předchozím příkladu, a tedy povinné členství Třídy není zajištěno. Toto integritní omezení je nutné vynutit jinými prostředky na úrovni fyzického modelu, tj. buď v databázovém stroji nebo ve vrstvě aplikace.

TŘÍDA(<u>kód</u>, třídní)
ŽÁK(<u>ev_č</u>, rč, jméno, *kód*)

Příklad 3

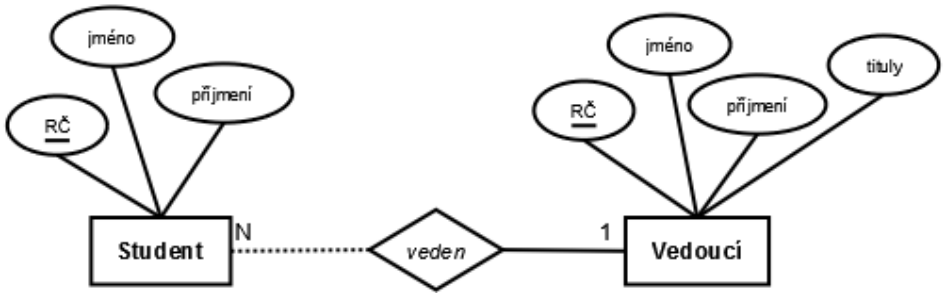
Do třetice prozkoumejme případ, kdy v databázi evidujeme i žáky, kteří nechodí do žádné třídy (studují například doma).



Řešení: Silné entitní typy Třída a Žák transformujeme opět do samostatných relací. Vztah 1:N má ale nyní nepovinné členství determinantu. V relačním modelu jej tedy nelze transformovat přilepením cizího klíče do relace determinantu, neboť tento cizí klíč by byl prázdný v případech žáků nechodících do žádné třídy. Prázdné hodnoty nejsou v RMD povoleny, je tedy nutno zavést pro vztah samostatnou vztahovou relaci Chodí. Klíčem nové relace bude sloupec „ev_č“, protože každý žák chodí jen do jedné třídy, jeho evidenční číslo smí být tedy v relaci Chodí pouze jednou. Naproti tomu kód třídy se může v relaci Chodí opakovat, protože do jedné třídy může chodit více žáků. Povinné/nepovinné členství Třídy opět relační model není schopen explicitně řešit.

TŘÍDA(<u>kód</u>, třídní)
ŽÁK(<u>ev_č</u>, rč, jméno)
CHODÍ(<u>*ev_č*</u>, *kód*)

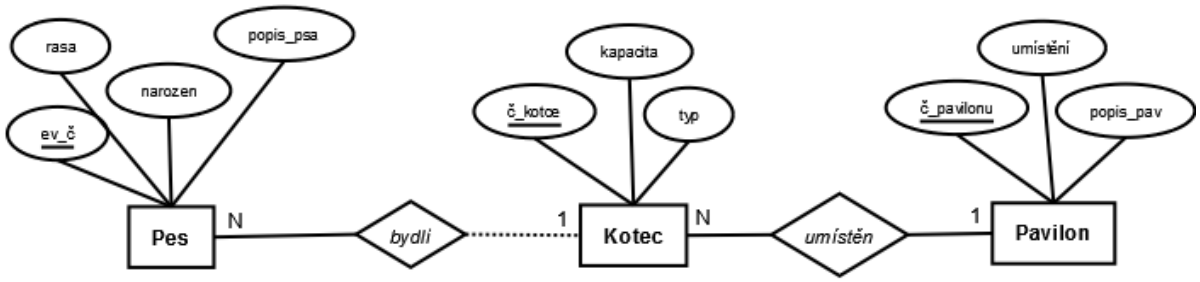
Příklad 4



Řešení: Tento příklad je analogický předchozímu příkladu. Řešení je samozřejmě obdobné, pouze nyní z důvodu shodně pojmenovaných atributů rozlišíme cizí klíče ve vztahové relaci Veden. Zde tedy vidíme, jak je vhodné pojmenovávat všechny atributy diagramu unikátními názvy, abychom byli na první pohled schopni rozeznat např. atribut jméno vedoucího od atributu jméno studenta.

VEDOUCÍ(RČ, jméno, příjmení, tituly)
STUDENT(RČ, jméno, příjmení)
VEDEN(*RČ_studenta*, *RČ_vedoucího*)

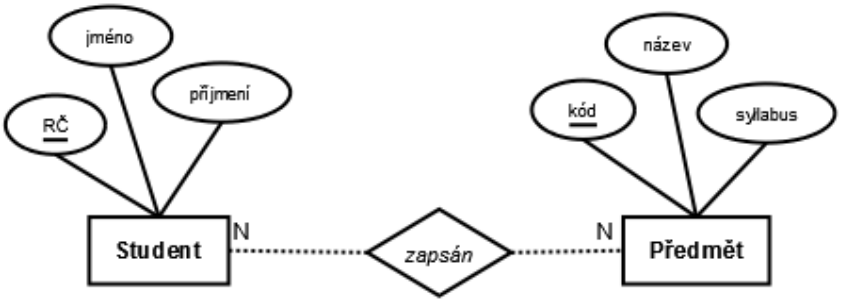
Příklad 5



Řešení: Příklad opět využívá pouze dříve procvičených postupů. Silné entitní typy převedeme na samostatné relace, vztahy 1:N s povinným členstvím determinantu řešíme cizím klíčem přilepeným k relaci determinantu. Povinné členství Pavilonu ve vztahu Umístěn nelze v relačním modelu zajistit. Atributy raději pojmenováváme unikátně, tedy například „popis_psa“ respektive „popis_pav“.

PES(ev_č, rasa, narozen, popis_psa, *č_kotce*)
KOTEC(č_kotce, kapacita, typ, *č_pavilonu*)
PAVILON(č_pavilonu, umístění, popis_pav)

Příklad 6



Řešení: Vztah N:N se transformuje do relačního modelu s použitím vztahové relace. Relaci pojmenujeme stejně jako vztah. Vztahová relace bude mít dva atributy, a sice cizí klíče odkazující do relací odpovídající

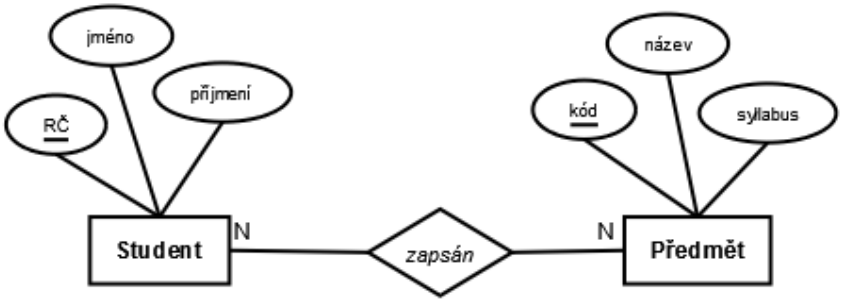
hlavním entitním typům. Primárním klíčem nové relace budou oba atributy, neboť každá dvojice {rč, kód} se smí vyskytnout jen jednou – student si nemůže vícekrát zapsat tentýž předmět.

Vztah *zapsán* má v obou směrech nepovinné členství. Tento fakt je transformací do RMD zachycen: pokud například dané rodné číslo nebude mít zapsaný žádný předmět, nebude se vůbec vyskytovat v relaci *Zapsán*. Podobně je tomu i pro kód předmětu. Na tomto místě je třeba upozornit, že i kdyby měl vztah na obou stranách povinné členství, transformace do relací by vypadala stejně. U vztahu N:N nelze v relačním modelu povinné členství zajistit ani na jedné straně vztahu. Toto integritní omezení je nutné vynutit jinými prostředky na úrovni fyzického modelu, tj. buď v databázovém stroji nebo ve vrstvě aplikace.

STUDENT(RČ, jméno, příjmení)
PŘEDMĚT(kód, název, syllabus)
ZAPSÁN(*RČ, kód*)

Příklad 7

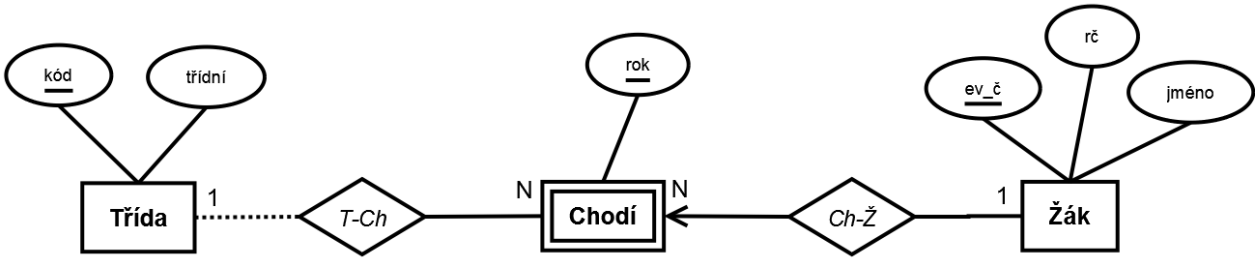
Podívejme se nyní na případ, kdy bude u vztahu *zapsán* v obou směrech **povinné** členství, tj. každý student musí mít zapsaný aspoň jeden předmět a každý předmět musí mít zapsaný aspoň jeden student.



Řešení: I v tomto případě bude transformace do relací vypadat stejně jako v předchozím příkladu. U vztahu N:N totiž nelze samotným relačním modelem zajistit povinné členství ani na jedné straně vztahu. Může se například stát, že bude existovat prvek relace *Předmět* s takovým kódem, jaký se vůbec nevyskytuje v relaci *Zapsán*. Toto integritní omezení je tedy nutné vynutit jinými prostředky na úrovni fyzického modelu, tj. buď v databázovém stroji nebo ve vrstvě aplikace.

STUDENT(RČ, jméno, příjmení)
PŘEDMĚT(kód, název, syllabus)
ZAPSÁN(*RČ, kód*)

Příklad 8



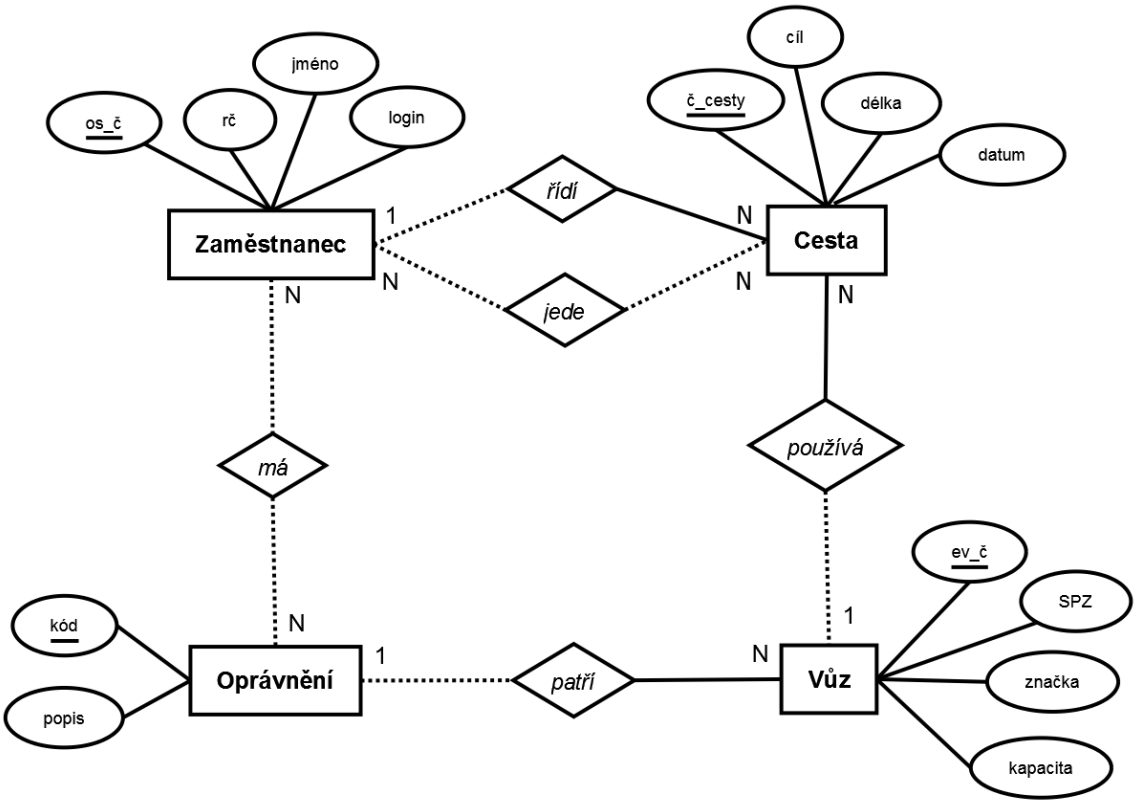
Řešení: Situace je podobná jako v předchozích dvou příkladech, jedná se de facto o dekomponovaný vztah N:N. Vztahová relace vznikne tentokrát transformací slabého entitního typu. Slabý entitní typ se transformuje na samostatnou relaci (obdobně jako silný entitní typ), jediným rozdílem je fakt, že součástí primárního klíče je cizí klíč odkazující na identifikačního vlastníka.

Oba vztahy transformujeme přilepením cizího klíče k determinantu, tj. ke vztahové relaci. Ta bude mít kromě dvou atributů – cizích klíčů – ještě další atribut „rok“, rozlišující školní rok. Primárním klíčem nové relace budou atributy „ev_č“ a „rok“: cizí klíč „ev_č“ musí být součástí primárního klíče (PK), neboť se jedná o slabou entitu, „rok“ musí být součástí PK z toho důvodu, že daný žák smí daný rok chodit pouze do jedné třídy, tj. dvojice {ev_č, rok} musí být v relaci *Chodí* vždy unikátní.

Vztah *Ch-Ž* mezi entitami *Chodí* a *Žák* má z obou stran povinné členství, povinné členství je však transformací zaručeno pouze u determinantu, tj. u relace *Chodí*, kde lze vynutit neprázdnost cizího klíče. Naproti tomu samotnými relacemi nelze zajistit, aby každé rodné číslo chodilo aspoň do jedné třídy, jak to nařizuje ER model. Může se totiž stát, že bude existovat prvek relace *Žák* s takovým rodným číslem, jaké se vůbec nevyskytuje v relaci *Chodí*. Toto integritní omezení je tedy nutné ošetřit např. triggerem na úrovni databázového stroje nebo jinak na úrovni aplikace.

ŽÁK(ev_č, rč, jméno)
TŘÍDA(kód, třídní)
CHODÍ(*kód, ev_č*, rok)

Příklad 9

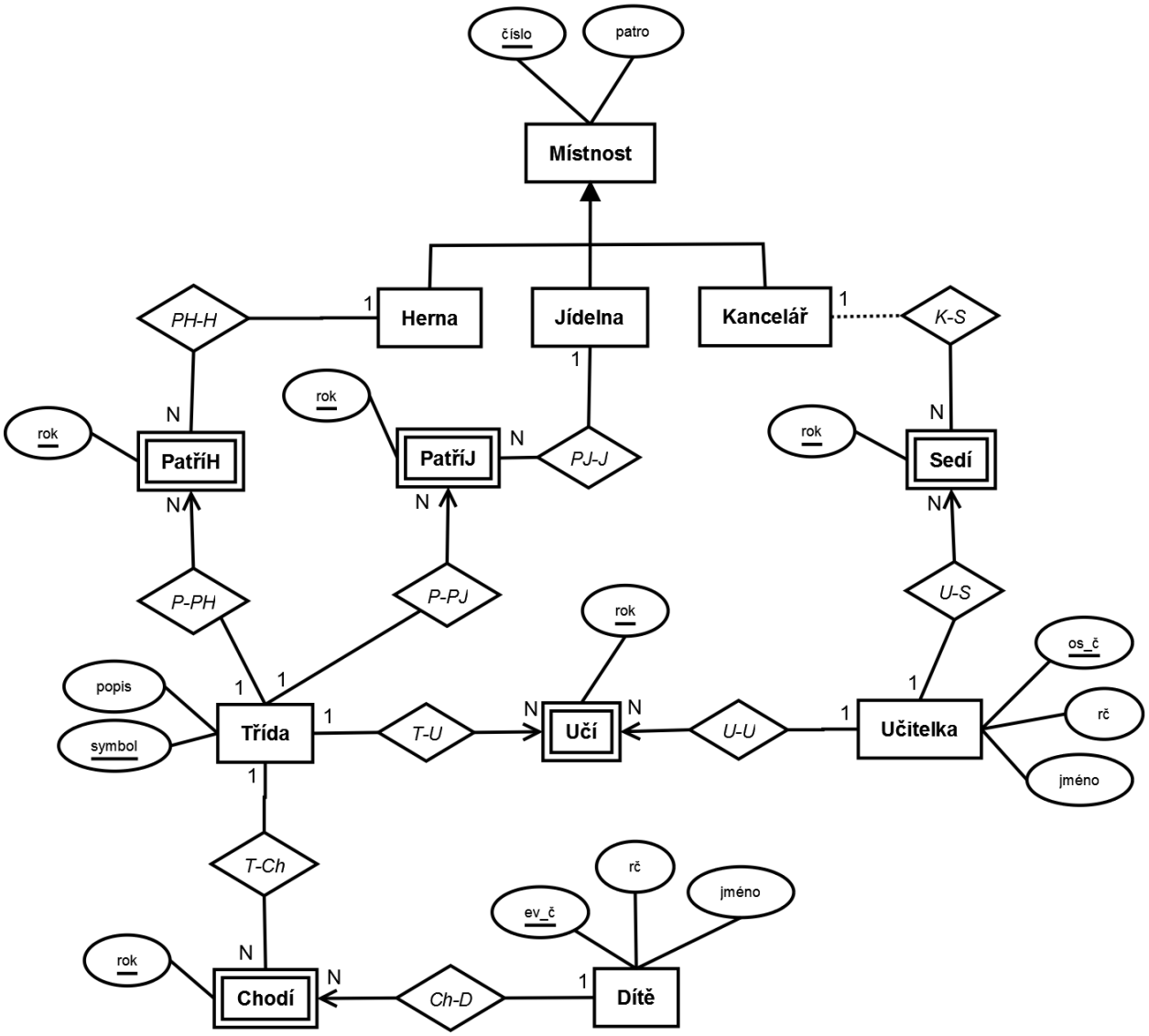


Řešení: Zde již máme poněkud složitější ER model. Zavedeme relaci pro každý entitní typ, dále musíme vhodně transformovat vztahy. Vztah *řídí* je typu 1:N s povinným členstvím determinantu, transformujeme jej tedy přilepením cizího klíče „os_č“ k relaci *Cesta*. Vztah *Jede* je typu N:N, transformujeme jej tedy zavedením nové vztahové relace s patřičnými cizími klíči, viz též příklad 6. Vztah *používá* je typu 1:N s povinným členstvím determinantu, transformujeme jej přilepením cizího klíče „ev_č“ k relaci *Cesta*. Vztah *patří* je typu 1:N s povinným členstvím determinantu, transformujeme jej přilepením cizího klíče „kód“ k relaci *Vůz*. Vztah *Má* je typu N:N, transformujeme jej zavedením nové vztahové relace s patřičnými cizími klíči.

Pro povinná resp. nepovinná členství platí tvrzení z předešlých příkladů: u vztahu N:N nelze relačním modelem zajistit povinná členství ani na jedné straně vztahu, u vztahu 1:N lze relačním modelem zajistit povinné členství pouze u determinantu vztahu.

ZAMĚSTNANEC(<u>os_č</u>, rč, jméno, login)
OPRÁVNĚNÍ(<u>kód</u>, popis)
VŮZ(<u>ev_č</u>, SPZ, značka, kapacita, *kód*)
CESTA(<u>č_cesty</u>, cíl, délka, datum, *os_č_řidič, ev_č*)
JEDE(<u>*os_č_spolujezdec, č_cesty*</u>)
MÁ(<u>*os_č, kód*</u>)

Příklad 10



Řešení: V tomto ER modelu se vyskytují silné a slabé entitní typy a ISA hierarchie. Pro slabé i silné entitní typy zavedeme samostatné relace, řešení slabých entitních typů viz příklad 8. ISA hierarchii můžeme řešit různými způsoby. Zde zvolíme reprezentaci třemi relacemi pro entitní podtypy, nebudeme zavádět relaci pro entitní nadtyp (*Místnost*). Důvodem je fakt, že entitní typ *Místnost* není zapojen do žádného vztahu, navíc společných atributů je poměrně málo, není tedy problém je opakovat v entitních podtypech. Všechny vztahy jsou typu 1:N, řešíme je tedy přilepením cizího klíče k determinantu. Pro povinná a nepovinná členství platí rozbor uvedený v předchozích příkladech.

HERNA(<u>číslo_her</u>, patro)
JÍDELNA(<u>číslo_jíd</u>, patro)
KANCELÁŘ(<u>číslo_kanc</u>, patro)
TŘÍDA(<u>symbol</u>, popis)

UČITELKA(os_č, rč_učit, jméno)
DÍTĚ(ev_č, rč_dítě, jméno)
PATŘÍH(*číslo_her, symbol*, rok)
PATŘÍJ(*číslo_jíd, symbol*, rok)
SEDÍ(*číslo_kanc, os_č*, rok)
UČÍ(*symbol, os_č*, rok)
CHODÍ(*symbol, ev_č*, rok)

Úlohy k procvičení

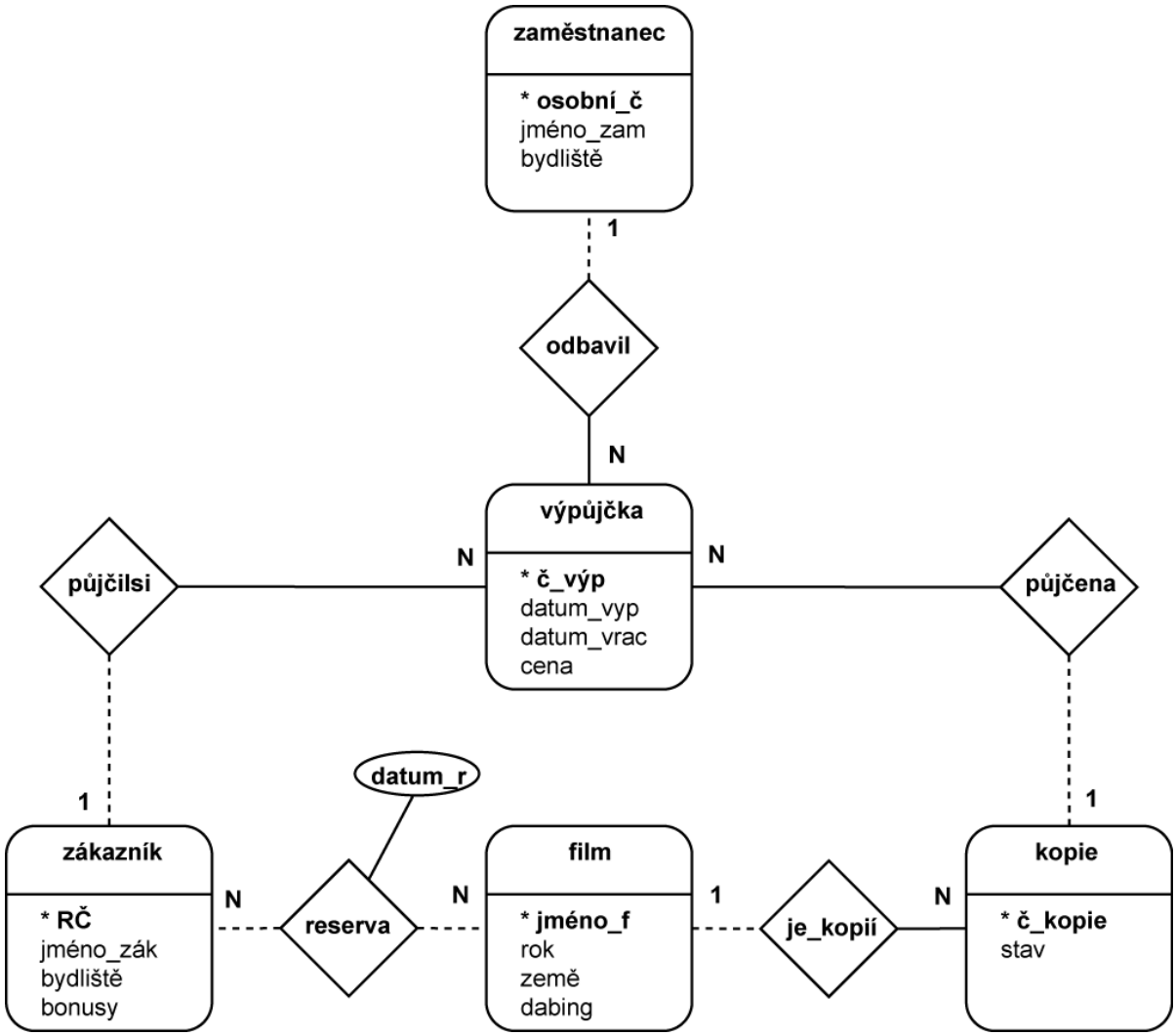
1. Transformujte ER-modely příkladu 1 z kapitoly 3 do relačního modelu a proveďte všechny dotazy. Kromě uvedených dvou variant uvažujte také, že chcete odlišit různá data promítání filmů.

   Dotazy:

- vypiš názvy kin, která mají kapacitu větší než 100 míst
- vypiš názvy filmů, které byly natočeny v USA
- vypiš názvy kin, která mají dolby a nehrají zrovna žádný film
- vypiš filmy, které se hrají v kině Dukla
- vypiš názvy a adresy kin, kde se dává film Titanic
- vypiš názvy a adresy kin, kde se dává nějaký film z Japonska

---

2. Transformujte do RMD model dle obrázku:



Proveďte dotazy:

- vypiš jména zákazníků s jejich adresami (pozn.: co duplicity?)
- vypiš výpůjčky, které byly vráceny do 31.12.2008
- najdi země, na jejichž filmy jsou nějaké reservace
- najdi dvojice zákazníků, kteří mají stejnou adresu
- najděte zákazníky, kteří si půjčili film Údolí včel
- najdi jména zákazníků, kteří mají reservován nějaký film
- najdi jména zákazníků, kteří nemají reservován žádný film
- najdi jména filmů, které jsou právě půjčeny (předpokládejte, že pro dosud nevrácené výpůjčky je datum vrácení nastaveno na 0)
- najděte jména zaměstnanců, kteří někdy půjčili film někomu z Polné

3. Transformujte do relačního modelu Vaše ER-modely vytvořené v kapitole 3, příklad 19.

7 Jazyk SQL

Cíl kapitoly

Kapitola představuje jazyk SQL a uvádí jeho základní součást – jazyk pro definici dat. Rovněž se probírá část jazyka pro manipulaci s daty, a sice nástroje pro aktualizaci tabulek. Cílem je naučit se

- jaký je princip jazyka SQL
- jaké jsou součástí jazyka SQL
- jak se vytváří tabulka včetně veškerých integritních omezení
- jak se mění struktura existující tabulky
- jak se odstraňuje tabulka
- jak se vkládají data do tabulky
- jak se mění existující data v tabulce
- jak se mažou data z tabulky

Klíčové pojmy

Jazyk pro definici dat, příkazy create table, alter table, insert, update, delete from, drop.

Úvod

SQL je neprocedurální jazyk

- říká co chceme udělat, nikoli jak

Má několik částí, ty základní jsou

- definice dat (DDL – data definition language)
- manipulace s daty (DML – data manipulation language)
- přístupová práva (DCL – data control language)

Základní vlastnosti

- data uložena ve formě **tabulek**
  - skutečné (odpovídají schematu DB)
  - virtuální (pohledy)
- vrací data, není třeba se starat o jejich umístění či strukturu
- poloha tabulek v DB není podstatná, jsou identifikovány jménem
- pořadí sloupců v tabulkách není podstatné, jsou identifikovány jménem
- pořadí řádků v tabulkách není podstatné, jsou identifikovány hodnotami ve sloupcích

Součásti jazyka SQL

- jazyk pro definici dat
- jazyk pro manipulaci s daty
  - interaktivní
  - v hostitelské verzi
- definice pohledů
- definice IO
- definice přístupových práv
- jazyk modulů
- řízení transakcí

Definice dat

Příkaz CREATE TABLE

- vytvoří schema a prázdnou tabulku (pokud již neexistuje)
- tabulka má definovány sloupce vč. jejich IO a také tabulková IO

Syntaxe (zjednodušeně)

```
CREATE TABLE název_tab (    seznam prvků oddělený čárkami )
```

Prvek může být:

- definice sloupce
- definice IO tabulky

Definice sloupce - syntaxe

```
jméno_sloupce    datový_typ    [IO_sloupce]
```

- např.

```
rodné_číslo    DECIMAL(10)    UNIQUE
```

Př. jednoduchá tabulka bez IO:

```
CREATE TABLE Osoba (
        jméno           VARCHAR (20),
        příjmení        VARCHAR(30),
        rč              DECIMAL(10) )
```

Základní datové typy (PostgreSQL)

- INTEGER – celé číslo
- DECIMAL(p,q) – dekadické číslo s p číslicemi a q číslicemi za desetinnou čárkou
- REAL – číslo s plovoucí desetinnou čárkou
- CHAR (n) – řetězec znaků délky n
- VARCHAR(n) – řetězec znaků délky max. n

- DATE – datum ve tvaru rrrr-mm-dd
- BOOLEAN – logická hodnota
- ...

Hodnota NULL:

- speciální hodnota, která náleží do všech datových typů
- indikuje prázdný prvek (nedefinovaná, neznámá hodnota)
    - nula není prázdná hodnota!
- pozor na ni, způsobuje potíže
    - tříhodnotová logika
    - často neintuitivní chování v agregačních funkcích
    - potíže při vyhledávání (např. podmínka se často považuje za splněnou, pokud je jedním z operandů NULL)

IO sloupce

- umožňuje omezit množinu platných hodnot daného atributu (při vkládání / aktualizaci)

- NOT NULL – sloupec nesmí mít prázdnou hodnotu
- DEFAULT X – definice implicitní hodnoty X
- UNIQUE – sloupec musí mít unikátní hodnoty (v rámci dané tabulky)
- PRIMARY KEY – sloupec je primárním klíčem
    - sémanticky totéž jako NOT NULL + UNIQUE
- REFERENCES – odkazuje jako cizí klíč na atribut jiné tabulky
- CHECK – přídavné IO, obecná logická podmínka

Např. tabulka s IO sloupce

```
CREATE TABLE Osoba (
        os_č        INTEGER             PRIMARY KEY,
        rč          DECIMAL(10)         UNIQUE,
        jméno       VARCHAR (20)        NOT NULL,
        příjmení    VARCHAR(30)         NOT NULL,
        login       CHARACTER(8)  UNIQUE,
        místnost    INTEGER       REFERENCES    Místnost(č_míst)
);

CREATE TABLE Místnost (
        č_míst      INTEGER       PRIMARY KEY,
        patro       INTEGER       CHECK (patro<3)
);
```

IO tabulky

- umožňuje definovat IO pro více sloupců najednou

- UNIQUE
- PRIMARY KEY – definice primárního klíče
- FOREIGN KEY – cizí klíč na atributy jiné tabulky
- CHECK – přídavné IO, obecná logická podmínka

```
CREATE TABLE Osoba (
        rč              DECIMAL(10)             NOT NULL,
        jméno           VARCHAR (20)            NOT NULL,
        příjmení        VARCHAR(30)             NOT NULL,
        PRIMARY KEY (jméno, příjmení)    );
```

- CHECK může obsahovat i vnořený SELECT blok (v PostgreSQL nejde)
- IO tabulky může být pojmenováno - CONSTRAINT

```
CREATE TABLE Výrobek (
        Id              INTEGER                 PRIMARY KEY,
        Název           VARCHAR(128)            UNIQUE,
        Cena            DECIMAL(6,2)            NOT NULL,
        JeNaSkladě      BOOL                    DEFAULT TRUE,
        CONSTRAINT chk CHECK (
                (Id = 0     OR     Id > ALL (SELECT Id FROM Výrobek) )
                AND    Cena > 0)                );
```

Referenční integrita

- při aktualizaci tabulek může nastat porušení cizích klíčů
    - pokus o vložení řádku s hodnotou cizího klíče, která se nevyskytuje v referované tabulce
    - pokus o smazání záznamu v referované tabulce, na který je odkazováno odjinud
- lze nařídit akce, provedené při nastalé situaci

    {ON UPDATE | ON DELETE}   AKCE

- CASCADE – odkazující záznamy se smažou nebo se přepíší aktualizovanou hodnotou
- SET NULL   - odkazující záznamy se nastaví na NULL
- SET DEFAULT - odkazující záznamy se nastaví na impl. hod.

```
CREATE TABLE Osoba (
        jméno           VARCHAR (20)            NOT NULL,
        příjmení        VARCHAR(30)             NOT NULL,
        místnost        INTEGER,
        PRIMARY KEY (jméno, příjmení)     );
```

```
CREATE TABLE Vozidlo (
        SPZ             VARCHAR(10)         PRIMARY KEY,
        jméno           VARCHAR(20)         NOT NULL,
        příjmení        VARCHAR(30)         NOT NULL,
        FOREIGN KEY (jméno, příjmení) REFERENCES Osoba(jméno,příjmení) ON UPDATE
        CASCADE);
```

Příkaz ALTER TABLE

- změna definice tabulky
- pokud tabulka obsahuje data, nemusí se povolit (např. změna primárního klíče)

```
ALTER TABLE table-name
... ADD [COLUMN] column-name      column-definition
... DROP [COLUMN] column-name
... RENAME [COLUMN] column-name TO new-name
```

- a spousta dalších věcí (viz další přednášky)
  [xx] značí nepovinný řetězec

```
CREATE TABLE Výrobek (
        Id              INTEGER             PRIMARY KEY,
        Název           VARCHAR(128)        UNIQUE,
        Cena            DECIMAL(6,2)        NOT NULL,
        JeNaSkladě      BOOL                DEFAULT TRUE,
        CONSTRAINT chk CHECK (Id > 0)    )

ALTER TABLE Výrobek DROP JeNaSkladě;          --odstranění sloupce
ALTER TABLE Výrobek ADD DatumVýroby DATE;         --přidání sloupce
ALTER TABLE Výrobek RENAME Id TO Číslo;         --přejmenování sloupce
```

Příkaz DROP TABLE

- odstraní tabulku včetně její definice
- pro smazání obsahu tabulky, ale ponechání jejího schematu v DB viz příkaz DELETE FROM

```
DROP TABLE Výrobek
```

Aktualizace dat

Příkaz INSERT INTO

- vloží data do tabulky
- lze vložit buď všechny sloupce nebo jen některé

```
INSERT INTO table_name (col1, col3) VALUES (val1, val3)
INSERT INTO table_name VALUES (val1, val2, val3)
```

- lze vložit více řádků naráz (množinově)

```
INSERT INTO table_name1 (col1, col3)
                SELECT a, b
                FROM table_name2
                WHERE condition

INSERT INTO Osoba (jméno, příjmení) VALUES (‘Jan‘, ‘Drda‘)

CREATE TABLE Vip (      jméno VARCHAR(20),
                        příjmení VARCHAR(30),
                        čistý_plat FLOAT    )
INSERT INTO Vip    SELECT jméno, příjmení, plat*0.8
                   FROM Osoba
                   WHERE plat>100000
```

Příkaz UPDATE

- změní data v tabulce
- mění ty řádky, které splní podmínku
- lze se odkazovat na přepisovaná data

```
UPDATE Osoba SET jméno=‘Honza‘ WHERE jméno=‘Jan‘;
UPDATE Vip SET čistý_plat = čistý_plat*0.9;
```

Příkaz DELETE FROM

- odstraní data z tabulky
- smaže ty řádky, které splní podmínku, není-li zadána podmínka, odstraní všechny (ale ne tabulku jako takovou – to se udělá DROP TABLE)

```
DELETE FROM Vip
```

Kontrolní otázky

- Jak se definuje primární a cizí klíč?
- Jak se liší definice IO sloupce a IO tabulky?
- Jaké jsou možnosti řízení referenční integrity?
- Jak se liší příkazy DROP TABLE a DELETE FROM table?

Řešené příklady

Poznámka

V příkladech v tomto textu je použit dialekt jazyka SQL, pocházející z databázového stroje PostgreSQL. Použijete-li jiný databázový stroj, některé syntaktické konstrukce nemusí fungovat správně. Vždy je nutné si ověřit, jakou syntaxi s jakou sémantikou Vámi vybraný stroj používá.

Příklad 1

Vytvořte jednoduchou databázi žáků ve třídách. Jeden žák může chodit pouze do jedné třídy, ale v jedné třídě může být více žáků. U třídy evidujeme její kód (např. 1A) a jméno třídního, identifikátorem třídy bude její kód. U žáka evidujeme rodné číslo, jméno a do jaké třídy chodí, identifikátorem žáka bude pro jednoduchost rodné číslo (nevýhody tohoto řešení viz kapitola 3). Naplňte tabulky testovacími daty a vyzkoušejte jednoduché dotazy.

Řešení:

Vytvoříme dvě tabulky, které budou vzájemně provázané. V jedné tabulce budeme mít seznam tříd, v druhé tabulce seznam všech žáků, u každého žáka pak budeme mít „poznamenáno“, do které třídy chodí. Tabulka třída bude mít dva sloupce: kód_třídy (datový typ: dva znaky) a třídní (datový typ: řetězec 0 až 50 znaků, může být prázdný). Tabulka žák bude mít tři sloupce: rč (datový typ: řetězec 0 až 11 znaků), jméno (datový typ: řetězec 0 až 50 znaků, nesmí být prázdný) a kód_třídy (odkaz do tabulky třída, datový typ stejný jako odpovídající sloupec v tabulce třída, nesmí být prázdný). Zopakujme, že každý SQL příkaz musí končit středníkem.

název sloupce

datový typ

sloupec slouží jako identifikátor, tzv. **primární klíč**

```
CREATE TABLE trida (
      kod_tridy char(2) primary key,
      tridni varchar(50)
);
```

nesmí být prázdné

odkaz do tabulky třída, tzv. **cizí klíč**

```
CREATE TABLE zak (
      rc varchar(11) primary key,
      jmeno varchar(50) not null,
      kod_tridy char(2) not null references trida(kod_tridy)
);
```

Naplníme tabulky alespoň nějakými daty.

vkládáme data do všech sloupců

```
INSERT INTO trida VALUES ('1A','Petr Kus');
INSERT INTO trida(kod_tridy) VALUES ('1B');
```

výběr sloupců, do nichž vkládáme data

hromadné vložení více řádků

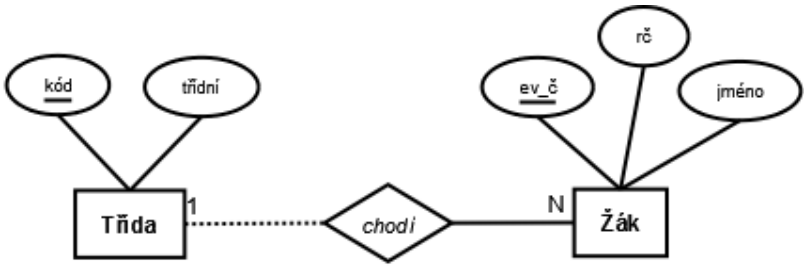
```
INSERT INTO zak VALUES
      ('123456/1234','Iva Mala', '1A'),
      ('123457/1234','Petr Fucik', '1A'),
      ('123458/1234','Mia Farrow', '1B'),
      ('123459/1234','Ian Tyson', '1B');
```

řádek nebude vložen, neb hodnota '3A' není obsažena v tabulce třída

```
INSERT INTO zak VALUES ('123456/1234','Iva Mala', '3A');
```

Dále využijeme především výsledků řešených příkladů z předchozích kapitol a budeme realizovat uvedené relace jako tabulky v SQL. Pro lepší představu vazby mezi abstraktním modelem a konkrétní realizací v databázovém stroji ponecháme v textu též příslušné ER diagramy. Pro bezproblémové vyzkoušení v databázovém stroji budeme zapisovat kód SQL bez diakritiky. V dalším kroku vložíme do tabulek nějaká data a budeme měnit definice tabulek.

Příklad 2



Relační model:

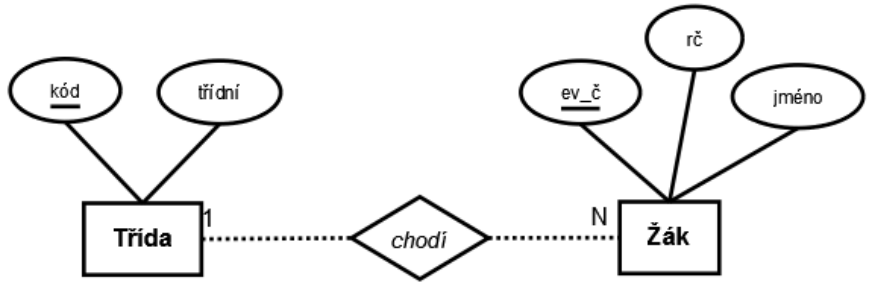
TŘÍDA(<u>kód</u>, třídní)
ŽÁK(<u>ev_č</u>, rč, jméno, *kód*)

SQL: viz též řešení předchozího příkladu. V tabulce Třída povolujeme prázdné hodnoty, v ostatních atributech nikoli. U primárních klíčů nemusíme explicitně uvádět not null, neboť neprázdnost je automaticky zajištěna tím, že je tento sloupec primárním klíčem. Datové typy zvolíme intuitivně, v praxi bychom je volili dle požadavků na systém.

```
CREATE TABLE trida (
      kod         char(2)     primary key,
      tridni      varchar(50)
);
```

```
CREATE TABLE zak (
      ev_c   integer      primary key,
      rc     varchar(11)  not null,
      jmeno  varchar(50)  not null,
      kod    char(2)      not null references trida(kod)
);
```

Příklad 3



Relační model:

TŘÍDA(<u>kód</u>, třídní)

ŽÁK(<u>ev_č</u>, rč, jméno)

CHODÍ(<u>*ev_č*</u>, *kód*)

SQL: přímým přepisem relačího modelu do SQL získáme následující tabulky.

```
CREATE TABLE trida (
      kod          char(2)      primary key,
      tridni       varchar(50)
);

CREATE TABLE zak (
      ev_c   integer      primary key,
      rc     varchar(11)  not null,
      jmeno  varchar(50)  not null
);

CREATE TABLE chodi (
      ev_c   varchar(11)  primary key references zak(ev_c),
      kod    char(2)      not null references trida(kod)
);
```

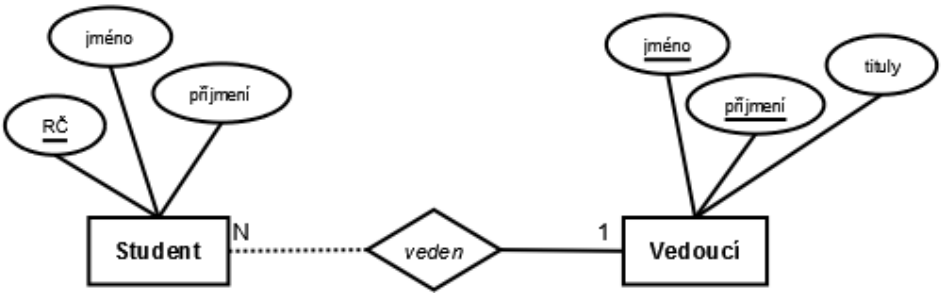
Vztahovou tabulku lze definovat též s využitím tabulkových integritních omezení pro zápis primárního a cizího klíče.

```
CREATE TABLE chodi (
      ev_c   varchar(11),
      kod    char(2)      not null,
      foreign key (ev_c) references zak(ev_c),
      foreign key (kod)  references trida(kod),
      primary key (ev_c)
);
```

Narozdíl od relačního modelu SQL připouští prázdné hodnoty atributů. Uvedený příklad lze tedy řešit v SQL i bez použití vztahové relace, pouze s přilepeným cizím klíčem, který tentokrát může být prázdný. V SQL je totiž referenční integrita splněna i pro prázdný cizí klíč. Srovnejte s řešením příkladu 2. V tabulce Třída použijeme pojmenované tabulkové integritní omezení, kontrolující, že třídní ani kód třídy není prázdný řetězec (pozor, prázdný řetězec není ekvivalentní hodnotě NULL, prázdný řetězec je regulérní hodnota a splňuje nařízení NOT NULL).

```
CREATE TABLE trida (
      kod          char(2) primary key,
      tridni       varchar(50)
      constraint ChkTridni CHECK(tridni<>'' and kod<>'')
);

CREATE TABLE zak (
      ev_c   integer      primary key,
      rc     varchar(11)  not null,
      jmeno  varchar(50)  not null,
      kod    char(2)      references trida(kod)
);
```

Příklad 4



Relační model (pozor, složeným primárním klíčem vedoucího je dvojice {jméno,příjmení}):

VEDOUCÍ(jméno, příjmení, tituly)

STUDENT(RČ, jméno, příjmení)

VEDEN(*RČ_studenta*, *jméno_ved, příjmení_ved*)

SQL: řešení s použitím vztahové tabulky. Pozor, cizí klíč je složený ze dvou sloupců!

```
CREATE TABLE vedouci (
      jmeno        varchar(50),
      prijmeni     varchar(50),
      tituly       varchar(50),
      primary key (jmeno, prijmeni)
);

CREATE TABLE student (
      rc           varchar(11)  primary key,
      jmeno        varchar(50)  not null,
      prijmeni     varchar(50)  not null
);


CREATE TABLE veden (
      rc_stud      varchar(11)  primary key references student(rc),
      jm_ved       varchar(50) not null,
      pr_ved       varchar(50) not null,
      foreign key (jm_ved, pr_ved) references vedouci(jmeno, prijmeni)
);
```

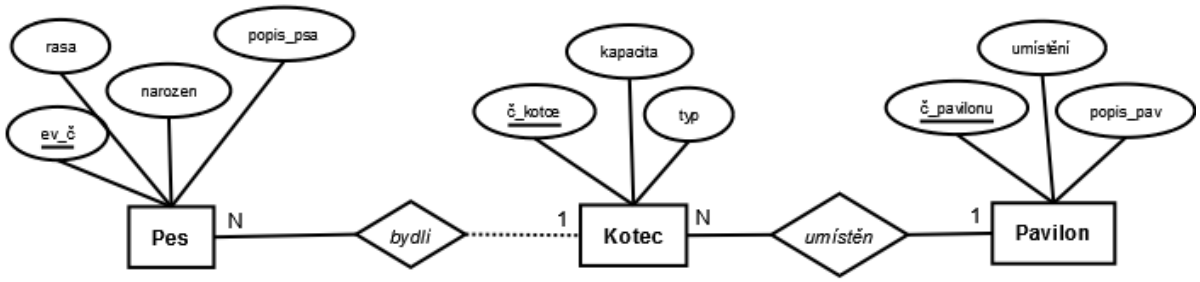
složený primární klíč

složený cizí klíč

Řešení s použitím prázdného cizího klíče (tabulka Vedoucí je shodná jako nahoře, vztah veden je řešen prázdným cizím klíčem v tabulce Student).

```
CREATE TABLE student (
      rc           varchar(11)  primary key,
      jmeno        varchar(50)  not null,
      prijmeni     varchar(50)  not null,
      jm_ved       varchar(50),
      pr_ved       varchar(50),
      foreign key (jm_ved, pr_ved) references vedouci(jmeno, prijmeni)
);
```

Příklad 5



Relační model:

PES(<u>ev_č</u>, rasa, narozen, popis_psa, *č_kotce*)

KOTEC(<u>č_kotce</u>, kapacita, typ, *č_pavilonu*)

PAVILON(č_pavilonu, umístění, popis_pav)

SQL: budeme požadovat, aby všechny atributy byly neprázdné, tj. not null. Pro evidenční číslo psa použijeme datový typ serial, který je celočíselným datovým typem jako integer, zajišťuje ale sám inkrementaci a vyplnění nové unikátní hodnoty (podobně jako v MS Access automatické číslo). Tabulky je třeba definovat ve správném pořadí kvůli referenční integritě. V definici cizích klíčů použijeme řízení referenční integrity. Zopakujme možnosti tohoto řízení:

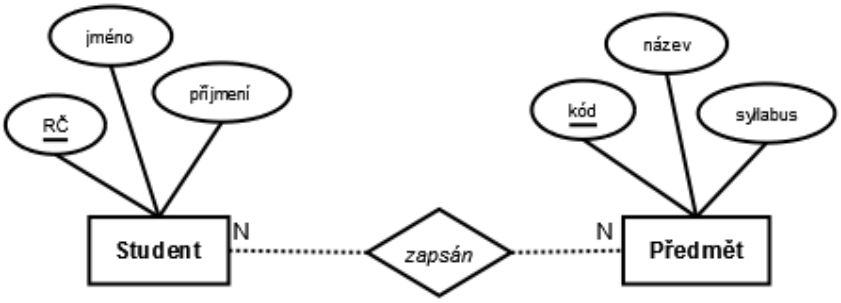
- lze ošetřit, co se stane, když se někdo snaží *smazat* řádek tabulky, na který odkazuje nějaký cizí klíč: ON DELETE ... a co se stane, když se někdo snaží *změnit* obsah buňky, na kterou odkazuje nějaký cizí klíč: ON UPDATE ...
- akce, které můžeme v těchto situacích nařídit (dosadíme za tři tečky výše) jsou:
    - CASCADE – změna či smazání odkazované buňky se kaskádně projeví i v odkazující tabulce
    - SET NULL – cizí klíč v odkazující tabulce se nastaví na hodnotu NULL
    - SET DEFAULT – cizí klíč v odkazující tabulce se nastaví na implicitní hodnotu
    - RESTRICT – akce se zakáže – výchozí chování

```
CREATE TABLE pavilon (
      c_pavilonu   int          primary key check(c_pavilonu>0),
      umisteni     varchar(50)  not null,
      popis_pav    varchar(200) not null
);

CREATE TABLE kotec (
      c_kotce      int   primary key check(c_kotce>0),
      kapacita     int   not null check(kapacita>0),
      typ          varchar(10)  not null,
      c_pavilonu   int references pavilon(c_pavilonu) on delete cascade
);

CREATE TABLE pes (
      ev_c         serial       primary key,
      rasa         varchar(50)  not null,
      narozen      varchar(50)  not null,
      popis_psa    varchar(200) not null,
      c_kotce      int not null references kotec(c_kotce)
                   on delete restrict on update cascade
);
```

při smazání pavilonu se samočinně smažou i kotce, které v něm sídlí

smazání kotce se nepovolí, pokud v něm bydlí nějaký pes; změna čísla kotce se samočinně aplikuje i do cizího klíče

Příklad 6



Relační model:

STUDENT(RČ, jméno, příjmení)
PŘEDMĚT(kód, název, syllabus)
ZAPSÁN(*RČ, kód*)

SQL: budeme požadovat, aby všechny atributy byly neprázdné, tj. not null.

```
CREATE TABLE student (
      rc          varchar(11) primary key,
      jmeno       varchar(50) not null,
      prijmeni    varchar(50) not null
);

CREATE TABLE predmet (
      kod         varchar(3)  primary key,
      nazev       varchar(50) not null,
      syllabus    varchar(50) not null
);
```

V případě tabulky *zapsán* upozorňujeme na nutnost definovat složený primární klíč jako tabulkové integritní omezení. **Nelze tedy psát**:

```
CREATE TABLE zapsan (
      rc    varchar(11) not null references student(rc) primary key,
      kod   varchar(3)  not null references predmet(kod) primary key
);
```

Správný zápis:

```
CREATE TABLE zapsan (
      rc    varchar(11) not null references student(rc),
      kod   varchar(3)  not null references predmet(kod),
      primary key (rc, kod)
);
```

Ještě uvedeme jiný způsob zápisu cizího klíče formou tabulkového IO:

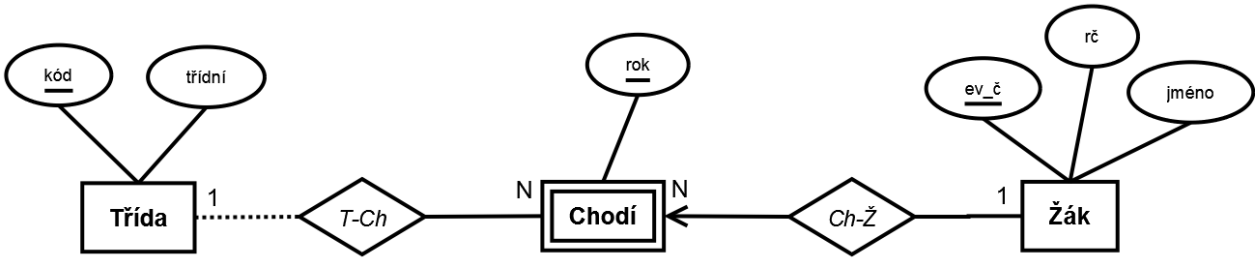
```
CREATE TABLE zapsan (
      rc    varchar(11)  not null,
      kod   varchar(3)   not null,
      foreign key (rc)   references student(rc),
      foreign key (kod)  references predmet(kod),
      primary key (rc, kod)
);
```

Pozor, následující zápis je **rovněž chybně**:

```
CREATE TABLE zapsan (
      rc    varchar(11)       not null,
      kod   varchar(3)        not null,
      foreign key (rc, kod)   references student(rc) predmet(kod),
      primary key (rc, kod)
);
```

Cizí klíč nemá být v tomto příkladu složený, jedná se o dva různé cizí klíče.

Příklad 7



Relační model:

ŽÁK(ev_č, rč, jméno)
TŘÍDA(kód, třídní)
CHODÍ(*kód, ev_č*, rok)

SQL: zde povolíme některé prázdné atributy, to lze vyznačit v definici sloupce buď explicitním uvedením klíčového slova null nebo prostě vynecháním příznaku not null. Zopakujme, že narozdíl od SQL, v relačním modelu prázdné hodnoty atributů neexistují, všechny komponenty n-tic relací musí mít vždy definovanou hodnotu. **Srovnejte** tento příklad s příklady 2 a 3 z této kapitoly.

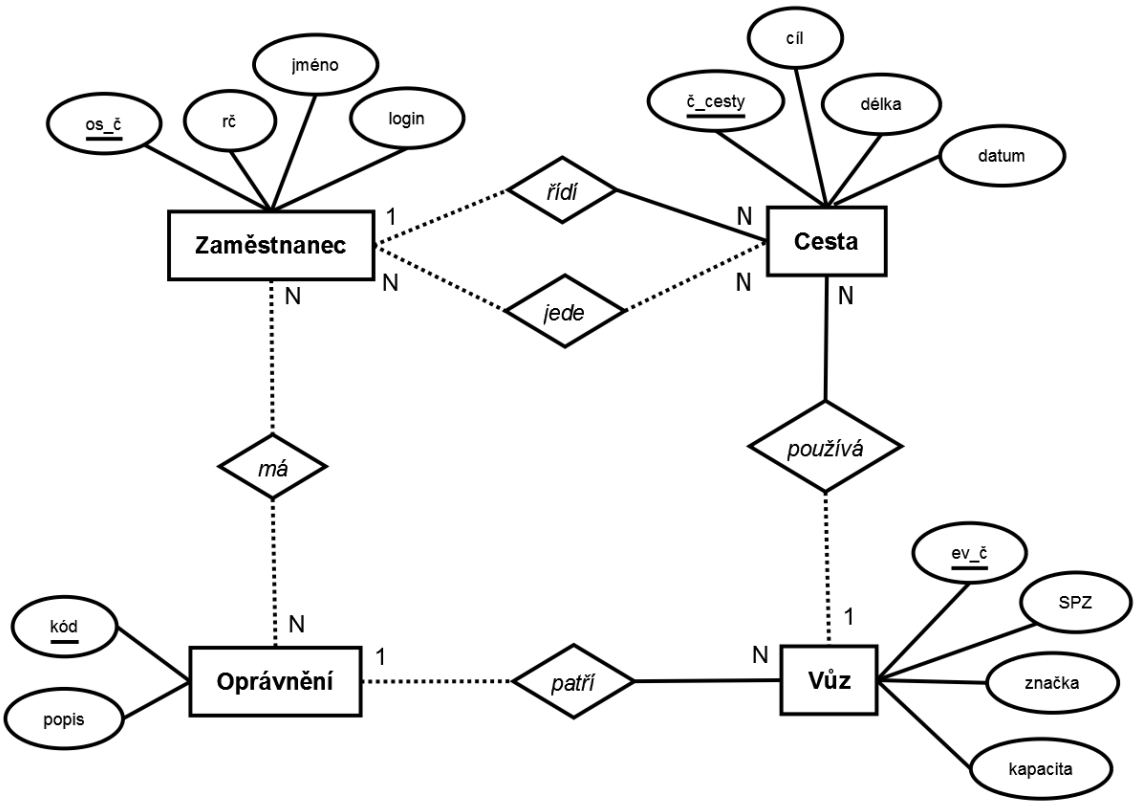
```
CREATE TABLE trida (
      kod         char(2) primary key,
      tridni      varchar(50)
);
```

```
CREATE TABLE zak (
      ev_c         integer      primary key,
      rc           varchar(11)  not null,
      jmeno        varchar(50)  not null
);
```

Ve vztahové tabulce *Chodí* použijeme pro ukázku kontrolu integritního omezení, že sloupec rok musí mít hodnotu mezi čísly 1900 a 2050. U sloupců „ev_č“ a „rok“ není nutné uvádět not null, neboť jejich neprázdnost je automaticky zajištěna tím, že jsou součástí primárního klíče.

```
CREATE TABLE chodi (
      ev_c   varchar(11)  references zak(ev_c),
      kod    char(2)      not null references trida(kod),
      rok    integer      check (rok>1900 and rok<2050),
      primary key (ev_c, rok)
);
```

Příklad 8



Relační model:

ZAMĚSTNANEC(os_č, rč, jméno, login)
OPRÁVNĚNÍ(kód, popis)
VŮZ(ev_č, SPZ, značka, kapacita, *kód*)
CESTA(č_cesty, cíl, délka, datum, *os_č_řidič, ev_č*)
JEDE(*os_č_spolujezdec, č_cesty*)
MÁ(*os_č, kód*)

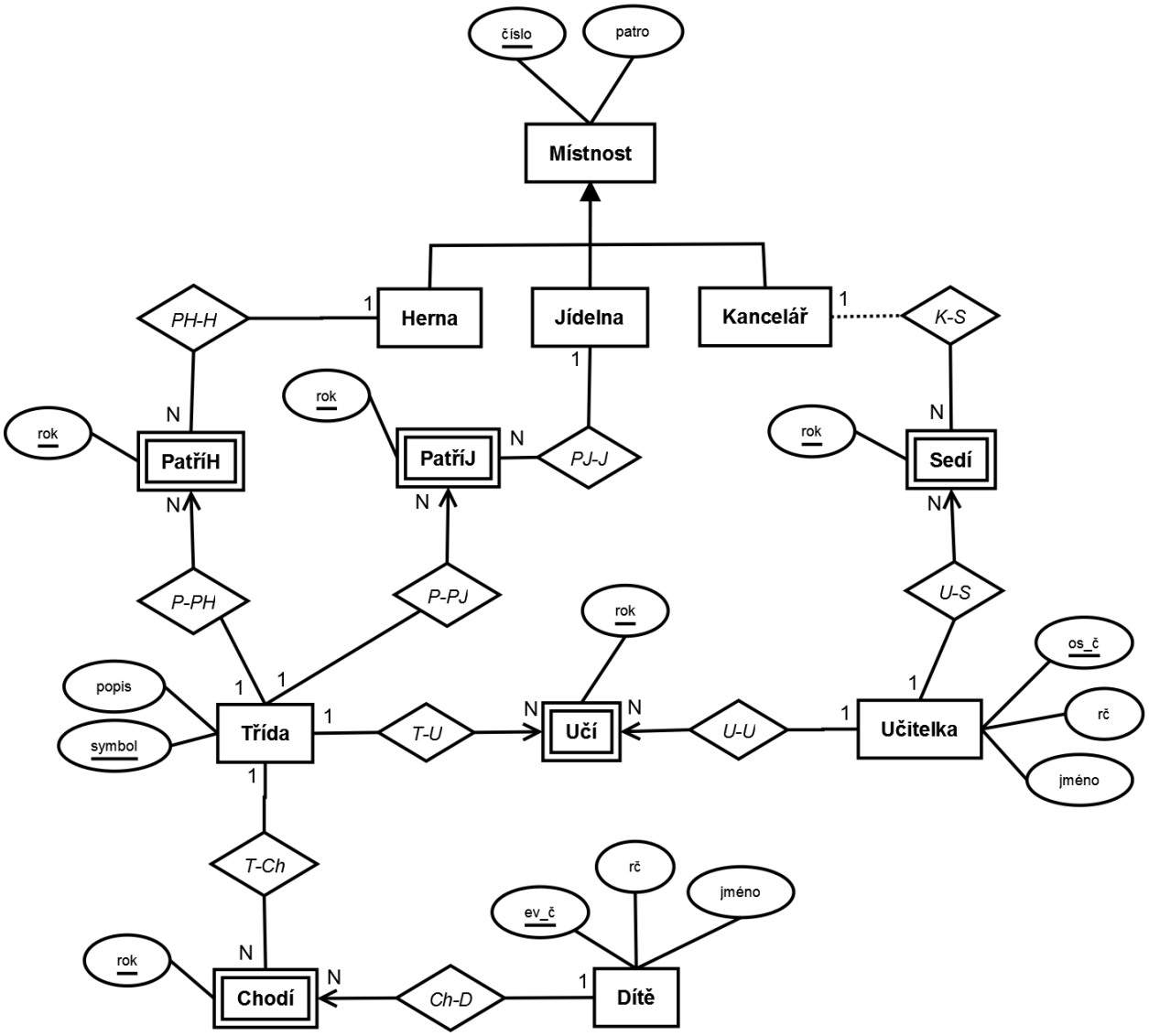
SQL: zopakujme, že u sloupců, které tvoří primární klíč, není nutné uvádět not null, neboť jejich neprázdnost je automaticky vynucena příznakem primárního klíče. Naproti tomu u cizích klíčů není neprázdnost automaticky vynucována, tedy definujeme je jako null nebo not null podle toho, zda má daná tabulka povinnou nebo nepovinnou účast ve vztahu. U tabulky Vůz dále nadefinujeme implicitní (výchozí) hodnotu slouce kapacita na hodnotu 5.

```
CREATE TABLE zamestnanec (
      os_c   integer            primary key,
      rc     varchar(11)        not null,
      jmeno varchar(100)        not null,
      login  char(8)            not null
);

CREATE TABLE opravneni (
      kod    char(3)            primary key,
      popis varchar(100)        not null
);

CREATE TABLE vuz (
      ev_c         integer      primary key,
      SPZ          varchar(10)  not null,
      znacka       varchar(100) not null,
      kapacita     int          not null default 5 check(kapacita>0),
      kod          char(3)      not null references opravneni(kod)
);

CREATE TABLE cesta (
      c_cesty      serial       primary key,
      cil          varchar(100)  not null,
      delka        real         not null check(delka>0),
      datum        date         not null,
      os_c_ridic   varchar(11)  not null references zamestnanec(os_c),
      ev_c         varchar(10)  not null references vuz(ev_c)
);

CREATE TABLE jede (
      os_c_spoluj  varchar(11)  not null references zamestnanec(os_c),
      c_cesty      int          not null references cesta(c_cesty),
      primary key (os_c_spoluj, c_cesty)
);
```

není nutné, sloupce jsou klíčové

```
CREATE TABLE ma (
      os_c   varchar(11)  references zamestnanec(os_c),
      kod    char(3)      references opravneni(kod),
      primary key (os_c, kod)
);
```

Příklad 9



Relační model:

HERNA(<u>číslo_her</u>, patro)
JÍDELNA(<u>číslo_jíd</u>, patro)
KANCELÁŘ(<u>číslo_kanc</u>, patro)
TŘÍDA(<u>symbol</u>, popis)
UČITELKA(<u>os_č</u>, rč_učit, jméno)
DÍTĚ(<u>ev_č</u>, rč_dítě, jméno)
PATŘÍH(*číslo_her, <u>symbol</u>*, <u>rok</u>)
PATŘÍJ(*číslo_jíd, <u>symbol</u>*, <u>rok</u>)
SEDÍ(*číslo_kanc, <u>os_č</u>*, <u>rok</u>)
UČÍ(*<u>symbol, os_č</u>*, <u>rok</u>)
CHODÍ(*symbol, <u>ev_č</u>*, <u>rok</u>)

SQL: připomínáme, že pro ISA hierarchii platí, že entitní podtypy nemají žádné společné prvky a dále že každý prvek entitního nadtypu musí být prvkem právě jednoho podtypu. Převedeno do naší situace, každá místnost musí být buď herna, nebo jídlena nebo kancelář, neexistuje databází evidovaná místnost, která by byla něčím jiným (třeba toaletou). Tento požadavek máme řešen tím, že pro každý entitní typ zavádíme samostatnou relaci, resp. tabulku, jiné místnosti než herny, jídelny a kanceláře tedy v databázi uchovávat nelze. Z druhé strany, je-li nějaká místnost hernou, je POUZE hernou a není zároveň jídelnou ani kanceláří; totéž platí pro ostatní podtypy. V databázi se tento požadavek projeví tak, že čísla heren se mají lišit od čísel jídelen a čísel kanceláří. To ovšem ve většině DBMS nelze zajistit v definici tabulek, je nutné to zajistit jinak, např. pomocí triggeru nebo na úrovni aplikace. Podle normy SQL by mělo být možné definovat integritní omezení obsahující vnořený SELECT blok, v PostgreSQL však tato možnost není implementována.

```
CREATE TABLE herna (
      cislo_her    int primary key check(cislo_her>0),
      patro        int not null check(patro>0)
);

CREATE TABLE jidelna (
      cislo_jid    int primary key check(cislo_jid>0),
      patro        int not null check(patro>0)
);

CREATE TABLE kancelar (
      cislo_kanc   int primary key check(cislo_kanc>0),
      patro        int not null check(patro>0)
);

CREATE TABLE trida (
      symbol       varchar(20) primary key,
      popis        varchar(500) not null
);

CREATE TABLE ucitelka (
      os_c         integer     primary key,
      rc_ucit      varchar(11)  not null,
      jmeno        varchar(50)  not null
);

CREATE TABLE dite (
      ev_c         integer     primary key,
      rc_dite      varchar(11)  not null,
      jmeno        varchar(50)  not null
);
```

```
CREATE TABLE patriH (
      cislo_her    int not null references herna(cislo_her),
      symbol       varchar(20) not null references trida(symbol),
      rok          int not null check(rok>1950 and rok<2050),
      primary key (symbol, rok)
);

CREATE TABLE patriJ (
      cislo_jid    int not null references jidelna(cislo_jid),
      symbol       varchar(20) not null references trida(symbol),
      rok          int not null check(rok>1950 and rok<2050),
      primary key (symbol, rok)
);
CREATE TABLE sedi (
      cislo_kanc   int not null references kancelar(cislo_kanc),
      os_c         varchar(11) not null references ucitelka(os_c),
      rok          int not null check(rok>1950 and rok<2050),
      primary key (os_c, rok)
);

CREATE TABLE uci (
      symbol       varchar(20) not null references trida(symbol),
      os_c         varchar(11) not null references ucitelka(os_c),
      rok          int not null check(rok>1950 and rok<2050),
      primary key (symbol, os_c, rok)
);

CREATE TABLE chodi (
      symbol       varchar(20) not null references trida(symbol),
      ev_c         varchar(11) not null references dite(ev_c),
      rok          int not null check(rok>1950 and rok<2050),
      primary key (ev_c, rok)
);
```

Příklad 10

Použijeme nyní tabulky vytvořené v příkladu 7 této kapitoly a ukážeme si na nich další příkazy jazyka pro definici dat. Nejprve zopakujeme základní definice tabulek.

```
CREATE TABLE trida (
      kod          char(2) primary key,
      tridni       varchar(50)
);

CREATE TABLE zak (
      ev_c         integer      primary key,
      rc           varchar(11)  not null,
      jmeno        varchar(50)  not null
);
```

```
CREATE TABLE chodi (
      ev_c   varchar(11)  references zak(ev_c),
      kod    char(2)      not null references trida(kod),
      rok    integer      check (rok>1900 and rok<2050),
      primary key (ev_c, rok)
);
```

Změny definice tabulky se činí příkazem alter table. Příkaz má velmi mnoho funkcí, procvičíme si alespoň ty základní.

- Změnit sloupec „třídní“ v tabulce Třída tak, aby byl not null:

```
ALTER TABLE trida ALTER COLUMN tridni SET not null;
```

- Změnit sloupec „třídní“ v tabulce Třída tak, aby byl implicitně prázdný řetězec:

```
ALTER TABLE trida ALTER COLUMN tridni SET DEFAULT '';
```

- Změnit sloupec „jméno“ v tabulce Žák tak, aby nebyl not null:

```
ALTER TABLE zak ALTER COLUMN jmeno DROP not null;
```

- Přidat sloupec „místnost“ k tabulce Třída:

```
ALTER TABLE trida ADD COLUMN mistnost int not null default 1;
```

definice sloupce podle stejných pravidel jako v příkazu CREATE TABLE

- Přidat k tabulce Třída nepojmenované tabulkové integritní omezení, kontrolující, aby sloupec místnost byl větší než 0

```
ALTER TABLE trida ADD CHECK(mistnost>0);
```

- Přidat k tabulce Třída tabulkové integritní omezení pojmenované „maxmist“, kontrolující, aby sloupec místnost byl menší než 200

```
ALTER TABLE trida ADD CONSTRAINT maxmist CHECK(mistnost<200);
```

definice tabulkového IO podle stejných pravidel jako v příkazu CREATE TABLE

- Odebrat z tabulky Třída integritní omezení pojmenované „maxmist“

```
ALTER TABLE trida DROP CONSTRAINT maxmist;
```

- Přejmenovat sloupec „místnost“ v tabulce Třída na „c_mistnosti“

```
ALTER TABLE trida RENAME COLUMN mistnost TO c_mistnosti;
```

- Odebrat sloupec „c_mistnosti“ z tabulky Třída:

```
ALTER TABLE trida DROP COLUMN c_mistnosti;
```

Vkládání dat do tabulky se činí příkazem insert into. Příkaz má více variant, lze vkládat do všech sloupců naráz nebo pouze do vybraných sloupců. Lze vkládat hromadně více řádků naráz a to buď přímým zápisem hodnot nebo lze vložit výsledek dotazu.

- Vložit jeden řádek do tabulky Třída, vkládáme data do všech sloupců naráz:

```
INSERT INTO trida VALUES ('1A','Petr Kus');
```

- Vložit jeden řádek do tabulky Třída, vkládáme data jen do vybraných sloupců. Zbylé sloupce jsou nastaveny buď na jejich výchozí (default) hodnotu, nebo v případě nedefinované defaultní hodnoty na NULL. Pokud je sloupec definován jako not null, zároveň nemá definovanou výchozí (default) hodnotu a v příkazu INSERT do něj nevkládáme data, příkaz INSERT se neprovede z důvodu porušení integritního omezení not null.

```
INSERT INTO trida(kod) VALUES ('1B');
```

výběr sloupců, do nichž vkládáme data

- Vložit hromadně více řádků do tabulky Žák:

```
INSERT INTO zak VALUES
      ('1','123456/1234','Iva Mala'),
      ('2','123457/1234','Petr Fucik'),
      ('3','123458/1234','Mia Farrow'),
      ('4','123459/1234','Ian Tyson');
```

- Vložit řádky do tabulky Chodí:

```
INSERT INTO chodi VALUES
      ('1', '1A', 2010),
      ('2', '1A', 2011),
      ('3', '1B', 2010),
      ('4', '1B', 2011);

INSERT INTO chodi VALUES ('1', '3A', 2010);
```

porušení referenční integrity: řádek nebude vložen, neb hodnota ‘3A’ není obsažena v tabulce třída

- Vytvořit tabulku Chodí2010 obsahující ty řádky z tabulky Chodí, které mají rok 2010:

```
CREATE TABLE chodi2010 AS SELECT * FROM chodi WHERE rok=2010;
```

vytvoří tabulku na základě dat vybraných z jiné tabulky

výběr dat z tabulky – bude procvičeno v další kapitole

- Přejmenovat tabulku Chodí2010 na Patří:

```
ALTER TABLE chodi2010 RENAME TO patri;
```

- Přidat do tabulky Patří záznamy z tabulky Chodí, které mají rok 2011:

```
INSERT INTO patri SELECT * FROM chodi WHERE rok=2011;
```

- Odstranit tabulku Patří:

```
DROP TABLE patri;
```

Úprava existujících dat v tabulce se činí příkazem update. Tento příkaz změní hodnoty v zadaných sloupcích na nové hodnoty, lze se přitom odkazovat na původní hodnoty sloupců. Hodnoty se změní v těch řádcích, které splní zadanou logickou podmínku. Není-li zadána žádná podmínka, budou ovlivněny všechny řádky tabulky.

- Změnit jméno dívky Iva Malá na Yvette Velká:

```
UPDATE zak SET jmeno='Yvette Velka' WHERE jmeno='Iva Mala';
```

nastavení nové hodnoty sloupce jméno

logická podmínka

- Zvýšit v tabulce Chodí hodnotu sloupce „rok“ o jedničku (ve všech řádcích):

```
UPDATE chodi SET rok=rok+1;
```

lze se odkazovat na původní hodnoty

- Změnit v tabulce Třída jméno třídního u třídy 1A na výchozí (default) hodnotu:

```
UPDATE trida SET tridni = DEFAULT;
```

- Test, jak funguje referenční integrita. Změňte v tabulce Žák evidenční číslo Yvetty Velké na '18'. Operace by neměla proběhnout, protože na rodné číslo v tabulce Žák se odkazuje záznam z tabulky Chodí. Jelikož byl cizí klíč „ev_č“ v tabulce Chodí definován bez explicitního řízení referenční

integrity, je to jako kdybychom ho definovali s příznakem ON DELETE RESTICT ON UPDATE RESTRICT, operace se tedy nepovolí.

```
UPDATE zak SET ev_c = '18' WHERE jmeno = 'Yvette Velka';
```

- V návaznosti na minulý bod změníme u tabulky Chodí vlastnosti cizího klíče „ev_č“ tak, aby obsahoval řízení referenční integrity ON UPDATE CASCADE. Referenční IO nelze u sloupce přímo měnit, je nutné jej nejprve odstranit příkazem DROP CONSTRAINT a poté znovu nadefinovat jinak. Jelikož toto referenční integritní omezení jsme při definici tabulky nepojmenovali, vzniká ovšem otázka, jak je odstranit. Řešení je následující: databázový stroj PostgreSQL všechna integritní omezení pojmenovává svými vnitřními názvy, pokud není dodán název uživatelem. Vypíšeme-li si vlastnosti tabulky Chodí příkazem \d chodi, lze si přečíst, jaký název byl přiřazen hledanému IO. Nejspíše půjde o název podobný tomuto: „chodi_ev_c_fkey“. Pomocí tohoto názvu odstraníme nevyhovující integritní omezení a následně jej nadefinujeme znovu a lépe (tentokrát ovšem ve formě nepojmenovaného tabulkového IO).

```
ALTER TABLE chodi DROP CONSTRAINT chodi_ev_c_fkey;
ALTER TABLE chodi ADD FOREIGN KEY (ev_c) REFERENCES zak(ev_c) ON UPDATE CASCADE
ON DELETE CASCADE;
```

- Změňte v tabulce Třída jméno Yvetty Velké zpět na Iva Malá a zároveň její evidenční číslo změňte na '18'. Po provedení příkazu se můžete příkazem `select * from chodi;` podívat, že se díky definici cizího klíče ON UPDATE CASCADE rodné číslo nyní změnilo i ve vázané tabulce.

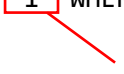
```
UPDATE trida SET jmeno = 'Iva Mala', ev_c = '18'
WHERE jmeno = 'Yvette Velka';
```

jiný zápis téhož

```
UPDATE trida SET (jmeno, ev_c) = ('Iva Mala', '18')
WHERE jmeno = 'Yvette Velka';
```

- Test, jak funguje kontrola unikátnosti primárního klíče. Změňte v tabulce Třída evidenční číslo Petra Fučíka na '1':

```
UPDATE trida SET ev_c = '1' WHERE jmeno = 'Petr Fucik';
```



operace neproběhne, protože se
porušuje unikátnost primárního klíče:
číslo 123456/5555 už v tabulce existuje

Odstranění řádků z tabulky se provede příkazem `delete from`. Příkaz odstraní ty řádky, které splní zadanou logickou podmínku. Není-li zadána žádná podmínka, odstraní se všechny řádky tabulky, příkaz ovšem neodstraní tabulku jako takovou (to se zařídí příkazem DROP TABLE). Jelikož jsme nadefinovali cizí

klíč „ev_č“ v tabulce Chodí též jako ON DELETE CASCADE, odstraní se i vázané řádky z tabulky Chodí, přesvědčte se o tom příkazem SELECT.

- Odstraňte žáky, kteří mají ve jméně písmeno F:

```
DELETE FROM zak WHERE jmeno LIKE '%F%';
```

test podobnosti řetězců: zástupný znak % reprezentuje libovolný řetězec (i prázdný), znak _ reprezentuje právě jeden libovolný znak

Úlohy k procvičení

1. Uvažujme seznam kin a seznam filmů.
   kino má atributy: název, adresa, počet míst, Dolby ano/ne
   film má atributy: název, rok vzniku, země vzniku, dabing ano/ne

   Uvažujte několik variant v kombinacích, řešte bez a s použitím prázdných hodnot (NULL):
   A. v každém kině se může dávat víc filmů, každý film se může dávat pouze v jednom kině.
   B. v každém kině se může dávat víc filmů a každý film se může dávat ve více kinech.
   1. chceme evidovat jen filmy, které jsou někde dávány
   2. chceme evidovat všechny filmy

   Vytvořte a proveďte následující příkazy v jazyce SQL:

   - definujte všechny tabulky v daných variantách, vč. IO, zaměřte se na použití IO sloupce a IO tabulky
   - vytvořte tabulky s cizím klíčem, při smazání odkazovaného řádku se smažou i odkazující
   - přidejte k tabulce KINO sloupec jméno vedoucího
   - odstraňte z tabulky FILM informaci o dabingu
   - změňte název sloupce „počet míst“ na "kapacita"
   - přidejte k tabulce KINO sloupec id_kina
   - vložte do tabulek nějaká data (aspoň jedno kino z Jihlavy)
   - vytvořte tabulku "cizí_filmy", do níž vložíte všechny filmy, které nejsou z ČR
   - změňte u filmů z ČR zemi z ČR na Česko
   - odstraňte z tabulky KINO všechna kina z Jihlavy

---

2. Vytvořte v SQL tabulky pro Vaše relace vytvořené v příkladech ke kapitole 6.

8 Jazyk SQL – příkaz SELECT

Cíl kapitoly

V této kapitole se seznámíme s použitím příkazu SELECT, který slouží v jazyce SQL k výběru dat z tabulek. Bude též uvedena souvislost sémantiky příkazu SELECT s relační algebrou. Cílem je naučit se zejména

- obecně syntaxi příkazu SELECT
- obecně sémantiku příkazu SELECT
- blok SELECT-FROM-WHERE
- použití agregačních funkcí
- konstrukt GROUP BY
- množinové operace
- spojení

Klíčové pojmy

Jazyk pro manipulaci s daty, příkaz SELECT, klauzule FROM, WHERE, ORDER BY, GROUP BY, HAVING, NATURAL JOIN, LEFT JOIN.

Úvod

Příkaz SELECT slouží k formulaci dotazů nad tabulkou. Vstupem jsou tabulky (tj. v podstatě relace), výstupem je opět tabulka (tj. v podstatě relace – rozdíl mezi tabulkou a relací viz předchozí přednášky). Pořadí řádků výsledku není zaručeno (bez explicitního zadání).

Sémantiku dotazů lze dobře vyjádřit relační algebrou, ovšem pozor: ani výraz v relační algebře, ani výraz v SQL nedefinuje způsob provedení dotazu, neboť SŘBD provádí optimalizaci dotazů. Rozdíl výraz vs. dotaz viz předchozí přednášky

Blok SELECT-FROM-WHERE (zjednodušeně)

```
SELECT col1, col3
FROM table_name
WHERE condition
```

Obecněji (s přepisem do relační algebry)

```
SELECT A1, A2, ..., An
FROM R1, R2, ... , Rm
WHERE φ
```

odpovídá

$$(R1 \times R2 \times ... \times Rm)(\varphi)\ [A1, A2, ..., An]$$

- neexistence podmínky se chápe jako TRUE

- přepis do relační algebry není zcela ekvivalentní, neboť SQL má oproti RMD některé odlišnosti
  - standardní SELECT neodstraňuje z výsledku duplicitní n-tice (tj. výsledek nemusí být relace), to se zařídí klauzulí DISTINCT
  - SQL uvažuje prázdné hodnoty, tj. tříhodnotová logika apod.
- chceme-li vybrat všechny sloupce, místo jejich výčtu lze použít hvězdičku *
- chceme-li výsledek uspořádat, přidáme klauzuli

```
ORDER BY col_name [{ASC | DESC}
```

Schema použité v příkladech

Máme databázi silnic a mostů, které na nich leží

- na jedné silnici může stát víc mostů
- jeden most může stát de iure na více silnicích
  - určitý úsek silnice může být očíslován více čísly
- atributy silnice
  - číslo, třída, start, cíl, pověřený vlastník, délka, datum poslední opravy
- atributy mostu
  - číslo, nosnost, šířka, technologie

Silnice

číslo_s	třída	start	cíl	vlastník	délka	oprava
21	1	D5	Německo	stát	62	12.6.2009
53	1	Znojmo	Pohořelice	stát	38	30.5.2007
230	2	Nepomuk	Bečov nad Teplou	Plzeňský kraj	88	14.2.1999
276	2	Bělá pod Bezdězem	Kněžmost	Liberecký kraj	21	21.8.2003
403	2	Kouty	Telč	kraj Vysočina	36	2.10.2001
602	2	Pelhřimov	Starý Lískovec	kraj Vysočina	108	5.8.2008
0395	3	Kostelec	Cejle	kraj Vysočina	2,5	28.9.1992
1314	3	Štoky	Smrčná	kraj Vysočina	6,3	10.7.2000

Most

číslo_m	nosnost	šířka	technologie
2469	55	11	beton
8963	40	8	beton
8965	42	9	beton
500489	3,5	4	kámen

Leží_na

číslo_m	číslo_s
2469	21
2469	230
8963	602
8965	602
500489	0395

Jednoduché dotazy

```
SELECT *
FROM Silnice;                    -- vrátí celou tabulku Silnice
```

```
SELECT číslo_s, délka
FROM Silnice
WHERE vlastník = 'kraj Vysočina' AND třída < 3;
```

číslo_s	délka
403	36
602	108

```
SELECT DISTINCT technologie
FROM Most
WHERE nosnost < 50
ORDER BY technologie DESC;
```

technologie
kámen
beton

Příkaz SELECT detailněji – 1. část

Příkaz sestává z 2 až 5 klauzulí (+ ORDER BY)

1. SELECT
    - projekce na sloupce; definice nových a agregovaných sloupců
2. FROM
    - seznam tabulek, z nichž čerpáme, je-li jich zadáno víc, provede se kartézský součin
3. WHERE
    - selekce řádků z relace dané klauzulí FROM
4. GROUP BY
    - přes které atributy se výsledek předchozího agreguje
5. HAVING
    - selekce řádků z relace dané klauzulí GROUP BY

Klauzule SELECT

```
SELECT    [{ALL | DISTINCT}]    výraz
```

Výraz může obsahovat

- výčet sloupců nebo hvězdičku (za všechny sloupce)
- konstantu – ve výsledku bude sloupec plný konstanty
- agregační funkci
- DISTINCT odstraňuje z výsledku duplicitní řádky
    - vliv na agregační funkce, vstupuje do nich méně řádků
- prvek výrazu může být pojmenovaný

```
SELECT číslo_s AS 'Číslo silnice'
```

- v PostgreSQL se dá samotný SELECT použít pro jednoduché výpočty nebo volání funkcí

```
SELECT 1+1;
SELECT power(2, 8);
```

Agregační funkce

Slouží k výpočtu nějaké souhrnné hodnoty, kombinující údaje z více řádků (tj. ze skupiny řádků). Nejčastěji se používají v kombinaci s klauzulí GROUP BY (viz dále), není-li klauzule GROUP BY přítomna, vstupem agregační funkce jsou všechny řádky dotazu. Agregační funkce se aplikuje vždy na jeden sloupec.

K nejčastějším agregacím patří určení počtu prvků, součet, průměr, maximum, minimum apod., PostgreSQL zná ještě např. směrodatnou odchylku, rozptyl, ad.

Agregační funkce

- pracují na množině řádků s daným sloupcem:

```
agregační_funkce ( [{ALL | DISTINCT}] jméno_sloupce )
```

např. COUNT(col) – počet neprázdných prvků ve sloupci col
analogicky SUM, MAX, MIN, AVG

- bez zadání DISTINCT počítají s duplicitními řádky
- pozor na prázdné hodnoty NULL
    - každá agregační funkce se k hodnotám NULL chová jinak: COUNT(NULL) vrací nulu (0), SUM(NULL) vrací NULL

```
SELECT MAX(délka) AS ‘Nejdelší‘
FROM Silnice
```

Bez zadání GROUP BY je nelze za SELECTem kombinovat s dalšími sloupci (viz dále), tj. není možné například:

```
SELECT příjmení, AVG(plat)
FROM osoba;
```

- V uvedeném příkladu totiž agregační funkce vytvoří z celé tabulky osoba jediné číslo - průměrný plat všech osob; stroj by tudíž nevěděl, jaké příjmení z mnoha možných má tomuto jedinému číslu přiřadit. Dotaz tedy nemá smysluplnou sémantiku a jako takový není povolen.

Klauzule FROM

```
FROM seznam_tabulek
```

- je-li v seznamu více než jedna tabulka, do dalšího vyhodnocení vstupuje jejich kartézský součin
- místo skutečných tabulek lze (v některých SŘBD) použít i pohledy nebo vnořený SELECT blok
- prvek seznamu může být pojmenovaný (2 způsoby)

```
... FROM Most AS M
... FROM Most M
```

Klauzule WHERE

```
WHERE logická_podmínka_selekce
```

- ze základní kolekce n-tic, dané výsledkem klauzule FROM (tj. tabulka, kart. součin, JOIN) do výsledku postupují jen ty n-tice, které splňují podmínku – selekce
- logické spojky AND, OR, NOT
- např.

```
SELECT číslo_s, délka
FROM Silnice
WHERE vlastník = 'kraj Vysočina' AND třída < 3
```

- test na vztah mezi dvěma atributy či atributem a hodnotou

  =, <>, <, >, <=, >=
- interval hodnot

  výraz1 [NOT] BETWEEN výraz2 AND výraz3
- srovnání řetězců

  výraz1 [NOT] LIKE maska
  - maska může obsahovat

    % (libovolný řetězec)

    _ (libovolný jeden znak)
  - např.

```
SELECT * FROM Silnice
WHERE vlastník LIKE '%kraj%'
```

- test na (ne)prázdnou hodnotu

  výraz IS [NOT] NULL
- příslušnost prvku do množiny

  výraz1 [NOT] IN výraz2
  - odpovídá zápisu výraz1 $\in$ výraz2 resp. výraz1 $\notin$ výraz2
  - např.

```
SELECT * FROM Silnice
WHERE vlastník IN ('stát', 'kraj Vysočina', 'Liberecký kraj')

SELECT třída FROM Silnice
WHERE číslo_s NOT IN (SELECT číslo_s FROM Leží_na)
```

- test (ne)prázdnosti množiny

  [NOT] EXISTS poddotaz

- např.

```
SELECT * FROM Silnice S
WHERE EXISTS (SELECT * FROM Leží_na L
                    WHERE S.číslo_s = L.číslo_s)
```

- rozšířené kvantifikátory

  výraz1 { = | <> | < | > | <= | >= } { ANY | ALL } poddotaz

  { aspoň jeden řádek | všechny řádky } poddotazu vyhovují srovnání

  - např.

```
SELECT * FROM Silnice
WHERE délka > ALL (SELECT délka FROM Silnice
                          WHERE vlastník = 'stát')
```

Dotazy nad více tabulkami – příklady

Najdi čísla silnic, na nichž leží nějaký most

- stačí sáhnout do jediné tabulky – všechny silnice s mosty jsou totiž evidovány ve vztahové tabulce Leží_na

```
SELECT číslo_s
FROM Leží_na;
```

číslo_s
21
230
602
602
0395

Najdi čísla a délky silnic, na nichž leží nějaký most

- tečková notace odkazů na použité tabulky

```
SELECT číslo_s, délka
FROM Silnice S, Leží_na L
WHERE S.číslo_s = L.číslo_s
```

číslo_s	délka
21	62
230	88
602	108
602	108
0395	2,5

- sémantika:
  - struktura použitých tabulek je následující:

    Tabulka *Leží_na* je vztahová tabulka, reprezentující vztah M:N mezi silnicí a mostem, cizími klíči odkazuje jednak na tabulku Silnice (na číslo_s), jednak na tabulku Most (na číslo_m). Podíváme-li se do sloupce číslo_s v tabulce Leží_na, budou v něm uložena čísla všech silnic, na kterých leží nějaký most. Pokud na nějaké silnici leží mostů více, bude se číslo silnice ve sloupci opakovat tolikrát, kolik mostů na dané silnici leží. Za každý výskyt vztahu most-silnice musí být v tabulce jeden záznam.

- uvedený příklad tedy pracuje takto:
  - najde v tabulce Silnice takové záznamy, k nimž existuje odpovídající záznam v tabulce Leží_na
  - z vybraných záznamů pak zobrazí pouze sloupce číslo_s a délka.

  přesněji
  - nejprve se vytvoří kartézský součin tabulky Silnice a Leží_na
  - poté se z kombinovaných řádků vyberou takové, u nichž platí, že číslo_s původem z tabulky Silnice a číslo_s původem z tabulky Leží_na jsou stejná

- jak by se uvedeného docílilo v relační algebře?
  - můžeme pochopitelně také využít kartézského součinu a následné selekce s projekcí

    (Silnice x Leží_na)(S.číslo_s = L.číslo_s)[S.číslo_s, délka]

  - výhodnější je ale použít operaci spojení - v tomto případě přirozené spojení

    (Silnice * Leží_na)[číslo_s, délka]

- příklad vlastně v SQL realizoval operaci přirozeného spojení, tj. speciální případ spojení přes rovnost: onu rovnost odpovídajících si atributů v SQL zápisu hezky vidíme

Najdi unikátně čísla silnic, na nichž leží nějaký most, v prvním sloupci nechť je slovo „Číslo:“

```
SELECT DISTINCT ‘Číslo:‘, číslo_s
FROM Leží_na;
```

‘Číslo:‘	číslo_s
Číslo:	21
Číslo:	230
Číslo:	602
Číslo:	0395

Najdi šířky mostů v cm, které leží na silnicích první třídy

- aritmetika

```
SELECT M.šířka * 100
FROM Most M, Silnice S, Leží_na L
WHERE
        S.číslo_s = L.číslo_s
        AND M.číslo_m = L.číslo_m
        AND S.třída = 1
```

- sémantika, jak to pracuje
    - nejprve se podíváme, ve kterých tabulkách se nalézají požadované informace:
        - šířky mostů jsou v tabulce Most
        - třídy silnic jsou v tabulce Silnice
        - který most na které silnici leží najdeme v Leží_na
    - při řešení úlohy musíme tedy sáhnout do všech tří tabulek
    - v relační algebře bychom opět použili spojení, selekci a projekci:

      (Silnice * Leží_na * Most)(třída=1)[šířka]

Příkaz SELECT detailněji – 2. část

Klauzule GROUP BY (column)

GROUP BY pracuje tak, že n-tice vystoupivší z klauzulí FROM a WHERE se poslučují podle sloupce *column* tak, že za každou skupinu n-tic se stejnými hodnotami ve sloupci column se do výsledku dostane jedna n-tice. Ve slučujícím sloupci je pak právě ta hodnota z řádků, z nichž byl „superřádek“ sloučen (byly všecky stejné). Za GROUP BY lze uvést více sloupců, pak se seskupuje podle všech možných kombinací.

Co ostatní sloupce (kde mohla být obecně v každém původním řádku jiná hodnota)?

- buď se ve výsledku neobjeví
- nebo se na jejich hodnoty použije nějaká agregační funkce a vyrobí se jediná hodnota
    - zobecnění agreg. funkcí, nevznikne jednořádková tabulka jako předtím, ale tabulka s tolika řádky, kolik dá GROUP BY

Máme-li v dotazu tuto klauzuli, co smíme mít za klauzulí SELECT?

- ty sloupce, které jsou uvedeny za GROUP BY, tj. podle kterých se slučuje
- agregační funkce nad jinými sloupci, které pro každou skupinu řádků vygenerují jednoznačnou hodnotu
- nic jiného tam být nesmí, i když se nám zdá, že by to třeba fungovalo, protože víme, že hodnoty jsou jednoznačné – překladač to nepozná

Příklad:
najdi celkovou délku silnic pro každého vlastníka

```
SELECT vlastník, SUM(délka) AS ‘Celková délka‘
FROM Silnice
GROUP BY vlastník
```

číslo_s	třída	start	cíl	vlastník	délka	oprava
21	1	D5	Německo	stát	62	12.6.2009
53	1	Znojmo	Pohořelice	stát	38	30.5.2007
230	2	Nepomuk	Bečov nad Teplou	Plzeňský kraj	88	14.2.1999
276	2	Bělá pod Bezdězem	Kněžmost	Liberecký kraj	21	21.8.2003
403	2	Kouty	Telč	kraj Vysočina	36	2.10.2001
602	2	Pelhřimov	Starý Lískovec	kraj Vysočina	108	5.8.2008
0395	3	Kostelec	Cejle	kraj Vysočina	2,5	28.9.1992
1314	3	Štoky	Smrčná	kraj Vysočina	6,3	10.7.2000

vlastník	Celková délka
stát	100
Plzeňský kraj	88
Liberecký kraj	21
kraj Vysočina	152,8

Klauzule HAVING

- kriterium pro skupiny vzniklé v GROUP BY

```
HAVING logická_podmínka_selekce
```

- následuje za GROUP BY a vybírá pouze ty „superřádky“, které vyhovují podmínce
- analogie klauzule WHERE za FROM

Příklad:

najdi celkovou délku silnic pro každého vlastníka, zobraz však jen ty, které mají celkovou délku větší nebo rovno 100km

```
SELECT vlastník, SUM(délka) AS ‘Celková délka‘
FROM Silnice
GROUP BY vlastník
HAVING ‘Celková délka‘ >= 100
```

vlastník	Celková délka
stát	100
~~Plzeňský kraj~~	~~88~~
~~Liberecký kraj~~	~~21~~
kraj Vysočina	152,8

vlastník	Celková délka
stát	100
kraj Vysočina	152,8

Množinové operace

Příkaz SELECT vrací tabulku (někdy i relaci), můžeme tedy na výstupy více příkazů SELECT použít množinové operace:

- UNION
- INTERSECT
- EXCEPT

Operandy (tj. tabulky) musí být kompatibilní. Bez zadání klauzule ALL se automaticky eliminují duplicity (tj. výsledek je relace).

Příklad

najdi čísla silnic, na nichž neleží žádný most

Analýza: jsou to takové silnice, jejichž číslo není uvedeno v tabulce Leží_na. Výsledek tedy získáme například tak, že vezmeme seznam čísel všech silnic a od něj množinově odečteme seznam čísel silnic, na nichž leží aspoň jeden most. To, co zbyde, jsou tudíž čísla silnic bez mostů.

```
(SELECT číslo_s FROM Silnice)
EXCEPT
(SELECT číslo_s FROM Leží_na);
```

Spojení

Spojení je jedna z nejdůležitějších operací používaných v databázi, umožňuje integrovat data z více tabulek, nejčastěji z těch, mezi nimiž je nějaký vztah. Ve starších verzích SQL bylo nutné spojení zapisovat explicitně pomocí kartézského součinu a následné selekce za klauzulí WHERE (viz jednoduchý příklad uvedený výše). Novější verze SQL (ne však všechny SŘBD) umožňují i elegantnější zápis spojení za klauzulí FROM, a to spojení všech možných druhů: přirozené, theta-spojení (přes podmínku), vnější spojení.

Implementace spojení nejrůznějšího druhu

- pomocí dosud probraných konstruktů
  - vnitřní Θ-spojení nebo přirozené spojení zkonstruujeme jako omezený kartézský součin
    `SELECT ... FROM tbl1, tbl2 WHERE tbl1.A = tbl2.B`
  - levé/pravé polospojení se definuje výběrem sloupců za SELECT
    `SELECT tbl1.* FROM tbl1, tbl2 WHERE tbl1.A = tbl2.B`
- nové konstrukty SQL92
  - kartézský součin
    `... FROM tbl1 CROSS JOIN tbl2 ...`
  - přirozené spojení
    `... FROM tbl1 NATURAL JOIN tbl2 ...`
  - spojení přes podmínku
    `... FROM tbl1 JOIN tbl2 ON A<B ...`

- levé/pravé/plné vnější spojení
  ... FROM tbl1 NATURAL OUTER JOIN tbl2 ...
  ... FROM tbl1 {LEFT | RIGHT | FULL} {OUTER | INNER } JOIN tbl2 ON cond ...

Spojení v PostgreSQL

- vnější spojení se předpokládá, kdykoli se použije slovo LEFT, RIGHT nebo FULL, tj.
  ... FROM tbl1 LEFT JOIN tbl2 ...
  je stejné jako
  ... FROM tbl1 LEFT OUTER JOIN tbl2 ...

  - analogicky pro ostatní vnější spojení

- existuje ještě konstrukt USING
  ... FROM tbl1 JOIN tbl2 USING(a, b)
  což je stejné jako

  ... FROM tbl1 JOIN tbl2 ON tbl1.a = tbl2.a AND tbl1.b = tbl2.b ...

  USING ovšem do výsledku pustí jen jednu kopii sloupce *a* a jednu kopii sloupce *b*, ne obě

Příklad (viz též minule)
najdi čísla a délky silnic, na nichž leží nějaký most

číslo_s	délka
21	62
230	88
602	108
602	108
0395	2,5

- bez spojení (staré SQL)

```
SELECT číslo_s, délka
FROM Silnice S, Leží_na L
WHERE S.číslo_s = L.číslo_s
```

- se spojením

```
SELECT číslo_s, délka
FROM Silnice NATURAL JOIN Leží_na;
```

nebo

```
SELECT číslo_s, délka
FROM Silnice JOIN Leží_na USING(číslo_s)
```

nebo

```
SELECT číslo_s, délka
FROM Silnice S JOIN Leží_na L ON S.číslo_s=L.číslo_s;
```

Příklad (viz též minule)
najdi šířky mostů v cm, které leží na silnicích první třídy

- bez spojení

```
SELECT M.šířka * 100
FROM Most M, Silnice S, Leží_na L
WHERE
        S.číslo_s = L.číslo_s
        AND M.číslo_m = L.číslo_m
        AND S.třída = 1
```

- se spojením

```
SELECT šířka * 100
FROM    Most    NATURAL JOIN    Leží_na    NATURAL JOIN Silnice
WHERE    třída = 1;
```

- při spojování více tabulek je třeba dbát na správné pořadí tabulek, jistota je ohraničit každou spojovanou dvojici závorkami

Nové konstrukty pro spojení
- podstatně zvyšují čitelnost a srozumitelnost kódu
- jazyk SQL se více přiblížil relační algebře
    - nyní má tedy znalost relační algebry ještě větší opodstatnění, neboť je to formální popis operací nezávislý na implementaci

- v praxi je podstatné především přirozené spojení
    - pokud při definici cizích klíčů striktně dodržujeme zásadu, že se cizí klíč i primární klíč odkazované tabulky jmenují stejně, pak nám přirozené spojení stačí ve většině situací
    - **Upozornění**: konstrukt NATURAL JOIN funguje korektně pouze tehdy, když se ve spojovaných tabulkách sloupce jmenují skutečně zcela stejně! Pokud mají sloupce odlišné názvy, pak výsledek operace „NATURAL JOIN“ není přirozené spojení, a to ani tehdy, pokud je mezi tabulkami navázána referenční integrita pomocí FOREIGN KEY! Pokud si tedy nejsme jisti, je bezpečnější používat konstrukci JOIN ON nebo JOIN USING, popřípadě spojení pomocí podmínky WHERE (staré SQL).

Kontrolní otázky

- Jak se liší klauzule WHERE a HAVING?
- Jaké znáte různé syntaktické zápisy, realizující přirozené spojení?
- Jak funguje konstrukt GROUP BY?

Řešené příklady

Použijeme především příklady z kapitoly 5.
Mějme databázi studentů a předmětů, na které se zapisují a které absolvují. Databáze je realizována třemi relacemi se schematy (pozn. primární klíč rodné číslo není optimální, viz kapitola 3):

STUDENT(RČ, JMÉNO, SPECIALIZACE)
PŘEDMĚT(KÓD, NÁZEV, SYLLABUS, GARANT)
ZÁPIS(*RČ, KÓD*, SEMESTR, ZNÁMKA)

Pro tyto relace vytvoříme v SQL tabulky, datové typy a integritní omezení zvolíme intuitivně. Tabulky budeme zapisovat s velkým písmenem na začátku, sloupce malými písmeny a klíčová slova SQL kapitálkami.

```
CREATE TABLE Student (
      rc VARCHAR(11) PRIMARY KEY,
      jmeno VARCHAR(50) NOT NULL,
      specializace VARCHAR(10) NOT NULL
);

CREATE TABLE Predmet (
      kod CHAR(5) PRIMARY KEY,
      nazev VARCHAR(20) NOT NULL,
      syllabus TEXT,
      garant VARCHAR(50) NOT NULL
);

CREATE TABLE Zapis (
      rc VARCHAR(11) REFERENCES STUDENT(RC),
      kod CHAR(5) REFERENCES PREDMET(KOD),
      semestr CHAR(5) NOT NULL,
      znamka INT NOT NULL CHECK(ZNAMKA BETWEEN 1 AND 4),
      PRIMARY KEY (rc, kod, semestr)
);
```

totéž jako
znamka >= 1 and znamka <= 4

Do tabulek vložíme testovací data.

```
INSERT INTO Student VALUES
      ('800520/1234','Jan Fejfar', 'SWI'),
      ('123457/1234','Igor Holub', 'SWI'),
      ('123458/1234','Jaroslav Teska', 'DBS'),
      ('123459/1234','Jan Novak', 'DBS'),
      ('123460/1234','Jan Bok', 'DBS');


INSERT INTO Predmet(kod, nazev, garant) VALUES
      ('01DBS','Databaze','Blaha'),
      ('01SQL','Jazyk SQL','Blaha'),
      ('02MAA','Matematika','Janecek'),
      ('02SJ','Spanelsky jazyk','Garcia'),
      ('02LS','Logicke systemy','Farada');

INSERT INTO Zapis VALUES
      ('800520/1234','01DBS','2010Z',4),
      ('800520/1234','01SQL','2010Z',1),
      ('800520/1234','01DBS','2011Z',3),
      ('123459/1234','02MAA','2010L',3),
      ('123459/1234','02LS','2010L',2),
      ('123459/1234','01DBS','2010L',1),
      ('123460/1234','02MAA','2011L',3),
      ('123460/1234','02LS','2011L',2);
```

Příklad 1

Napište v SQL následující dotazy.

a. Seznam všech studentů.

Relační algebra:

```
STUDENT
```

SQL:


```
SELECT *
FROM Student;
```

b. Seznam všech specializací studentů.

Relační algebra:

```
STUDENT[specializace]
```

SQL: Výstupem budou všechny hodnoty sloupce specializace, SQL narozdíl od relační algebry neeliminuje duplicity. Abychom odstranili případné duplicitní řádky, musíme použít klauzuli DISTINCT. Vyzkoušejte s klauzulí DISTINCT i bez ní. Všimněte si, že projekce v relační algebře odpovídá výběru sloupců za klauzulí SELECT.

```
SELECT DISTINCT specializace
FROM Student;
```

analogie projekce v relační algebře

c. Seznam všech studentů zaměřených na SW inženýrství.

Relační algebra:

```
STUDENT(specializace=SWI)
```

SQL: Pro specifikaci logické podmínky použijeme klauzuli WHERE. Všimněte si, že selekce v relační algebře odpovídá podmínce za klauzulí WHERE.

```
SELECT *
FROM Student
WHERE specializace = 'SWI';
```

analogie selekce v relační algebře

d. Seznam všech předmětů, které negarantuje Bláha. Seřaďte podle abecedy podle názvu.

Relační algebra: V RMD nelze řadit, relace nemají definované pořadí prvků.

```
PŘEDMĚT(garant<>Bláha)
```

SQL:

```
SELECT *
FROM Predmet
WHERE garant <> 'Blaha'
ORDER BY nazev ASC;
```

řazení podle zadaného sloupce vzestupně, pro sestupné řazení by se zapsalo místo ASC slovo DESC

e. Jména všech všech studentů zaměřených na SW inženýrství.

Relační algebra:

```
STUDENT(specializace=SWI)[jméno]
```

SQL:

```
SELECT jmeno
FROM Student
WHERE specializace = 'SWI';
```

f. Kódy všech předmětů, které si zapsal student s RČ 800520/1234.

Relační algebra:

```
ZÁPIS(rč='800520/1234')[kód]
```

SQL:

```
SELECT kod
FROM Zapis
WHERE rc = '800520/1234';
```

g. Kódy všech předmětů, které si zapsal student Jan Novák.

Relační algebra:

```
( STUDENT * ZÁPIS ) (jméno='Jan Novák') [kód]
```

nebo

```
( STUDENT(jméno='Jan Novák') * ZÁPIS ) [kód]
```

SQL: V SQL se vždy provede nejprve spojení, teprve poté selekce a nakonec projekce.

```
SELECT kod
FROM Zapis NATURAL JOIN Student
WHERE jmeno = 'Jan Novak';
```

přirozené spojení tabulek, vazební sloupec se automaticky nastaví na „rc“

- jiné řešení (podle starší normy jazyka)

```
SELECT kod
FROM Zapis, Student
WHERE jmeno = 'Jan Novak' AND
      Zapis.rc = Student.rc;
```

kartézský součin tabulek

definice podmínky pro spojení je nutná, neboť jsme v klauzuli FROM použili kartézský součin

tato konstrukce by se v relační algebře dala zapsat jako
(ZÁPIS × STUDENT) (jméno='Jan Novák' ∧ ZÁPIS.rč = STUDENT.rč) [kód]

- ještě jiné řešení

```
SELECT kod
FROM Zapis JOIN Student USING(rc)
WHERE jmeno = 'Jan Novak';
```

obecnější zápis spojení:
spojení přes rovnost
v zadaném sloupci

- a ještě jiné řešení

```
SELECT kod
FROM Zapis JOIN Student ON(Zapis.rc = Student.rc)
WHERE jmeno = 'Jan Novak';
```

nejobecnější zápis spojení:
Θ-spojení se zadanou
vazební podmínkou

tato konstrukce by se v relační algebře dala zapsat jako
`(ZÁPIS [rč = rč] STUDENT) (jméno='Jan Novák') [kód]`

h. Garanti všech předmětů, z nichž byla udělena známka 3.

Relační algebra:

```
( ZÁPIS(známka=3) * PŘEDMĚT ) [garant]
```

SQL:

```
SELECT DISTINCT garant
FROM Zapis NATURAL JOIN Predmet
WHERE znamka = 3;
```

Ostatní zápisy SQL lze rovněž použít, viz předchozí úloha (ponecháváme jako samostatné cvičení). Vyzkoušejte si též variantu dotazu bez DISTINCT.

i. Seznam všech předmětů, které si někdo zapsal.

Relační algebra:

```
ZÁPIS *> PŘEDMĚT
```

nebo

```
( ZÁPIS * PŘEDMĚT ) [kód, název, syllabus, garant]
```

SQL: V SQL neexistuje přímo operace polospojení, sloupce je tedy nutné specifikovat projekcí za klauzulí SELECT

```
SELECT DISTINCT kod, nazev, syllabus, garant
FROM Zapis NATURAL JOIN Predmet;
```

nebo

```
SELECT DISTINCT Predmet.*
FROM Zapis NATURAL JOIN Predmet;
```

výběr všech sloupců zadané tabulky

j. Seznam předmětů, které si zatím nikdo nezapsal.

Relační algebra:

```
PŘEDMĚT – ( ZÁPIS <* PŘEDMĚT )
```

SQL: V SQL lze tento dotaz realizovat několika způsoby, ukážeme si tři z nich. První z nich je přímým přepisem relační algebry.

```
(SELECT DISTINCT * FROM Predmet)
EXCEPT
(SELECT Predmet.* FROM Zapis NATURAL JOIN Predmet);
```

množinový rozdíl, pozor na kompatibilitu operandů!

- jiné řešení s vnořeným SELECT blokem využívá logické podmínky „vyber takové předměty, jejichž kód není prvkem množiny kódů v tabulce Zápis“

```
SELECT DISTINCT *
FROM Predmet
WHERE kod NOT IN (SELECT kod FROM Zapis);
```

operátor IN testuje přítomnost prvku v zadané množině, pozor jen na kompatibilitu prvků množiny s hledaným prvkem!

- ještě jiné řešení s vnějším spojením, kde hledáme ty „nespojitelné“ řádky – obsahují prázdné hodnoty. Pro lepší pochopení funkce si nejprve prohlédněte výsledek dotazu

```
SELECT *
FROM Predmet NATURAL LEFT OUTER JOIN Zapis;
```

místo NATURAL by šlo specifikovat způsob spojení též pomocí JOIN ... USING() nebo pomocí JOIN ... ON()

Ve výsledku jsou dobře vidět buňky, které jsou prázdné. Španělský jazyk nebylo možné spojit s žádným řádkem tabulky Zápis, proto jsou všechny sloupce pocházející z tabulky Zápis (sloupce „rč“, „semestr“, „známka“) prázdné. Lze tedy nyní použít podmínku, pomocí které budeme zobrazovat pouze ty řádky, které obsahují třeba ve sloupci „rč“ NULL.

```
SELECT Predmet.*
FROM Predmet NATURAL LEFT OUTER JOIN Zapis
WHERE rc IS NULL;
```

test prázdné hodnoty, nelze testovat pomocí operátoru =

k. Jména všech studentů, kteří si zapsali nějaký předmět garantovaný Bláhou.

Relační algebra:

```
( STUDENT * ZÁPIS * PŘEDMĚT )(garant=Bláha)[jméno]
```

SQL:

```
SELECT DISTINCT jmeno
FROM Student NATURAL JOIN Zapis NATURAL JOIN Predmet
WHERE garant = 'Blaha';
```

l. Jména a specializace studentů, kteří si zapsali **pouze** předměty garantované Bláhou.

Relační algebra:

```
 ( ( STUDENT * ZÁPIS * PŘEDMĚT(garant= Bláha) ) -
   ( STUDENT * ZÁPIS * PŘEDMĚT(garant<>Bláha) ) ) [jméno, specializace]
```

SQL: V SQL je opět možné použít více řešení, viz bod j. Řešení s množinovým rozdílem:

```
(SELECT DISTINCT jmeno, specializace
 FROM Student NATURAL JOIN Zapis NATURAL JOIN Predmet
 WHERE garant = 'Blaha')
EXCEPT
(SELECT DISTINCT jmeno, specializace
 FROM Student NATURAL JOIN Zapis NATURAL JOIN Predmet
 WHERE garant <> 'Blaha');
```

- jiné řešení s vnořeným SELECT blokem:

```
SELECT DISTINCT jmeno, specializace
FROM Student NATURAL JOIN Zapis NATURAL JOIN Predmet
WHERE garant = 'Blaha' AND
   rc NOT IN (SELECT rc
              FROM Student NATURAL JOIN Zapis NATURAL JOIN Predmet
              WHERE garant <> 'Blaha');
```

m. Jména studentů, kteří si nic nezapsali.

Relační algebra:

```
( STUDENT – ( STUDENT <* ZÁPIS ) ) [jméno]
```

nebo

```
STUDENT[jméno] – ( STUDENT * ZÁPIS )[jméno]
```

SQL: V SQL je opět možné použít více řešení, viz bod j a l. Řešení s množinovým rozdílem:

```
(SELECT DISTINCT jmeno FROM Student)
EXCEPT
(SELECT jmeno FROM Student NATURAL JOIN Zapis);
```

- jiné řešení s vnořeným SELECT blokem:

```
SELECT DISTINCT jmeno
FROM Student
WHERE rc NOT IN (SELECT rc FROM Student NATURAL JOIN Zapis);
```

- ještě jiné řešení s vnějším spojením

```
SELECT DISTINCT jmeno
FROM Student NATURAL LEFT OUTER JOIN Zapis
WHERE semestr IS NULL;
```

n. Garanti předmětů, které si nezapsal student Jan Bok.

Relační algebra:

```
PŘEDMĚT[garant] – ( STUDENT(jméno=’Jan Bok’) * ZÁPIS * PŘEDMĚT )[garant]
```

SQL: Řešení s množinovým rozdílem:

```
(SELECT DISTINCT garant FROM Predmet)
EXCEPT
(SELECT garant
 FROM Student NATURAL JOIN Zapis NATURAL JOIN Predmet
 WHERE jmeno='Jan Bok');
```

- jiné řešení s vnořeným SELECT blokem:

```
SELECT DISTINCT garant
FROM Predmet
WHERE garant NOT IN
   (SELECT garant
    FROM Student NATURAL JOIN Zapis NATURAL JOIN Predmet
    WHERE jmeno='Jan Bok');
```

V dalších úkolech již nelze použít relační algebru, řešení zapisujeme pouze v SQL.

o. Vypiš celkový počet studentů.

SQL: Použijeme agregační funkci COUNT, která vrací počet neprázdných hodnot v zadaném sloupci. Je-li zapsána s hvězdičkou, vrací počet řádků dotazu.

```
SELECT COUNT(*)
FROM Student;
```

p. Vypiš průměrnou známku předmětu Databáze.

SQL: Použijeme agregační funkci AVG.

```
SELECT AVG(znamka)
FROM Zapis NATURAL JOIN Predmet
WHERE nazev = 'Databaze';
```

q. Vypiš nejhorší známku Jana Nováka.

SQL: Použijeme agregační funkci MAX.

```
SELECT MAX(znamka)
FROM Zapis NATURAL JOIN Student
WHERE jmeno = 'Jan Novak';
```

r. Vypiš názvy předmětů a pro každý z nich nejlepší zapsanou známku.

SQL: Použijeme agregační funkci MIN. Aby se agregační funkce neaplikovala na celou tabulku, ale na skupinky řádků se shodným názvem předmětu, musíme tabulku seskupit klauzulí GROUP BY. Zopakujme, že při použití seskupování smí být za klauzulí SELECT uveden pouze sloupec, podle kterého seskupujeme, na ostatní sloupce musí být aplikována nějaká agregační funkce.

```
SELECT nazev, MIN(znamka)
FROM Zapis NATURAL JOIN Predmet
GROUP BY nazev;
```

s. Vypiš názvy předmětů a pro každý z nich nejlepší zapsanou známku. Chceme zobrazit pouze známky předmětů vypsaných v letním semestru (kód semestru obsahuje znak L)

SQL: Použijeme agregační funkci MIN. Ještě předtím, než tabulku seskupíme klauzulí GROUP BY, vybereme jen předměty vypsané v letním semestru. Upozorněme, že nelze použít klauzuli HAVING, tj. výběr seskupených řádků, protože v klauzuli HAVING smíme použít buď pouze sloupce, podle nichž se seskupuje, nebo sloupce obalené nějakou agregační funkcí – vyzkoušejte si.

```
SELECT nazev, MIN(znamka)
FROM Zapis NATURAL JOIN Predmet
WHERE semestr LIKE '%L'
GROUP BY nazev;
```

t. Vypiš názvy předmětů a pro každý z nich nejlepší zapsanou známku. Chceme zobrazit pouze předměty, v nichž nebyla nikdy zapsána jednička. Seřadit podle nejlepší známky sestupně.

SQL: Použijeme agregační funkci MIN. Po seskupení tabulky klauzulí GROUP BY, vybereme jen ty předměty, mající minimální známku > 1. Použijeme klauzuli HAVING, tj. výběr seskupených řádků. Sloupec s nejlepší známkou pojmenujeme „nejlepsi“.

```
SELECT nazev, MIN(znamka) AS nejlepsi
FROM Zapis NATURAL JOIN Predmet
GROUP BY nazev
HAVING MIN(znamka) > 1
ORDER BY nejlepsi DESC;
```

v PostgreSQL nejde z nějakého důvodu za HAVING použít pojmenování sloupce

u. Najdi rč studentů, kteří si zapsali nějaký předmět vícekrát než jednou.

SQL: Tabulku Zápis seskupíme podle rč a kódu předmětu a v každé z těchto skupinek spočítáme počet řádků. Ze vzniklých superřádků vybereme jen takové, které mají tento počet větší než 1.

```
SELECT rc
FROM Zapis
GROUP BY rc, kod
HAVING COUNT(*) > 1;
```

v. Najdi jména studentů, kteří si zapsali nějaký předmět vícekrát než jednou.

SQL: Spojíme nejprve tabulky Zápis a Student. Vzniklou tabulku potřebujeme seskupit podle rč a kódu předmětu, viz předchozí úkol. Jenže pozor, za klauzulí SELECT potřebujeme uvést sloupec „jméno“, podle kterého neseskupujeme, a to nám databáze nedovolí (vyzkoušejte si to). Musíme tedy „jméno“ přidat do seznamu sloupců, podle nichž seskupujeme. Na fungování dotazu to nebude mít vliv, protože víme, že každému rodnému číslu odpovídá právě jedno jméno, tím pádem se skupinky nijak nezmění. Proč nesekupujeme tabulku rovnou podle dvojice „jméno“, „kód“, abychom se vyhnuli této konstrukci? Je to proto, že jedno jméno může mít více studentů (jméno není obecně unikátní), čímž by v seskupování mohlo dojít ke sloučení více různých osob do jedné skupinky.

```
SELECT jmeno
FROM Zapis NATURAL JOIN Student
GROUP BY rc, kod, jmeno
HAVING COUNT(*) > 1;
```

Alternativně lze použít vnořený SELECT v klauzuli FROM, kde spojíme výstup dotazu podle bodu u s tabulkou Student, abychom získali jméno studenta.

```
SELECT jmeno
FROM Student NATURAL JOIN
   (SELECT rc
    FROM Zapis
    GROUP BY rc, kod
    HAVING COUNT(*) > 1) AS S;
```

používáme-li v klauzuli FROM vnořený SELECT, musíme jej v PostgreSQL pojmenovat

Úlohy k procvičení

1. Uvažujme seznam kin a seznam filmů (viz též předchozí kapitoly).
   kino má atributy: název, adresa, počet míst, Dolby ano/ne
   film má atributy: název, rok vzniku, země vzniku, dabing ano/ne

   Uvažujte několik variant v kombinacích, řešte bez a s použitím prázdných hodnot (NULL):
   A. v každém kině se může dávat víc filmů, každý film se může dávat pouze v jednom kině.
   B. v každém kině se může dávat víc filmů a každý film se může dávat ve více kinech.
   1. chceme evidovat jen filmy, které jsou někde dávány
   2. chceme evidovat všechny filmy

   Zapište v SQL a vyzkoušejte následující dotazy:
   - vypiš názvy kin, která mají kapacitu větší než 100 míst
   - vypiš názvy filmů, které byly natočeny v USA
   - vypiš názvy kin, která mají dolby a nehrají zrovna žádný film
   - vypiš filmy, které se hrají v kině Dukla
   - vypiš názvy a adresy kin, kde se dává film Titanic
   - vypiš názvy a adresy kin, kde se dává nějaký film z Japonska

---

2. Vyzkoušejte nějaké dotazy nad Vašimi tabulkami z předchozí kapitoly.

---

3. Uvažujme lékařskou praxi skupiny lékařů podporovanou relační databází se třemi tabulkami definovanými schématy

   LÉKAŘ(Č_LICENCE, JMÉNO_L, SPECIALIZACE)
   PACIENT(Č_PACIENTA, JMÉNO_P, ADRESA, TELEFON, D_NAROZENÍ)
   NÁVŠTĚVA(Č_LICENCE, Č_PACIENTA, TYP, DATUM, DIAGNÓZA, CENA)

   kde TYP znamená typ návštěvy (domů na zavolání, do ordinace, do nemocnice apod.). Vytvořte tabulky a napište v SQL tyto požadavky:

   - Seznam všech registrovaných pacientů.
   - Seznam všech specializací lékařů.
   - Seznam všech orthopédů.
   - Jména všech orthopédů.
   - Seznam všech pacientů narozených před r. 1920.

- Licence všech lékařů, které navštívila Marie Nová.
- Jména všech lékařů, kteří byli na návštěvě domů na zavolání.
- Seznam všech navštívených pacientů.
- Seznam všech nenavštívených pacientů.
- Jména a adresy všech pacientů, kteří byli vyšetřováni Dr. Lomem 23.4.1992.
- Jména a adresy všech pacientů, u kterých byla určena diagnóza HlV-positivní.
- Nalezni jména a specializace všech lékařů, kteří určili diagnózu vřed na dvanácterníku.
- Jména a adresy pacientů, kteří byli vyšetřováni pouze Dr. Čermákem.
- Čísla licencí a jména lékařů, kteří nemají žádný záznam o léčbě.
- Která specializace neměla dosud nic na práci.

Příklad 3. převzat z [1].

9 Jazyk SQL – další konstrukce, příklady

Cíl kapitoly

V této kapitole zopakujeme a rozšíříme některé znalosti o jazyce SQL. Cílem je zopakovat si

- funkci spojení v relační algebře a v SQL
- sémantiku příkazu SELECT

Dále je cílem naučit se:

- další konstrukty jazyka SQL
  - CASE
  - COALESCE
- detaily příkazu ALTER TABLE

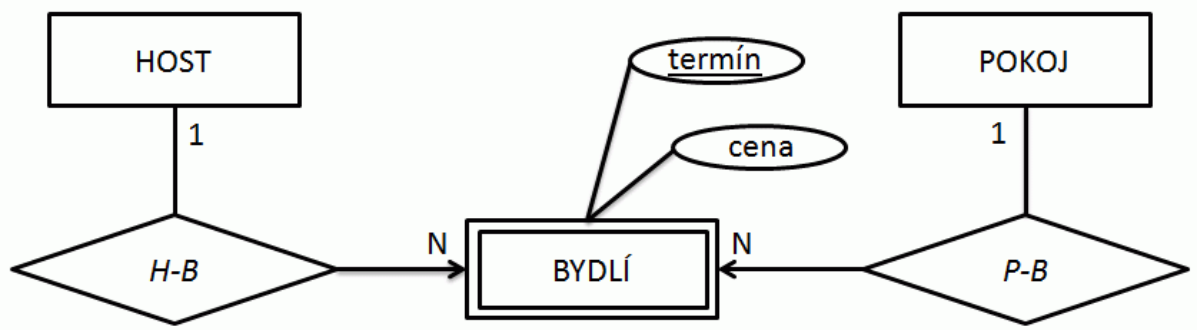
Klíčové pojmy

Spojení, příkaz SELECT, konstrukt CASE, COALESCE, příkaz ALTER TABLE.

Ilustrativní příklady na spojení

Evidence hostů a jejich ubytování na pokojích

Chceme evidovat historii ubytování => je nutný vztah M:N mezi hostem a pokojem.



Převod do tabulek (podtrženě primární klíč)

- HOST(jméno, příjmení, RČ, bydliště)
- POKOJ(číslo, kapacita, WC)
- BYDLÍ(RČ, číslo, termín, cena)

Příklad na dotaz

dotaz: vypiš příjmení všech hostů, kteří bydleli na pokoji č. 1

- princip řešení:

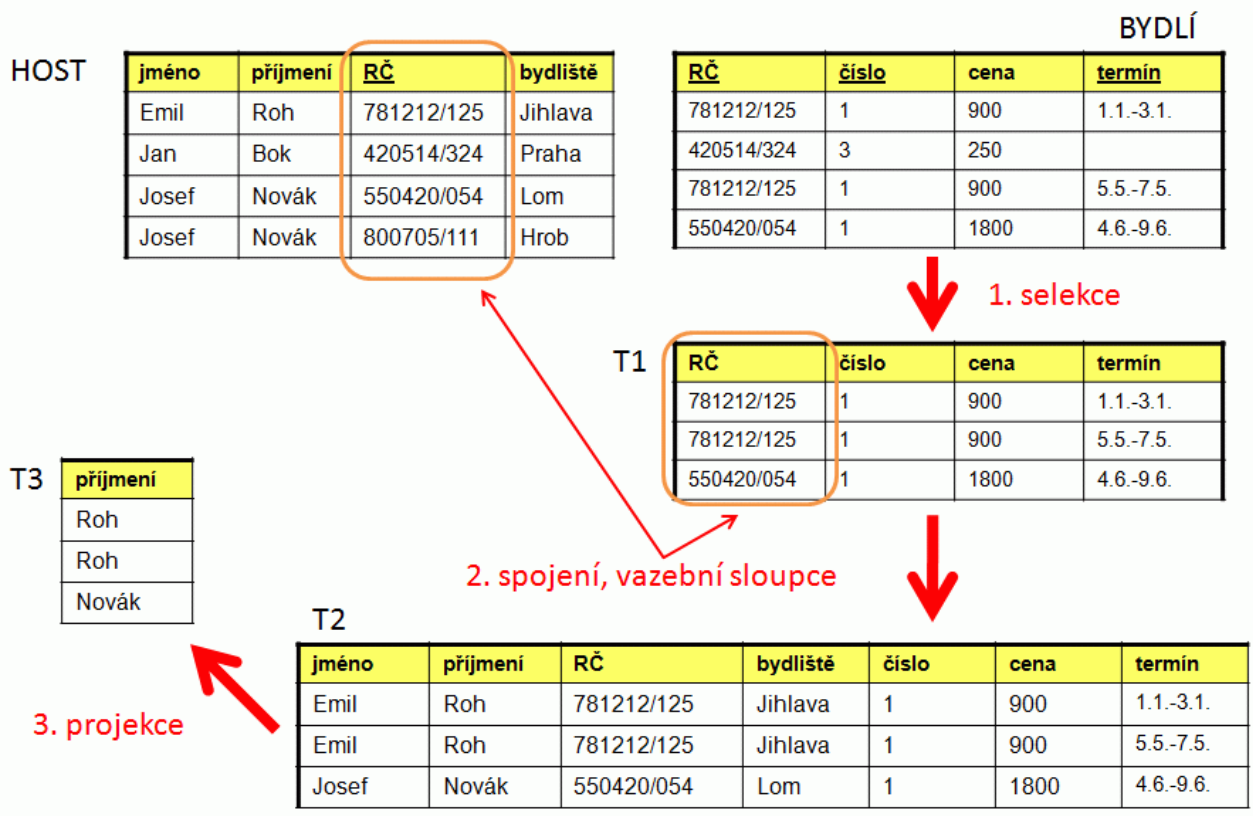
1. selekce tabulky BYDLÍ na řádky s číslem pokoje 1
   T1(RČ, číslo, termín, cena) s méně řádky než je v tabulce BYDLÍ
2. spojení tabulky T1 s tabulkou HOST přes vazební sloupec „RČ“
   T2(jméno, příjmení, RČ, bydliště, číslo, termín, cena)
3. projekce tabulky T2 na sloupec „příjmení“
   T3(příjmení)

- řešení v relační algebře:

(BYDLÍ(číslo=1) * HOST) [příjmení]

nebo

(BYDLÍ * HOST) (číslo=1) [příjmení]



- řešení v SQL:

```
SELECT příjmení
FROM    BYDLÍ   NATURAL JOIN   HOST
WHERE    číslo=1;
```

Další příklad na dotaz

dotaz: vypiš příjmení všech hostů a u těch, kteří někde bydlí nebo bydleli, vypiš číslo pokoje

- princip řešení:

1. levé vnější přirozené spojení tabulky HOST s tabulkou BYDLÍ přes vazební sloupec „RČ“
   T1(jméno, příjmení, RČ, bydliště, číslo, termín, cena)
2. projekce tabulky T1 na sloupce „příjmení“ a „číslo“
   T2(příjmení, číslo)

HOST

jméno	příjmení	RČ	bydliště
Emil	Roh	781212/125	Jihlava
Jan	Bok	420514/324	Praha
Josef	Novák	550420/054	Lom
Josef	Novák	800705/111	Hrob

BYDLÍ

RČ	číslo	cena	termín
781212/125	1	900	1.1.-3.1.
420514/324	3	250	1.1.-3.1.
781212/125	1	900	5.5.-7.5.
550420/054	1	1800	4.6.-9.6.

1. vnější spojení, vazební sloupce

T1

jméno	příjmení	RČ	bydliště	číslo	cena	termín
Emil	Roh	781212/125	Jihlava	1	900	1.1.-3.1.
Emil	Roh	781212/125	Jihlava	1	900	5.5.-7.5.
Jan	Bok	420514/324	Praha	3	250	1.1.-3.1.
Josef	Novák	550420/054	Lom	1	1800	4.6.-9.6.
Josef	Novák	800705/111	Hrob	NULL	NULL	NULL

T2

příjmení	číslo
Roh	1
Roh	1
Bok	3
Novák	1
Novák	NULL

2. projekce

- řešení v relační algebře:
  není tak jednoduché, neboť vnější spojení není součástí relační algebry

- řešení v SQL:
  neubytovaní hosté budou mít místo čísla pokoje NULL

```
SELECT příjmení, číslo
FROM    BYDLÍ   RIGHT   NATURAL   JOIN   HOST;
```

Sémantika dotazu

Mějme dotaz

```
SELECT A1, agreg_f(A2) AS ‘Name‘
FROM R1, R2
WHERE φ
GROUP BY A1
HAVING ψ
ORDER BY Name
```

Sémantika, jak to pracuje (to neznamená, že takto to dělá SŘBD – před provedením dotazu předchází optimalizace):

- vytvoří se kartézský součin R1 a R2
- vyhodnotí se podmínka φ za WHERE
- seskupí se podle A1 (vznik „superřádků“)
- ze „superřádků“ se vyberou jen ty, splňující podmínku ψ
- výsledná tabulka se seřadí podle sloupce Name vzestupně
- zobrazíme sloupec A1 a agregovanou hodnotu sloupce A2 pojmenovanou jako 'Name'

V uvedených krocích lze pochopitelně použít další povolené konstrukty:

- za FROM: definici spojení, vnořené SELECT bloky...
- za WHERE a HAVING: vnořené SELECT bloky...
- za SELECT: vypočtené hodnoty, podmínku (viz dále), funkce...

Konstrukt CASE

CASE zajišťuje běžné podmíněné větvení, používá se např. pro transformaci typu hodnot apod. Existují dvě syntaktické varianty, méně obecná:

```
SELECT a,
        CASE a WHEN 1 THEN ’one’
                WHEN 2 THEN ’two’
                ELSE ’other’
        END
FROM test;
```

a obecnější verze s explicitní podmínkou:

```
SELECT a,
        CASE    WHEN a=1 THEN ’one’
                WHEN a=2 THEN ’two’
                ELSE ’other’
        END
FROM test;
```

Použití CASE pro zajištění IO, např. chceme zajistit, aby každou silnici první třídy vlastnil stát:

```
CREATE TABLE Silnice (
        číslo_s  INT             PRIMARY KEY,
        třída    INT             NOT NULL CHECK (třída>0)
        vlastník VARCHAR(50)     NOT NULL,
        CONSTRAINT chk_vlastník CHECK (
                CASE WHEN třída=1 THEN vlastník='stát' END )
);
```

Konstrukt COALESCE

Funkce COALESCE vybere ze seznamu v závorkách první hodnotu, která není NULL. Syntaxe:

```
COALESCE(A, B, C, D);
```

Používá se pro řešení případů, kdy očekáváme NULL, například u agregačních funkcí:

```
COALESCE(Vypujcky.Cena, "Cena neni urcena");
```

Výše uvedený příklad na pokoje a hosty: vypiš příjmení všech hostů a u těch, kteří někde bydlí nebo bydleli, vypiš číslo pokoje, u neubytovaných vypiš nulu:

```
SELECT příjmení, COALESCE(číslo, 0)
FROM    BYDLÍ   RIGHT   NATURAL   JOIN   HOST;
```

Příklad: chceme vypsat čísla silnic, počet mostů na každé silnici (vč. nul) a nejmenší nosnost mostu na každé silnici:

```
SELECT číslo_s,          COUNT(číslo_m),   MIN(COALESCE(nosnost, 100000000))
FROM  Silnice LEFT NATURAL   JOIN   (Leží_na   NATURAL   JOIN   Most)
GROUP BY číslo_s;
```

Detaily ALTER TABLE

Přidat, odebrat a přejmenovat sloupec již umíme. Uvažujme nyní tabulku

```
CREATE TABLE Výrobek (
        Id              INTEGER             PRIMARY KEY,
        Název           VARCHAR(128)        UNIQUE,
        Cena            DECIMAL(6,2)        NOT NULL,
        JeNaSkladě      BOOL                DEFAULT TRUE,
        CONSTRAINT chk CHECK ( Id > 0 )
);
```

Změna IO sloupce
Lze snadno měnit NOT NULL a DEFAULT:

```
ALTER TABLE Výrobek
        ALTER COLUMN Cena DROP NOT NULL;    -- odebrání IO

ALTER TABLE Výrobek
        ALTER COLUMN Cena SET DEFAULT 0.0;

ALTER TABLE Výrobek
        ALTER COLUMN JeNaSkladě DROP DEFAULT;
```

Primární a cizí klíče lze měnit hůř: musíme se v popisu tabulky podívat, jak stroj tato IO pojmenoval a pak pracovat s jejich jmény stejně, jako u tabulkových IO.

Změna IO tabulky
Tabulková IO nelze přímo měnit, je třeba staré odebrat a přidat nové.
Přidávat lze snadno

```
ALTER TABLE Výrobek
        ADD CONSTRAINT chk2 CHECK (Cena > 0);
```

nebo nepojmenované

```
ALTER TABLE Výrobek
        ADD CHECK (Cena > 0);
```

Pokud je chceme odebírat, musíme znát jejich název, je proto nejlepší si tabulková IO pojmenovávat rovnou při založení.

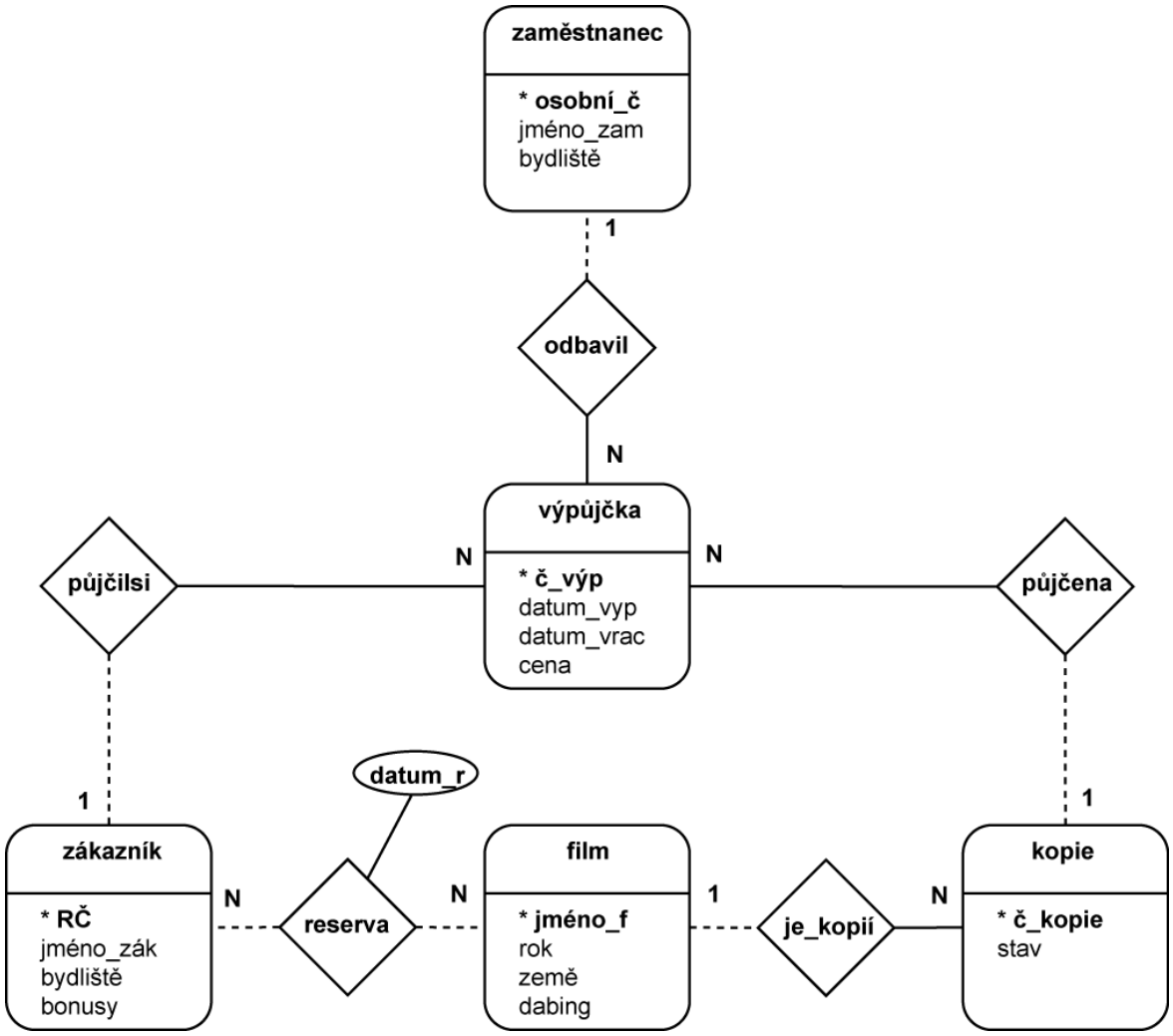
```
ALTER TABLE Výrobek
        DROP CONSTRAINT chk2;
ALTER TABLE Výrobek
        ADD CONSTRAINT chk2 CHECK (Cena > 0 AND Cena < 1000);
```

Přejmenování tabulky

```
ALTER TABLE Výrobek
        RENAME TO Zboží;
```

Úlohy k procvičení

Mějme ER model



Vytvořte pro tento model odpovídající tabulky v SQL a zapište a vyzkoušejte následující dotazy:

1. vypiš jména zákazníků s jejich adresami (pozn.: co duplicity?)
2. vypiš jména zákazníků s jejich adresami a seřaď vzestupně podle jména
3. vypiš výpůjčky, které byly vráceny do 31.12.2008
4. najdi země, na jejichž některé filmy jsou reservace
5. najdi dvojice zákazníků, kteří mají stejnou adresu
6. vypiš celkovou cenu všech výpůjček Arthura Robinsona v librách (kurs je 1:30)
7. zjisti celkový počet registrovaných zákazníků
8. kolik je reservovaných filmů

9. kolik je výpůjček levnějších než 300Kč
10. jaký je celkový počet výpůjček Jana Nováka
11. najdi pro každého zákazníka (r.č.) počet vypůjčených kopií
12. najdi r.č. zákazníků, kteří mají půjčeny víc než dvě kopie
13. najdi jména zákazníků, kteří mají půjčeny víc než dvě kopie
14. najděte zákazníky, kteří si půjčili film Údolí včel
15. najdi jména zákazníků, kteří mají reservován nějaký film (využijte EXISTS)
16. najdi jména zákazníků, kteří nemají reservován žádný film (využijte NOT EXISTS)
17. najdi jména zákazníků, kteří nemají reservován žádný film (využijte EXCEPT)
18. najdi jména zákazníků, kteří nemají reservován žádný český film
19. najdi jména zákazníků, kteří mají reservován nebo půjčen japonský film (využijte UNION)

10 Normální formy relací – nástin

Cíl kapitoly

V této kapitole nastíníme úvod do normálních forem relací a normalizace. Tato problematika se detailně probírá v předmětu Databázové systémy 2, kapitola proto neobsahuje příklady a cvičení. Cílem je seznámit se s

- motivací pro normální formy relací
- základy funkčních závislostí
- základními normálními formami
- způsobem testování třetí normální formy
- metodikou, jak normalizovat relace

Klíčové pojmy

Normální formy relací, třetí normální forma, funkční závislost, normalizace.

Motivace

Normalizace relací znamená úpravu jejich struktury tak, aby relační schema dobře reprezentovalo data, aby byla omezena redundance a přitom se neztratily vazby mezi daty. Například mějme relaci

PROGRAM(název_k, jméno_f, adresa, datum)

Problémy této relace:

- některá informace bude zbytečně opakována víckrát
- adresa kina bude uložena tolikrát, kolik filmů je v kině na programu
- změní-li se jméno ulice, musíme přepisovat mnoho
- nehraje-li kino nic, ztrácíme jeho adresu

Řešení: dekompozice na dvě schemata relací

KINO(název_k, adresa)
PROGRAM(název_k, jméno_f, datum)

- všechny problémy zmizely

Obecně:

- redundance – ukládání stejné informace víckrát na různých místech (zvyšuje prostorové nároky)
- mohou nastat aktualizační anomálie (insert/delete/update)
    - při vložení dat příslušejících jedné entitě je potřeba zároveň vložit data do jiné entity
    - při vymazání dat příslušejících jedné entitě je potřeba vymazat data patřící jiné entitě
    - pokud se změní jedna kopie redundantních dat, je třeba změnit i ostatní kopie, jinak se databáze stane nekonzistentní
- řešení: normalizace schematu

Co je důvodem problémů v příkladu? Hodnoty některých atributů závisejí na hodnotách jiných atributů:

- ke každému kinu přísluší právě jedna adresa
- pro každé kino a film existuje nejvýše jedno datum, kdy se film dává

V podstatě jde o funkci: přiřazení hodnoty jiné hodnotě. Lze se na to též dívat jako na speciální případ IO. Definujeme tzv. funkční závislosti:

název_k $\rightarrow$ adresa

{název_k, jméno_f} $\rightarrow$ datum

Funkční závislosti

Funkční závislost (FZ) je závislost mezi dvěma množinami atributů v rámci **jednoho schematu** relace. FZ je závislost mezi daty v jedné relaci, nikoli mezi entitami a daty nebo mezi dvěma entitami! FZ je integritní omezení – vymezuje množinu přípustných relací.

Funkční závislost vyplývá z obecné (konceptuální) závislosti dat, nikoli z jejich konkrétní podoby. Např. máme tabulku studijních výsledků

student	**předmět**	**známka**
Novák	Teorie obvodů	3
Novák	Úvod do algebry	3
Haničinec	Teorie obvodů	1
Haničinec	Matematika 1	1

Podle těchto dat to vypadá, že

student $\rightarrow$známka

to ale ve skutečnosti určitě obecně neplatí, je to pouze náhoda.

Normální formy

Aby schema relace splňovalo jisté požadavky (např. na malou redundanci, odstranění aktualizačních anomálnií atd.), zavádíme tzv. normální formy:

- první
- druhá
- třetí
- Boyce-Coddova
- (čtvrtá, pátá)

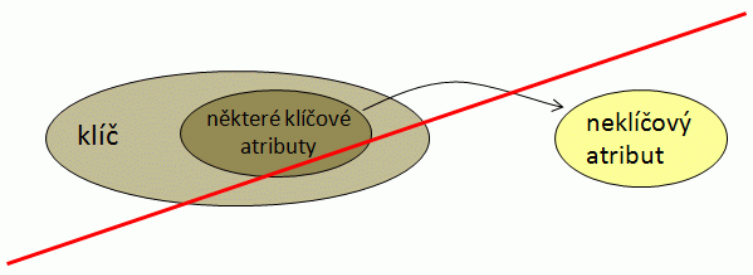
První normální forma (1NF)

- každý atribut schématu je elementární a nestrukturovaný, tj. databáze je plochá, tabulka je skutečně dvojrozměrný objekt

  OSOBA(jméno, rč, č_op) je v 1NF
  OSOBA(jméno, rč, adresa(město, ulice, čp)) není v 1NF

Druhá normální forma (2NF)

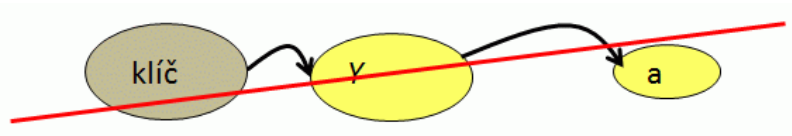
Neexistují závislosti atributů na podmnožině žádného klíče:



Např. PROGRAM(název_k, jméno_f, adresa, datum) není v 2NF, neboť platí, že název_k $\rightarrow$ adresa.

Třetí normální forma (3NF)

Nejdůležitější: žádný neklíčový atribut není tranzitivně závislý na žádném klíči:



Podle definice se třetí normální forma špatně testuje. Existuje ale sada testovacích podmínek, jimiž 3NF otestujeme snadno:

Relace je ve 3NF, platí-li pro každou funkční závislost X $\rightarrow$ *a* (kde X $\subseteq$ A, *a* $\in$ A) aspoň jedna z podmínek

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

Příklad:
Mějme relaci zobrazenou jako tabulku

Firma	**Kraj**	**Sídlo**
Agrostroj	Vysočina	Pelhřimov
KESAT	Vysočina	Jihlava
Chovanec Group	Pardubický	Pardubice

Víme, že platí následující FZ:

Firma $\rightarrow$ *všechno*

Sídlo $\rightarrow$ Kraj

Relace je ve 2NF, ale není ve 3NF (kvůli tranzitivní závislosti Kraje na klíči přes Sídlo). Důsledek: redundance hodnot Kraje.

Řešení: dekomponujeme relaci na dvě relace:

Firma	**Sídlo**
Agrostroj	Pelhřimov
KESAT	Jihlava
Chovanec Group	Pardubice

Sídlo	**Kraj**
Pelhřimov	Vysočina
Jihlava	Vysočina
Pardubice	Pardubický

Obě schemata jsou nyní ve 3NF.

Jak normalizovat
Mějme schema relace R(A,B,C), kde atribut A je klíčem. To, že je A klíčem znamená mj. že:

- A $\rightarrow$ B
- A $\rightarrow$ C

Klíč totiž jednoznačně určuje každou n-tici relace, každý atribut relace je na klíči přímo závislý. Možná namítnete, že relace R má tři atributy a výše uvedené závislosti jsou jen dvě; ta třetí je ovšem triviální, proto jsem ji neuvedl: A $\rightarrow$ A.

Přijdeme-li při analýze na to, že na relaci R platí také B $\rightarrow$ C, pak tato relace není ve 3NF: existuje zde tranzitivní závislost atributu C na klíči přes atribut B.

Řešení: dekomponujeme původní relaci na dvě a to tak, že problematickou funkční závislost umístíme do samostatné relace, atribut pravé strany z původní relace odtrhneme:

R1(B,C) -- nová relace

R2(A,B) -- původní relace bez pravé strany problematické funkční závislosti

O nic jsme přitom nepřišli, neboť pokud provedeme nyní spojení (R2 * R1), dostaneme původní relaci R.

Výhody normalizace:

- vede k lepšímu uložení dat v databázi
- omezuje redundanci
- snižuje riziko nekonzistence

Nevýhody normalizace:

- zvyšuje četnost operace spojení, která je paměťově i časově náročná, proto někdy raději volíme nenormalizované relace kvůli rychlejší odezvě
- může značně znepřehlednit databázi
- někdy normalizací sice vzniknou relace ve 3NF, ale dojde ke ztrátě některých závislostí mezi daty, je proto třeba obezřetnosti

Literatura

[1] Pokorný, J. – Halaška, I.: *Databázové systémy*. Skripta FEL ČVUT, Vydavatelství ČVUT, Praha, 1999.

[2] Pokorný, J. – Halaška, I.: *Databázové systémy*. Skripta FEL ČVUT, Vydavatelství ČVUT, Praha, 2004.

[3] Conolly, T. – Begg, C. – Holowczak, R.: *Mistrovství – databáze: Profesionální průvodce tvorbou efektivních databází*. 1. vydání, Computer press, Brno, 2009.

[4] *PostgreSQL: The world's most advanced open source database* [online]. 2011 [cit. 2011-10-07]. Dostupné z WWW: <http://www.postgresql.org/>.