Vysoká škola báňská – Technická univerzita Ostrava

TEORIE ZPRACOVÁNÍ DAT

Učební text

Jana Šarmanová

Ostrava 2007

Recenzoval: RNDr. Daniela Ďuráková, Ph.D.

Název: Teorie zpracování dat
Autor: Jana Šarmanová
Vydání: druhé přepracované, 2007
Počet stran: 169

Studijní materiály pro studijní program Informační a komunikační technologie
Jazyková korektura nebyla provedena.

Určeno pro projekt:
Operační program Rozvoj lidských zdrojů
Název: E-learningové prvky pro podporu výuky odborných a technických předmětů
Číslo: CZ.O4.01.3/3.2.15.2/0326
Realizace: VŠB – Technická univerzita Ostrava
Projekt je spolufinancován z prostředků ESF a státního rozpočtu ČR

ISBN 978-80-248-1498-8

POKYNY KE STUDIU

Teorie zpracování dat

Pro předmět **Teorie zpracování dat** studijního programu *Informační a komunikační technologie* jste obdrželi na CD studijní balík obsahující

- integrované skriptum pro distanční část studia
- příručku Průvodce studiem s podrobnějšími informacemi o předmětech:
  - harmonogramem předmětu s termínovanými úkoly
  - požadavky k zápočtu a ke zkoušce
  - kontakt na tutora

Prerekvizity

Pro studium tohoto předmětu se předpokládají následující znalosti:

- znalost alespoň jednoho klasického programovacího jazyka
- znalost základů matematické logiky a základních pojmů teorie množin

Cíl předmětu

Předmět Teorie zpracování dat je úvodem do studia databázových a informačních systémů, určený pro informatiky, případně budoucí specialisty – tvůrce databázových a informačních systémů. V této části se student seznámí s důležitými základními pojmy databázové technologie, informativně s metodami fyzické organizace databází a se síťovým datovým modelem. Podrobně pak je probírán základ datového modelování – relační datový model. Student se naučí správně navrhovat strukturu databáze a vyhledávat i odvozovat informace z ní.

Na TZD navazují další databázové předměty, předměty o informačních systémech, o datových skladech a dolování znalostí z dat.

Pro koho je předmět určen

Předmět je zařazen do bakalářského studia všech oborů studijního programu *Informační a komunikační technologie*, ale může jej studovat i zájemce z kteréhokoliv jiného oboru, pokud splňuje požadované prerekvizity.

Při studiu každé kapitoly doporučujeme následující postup:

Skriptum se dělí na části, kapitoly, které odpovídají logickému dělení studované látky, ale nejsou stejně obsáhlé. Předpokládaná doba ke studiu kapitoly se může výrazně lišit, proto jsou velké kapitoly děleny dále na číslované podkapitoly a těm odpovídá níže popsaná struktura.

Čas ke studiu: 1 hodina

Na úvod kapitoly je uveden **čas** potřebný k prostudování látky. Čas je orientační a může vám sloužit jako hrubé vodítko pro rozvržení studia celého předmětu či kapitoly. Někomu se čas může zdát příliš dlouhý, někomu naopak. Jsou studenti, kteří se s problematikou databází ještě nikdy nesetkali a naopak takoví, kteří již v tomto oboru mají bohaté zkušenosti.

Cíl Po prostudování tohoto odstavce budete umět

- popsat …
- uvést příklady z praxe, kdy …

Ihned potom jsou uvedeny cíle, kterých máte dosáhnout po prostudování této kapitoly – konkrétní dovednosti, znalosti.

Výklad

Následuje vlastní výklad studované látky, zavedení nových pojmů, jejich vysvětlení, vše doprovázeno řešenými příklady z praxe. Při výkladu je základní text doplňován tučně vyznačenými důležitými a novými pojmy. Kurzivou jsou psány příklady z praxe, buď v rámci textu nebo jako ucelené řešené příklady samostatně označené.

Shrnutí pojmů

Na závěr kapitoly jsou zopakovány hlavní pojmy, které si v ní máte osvojit. Pokud některému z nich ještě nerozumíte, vraťte se k nim ještě jednou.

Otázky

Pro ověření, že jste dobře a úplně látku kapitoly zvládli, máte k dispozici několik teoretických otázek.

Úlohy k řešení

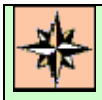
Protože většina teoretických pojmů tohoto předmětu má bezprostřední význam a využití v praxi softwarového inženýra, jsou Vám nakonec předkládány i praktické úlohy k řešení. V nich je hlavní význam kurzu a schopnost aplikovat čerstvě nabyté znalosti při řešení reálných situací hlavním cílem kurzu.

Klíč k řešení

Výsledky zadaných příkladů – stejně jako teoretických otázek výše jsou uvedeny v závěru učebnice v klíči k řešení. Používejte je až po vlastním vyřešení úloh, jen tak si sebekontrolou ověříte, že jste obsah kapitoly skutečně úplně zvládli.

Příprava na tutoriál

Souhrn znalostí nebo vypracovaných úloh, se kterými máte přijít na tutoriál. Mohou to být náměty k diskusím, otázky k promýšlení. Studující se tak mohou připravit na společná setkání a výsledkem je omezení okamžitých improvizací. .



Průvodce studiem

V tomto rámečku budou občas napsány pokyny o tom, co je důležité umět, co stačí jen přečíst informativně apod.

Úspěšné a příjemné studium s touto učebnicí Vám přeje autorka kurzu

Jana Šarmanová.

OBSAH

1. ZPRACOVÁNÍ DAT

Čas ke studiu kapitoly: 2 hodiny

1.1. Úlohy zpracování dat

Cíl Po prostudování tohoto odstavce budete umět

- popsat problém evidence a zpracování velkého množství dat
- definovat základní pojmy zpracování dat
- uvést příklady z praxe, kdy je třeba evidovat a zpracovávat data

Výklad

❑ Co je úkolem zpracování dat

V praktickém životě je často zapotřebí **evidovat údaje o nějaké skutečnosti**. Například o skupině lidí (zaměstnanců, studentů, členů sportovního oddílu apod.), o zvířatech nebo rostlinách (evidence zoologické nebo botanické zahrady apod.), o množině věcí (knihy ve veřejné knihovně, inventář firmy, materiálu na skladě apod.) nebo jevů (počasí, provedených lékařských výkonech apod.).

Vést evidenci znamená udržovat o takových souborech objektů přehled, údaje mít vhodně uspořádány, aby se v nich údaje dobře vyhledávaly, v případě potřeby opravovaly a doplňovaly, někdy počítaly údaje nové, z původních odvozené, vytvářely různé výsledné přehledné tabulky apod.

Objekty (lidi, zvířata, věci, jevy) obvykle popisujeme pomocí jejich **vlastností** (zaměstnanec firmy má jméno, adresu, funkci, plat; kniha v knihovně má název, autora, rok vydání, cenu, …). Při evidencích se pak předem rozhodneme, které vlastnosti potřebujeme sledovat. Vybrané vlastnosti budeme nazývat **atributy**, pojmenujeme si je a zvolíme nějakou formu evidence: na papír či do sešitu nalinkujeme tabulku, sloupce popíšeme názvy vybraných evidovaných vlastností, celou tabulku pojmenujeme typem evidovaných objektů.

Příklad 1.1.

Evidence dat o zaměstnancích v tabulce. Zaměstnanci (objekty) jsou zapisováni v pořadí, jak byli do firmy přijati.Potřebujeme evidovat jejich atributy jméno, adresu, funkci, plat. Pojmenujeme tabulku Zaměstnanec a její strukturu (= seznam evidovaných vlastností, atributů) zapíšeme:

Zaměstnanec (jméno, adresa, funkce, plat)

Tabulka vypadá takto:

Zaměstnanec

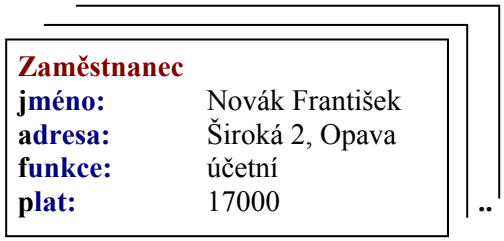
jméno	**adresa**	**funkce**	**plat**
Žižka Kamil		svářeč	12000
Bednářová Petra		uklízečka	8000
…			
Novák František	Široká 2, Opava	účetní	17000

Při zvyšujícím se počtu evidovaných objektů se brzy objeví nevýhody této tabulkové formy. Má-li tabulka již desítky řádků, je vyhledávání zdlouhavé (hledat se musí postupně shora dolů). Při změnách hodnot údajů (slečna se provdá a změní jméno i adresu, úspěšný účetní dostane vyšší plat) se musí přepisovat údaje v políčkách tabulky, nebo celý řádek škrtnout a opsat dolů znovu. I při odchodu zaměstnanců vznikají vyškrtnuté řádky. Tabulka se stává nepřehlednou.

Jiný přirozený způsob ruční evidence je kartotéka. Místo tabulky se vyrobí kartotéční listy, na každém je „formulář" obsahující názvy evidovaných údajů. Každý objekt je zapsán na jednu evidenční kartu, všechny listy jsou umístěny do krabice nebo šuplíku. Výhodou je možnost ukládat listy v nějakém uspořádání (zaměstnanci abecedně podle jména, knihy podle názvu nebo autora apod.) a toto uspořádání dodržovat i při všech změnách, přidávání a rušení karet.

Příklad 1.2.

Evidence dat o zaměstnancích v kartotéce.



Vyhledávání podle jména, pokud je kartotéka takto setříděna a hledající zná abecedu, je rychlejší. Ovšem vyhledání zaměstnanců s bydlištěm v Opavě znamená opět systematické procházení celou kartotékou.

♦

Evidenci údajů je možno provádět i „ručně" s použitím vhodné organizace údajů – jak jsme viděli například v sešitě či kartotéce.

Zatím jsme používali pojmy údaj (množné číslo ~ data) a informace bez přesnějšího rozlišení. Dále budeme nazývat údaji či daty skutečné hodnoty sledovaných vlastností. Podle typu údaje to jsou čísla, časové údaje jako datum a čas, texty jako jména, názvy apod., logické hodnoty ano/ne, případně další typy.

Ovšem známe-li data, nemusí to ještě znamenat, že jsou nám k něčemu užitečná. Aby data dostala smysl, musíme znát jejich význam, jejich interpretaci. Teprve pak z nich dostáváme smysluplnou informaci.

Příklad 1.3.

Údaje: (Novák František, 3, Novák Jiří, 12000, učitel, 3) *samy o sobě nám moc užitečné nejsou. Není jasné ani u obou jmen, čí jsou to jména – zaměstnanec a jeho syn, bratři sportovci .... O čísle 12000 se můžeme dohadovat, že jde o plat a o údaji učitel, že jde o zaměstnání, ale nevíme to jistě, ani ke komu se údaje vztahují. O čísle 3 se dokonce nemůžeme ani dohadovat, co znamená.*

Pokud však víme, že jde o údaje z evidence rodičů dětí se strukturou Rodič (jméno-žáka, postupný-ročník, otec, plat, povolání), jsou údaje srozumitelné.

♦

❑ Základní pojmy zpracování dat

Daty nazýváme údaje získané měřením, pozorováním nebo jen pouhým zaznamenáním z reálné skutečnosti.

Informace jsou smysluplné interpretace dat a vztahů mezi nimi.

Zpracováním dat nebo také hromadným zpracováním dat nazýváme zpracování velkého množství údajů (obvykle desítky až stovky) o velkém množství objektů (obvykle od desítek po miliony i víc).

Objektem nazýváme člověka, zvíře, věc nebo jev reálného světa, pokud se tito stali předmětem našeho zájmu z hlediska evidence. Objekt je popisován množinou svých vlastností. Objekty mají velké množství vlastností, ovšem z hlediska evidence potřebujeme sledovat jen některé z nich.

Atributem nazýváme údaj o objektu, který nás zajímá z hlediska evidence.

Typem objektu (též strukturou objektu) budeme rozumět název množiny objektů a seznam jejich sledovaných atributů:

Jméno-typu-objektu (atribut1, atribut2, …, atributn)

Vést evidenci o objektech znamená

1. zaznamenat vhodně organizované údaje na nějaké médium
2. provádět změny údajů při změně evidované reality
3. provádět výběry informací podle různých kritérií
4. odvozovat a počítat z uložených údajů další
5. třídit údaje dle různých kritérií
6. zaznamenávat vztahy mezi údaji o objektech různých druhů
7. o všech údajích zaznamenaných i odvozených vydávat informace ve vhodné grafické úpravě

Informačním systémem obecně nazýváme organizaci údajů vhodnou pro systémové zpracování dat: pro jejich sběr, uložení a uchování, zpracování, vyhledávání a vydávání informací o nich, to vše pro účely rozhodování.

Shrnutí pojmů 1.1.

Data, informace.
Objekt, atribut. Typ objektu.
Zpracování dat, vedení evidence o objektech.
Informační systém.

Otázky 1.1.

Co znamená hromadné zpracování dat?

Co nazýváme objektem a co atributem?

Které z mnoha vlastností evidovaných objektů patří do evidence?

Jaké přirozené způsoby evidence znáte z praktického života?

Které úlohy jsou typické pro zpracování dat?

Úlohy k řešení 1.1.

1. Firma potřebuje evidenci svých zaměstnanců, kde někteří pracují doma a výsledky práce firmě odevzdávají. Určete, které údaje se musí uchovávat, aby se mohly realizovat následující činnosti:
   - podle zastávané funkce jim vyplácet pevnou mzdu a posílat ji na jejich účet v bance,
   - měsíčně počítat počet zaměstnanců, vyplacenou sumu na mzdy, minimální, maximální a průměrnou mzdu,
   - posílat odvody z mezd sociální a zdravotní pojišťovně, zdravotní pojišťovnu může mít každý zaměstnanec jinou,
   - podle jazykových znalostí jim zadávat práci pro zahraniční zákazníky,
   - podle vzdělání je posílat na specializovaná odborná jednání se zákazníky.

   Výsledek zapište pomocí typu objektu.
2. Najděte v praxi alespoň dalších 5 případů, kdy je zapotřebí dlouhodobě evidovat údaje o nějakém typu objektů. Zapište je jako typy sledovaných objektů.
3. Ke každé evidenci z úlohy 2 najděte alespoň 5 typů informací, které z evidence můžete získat.
4. Pokuste se ke každé požadované informaci popsat postup, jak se z dat získá.

1.2. Agendové zpracování dat

Cíl Po prostudování tohoto odstavce budete umět

- vyjmenovat výhody automatizovaného zpracování dat proti ruční evidenci
- popsat realizaci automatizovaného zpracování dat v klasických programovacích jazycích
- popsat problémy, které při tomto řešení vznikají a na příkladech z programátorské praxe vysvětlit jejich příčiny
- uvést na příkladech výhody a nevýhody agendového zpracování dat
- zdůvodnit, pro které úlohy s databází je výhodné i nyní použít klasický programovací jazyk

Výklad

Využití počítačů pro zpracování dat

Přesto, že první počítače vznikly pro matematické a technické výpočty, velmi brzy se přirozeně začaly používat i pro úlohy zpracování dat. Toto jejich využití bylo dokonce dominantní po dlouhou dobu. Až s rozšířením textových dokumentů, multimediálních prvků a především po vzniku Internetu je procento těchto strukturovaných dat v databázích v porovnání s dalšími typy informací relativně menší.

Od historických dob prvních počítačů dodnes se úlohy evidence dat programovaly v dostupných programovacích jazycích a na dostupných počítačích. Protože většinou nešlo o jediný program, ale o

sadu programů, řešících konkrétní úlohy – agendy, říká se počátečním etapám úloh tohoto typu **agendové zpracování dat**.

Později se vyvinula nová technologie zpracování dat, nazývaná databázovou. O ní pojednávají všechny následující kapitoly. Přes její výhody proti klasickému agendovému zpracování se však dodnes vyskytují nové implementace agendového typu, a proto se s jeho vlastnostmi seznámíme podrobněji.

❑ Historické agendové zpracování dat

se vyznačovalo těmito znaky:

Ke zpracování dat se užívaly střední a velké „sálové“ počítače té doby.

Programovacími jazyky byly původně assembler, Fortran, specializovaný Cobol, později univerzální PL/1, Pascal.

Původní agendy se zpracovávaly v dávkách:

- data se ručně zapisovala do formulářů
- z formulářů se zaznamenávala na vstupní médium pro počítač, na mg. pásku, štítek, disketu
- formou primárního zpracování se data načetla do počítače; přitom se prováděly vstupní kontroly formální a částečně i logické správnosti dat, případně se provádí korekce dat; data jsou uložena na sekundární médium do vnější paměti počítače,
- řadou sekundárních zpracování se pak nad daty provádějí potřebné výpočty, třídění, výběry, tisky sestav; obvykle aplikační programy řeší jednotlivé úlohy, soubor programů pak tvoří ucelenou agendu.
- agendy obvykle řeší menší evidenční úlohy – jedna pro evidenci zaměstnanců, jiná pro majetek firmy, další pro sklad materiálu apod.; často každá agenda v jiném programovacím jazyce, s vlastními daty.

❑ Závislost dat a programů.

U agend existuje **plná závislost dat a programů**. Každý program řeší nejen vlastní aplikační problém, ale i formát fyzického uložení dat na médiu. Navazující úlohy musí respektovat již vytvořené – deklarované fyzické struktury dat. Při změně datové struktury v jednom programu je nutné měnit a kompilovat i všechny další programy, které s touto strukturou pracují, i když se v jejich funkčnosti nic nemění. Odtud plyne nízká efektivnost datových struktur i programů.

Možností, jak tyto stálé změny mnoha programů kvůli struktuře dat omezit, je sbírat a ukládat data specielně pro každou agendu. Tak se ale stává, že stejná data se objeví v různých souborech, pokaždé z jiného důvodu a v jiné souvislosti, případně i s jiným formátem.

Pak mezi různými agendami nejsou žádné nebo jen minimální vazby; vzniká tak izolovanost dat, redundance dat, nekonzistence dat, neintegrita dat. Vysvětleme si tyto pojmy podrobněji.

❑ Problémy agendového zpracování

Z popisu agend vyplývají některé jejich nedostatky, o dalších se ještě zmíníme v následujícím přehledu:

1. **Redundance:** aplikační programy vytvářené postupně různými programátory dle požadavků uživatelů vedou téměř vždy k tomu, že se některé informace ve více souborech opakují, jsou redundandní. Redundance je zdrojem mnoha dalších problémů a bude o ní ještě mnohokrát řeč.

2. **Konzistence:** konzistencí nazýváme vzájemnou shodu údajů. Postupem času (vlivem nedostatečné kontroly v programech, které o sobě nevědí, či vlivem nedostatečné disciplíny uživatele při údržbě dat) se stejné hodnoty, zapsané na různých místech v datových souborech, začnou rozcházet. Při změnách hodnot se oprava položky neprovede na všech místech, kde je položka zapsána, současně jsou v datech hodnoty staré i nové, data ztrácí konzistenci.
3. **Integrita:** aby agenda byla použitelná, musí být uložená data aktuální, tedy popisovat skutečnost z reálného světa - tuto vlastnost nazýváme integritou dat. Integrita souvisí s konzistencí takto: data plně integritní (shodná s realitou) jsou také konzistentní. Ovšem data konzistentní nemusí být integritní (mohou sice konzistentně popisovat realitu, ale zastaralou nebo jinak neplatnou).
4. Problémem tedy je ve všech souvisejících programech **zabezpečit, aby chybou či nedůsledností uživatele** nebyla porušena integrita a konzistence dat**.**
5. **Obtížná dosažitelnost dat:** u rozsáhlejší agendy k navrženým datovým souborům existuje řada aplikačních programů, které řeší konkrétní požadavky, předem specifikované uživatelem. Pokud se objeví potřeba zodpovědět nový typ dotazu, je nutno napsat nový aplikační program, který data zpracuje a vydá výstupní informaci. Bez pomoci programátora je nepředpokládaná informace nezjistitelná, byť se v datech nachází**.**
6. **Izolovanost dat:** data bývají roztroušena v různých souborech, soubory mohou být různě organizovány, data různě formátována. To vše komplikuje tvorbu nových aplikačních programů a téměř znemožňuje možnost realizovat vazby mezi datovými strukturami**.**
7. **Současný přístup více uživatelů:** větší systémy obvykle nevystačí s jednouživatelským provozem, ale vyžadují současný přístup více uživatelů k datům. Pak je nutné, aby programy vzájemně spolupracovaly, jejich činnosti byly koordinovány. Jestliže jeden uživatel aktualizuje údaje a druhý z nich pořizuje sestavu, může být sestava nesmyslná. U dat s vysokou redundancí je téměř nemožné udržet u agendového zpracování konzistenci při povolení víceuživatelského přístupu.
8. **Ochrana proti zneužití:** při zpracování důvěrných či tajných dat není přípustné, aby měl kdokoliv přístup ke všem informacím, případně mohl provádět s daty libovolné operace. Při klasickém zpracování však musí mít programátor aplikačních programů k dispozici tolik podrobností o uložení dat, že to ochranu dat prakticky znemožňuje.

❑ Klasické programovací jazyky a databáze

V současné době vznikající nové úlohy agendového typu bývají často nazývány databází. Jsou to aplikace v klasických programovacích jazycích, u nichž po čase mohou vznikat stejné problémy, jaké uvádíme pro agendové zpracování. Jaký je rozdíl mezi dnešními agendami a popsanými historickými agendami?

- Používají se všechny typy počítačů a jejich HW vybavení i prakticky všechny programovací jazyky.
- Vhodným systémovým návrhem jako výsledkem předběžné analýzy řešených úloh je možné řadě pozdějších komplikací předejít, avšak pokud není stanovena jednotná norma na formu datových struktur, tvorbu souborů a přístup k nim, není možno řešit tyto úlohy univerzálně a odstranit tak nevýhody agend.
- Postupně byla formulována řada teoreticky zdůvodněných nebo prakticky podložených metodologií (strukturované programování, normované programování, modulární programování, databázová technologie, objektově orientovaná analýza, návrh a programování). Avšak ani tyto nástroje zcela nevylučují vznik agendových problémů. Životní realita téměř vždy přináší nové požadavky a potřebu změn existujících programů. Pak se často volí rychlejší, jednodušší cesta jejich splnění a popsané problémy jsou tu opět.

- S rozvinutou databázovou technologií čím dál častěji i tyto jazyky mívají databázovou nadstavbu. Jsou to sady příkazů, které umožňují manipulovat s daty obdobně, jako databázové systémy *(příkladem je programovací jazyk Borland Delphi nebo Borland C-Builder)*. Nejčastěji je přístup k databázi založen na jazyce SQL, o němž bude řeč níže.

Jednou z řemeslných dovedností programátora by mělo být umět zvolit správný nástroj (zde programovací prostředí) pro každý typ řešené úlohy. Pro klasické úlohy evidence údajů jsou vhodné databázové technologie, o nichž bude celá tato učebnice.

Existují však úlohy s databází, kdy je přesto vhodné volit klasický programovací jazyk – případně s databázovou podporou. Jde o úlohy přesahující prostou evidenci údajů: úlohy s rozsáhlými technickými nebo matematickými výpočty, úlohy s řízením provozu v reálném čase apod. I zde jsou ukládána rozsáhlá data, ale hlavním úkolem není jejich evidence, data pouze podporují jiné typy úloh.

Shrnutí pojmů 1.2.

Agendové zpracování dat, agenda.
Závislost dat na programech.
Redundance, nekonzistence, neintegrita, špatná dosažitelnost dat.
Víceuživatelský provoz agend.
Bezpečnost dat před zanesením chyb nebo zneužitím.

Otázky 1.2.

1. Jak se programují úlohy evidence dat v tzv. agendovém zpracování?
2. Co je redundance dat a jak vzniká při agendovém zpracování?
3. Jaké problémy mohou plynout z redundance dat a proč?
4. Co je integrita dat a jaký je její vztah ke konzistenci?
5. Které další problémy nastávají při realizaci úloh evidence dat klasickými programovacími jazyky?
6. Existují v současné době nové programy agendového typu?

1.3. Databázové zpracování dat

Cíl Po prostudování tohoto odstavce budete umět

- popsat důvody vzniku databázové technologie
- formulovat paradigma databázové technologie
- vysvětlit rozdíly mezi agendovým a databázovým zpracováním dat

Výklad

❑ Vznik a základní pojmy databázové technologie

Zkušenosti získané při programování agend ukázaly, že všechny agendy používají jen několik málo datových typů a několik málo typů operací s daty. Aplikační programy realizující jednotlivé úlohy a agendy se liší tím, v jakém pořadí a s kterými údaji tyto operace provádějí.

Příklad 1.4.

Při evidenci zaměstnanců potřebujeme uložit atributy rodné číslo, jméno, datum nástupu, funkci, plat, procento daně, zda má řidičský průkaz, ... Z hlediska datových typů to jsou čísla, texty, datumové údaje, logické údaje. Evidované údaje potřebujeme: kontrolovat jejich správnost při ukládání (RČ splňuje pravidla, funkce je ze známé množiny, plat odpovídá funkci, ...), vyhledávat (podle jména, řidičáku, funkce, ...), třídit (podle jména, data nástupu,, funkce, oddělení, ...), dopočítávat (čistý plat z hrubého a daně, datum narození, věk či pohlaví z RČ, ...), sčítat, průměrovat, ... (suma platů pro výplatu, průměr platů za oddělení, firmu pro statistiky, ...).

Při evidenci materiálu na skladě potřebujeme ukládat jeho atributy: název, jednotkovou cenu, procento DPH, typ materiálu, množství na skladě, datum a množství příjmu a výdeje, pro kterou zakázku byl vydán, zda je materiál volně k prodeji, ... Z hlediska datových typů to jsou opět čísla, texty, datumové údaje, logické údaje. Evidované údaje potřebujeme: kontrolovat jejich správnost při ukládání (množství přijaté je kladné, typ je ze známé množiny, DPH odpovídá zákonu, ...), vyhledávat (podle názvu typu, ...), třídit (podle názvu, typu, data výdeje, zakázky, ...), dopočítávat (celková cena z jednotkové a množství, cena včetně DPH, ...), sčítat, průměrovat, ... (suma za sklad, suma za zakázku, průměr cen za zakázky pro statistiky, ...).

♦

To vše vedlo k návrhu a vytvoření nových programových systémů následujících vlastností:

- definují **seznam datových typů,** které jsou v programovém systému použitelné; pro tyto typy dat programový systém vytváří fyzickou strukturu na disku a automaticky řeší všechny přístupy k datům; proti programovacím jazykům používají některé datové typy navíc nebo jiné (reálná čísla s pevně umístěnou desetinnou tečkou, datumové a časové údaje, …);
- obsahují prostředky pro definování **důležitých vlastností popisovaných objektů**;
- programový systém řeší způsob, jak zaznamenat **vztahy mezi objekty;**
- obsahují **soubor instrukcí,** které nad definovanými daty provádějí jednotlivé operace; každá instrukce je vlastně mohutnou procedurou, v níž je řešen fyzický přístup k datům i realizace vlastní operace; jinak než prostřednictvím systému není možno s daty pracovat.

Takovým programovým systémům - vlastně programovacím jazykům vyšší úrovně, pracujícími s rozsáhlými datovými soubory - říkáme **systémy řízení báze dat.** Systémy řízení báze dat dodávají výrobci programového vybavení.

Základní databázovou úlohou je evidence řady údajů – atributů o sledovaných objektech.

Formálně je atribut popsán svým názvem (identifikátorem) a datovým typem, v programovacím jazyce je realizován **položkou**. Položkou nazýváme logicky dále nedělitelnou jednotku ve struktuře dat. Možnosti definovat různé datové typy položek úzce souvisí s použitým programovacím prostředím – každý jazyk může obsahovat poněkud jinou množinu datových typů, i když základní typy číselné, textové, datumové se vyskytují vždy.

Někdy není atribut jednoduchý, ale může se několikrát opakovat a pak mu říkáme multiatribut. Realizován je buď **polem** (array) – položka se opakuje daný počet-krát, nebo jinak, pokud se opakuje předem neznámý počet krát.

Některý atribut není jednoduchý, ale skládá se z několika položek obecně různého typu. Pak jej nazýváme **skupinovou položkou** (record).

Příklad 1.5.

V následujícím typu objektu

Zaměstnanec (jméno (křestní, příjmení, titul), adresa (ulice, obec, PSČ), vzdělání:multi, …)

jsou jméno a adresa *skupinové položky,* vzdělání *opakující se položka neznámý počet-krát.*

♦

Při výběru typu položky se rozhodujeme pro ten datový typ, který nejlépe a přitom nejúsporněji z hlediska uložení i zpracování dat zaznamená daný údaj. Položky zaznamenávají jednotlivé údaje o objektu a teprve celá posloupnost položek popisuje objekt. Taková struktura položek, která má ucelený význam (zachycuje všechny potřebné údaje o sledovaném objektu) se nazývá **záznamem** (větou, recordem). Je to obvykle skupinová položka.

Záznam tedy podává kompletní informaci o jednom objektu našeho zájmu. Obecně délka věty může být proměnná, jestliže údaje o různých objektech mají různou vnitřní strukturu (voják-nevoják u mužů, počet dětí u žen ap.). Častěji (a z programátorského hlediska pohodlněji) se používají věty s pevnou strukturou údajů a tedy s pevnou délkou věty.

Množinu výskytů záznamů stejného typu, která zaznamenává ucelenou informaci o množině sledovaných objektů a která je uložena na paměťovém médiu, nazýváme **datovým souborem**. Množiny záznamů si můžeme snadno představit ve tvaru **tabulky**, kde každý objekt je popsán jedním řádkem a každý atribut objektu je v jednom sloupci.

Množinu datových souborů, uchovávajících data o nějakém uceleném úseku reality, nazýváme **databází.**

Programový systém (prázdný, bez datových souborů a bez aplikačních programů), umožňující definování datových struktur a datových souborů, řešící fyzické uložení dat ve vnější paměti počítače, umožňující manipulaci s daty a formátování vstupních i výstupních informací, nazýváme **systémem řízení báze dat (SŘBD).**

Příklad 1.6.

Dosud jsme nejmenovali žádné SŘBD na našem trhu. Existuje jich mnoho, z nejznámějších jmenujme např. MS Access, Visual FoxPro, Oracle, MS SQL Server. Některé z nich jsou určeny pro databáze menších rozsahů a informační systémy nad databází pracující s menším počtem uživatelů. Jiné jsou velké (taky drahé) a zvládají jak rozsáhlé databáze (statisíce i miliony záznamů), tak vysoký počet uživatelů (stovky, tisíce, ...).

♦

Poznámka:

Někdy se v programátorském žargonu zkráceně místo SŘBD říká „databáze“. Není to přesné a v této učebnici se budeme držet přesně zavedených pojmů. Pokud se z vás časem stanou profesionálové a budete se dobře orientovat v tom, s kým a o čem mluvíte, je možné opět začít používat pojmy volněji.

Aplikační úlohou nad SŘBD nazýváme konkrétní program napsaný pomocí programových prostředků použitého SŘBD nad konkrétní databází, pro tuto úlohu vytvořenou.

Aplikační úlohy nad společnou databází pak mohou tvořit ucelený systém, nazývaný **databázovým nebo (automatizovaným) informačním systémem** (dále jen IS) nad použitým SŘBD. V tomto pojetí tedy databázovým systémem rozumíme celek, řešící rozsáhlejší oblast aplikační, naprogramovaný v jednom SŘBD s vhodně navrženými datovými strukturami tak, aby všechny aplikační úlohy k nim

měly optimální přístup. Řeší uložení, uchování, zpracování a vyhledávání informací a umožňuje jejich formátování do uživatelsky přívětivého tvaru.

V dalším výkladu budeme pod pojmem informační systém rozumět vždy automatizovaný informační systém, pokud výslovně neuvedeme jinak.

❑ Paradigma databázové technologie

Definování datových typů a operací nad daty není vše, čím se liší databázová technologie od klasického programování. Nejpodstatnější rozdíl, základní princip či tzv. paradigma databázové technologie je

oddělení datových struktur od programů

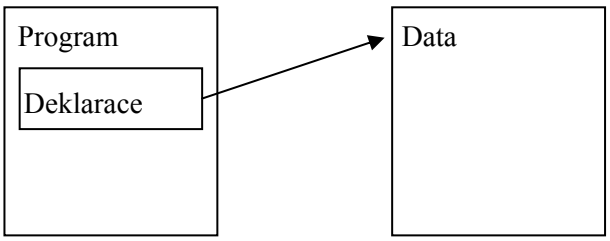
Tuto vlastnost zabezpečuje v SŘBD možnost definovat datové a programové struktury samostatně a nezávisle na sobě. Struktury datových souborů jsou uloženy samostatně nebo jsou součástí datových souborů. Programy s nimi pracují tak, že si načtou strukturu dat a pak s datovým souborem mohou provádět potřebné operace. Při změně datové struktury není nutné měnit programy, při změně programů není nutné měnit datové struktury.

Příklad 1.7.

Při programování evidence zaměstnanců např. v Pascalu je deklarace záznamu Zaměstnanec (jméno, adresa, funkce, plat) součástí programu:

```
Program Evidence_zamestnancu;
 ...
var Zamestnanec: record of jmeno: string [1..20];
                           adresa: string [1..50];
                           funkce: string [1..10];
                           plat: integer
                 end;
Begin
 ...
End.
```

Situace vypadá takto:



Změní-li se struktura souboru například přidáním nového atributu `věk`, *je nutné změnit deklaraci a přeložit program znovu i v případě, že na funkčnosti programu se nic nezměnilo. Existuje-li již řada programů pracujících se záznamem* Zamestnanec, *je potřeba upravit všechny. Dále je třeba napsat konverzní program pro překopírování staré struktury do nové, přeložit a konverzi provést.*

V databázové technologii se deklarace záznamu píše ne do programu, ale do hlavičky datového souboru. Definice struktury souboru se provede samostatným příkazem SŘBD například tvaru:

```
CREATE TABLE Zamestnanec (jmeno CHAR(20), adresa CHAR(50),
                          funkce CHAR(10), plat NUMBER(8,2));
```

Tím SŘBD vytvoří tabulku zadaného jména, která má v hlavičce zadanou deklaraci. Nemusí zatím existovat žádný program pracující s touto tabulkou.

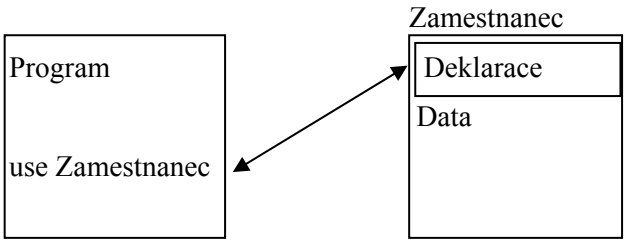
Při psaní programu stačí programátorovi znát identifikátor datového souboru, identifikátory a datové typy atributů a není třeba znát jejich fyzické umístění v záznamu. Po spuštění programu si program najde datový soubor dle názvu, z hlavičky načte jeho strukturu, tj. identifikátory atributů a jejich umístění v záznamu. Tak může načíst nebo zapisovat ty atributy, se kterými pracuje.

Při změně struktury souboru např. přidáním atributu `věk` *provede programátor pouze změnu datového souboru vhodným příkazem, například*

```
ALTER TABLE Zamestnanec ADD (vek NUMBER (2))
```

Příkaz provede nejen změnu popisu v hlavičce tabulky, ale provede automaticky i konverzi dat ze staré struktury do nové. Na programech pracujících s datovým souborem není třeba nic měnit. Opět si po spuštění načte strukturu (tentokrát novou) a vybere podle ní potřebná data pro svou práci.

Situace vypadá takto:



♦

Shrnutí pojmů 1.3.

Systém řízení báze dat.
Databáze, aplikační úloha nad databází implementovaná v SŘBD.
Automatizovaný informační systém.
Paradigma databázové technologie.

Otázky 1.3.

1. Co je systém řízení báze dat?
2. Jaký je základní rozdíl mezi klasickými programovacími jazyky a systémy řízení báze dat?
3. Co nazýváme aplikační úlohou v informačním systému?
4. Co nazýváme paradigmatem databázové technologie a v čem je jeho princip?

2. DATABÁZOVÁ TECHNOLOGIE

Čas ke studiu kapitoly: 3 hodiny

Cíl Po prostudování této kapitoly budete umět

- definovat a používat pojmy databázové technologie
- popsat její základní vlastnosti
- popsat etapy budování databáze

Výklad

2.1. Obecné vlastnosti databázové technologie

❑ Vlastnosti databázové technologie

Výše popsané problémy agend a vyextrahování základních typů dat a operací s nimi vedly ke vzniku SŘBD splňujících následující vlastnosti:

1. Paradigma databázové technologie - **oddělení datových struktur od programů**.

2. Existuje seznam základních datových typů, které jsou v SŘBD definovány; jejich kombinace vytvářejí libovolné uživatelem definované datové struktury; pro tyto typy dat SŘBD vytváří fyzickou strukturu na disku, automaticky řeší všechny přístupy k datům. Součástí SŘBD je soubor prostředků, pomocí nichž se datové struktury definují a který nazýváme **jazykem pro definici dat** (JDD).

3. Existuje soubor instrukcí, které nad definovanými daty provádějí jednotlivé operace; každá instrukce je vlastně mohutnou procedurou, v níž je řešen fyzický přístup k datům i realizace vlastní operace; jinak než prostřednictvím systému není možno s daty pracovat. Tento soubor instrukcí nazýváme **jazykem pro manipulaci s daty** (JMD).

4. SŘBD řeší způsob, jak zaznamenat **vztahy mezi objekty**.

5. Data je možno zpracovávat libovolným i předem nepředpokládaným způsobem. Pro zodpovězení dotazů náhodných uživatelů je v SŘBD další typ jazyka - **dotazovací jazyk**. Ten umožňuje formulovat většinu dotazů na informace v databázi uložené bez nutnosti psát program pro jejich vyhledání.

6. SŘBD umožňuje **víceuživatelský přístup** k informacím; buď k tomu poskytuje v rámci JMD prostředky aplikačnímu programátorovi, nebo řeší standardní situace víceuživatelského přístupu automaticky.

7. SŘBD umožňuje **ochranu dat** před zneužitím použitím hesel, definováním přístupových práv na úrovni souborů, záznamů, položek, rozlišením práv pro zápis, čtení, modifikace. Tak ani programátor, znalý struktury dat, nemá přístup k reálným datům.

❑ Uživatelé databázové technologie

Se SŘBD pracuje na různých úrovních několik typů uživatelů, dělících se podle způsobu komunikace s databází:

- **Správce nebo administrátor báze dat** je profesionální analytik a systémový programátor, který rozhoduje o tom, která data a jak budou v bázi uložena. Určuje metody přístupu k datům, pokud je to nutné, modifikuje struktury dat, přiděluje přístupová práva k datům, rekonstruuje databázi v případě jejího poškození ap.
- **Aplikační programátor** je profesionální programátor, který programuje aplikační úlohy nad definovanými datovými strukturami pomocí programových prostředků SŘBD. Nemusí znát strukturu celé databáze, stačí mu znalost struktur, se kterými bude pracovat a které mu zadá správce. Tak může nad jednou databází pracovat bez nutnosti vzájemné komunikace řada programátorů.
- **Příležitostný uživatel** je jakýkoliv uživatel, který umí prostřednictvím dotazovacího jazyka formulovat svůj dotaz (takový, který databázový systém sám nemá zabudován ve svých aplikačních programech).
- **Naivní uživatel** je takový uživatel (obvykle neprogramátor), který prostřednictvím aplikačních programů pracuje s databází a používá tak databázi jako informační systém pro ukládání, zpracování a vyhledávání informací. Především pro tyto uživatele se databázové systémy vytvářejí.
- Nezmiňujeme se zde samozřejmě o programátorech, kteří SŘBD vytvářejí. S těmi se většinou nesetká ani správce báze či aplikační programátor, stejně jako se programátor v C++ obvykle nesetká s autory překladače.

Poznámka: V současné době existuje již řada SŘBD, které umožňují pro menší úlohy, aby uživatel počítače (často vlastník a jediný uživatel) si sám definoval struktury souborů a psal jednoduché programy, nebo alespoň pracoval s databází konverzačně. Takový uživatel je sám sobě správcem, aplikačním programátorem i naivním uživatelem. Obvykle však (pokud nejde o profesionálního programátora) se omezuje na malé aplikace. I on by měl znát základy práce s databází, ovšem jejich neznalost a nedodržování nemívají velké následky.

Shrnutí pojmů 2.1.

Paradigma databázové technologie.
Jazyk pro definici dat, jazyk pro manipulaci s daty, dotazovací jazyk.
Uživatelé databázové technologie – administrátor databáze, aplikační programátor, naivní uživatel, příležitostný uživatel.
Víceuživatelský přístup, ochrana dat z databáze.

Otázky 2.1.

1. Jaký je rozdíl mezi jazykem pro definici dat a jazykem pro manipulaci s daty?
2. Co nazýváme dotazovacím jazykem a komu slouží?
3. Které další problémy zpracování dat řeší SŘBD?

2.2. Entity, atributy, vztahy, integritní omezení

❑ Pojmy entita, atribut, doména, integritní omezení

V teorii databázových systémů je nutné zavést přesnější pojmy, než dosud používané a převzaté z agendového zpracování úloh.

Entitou rozumíme libovolnou existující osobu, zvíře, věc či jev (obecně objekt) reálného světa. Entita musí být rozlišitelná od ostatních entit a existovat nezávisle na nich.

Atribut je charakteristika, vlastnost entity, údaj o objektu. Atribut přiřadí každé entitě z množiny entit hodnotu z nějaké neprázdné množiny hodnot, nazvané **doména atributu** (obor hodnot atributu). Atribut je tedy zobrazení množiny entit do domény atributu. Je zadán svým názvem (identifikátorem) a datovým typem.

Typem entity nazýváme množinu objektů stejného typu, charakterizovaných názvem typu a strukturou jejich atributů. Jednotlivé entity nazýváme také výskyty nebo **instancemi** objektů entitního typu. Instance entity je tedy konkrétní n-tice hodnot atributů jedné konkrétní entity.

Při formulaci zadání databázové úlohy se tedy obvykle zadává množina typů entit, které budou v databázovém systému evidovány a zpracovávány. Jejich výběr obvykle není jednoduchou záležitostí a bude diskutován u datové analýzy.

Jeden atribut nebo množinu atributů, které jednoznačně určují entitu v množině entit, nazveme **klíčovým atributem.** Kandidátů na klíčový atribut může být mezi atributy více; pak vybíráme za klíč ten, který je z hlediska zpracování dat nejefektivnější. Někdy se volí pro pohodlné zpracování jako klíč i uměle definovaný atribut. Atributy patřící k některému klíči nazýváme **primárními**. Atributy, které nepatří k žádnému klíči nazýváme **sekundárními**.

Příklad 2.1.

Entitní typ: zaměstnanec firmy: Zam(RČ, jméno, adresa, funkce, plat, dat_nar, místo-nar)
Entita: Novák Josef z Karviné
Instance entity : (444444.4444, Novák Josef, Karviná, zámečník, 9000, 4.8.1968, Ostrava)
Atributy: jméno, plat, ...
Doména jména: množina možných jmen
Doména platu: množina čísel <0,30000>
Zobrazení odpovídající atributu plat : plat(Novák Josef) = 10000
kandidáti na primární klíč: jméno, datum a místo narození
rodné číslo
sekundární atributy: adresa, funkce, plat

♦

Obecně entity mohou mít velmi mnoho atributů. Pro zpracování v IS je podstatné vybrat z nich alespoň jeden klíčový a pak všechny ty, které jsou nutné z informačního hlediska pro zamýšlené zpracování dat – které nás zajímají z hlediska evidence.

Na hodnoty atributů mohou být kladeny omezující podmínky, které upřesňují jejich doménu a které nazýváme **integritní omezení (IO)**. Níže uvedeme IO i jiná, než na hodnoty atributů.

Často znázorňujeme množinu entit jako datovou matici či **tabulku**, která má název (= název typu entity) a seznam sloupců (= struktura typu entity). Každý sloupec má název (= název atributu) a datový typ. Pak každá entita (instance entity) je znázorněna v tabulce jedním řádkem, každý typ atributu je definován jedním sloupcem, hodnota atributu dané entity je v odpovídajícím řádku a sloupci tabulky.

❑ Vztah entit

Mezi entitami může v realitě existovat nějaký vztah, který nás zajímá a chceme jej také evidovat. Takový vztah pak zase pojmenujeme a popíšeme jeho strukturou – tentokrát bude struktura vztahu popsána seznamem typů entit, které do něj vstupují. Uveďme nejprve definici nejčastěji se vyskytujícího vztahu mezi dvěma typy entit:

Definice:

Mějme dvě množiny entit E1, E2. Pak mohou existovat dvojice (e1, e2), ei ∈ Ei, které jsou mezi sebou v nějakém vztahu **v**. Existuje-li vztah **v** a zajímá nás z hlediska evidence, můžeme dvojici (e1, e2) považovat za entitu (tentokrát popisující **vztah objektů**, nikoliv objekt) a množinu všech takových dvojic, které jsou mezi sebou v témže vztahu **v**, nazýváme **typem vztahu V** mezi množinami entit E1, E2.

Zobecnění definice pro k-tice typů entit:

Definice:

Mějme uspořádaný seznam (ne nutně různých) množin entit E1, E2, ..., Ek a nechť k-tice entit (e1,e2,...,ek), ei ∈ Ei je vzájemně mezi sebou v nějakém vztahu **v**. Zajímá-li nás vztah **v** z hlediska evidence, pak můžeme k-tice (e1, e2, ..., ek) považovat za vztahovou entitu a množinu všech takových k-tic, které jsou mezi sebou v témže vztahu **v**, nazvat typem vztahu V mezi množinami entit E1, ...,Ek.

Nejčastější je případ k=2, vztah binární. Vztahy entit dále klasifikujeme dle počtu možných vazeb jedné entity k entitám druhé množiny. Pro binární vztahy:

1. nejjednodušší je vztah **1:1**, kdy každá entita z první množiny entit je spojena vztahem s nejvýše jednou entitou druhé množiny.

Příklad 2.2. *vztah "je vedoucím katedry" mezi množinami entit*
E1 = Zaměstanec VŠ a E2 = Katedra VŠ
zapíšeme. VEDOUCÍ_KAT(Zaměstnanec, Katedra) ♦

2. obecnější je případ **1:N**, kdy každá entita z E1 je spojena vazbou s žádnou či více entitami z E2, ale každá entita z E2 je spojena vazbou nejvýše s jednou entitou z E1.

Příklad 2.3. *vztah "je členem katedry" mezi entitami Zaměstnanci a Katedry*
zapíšeme: ČLEN_KAT(Zaměstnanec, Katedra). ♦

3. nejobecnější je případ **M:N**, kdy nejsou kladena žádná omezení na množiny entit spojených příslušným vztahem; každá entita z E1 tedy může mít vztah k N entitám z E2 a naopak každá entita z E2 může mít vztah k N entitám z E1.

Příklad 2.4. *E1 je soubor Firem*
E2 je soubor Výrobků,
vztah V je "Firma vyrábí Výrobek" ... VYRÁBÍ(Firma, Výrobek) ♦

I když vztahy dvou množin entit jsou nejčastější, existují i složitější vztahy mezi třemi a více množinami entit (n-ární vztahy) nebo vztahy unární.

Příklad 2.5. ternárního vztahu:
E1 je soubor učitelů, E2 je soubor vyučovaných předmětů, E3 je soubor tříd
Binární vztahy mezi E1, E2, E3:
V1 : "učitel učí předměty" typu M:N
V2 : "třída má předepsány předměty" typu M:N
V3 : "učitel učí ve třídě: typu M:N

Z uvedené trojice vazeb V1 - V3 nevyplývá, který učitel učí který předmět ve které třídě. Tuto informaci musíme popsat novou vazbou mezi trojicí entit
V4 : "Učitel učí Předmět ve Třídě"
zapíšeme: UČÍ(Učitel, Předmět, Třída) ♦

Obdobně mohou existovat vztahy mezi více množinami entit (n-ární vztahy) nebo vztahy unární.

Příklad 2.6. *E1 je soubor zaměstnanců*
Vztah mezi E1 a E1:
V1 : "je vedoucím zaměstnance" typu 1:N ♦

Vztahy mezi entitami je nutno také formálně popsat, proto jsme zavedli pojem **vztahová entita** a její typ. Na rozdíl od entity, popisující některý objekt, popisuje vztahová entita některý vztah mezi objekty. Typ entity opět pojmenujeme názvem vztahu a jeho atributy budou tvořit typy entit, které do popisovaného vztahu vstupují. Instance vztahu pak budou konkrétní dvojice či n-tice entit, vstupující do vztahu.

Vztahu se říká **vazba bez informace**, když obsahuje jako atributy pouze typy entit vstupující do vztahu. Vazební entita však může někdy obsahovat i další atributy, zaznamenávající vlastnosti vazby, které nejsou mezi atributy jednotlivých entit. Pak ji nazýváme **vazbou s informací**.

Příklad 2.7. *typ vztahové entity:* UČÍ (Učitel, Předmět)
instance vztahu: (Horák Pavel, Databázové systémy)
Příklad 2.8. *entity: Muž, Žena*
vztah: MANŽELSTVÍ (Muž, Žena)
další atributy vztahu: dat-sňatku, dat-nar-1-dítěte
instance vztahu: (Kovář Karel, Železná Marie, 5.6.93, 5.6.94) ♦

Poznámka: V příkladech jsme dodržovali toto pravidlo o zápisu identifikátorů (pojmenování) atributů, entit a vazeb: atributy zapisujeme malými písmeny,
entity s prvním písmenem velkým,
vztahy velkými písmeny.

Není to povinné značení, ale jeho dodržování nám velmi urychlí orientaci v tom, jaká je role jednotlivých identifikátorů. Proto jej budeme i nadále dodržovat.

Integritní omezení (IO) mohou upřesňovat nejen hodnoty atributů, ale mohou se týkat i entit a jejich vazeb. Obecně každou doplňující informaci o objektech, atributech a vazbách, která plyne z reality a kterou je nutno brát v úvahu v IS, nazýváme integritním omezením (uvádí, jak zabezpečit shodu reality a databáze, tedy integritu databáze).

Příklad 2.9. *entity:* Zam(jméno, rod_cis, plat, fce)
Kat(číslo_kat, název_kat)

***IO pro hodnotu atributu**: rod_cis je deseticiferné číslo, kde první dvojice je ..., druhá dvojice je ..., třetí ..., ciferný součet celého čísla je dělitelný 11.*
***IO pro příslušnost entity k množině Zam**: člověk daného jména a rodného čísla je Zaměstnancem naší školy.*
***IO pro vztah ČLEN_KAT:** každý zaměstnanec je členem právě jedné katedry.*

Shrnutí pojmů 2.2.

Entita, atribut, typ entity, vztah entit, typ vztahu, vztah s/bez informace, integritní omezení.

Otázky 2.2

1. Jaký je rozdíl mezi entitou a typem entity?
2. Jak definujeme typ entity?
3. Co je atribut a které atributy jsou v rámci typu entity důležité?
4. Je tatáž entita popsána vždy stejnými atributy?
5. Co je vztah mezi entitami a kolik entit může do takového vztahu vstupovat?
6. Jaký je rozdíl mezi vztahem entit a typem vztahu?
7. Co je vztah s informací a vztah bez informace?
8. Co jsou integritní omezení a čeho se týkají?

Úlohy k řešení 2.2.

1. Najděte alespoň 3 vlastní příklady na každý z pojmů uvedených v otázkách 2.2.

2.3. Architektura databáze

❑ Postup při budování databáze

Vytváření struktury databáze znamená postupné vyřešení úkolů, které je možno rozdělit do několika úrovní podle míry abstrakce.

Zadání vychází z potřeb reálného světa. Určí se takové typy objektů a údajů o objektech, které je potřeba evidovat a které chceme zahrnout do databáze informačního systému. Tyto požadavky má popisovat zadavatel. Vybrat podstatné a dostatečné informace pro IS však zdaleka nebývá jednoduché a často zadavatel spolupracuje s analytikem IS.

Analytik pak provádí analýzu zadání, datovou a funkční, případně časovou.

IS je používán obvykle řadou uživatelů, přičemž každý z nich pracuje jen s částí celé databáze. Pohledy jednotlivých uživatelů na databázi nazýváme **externí schémata**. Při zadání se obvykle vychází z požadavků jednotlivých uživatelů, tedy z externích schémat. Datový analytik musí provést integraci všech požadavků tak, aby překrývající se požadavky nevedly k opakovanému výskytu entit a atributů.

Datová analýza je proces poznávání objektů reálného světa, jejich vlastností a vazeb: vytipování, které jsou potřebné pro zamýšlený informační systém, jakými entitami a atributy budou objekty popsány. Výsledkem datové analýzy (po integraci požadavků z externích schémat) je informační struktura zvaná **konceptuální schéma** databáze.

Analýza chování objektů v reálném světě se nazývá **funkční analýza**. Popisuje jednotlivé akce prováděné nad objekty reálného světa, které jsou zaznamenány v konceptuálním schématu databáze. Výsledkem funkční analýzy je pojmenování a popis akcí, které se nad datovými strukturami provádějí. Při funkční analýze se mimo jiné ověřuje, zda jsme při datové analýze nezapomněli na některé atributy.

Proces návrhu databáze na logické úrovni, který z požadavků reality definuje strukturu databáze, se nazývá **konceptuálním modelováním.** Výsledek se popisuje jazykem sice formalizovaným, ale současně pochopitelným i zadavateli.

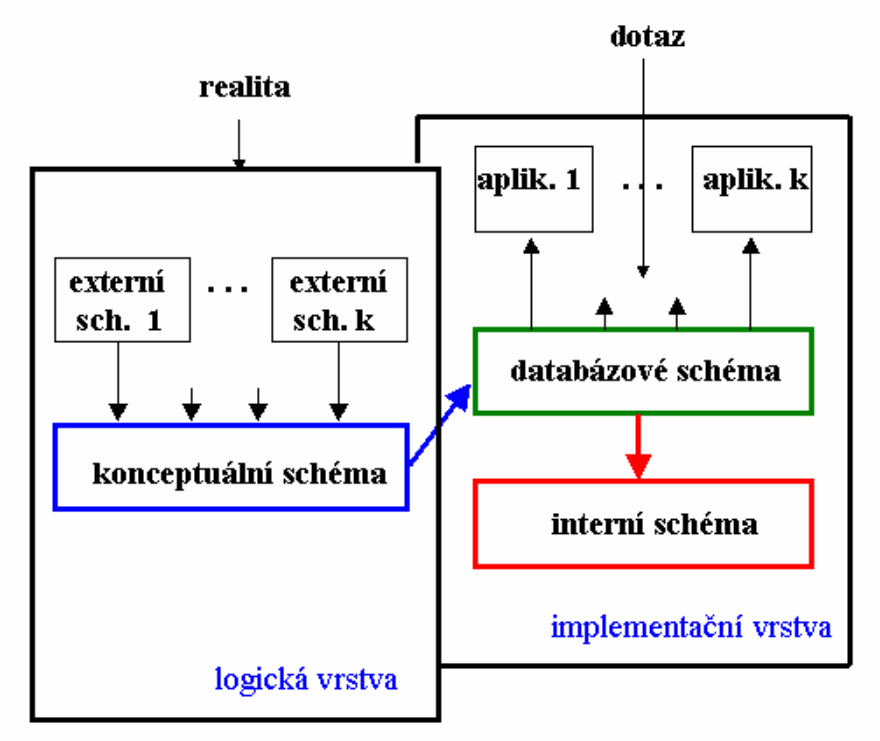
Konceptuální schéma je logickým popisem struktury a chování báze. Zatím není podstatné, jak bude báze implementována. Na níže uvedeném obrázku jsou zakreslena na popředí v **logické vrstvě** celé struktury.

Zadní vrstva na obrázku tvoří **implementační vrstvu.** Rozhodnutí o vlastní implementaci báze souvisí s výběrem použitého SŘBD a realizací popsaných datových struktur, jejich vazeb a akcí nad nimi prováděných. Tentýž konceptuální model je možno realizovat v různých SŘBD a jim odpovídajících datových modelech.

Realizací konceptuálního schématu v konkrétním SŘBD je **databázové schéma**, tedy definice datových struktur a jejich vazeb pomocí prostředků použitého SŘBD.

Realizací externích schémat jsou data viděná jednotlivými **aplikačními programy** nad databázovým schématem, které provádějí akce jednotlivých uživatelů.

Interní schéma databáze popisuje nejnižší fyzickou úroveň uložení dat na médiu počítače. Definuje fyzické záznamy, fyzickou reprezentaci jejich položek, sdružování záznamů do souborů, charakteristiky těchto souborů. Souvisí bezprostředně s použitým SŘBD a ten ji také realizuje. Ani aplikační programátor neřeší fyzické uložení dat.



Obr. 1. Třístupňová architektura databáze

Popsaný postup se někdy nazývá třístupňovou architekturou databáze. Tři úrovně tvoří databáze na 3 stupních vývoje: I. **konceptuální schéma** neboli logický popis databáze

II. **databázové schéma**, popis databáze definované v konkrétním typu SŘBD

III. **interní či fyzické schéma**, konkrétní implementace datových souborů

Schéma vývoje databáze odpovídá svou logikou životnímu cyklu vývoje informačního systému - zatím pokud jde o datovou stránku IS. Popis reality odpovídá zadání. Analýza externích schémat a jejich integrace do konceptuálního schématu odpovídá datové analýze, analýza požadavků na chování

systému funkční analýze. Převod logického modelu do databázového odpovídá datovému návrhu. Návrhem implementace funkcí se zatím zabývat nabudeme. Realizace aplikačních úloh pak je implementací.

❑ Datové modely

Pro popis schémat databáze na různých úrovních používáme tzv. datové modely. **Datový model** je souhrn prostředků pro

- popis datových struktur pomocí typů entit
- přiřazení popisných atributů jednotlivým typům entit
- popis vazeb mezi daty pomocí typů vztahů
- popis integritních omezení k vyjádření souladu s realitou.

Datové modely obvykle rozdělujeme do tří skupin, odpovídajících výše uvedeným úrovním a vrstvám třístupňové architektury databáze:

❑ Konceptuální datové modely

Modelují realitu pomocí objektů a jejich vlastností na logické úrovni bez bližší specifikace o budoucí implementaci. Někdy jsou nazývány modely objektovými. Jsou určeny k záznamu konceptuálního modelu, tedy zápisu struktury databáze na logické úrovni. Konceptuální schéma je výsledkem datové analýzy a je nutné, aby mu rozuměl i zadavatel - neprogramátor, ať pro konzultace nad zadáním, zpřesňování modelu jako odrazu reality či pro definici přesného zadání před podpisem smlouvy.

V SŘBD se většinou neukládá význam jednotlivých údajů, jsou známy jen identifikátory entit a atributů a datové typy atributů, nejvýše stručné komentáře o atributech. Z vlastní databáze pak není možno rekonstruovat význam údajů a akcí nad nimi. Je proto nutné někde popsat význam dat v databázi, význam vztahů mezi nimi, význam akcí prováděných nad daty. Informace o tom, jak interpretovat data v databázi, jsou uloženy v konceptuálním schématu.

Podrobně budou metody záznamu konceptuálního schématu popsány v kapitole 3.

❑ Záznamově orientované datové modely

Modelují realitu na logické úrovni, ale ve tvaru odpovídajícímu požadavkům typu datového modelu SŘBD při budoucí implementaci. Vycházejí z přirozené představy, že vlastnosti jednoho objektu tvoří logický celek (n-tici), který je pohodlné zobrazit jako záznam.

Podle typu datového modelu se dělí na
relační
síťové (+ zvláštní případ síťového modelu – hierarchické)

Relační model popisuje vztahy tabulkami, do kterých se zapisují objekty vstupující do vztahů, příp. jejich klíče. Hlavní výhodou relačního modelu je fakt, že entity i vztahy realizuje jako dvourozměrné tabulky, které se snadno implementují jako soubor.

Podrobněji probereme tento nejrozšířenější datový model v kapitole 4.

Síťový a hierarchický model znázorňují vztah pomocí adresy nebo čísla záznamu např. tak, že ke každému záznamu je připojena systémová část s tolika odkazy, ke kolika jiným typům záznamů je vázán.

Reprezentace vztahu odkazem předpokládá, že při popisu struktury záznamu představujícího entitu víme, ke kolika jiným entitám může mít vztah. Popis vztahu tímto způsobem tedy není tak pružný, ale je mnohem úspornější a především rychlejší v provozu.

Podrobněji probereme tento historicky první datový model v kapitole 6.

- **Fyzické datové modely**

Na nejnižší úrovni jsou data fyzicky uložena na vnějších paměťových médiích. Pro operační systém je databáze soustavou souborů, která se neliší od jiných datových souborů. Operační systém poskytuje základní typy organizací dat, především sekvenční přístup. Aby se tyto jednoduché prostředky operačního systému daly využít pro realizaci podstatně složitějších organizací dat z úrovně konceptuální, je nutné vytvořit vhodný interní datový model, implementovaný v SŘBD. Některé SŘBD dokonce nepoužívají služeb operačního systému při ovládání souborů a řeší všechny přístupy na vnější paměť vlastními prostředky.

Konkrétním fyzickým organizacím dat, které se u SŘBD používají, věnujeme kapitolu 5.

Komponenty SŘBD v návaznosti na tři stupně DBS

SŘBD, v němž se implementuje databázový systém, můžeme podle návaznosti na tříúrovňové schéma DBS rozložit do podsystémů.

Nejnižší úroveň, odpovídající interní úrovni databáze a pro uživatele neviditelnou, ovládá subsystém pro ovládání souborů. Zahrnuje fyzickou organizaci datových souborů, vlastní uložení dat na vnějším médiu a realizaci přenosů dat. Často SŘBD řeší jen organizaci dat a pro komunikaci s vnější pamětí používá služeb operačního systému. Některé SŘBD však i tyto přenosy realizují samy.

Střední úroveň databázovou zajišťuje jeden z jazyků SŘBD - jazyk pro definici dat. Ten umožní správci databáze definovat strukturu databáze, příp. jednotlivých externích pohledů.

Aplikační programy se realizují pomocí jazyka pro manipulaci s daty a programovacího jazyka SŘBD. S daty z databáze pracují tak, že si nejprve načtou strukturu dat (tj. pro daný typ entity seznam atributů, jejich datových typů a umístění v záznamu) a pak s nimi provádějí potřebné operace. Aplikační programátor zná jen logickou strukturu databáze (název tabulek, jména a datové typy atributů), nebo dokonce jen tu její část, se kterou pracuje jeho aplikace. Přístup k datům řeší SŘBD sám na interní úrovni.

Případné náhodné dotazy, které neumí zodpovědět žádná funkce informačního systému, se řeší pomocí dotazovacího jazyka. Uživatel opět musí znát pouze názvy typů entit a jejich atributů, přístup k datům řeší opět SŘBD sám.

Shrnutí pojmů 2.3.

Třístupňová architektura databáze.

Datové modely. Konceptuální či logické modely. Záznamově orientované modely, relační a síťový datový model. Interní či fyzické datové modely.

Zabezpečení jednotlivých úrovní databáze.

Otázky 2.3.

1. Jaký je postup při budování databáze, ze kterých základních kroků se skládá ?
2. Co je datový model obecně?
3. Které typy datových modelů znáte a čím se liší?
4. Jak souvisí třístupňová architektura databáze s typy datových modelů?
5. Kdo nebo který HW či SW zabezpečuje jednotlivé úrovně databáze?

2.4. Databázové jazyky, nezávislost dat

❑ Databázové jazyky

Jazyky používané pro práci s databázovými systémy se dělí do několika typů. Většina SŘBD nepoužívá některý z klasických programovacích jazyků, ale má definován vlastní jazyk. V manuálech bývá prezentován jako seznam příkazů a funkcí daného systému, ovšem příkazy se dělí podle druhu své činnosti do několika skupin:

1. Jazyk pro definici dat (JDD) obsahuje příkazy, které umožňují (i interaktivně, bez psaní programu) definovat datové struktury, tedy definovat strukturu tabulky. Obsahuje příkazy pro

- seznam datových typů a datových struktur pro definici typu atributu,
- definice, modifikace a rušení typu entity,
- definice, modifikace a rušení typu vazby.

2. Jazyk pro manipulaci s daty (JMD) obsahuje příkazy, které umožňují pracovat s daty v tabulkách databáze. Obsahuje příkazy pro

- manipulace s atributy (ukládání a kontroly, ...)
- manipulace s entitami (ukládání, modifikace, rušení, výběry)
- manipulace s množinami entit (zobrazení, sjednocení, ...)
- manipulace s vazbami entit

3. Programovací jazyk pro zápis algoritmu může být

- v hostitelském jazyce (Cobol, C, Pascal, ...), pak jsou výše uvedené JDD a JDM vytvořeny jako procedury v hostitelském jazyce a celý SŘBD tvoří nadstavbu tohoto jazyka;
- vlastní jazyk SŘBD, obsahující (mimo příkazy JDD a JMD) programové struktury pro záznam algoritmů - příkazy pro větvení a cykly, pro komunikaci s uživatelem, pro formátování vstupů a výstupů, pro tvorbu menu a oken ap.

  Obvykle jazyky hostitelské, univerzální a tedy schopné realizovat i DB úlohy, jsou nazývány jazyky 3. generace 3GL. Na rozdíl od nich jazyky specifické pro jeden SŘBD, zvláště pak založené na standardním SQL přístupu k databázi, jsou nazývány jazyky 4. generace - 4GL. 4GL tedy není jediný konkrétní programovací jazyk, i když některé SŘBD svůj programovací jazyk takto nazývají.

4. Dotazovací jazyk, který podle typu dělíme do dvou skupin:

- procedurální, který popisuje způsob, **jak** data v databázi hledat, zapisuje algoritmy pro vyhledání informací
- deskriptivní, který zapisuje jen, **co** v databázi hledat, a to pomocí vlastností hledaných objektů.

❑ Nezávislost dat

Důležitou vlastností databázové technologie je nezávislost dat na aplikačních programech. Rozumíme tím možnost změnit definice dat na nižší úrovni abstrakce, aniž by se tím ovlivnila definice dat na vyšší úrovni. Jedná se o dvě úrovně nezávislosti dat:

- **fyzická nezávislost** dat umožňuje změnit fyzickou úroveň popisu dat, aniž by se musely měnit aplikační programy; někdy se touto změnou způsobu uložení dat na disku mění potřebná kapacita pro uložení souborů, někdy se toho využívá pro zvětšení výkonu celého systému.

Příklad 2.10. *Autoři SŘBD se rozhodnou v nové verzi pro radikální změnu formátu datových souborů z důvodu snížení počtu přenosů dat mezi diskem a pamětí a tím výrazného zkrácení odezvy uživateli.Data jsou nově zaznamenána jinak, to ale nesmí vyžadovat přepracování aplikačních programů.* ♦

Příklad 2.11. *zkušenost ukáže, že pro záznam jména stačí místo dosud používaných 40 znaků jen 30 znaků; předefinováním této položky se změní délka záznamu a samozřejmě i struktura souboru; aplikační programy však zůstanou beze změny.* ♦

- **logická nezávislost** dat umožňuje změnit konceptuální úroveň popisu dat, aniž by bylo třeba přepisovat aplikační programy. Při provozu DBS se často vyskytují dodatečné požadavky na změny či doplňky v logické struktuře dat, ty se musí promítnout i do databázového schématu.

  Příklad 2.12. *u souboru zaměstnanců je nutno zavést navíc evidenci o čísle havarijní pojistky auta, tedy přidat jeden atribut do typu entity; tím se změní jak logická, tak fyzická úroveň záznamu, ovšem aplikační programy dosud vytvořené budou pracovat beze změny; pro práci s novou položkou se vytvoří nové aplikační programy.* ♦

Shrnutí pojmů 2.4.

Databázové jazyky a jejich typy.
Jazyk pro definici dat – JDD.
Jazyk pro manipulaci s daty – JMD.
Programovací jazyk – hostitelský nebo vlastní.
Dotazovací jazyk – deskriptivní nebo procedurální.
Nezávislost dat na aplikačních programech. Fyzická a logická nezávislost.

Otázky 2.4.

1. Co je databázový jazyk a jaké jejich typy rozlišujeme ?
2. Co umožňuje JDD?
3. Co umožňuje JMD?
4. Jak se dají rozdělit programovací jazyky používané v databázové technologii?
5. Co jsou dotazovací jazyky a jak je dělíme?
6. Co znamená nezávislost dat na programech a jaké typy nezávislosti rozlišujeme?
7. Čím je nezávislost dat způsobena?

3. KONCEPTUÁLNÍ SCHÉMA DATABÁZE

Čas ke studiu: 3 hodiny

Cíl Po prostudování této kapitoly budete umět

- formulovat zadání informačního systému
- popsat metody zápisu konceptuálního modelu
- popsat nástroje logického modelu – lineární zápis, ERD, datový slovník, IO
- na jednoduchých úlohách z reality prakticky tyto metody použít a vytvořit datový model informačního systému

Výklad

3.1. Zadání informačního systému

❑ Životní cyklus vývoje IS

Informační systémy obdobně jako jiné programové systémy se budují podle nám již známých (ze SW inženýrství) postupů. Celý proces od rozhodnutí o budování systému až po ukončení jeho vývoje, jeho využívání a údržbu nazýváme **životním cyklem vývoje SW díla** – v našem případě informačního systému.

Informační systémy jsou jedním z druhů počítačových programových systémů a proto pro ně platí vše, co ze SW inženýrství známe. Protože však jde o speciální typ systémů, bude nutné některé etapy životního cyklu probrat mnohem detailněji, případně zavést řadu nových pojmů a technik.

Existuje více způsobů, jak rozložit vývoj informačního systému do etap, liší se obvykle jen různou mírou podrobnosti či přeléváním některých činností z jedné etapy do druhé. Často uváděný životní cyklus informačního systému nazýváme vodopádovým. Podrobně se jeho etapami budeme zabývat v navazujícím předmětu Databázové a informační systémy. Nyní probereme jen stručně jeho první 2 etapy – zadání a analýzu. Podrobně pak v následujících kapitolách probereme datovou analýzu, která je hlavní náplní tohoto předmětu.

❑ Zadání informačního systému

Na začátku existence informačního systému je rozhodnutí nějaké osoby využít počítače či počítačové sítě pro řešení nějakého reálného evidenčního problému. Této osobě budeme říkat **zadavatel** a poněkud zjednodušeně řekneme, že zadavatel formuluje požadavky na budoucí dílo. Při větších IS může být zadavatel reprezentován více lidmi, kde každý upřesňuje jinou část svých požadavků.

Zadání je vhodné požadovat písemně ve formě "odborného článku", tj. první verze zadání. To provádí osoba znalá oboru, která nemusí nic vědět o počítači a zadání obvykle formuluje slovně. Proto zadání bývá často nejednoznačné, neúplné, nepřesné. Protože cena za předělání programu po realizaci takového zadání by byla vysoká, je vhodné upřesňovat zadání již v této etapě. Upřesnění

provádí řešitel – informatik **analytik** dotazováním na další podrobnosti požadavků. Aby si vzájemně dobře zadavatel s analytikem rozuměli, používají se jednoduché formální metody pro záznam zadání a potom i výsledků analýzy, zadavateli přirozeně srozumitelné.

Požadavky na IS můžeme rozdělit na požadavky

- funkční, týkající se věcné náplně IS, co se má evidovat v databázi a co se má s daty dělat,
- nefunkční, koncepční, týkající se použitého HW, SW, organizace práce apod.

Je vhodné s tímto dělením předem seznámit zadavatele. Pokud nejsou dostatečně v zadání jednotlivé části formulovány, je nutné se na ně doptat.

❑ Funkční požadavky na IS

Funkčními požadavky rozumíme požadavky na **věcný, problémový obsah** systému. Zkušený analytik umí již při studiu zadání rozpoznat jeho neúplnosti nebo rozpory. Pak formou diskuze se zadavatelem musí tyto problémové části dořešit. Je vhodné, když si vypracuje jakousi **šablonu otázek**, podle níž se systematicky ptá, případně požádá zadavatele o dodržení úplnosti těchto informací. Má tak větší jistotu, že mu zadavatel uvede všechny informace o zadávaném IS.

Společně vytvořené modely jsou vhodné jako prostředek komunikace analytika se zadavatelem pro upřesňování a zpodrobňování zadání. Analytik tak bude mít dobrý základ pro následnou analýzu.

Základní struktura takové šablony je: **proč, k čemu, kdo, vstupy, výstupy, funkce, okolí**.

Pro datovou analýzu potřebujeme především zadání vstupů a výstupů, ty probereme podrobněji. Ostatní body budeme potřebovat v příštím semestru v DAIS.

PROČ vůbec je zapotřebí nový informační systém

Zdánlivě divná otázka obvykle uvede zadavatele do rozpaků. Úkolem je popsat okolnosti rozhodnutí o budování IS: popsat současný stav evidence, stručně vysvětlit, proč tento stav nevyhovuje, charakterizovat nové potřeby a představy o tom, jak má nový systém fungovat.

Může se i stát, že si zadavatel uvědomí, že nový systém není zapotřebí.

K ČEMU má systém sloužit

Opět zdánlivě divná otázka má dát odpověď na to, co je hlavní prioritou pro funkci systému: vnitřní evidence, vnější reprezentace informací apod.

Samozřejmě jeden systém může mít více priorit. Odpověď zadavatele pomáhá i jemu ujasnit si jejich existenci a pořadí. Tím odpovídá na otázku, které funkce bude nutné optimalizovat a které případně budou poněkud potlačeny - z hlediska rychlosti odezvy systému, z hlediska snadného ovládání systému, z hlediska bezpečnosti dat, bezpečnosti přístupu k funkcím apod.

KDO s ním bude pracovat - běžně, příležitostně, zřídka

Otázka souvisí bezprostředně s předcházející odpovědí: primární funkce systému budou sloužit hlavním uživatelům systému s nejčastějším přístupem. Případné další funkce mohou sloužit občasným uživatelům nebo i náhodným dotazům na informace z databáze.

Jaké budou VSTUPY do systému

Vstupy znamenají informace, které mají být v IS evidované. Je to seznam entit a jejich atributů. Jsou základem pro datovou analýzu, proto mají obsahovat skutečně podrobný výčet. Protože analytik nemusí být odborníkem v oblasti, pro niž je IS budován, jsou vhodné i komentáře vysvětlující odborné pojmy nebo zdůvodnění speciálních potřeb.

Příklad:

IS veřejné knihovny potřebuje evidovat: knihy (název knihy, autory, ISBN, přírůstkové číslo, vydavatele, rok vydání), čtenáře (jméno, adresa, telefon, číslo průkazu), výpůjčky knih (datum výpůjčky, datum vrácení, číslo čtenáře, přírůstkové číslo knihy).

Přírůstkové číslo je jednoznačné v rámci celé knihovny pro každý exemplář knihy, ISBN je číslo titulu pro jedno vydání knihy.

♦

Jaké budou VÝSTUPY ze systému

Výstupy znamenají všechny výstupní sestavy, které budou uživatelé potřebovat. Stačí název sestav a seznam informací na nich, lepší jsou načrtnuté ukázky nebo existující sestavy.

Příklad:

IS veřejné knihovny bude potřebovat: inventurní seznam knih (název knihy, autoři, ISBN, přírůstkové číslo), výpůjční lístek (jméno čtenáře, datum výpůjčky, název, autoři, ISBN, přírůstkové číslo), upomínku nevrácené výpůjčky (jméno a e-mailovou adresu čtenáře, název knihy, přírůstkové číslo) atd.

♦

Tyto informace jsou jednak určeny k analýze výstupních funkcí, jednak jako kontrola úplnosti zadaných vstupů. Může se totiž stát, že zadavatel požaduje na výstupu informace, které na vstupu nezadal, ani se nedají ze vstupů odvodit. Ty je pak nutné buď doplnit do vstupů, nebo zrušit požadované výstupy.

Jaké FUNKCE bude systém plnit

Tyto informace jsou jednak určeny k zadání všech potřebných funkcí IS, jednak jako kontrola úplnosti zadaných vstupů. Může se totiž stát, že zadavatel požaduje na výstupu informace, které na vstupu nezadal, ani se nedají ze vstupů odvodit.

I tyto informace jsou podkladem pro funkční analýzu. I zde slouží jako kontrola úplnosti zadaných vstupů. Opět se může stát, že zadavatel nezadal některé vstupní informace, které jsou pro správné fungování funkcí nezbytné. Ty pak analytik doplní do vstupů. Někdy se ukáže potřeba doplnit pomocné další atributy, které budou sloužit správné funkčnosti.

Jaké je OKOLÍ systému

Okolí systému znamená definovat všechny objekty (budeme je nazývat aktéry), kteří jsou zdrojem informací plynoucích do systému nebo cílem informací ze systému. Aktéry mohou být lidé, instituce nebo jiné SW systémy. Aktér není úplně totéž co uživatel.

❑ Nefunkční požadavky

Mimo věcnou náplň systému je nutné v zadání uvést řadu dalších informací o okolnostech řešení. Tyto další požadavky, zvané nefunkční, jsou omezení kladená na systémové služby. Jsou několika typů:

1. Požadavky na prioritní vlastnosti **výsledného programu**: efektivita (rychlost řešení, výkon = rychlost odezvy IS, paměťová náročnost), spolehlivost, přenositelnost do jiných SW prostředí, variabilita řešení pro různé uživatele.
2. Požadavky na **způsob řešení**: mohou předepisovat metody a použití standardů z oblasti metodiky vývoje, dokumentace, programovací jazyk, programovací a uživatelské prostředí. Celkem podmínky dodání, implementační požadavky a použití standardů.

3. **Vnější požadavky** ostatní nefunkční požadavky, jako cenová omezení, doba řešení a harmonogram, podmínky dodání, dodržení firemních standardů, omezení daná legislativou (platné zákony, vyhlášky), požadavky na spolupráci nového systému s již existujícími systémy, daná organizační struktura firmy, bezpečnost, atesty apod.

Tyto požadavky se nevyužijí při analýze systému (mimo případně předepsanou metodiku pro analýzu), tam se řeší jen věcná náplň. Nefunkční požadavky se zohledňují až v etapě návrhu implementace nebo při uzavírání smlouvy.

❑ Analýza informačního systému

Analýza nebo také specifikace problému znamená zpracování zadání do několika modelů budoucího systému, které mohou sloužit jako podklad pro implementaci. Pro analýzu je navržena řada metod. Podle toho, co analyzují je dělíme na metody pro

- datovou analýzu
- funkční analýzu
- dynamickou analýzu

Zatím se budeme věnovat jen datové analýze, ostatní analýzy i zbytek životního cyklu probereme v DAIS. Profesionální datová analýza vyžaduje zavedení řady nových pojmů a ty probereme v následující kapitole 4. Prozatím se naučíme zaznamenat její výsledek – konceptuální schéma databáze a datovou analýzu budeme provádět jen „intuitivně“. To znamená, že strukturu databáze budeme navrhovat tak, jak se nám bude jevit nejlépe pro požadované evidence objektů a jejich atributů.

3.2. Prostředky pro zápis konceptuálního modelu

❑ Logický popis databáze

Konceptuální model je výsledkem datové analýzy. Je to schematický model části reality, o níž se povede evidence v budovaném IS. Vytváří ho datový analytik na základě zadání zadavatele a budoucího uživatele. Ideální zadání by mělo být **úplné, jednoznačné, přesné, bezesporné**. Ovšem zadavatel většinou není profesionální analytik a neumí zadání takto formulovat. Proto i během analýzy musí zadavatel odpovídat na doplňující dotazy, doplňovat, upřesňovat i pozměňovat původní zadání. Je tedy nutné, aby rozuměl konceptuálnímu modelu, potvrzoval jeho správnost nebo odhaloval věcné chyby, neúplnosti v zadání ap. Proto konceptuální model musí používat jazyk, který bude dostatečně přesný pro modelování reality, jejích základních objektů a jejich atributů, vazeb mezi objekty a jejich vlastností.

Nejčastěji se pro záznam struktury databáze na konceptuální úrovni používá Chenův **E-R model**, používající kombinace textových formálních zápisů, grafického zobrazení typů entit, atributů a vztahů mezi entitami i doplňujících textových informací. Model E-R má podobný význam, jako vývojový diagram pro návrh algoritmu. Je určen pro návrh struktury databáze, pro popis hlavních vlastností dat v databázi uložených. Je nezávislý na implementaci a pro svou jednoduchou strukturu slouží jako společný jazyk uživatelů, zadavatelů a projektantů systému.

Někdy se už na konceptuální úrovni používá relační model dat, zvláště pokud se předpokládá budoucí implementace v tomto modelu. Existují i další prostředky pro záznam konceptuálního modelu.

S rozvojem objektové technologie byly vyvinuty i metody objektového datového modelování. Vychází z principů E-R diagramu, upravuje a rozšiřuje jeho možnosti o řadu přesněji popsaných integritních omezení a o pojmenování operací s daty.

Někdy je realita složitější, než mohou zachytit formální prostředky. Pro formulaci požadavků, které nejsou použitým modelem formalizovatelné, se používá přirozený jazyk - formou poznámek (integritních omezení), které doplňují formální schéma.

Konceptuální schéma je pro implementaci transformováno do databázového schématu a přitom se může ztratit část informací. Pak je potřeba důsledného doprogramování všech popsaných integritních omezení.

3.3. E-R model pro zápis konceptuálního schématu

Často používaným modelem pro popis logické struktury dat na konceptuální úrovni je E-R diagram (Entity-Relationship-Diagram, zkráceně **ERD**). Ten popisuje objekty a jejich vztahy buď textovým zápisem nebo pomocí E-R diagramů. Zvláště grafické zobrazení konceptuálního schématu je velmi názorné.

❑ Záznam atributů, entit, vztahů

1. Lineární textový zápis popisuje entity a vazby již známým způsobem

   Typ_entity (klíč, atrib1, atrib2, . . .)

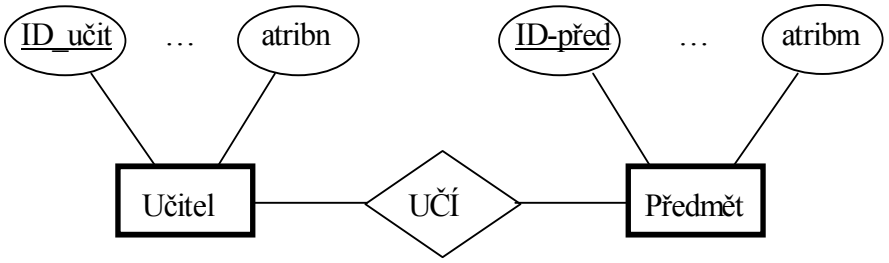
   TYP_VZTAHU (Typentity1, Typentity2, . . ., atrib1, atrib2, …)

Příklad 3.1. *E : Učitel (ID_učit, jméno, katedra, ...)*
Předmět (ID_před, název, ...)
V : UČÍ (Učitel, Předmět) ♦

2. Výhodný a nejčastěji používaný je typový E-R diagram, který graficky znázorňuje objekty a vztahy formou grafu, kde

   - uzel obdélníkový označuje entitní typ
   - uzel oválný označuje atributy; atributy jsou spojeny hranami s odpovídajícím entitním typem; klíčové atributy se podtrhnou
   - uzel kosočtvercový reprezentuje vztah mezi entitními typy, hranami je spojen s těmito entitními typy.

Příklad 3.2.



♦

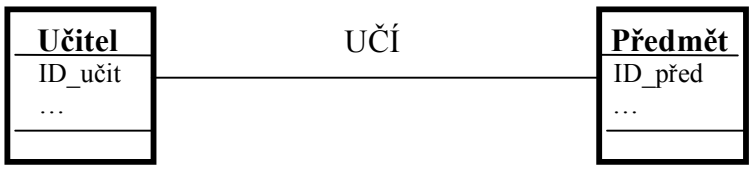
Často se používá stručnější varianta bez atributů, nebo i bez kosočtvercových uzlů a pak se název vztahu píše na hranu spojující entitní typy.

Příklad 3.3.

♦

3. Stále častěji se místo klasického ERD používá diagram objektový. Základní rozdíl je v tom, že obdélníkový uzel je rozdělen horizontálně na 2 – 3 části. V horní je opět název typu entity, střední obsahuje seznam atributů a dolní seznam operací definovaných nad tímto typem entity. Ve stručnější verzi se operace nebo i atributy vynechávají. Vztahy se zapisují na hranu.

Příklad 3.4.



nebo jen

♦

❑ Integritní omezení

Integritní omezení (IO) jsou logická omezení na typy a hodnoty atributů, entit a vazeb tak, aby schéma konceptuální co nejpřesněji odpovídalo zobrazované realitě. Zadávají se graficky v ERD nebo popisem v datovém slovníku nebo doplňujícím textem formou poznámky za datovým slovníkem.

❑ IO týkající se atributů

1. Základní specifikace atributů

Atributy entit jsme zadávali jen pomocí jejich názvů bez dalších podrobností. Aby typy entit byly zadány úplně, je zapotřebí ke každému atributu určit ještě několik doplňujících údajů. Všechny je uspořádáme do popisné tabulky atributů – datového slovníku (Data Dictionary, DD):

- jméno atributu jako identifikátor
- syntaktický typ atributu, jeho doménu
- popis a význam atributu
- velikost a formát vnější reprezentace atributu
- množinu operací, které lze nad jeho hodnotami provádět
- KEY - příznak, zda je atribut klíčový, zda je součástí primárního klíče
- NULL - zda je přípustné, aby měl atribut hodnotu nevyplněnu či je zadání hodnoty povinné
- formou poznámky další IO plynoucí z reality, která ve výše uvedených informacích nejsou uvedena (předdefinovaná hodnota, výpočet, popis kontroly ap.)
- případně další systémové informace (indexování ap.)

Příklad 3.5. Zam (rod_cis, jméno, funkce, plat, nástup, řidič)

iden_atrib	dat_typ	délka	KEY	NULL	…	IDX	IO	význam
rod_cis	num	10,4	A	N			*1)	
jméno	char	30	N	N				příjmení křestní, titul
funkce	char	10	N	A				
plat	num	8,2	N	A				
nástup	date		N	N				
řidič	logic		N	A				

*1) tvar 1122334444, kde 11 = ...

♦

2. Neatomické atributy

Konceptuální model nemusí být omezen jen na použití atomických (dále nedělitelných) atributů. Může používat složitějších struktur vlastností objektů. Pokud použitý SŘBD umožňuje realizovat jen atomické atributy, je nutno konceptuální schéma transformovat a složitější struktury převést na atomické. Neatomické atributy jsou dvojího druhu:

- Skupinové atributy. Jejich struktura nemusí být jednoúrovňová, ale může vytvářet obecně celou hierarchii. Skupinový atribut vznikne složením jeho složek. V obecných úvahách je užitečné užívat skupinové atributy, pokud potřebujeme někdy popisovat celou skupinu, jindy její jednotlivé složky. Lineární zápis může vypadat např.

  Zam (…, adresa (ulice, město, PSČ, stát), …)

  Transformace na atomické atributy se obvykle provede odstraněním názvu skupinového atributu a ponecháním jen atributů vnitřních. Potřebujeme-li pak pracovat se skupinovou položkou, musíme vyjmenovat celý seznam jejích prvků.

  Zam (…, ulice, město, PSČ, stát, …)

- Vícehodnotové atributy jsou opakující se stejné položky, tedy jsou představovány vektorem hodnot pevné nebo proměnné délky. Obvykle se v obecném konceptuálním schématu zaznamenávají jako vícehodnotové

  Zam (. . ., dítě (jméno, d_rod_cis): multi, . . ., plat (leden, únor, …, prosinec), …)

  a později se transformují na atomické atributy buď zrušením názvu multiatributu a opakováním složek atributu pro známý počet opakování, nebo záznamem multiatributu do samostatné tabulky při neznámém počtu opakování. O realizaci vazby mezi oběma tabulkami si řekneme níže.

  Zam (rod_cis, . . ., leden, únor, …, prosinec, …)
  Dítě (rod_cis, jméno, d_rod_cis)

3. ISA hierarchie

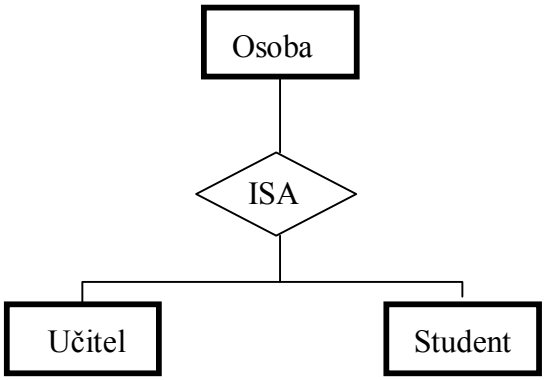
Někdy se mezi sledovanými objekty vyskytují objekty stejného typu, popsané řadou stejných atributů a lišících se jen v některých atributech. Jestliže při většině manipulací s daty obou typů entit se provádějí akce nad společnými údaji, bude vhodné definovat společný typ entity. Ten bude mít atributy společné, příznak rozlišení obou typů a pro každý typ pak rozdílné speciální atributy.

Pokud je většina atributů rozdílných nebo většina akcí nad nimi rozdílných, je vhodnější definovat pro každý typ objektu samostatný typ entity.

Takovému vztahu se říká ISA hierarchie a rozhodnutí o vyřešení této situace patří k datové analýze. Graficky se znázorňuje ISA hierarchie podle následujícího příkladu.

Příklad 3.6.

V databázi potřebujeme evidovat učitele a studenty. Pro obě skupiny potřebujeme řadu osobních údajů, o učitelích pak příslušnost ke katedře, funkci a plat, o studentech příslušnost k fakultě, ročníku a oboru. Tedy všechny osoby jsou buď učitelé nebo studenti. Zakreslíme:



Pro realizaci můžeme navrhnout buď jeden typ entity Osoba *vždy s některými atributy nevyužitými, nebo dva* Učitel a Student. *Zřejmě záleží na počtu shodných a rozdílných atributů, případně na operacích, které nad nimi budeme provádět.*

♦

Rozhodnutí o realizaci je součástí datové analýzy, je třeba zvážit výhody a nevýhody obou řešení.

IO týkající se vlastností vztahů mezi entitami.

4. Kardinalita vztahu

je důležitým IO, protože z něj plyne řada důsledků jak v datové, tak funkční analýze. Máme-li definován vztah typů entit E1 a E2, může tento binární vztah mít jeden ze tří poměrů:

1:1 jedné e1 ∈ E1 odpovídá ve vztahu nejvýše jedna e2 ∈ E2 a naopak, jedné e2 ∈ E2 odpovídá nejvýše jedna e1 ∈ E1;

1:N jedné e1 ∈ E1 odpovídá ve vztahu obecně několik e2 ∈ E2, ale jedna e2 ∈ E2 má vztah pouze k jedné entitě e1 ∈ E1;

M:N jedné e1 ∈ E1 odpovídá ve vztahu obecně několik entit e2 ∈ E2 a naopak jedna e2 ∈ E2 má vztah k několika entitám e1 ∈ E1.

Unární vztahy obvykle převádíme na binární tak, že zaznamenáme dvě kopie E1 a E2 základní entity E. Kardinalitu unární vazby tak určíme obdobně jako u binární vazby.

N-ární vazby pak mohou mít poměry 1:1:1, 1:M:1, 1:M:N, M:N:K, . . .

Kardinalitu vztahu zaznamenáváme do ERD vypsáním nebo vykreslením.

Příklad 3.7. *Každý učitel může učit víc předmětů, každý předmět může být učen více učiteli.*

nebo

♦

Povinnost členství ve vztahu

Do některých vztahů musí vstupovat každá entita množiny entit, do jiného vztahu ne, záleží na modelované realitě. Definujeme tedy dva druhy členství ve vztahu:

- povinné (obligatorní)
- nepovinné (fakultativní)

Přitom může mít jedna entita povinné členství, druhá nepovinné.

Typ povinnosti členství zaznamenáváme pomocí lineárního zápisu také

VZTAH (E1:(min, max), E2:(min, max))

kde min je hodnota 0 (nepovinné) nebo 1 (povinné), max je hodnota 1 nebo M dle kardinality vztahu.

Graficky se povinnost členství ve vztahu zaznamenává v E-R diagramu kolečkem plným na straně příslušného entitního typu, nepovinnost kolečkem prázdným.

Příklad 3.8. *Učitel musí učit, předmět nemusí mít definován učitele.*

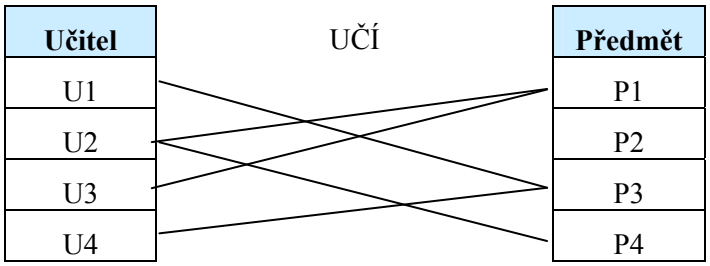
♦

Povinnost členství ve vztahu je důležité IO. Vyjadřuje, že entita jednoho typu nemůže existovat bez zapojení do vztahu s entitou druhého typu.

Výskytový diagram

Dobrý způsob záznamu vazeb mezi entitami je pomocí tzv. výskytového diagramu, který zobrazuje jednotlivé výskyty entit a jejich vztahů

Příklad 3.9. *Učitel musí vyučovat, může učit více předmětů, předmět nemusí mít definovaného učitele nebo jej může učit více učitelů.*



♦

Tento způsob zobrazení vztahů je vhodné používat jako pomocný při ujasňování kardinality vztahu a povinnosti členství ve vztahu.

Slabé entitní typy

Součástí klíče některých etitních typů nemusí být jen jejich vlastní atributy. Někdy nejsou dvě entity rozlišitelné pomocí svých atributů, jsou rozlišitelné až pomocí toho, že jsou povinně ve vztahu k entitě jiného typu. Pak takový typ entity nazýváme slabý entitní typ. Příslušný druhý typ entity pak nazýváme identifikačním vlastníkem a typ vztahu identifikačním vztahem. Slabý entitní typ má vždy povinné členství v identifikačním vztahu, opačně to nemusí platit.

Graficky v ERD se slabý entitní typ znázorňuje dvojitým obdélníkem, identifikační vztah dvojitým kosočtvercem.

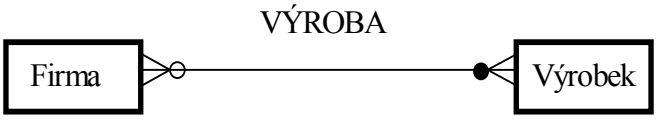
Příklad 3.10. *Diplomová práce má název, ale různí učitelé mohou nazvat svá témata stejně. Teprve přiřazením tématu k zadavateli je práce určena jednoznačně.*

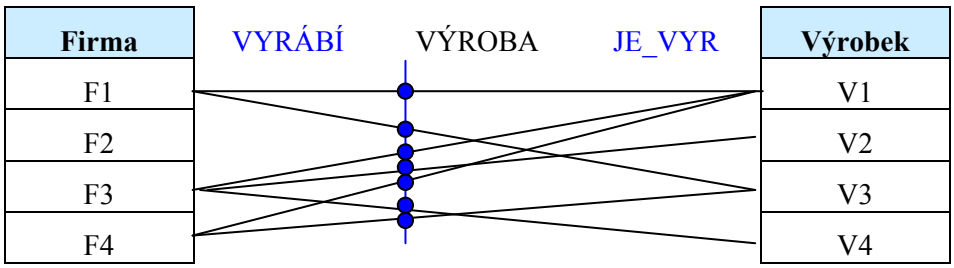
♦

Dekompozice vztahu M:N

Konceptuální schéma je nezávislé na následně použitém SŘBD a proto na logické úrovni je dostačující definovat existující vztahy typu M:N. Ovšem řada SŘBD neumí přímo vyjádřit vztahy M:N, umí pouze 1:N. I z jiných důvodů bývá vhodné rozložit všechny vztahy M:N na dva vztahy 1:N. Jak, to nám naznačí následující výskytový diagram pro vztah

Příklad 3.11.





Z obrázku je patrno, že je nutno evidovat platné dvojice, přičemž firma i výrobek se opakují. Celá dvojice popisuje vztah.

♦

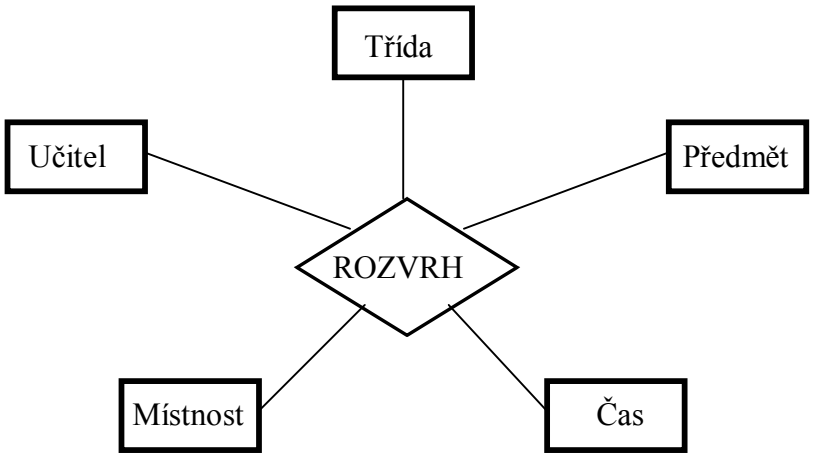
Vidíme, že vazba typu M:N se transformuje přidáním tzv. vazební entity, která má s původními entitami vztahy 1:M a N:1, obě s povinným členstvím na straně vazební tabulky.

Obdobně se pomocí vazební entity řeší vztahy unární s kardinalitou M:N.

Kardinalita a dekompozice n-árních vztahů

Vztahy n-ární pro n>2 s kardinalitou M:N: … se transformují opět pomocí vazební entity na vztahy s kardinalitou nejvýše 1:M.

Příklad 3.12. *Úplný rozvrh hodin je vztahem mezi entitními typy Učitel, Předmět, Třída, Místnost, Čas.*



Množiny základních pěti entit obsahují seznamy učitelů, tříd, předmětů, místností a vyučovacích hodin, vazební entita ROZVRH obsahuje ty pětice, které tvoří skutečný reálný rozvrh.

♦

❑ Úplný konceptuální model struktury databáze

Výsledkem datové analýzy je konceptuální schéma struktury databáze. Úplné schéma tedy obsahuje:

- lineární zápis seznamu typů entit s jejich atributy a lineární zápis seznamu vztahových entit,
- úplný grafický tvar ERD ve dvou úrovních
  - konceptuální schéma
  - transformovaný pro databázové schéma
- datový slovník,
- seznam dalších IO týkajících se entit, atributů a vztahů.

Poznámka: dosud jsme návrh entitních typů, jejich atributů a vazeb prováděli intuitivně. Nejednoznačné situace se však někdy nepodaří vyřešit optimálně a dosud neznáme pravidla, podle kterých rozpoznáme případné chyby. V kapitole o relačním datovém modelu se k datové analýze znovu vrátíme. Pomocí jeho nástrojů a pravidel se naučíme analyzovat strukturu databáze systematicky.

Z této kapitoly si odnášíme především zápis a zobrazení výsledku datové analýzy.

Příklad 3.13.

*Navrhněte strukturu databáze pro informační systém **ABC** soukromého zdravotnického střediska s několika lékaři. Je potřeba evidovat lékaře (osobní údaje a specializaci), pacienty (osobní údaje, pojišťovna), objednané pacienty a uskutečněné návštěvy u lékaře i lékařů u pacientů (datum a čas,diagnóza,cena pro pojišťovnu).*

Řešení:

Lineární zápis: Lékař (RČ_L, jméno_L, specializace)
Pacient (RČ_P, jméno_P, adresa, pojišťovna)
Návštěva (RČ_L, RČ_P, datum, čas, diagnóza, cena)

Poznámky:

- *entita návštěva je vlastně vazební entita mezi lékařem a pacientem s vlastními vazebními atributy (= atributy týkajícími se vazby, ne původních entit)*
- *Objednávky se zapisují přímo do tabulky Návštěva bez diagnózy a ceny, po uskutečnění návštěvy se tyto atributy doplní. Neuskutečněné návštěvy se poznají porovnáním data návštěvy s dnešním datem.*

ERD: zde jen jedna varianta

Datový slovník + IO:

entita	atribut	dat_typ	délka	KEY	NULL	IDX	IO	význam
Lékař	RČ_L	num	10,4	A	N	A	*1)	
	jméno_L	char	30	N	N	A		příjmení křestní, titul
	specializace	char	10	N	N	N		
Pacient	RČ_P	num	10,4	A	N	A	*1)	
	jméno_P	char	30	N	N	A		příjmení křestní, titul
	adresa	char	60	N	A	N		
	pojišťovna	num	3	N	N	N		
Návštěva	RČ_L	num	10,4	A	N	A		přenos z Lékař
	RČ_P	num	10,4	A	N	A		přenos z Pacient
	datum	date	8	A	N	N		
	čas	time	5	A	N	N		
	diagnóza	char	60	N	A	N		
	cena	num	10,2	N	A	N		

*1) tvar 1122334444, kde …

Shrnutí pojmů 3.

Konceptuální schéma, logický popis databáze, Chenův ERD.
Zápis a zobrazení typu entity, atributů, typu vztahu.
Zápis integritních omezení, základní specifikace atributů, datový slovník.
Řešení neatomických atributů, ISA hierarchie.
IO mezi entitami, kardinalita vztahu, povinnost členství ve vztahu, výskytový diagram, slabí entitní typy, dekompozice vztahu M:N. n-ární vztahy, jejich kardinalita a dekompozice.
Úplný konceptuální model struktury databáze.

Otázky 3.

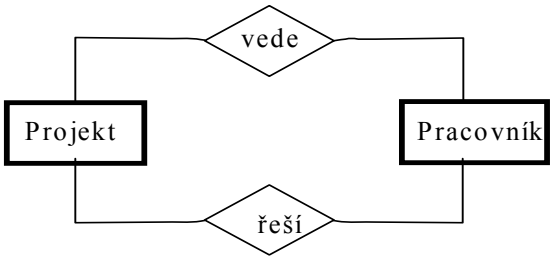
1. Co je konceptuální schéma databáze a jak se zobrazuje?
2. Co je lineární zápis entit a vazeb?
3. Co je ERD a jak souvisí s konceptuálním schématem?
4. Která integritní omezení se zobrazují do ERD a jak?
5. Jak se vyjádří integritní omezení, která se nezakreslují do EDR?
6. Co je datový slovník a k čemu slouží?
7. Co vše obsahuje úplný konceptuální model databáze?

Úlohy k řešení 3.

1. ATRIBUTY
   Zamyslete se nad tím, zda je vhodné rozčlenit dané atributy, za jakých okolností a jak:
   a) Úplné jméno + tituly
   b) Adresa
   c) Rodné číslo

2. ENTITY
   Je možné ztotožnit entitu (instanci typu entity) se souhrnem hodnot jejích atributů ?

3. VZTAHY
   Nakreslete výskytový E-R diagram pro entity Projekt(název, …) a Pracovník(jméno, ...) dle zadaného E-R diagramu:



4. Pro každou dvojici verbálních pravidel určete dva entitní typy a jeden typ vztahu. Ke každému stanovte kardinalitu a povinnost členství ve vztahu. Vyjádřete E-R diagramem.
   a) Oddělení zaměstnává libovolné množství osob, osoba je zaměstnána maximálně v jednom oddělení.
   b) Vedoucí řídí maximálně jedno oddělení, oddělení má maximálně 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.
   f) Objednávka zboží může být na více výrobků, výrobek se může objevit na více objednávkách.

g) Zákazník může předložit řadu objednávek, každá objednávka je právě od jednoho zákazníka.

5. ČLENSTVÍ VE VZTAHU

Stanovte vhodné typy členství pro entitní typy v následujících případech. Nakreslete E-R diagramy
a vztahové diagramy.

Entitní typy:	Vztahy:
a) Dům, Osoba	VLASTNICTVÍ
b) Dům, Nájemník	OBÝVÁ
c) Dům, Osoba	OBÝVÁ
d) Objednávka, Položka_objednávky	OBSAHUJE
e) Zákazník_banky, Bankovní_účet	MÁ_PŘIDĚLEN
f) Zaměstnanec, Kvalifikace	MÁ

6. Jsou následující tvrzení pravdivá ?

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 mezi X a Z.

7. DOMY

Navrhněte schéma, které zaznamená atributy: adresa domu, počet bytů v domě, jméno majitele domu,jméno nájemníka bytu. Schéma má vystihovat : která osoba je majitelem daného domu, které osoby bydlí v daném domě. Majitel domu nemusí bydlet ve vlastním domě. Nakreslete E-R graf schématu.

8. ZAMĚSTNANCI

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 a až 12 měsíčních výplat, které představují částky vyplacené v posledních 12 měsících. Zaměstnanec mohl získat více kvalifikačních stupňů.

Definujte entitní typ ZAMĚSTNANEC

a) připouští-li model vícehodnotové atributy,

b) nepřipouští-li model vícehodnotové atributy.

9. REKVALIFIKAČNÍ KURZY

Frekventanti rekvalifikačních kurzů jsou rozděleni do studijních skupin. Každou skupinu může učit několik učitelů, každý učitel může učit více skupin. Jedna skupina používá vždy stejnou učebnu, ovšem více skupin může používat stejnou učebnu v různých časech. Navrhněte E-R diagram, entitní typy a existující vztahové typy tak, aby všechny vztahy byly jen typu 1:N a všechny položky byly atomické.

10. JEDNODUCHÁ UNIVERZITA

Uvažme jednu univerzitu 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 funkční zařazení. Zaměstnanci vypisují přednášky, ne každý však musí vypsat v daném roce přednášku. Přednášky

jsou dány v rámci fakulty kódem, mohou mít stejné názvy, konají se v daný den a hodinu v dané místnosti. Studenti se zapisují na přednášky a vykonávají z nich zkoušky s daným hodnocením.

a) Navrhněte E-R diagram s odpovídajícími IO. Další explicitní IO zapište v přirozeném jazyce.

b) Rozeberte případy, kdy je vhodné uvažovat katedru jako entitní typ a kdy jako atribut.

c) Uvažujte funkce profesor, docent a odborný asistent. Profesor může zaměstnávat studenty jako pomocné vědecké síly na řešení projektů, docent nebo profesor může být vedoucím projektu. Projekt je dán číslem a názvem.

Zadání projektu

Z reálného světa si zvolte vlastní zadání pro malý informační systém s evidencí několika základních entit (asi 5), mezi nimiž se vyskytují vazby. Zadání konzultujte se svým cvičícím či tutorem. Zvolte si jiné zadání, než se vyskytuje v příkladech této učebnice.

Podle odsouhlaseného zadání zpracujte nejprve intuitivně navržený konceptuální datový model.

Na konci kapitoly 4. budete v projektu pokračovat.

Výsledný projekt odevzdejte ve formátu DOC nebo PDF prostřednictvím podatelny na internetové adrese barborka.vsb.cz/podatelna/

4. RELAČNÍ DATOVÝ MODEL

Cíl Po prostudování celé kapitoly budete umět

- definovat relaci, relační schéma a schéma relační databáze, rozumět definicím a na příkladech vysvětlit jednotlivé pojmy,
- vyhledávat informace v relační databázi pomocí procedurálního jazyka – relační algebry i pomocí deskriptivního jazyka SQL,
- definovat pojem funkční závislost mezi podmnožinami atributů, rozumět mu a bezpečně rozeznávat funkční závislosti v praktických úlohách,
- definovat pomocí normálních forem i vysvětlit na příkladech, co je dobře navržené relační schéma a na základě toho navrhnout optimální strukturu relační databáze.

4.1. Relační schéma, relace

Čas ke studiu: 1 hodina

Cíl Po prostudování tohoto odstavce budete umět

definovat a vysvětlit na příkladech základní pojmy relačního modelu

Výklad

Historie relačního modelu

Historicky prvním datovým modelem v databázové technologii byl model hierarchický a pak jeho zobecnění, model síťový. Jeho standard byl definován v roce 1970 a týkal se především implementace síťového SŘBD, příkazů pro definice dat, způsobu práce a příkazů pro manipulace s daty. Měl výhody i nevýhody. Výhodou byla především rychlost realizace vazeb mezi entitami, nevýhodou především nutnost profesionálního návrhu struktury databáze. Změny struktury databáze znamenaly velkou práci.

Nový datový model – relační (RDM), založený na teoretických matematických základech, vznikl jen o málo později. Poprvé byl zveřejněn v roce 1971 v článku pracovníka firmy IBM, matematika Dr. Codda. V něm byl definován teoreticky relační model jako matematické zobecnění pojmu soubor pomocí pojmu matematické relace. Dr. Codd uvedl matematické definice pojmů (nám již známých z konceptuálního modelu) entita a její typ, vazba a její typ, atribut a jeho hodnoty, struktura databáze. Definoval jazyk, pomocí něhož se vyhledávají informace a manipuluje s daty v databázi. Konečně zavedl pojem funkčních závislostí mezi množinami atributů, které ovlivňují správnost struktur entit a definoval pravidla pro rozpoznání správné struktury databáze.

Protože pravidlům pro RDM bychom zatím nerozuměli, probereme je na konci této kapitoly, až se seznámíme s pojmy a teorií RDM.

❑ Základní pojmy relačního datového modelu

Definice 4.1.

Relační schéma R je výraz tvaru **R**(A, f), kde **R** je jméno schématu, A = {A_1, A_2,..., A_n} je konečná množina jmen atributů, f je zobrazení přiřazující každému jménu atributu A_i neprázdnou množinu (obor hodnot atributu), kterou nazýváme doménou atributu D_i, tedy $f(A_i) = D_i$.

Definice 4.2.

Relace R s relačním schématem **R** je konečná podmnožina kartézského součinu domén Di, příslušejících jednotlivým atributům Ai, tedy

$$R \subset D_1 \times D_2 \times \ldots \times D_n.$$

Číslo n nazýváme stupněm relace, o relaci R říkáme, že je typu **R** nebo že je instancí relačního schématu **R**.

Nové pojmy relační schéma a relace definují nám již známé pojmy typ entity (zobrazený jako název a hlavička datové tabulky) a množina entit (zobrazená jako obsah tabulky, množina jejích řádků), které jsme dosud používali v konceptuálním modelu. Tyto přesné definice jsou nutné k definování jazyka pro manipulace s databází a pro definici pravidel, které musí splňovat správně navržená databáze bez redundancí.

Relaci je výhodné znázorňovat jako dvourozměrnou tabulku, kde každý řádek odpovídá jedné entitě, každý sloupec jednomu atributu. Jména atributů musí být v rámci relace navzájem různá. Často se tedy v relačním modelu místo pojmu relace používá pojem tabulka.

Příklad 4.1.

Jméno relačního schématu: **`Učitel`**

Jména atributů: `ČU,jméno,funkce,plat`

Domény: D1 je množina řetězců tvaru Un, kde n je přirozené číslo; prvky množiny jsou osobní čísla učitelů;
D2 je množina možných jmen učitelů;
D3 množina textových řetězců označujících funkční zařazení učitelů v zaměstnání;
D4 množina nezáporných celých čísel<10000;

Zobrazení: `f(ČU)=D1, f(jméno)=D2, ...`

Relační schéma: **`Učitel`**`(ČU,jméno,funkce,plat)`

Relace: `Učitel={(U12,Čáp,docent,4000),(U27,Holub,asistent,3000),`
`(U39,Kos,asistent,3000),(U43,Orel,docent,4000), …}`

Zobrazení relace tabulkou

Učitel

ČU	jméno	fce	plat
U12	Čáp Ladislav	docent	4000
U27	Holub Stanislav	asistent	3000
U39	Kos Zbyněk	asistent	3000
U43	Orel David	docent	4000
…			

♦

Protože relace odráží část reálné skutečnosti odpovídající aplikačním úlohám, není prvkem relace jakýkoli prvek z $D_1 \times D_2 \times \ldots \times D_n$, ale jen prvky vyhovující zadaným podmínkám vyjádřeným jako integritní omezení (tentokrát IO na objekty = které objekty z reality mají být do relace zahrnuty).

Příklad 4.2.

Pro schéma **Učitel** (ČU, jméno, fce, plat) *je relace* Učitel ⊂ ČU x jméno x fce x plat

Učitel

ČU	jméno	fce	plat
U1	Anděl Jiří	asistent	1000
U1	Anděl Jiří	asistent	2000
U1	Anděl Jiří	asistent	3000
…			
U128	Anděl Jiří	profesor	3000
…			
U2	Beneš Vladimír	asistent	1000
U2	Beneš Vladimír	asistent	2000
U2	Beneš Vladimír	asistent	3000
…			
U128	Žižka Jan	asistent	1000
…			
U128	Žižka Jan	profesor	3000

Kartézský součin obsahuje všechny kombinace hodnot všech domén. Z nich jen některé patří do relace = podmnožiny kartézského součinu. Jsou to ty prvky (řádky), které odrážejí evidovanou skutečnost formulovanou integritním omezením o prvcích relace. To znamená ty, které přísluší skutečným učitelům naší evidence (Žižka Jan je profesorem s osobním číslem U128 a platem 3000).

♦

Poznámka: Některé učebnice a zvláště manuály některých SŘBD používají pojem relace pro označení vazby mezi entitami, ne pro relaci (tabulku) výše definovanou. Jde o **chybný překlad** původního slova „Relationship“ odpovídajícího relační vazbě. I když bychom mohli používat místo vazba relací pojem „relace relací“, z pochopitelných důvodů tento složitý pojem zavádět nebudeme. RDM se nazývá relační podle definice relace ~ tabulky, ne podle relační vazby.

♦

Na rozdíl od matematických relací jsou databázové relace proměnné v čase. Aktualizace databáze, která umožňuje zachytit v databázi změny nastávající v reálném světě, spočívá ve změně aktuálních relací přidáváním, rušením prvků relací nebo změnou hodnot některých atributů.

Z definice relace vyplývají tyto jejich tabulkové vlastnosti

- homogenita sloupců (hodnoty odpovídají příslušné doméně)
- každý údaj (hodnota atributu ve sloupci) je atomickou položkou,
- na pořadí řádků nezáleží (relace je množina)
- na pořadí sloupců nezáleží (pojmenované atributy také tvoří množinu)
- každý řádek tabulky je jednoznačně identifikovatelný hodnotami jednoho nebo několika atributů (primárního klíče).

Definice 4.3.

Schéma relační databáze je konečná množina relačních schémat $\{\mathbf{R_1}(A_1,f_1), \mathbf{R_2}(A_2,f_2), \mathbf{...}, \mathbf{R_m}(A_m,f_m)\}$.

Definice 4.4.

Relační databází v daném časovém okamžiku je konečná množina relací $R_1, R_2, ..., R_m$, tzv. aktuálních relací, kde R_i je typu $\mathbf{R_i}$.

Definice 4.5.

O množině relací $R_1, ..., R_k$ vyhovujících všem daným IO řekneme, že je konzistentní.

❑ Realizace vazeb v relačním modelu

Charakteristickou vlastností relačního modelu je realizace vazeb opět pomocí relací ~ tabulek. Již v konceptuálním modelu jsme některé entity nazývali vazebními, pokud reprezentovaly vazbu mezi dvěma či více entitami. Vzpomeňme příklad binární vazby mezi Firmou a Výrobkem, nazvanou VÝROBA(Firma, Výrobek) nebo n-ární vazbu ROZVRH(Třída, Předmět, Učitel, Místnost, Hodina).

Relační model obecně **realizuje vazbu pomocí relace**, v níž každou entitu do vazby vstupující zastupuje její primární klíč.

V některých případech (vazby 1:1, 1:M) však není nutné použít samostatnou vazební tabulku, ale stačí doplnění tabulky na straně M o další atribut - **cizí klíč**, jak si ukážeme na příkladech.

Příklad 4.3.

Binární vazbu kardinality M:N relací Učitel(ČU, jméno, fce, plat) *a* Předmět(ČP, název), *nazvanou* UČÍ(Předmět, Učitel) *realizujeme pomocí vazební relace s klíči obou relací*

Učí (ČU, ČP)

Pokud vazba má vlastní atributy, týkající se ne objektů učitel a předmět, ale vazby samotné – například počet hodin, který učí konkrétní učitel v konkrétním předmětu, pak jsou tyto atributy přidány do vazební relace:

Učí (ČU, ČP, hodin)

Když začne učitel učit nový předmět, přidá se do Učí nový řádek, při změně učitele učícího matematiku se modifikuje v příslušném řádku ČU, pokud ze zruší předmět, smažou se řádky s příslušným ČP.

♦

Příklad 4.4.

Binární vazbu 1:M relací Učitel *a* Katedra(ČK, název_kat) *je opět možno realizovat vazbu* ČLEN_KAT (Učitel, Katedra) *pomocí vazební relace s atributy*

Člen_kat (ČU, ČK)

Ovšem při podrobnější analýze zjistíme, že počet řádků této relace je stejný, jako relace Učitel – každému učiteli odpovídá jeden řádek s číslem katedry. V tomto případě je možné vazební tabulku spojit s tabulkou Učitel, shodné sloupce Učitel.ČU *a* Učí.ČU *sloučit, takže vznikne rozšířená tabulka*

Učitel (ČU, jméno, fce, plat, ČK)

V ní atribut ČK, který realizuje vazbu Učí, nazýváme cizím nebo vazebním klíčem.

♦

Příklad 4.5.

Vazba n-ární se realizuje opět pomocí vazební relace. Případ ROZVRH(Třída, Předmět, Učitel, Místnost, Hodina) by realizovala tabulka

Rozvrh (ČT, ČP, ČU, ČM, Hod)

kde ČT je číslo třídy, ČM číslo místnosti, Hod je hodina v týdnu.

♦

Celkově můžeme konstatovat, že vazba mezi relacemi se obecně vždy dá realizovat pomocí vazební tabulky. V některých případech je možné vazební tabulku sloučit s jednou z tabulek do vazby vstupujících a spojit dva totožné sloupce. Nově připojené sloupce k původní tabulce nazýváme cizí klíče. Tuto technologii s cizími klíči používáme, pokud již při datové analýze zjistíme, že některý typ vazby bude potřebný pro náš informační systém.

Velká výhoda relačního modelu je však také v tom, že i předem nepředpokládané vazby můžeme realizovat bez zásahu do původní struktury relací. Pokud se až později ukáže potřeba evidence nového, nepředpokládaného typu vazby mezi existujícími entitami, stačí definovat novou vazební relaci. Jejími atributy budou primární klíče provázaných relací, případně další atributy vazby. S novou relací se pak pracuje pomocí běžných příkazů práce s relacemi.

CD-ROM

Na CD-ROMu s tímto výukovým textem jsou animované příklady na právě probrané případy realizace vazeb v RMD. Soubory s animacemi jsou pojmenované:

- Animace\relacni model\vazba_1k1.exe
- Animace\relacni model\vazba_1kN.exe
- Animace\relacni model\vazba_MkN.exe
- Animace\relacni model\vazba_unarni.exe
- Animace\relacni model\vazba_ternarni.exe

Shrnutí pojmů 4.1.

Relační schéma, relace. Atribut, doména. Zobrazení množiny atributů do množiny domén. Stupeň relace. Zobrazení relačního schématu a relace pomocí tabulky.
Souvislost relačního schématu a relace s typem entity a entitami.
Schéma relační databáze, relační databáze. Aktuální relace. Konzistence databáze.
Realizace vazeb různých typů v RMD.

Otázky 4.1.

1. Definujte relační schéma a na vlastním příkladě vysvětlete všechny pojmy v definici použité.
2. Definujte relaci a na praktickém příkladu vysvětlete všechny pojmy v definici použité: doména, kartézský součin domén, podmnožina kartézského součinu, integritní omezení pro prvky relace.
3. Definujte schéma relační databáze a na vlastním příkladě popište jednoduché schéma databáze z praxe.
4. Definujte relační databázi a uveďte příklad některé reálné databáze z praxe.
5. Co znamená, že je databáze konzistentní?
6. Co znamená, že je databáze integritní?
7. Jakými způsoby se realizují vazby entit v relačním datovém modelu? Rozlište realizaci vazby dle počtu entit do vazby vstupujících (1, 2, více) a podle kardinality vazby.

Úlohy k řešení 4.1.

1. Projděte si znovu příklady 4.3. – 4.5. na realizaci vazeb v této kapitole. Ke každému příkladu zobrazte formou tabulek: původní relace, vazební relaci a pokud to jde, sloučení vazební relace s jednou z původních a vznik cizího klíče.

4.2. Relační jazyky pro vyhledávání informací

Čas ke studiu: 6 hodin

Cíl Po prostudování této kapitoly budete umět

- Definovat operace relační algebry a pomocí nich napsat výraz pro vyhledání libovolné informace z databáze
- Definovat relační kalkuly
- Pomocí deskriptivních dotazovacích jazyků SQL a QBE zapsat dotaz pro vyhledání libovolné informace z databáze.

Výklad

❑ Rozdělení relačních dotazovacích jazyků

Protože skutečnost, jejíž obraz zachycuje databáze, je proměnná v čase, musí být i obsah databáze proměnný. Musí tedy existovat prostředek, který umožní obsah databáze měnit. Takovým prostředkem je jazyk pro manipulaci s daty (JMD). JMD musí umožňovat základní databázové operace vyhledávání, vkládání, rušení a modifikaci prvků relace. Nejdůležitější je operace vyhledávání. Víme, že i při modifikaci a vypouštění prvků se odpovídající prvek musí nejprve vyhledat. Ostatní operace jsou z hlediska JMD triviální.

Jazyky pro **formulaci požadavků** na výběr dat z relační databáze (dotazovací jazyky) se dělí do dvou skupin:

1. jazyky založené na **relační algebře**, kde jsou výběrové požadavky vyjádřeny jako posloupnost speciálních operací prováděných nad daty; dotaz je tedy zadán algoritmem, jak vyhledat požadované informace;
2. jazyky založené na **predikátovém kalkulu**, které požadavky na výběr zadávají jako predikát charakterizující vybranou relaci; je úlohou překladače jazyka nalézt odpovídající algoritmus;
   tyto jazyky se dále dělí na
   - **n-ticové relační kalkuly**
   - **doménové relační kalkuly**.

Oba typy jazyků jsou ekvivalentní, pokud se týče možností formulace výběrových podmínek. Každý požadavek formulovaný v relační algebře se dá vyjádřit v relačním kalkulu a naopak. Skutečně realizované jazyky se liší bohatší syntaxí a poskytují uživateli další pomocné funkce.

4.2.1. Relační algebra

Relační algebra je velmi silný dotazovací jazyk vysoké úrovně. Nepracuje s jednotlivými enticemi relací, ale s celými relacemi. Operátory relační algebry se aplikují na relace, výsledkem jsou opět relace. Protože relace jsou množiny, přirozenými prostředky pro manipulaci s relacemi budou množinové operace.

I když relační algebra v této podobě není vždy implementována v jazycích SŘBD, je její zvládnutí nutnou podmínkou pro správnost manipulací s relacemi. I složitější dotazy jazyka SQL, který je deskriptivním dotazovacím jazykem, mohou být bez zkušeností s relační algebrou problematické.

❑ Základní operace relační algebry

Jsou dány relace R a S.

Množinové operace

sjednocení relací téhož stupně $R \cup S = \{x \mid x \in R \vee x \in S\}$

průnik relací $R \cap S = \{x \mid x \in R \wedge x \in S\}$

rozdíl relací $R - S = \{x \mid x \in R \wedge x \notin S\}$

kartézský součin relace R stupně m a relace S stupně n

$$R \times S = \{rs \mid r \in R \wedge s \in S\}, \text{ kde } rs = \{r_1,...,r_m,s_1,...s_n\}$$

Další operace relační:

projekce (výběr sloupců) relace R na komponenty $A=\{A_{i1},...A_{im}\}$, $1 \le i_m \le n$, $i_j \ne i_k$ pro $j \ne k$

$$R[A] = \{r[A] \mid r \in R\}, \quad \text{kde } R[A] = (r_{i1}, \ldots, r_{im}) \text{ pro } r \in R$$

selekce (výběr řádků) z relace R podle podmínky P je

$$R(P) = \{r \mid r \in R \wedge P(r)\}$$

spojení relací R s atributy A a S s atributy B dle relačního operátoru $\Theta = \{<,=,>,<=,>=,<>\}$ v atributu A_i relace R a v atributu B_j relace S je

$$R[A\Theta B]S = \{rs \mid r \in R \wedge s \in S \wedge r[A_i] \; \Theta \; s[B_j]\}$$

Θ-spojení lze definovat jako kartézský součin operandů následovaný selekcí.
Nejčastěji se používá spojení relací podle operátoru "="; v tom případě by ve výsledné relaci byly dva sloupce shodné, proto se zavádí operace

$$R[A*B]S = \{R[A=B]S\},$$

která automaticky jeden ze shodných sloupců vypouští.

přirozené spojení relací R(A) a S(B)

$$R[*]S = ((RxS)[P])[A_1,...,A_k,C_1,..,C_{m+n-k}]$$

kde A_i jsou všechny atributy se shodnými jmény jako atributy z B a C_i jsou ostatní atributy z A i B; ze součinu RxS se vyberou ty prvky, které mají stejné hodnoty na stejně pojmenovaných atributech. Je to zvláštní případ obecného spojení.

Jinak řečeno, přirozené spojení je spojení na základě rovnosti hodnot stejnojmenných atributů.

Příklad 4.6.

Jsou dány relace X, Y. Pak kartézský součin a přirozené spojení jsou:

X

A	B	C
3	S	1
5	C	2
7	T	3

Y

C	D	E
2	ff	S
2	gg	T
3	hh	C
6	jj	D

XxY

A	B	X.C	Y.C	D	E
3	S	1	2	ff	S
3	S	1	2	gg	T
3	S	1	3	hh	C
3	S	1	6	jj	D
5	C	2	2	ff	S
5	C	2	2	gg	T
5	C	2	3	hh	C
5	C	2	6	jj	D
7	T	3	2	ff	S
7	T	3	2	gg	T
7	T	3	3	hh	C
7	T	3	6	jj	D

X[*] Y

A	B	C	D	E
5	C	2	ff	S
5	C	2	gg	T
7	T	3	hh	C

♦

Příklad 4.7.

Jsou dány relace databáze VSB-TUO*:* Zaměstnanec (rodcis, jmeno, adresa, katedra), Student (rodcis, jmeno, adresa, fakulta). *Pomocí relační algebry napište*

a) seznam jmen a adres všech studentů [projekce, výběr některých sloupců celé tabulky]

Student [jmeno, adresa]

b) seznam všech údajů o studentech z fakulty elektrotechniky [selekce, výběr některých řádků splňujících podmínku]

Student (fakulta = „FEI“)

c) seznam jmen a rodných čísel studentů z fakulty elektrotechniky [selekce a pak projekce]

Student (fakulta = „FEI“) [jmeno, rodcis]

Pozor! je možno v tomto případě zaměnit pořadí selekce a projekce?

d) rodná čísla a jména studentů, zaměstnaných na této škole [= průnik obou relací, ale protože relace nemají stejnou strukturu, nejprve provedeme projekci na rodcis a jmeno *]*

Student [jmeno, rodcis] ∩ Zaměstnanec [jmeno, rodcis]

e) seznam jmen a adres studentů, kteří nepracují na této škole [od studentů odečteme zaměstnance, opět nejprve sjednotíme struktury]

Student [jmeno, adresa] - Zaměstnanec [jmeno, adresa]

f) seznam jmen a rodných čísel všech osob (studentů i zaměstnanců), majících právo se stravovat v menze [sjednocení, včetně vyloučení případných duplicit zaměstnaných studentů]

Student [jmeno, rodcis] ∪ Zaměstnanec [jmeno, rodcis] ♦

Příklad 4.8.

Mějme relaci učitel: **U** (ČU, jméno, fce, plat) *a vazbu učí* : **V** (ČU, ČP, hodin)

Máme najít čísla těch učitelů, kteří

vyučují alespoň jeden předmět:	V[ČU]
nevyučují žádný předmět:	U[ČU] - V[ČU]
vyučují předmět P2:	(V(ČP = 'P2'))[ČU]
vyučují alespoň jeden předmět, ale ne předmět P2:	V[ČU] - ((V(ČP = 'P2'))[ČU])
nevyučují předmět P2:	U[ČU] - ((V(ČP = 'P2'))[ČU])
vyučují jiný předmět, než P2:	(V(ČP <> 'P2'))[ČU]
vyučují pouze předmět P2:	((V(ČP = 'P2'))[ČU]) - ((V(ČP <> 'P2'))[ČU])

Najděte jména učitelů, kteří

vyučují alespoň jeden předmět:	(V[ČU][*]U)[jméno]

♦

Uvedená množina operací relační algebry **není minimální**, tj.existují operace, které se dají vyjádřit pomocí ostatních. Např. R∩ S = R - (R - S). O spojení již jsme uvedli, že je realizovatelné pomocí součinu a selekce.

Dotazovací jazyk, který má realizovány všechny operace relační algebry, se nazývá **relačně úplný**.

K relačním operacím se často pro rozšíření možností jazyka přidávají aritmetické a řetězcové operace nad hodnotami atributů nebo agregační funkce, které množinám přiřazují číslo (počet prvků množiny, součet hodnot atributu ap.). Ty se naučíme používat poprvé u jazyka SQL.

Shrnutí pojmů 4.2.1.

Relační algebra, operace s relacemi.
Množinové operace sjednocení, průnik, rozdíl, kartézský součin.
Relační operace selekce, projekce, spojení, přirozené spojení.
Databázové operace pro manipulaci s daty.

Otázky 4.2.1.

1. Jaké typy dotazovacích jazyků se používají v relačním datovém modelu?
2. Jaký vztah je mezi jazykem pro manipulaci s daty a dotazovacím jazykem?
3. Co je a k čemu k čemu je relační algebra?
4. Které operandy a které operace používá relační algebra? Definujte tyto operace.
5. Které operace RA se dají nahradit ostatními?

Úlohy k řešení 4.2.1.

1. Je dána databáze ŠKOLA se schématy

   Učitel (ČU, jméno, fce, plat)
   Předmět (ČP, název)
   Učí (ČU, ČP, hodin)

Najděte pomocí relační algebry jména těch učitelů, kteří

a) vyučují alespoň jeden předmět

b) nevyučují žádný předmět

c) vyučují předmět Překladače

d) vyučují alespoň jeden předmět, ale ne předmět Překladače

e) nevyučují předmět Překladače

f) vyučují jiný předmět, než Překladače

g) vyučují pouze předmět Překladače

2. Soukromá lékařská praxe je podporovaná relační databází se třemi relacemi se schématy:

   Lékař (číslo_licence, jménoL, specializace)
   Pacient (číslo_pacienta, jménoP, adresa, telefon, dat_narození)
   Návštěva (číslo_licence, číslo_pacienta, typ, datum, diagnóza, cena)

   kde typ znamená typ návštěvy (domů na zavolání, v ordinaci, do nemocnice ap.) Napište pomocí operací relační algebry tyto požadavky:

   a) seznam všech specializací lékařů

   b) jmenný seznam všech ortopédů

   c) jmenný seznam pacientů starších 65 let

   d) seznam licencí lékařů, které navštívila paní Marie Nová

   e) jména lékařů, kteří byli na návštěvě domů na zavolání

   f) jména a adresy pacientů, kteří byli vyšetřeni dr. Lomem dne 23.5.2003

   g) jména a adresy pacientů, kterým byla určena diagnóza HIV+

   h) jména a specializace lékařů, kteří určili diagnózu vřed na dvanácterníku

   i) jména a adresy pacientů, kteří byli vyšetřováni pouze dr. Čermákem.

4.2.2. N-ticový relační kalkul

Jako jazyk pro výběr informací z relační databáze lze také využít jazyk matematické logiky. V Coddově definici RMD byl zaveden n-ticový relační kalkul, později se objevil přirozenější doménový relační kalkul. Pod pojmem relační kalkul tedy zahrnujeme oba jazyky.

Název n-ticového relačního kalkulu je odvozen z oboru hodnot jeho proměnných a je definován jako predikátový počet. Vychází z toho, že relace je množina prvků ~ n-tic. Je základem pro jazyk typu SQL, který je rozšířený prakticky ve všech SŘBD, je používán jako standard a který popíšeme níže.

Syntaxe kalkulu je přizpůsobena programovacímu jazyku. Obvyklé matematické vyjádření množiny prvků x splňujících podmínku F(x)

$$\{ x \mid F(x) \}$$

nahradíme čtivějším a programovacímu jazyku bližším zápisem

x WHERE F(x)

Definice 4.6.

Abecedu n-ticového relačního kalkulu tvoří:

- atomické konstanty (hodnoty atributů), zapisují se v apostrofech,
- n-ticové proměnné (proměnné, jejichž oborem hodnot jsou n-tice); označujeme je identifikátory; za n-ticové proměnné lze dosazovat n-tice relací;
- komponenty proměnných (indexové konstanty), odvolávky na atributy relací; označíme je jménem atributu a odkazem na relaci,
- predikátové konstanty (jména relací),
  predikátové operátory binární < <= > >= = <> , obecně *
- logické operátory a kvantifikátory OR AND NOT EXIST FORALL
- oddělovače ( )

Formulí n-ticového relačního kalkulu je

- atomická formule R(r), kde R jméno relace, r je n-ticová proměnná; formule znamená, že r je prvkem relace R;
- atomické formule r.a * s.b, r.a * 'k' , 'k' * s.b, kde r, s jsou n-ticové proměnné, a, b jsou komponenty proměnných (atributy), 'k' je atomická konstanta, * je binární operátor, r.a je atribut a n-tice r, s.b atribut b n-tice s;
- jsou-li F1 a F2 formule, pak také F1 OR F2, F1 AND F2, NOT F1 jsou formule;
- je-li F formule, pak také EXIST r(F(r)), FORALL r(F(r)) jsou formule;
- nic jiného není formule.

Podobně jako v predikátovém počtu jsou proměnné vázané kvantifikátory EXIST a FORALL nazývány vázanými proměnnými, ostatní n-ticové proměnné jsou volné.

Formule relačního kalkulu reprezentuje **vyhledávací podmínku.**

Příklad 4.9.

Konstanty atomické = hodnoty z domén atributů: `8, 'Kovář', 'docent'`
Konstanty predikátové = názvy relací: `Zaměstnanec, Učitel`
N-ticové proměnné = proměnné, jejichž oborem hodnot jsou n-tice: `xZam, yUčitel, x, y`
Komponenty proměnných = označení jednotlivých atributů relací, označené prefixem (názvem relace) a názvem atributu: `Zam.jméno, Učitel.plat, xZam.rodcis`

♦

Definice 4.7.

Výraz n-ticového relačního kalkulu je výraz tvaru

x WHERE F(x)

kde x je jediná volná proměnná ve formuli F.

Výraz n-ticového relačního kalkulu určuje **výslednou relaci** tvořenou všemi možnými hodnotami proměnné x, které splňují formuli F(x).

x na levé straně definuje seznam komponent proměnných, které definují schéma výstupní relace. Je to buď již dříve definovaná entita, množina entit nebo seznam komponent volných proměnných.

Příklad 4.10.

V relaci Učitel(ČU, jméno, fce, plat) *hledáme*

úplný seznam všech učitelů: `xUčitel WHERE Učitel(xUčitel)`
jména a osobní čísla učitelů: `x.jméno,x.ČU WHERE Učitel(x)`
jména všech docentů: `x.jméno WHERE Učitel(x) AND x.fce = 'docent'`

♦

❑ Vztah relační algebry a relačního kalkulu

Základní operace relační algebry se dají vyjádřit pomocí výrazů n-ticového relačního kalkulu, tedy n-ticový relační kalkul je relačně úplný.

Platí: $R \cup S$	=>	x WHERE R(x) OR S(x)
$R \cap S$	=>	x WHERE R(x) AND S(x)
R - S	=>	x WHERE R(x) AND NOT S(x)
R x S	=>	x, y WHERE R(x) AND S(y)
$R[a_1,a_2,...,a_k]$	=>	$x.a_1$, $x.a_2$,..., $x.a_k$ WHERE R(x)
R(P)	=>	x WHERE R(x) AND P
R[A*B]S	=>	x, y WHERE R(x) AND S(y) AND x.A * y.B

Příklad 4.11.

Mějme opět relace z minulého příkladu. Učitel (ČU, jméno, fce, plat), Učí (ČU, ČP, hodin).

Pak `Učitel(x)` *nabývá hodnoty TRUE nebo FALSE, je-li* `x` *v tabulce* `Učitel` *nebo ne*; *obdobně* Učí(x).

Určete jmenný seznam učitelů (projekce)
x.jméno WHERE Učitel(x)
Seznam čísel učitelů, kteří učí předmět P2 (selekce a projekce)
x.ČU WHERE Učí(x) AND Učí.ČP='P2'
Seznam čísel učitelů a jejich počtu hodin v jednotlivých předmětech
x.ČU, x.hodin WHERE Učí(x)
jmenný seznam učitelů a jejich úvazků v hodinách (přirozené spojení)
y.jméno,x.hodin WHERE Učitel(y) AND Učí(x) AND y.ČU=x.ČU

♦

Poznámka:

Relační kalkul, jak byl zatím definován, umožňuje zapsat i nekonečné relace, např.

t WHERE NOT R(t)

Tento výraz definuje všechny n-tice, které nepatří do relace R, tedy nekonečné množství n-tic.

Výrazy relačního kalkulu se proto omezují jen na tzv. bezpečné výrazy, které definování relací nekonečných nedovolují.

Protože na relačním kalkulu je založen velmi rozšířený dotazovací jazyk SQL, budeme podrobně procvičovat dotazovací schopnosti kalkulu až tam.

Shrnutí pojmů 4.2.2.

Abeceda, formule n-ticového relačního kalkulu.

Výraz n-ticového relačního kalkulu a jeho význam.

Vztah relační algebry a n-ticového relačního kalkulu.

Otázky 4.2.2.

1. Co je relační kalkul a na jakém teoretickém základě je založen?
2. Odkud pochází název „n-ticový“ kalkul?
3. Definujte výraz n-ticového relačního kalkulu a uveďte ke každému pojmu definice příklad z reality.

Úlohy k řešení 4.2.2.

1. Napište všechny dotazy z úlohy 4.2.1./2 pomocí n-ticového relačního kalkulu.

4.2.3. Jazyk SQL

Jazyk SQL (Strucrured Query Language) byl původně navržen u firmy IBM jako dotazovací jazyk (původní název Sequel). Jeho základem je n-ticový relační kalkul, syntaxe je mírně upravena.

Obsahuje však navíc i příkazy

- pro vytvoření a modifikace tabulek,
- pro ukládání, modifikaci a rušení dat v databázi
- a řadu dalších příkazů, které z něj dělají kompletní jazyk pro práci s databází.

Příkazy jazyka pro definici dat

Vytvoření definice tabulky:

CREATE TABLE tab (ident1 {NUMBER | CHAR | DATE }(délka)

[, ident2 ... , ident3 ...])

Příklad 4.12.

CREATE TABLE Včelaři (jméno CHAR(20), adresa CHAR(30), okres CHAR(15),
včel NUMBER(4), medu NUMBER(4));

jméno	adresa	okres	včel	medu

Modifikace struktury tabulky

ALTER TABLE tab {MODIFY (ident dat_typ (nový rozměr)) |
ADD ident dat_typ (rozměr) |
DROP ident}

Příklad 4.13.

Sloupec pro medu NUMBER(4) *už nestačí rozsah, je třeba přidat 2 cifry.*
ALTER TABLE Včelaři MODIFY (medu NUMBER (6)); ♦

Příklad 4.14.

Do tabulky Včelaři *je nutno přidat sloupec o tom, jestli se u včelstev vyskytuje nemoc varoáza.*
ALTER TABLE Včelaři ADD (varoáza NUMBER (1)); ♦

Přejmenování tabulky nebo sloupce tabulky

RENAME TABLE tab_stará TO tab_nová

RENAME COLUMN tab.sloupec_starý TO tab.sloupec_nový

Zrušení tabulky včetně definice

DROP TABLE tab

Vytvoření a rušení indexu:

CREATE [DISTINCT] INDEX index ON tab (seznam_klíčů)

DROP INDEX index

Klauzule DISTINCT znamená požadavek jednoznačnosti indexu

Příklad 4.15.

Potřebujeme vytvořit index okresů včelařů.

CREATE DISTINCT INDEX ind_včel ON Včelaři (okres);

Index zaměstnanců fakulty podle kateder

CREATE INDEX ind_fakul ON fakulta (kat, jméno); ♦

❑ Příkazy jazyka pro modifikace dat

❑ Vkládání nových řádků

se provádí příkazem INSERT. Proti použití na počátku je možno specifikovat i seznam a pořadí sloupců, do kterých se budou hodnoty ukládat a tak není nutné uvádět i hodnoty sloupců nevyplňovaných NULL (protože jejich hodnoty nejsou známy nebo budou dopočítány nebo doplněny později).

INSERT INTO tab [seznam_sloupců] VALUES (seznam_hodnot)

Příklad 4.16.

Nový včelař se přihlásil do organizace, ale dosud nemá žádné výnosy.

INSERT INTO Včelaři jméno, adresa, okres
VALUES (‘Novotný Karel’,’Opava, Květná 34’,’Opava’);

Bez specifikace sloupců bychom museli uvést všechny hodnoty sloupců:

INSERT INTO Včelaři
VALUES (‘Novák Jiří’,’Kravaře, Lidická 32’,’Opava’,NULL,NULL);

jméno	adresa	okres	včel	medu
Novotný Karel	Opava, Květná 34	Opava	NULL	NULL
Novák Jiří	Kravaře, Lidická 32	Opava	NULL	NULL

♦

Pomocí příkazu INSERT je možno také naplňovat řádky tabulky hodnotami z jiné tabulky tak, že místo klauzule VALUES použijeme SELECT, v němž budou zadány řádky i sloupce jiných tabulek, které se do naší tabulky mají přenést.

Protože dosud příkaz SELECT neznáme, uvedeme tuto variantu Příkazu INSERT později.

- **Modifikace hodnot** v řádcích tabulky

UPDATE tab
SET ident1=výraz1 [, ident2=výraz2,...] [WHERE podm]

Příklad 4.17.

Včelař Beran Alois se přestěhoval do Benešova a má nyní 92 včelstev.

UPDATE Včelaři
SET adresa = 'Benešov, Šeříková 55', včel = 92
WHERE jméno = 'Beran Alois'; ♦

Příklad 4.18.

Všichni zaměstnanci katedry 345 dostali přidáno na platu 300 Kč.

UPDATE Zam
SET plat = plat + 300
WHERE kat = 345; ♦

Příklad 4.19.

Prémii ve výši 20% platu dostali všichni zaměstnanci s uvedením „ano" ve sloupci odměny.

UPDATE Zam
SET plat = plat * 1.2
WHERE odměny = „ano" ♦

- **Rušení řádků** tabulky

DELETE FROM tab WHERE podm

Příklad 4.20.

Zaměstnanci Kurel a Žižka nedostanou žádné prémie, vymažou se z tabulky Odměny.

DELETE FROM Odměny
WHERE jméno = 'Kurel' OR jméno = 'Žižka'; ♦

Výběr informací z tabulky

Následující příkaz SELECT reprezentuje vlastní dotazovací jazyk. Jeho použitím je možno nejen vyhledávat údaje v databázi obsažené, ale i údaje odvozené, případně vhodně setříděné. Základní nejjednodušší tvar příkazu je

SELECT {seznam_sloupců | *}
FROM tab
[WHERE podm]

Použití znaku * místo seznamu sloupců znamená výpis všech sloupců tabulky.

Příkaz jazyka SQL: SELECT $A_1, A_2, \ldots, A_k$ FROM R WHERE podm

odpovídá výrazu relační algebry: (R (podm)) $[A_1, A_2, \ldots, A_k]$

nebo výrazu n-ticového relačního kalkulu: $x.A_1, x.A_2, \ldots, x.A_k$ WHERE R(x) AND podm

Jednoznačnost prvků výsledné relace nezajišťuje jazyk SQL automaticky, ale musí se zadat v příkazu klauzulí DISTINCT (nebo UNIQUE)

Příklad 4.21.

Celou tabulku vytiskneme příkazem

SELECT * FROM Včelaři; ♦

Příklad 4.22.

Jmenný seznam a adresy všech včelařů dostaneme příkazem

SELECT jméno, adresa FROM Včelaři; ♦

Příklad 4.23.

Seznam okresů, v nichž jsou organizováni Včelaři, dostaneme

SELECT DISTINCT okres FROM Včelaři; ♦

Příklad 4.24.

Seznam včelařů z okresu Opava

SELECT * FROM Včelaři WHERE okres = „Opava“; ♦

❑ **Podmínka selekce**

se zapisuje za klauzulí WHERE. Ve výběrové podmínce je možno používat:

konstanty, identifikátory sloupců,
relační operátory: = <> < <= > >=
logické operátory: **NOT AND OR**
další operátory: **BETWEEN dolní mez AND horní mez**
IN(seznam_prvků_množiny)
IS NULL
LIKE vzor ... pro porovnání řetězců podobně jako u hvězdičkové konvence:
% ... odpovídá skupině znaků
_ ... podtržítko zastupuje jeden znak

Příklad 4.25.

Seznam včelařů z okresů Praha, Ústí a Ostrava, kteří mají mezi 50 a 100 včelstvy:

```
SELECT *
    FROM Včelaři
    WHERE okres IN('Praha','Ústí','Ostrava')
        AND včel BETWEEN 50 AND 100;
```
♦

Příklad 4.26.

Seznam včelařů se jménem začínajícím písmenem K.

```
SELECT *
    FROM Včelaři
    WHERE jméno LIKE 'K%';
```
♦

Příklad 4.27.

Seznam včelařek (= jméno končí na -ová).

```
SELECT *
    FROM Včelaři
    WHERE jméno LIKE '%ová';
```
♦

❑ Setřídění výsledku

výsledných řádků podle třídicího klíče, ne podle pořadí uložení v souboru:

SELECT {seznam_sloupců | *}
FROM tab
[WHERE podm]
[ORDER BY třídicí klíč [DESC]]

Příklad 4.28.

Seznam včelařů v pořadí podle získaného medu od 500 kg výše.

```
SELECT *
    FROM Včelaři
    WHERE medu>500
    ORDER BY medu DESC; ♦
```

Pokud jsou v třídicím klíči prázdné hodnoty, uvádí se tyto řádky vždy na začátku tabulky při sestupném i vzestupném třídění.

❑ Spojení

více tabulek (vazba) se provede uvedením všech tabulek za FROM a podmínka spojení se uvede jako součást výběrové podmínky za WHERE. **Bez této podmínky** by se provedl prostý **kartézský součin** vyjmenovaných tabulek. Rozlišení stejnojmenných sloupců provedeme uvedením jména tabulky před jménem sloupce a oddělené tečkou.

SELECT {seznam_sloupců | *}
FROM seznam_tabulek
[WHERE podm_spojení [AND podm_selekce]]

Příklad 4.29.

Jmenný seznam včelařů s adresami je v tabulce Včelaři a záznamy o každoročních sklizních v tabulce med. Zajímá nás adresa včelaře Žáry a kolik získal medu v roce 1994.

```
SELECT Včelaři.jméno, adresa, medu
    FROM Včelaři, Med
    WHERE rok = 1994 AND Med.jméno = 'Žára' AND Včelaři.jméno = Med.jméno; ♦
```

Jiný způsob zápisu spojení tabulek s vhodnější syntaxí, která odděluje podmínku spojení od klasické podmínky selektivní, je

SELECT {seznam_sloupců | *}
FROM tabulka1 JOIN tabulka2 ON podm_spojení
[WHERE podm_selekce]

Podmínka spojení přitom může být složená, můžeme spojovat tabulky pomocí více sloupců současně.

Příklad 4.30.

Jmenný seznam včelařů s adresami je v tabulce Včelaři (jmeno, adresa, okres) a záznamy o každoročních sklizních v tabulce Med (jmeno, rok, medu, vcel). Zajímá nás adresa včelaře Žáry a kolik získal medu v roce 1994.

```
SELECT Včelaři.jméno, adresa, medu
    FROM Včelaři JOIN Med ON Včelaři.jméno = Med.jméno
    WHERE rok = 1994 AND Med.jméno = 'Žára' AND; ♦
```

Spojení více než dvou tabulek se provádí buď dalším JOIN nebo lépe bez JOIN klasickou podmínkou spojení za WHERE.

Příklad 4.31.

Jmenný seznam včelařů s adresami je v tabulce Včelaři (jmeno, adresa, okres, id_organ), záznamy o každoročních sklizních v tabulce Med (jmeno, rok, medu, vcel) a seznam okresních včelařských organizací.v tabulce Organizace_vcel (id_organ, okres, predseda)

Zajímá nás okres a předseda včelaře Žáry a kolik získal medu v roce 1994.

```
SELECT Organizace_vcel.okres, predseda,Včelaři.jméno, medu
    FROM Organizace_vcel JOIN (Vcelari JOIN Med ON Včelaři.jméno = Med.jméno)
        ON Vcelari.id_organ=Organizace_vcel.id_organ
    WHERE rok = 1994 AND Med.jméno = 'Žára' AND;
```

Tentýž dotaz bez použití JOIN – pro spojení více tabulek je tato varianta přehlednější:

```
SELECT Organizace_vcel.okres, predseda,Včelaři.jméno, medu
    FROM Včelaři, Med, Organizace_vcel
    WHERE Včelaři.jméno = Med.jméno AND
        Vcelari.id_organ=Organizace_vcel.id_organ AND
        rok = 1994 AND Med.jméno = 'Žára' AND; ♦
```

- **Použití prefixů**

Pokud je název tabulky jako prefix nepohodlně dlouhý, nebo pokud potřebujeme jednu tabulku označit dvakrát pokaždé jinak (např. pro realizaci unární vazby), můžeme za klauzulí FROM každé tabulce zadat vlastní prefix.

Podmínka spojení může být nejen na rovnost hodnot.

```
SELECT {seznam_sloupců | *}
    FROM  tab1  P1, tab2  P2, . . .
```

Příklad 4.32.

Dotaz z příkladu 4.29 by tak vypadal jednodušeji

```
SELECT V.jméno, adresa, medu
    FROM Včelaři V, Med M
    WHERE rok = 1994 AND M.jméno = 'Žára' AND V.jméno = M.jméno; ♦
```

Příklad 4.33.

V tabulce Včelaři (jmeno, adresa, okres, preds) je také jméno předsedy místní organizace - u každého včelaře. Zajímá nás seznam včelařů, kteří získali více medu, než jejich předseda. Musíme použít 2 x tutéž tabulku, jednou pro seznam včelařů – označíme X, podruhé pro seznam jejich předsedů – označíme Y.

```
SELECT X.jméno, X.medu, Y.medu
    FROM Včelaři X, Včelaři Y
    WHERE X.předs = Y.předs AND X.medu >Y.medu; ♦
```

- **Spojení vnitřní a vnější, levé a pravé**

Prozatím jsme používali jen tzv. **vnitřního spojení** (též INNER JOIN), kdy se do výsledku dostanou jen ty řádky z obou tabulek, pro které je splněna podmínka spojení. Pokud se při spojování tabulek nenajde v jedné tabulce odpovídající řádek ze druhé tabulky, ve výsledku spojení se odpovídající hodnota spojovacího klíče neobjeví.

Může však nastat situace, že požadujeme ve výsledné tabulce i všechny řádky obou tabulek. Potom pokud je podmínka spojení splněna, tvoří řádky obou tabulek řádek ve výsledku (jako u vnitřního spojení). Pokud spojovací podmínka není splněna, ve výsledku je řádek jedné z tabulek a odpovídající hodnoty ze druhé tabulky zůstanou prázdné. Takové spojení nazýváme **vnějším spojením** a zapíšeme OUTER JOIN.

Většinou potřebujeme doplnit řádky jen z jedné z obou tabulek – levé nebo pravé podle toho, jak jsou zapsány za FROM. Pak to zapíšeme LEFT JOIN, pokud se mají připojit všechny řádky tabulky levé, nebo RIGHT JOIN, pokud se mají doplnit všechny řádky tabulky pravé. Tato spojení nazýváme **levé vnější** a **pravé vnější spojení**.

Příklad 4.34.

V tabulce Včelari (jmeno, adresa, okres) jsou základní údaje a v tabulce Med (jmeno, rok, medu, vcel) záznamy o každoročních sklizních. Ve Vcelari mohou být členové, kteří v roce 1989 ještě včely nepěstovali, ale ve výsledném seznamu jejich jména požadujeme s nulovou snůškou medu.

```
SELECT X.jméno, M.medu
        FROM Včelaři X  LEFT JOIN Med M ON M.jméno = X.jméno
        WHERE rok = 1989;♦
```

❑ Výrazy a funkce, agregované funkce

Pro vytváření výrazů používá SQL aritmetických operátorů a závorek v obvyklých významech. Místo jména sloupce můžeme použít výraz, a to v seznamu za SELECT či v podmínce za WHERE. Pokud je výraz použit jako prvek seznamu za SELECT a chceme příslušnému sloupci na výstupu přiřadit sloupcový nadpis vlastní, zapíšeme ho po mezeře za výrazem. Pokud se nadpis skládá z více slov, uzavírá se do závorek.

Příklad 4.35.

Vypište pro každého včelaře průměrnou snůšku medu na včelstvo.

```
SELECT jméno, medu/včel
        FROM Včelaři; ♦
```

❑ Náhrada hodnoty NULL

Pokud je ve výrazu proměnná, která má hodnotu NULL, má celý výraz hodnotu NULL. Někdy se takový výstup nehodí a chceme nahradit hodnotu NULL konkrétní náhradní hodnotou (např. nulou pro číslo nebo mezerou pro text). K tomu slouží funkce

NVL (prom, náhr_hodn)

Příklad 4.36.

Sloupec s vypočítaným průměrem nazvěte průměr.

```
SELECT jméno, medu/NVL(včel,1) průměr FROM Včelaři; ♦
```

❑ Funkce nad atributy

Dále jazyk SQL používá následující funkce

- aritmetické: **POWER(číslo,mocnitel) mocnina**
  ROUND(číslo,poč_des_míst)
  TRUNC(číslo,poč_des_míst)
  ABS(číslo)
  SIGN(číslo)
  MOD(číslo1,číslo2)

SQRT(číslo)
GREATEST(seznam_hodnot) maximum
LEAST(seznam_hodnot) minimum

- řetězcové: **řetězec1 || řetězec2**
  LENGHT(řetězec)
  SUBSTR(řetězec,pozice,délka) výběr podřetězce
  INSTR(řetězec,podřetězec,pozice,pořadí)
  UPPER(řetězec)
  LOWER(řetězec)
  TO_NUMBER(řetězec) konverze text -> číslo
  TO_CHAR(řetězec[,formát]) datum či číslo -> text
  TO_DATE(řetězec[,formát]) text či číslo -> datum
  LPAD(řetězec,délka,[,znak]) doplní zleva
  RPAD(řetězec,délka,[,znak]) doplní zprava
  LTRIM(řetězec,množ_znaků) vypustí zleva
  RTRIM(řetězec,množ_znaků) vypustí zprava
  TRANSLATE(řetězec,množ_vzorů,množ_obrazů)
  DECODE(sl,vzor1,obraz1,...,implic)

  Decode je tzv. překlad tabulkou: obraz může být konstanta nebo jméno sloupce; může sloužit i jako přepínač, když ve sloupci nechceme homogenní hodnoty.

- datumové: číselné masky pro formátování datumových hodnot:

CC století	YYYY, YYY, YY, Y rok
Q kvartál	WW, W týden v roce, v měsíci
MM,MMM měsíc v roce	DDD, DD, D den v roce, měsíci, týdnu
HH hodina	MI minuta

Příklad 4.37.

U zaměstnanců Fakulty *potřebujeme tisknout také kód učitelské funkce podle pravidla* asistent=1, docent = 2, profesor = 3, ostatní zaměstnanci = 4.

```
SELECT jméno, funkce, DECODE (funkce,'asist',1,'doc',2,'prof',3,4)
    FROM Fakulta;
```

Jméno	funkce	kat
Adam Alois	doc	2
Beran Bedřich	topič	4
Cerman Cyril	asist	1

♦

Příklad 4.38.

U mimopražských včelařů budeme tisknout adresy, u pražských jen Praha (=okres*), sloupec nazveme* odkud.

```
SELECT jméno, DECODE (okres, ‚Praha', okres, adresa) odkud
    FROM Včelaři;
```

Jméno	odkud
Adam Alois	Praha
Beran Bedřich	U pošty 15, Dobřichovice
Cerman Cyril	Praha

♦

❑ Funkce agregované

V praxi se často potřebují některá data v databázi sčítat, průměrovat apod. Představíme-li si tyto operace nad tabulkou, jde buď o počty záznamů (řádků) splňujících nějakou podmínku, nebo o operace sloupcové: součet, průměr, minimum, maximum hodnoty některého sloupce.

I když nejde o operace relační (nepracují s prvky relace = řádky, ale s daty ve sloupci, tedy z každého prvku používají jen komponentu), existují ve všech databázových jazycích i v SQL funkce, které tyto výpočty provádějí. Nazýváme je souhrnně funkcemi agregovanými. Jsou to:

AVG ({[DISTINCT] sez_výr|*})	průměr ve sloupci
SUM ({[DISTINCT] sez_výr|*})	součet …
MIN ({[DISTINCT] sez_výr|*})	minimum …
MAX ({[DISTINCT] sez_výr|*})	maximum …
COUNT({[DISTINCT] sez_výr|*})	počet vyhovujících řádků

Příklad 4.39.

Určete průměrný, minimální a maximální počet včelstev v okrese Karviná.

```
SELECT AVG(včel), MIN(včel), MAX(včel)
    FROM Včelaři
    WHERE okres = 'Karviná';
```
♦

Příklad 4.40.

Kolik je v okrese Praha včelařů ?

```
SELECT COUNT(*) FROM Včelaři WHERE okres = 'Praha';
```
♦

Příklad 4.41.

Kolik medu získal nejlepší včelař z Ostravy?

```
SELECT MAX( medu)
    FROM Včelaři
    WHERE okres = 'Ostrava';
```
♦

Příklad 4.42.

V kolika okresech Včelaři pracují ?

```
SELECT COUNT (DISTINCT okres) FROM Včelaři;
```
♦

❑ Grupování a podmínka pro grupy

Někdy potřebujeme provést agregaci (určit počet, součet, průměr, …) ne pro celou tabulku nebo jednu definovanou selekci, ale pro jednotlivé **skupiny řádků se stejnou hodnotou některého atributu**.

Tabulku si můžeme představit uspořádanou tak, že vzniknou skupiny řádků se stejnou hodnotou vybraného atributu. Také pro tyto skupiny můžeme provádět operace (částečné počty, součty ap.). Skupinám řádků se stejnou hodnotou grupovacího klíče budeme říkat grupy.

Nyní si můžeme všechny atributy tabulky rozdělit podle významu do třech skupin:

(1) atributy definující grupu, tedy ve výsledku se **ke každé skupině** stejných hodnot těchto atributů vstupní tabulky **vytvoří jeden řádek** ve výstupní tabulce;

(2) atributy agregovatelné, které se za celou grupu sčítají, průměrují apod., tedy mají za celou skupinu opět jen jednu hodnotu;

(3) ostatní atributy s různými hodnotami ve skupině a tedy ve výsledku nemají žádný význam.

(3) jméno	(3) adresa	(1) okres	(2) včel	(2) medu
Beneš	…	Opava	…	100
Gajdoš	…	Opava	…	230
…				
Adam		Bruntál		320
…				
Žižka	…	Ostrava	…	125

(1) okres	(2) AVG (medu)	(2) AVG (včel)
Opava		
Bruntál		
…		
Ostrava		

Za klauzulí SELECT tedy mohou být jen atributy, podle nichž se grupuje, a agregované hodnoty atributů agregovatelných.

Vytvoření skupin se provede klauzulí GROUP BY klíč

SELECT {seznam_výrazů | *}
FROM seznam_tabulek
GROUP BY seznam_sloupců

Příklad 4.43.

Najděte průměrné množství medu v jednotlivých okresech.

```
SELECT okres, AVG(medu)
        FROM Včelaři
        GROUP BY okres; ♦
```

Příklad 4.44.

Rozdělte zaměstnance fakulty podle kateder a funkcí, pro každou tuto skupinu určete jejich počet a průměrný plat.

```
SELECT kat, funkce, COUNT(*), AVG(plat)
        FROM Fakulta
        GROUP BY kat, funkce;
```

Kat	funkce	COUNT(*)	AVG(plat)
401	prof	3	8700
401	doc	7	7800
401	asist	12	6500
402	prof	2	16500

♦

Pokud pracujeme se skupinami a chceme formulovat podmínku pro celou skupinu, nejen pro jednotlivé řádky původních tabulek, nedává se tato podmínka za WHERE, ale za HAVING:

SELECT {seznam_výrazů | *}
FROM seznam_tabulek
[WHERE podm_pro_řádek]
[GROUP BY seznam-klíčů
[HAVING podm_pro_skupinu]]

Příklad 4.45.

Najděte průměrný plat skupin na fakultě, které mají více než 25 členů.

```
SELECT kat, funkce, AVG(plat)
        FROM Fakulta
        GROUP BY kat, funkce
        HAVING COUNT(*) > 25; ♦
```

Příklad 4.46.

Najděte katedry, kde jsou alespoň 2 sekretářky.

```
SELECT kat
        FROM Fakulta
        WHERE funkce = 'sekretářka'
        GROUP BY kat
        HAVING COUNT(*) >= 2; ♦
```

Příklad 4.47.

Najděte katedry, kde je průměrný plat asistenta větší, než 5000.

```
SELECT kat, AVG(plat)
        FROM Fakulta
        WHERE funkce = 'asist'
        GROUP BY kat
        HAVING AVG(plat) > 5000; ♦
```

- **Úplná syntaxe příkazu SELECT**

Shrneme-li přidávané nepovinné klauzule příkazu pro vyhledávání informací, dostaneme

```
SELECT {ALL | DISTINCT } seznam_výrazů
        FROM tab[ tabn] [, ...]
        [ WHERE podm ]
        [ GROUP BY seznam_sloupců
          [ HAVING podm ] ]
        [ ORDER BY sl [ASC | DESC] [, ...] ]
```

Klauzule ALL platí implicitně (všechny řádky i duplicitní), DISTINCT odstraní duplicitní řádky.

Logické kvantifikátory

Některé složitější dotazy v relačním kalkulu vyžadují použití logických kvantifikátorů EXISTS a FORALL.

Existenční kvantifikátor v SQL je definován

```
SELECT . . .
    FROM . . .
    [WHERE [NOT] EXISTS (SELECT ...  FROM ...  [...]) ]
```

kde EXISTS(SELECT ...) = TRUE, je-li množina daná výrazem v závorkách neprázdná, jinak je FALSE;

Tentýž dotaz lze formulovat pomocí IN.
Všeobecný kvantifikátor v SQL neexistuje, využívá se vztahu:

FORALL x (P(x)) = NOT EXISTS x (NOT P(x))

Doporučený postup je
1. zapsat dotaz jako výraz relačního kalkulu včetně FORALL
2. pak jej mechanicky přepsat do SQL

Příklad 4.48.

Mějme relace Čtenář(id_čt, jméno,adresa),
Kniha (prir, ISBN, autor, název)
Rezer (id_čt, prir, datod).

Hledáme čtenáře s nějakou knihu rezervovanou (= existuje záznam s jejich id_čt v Rezer)

```
SELECT jméno
    FROM Čtenář
    WHERE EXIST (SELECT C.*
                 FROM Rezer R, Čtenář C
                 WHERE R.id_čt = C.id_čt)
```

♦

❑ Množinové operace

Z množinových operací dosud známe IN a NOT IN, kartézský součin, případně ALL, ANY.

Další operace jsou v SQL definovány takto, přičemž je často implementováno jen sjednocení:

Sjednocení

```
SELECT ... FROM ... [WHERE ... GROUP BY ... HAVING ... ]
UNION
(SELECT ... FROM ... [...])
[ORDER BY ...]
```

Příklad 4.49.

Abecední seznam strávníků v menze školy - seznam zaměstnanců i studentů.

```
SELECT jmeno FROM Student
UNION
(SELECT jmeno FROM Zam)
ORDER BY jmeno
```

Pozor na častou chybu záměny sjednocení a spojení! Zde není možno použít:

… FROM Student, Zam ... ♦

Průnik

```
SELECT ... FROM ... [...]
INTERSECT
(SELECT ... FROM ... [...]) [...]
```

Příklad 4.50.

Jmenný seznam zaměstnaných studentů na VŠB – průnik studentů a zaměstnanců.

```
SELECT jmeno FROM Student
INTERSECT
(SELECT jmeno FROM Zam)
```

Průnik se snadno nahradí projekcí a selekcí nad kartézským součinem. ♦

Rozdíl

SELECT ... FROM ... [...]
EXCEPT
(SELECT ... FROM ... [...]) [...]

Příklad 4.51.

Jmenný seznam studentů nezaměstnaných na VŠB – rozdíl studentů a zaměstnanců.

SELECT jmeno FROM Student
EXCEPT
(SELECT jmeno FROM Zam) ♦

❑ Podotázky

V SQL je možno dotazy řetězit, pro formulaci hlavního dotazu je možno použít výsledků dotazu jiného - poddotazu. Přípustné je použít podotázky v podmínce za WHERE podle následujících pravidel:

- pokud je výsledkem poddotazu jediná hodnota (relace o 1 řádku a 1 sloupci), pak kdekoliv místo hodnoty:
  výraz rel_oper (příkaz SELECT)
- je-li výsledkem poddotazu množina hodnot (relace o 1 sloupci):
  výraz [NOT] IN (příkaz SELECT)

Poddotazy je možno použít za klauzulí WHERE i v příkazech UPDATE a DELETE.

Příklad 4.52.

Hledáme včelaře patřící do organizace jako Kovář Karel (mají stejného předsedu).

1. buď dvěma dotazy s „ručním“ přenesením mezivýsledku

SELECT předs FROM Včelaři WHERE předs = ‘Kovář Karel’;

SELECT jméno, adresa FROM Včelaři WHERE předs = ‘ ...’;

kde ‘...’ je výsledek 1. dotazu, který si musíme zapamatovat a použít ve 2. dotazu,

2. nebo zřetězeně pomocí poddotazu

```
SELECT jméno, adresa
    FROM Včelaři
    WHERE předs=(SELECT předs
                 FROM Včelaři
                 WHERE jméno=‘Kovář Karel’); ♦
```

Příklad 4.53.

Najděte včelaře, kteří získali více medu než alespoň jeden včelař okresu Ústí (~ co má nejméně).

```
SELECT jméno, adresa
    FROM Včelaři
    WHERE medu > (SELECT MIN(medu)
                  FROM Včelaři
                  WHERE okres = ‘Ústí’); ♦
```

Příklad 4.54.

Najděte včelaře, kteří mají více včelstev než kterýkoliv včelař okresu Ústí (~ co má nejvíce).

```
SELECT jméno, adresa
    FROM Včelaři
    WHERE včel > (SELECT MAX{včel)
                  FROM Včelaři
                  WHERE okres = 'Ústí'); ♦
```

Příklad 4.55.

Najděte včelaře, kteří mají stejné množství včelstev a získali stejné množství medu jako včelař Kovář Karel.

```
SELECT jméno, adresa, včel, medu
    FROM Včelaři
    WHERE (včel, medu) = (SELECT včel, medu
                          FROM Včelaři
                          WHERE jméno = 'Kovář Karel'); ♦
```

Příklad 4.56.

Najděte funkce, které mají vyšší průměrný plat, než je průměrný plat docenta.

```
SELECT funkce, AVG(plat)
    FROM Fakulta
    GROUP BY kat
    HAVING AVG(plat) > (SELECT AVG(plat)
                        FROM Fakulta
                        WHERE funkce = 'doc'); ♦
```

❑ Pohledy

Pohled představuje **virtuální tabulku** - relaci přímo v databázi neexistující, ale definovatelnou jako výsledek některého příkazu SELECT. Může to být projekce či selekce existující tabulky či spojení několika existujících tabulek, může obsahovat i sloupce odvozené z existujících hodnot (virtuální sloupce) ap. Pohled se definuje (podobně jako skutečná tabulka) příkazem:

```
CREATE VIEW pohled AS
    SELECT {seznam_sloupců | *}
    FROM tab
    [WHERE podm]
    [GROUP BY seznam_sloupců]   [HAVING podm]
    [ORDER BY seznam_sloupců]
```

Dále se s pohledem pracuje jako se skutečnou tabulkou. Provedeme-li změny v pohledu, změní se i hodnoty v tabulce, z níž je pohled odvozen a naopak. Problémem je provedení změn ve virtuálních sloupcích, v pohledech setříděných atd., proto konkrétní implementace práci s pohledy omezují jen na některé funkce.

Příklad 4.57.

Pro konkrétní úlohu nás zajímá jen jméno, adresa a množství medu včelařů z Ostravy.

```
CREATE VIEW Ostraváci AS
    SELECT jméno, adresa, medu
        FROM Včelaři
        WHERE okres = 'Ostrava'; ♦
```

Příklad 4.58.

Pro jinou úlohu potřebujeme všechny údaje z obou tabulek Včelaři i med.

```
CREATE VIEW Všichni AS
        SELECT *
                FROM Včelaři V, Med M
                WHERE V.jméno = M.jméno; ♦
```

Při vytváření virtuálních sloupců zadáme jména nově vzniklých sloupců za názvem pohledu.

Příklad 4.59.

Potřebujeme tabulku ročních příjmů zaměstnanců.

```
CREATE VIEW Roční_příjem (jméno, měsíčně, ročně) AS
        SELECT jméno, plat, plat*12
        FROM Fakul; ♦
```

❑ Doplnění příkazů INSERT, UPDATE , DELETE

Pomocí příkazu INSERT je možno také naplňovat řádky tabulky hodnotami z jiné tabulky tak, že místo klauzule VALUES použijeme SELECT. Tímto příkazem zadáme řádky a sloupce nebo vypočtené hodnoty z jiných tabulek, které se mají do doplňované tabulky přenést.

```
INSERT INTO tab [seznam_sloupců]
    SELECT seznam_hodnot FROM tab2 WHERE podm ...
```

Příklad 4.60.

Do tabulky ostravských včelařů se opíší Včelaři z celostátní evidence:

```
INSERT INTO Ostr_Včelaři
        SELECT * FROM Včelaři WHERE okres = 'Ostrava'; ♦
```

Také u příkazů UPDATE a DELETE je možno za klauzulí WHERE používat všechny funkce, operátory, konstrukty, poddotazy jako u příkazu SELECT.

Příklad 4.61.

Prémii ve výši 20% platu dostali všichni zaměstnanci uvedení v seznamu Odměny (jmeno).

```
UPDATE Zam
        SET plat = plat * 1.2
        WHERE jméno IN (SELECT jméno FROM Odměny); ♦
```

Příklad 4.62.

Zaměstnanci ze seznamu NoZam (jmeno) *dostali výpověď.*

```
DELETE FROM Zam
        WHERE jméno IN (SELECT jméno FROM NoZam); ♦
```

❑ Příklady rozšířených možností SQL

Nové verze jazyka SQL obsahují mnohá rozšíření dosud probraných možností vyhledávání informací, jako jsou práce s nestrukturovanými atributy - s poli (array), nestrukturovaným fulltextem, prací s tezaurem apod. Seznámíme se s nimi až u konkrétních implementací. Zatím si uvedeme jen malé ukázky.

Příklad 4.63.

Neatomické atributy – pole, odhnízdění, poziční přístup

```
CREATE TABLE Články (id NUMBER(4),
                     autoři CHAR(15) ARRAY [20],
                     titul CHAR(100),
                     abstrakt FULLTEXT );

SELECT Z.id, A.jméno FROM Zprávy Z, UNNEST (Z.autoři) AS A(jméno) ...;

SELECT id, autoři[1],autoři[2] FROM Články WHERE ...; ♦
```

Příklad 4.64.

Je dáno schéma Zam(jméno, ..., vzdělání, jazyky), *hledáme vysokoškoláka se znalostí angličtiny a němčiny.*

```
SELECT jméno
    FROM Zam
    WHERE vzdělání CONTAINS RT(univerzita) AND
          jazyky CONTAINS 'angličtina' AND 'němčina' ♦
```

❑ Příkazy pro sdílení a ochranu dat

Již víme, že SQL neobsahuje jen prostředky pro práci s databází, ale i prostředky podporující správu databáze - příkazy pro přidělování a odebírání přístupových práv na různých úrovních různým uživatelům databáze.

V SŘBD, které tyto prostředky mají implementovány, platí: k tabulce, kterou uživatel vytvořil, má přístup jen on sám, pokud nezpřístupní svou tabulku jinému uživateli příkazem GRANT. Odebrat toto právo může příkazem REVOKE.

Příkaz pro přidělení přístupového práva má tvar

```
GRANT  tabulkové_právo
   ON  { tab | pohled }
   TO  { PUBLIC | sez_jmen_uživatelů }
```

kde jméno uživatele je příslušný login_name, PUBLIC znamená zveřejnění tabulky všem uživatelům. Tabulková práva vymezují jemně rozlišit přesné vymezení přístupu různých uživatelů k různým datům:

SELECT [(seznam_sloupců)]	jen čtení příslušných sloupců tabulky
INSERT	vkládat nové řádky do tabulky
UPDATE [(seznam_sloupců)]	modifikovat sloupce tabulky
DELETE	rušit řádky tabulky
ALTER	modifikovat strukturu tabulky
INDEX	vytvářet indexy nad sloupci tabulky
ALL	všechna výše uvedená práva

Odebrání přístupového práva se provede příkazem

```
REVOKE  tabulkové_právo
   ON  { tab | pohled }
   FROM { PUBLIC | sez_jmen_uživatelů }
```

Některé SŘBD rozšiřují přidělování a odebírání práv i na celou databázi.

Příklad 4.65.

Pro definovanou tabulku `Zam(RČ,jméno,adresa,plat,fce)` *povolte záznam údajů o zaměstnanci -* `RČ,jméno,adresa` *perzonalistce s loginem ABC20,* `plat,fce` *vedoucímu katedry DEF30, ostatním jen prohlížení* `jména,adresy,fce`*, správci databáze GHI40 všechna práva.*

```
REVOKE ALL ON Zam FROM PUBLIC
GRANT INSERT ON Zam TO ABC20
GRANT UPDATE(RČ, jméno, adresa) ON Zam TO ABC20
GRANT SELECT(RČ, jméno, adresa) ON Zam TO ABC20
GRANT UPDATE(plat, fce) ON Zam TO DEF30
GRANT SELECT(plat, fce) ON Zam TO DEF30
GRANT ALL ON Zam TO GHI40
```

♦

❑ Další možnosti SQL

SQL není jen dotazovací jazyk, ale obsahuje i příkazy další.

Probrali jsme jeho základní příkazy. Především obsahuje všechny potřebné příkazy JDD, definuje, modifikuje a ruší databázi, tabulky, indexy, pohledy. Dále obsahuje příkazy JMD, ukládá data do databáze, modifikuje je a ruší. Dotazovací jazyk reprezentuje příkaz SELECT se všemi svými variantami. SQL zahrnuje také příkazy sloužící správě databáze pro přidělování přístupových práv na různé úrovni různým uživatelům.

Ale to není všechno. SQL obsahuje i příkazy pro

- vytváření **modulů**,
- pro **řízení transakcí**,
- pro záznam některých **integritních omezení**,
- pro vytváření **hierarchických struktur** dat,
- pro práci se **systémovým katalogem** ap.

Seznámíme se s nimi až u konkrétních implementací.

❑ Vývoj a standardy SQL

- Vznik jazyka - jen dotazovací část, prototyp u IBM, Dr.Codd **1974 (Sequel)**
- Zprvu implementován živelně v různých SŘBD.
- První standardizace organizací ANSI - průnik existujících implementací **1986 (SQL86).**
- Rozšíření definičního jazyka o definice IO **1989 (SQL89)**.
- Rozšíření o nové datové typy, nové typy spojení, aplikační programování, další podpora transakcí, ... **1992 (SQL2 ⇒ SQL92)**
- Java **1998 (SQLJ)**
- Rozšíření o objektovou orientaci pro objektově-relační databáze, rekurzivní SELECT, triggery, procedury, ... **1999 (SQL3 ⇒ SQL99)**
- „Kontejner pro budoucí standardy“, temporální SQL, XML, ... **> 2000**

Většina současných implementací v SŘBD je na úrovni SQL92, méně SQL99. Některé implementace mají rozšíření i o prvky navíc proti standardům.

Význam definování a dodržování standardů

- přenositelnost aplikací v různých prostředích
- životnost aplikací v různých prostředích
- možnost společného přístupu v heterogenním prostředí
- jednodušší zaškolování

Shrnutí pojmů 4.2.3.

SQL a jeho význam. Příkazy jazyka pro definici dat v SQL.
Příkazy jazyka pro manipulaci s daty v SQL, základní tvar dotazu. Funkce, agregované funkce.
Grupování, výběrové podmínky pro grupy.
Poddotazy.
Pohledy
Přidělování a odebírání přístupových práv.
Vývoj a standardy SQL, význam standardů.

Otázky 4.2.3.

1. Které skupiny příkazů jazyka SQL jsme probrali?
2. Které příkazy SQL patří do JDD a proč?
3. Které příkazy SQL patří do JMD a proč?
4. Které příkazy SQL tvoří dotazovací jazyk proč?
5. Jaká je syntaxe úplného vyhledávacího příkazu SQL?
6. Co je výsledkem příkazu SELECT?
7. Jaké typy funkcí je možno používat v SQL?
8. Co je grupování a jak se s grupami pracuje?
9. Co je poddotaz a co je jeho výsledkem?
10. Co je pohled, co je jeho výsledkem a k čemu se používá?
11. Které příkazy jazyka SQL umožňují přidělování a odnímání přístupových práv uživatelům?

Úlohy k řešení 4.2.3.

1. Je dáno schéma relační databáze pro automatizaci provozu VEŘEJNÉ KNIHOVNY :

Knihy (ISBN, autor, název, země)
Exemplář (ISBN, PŘÍR, koupeno, cena)
Čtenář (číslo, jméno, adresa)
Výpůjčky (PŘÍR, číslo, půjčeno)
Rezervace (ISBN, číslo, rezervováno)

kde položky koupeno, půjčeno, rezervováno jsou datumové, ISBN je číslo titulu knihy (bez ohledu na počet kopií v knihovně), PŘÍR je přírůstkové číslo exempláře knihy, číslo je číslo průkazu čtenáře.

a) Vypište seznam jmen a adres čtenářů.

b) Vypište seznam čtenářů setříděný abecedně podle jmen.

c) Vypište knihy nakoupené po 20.3.1992.

d) Najděte autory, na jejichž některé tituly je rezervace před 31.12.92.

e) Najděte dvojice čtenářů se stejnou adresou.

f) Najděte všechny exempláře vydané na Slovensku a jejich cenu v SK.

g) Zjistěte celkový počet čtenářů knihovny.

h) Určete počet čtenářů, kteří mají rezervovanou nějakou knihu.

i) Určete počet titulů knihovny,které byly vydány na Slovensku.

j) Jaká je celková cena slovenských knih v knihovně ?

k). Určete pro každého čtenáře počet vypůjčených knih.

l). Najděte čtenáře, kteří mají půjčeno více než 5 knih.

m) Najděte všechny tituly vydané v SR, Rusku a MR.

n) Najděte čísla čtenářů, kteří mají současně zapůjčeny nějaké knihy a rezervovány nějaké tituly do konce roku 1992.

o) Najděte jména čtenářů, kteří mají rezervovánu knihu Babička.

4.2.4. Doménový relační kalkul

V doménovém relačním kalkulu jsou oborem hodnot jeho proměnných prvky domén (na rozdíl od n-tic prvků domén v n-ticovém kalkulu). Podle toho je modifikována také definice.

Definice 4.9.

Abeceda kalkulu obsahuje

- atomické konstanty;
- místo n-ticových proměnných jsou zavedeny doménové proměnné; ty nejsou strukturovány, odpadá tedy potřeba komponent proměnných,
- predikátové konstanty; predikáty příslušnosti k relaci jsou obecně n-ární dle stupně relace;
- predikátové operátory binární < <= > >= = <> , obecně *
- logické operátory a kvantifikátory OR AND NOT EXIST FORALL
- oddělovače ( )

Atomické formule doménového kalkulu jsou:

- R(x1,x2,...,xn), kde R je jméno relace stupně n, xi jsou buď doménové proměnné nebo atomické konstanty; tato atomická formule říká, že n-tice hodnot x1, ..., xn je prvkem relace R; Aby v této formuli nebyl seznam proměnných závislý na pořadí atributů, zapisuje se obvykle pro schéma relace R(A,B,C,...)
- Formule příslušnosti n-tice k relaci R(A:x1,B:x2,C:x3,...);
- x * y, kde x, y jsou doménové proměnné nebo atomické konstanty
  * je binární predikátový operátor.

Formule doménového kalkulu se vytváří stejně, jako u n-ticového kalkulu.

Definice 4.10.

Výraz doménového relačního kalkulu je výraz tvaru

x1 x2 ...xn WHERE F(x1,x2,...,xn)

kde x1, ... , xn jsou jediné navzájem různé volné doménové proměnné ve formuli doménového relačního kalkulu F.

Výraz relačního doménového kalkulu určuje relaci tvořenou všemi možnými n-ticemi hodnot proměnných x1, x2, ..., xn, které splňují formuli F.

Příklad 4.63.

Opět jsou dány relace Učitel (ČU, jméno, kat, plat), Úvazek (ČU, ČP, hodin). *Formulujte požadavky na*

a) seznam čísel učitelů a jejich počtu hodin v jednotlivých předmětech (úplný výpis relace)

xČU,xČP,xhodin WHERE úvazek(ČU: xČU,ČP:xČP,Hodin:xhodin)

b) jmenný seznam učitelů (projekce)

x.jméno WHERE EXIST z (učitel(ČU:z,Jméno:xjméno,Kat:z,Plat:z))

c) seznam čísel učitelů, kteří učí předmět P2 (selekce a projekce)

xČU WHERE EXIST z (úvazek(ČU:xČU,ČP:z,Hodin:z)) AND ČP='P2'

d) seznam učitelů a jejich úvazků v hodinách (přirozené spojení)

xču,xjm,xkat,xpl,xčp,xhod WHERE (učitel(ČU:xču,Jméno:xjm,Kat:xkat,Plat:xpl) AND úvazek(ČU:xču,ČP:xčp,Hodin:xhod))

♦

Poznámka: Pomocí existenčního kvantifikátoru doplňujeme do n-tice ty proměnné, které se nevyskytují v seznamu na levé straně; když zavedeme jinou notaci, můžeme existenční kvantifikátor vynechat: do formule znamenající příslušnost prvku k relaci zapíšeme jen volné proměnné, ostatní proměnné vázané existenčním kvantifikátorem vynecháme a chápeme existenční kvantifikaci implicitně.

Obdobně jako u n-ticového kalkulu se definují i zde bezpečné výrazy doménového relačního kalkulu.

Věta o ekvivalenci

1. Ke každému výrazu relační algebry existuje ekvivalentní bezpečný výraz n-ticového relačního kalkulu.
2. Ke každému bezpečnému výrazu n-ticového relačního kalkulu existuje ekvivalentní bezpečný výraz doménového relačního kalkulu.
3. Ke každému bezpečnému výrazu doménového relačního kalkulu existuje ekvivalentní výraz relační algebry.

Na základě doménového relačního kalkulu je vytvořen jazyk QBE.

Shrnutí pojmů 4.2.4.

Abeceda, formule doménového relačního kalkulu.
Výraz doménového relačního kalkulu.

Otázky 4.2.4.

1. Definujte doménový relační kalkul.
2. Jak se liší n-ticový a doménový relační kalkul?
3. Jaký význam má formule relačního kalkulu?
4. Jaký význam má výraz relačního kalkulu?

Úlohy k řešení 4.2.4.

1. Zapište pomocí doménového kalkulu dotazy a) – l) z úlohy 4.2.3./1.

4.2.5. Jazyk QBE

Jazyk QBE (Query By Example) byl původně vytvořen u firmy IBM v roce 1975 pro pohodlné **zadávání výběrových podmínek pro naivní uživatele**. Postupně se z něj vytvořil standard, užívaný u řady SŘBD, podobně jako jazyk SQL. Svými schopnostmi formulovat výběrové podmínky je stejně silným prostředkem, jako je jazyk SQL. V současnosti je nějaká jeho varianta součástí téměř všech SŘBD.

Také již není jen dotazovacím jazykem, ale obsahuje prostředky pro definici a manipulaci s daty, pro práci s pohledy, se systémovým katalogem ap. Je implementován v řadě SŘBD.

V tomto odstavci si uvedeme jen několik jednoduchých ukázek jeho dotazovacích možností.

Dotazy jsou vyjadřovány interaktivně pomocí příkladů (odtud název jazyka). Tabulky, z nichž se mají informace čerpat, se formou prázdných tabulkových schémat nebo formulářů zobrazují na obrazovce. Dotaz se zapíše tak, že do příslušných sloupců prázdné tabulky se vypíší ty hodnoty, které se ve sloupcích hledají.

Základní použití jazyka si ukážeme na příkladech.

Příklad 4.64.

Mějme tabulku zaměstnanců s osobním číslem a dalšími údaji

Zam(osob, jméno, adresa, plat).

Po volbě tabulky se zobrazí prázdný řádek tabulky.

Zam	osob	jméno	adresa	plat

♦

Do řádků tabulky se zapisují příkazy QBE, proměnné a výběrové podmínky.

Proměnná začíná podtržítkem: _jmeno, _plat, _novak, _x, _y

Na rozdíl od SQL jazyk QBE podporuje odstranění duplicitních řádků výsledné relace. Zajištění všech řádků se naopak zajistí klauzulí ALL.

1. Výpis všech sloupců **celé tabulky** se zadá znakem P. (= print) pod jméno relace.

Příklad 4.65.

Vypište úplný seznam zaměstnanců.

Zam	osob	jméno	adresa	plat
P.				

♦

2. Projekce (výběr sloupců) se zapíše znakem P. do příslušného sloupce a vyjádřením proměnné: P._Novák

Příklad 4.66.

Vypište jména a adresy všech zaměstnanců.

Zam	osob	jméno	adresa	plat
		P.ALL._Novák	P._Opava	

_Novák *je proměnná, tedy stačí* _x, *užíváme různé proměnné v různých sloupcích*

Zam	osob	jméno	adresa	plat
		P.ALL._x	P._y	

Prostou projekci můžeme někdy zapsat i bez proměnné (záleží na implementaci)

Zam	osob	jméno	adresa	plat
		P.		P.

♦

3. Selekce (výběr řádků) se zapíše hledanými hodnotami ve sloupcích s případným použitím relačního operátoru.

Příklad 4.67.

Vypište jména zaměstnanců bydlících v Opavě

Zam	osob	jméno	adresa	plat
		P._x	Opava	

♦

4. Selekce a projekce současně.

Příklad 4.68.

Vypište osobní čísla a jména zaměstnanců s platem menším než 3500.

Zam	osob	jméno	adresa	plat
	P.	P.		<3500

♦

5. Konjunkce více podmínek se píše do řádku vedle sebe.

Příklad 4.69.

Najděte zaměstnance mimo Ostravu s platem větším než 10000.

Zam	osob	jméno	adresa	plat
		P.	<>Ostrava	>10000

♦

Složitější podmínky, jako více podmínek na jeden sloupec, disjunkce podmínek a další implementují různé SŘBD různě.

Příklad 4.70.

Najdi jména a osobní čísla zaměstnanců s platem 3000 - 4000.

6.Konjunkce dvou a více podmínek **na jeden sloupec** se buď

- píše do speciálního okna na obrazovce (CONDITION BOX),
- nebo existuje možnost rozšířit tabulku o duplicitní sloupce:

Zam	osob	jméno	adresa	plat	plat
	P.	P.		>3000	<4000

- nebo se seznam podmínek píše na oddělený čárkou.

Zam	osob	jméno	adresa	plat
	P.	P.		>3000,<4000

♦

7. Disjunkce podmínek se píše do dvou a více řádků tabulky s použitím různých proměnných.

Příklad 4.71.

Najdi zaměstnance s platem pod 3000 nebo nad 30000.

Zam	osob	jméno	adresa	plat
		P._x		<3000
		P._y		>30000

♦

8. Vzestupné **setřídění** výstupní relace se předepíše AO(n), sestupné DO(n) a v závorce zapsané číslo n znamená pořadí třídicího klíče.

Příklad 4.72.

Vypište zaměstnance dle výše platu - od nejvyššího k nejnižšímu, při stejném platu dle abecedy.

Zam	osob	jméno	adresa	plat
	P.	P.AO(2)		P.DO(1)

♦

9. Pro dotaz vyžadující **spojení** dvou tabulek se vyvolají formuláře obou tabulek a podmínka spojení se vyjádří použitím stejné proměnné v obou tabulkách.

Příklad 4.73.

Je dána další relace Děti (jménod, rodcis, osob). Osob *je cizí klíč ze* Zam. *Vypište adresy všech dětí (= adresy rodičů).*

Zam	osob	jméno	adresa	plat
	_x		P._z	

Děti	jménod	rodcis	osob
	P._y		_x

♦

10. Agregace používají funkcí CNT, SUM, AVG, MIN, MAX, **grupování** pomocí G.

Zam	kat	osob	jméno	adresa	plat
	P.G.				SUM._x

♦

Poznámka: Různé SŘBD mohou zobrazovat prázdný řádek tabulky i v jiném tvaru, často jako formulář.

Příklad 4.74. *Zaměstnanci z Opavy s platem nad 3000 se zadají:*

Zaměstnanci	
Osobní číslo:	
Jméno:	
Adresa:	**Opava**
Plat:	**> 3000**

♦

Dále existují možnosti formulace ještě složitějších podmínek, výpočtu agregovaných hodnot, pohledů, práce se systémovým katalogem, prostředků pro definici dat ap. Každý SŘBD má implementovánu jinou množinu prostředků a je nutné se vždy znovu s nimi a jejich zápisem seznámit.

Shrnutí pojmů 4.2.5.

Dotazy pomocí jazyka QBE. Selekce, projekce, spojení, setřídění, agregace v QBE.

Otázky 4.2.5.

1. Co znamená QBE a jak tento jazyk vznikl?
2. Jaký je základní rozdíl mezi dotazem v SQL a QBE?
3. Jak se v jazyce QBE provede výběr informací odpovídající operacím projekce, selekce, spojení?
4. Jak se v jazyce QBE provede setřídění a agregace informací?

Úlohy k řešení 4.2.5.

1. Zapište pomocí jazyka QBE dotazy a) – l) z úlohy 4.2.3./1.

4.3. Datová analýza schématu relační databáze

Cíl Po prostudování této kapitoly budete vědět, že

- Datovou analýzu je možno provádět pomocí RMD s jistotou správného výsledku

Výklad

❑ Datová analýza pomocí relačního datového modelu (RDM)

Vytvoření konceptuálního schématu na základě intuitivního návrhu entit, jejich atributů a vazeb mezi entitami není jediným způsobem, jak navrhnout schéma relační databáze. Ukážeme si, že i entity s malým počtem atributů je možno navrhnout špatně, s redundancemi - pokud neznáme pravidla, která nám umožní rozpoznat, co je dobře nebo špatně navržené schéma relace.

Současně se vznikem teorie relačního modelu dat vznikla také metoda návrhu relačních schémat, založená na jiných principech, než intuitivním návrhem konceptuálního modelu, a to pomocí funkčních závislostí.

Podívejme se na úlohu návrhu struktury databáze takto: z reálného světa máme dánu množinu atributů, které chceme rozmístit do jednotlivých schémat relací. Názvy těchto relací (názvy typů entit) ani jejich počet předem neznáme. Úkolem je navrhnout schéma databáze bez redundancí.

Příklad 4.75.

Potřebujeme malou databázi s evidencí výuky a výsledků zkoušek studentů. Představme si relaci **R** (Přednáška, Učitel, Místnost, Hodina, Student, Známka).

Prvkem tohoto schématu je přednáška, učitel, který ji přednáší v uvedené místnosti a uvedenou hodinu, student, který přednášce naslouchá a známka, kterou dostane po vykonání zkoušky z daného předmětu. Klíčem tohoto schématu je (Hodina,Student*).*

Předmět	Učitel	Místnost	Hodina	Student	Známka
TZD	Šarmanová	NK301	Po9	Novák	2
TZD	Šarmanová	E320	Út3	Novák	2
TZD	Šarmanová	NK301	Po9	Kouba	3
TZD	Šarmanová	E320	Út3	Kouba	3
OS	Olivka	E322	Po7	Zíka	1
OS	Olivka	E322	Po7	Tupý	2
OS	Olivka	E322	Po7	Redl	2
OS	Olivka	E322	Po7	Bílý	1
. . .					

Snadno si všimneme následujících nedostatků a potíží:

- **redundance**, pro každého studenta navštěvujícího přednášku se opakují hodnoty přednáška, učitel, místnost, hodina,
- nebezpečí vzniku **nekonzistence** při modifikacích jako důsledek redundance,

- **anomálie při vkládání** záznamů: nemůžeme vložit učitele, který nepřednáší, neboť by nebyly obsazeny klíčové atributy,
- **anomálie při vypouštění** záznamů: přestane-li učitel přednášet, vypustíme prvky tohoto učitele, tím ztratíme informaci i o jeho jménu.

Je zřejmě nutné schéma databáze změnit, atributy rozdělit do více relačních schémat, navrhnout několik typů entit a typů vazeb mezi nimi. V následujícím seznamu jsou uvedena různá relační schémata databáze, popisující stejnou realitu. Pro jednoduchost místo atributů píšeme jen jejich počáteční písmena.

Rozklad 1.
RO1 = { R1(P, U), R2(H, M, P), R3(H, U, M), R4(P, S, Z), R5(H, S, M) }
kde
R1 zaznamenává, kdo (který učitel) co (který předmět) učí,
R2 je rozvrh hodin podle předmětů – kdy a kde se předmět učí,
R3 je rozvrh hodin podle učeben,
R4 zaznamenává výsledky zkoušek studentů v jednotlivých předmětech
R5 rozvrh hodin podle studentů

Další možné rozklady si již pojmenujete sami:
RO2 = { R1(P ,U), R2(H, M, P), R3(P, S, Z), R4(H, S, P) }
RO3 = { R1(P, U), R2(H, S, P), R3(P, S, Z), R4(H, S, M) }
RO4 = { R1(H, M, P, U), R2(P, S, Z), R3(H, S, M) }
RO5 = { R1(P, U), R2(H, S, M), R3(P, S, Z), R4(H, M, P) }
RO6 = { R1(P, U), R2(H, M, P), R3(H, S, M) }
RO7 = { R1(P, S, U, H, M), R2(P, S, Z) }

Když si všechna schémata přečteme a pojmenujeme, vidíme, že všechna schémata jsou smysluplná.

Otázkou tedy je, čím se vlastně od sebe liší a je-li některé z nich lepší než ostatní.

♦

❑ Postup při datové analýze v RDM

Jazykem RDM můžeme formulovat problém takto:

1. Ze **zadání** určíme popis modelované reality a potřebu evidence všech atributů sledovaných objektů {A1, A2, ...}
2. V rámci **datové analýzy** pak
   - Vytvoříme jediné **univerzální schéma** se všemi atributy

     RU (A1, A2, ...)
   - Popíšeme existující vztahy mezi atributy – funkční závislosti, tj. pro nás nové IO:

     F (f1, f2, ...)
   - Pomocí obou množin a pomocí vhodného algoritmu vytvoříme schéma databáze s dobrými vlastnostmi, bez redundancí:

     RU (A1, A2, ...) ⇒ **RO** {**R1** (A1, ...), **R2** (Ai, ...), ... }

K tomu si však musíme nejprve vysvětlit, co jsou funkční závislosti, jak se provádí rozklad relačního schématu na více schémat a konečně, jak se poznají „dobré vlastnosti“ relačních schémat. Potom můžeme popsat postupy, jak se k takovému výsledku dostat.
univerzálního schématu.

4.3.1. Funkční závislosti

Čas ke studiu: 3 hodiny

Cíl Po prostudování této kapitoly budete umět

- Definovat funkční závislosti podmnožin atributů
- Formulovat z požadavků na databázi platné funkční závislosti
- Odvozovat funkční závislosti ze zadaných, vytvořit jejich uzávěr a minimální pokrytí, najít neredundandní a minimální pokrytí

Výklad

Zdrojem informací pro upřesnění sémantiky schématu databáze jsou integritní omezení (IO). Čím více jich budeme mít k dispozici, tím lépe můžeme provést návrh schématu. Mimo dosud známá IO zavedeme nový pojem funkční závislosti (FZ). Ty nám pomohou lépe pochopit zdroj možných redundancí a definovat pravidla pro rozpoznání dobře navržených relačních schémat.

Definice 4.11.

Nechť **R**({A1,A2,...,An}, f) je relační schéma, nechť X, Y jsou podmnožiny množiny jmen atributů {A1,A2,...,An}. Řekneme, že **Y je funkčně závislá na X**, píšeme $X \rightarrow Y$, když pro každou možnou aktuální relaci R(A1,A2,...,An) platí, že mají-li libovolné dva prvky (= dva řádky) relace R stejné hodnoty atributů X, pak mají i stejné hodnoty atributů Y. Je-li $Y \subset X$ říkáme, že závislost $X \rightarrow Y$ je **triviální**.

Z definice plyne, že

- funkční závislosti jsou definovány mezi dvěma podmnožinami atributů v rámci jednoho schématu relace. Jde tedy o vztahy mezi atributy, nikoliv mezi entitami,
- funkční závislost je definována na základě všech možných aktuálních relací, není tedy možné soudit na funkční závislost z vlastností jediné (třeba aktuální) relace. Tak můžeme poznat pouze neplatnost funkční závislosti,
- funkční závislosti jsou tvrzení o reálném světě, o významu atributů nebo vztahů mezi entitami (jako každé IO), je nutné je brát v úvahu při návrhu schématu databáze.

Příklad 4.76.

Je dána aktuální relace dle relačního schématu z minulého příkladu.Podívejme se podrobněji na obsah tabulky:

Všimneme si vztahu mezi podmnožinami X = {místnost, hodina} a Y = {předmět}. Vždy, když 2 řádky tabulky mají stejné hodnoty atributů X, mají i stejné hodnoty atributů Y. V tabulce jsou vyznačeny stejnou barvou, X světlejším, Y tmavším odstínem. Slovně to můžeme formulovat takto: v dané místnosti a v danou hodinu v týdnu se učí jediný předmět. Tedy existuje funkční závislost M H → P. *Tuto FZ známe obecně z reality.*

P		M H			
Předmět	**Učitel**	**Místnost**	**Hodina**	**Student**	**Známka**
TZD	Šarmanová	B5	Po9	Novák	2
TZD	Tichá	D117	Út3	Novák	2
TZD	Šarmanová	B5	Po9	Kouba	3
TZD	Tichá	D117	Út3	Kouba	3
OS	Lavička	B2	Po7	Zíka	1
OS	Lavička	B2	Po7	Tupý	2
OS	Lavička	E322	Po7	Redl	2
OS	Lavička	E322	Po7	Bílý	1
. . .					

V uvažované škole platí, že každý předmět přednáší jeden učitel (= zadané IO: P → U*).*

Celkem v příkladě můžeme určit tuto množinu funkčních závislostí F

$$F = \{ MH \rightarrow P, P \rightarrow U, HU \rightarrow M, PS \rightarrow Z, HS \rightarrow M\}$$

Z aktuální relace by se mohlo usuzovat na platnost funkční závislosti M → H*, ovšem obecně to zřejmě není pravda. Nelze tedy z jedné relace dokázat platnost funkčního vztahu.*

Naopak negativní fakta mohou být zjistitelná, protože tvoří protipříklad: není pravda PU → M*, protože* TZD *se učí ve dvou posluchárnách v týdnu.*

♦

V následujících definicích si zavedeme některé nové pojmy, které budeme dále používat a také upřesníme již používaný, ale dosud jen intuitivně zavedený primární klíč relace.

Definice 4.12.

Nechť F je množina funkčních závislostí pro relační schéma **R**, nechť X → Y je funkční závislost. Řekneme, že **F logicky implikuje X → Y**, jestliže v každé relaci R, v níž jsou splněny závislosti z F, je splněna i závislost X → Y. Množinu všech závislostí, které jsou logicky implikovány množinou F, nazýváme **uzávěrem množiny** F, označujeme F+.

Definice 4.13.

Nechť X, Y jsou podmnožiny atributů schématu **R** s množinou závislostí F. Říkáme, že **Y úplně závisí na X**, jestliže X → Y a pro žádnou vlastní podmnožinu X'⊂ X není X' → Y. Jinými slovy Y je funkčně závislá na X, ale není funkčně závislá na žádné vlastní podmnožině X.

Definice 4.14.

Nechť **R** ({A1, A2, ..., An}, f) je relační schéma s množinou funkčních závislostí F, nechť X⊂ {A1,A2,...,An}. Řekneme, že X je **klíč schématu R**, jestliže
$X \rightarrow A1...An \subset F+$
pro každou vlastní podmnožinu $Y \subset X$ je $Y \rightarrow A1...An \not\subset F+$.

K závislosti všech atributů na klíči jsme tedy přidali podmínku minimality. Zřejmě můžeme klíč schématu definovat také jako takovou $X \subset A$, že A je úplně závislá na X.

V relačním schématu může být více klíčů (nazýváme je obvykle „kandidáty na klíč“) a z nich vybereme nejvhodnější jako **primární klíč**. Ten se případně používá i jako cizí klíč v jiných tabulkách.

Definice 4.15.

Atribut relačního schématu **R** se nazývá **primární**, je-li podmnožinou alespoň jednoho klíče schématu **R**. Ostatní atributy nazveme **sekundárními**.

❑ Armstrongovy axiomy

K určení klíče relačního schématu a k hledání logických implikací množiny závislostí potřebujeme nalézt uzávěr F+, nebo určit, zda daná závislost $X \rightarrow Y$ je prvkem F+. K tomu existují odvozovací pravidla nazývaná Armstrongovy axiomy. Pravidla definují, jak se ze známých funkčních závislostí odvodí další platné závislosti.

Tato pravidla jsou **úplná** (dovolují odvodit z dané množiny závislostí F všechny závislosti patřící do F+) a **bezesporná** (dovolují z F odvodit pouze závislosti patřící do F+).

Nechť A je množina atributů daného relačního schématu, F množina funkčních závislostí mezi atributy A. V následujících pravidlech označujeme sjednocení $X \cup Y$ jako XY.

Armstrongovy axiomy jsou:

A1: jestliže $Y \subset X \subset A$, pak F logicky implikuje $X \rightarrow Y$ (reflexivita, triviální fční závislost)

A2: jestliže $X \rightarrow Y$ a $Z \subset A$, pak $XZ \rightarrow YZ$ (rozšíření)

A3: jestliže $X \rightarrow Y$ a $Y \rightarrow Z$, pak $X \rightarrow Z$ (tranzitivita)

A4: jestliže $X \rightarrow Y$ a $X \rightarrow Z$, pak $X \rightarrow YZ$ (sjednocení)

A5: jestliže $X \rightarrow Y$ a $WY \rightarrow Z$, pak $XW \rightarrow Z$ (pseudotranzitivita)

A6: jestliže $X \rightarrow Y$ a $Z \subset Y$, pak $X \rightarrow Z$ (zúžení)

A7: jestliže $X \rightarrow YZ$, pak $X \rightarrow Y$ a $X \rightarrow Z$ (dekompozice)

Důsledkem sjednocení a dekompozice je:

$X \rightarrow A_1 \ldots A_n$ právě tehdy, když $X \rightarrow A_i$ pro všechna i.

Příklad 4.77.

Určete klíč relačního schématu se závislostmi
R (Jméno, Katedra, Předmět, Hodin)
F = {Jméno → Katedra, Jméno Předmět → Hodin}.

Pro stručnost zapíšeme zadání i další odvozování opět jen prvními písmeny atributů:

Zadání: A = {J, K, P, H }, F = {J → K, JP → H }

Odvození klíče: 1. J → K *(dáno v F)*
2. JP → KP *(aplikace rozšíření na 1.)*
3. JP → H *(dáno v F)*
4. JP → KPH *(aplikace sjednocení na 2. a 3.)*
5. JP → JKPH *(aplikace reflexivity na 4.)*

Neplatí například J → P, P → K, *podle důsledků pravidel je* JP *minimální a tedy k*líč. ♦

Definice 4.16.

Závislost, která má na pravé straně pouze jeden atribut, nazýváme **elementární**.

Platí:

F lze nahradit závislostmi, které vzniknou dekompozicí pravých stran závislostí na jednotlivé atributy.

Je-li F' množina elementárních závislostí, které vzniknou z F uvedeným způsobem, platí

$$F+ = F'+$$

Z F' lze odstraňovat závislosti, které jsou odvoditelné ze zbytku F'.

Definice 4.17.

Říkáme, že **závislost f je redundandní** v F', jestliže platí

$$(F' - \{f\})+ = F'+$$

Definice 4.18.

Pokrytí množiny funkčních závislostí F je taková množina G funkčních závislostí, pro niž platí G+ = F+. **Neredundandní pokrytí** je takové pokrytí, které neobsahuje redundandní závislosti.

- Neredundandní pokrytí není dáno jednoznačně, závisí na pořadí, ve kterém odebíráme neredundandní závislosti. Obecně tedy nemusí být podmnožinou původní množiny F, pokud vycházíme z F+, ne z F.

Příklad 4.78.

Určete neredundandní pokrytí množiny funkčních závislostí F:

$$F = \{X\rightarrow Y, Y\rightarrow X, Y\rightarrow Z, X\rightarrow Z\}$$

Řešení:

1. Uzávěr celé množiny F *je* : F+ = { X→XYZ, Y→YXZ, Z→Z }

2. (F - {X→Y})+ = {X→XZ, ... } *... menší než F+, vyloučení závislost není redundandní,*

3. (F - {Y→X})+ = {X→XZY, Y→YZ ... } *... menší než F+,*

4. (F - {Y→Z})+ = {X→XZY, Y→YXZ, Z→Z ... } *... stejný jako F+, Y→Z je redundandní,*

5. (F - {X→Z})+ = {X→XZY, Y→YXZ, Z→Z ... } *... stejný jako F+, X→Z je redundandní,*

ale ne obě (Y→ Z i X→ Z) *současně.*

Výsledek tedy není jednoznačný, záleží na pořadí odebírání redundandních závislostí:

buď Fnered = { X → Y, Y → X, X → Z }
nebo Fnered = { X → Y, Y → X, Y → Z }

♦

Příklad 4.79.

Určete neredundandní pokrytí množiny funkčních závislostí F:

F = { AB→C, C→A, BC→D, ACD→B, D→EG, BE→C, CG→BD, CE→AG }

Řešení:

1. Nejprve upravíme F, aby obsahovala jen elementární závislosti

F': AB→C, C→A, BC→D, ACD→B, D→E, D→G, BE→C, CG→B, CG→D, CE→A, CE→G

Zde CE→A, CG→B *jsou redundandní, vyloučíme je v uvedeném pořadí a dostaneme výsledek*

F1: AB→C, C→A, BC→D, ACD→B, D→E, D→G, BE→C, CG→D, CE→G

2. *Jestliže zvolíme při odstraňování redundandních závislostí jiné pořadí* CE→A, CG→D, ACD→B, *obdržíme:*

F2: AB→C, C→A, BC→D, D→E, D→G, BE→C, CG→B, CE→G

♦

Při provádění dekompozicí univerzálního schématu R(A) se zadanou množinou funkčních závislostí F často není nutné znát celý uzávěr F+, ale stačí uzávěr podmnožiny atributů X ⊂ A vzhledem k F.

Definice 4.19.

Uzávěrem podmnožiny atributů X ⊂ A vzhledem k F nazveme množinu všech atributů funkčně závislých na X a označíme jej X+.

Definice 4.20.

Jestliže X→Y a pro nějaké C∈ X platí (X - C)+ = X+, říkáme, že **atribut C je redundandní** pro zadanou závislost.

Definice 4.21.

Pokrytí, v jehož závislostech neexistují žádné redundandní atributy, nazýváme **minimálním pokrytím**.

Význam minimálního pokrytí je v tom, že pro manipulaci s IO (např. testování jejich splnění při aktualizaci relací) jich má být co nejméně.

Příklad 4.80.

Je dáno schéma R(A,B,C,D,E). *Určete minimální pokrytí* Fmin *množiny funkčních závislostí*

F = {ABC→D, E→C, AB→E, C→D}

Řešení: A+ = {A}, B+ = {B}, C+ = {CD}, D+ = {D}, E+ = {ECD}

AB+ = {ABECD} … v ABC → D je C redundandní

Výsledek: Fmin = {AB→DEC, E→C, C→D}

♦

Platí:

- Při eliminaci redundandních atributů se nenaruší uzávěr množiny funkčních závislostí, z redukovaných závislostí je možno získat původní. Z redukovaných závislostí se také nedají získat jiné závislosti, než ty původní. Platí tedy

F+ = F1+ = F2+ = Fmin+

- Odstranění redundandních závislostí a redundandních atributů nelze provádět v libovolném pořadí. Pro získání minimálního pokrytí je nutno odstranit nejprve redundandní atributy a potom závislosti.

Při praktickém provádění dekompozice univerzálního schématu vidíme již na jednoduchých příkladech, že i při několika atributech a závislostech je nalezení minimálního pokrytí ručně obtížné. Pro usnadnění si uvedeme několik algoritmů, které některé operace návrhu usnadní. Všimněme si na nich, že pro určení příslušnosti závislosti X→ Y k uzávěru F+ není nutné spočítat celý uzávěr F+, ale stačí výpočet X+.

Algoritmus 4.1.

Určení uzávěru X+.

Vstup: množina funkčních závislostí F nad množinou atributů A relace R(A),
kde |F| = m, |A| = n, podmnožina X ⊂ A
Výstup: uzávěr X+
Struktura dat: LS[i], PS[i] jsou množiny atributů na levé a pravé straně funkčních závislostí F,
UZX obsahuje po skončení algoritmu množinu atributů X+.
Algoritmus:

```
begin
    UZX:= X;  POKR:= false;
    while (not POKR) do
      begin
        POKR:= true;
        for i:= 1 to m do
          begin
            if (LS[i] ⊂ UZX) and (PS[i] ⊄ UZX)
              then begin
                     UZX:= UZX ∪ PS[i];  POKR:= false
                   end
          end
      end
end;
```

Příklad 4.81.

Je dáno F = {A→B, AC→D, B→C}, určete A+, B+, AC+, AB+.

Řešení:
A+ = {A, C, B}
B+ = {B,C}
AC+ = {A,C,B,D}
AB+ = {A,B,C,D}

♦

Algoritmus 4.2.

Určení příslušnosti závislosti X→C k X+

Vstup: F nad množinou atributů A relace R(A), kde |F| = m, |A| = n, závislost X→C
Výstup: logická hodnota PRIS - příslušnost X→C k X+
Algoritmus:

```
begin
    urči UZX;
    if C ⊂ UZX then PRIS:= true
                else PRIS:= false
end;
```

Poznámka: Klíč schématu R(A) se zadanými F je možno určit tak, že určíme uzávěry různých levých stran všech závislostí nebo jejich částí. Klíčem jsou ty podmnožiny A, jejichž uzávěrem jsou všechny atributy z A.

Algoritmus 4.3.

Určení neredundandního pokrytí pro množinu elementárních funkčních závislostí.

Vstup: F' nad množinou atributů A relace R(A)
Výstup: neredundandní pokrytí G
Algoritmus:
```
begin
    G:= F';
    foreach f ∈ F do
        if f ∈ (G - {f})+ then G:= G - {f}
end
```

Příklad 4.82.

Je dáno relační schéma R(A, B, C, D, E, G) *a fční závislosti*
F={AB→C, C→A, BC→D, ACD→B, D→EG, BE→C, CG→BD, CE→AG}
Určete minimální neredundandní pokrytí množiny fčních závislostí F.

Řešení:

1. *Pomocí* ***uzávěrů podmnožin atributů*** *určíme* ***klíč schématu*** *a odtud* ***primární a sekundární atributy****:*
   A+={A}, B+={B}, C+={CA}, D+={DEG}, E+={E}, G+={G}
   AB+={ABCDEG}, BC+={BCADEG}, BE+={BECADG}, CG+={CGBDAE},
   CE+={CEAGBD}, AC+={AC}, AD+={ADEG}, AE+={AE}, ... , CD+={CDABEG}
   ACD+={ACDBEG}, ...
   výsledek1:
   klíče: AB, BC, BE, CE, CG, CD,
   nadklíče: ACD, ...
   A,B,C,D,E,G ... *primární atributy, žádný sekundární*
2. *upravíme* F*, aby obsahovala jen* ***elementární závislosti****:*
   výsledek2:
   F = {AB→C,C→A,BC→D,ACD→B,D→E,D→G,BE→C,CG→B, CG→D, CE→A, CE→G}
3. *určíme* ***minimální pokrytí****, tj. určíme redundandní atributy ve funkčních závislostech (využijeme uzávěry z 1.) a odstraníme je:*
   AB, BC, BE, CG, CE *nemají redundance*
   ACD→B ... *protože* CD+={CDABEG}, platí i CD → B, A *je redund.*
   výsledek3:
   Fmin={AB→C,C→A,BC→D,CD→B,D→E,D→G,BE→C,G→B,CG→D,CE→A, CE→G}
4. *určíme* ***neredundandní pokrytí****, tj. určíme a odstraníme redundandní závislosti z* Fmin:
   postupně testujeme závislosti,
   a) odstraníme CE→A, CG→B v tomto pořadí, dostaneme:
   Výsledek a: F1nered={AB → C, C → A, BC → D, ACD → B, D → E, D → G, BE → C,
   CG→D, CE→G}
   b) zvolíme jiné pořadí CE→A, CG→D, CD→B, *obdržíme:*
   Výsledek b: F2nered = {AB→C, C→A, BC→D, D→E, D→G, BE→C, CG→B, CE→G}

♦

Shrnutí pojmů 4.3.1.

Univerzální schéma databáze.
Funkční závislost mezi podmnožinami atributů relačního schématu. Triviální závislost.
Úplná funkční závislost.
Klíč relačního schématu, primární klíč. Primární a sekundární atributy.
Armstrongovy axiomy, odvozovací pravidla pro funkční závislosti. Elementární závislost.
Uzávěr funkčních závislostí. Pokrytí množiny funkčních závislostí.
Redundandní závislost množiny závislostí. Neredundandní pokrytí.
Redundandní atribut funkční závislosti. Minimální pokrytí.
Algoritmus pro určení uzávěru podmnožiny atributů.
Algoritmus pro určení příslušnosti dané funkční závislosti k uzávěru.
Algoritmus pro určení neredundandního pokrytí pro množinu elementárních funkčních závislostí.

Otázky 4.3.1.

1. Co je univerzální schéma relace?
2. Může databáze v databázové technologii obsahovat redundance? Proč?
3. Které problémy mohou vznikat, pokud schéma relační databáze obsahuje redundance?
4. Jaká je hrubá osnova postupu při datové analýze pomocí relačního datového modelu?
5. Definujte funkční závislost podmnožin atributů relačního schématu a uveďte příklady závislostí z reality.
6. Jaké jsou zdroje funkčních závislostí?
7. Co je úplná funkční závislost?
8. Co je klíč relačního schématu, jak se určí a kolik klíčů může jedno schéma mít?
9. Co je primární klíč relačního schématu?
10. Které atributy relačního schématu jsou primární a které sekundární?
11. Co jsou, jaké jsou a k čemu jsou Armstrongovy axiomy?
12. Definujte uzávěr funkčních závislostí. K čemu slouží?
13. Definujte pokrytí množiny funkčních závislostí. K čemu slouží?
14. Co je redundandní funkční závislost a co neredundandní pokrytí?
15. Co je redundandní atribut v rámci funkční závislost a co minimální pokrytí?
16. Formulujte algoritmus pro určení uzávěru podmnožiny atributů. K čemu slouží?
17. Formulujte algoritmus pro určení příslušnosti dané funkční závislosti k uzávěru. K čemu slouží?
18. Formulujte algoritmus pro určení neredundandního pokrytí pro množinu elementárních funkčních závislostí. K čemu slouží?

Úlohy k řešení 4.3.1.

Pro následující úlohy 1. až 4. určete schéma univerzální relace navrhované databáze, klíč univerzálního schématu a formulujte všechny funkční závislosti. Navrhněte strukturu databáze jako množinu relací ve 3NF nebo BCNF. Podtrhněte primární klíče relací, zakroužkujte cizí klíče.

1. Zubní lékař potřebuje evidenci svých pacientů (rodné číslo, jméno, adresu, pojišťovnu), výkonů na nich provedených (datum a druh výkonu), objednávky pacientů (datum a čas). Pro výkony existuje celostátní číselník (kód, název a cena výkonu). Pro jednoduchost předpokládejte, že za jednu návštěvu lékaře je proveden jeden výkon, současně je ošetřován nejvýše jeden pacient.
2. Navrhněte strukturu databáze pro infor. systém ABC soukromého zdravotnického střediska s několika lékaři. Je potřeba evidovat lékaře (osobní údaje a specializaci), pacienty (osobní údaje, pojišťovna), objednané pacienty a uskutečněné návštěvy u lékaře i lékařů u pacientů (datum a čas, diagnóza, cena pro pojišťovnu).
3. Firma potřebuje vést evidenci svých zaměstnanců (RČ, osob_číslo, jméno, adresa, fce, plat), jejich dětí (RČ, jméno, věk), seznam zakázek (dat_smlouvy, zakázka, popis, zákazník, cena_smluvní) a informace, kteří zaměstnanci pracují na kterých zakázkách (datum, kdo, na které zakázce, název_práce, cena_práce).
4. REZERVACE MÍST NA POČÍTAČOVÉ UČEBNĚ. Máme několik učeben, na každé je určitý počet počítačů. Každý počítač je jednoznačně identifikovatelný. Studenti si mohou rezervovat místo u konkrétního stroje vždy na 1 hodinu denně (počínaje celou hodinou). V každé chvíli musí být jednoznačně určeno, zda lze provést rezervaci na daný stroj v danou dobu, zjistit, na které učebně (u kterého stroje) byl, je nebo bude konkrétní student. Na každý den může mít student nejvýše jednu rezervaci.
5. Je dána relace R(W, X, Y, Z) a funkční závislosti F : $W \to Z$, $YZ \to X$, $WZ \to Y$.

   a) pomocí Armstrongových pravidel dokažte, že

   WZ+ = {W, X, Y, Z}, tj. WZ je nadklíč relace R,
   W+ = {W, X, Y, Z}, tj. W je klíč (nemá vlastní podmnožinu),
   YZ+ = {X, Y, Z}

   b) totéž dokažte pomocí algoritmu pro výpočet uzávěru;

   c) Dokažte, že neplatí $X \to YZ$.

6. Pro relační schéma R(W, X, Y, Z) jsou dány funkční závislosti
   $F=\{W\to Z, YZ\to X, WZ\to Y, W\to Y\}$.
   Ukažte, že
   a) $W\to Z$ není redundandní,
   b) $W\to Y$ je redundandní,
   c) eliminujte $W\to Y$ z F a pak ukažte, že $YZ\to X$ ani $WZ\to Y$ nejsou redundandní.
7. Zkonstruujte minimální pokrytí množiny $F=\{AB\to C, A\to D, D\to B\}$
8. Pro množinu funkčních závislostí $F=\{A\to B, B\to C, AC\to D\}$
   a) najděte A+
   b) najděte F+
9. Vytvořte dvě neredundandní pokrytí F1, F2 k následující množině funkčních závislostí eliminací redundandních závislostí

$$F=\{A\rightarrow B,\ B\rightarrow A,\ B\rightarrow C,\ A\rightarrow C\}$$

Dokažte dále, že $F3=\{A\rightarrow B,\ B\rightarrow A,\ AB\rightarrow C\}$ tvoří neredundandní pokrytí.

10. Vytvořte minimální pokrytí k následujícím množinám funkčních závislostí :
    a) $F1 = \{B\rightarrow D,\ E\rightarrow C,\ AC\rightarrow D,\ CD\rightarrow A,\ BE\rightarrow A\}$
    b) $F2 = \{A\rightarrow CDE,\ B\rightarrow CE,\ AD\rightarrow E,\ CD\rightarrow F,\ BD\rightarrow A,\ CED\rightarrow ABD\}$
    c) $F3 = \{D\rightarrow C,\ AB\rightarrow C,\ AD\rightarrow B,\ BD\rightarrow A,\ CA\rightarrow B\}$

11. Je dáno relační schéma $R(A, B, C, D, E, F)$ a množina funkčních závislostí
$$F = \{ABC\rightarrow DE,\ DE\rightarrow BC,\ DE\rightarrow F,\ AB\rightarrow D,\ E\rightarrow C\}.$$
Najděte klíče schématu R a klíčový atribut závislý částečně na druhém klíči schématu.

4.3.2. Dekompozice relačních schémat

Čas ke studiu: 1 hodina

Cíl Po prostudování této kapitoly budete vědět

- co je dekompozice relačního schématu, jaké má mít vlastnosti správná dekompozice a jak se správná dekompozice rozpozná.

Výklad

❑ Dekompozice relačního schématu

Již u jednoduchých příkladů jsme zjistili, že redundance a odtud další nepříjemné vlastnosti relačního schématu plynou z toho, že některé atributy jsou funkčně závislé nejen na klíči, ale i na jeho části. Této vlastnosti relačního schématu se zbavíme vhodným rozdělením relačního schématu, tzv. dekompozicí.

Definice 4.22.

Dekompozice relačního schématu R(A,f) je množina relačních schémat

$$\mathbf{RO} = \{\mathbf{R_1}(A_1,f_1),\ ...,\ \mathbf{R_k}(A_k,f_k)\},\quad \text{kde } A = A_1\cup A_2\cup \ldots \cup A_k.$$

V dalším výkladu se omezíme jen na **binární dekompozici**, tj. rozklad schématu na dvě schémata. Není to na újmu obecnosti, neboť obecnou dekompozici lze realizovat jako posloupnost binárních dekompozicí a pro nás se studium vlastností dekompozicí ulehčí.

Otázkou nyní je, jak dalece souvisí schémata získaná dekompozicí s původními schématy z hlediska jejich sémantiky a jak si jsou podobná data v původní a nové relační databázi. Intuitivně můžeme formulovat dvě pravidla:

1. výsledné relace by měly obsahovat stejná data, jaká by obsahovala původní databáze,
2. výsledná schémata by měla mít stejnou sémantiku,zadanou pomocí IO, která jsou v našem relačním přístupu vyjádřena funkčními závislostmi.

Vlastnosti vyjádřené oběma pravidly nyní upřesníme.

❑ Zachování informace při dekompozici

Definice 4.23.

Nechť R(A,f) je relační schéma, RO={R1(A1), R2(A2)} jeho rozklad, F množina funkčních závislostí. Řekneme, že při rozkladu **nedojde ke ztrátě informace** vzhledem k F (dekompozice je bezeztrátová), jestliže pro každou relaci R(A) splňující F platí:

$$R = R[A_1]\ [*]\ R[A_2]$$

Rozpoznat, zda přirozeným spojením rozložených relací dostaneme opravdu původní relaci (ne schéma, ale tytéž prvky relace), není jednoduché. Odpověď na otázku, jak poznat binární dekompozici, při níž nedojde ke ztrátě informace nám dává následující věta.

Věta 4.1. o ztrátě informace

Nechť RO = {R1(B),R2(C)} je dekompozice relačního schématu R(A), tedy A = B ∪ C a F je množina funkčních závislostí. Pak při rozkladu RO nedochází ke ztrátě informace vzhledem k F právě tehdy, když

$$B \cap C \rightarrow B - C \text{ nebo } B \cap C \rightarrow C - B.$$

Uvedené závislosti nemusí být v F, stačí, když budou v F+.

Příklad 4.83.

Pro relační schéma R(jméno, katedra, předmět, hodin*) s klíčem* {J, P} *je dána dekompozice* RO = {R1(J, K), R2(J, P, H)} *a funkční závislostí* F = {J → K}. *Je u tento rozklad bezeztrátový?*

Řešení:

{J, K} ∩ {J, P, H} = {J}, {J, K} - {J, P, H} = {K}
J → K … *patří do F+*

Tedy při rozkladu RO nedošlo ke ztrátě informace.

♦

Příklad 4.84.

Pro relační schéma R (jméno, katedra, předmět) *je dána dekompozice* RO = {R1(J,K), R2(K,P)} *a množina závislostí* F = {J → K}. *Je u tento rozklad bezeztrátový?*

Řešení:

{J,K} - {K,P} = {J}
{K,P} - {J,K} = {P} ⎫ K → J ⎫
{J,K} ∩ {K,P} = {K} ⎭ ⎭ K → P

Při rozkladu RO zřejmě dochází ke ztrátě informace vzhledem k F:*neboť ani* K → J, *ani* K → P *není prvkem F.*

♦

Příklad 4.85.

Mějme schéma R(A,B,C), *kde* A,B,C *jsou disjunktní podmnožiny atributů a funkční* závislost F ={B→C}. Otestujte bezeztrátovost rozkladů

RO1 = {R1(B, C), R2(A, B)}

RO2 = {R1(B, C), R2(A, C)}

Řešení:

Rozložíme-li R *na schémata* R1 (B, C) a R2 (A, B)*, je provedená dekompozice bezeztrátová, neboť platí:* {B,C} ∩ {A, B} → {B,C} – {A, B}

Naopak, je-li dekompozice R1 (B, C) a R2 (A, B) *bezeztrátová, musí platit buď* B → C *nebo* B → A.

Jiná dekompozice R1 (B, C) a R2 (C, A) *není bezeztrátová. Neplatí totiž ani* C→ B, ani C→A.

♦

❑ Zachování funkčních závislostí při dekompozici

Dalším důležitým požadavkem na proces rozkladu je zachování funkčních závislostí. Ty představují integritní omezení původní relace a v zájmu zachování reality musí být zachovány.

Definice 4.24.

Nechť R(A, f) je relační schéma, RO = {R1(B), R2 (C)} je jeho dekompozice s množinou závislostí F.

Projekcí F[B] množiny funkčních závislostí F na množinu atributů A nazveme množinu prvků X → Y z F+ takových, že X ∪ Y⊆ B.

Řekneme, že dekompozice RO **zachovává množinu funkčních závislostí F** (zachovává pokrytí závislostí), jestliže množina závislostí (F[B] ∪ F[C]) logicky implikuje závislosti v F, tedy

F+ = (F[B] ∪ F[C])+.

Příklad 4.13.

Uvažme relační schéma *ADRESA(Město, Ulice, PSČ)* se závislostmi *F = {MU → P, P → M}* a jeho rozklad *RO = {UP(U, P), MP(M, P)}*.

ADRESA

M	U	P
Brno	Mandloňová	621 00
Brno	Nezvalova	612 00
…		

UP

U	P
Mandloňová	621 00
Nezvalova	612 00

MP

M	P
Brno	621 00
Brno	612 00
…	

Při rozkladu RO

- *nedochází ke ztrátě informace vzhledem k daným závislostem, protože*

  {U,P}∩{M,P} → {M,P} - {U,P}

- *nezachovává množinu závislostí F:*

  projekce F[U, P] *obsahuje pouze triviální závislosti,*

  projekce F[M,P] *závislost* P → M *a triviální závislosti neimplikují* MU → P, *neboť* M,U,P *nejsou ve stejné relaci.*

To pak může vést k porušení IO: např. relace UP *a* MP *splňují závislosti odpovídající projekcím F, ale jejich přirozené spojení porušuje závislost* MU → P, *tj.jednoznačnost* PSČ *pro dané město a ulici.*

♦

Vedle rozkladů, při kterých nedochází ke ztrátě informace, ale nezachovávají danou množinu závislostí, existují i dekompozice, které zachovávají množinu závislostí, ale dochází při nich ke ztrátě informace.

Příklad 4.86.

Je dáno schéma R (A, B, C, D), *rozklad* RO = {R1 (A, B), R2 (C, D)} *se závislostmi* F = {A→B, C→D}. *Otestujte bezeztrátovost informace i zachování fčních závislostí.*

Řešení:

1. {A,B}∩ {C,D} = ∅
{A,B} - {C,D} = {A,B} ∅ → nic
{C,D} - {A,B} = {C,D}

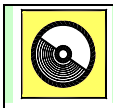
Rozklad není bez ztráty informace.

2. F[A, B] *obsahuje* A→B, F[C, D] *obsahuje* C→D

Rozklad zachovává množinu fčních závislostí..

♦

Uvedené vlastnosti dekompozicí použijeme nyní k takovým rozkladům, aby vzniklá relační schémata měla optimální vlastnosti.



CD-ROM

Na CD-ROMu s tímto výukovým textem jsou animované příklady na právě probrané provádění dekompozice. Soubory s animacemi jsou pojmenovány

- Animace\relacni model\dekomp.exe
- Animace\relacni model\dekomp_OK.exe
- Animace\relacni model\dekomp_ztrataINF.exe
- Animace\relacni model\dekomp_ztrataFZ.exe

Shrnutí pojmů 4.3.2.

Dekompozice relačního schématu, obecná dekompozice, binární dekompozice.
Vlastnosti správné dekompozice.
Bezeztrátovost informace.
Projekce funkčních závislostí na podmnožinu atributů schématu.
Zachování množiny funkčních závislostí.

Otázky 4.3.2.

1. Definujte dekompozici relačního schématu.
2. Proč se dekompozice relačního schématu provádí?
3. Jaké vlastnosti musí mít správná dekompozice?
4. Definujte bezeztrátovost informace při dekompozici relačního schématu a uveďte kritérium, podle kterého se správnost otestuje.
5. Definujte zachování funkčních závislostí při dekompozici relačního schématu a uveďte kritérium, podle kterého se správnost otestuje.

Úlohy k řešení 4.3.2.

1. Je dáno schéma R(A, B, C, D, E) a F = {A→B, BC→D, D→BC, C→A}

   a) Najděte klíč schématu R

   Jsou následující dekompozice správné ? Proč?

   b) RO = {R1(A, B), R2 (B, C, D)}

   c) RO = {R1(A, B), R2 (B, C, D, E)}

2. Je dáno relační schéma R (X, Y, Z, U, V) s funkčními závislostmi F = {XY→U, Y→V, U→Z} a rozkladem RO = {R1(X, Y, U), R2 (Y, Z, U, V)}. Je rozklad RO správný? Proč?

4.3.3. Normální formy relací

Čas ke studiu: 3 hodiny

Cíl Po prostudování této kapitoly budete vědět

- proč je nutné definovat pravidla pro rozpoznání správně definovaných relačních schémat
- co jsou a jaké jsou normální formy relačních schémat
- co je multizávislost podmnožin atributů a jak se odstraní

Výklad

❑ Pravidla pro definování dobrých vlastností relačních schémat

Již několikrát jsme se zmínili o pravidlech, pomocí nichž rozeznáme, zda jsou relační schémata navržena bez redundancí a tím i bez dalších špatných vlastností z redundance plynoucích – anomálií při vkládání a při rušení záznamů tabulky.

Tato pravidla jsou formulována pro relační schéma většinou pomocí funkčních závislostí platných nad atributy schématu. Nazýváme je normálními formami.

❑ První normální forma

Při návrhu relačních schémat jsme dosud používali **atributů atomických**. Pojem atomický atribut není přesně definován, myslí se jím atributy, jejichž domény jsou v jistém smyslu množiny jednoduchých hodnot, čísel, znaků, slov v nějaké abecedě ap., logicky dále nedělitelných.

Definice 4.25.

Jestliže relační schéma obsahuje pouze atomické atributy říkáme, že je normalizovanou relací nebo že je **v první normální formě** (1NF).

Platí: relační model dat pracuje pouze s relacemi v 1NF.

Poznámka: Relaci v 1NF dostaneme z relace s neatomickými atributy buď doplněním opakovaných hodnot u vícehodnotových atributů (pak relace obsahuje redundanci), nebo dekompozicí relace na více relací.

Nejsou-li všechny atributy atomické, převedeme je na (obvykle několik) tabulky s atomickými atributy těmito technikami:

1. atribut opakující se n-krát: n atributů vedle sebe (pozor na formulaci dotazů) nebo nová tabulka,
2. atribut opakující se ?-krát: nová tabulka s cizím klíčem,
3. složený atribut: rozklad na několik atomických atributů, odpadne název složeného atributu.

Příklad 4.87.

Převedení neatomických atributů do 1NF

1. plat se opakuje 12-krát (leden až prosinec), po úpravě odpadne atribut plat.

jméno	plat(m1, m2,… ,m12)

⇒

jméno	m1	m2	…	m12

2. dítě zaměstnance se opakuje proměnný počet-krát, po úpravě se děti umístí do nové tabulky s cizím klíčem.

jméno	funkce	…	dítě

⇒

jméno	funkce	…

+

jméno	dítě

3. adresa jako skupinový atribut, po úpravě odpadne atribut adresa.

jméno	adresa(ulice,obec,PSČ)

⇒

jméno	ulice	obec	PSČ

Příklad 4.88.

Analýzou úseku reálného světa je navrženo relační schéma Zaměstnanci *ve tvaru*

Zaměstnanci

vedoucí	čís_kat	pracovníci (jméno, dat_nar, plat, …) : multi			
Pták	231	Čáp	1.9.39	3400	…
		Kos	24.4.45	2900	
		Orel	12.12.56	3200	
		…			
		Skřivan	9.5.45	4500	
Ptáček	232	Holub	4.12.42	5500	
		Sova	4.7.85	8300	
		…			

Atribut pracovníci je vícehodnotový, není atomický a tak v jednotlivých řádcích tabulky ve sloupci pracovníci jsou opět celé tabulky, tedy relace. Doménou atributu pracovníci je množina všech relací se třemi pevně zvolenými atributy s doménami {jména}, {data narození} a {platy} zaměstnanců.

Pokud klíč takové nenormalizované relace je atomický, lze ji nahradit relací normalizovanou se stejným informačním obsahem. Triviální způsob normalizace je ovšem zaplacen redundancí, tu pak odstraňujeme dekompozicí.

Převedení tabulky do 1NF doplněním opakovaných hodnot a pak dekompozicí pro odstranění redundance:

Zaměstnanci

vedoucí	čís_kat	jméno	dat_nar	plat	…
Pták	231	Čáp	1.9.39	3400	…
Pták	231	Kos	24.4.45	2900	
Pták	231	Orel	12.12.56	3200	
Pták	231	…			
Pták	231	Skřivan	9.5.45	4500	
Ptáček	232	Holub	4.12.42	5500	
Ptáček	232	Sova	4.7.85	8300	
Ptáček	232	…			

po dekompozici:

Katedry

vedoucí	čís_kat
Pták	231
Ptáček	232
…	

Zaměstnanci

jméno	dat_nar	plat	…
Čáp	1.9.39	3400	…
Kos	24.4.45	2900	
Orel	12.12.56	3200	
…			
Skřivan	9.5.45	4500	
Holub	4.12.42	5500	
Sova	4.7.85	8300	
…			

♦

❑ Druhá normální forma

Definice 4.26.

Relační schéma je **ve druhé normální formě** (2NF), jestliže je v první normální formě a každý sekundární atribut je úplně závislý na každém klíči schématu R.

Příklad 4.89.

Je dána relace v 1NF FirmaP(firma, adresa, zboží, množ). *Je toto schéma ve 2NF?*

FirmaP

firma	adresa	zboží	množ
AA	XAA	zb1	500
AA	XAA	zb2	200
BB	XBB	zb2	600
BB	XBB	zb3	700
…			

⇒

Firma N

firma	adresa
AA	XAA
BB	XBB
…	

+

Výroba

firma	zboží	množ
AA	zb1	500
AA	zb2	200
BB	zb2	600
BB	zb3	700
…		

Řešení: Platí F = {firma→adresa, firma, zboží → množ}
Klíč původního schématu: {firma, zboží}
Protože platí také firma→adresa, *je* adresa závislá na podklíči
Důsledky: redundance adresy ⇒
když firma přestane dočasně vyrábět, tak nevíme ani, že existuje a kde sídlí firmu, která nevyrábí, není možno zapsat
Výsledek: schéma není ve 2NF ⇒ *rozklad na 2 tabulky do 2NF: RO = {FirmaN, Výroba}* ♦

Příklad 4.90.

Vezměme opět schéma Učitel(Jméno, Katedra, Předmět, Hodin) *s klíčem* (Jméno, Předmět) a *závislostí* {Jméno → Katedra}. *Tato relace je v 1NF, ale není ve 2NF, protože existuje atribut Katedra, který není primární a není úplně závislý na klíči* (Jméno, Předmět). *Rozklad* RO = {R1(J,K), R2(J,P,H)*} již je ve 2NF, jak se dá snadno ověřit.*

Důsledky: redundance katedry pro stejné jméno
učitel dosud neučí → není možno ho zapsat
učitel přestane učit → musí se zrušit i jeho jméno

♦

❑ Třetí normální forma

Schémata ve 2NF však mohou mít další typ anomálií, podobných předchozím, avšak z poněkud jiných příčin. Uvažme příklad:

Příklad 4.91.

Relační schéma Učitelé(ČU, Jméno, Plat, Funkce) *s jediným klíčem ČU je ve 2NF. Uvažme další, z reálného světa odpozorovanou, funkční závislost Funkce → Plat (výše platu je podle předpisu určena zastávanou funkcí učitele).Opět můžeme zjistit následující potíže*

- *redundance, plat je uváděn opakovaně pro každého učitele se stejnou funkcí,*
- *nebezpečí vzniku nekonzistence, plynoucí z redundance: že zapomeneme provést změny u všech prvků relace například při změně výše platu pro funkci;*
- *anomálie při vkládání; pokud žádný učitel není asistentem, nemůžeme ani vložit informaci o tom, jaký má asistent plat;*

- *anomálie při vypouštění; při vypuštění jediného profesora ztratíme i informaci o tom, jaký má profesor plat.*

Příčinou těchto anomálií u relací ve 2NF jsou opět jisté typy funkčních závislostí, tentokrát tranzitivní závislost sekundárního atributu na klíči přes jiný sekundární atribut. Zavedeme si definici tranzitivní závislosti a pak relací ve 3NF.

♦

Definice 4.27.

Nechť X,Y, Z jsou podmnožiny množiny atributů A schématu R, nechť platí $Y \not\subset X$, $X \cap Z = \emptyset$, $Y \cap Z = \emptyset$ a nechť F je množina závislostí. Jestliže $X \rightarrow Y \in F$, $Y \rightarrow Z \in F^+$ a $Y \rightarrow X \notin F^+$, pak také $X \rightarrow Z \in F^+$ a $Z \rightarrow X \notin F^+$ a říkáme, že Z je **tranzitivně závislá** na X.

Poznámka: V definici nejde o jakoukoliv tranzitivitu, ale podmínky definice mají tento význam: pro $X, Y, Z \subseteq A$ předpoklad $Y \not\subset X$ znamená, že se uvažují jen netriviální závislosti,

$X \cap Z = \emptyset$ -“-

$Y \cap Z = \emptyset$ -“-

$\neg (Y \rightarrow X)$ .znamená, že Y není jiný klíč v (X, Y, Z)

Definice 4.28.

Relační schéma je ve **třetí normální formě** (3NF), jestliže je ve 2NF a žádný sekundární atribut není tranzitivně závislý na žádném klíči schématu R.

Jiná formulace 3NF:

Relační schéma je ve **třetí normální formě**, jestliže je ve 2NF a neexistuje závislost mezi sekundárními atributy.

Relace ve 2NF, které nejsou ve 3NF, se mohou rozložit vhodnou dekompozicí do 3NF, přičemž nedochází ke ztrátě informace a dekompozice zachovává množinu závislostí.

Příklad 4. 92.

Je schéma FirmaP (firma, město, obyvatel) *ve 3NF?*

FirmaP

firma	město	obyvatel
AA	Ostrava	320000
BB	Karviná	100000
CC	Ostrava	320000
..		

$\Rightarrow$

firma	město
AA	Ostrava
BB	Karviná
…	

+

město	obyvatel
Ostrava	320000
Karviná	100000
…	

Platí: F = {firma → město, město → obyvatel}, odtud
Klíč: firma
Primární atributy: firma, *sekundární atributy:* město, obyvatel
Schéma je ve 2N, není ve 3NF..
Existuje závislost: město → obyvatel *tedy závislost mezi sekundárními atributy (neboli tranzitivní závislost sekundárního atributu na klíči);*
Důsledky: redundance hodnot obyvatel *pro stejné* město
zruší-li se firma, ztratíme i informaci o počtu obyvatel jejího města,
uložit počet obyvatel města bez firem nelze.

♦

Příklad 4.93.

Schéma Učitel (ČU, jméno, plat, funkce) se zadanou F = {..., funkce → plat} je ve 3NF?

Řešení:

Klíč je ČU, funkce *i* plat *jsou sekundární atributy, schéma je ve 2NF, není ve 3NF.*

♦

❑ Boyce - Coddova normální forma

Dalším omezením třídy relací ve 3NF - vyloučením všech netriviálních funkčních závislostí mezi atributy kromě závislostí na klíči - dostaneme relace v Boyce - Coddově normální formě.

Definice 4.29.

Relační schéma R(A) s množinou funkčních závislostí F je v **Boyce - Coddově normální formě** (BCNF), jestliže vždy, když X → Y a Y ∉ X, pak X obsahuje klíč schématu R.

Věta:

Každé schéma v BCNF je zároveň ve 3NF.

Důkaz: Jinak by existovala

- buď závislost Y → Ai, kde Y ⊂ Ai a Ai je sekundární atribut,
- nebo tranzitivní závislost X → Y → Ai, přičemž neplatí Y → X.

V obou případech Y neobsahuje klíč a to je ve sporu s tím, že schéma je v BCNF.

Poznámka: X → Y a Y ∉ X znamená, že X obsahuje klíč, tedy existují jen závislosti na klíčích.

Poznámka: Mějme schéma R (A,B,C,D,... , X, Y, Z, U, ...)

atributy primární sekundární

Pak normální formy znamenají, že ve schématu jsou závislosti:

2NF : sekundární atributy úplně závislé na klíčích, ne na podklíčích

3NF : sekundární atributy nejsou závislé na sekundárních

BCNF : existují jen závislosti na klíčích, tj. vyloučí i závislosti primární → primární

Příklad 4. 94.

Je schéma R (A, B, C, D) *se závislostmi* F ={A → B, C → D} *v BCNF?*

Řešení:

Klíč: A+ = {A, B}, B+ = {B}, C+ = {C, D}, D+ = {D},

AC+ = {A, C, B, D} ⇒ AC je klíč ⇒ R *není v BCNF*

♦

Příklad 4. 95.

Je správný rozklad RO = {R1 (A,B), R2 (C,D)} *schématu R z minulého příkladu?*

Řešení:

{A, B} ∩ {C, D} = ∅ … *rozklad není správný, není bezeztrátový.*

Jak dostaneme správný rozklad? V následující kapitole pomocí algoritmy syntézy.

♦

❑ Existence schémat v normálních formách

Obecně si nyní položíme otázku, zda existuje třída relačních schémat, která by byla oproštěna od všech uvedených anomálií (jde zřejmě o 3NF a BCNF).

Platí:

- existuje algoritmus dekompozice libovolného relačního schématu na schémata ve 3NF, přičemž tato dekompozice zachovává funkční závislosti původního schématu a nedochází při ní ke ztrátě informace;
- existuje také algoritmus dekompozice libovolného relačního schématu na schémata v BCNF, při nichž nedochází ke ztrátě informace; ovšem pro některá schémata neexistuje dekompozice na schéma v BCNF, která by zachovávala jejich množinu funkčních závislostí.

Příklad 4.96.

Uvažme opět schéma *ADRESA(Město, Ulice, PSČ)* s funkčními závislostmi z reality

$$F = \{město, ulice \rightarrow PSČ,\ PSČ \rightarrow město\}$$

(Neplatí ani U → P, ani $M \rightarrow P$.) Úkolem je převést toto schéma do BCNF.

Řešení:Zapíšeme si atributy ve zkrácené formě: Adresa (M, U, P), F = {MU → P, P → M}.

Klíč: M+ = {M}, U+ = {U}, P+ = {P, M}

MU+ = {M, U, P}, MP+ = {M, P}, UP+ = {U, P, M} ... *existují 2 klíče* MU *a* UP

Primární atributy: M, U, P, *sekundární nejsou.*

Schéma je ve 3NF, neboť neexistence sekundárních atributů nemůže porušit 2NF ani 3NF. Schéma ale není v BCNF, protože existuje závislost P → M *a ta není závislostí na klíči. Je to závislost mezi primárními atributy.*

Pokusíme se o rozklad.. Otestujeme rozklad RO = {UP (U, P), MP (M, P)}.

1. test na ztrátu informace: {UP} ∩ {MP} → {MP} - {UP}

P → M ... *rozklad v pořádku*

2. test na ztrátu fčních závislostí:

$$F+ = \{MU \rightarrow MUP,\ P \rightarrow PM\}$$

v UP *existují jen triviální závislosti, v* MP *navíc závislost* P → M, *v jejich sjednocení chybí* MU→P, *tedy daný rozklad* RO *nezachovává množinu fčních závislostí; jinak řečeno, přirozené spojení* UP [*] MP *porušuje* MU → P.

3. úkol je rozložit schéma tak, aby každá závislost byla pokryta v některé relaci, nelze, protože je tam závislost mezi primárními atributy;

Výsledek: schéma je ve 3NF, nelze do převést do BCNF.

♦

❑ Čtvrtá normální forma

Závěrem uvedeme ještě jiný typ závislosti, než funkční závislost, tzv. multizávislost.

Definice 4.30.

Je dáno relační schéma R(A) s množinou atributů A. Nechť X,Y jsou podmnožiny A. Řekneme, že **Y multizávisí na X**, když pro každou možnou aktuální relaci R schématu R(A) a pro každé dva prvky a, b relace R, pro které platí a[X] = b[X], existují prvky u, v relace R takové, že

1. a[X] = b[X] = u[X] = v[X]
2. u[Y] = a[Y] a u[A-X-Y] = b[A-X-Y]
3. v[Y] = b[Y] a v[A-X-Y] = a[A-X-Y]

Označujeme X →> Y.

Příklad 4.97.

Mějme relaci s katalogem automobilových modelů Auta (model, země_původu, počet_válců):

Auta

	model	země_původu	počet_válců
	Škoda-Felicia	ČR	4
*	Ford-Mondeo	USA	6
*	Ford-Mondeo	Kanada	4
**	Ford-Mondeo	USA	6
*	Ford-Mondeo	Kanada	4
	Renault	Francie	4
	…		

Zřejmě země_původu *multizávisí na* model*, ovšem tato multizávislost je triviální a nevede k žádným problémům při práci s relací* Auta.. *Atribut* počet_válců *je nezávislý na* země_původu*, ale multizávisí na* model*, ovšem nezávisle na* země_výroby. *Model Ford se vyrábí v provedeních 4 a 6 válců, oba modely se vyrábí jak v USA, tak v Kanadě. Ohvězdičkované řádky tabulky ukazují redundanci takového schématu. Problémy s aktualizací jsou zřejmé. Je tedy vhodné provést dekompozici na dvě schémata:*

Auta (model, počet_válců)
Odkud (model, země_původu)

Auta

model	země_původu
Škoda-Felicia	ČR
Ford-Mondeo	USA
Ford-Mondeo	Kanada
Renault	Francie
…	

+

Odkud

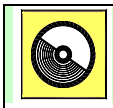
model	počet_válců
Škoda-Felicia	4
Ford-Mondeo	6
Ford-Mondeo	4
Renault	4
…	

Jakmile však přestanou v USA vyrábět Fordy se 6 válci, zruší se řádek označený (**) a dekompozici nemůžeme provést, protože bychom tuto informaci ztratili. Existuje sice nadále jakási souvislost mezi modely, zemí a počtem válců, nechápe se však v tomto případě jako multizávislot. Pojem multizávislosti X na Y tedy závisí na kontextu tvořeným zbylými atributy, tj. A-X-Y.

♦

Definice 4.31.

Pokud je relace ve 3NF a neobsahuje žádné multizávislosti, říkáme, že je ve **čtvrté normální formě** (4NF).



CD-ROM

Na CD-ROMu s tímto výukovým textem jsou animované příklady na právě probrané převedení nižší NF na vyšší. Soubory s animacemi jsou pojmenovány

- Animace\relacni model\1NFna2NF.exe
- Animace\relacni model\2NFna3NF.exe

Shrnutí pojmů 4.3.3.

Normální formy relačních schémat.
1NF, 2NF, 3NF, BCNF.
Multizávislost, 4NF.
Existence optimálních relačních schémat.

Otázky 4.3.3.

1. Co jsou normální formy relačních schémat obecně?
2. Definujte postupně 1NF, 2NF, 3NF, BCNF a na každou normální formu uveďte příklad schématu, které ji splňuje.
3. Na každou normální formu uveďte příklad schématu, které ji nesplňuje a přitom splňuje formy nižší.
4. Vysvětlete multizávislost podmnožin atributů relačního schématu a uveďte příklad.
5. Definujte 4NF a uveďte příklad schématu, které ji splňuje.
6. Definujte 4NF a uveďte příklad schématu, které ji nesplňuje a obsahuje multizávislost.

Úlohy k řešení 4.3.3.

1. Jsou dána relační schémata **R** s funkčními závislostmi F. Určete klíč schématu, primární a sekundární atributy, ve které normální formě schéma je a zdůvodněte, proč:

   a) **R**(A, B, C, D), F = {A → C, AC → B, B → D}

   b) **R**(A, B, C, D, E), F = {C → E, D → AB, C → E}

   c) **R**(A, B, C, D, E), F = {BCD → A, A → E, C → D}

   d) **R**(A, B, C, D, E), F = {DE → A, ED → BC}

   e) **R**(A, B, C, D, E), F = {AB → CD, C → E}

4.3.4. Algoritmy návrhu schématu relační databáze

Čas ke studiu: 3 hodiny

Cíl Po prostudování této kapitoly budete umět

- navrhnout dvěma způsoby optimální schéma databáze bez redundancí

Výklad

❑ Předpoklady pro algoritmy návrhu struktury databáze

Závěrem si popíšeme dva algoritmy návrhu schématu relační databáze. V obou případech jde vlastně o dekompozici univerzálního schématu relace do 3NF nebo BCNF.

Jsou to

- **algoritmus dekompozice** relačních schémat (též shora dolů, postupně nahrazuje jedno schéma dvěma)
- **algoritmus syntézy** relačních schémat (též zdola nahoru, syntézou přímo z funkčních závislostí).

Pro správnou funkci algoritmů musí být splněny následující předpoklady:

- předpoklad **schématu univerzální relace**; předpoklad se týká množiny atributů rozkládané relace. Atributy musí mít jednoznačná jména a každé jméno atributu musí mít pouze jeden význam. Existuje-li ve více relacích stejné jméno atributu, mají všechny tyto atributy stejnou doménu.

  Příklad 4.98. *je-li atribut* známka *vyhrazen pro známku studenta u zkoušky, nesmí se použít např. také pro hodnocení kvalifikace učitele.*

- předpoklad **jednoznačnosti vztahů**; tento předpoklad říká, že pro každou podmnožinu atributů X existuje nejvýše jeden typ vztahu. Jinými slovy ke každé podmnožině atributů univerzálního schématu přísluší nejvýše jeden typ vztahové entity.

  Příklad 4.99. vazby *VEDOUCÍ_PROJEKTU (Učitel, Student) a UČÍ (Učitel, Předmět, Student)* nesplňují tento předpoklad, neboť vedoucí projektů a přednášející jsou ve dvou různých vztazích ke studentům.

  Předpoklad jednoznačnosti vztahů sice poněkud omezuje volnost v modelování, jednoznačnost však nemusí být řešena později na databázové úrovni.

Cílem našeho návrhu bude databázové schéma ve 3NF nebo BCNF, ve kterém nedochází ke ztrátě informace vzhledem k zadané množině funkčních závislostí F a které zachovává množinu závislostí F. Ne vždy lze obě tyto vlastnosti jednoduše splnit.

Uvedeme si nyní první algoritmus dekompozice. Algoritmus bude provádět binární dekompozice schématu R tak dlouho, až budou výsledná schémata v požadované BCNF. Modifikací bychom dostali algoritmus pro získání databáze ve 3NF.

❑ Algoritmus dekompozice

Algoritmus pro konstrukci relačního schématu databáze v BCNF dekompozicí.

U výsledného schématu nedochází ke ztrátě informace, ovšem nemusí být zachována množina závislostí. Nevýhodou algoritmu je nutnost výpočtu F+, který může být velmi (exponenciálně) velký vzhledem k A.

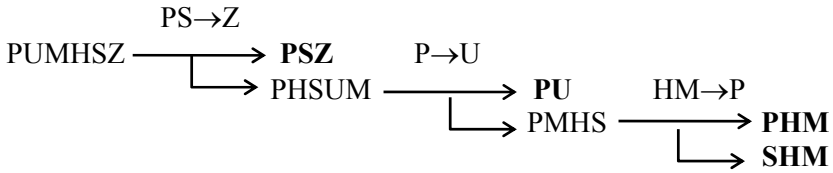
Vstup: Univerzální relační schéma R(A) stupně n s množinou funkčních závislostí F.
Výstup: Relační schéma databáze RO = {Ri(Ai,Fi)}, 1 <= i <= n
Struktury dat: VYSLED obsahuje po skončení algoritmu RO, POKR je typu Boolean
Algoritmus: begin
vysled:={R};
POKR:=false;
Vytvoř F+;
while (not POKR) do
if ve VYSLED existuje Ri, které není v BCNF
then begin
pro netriviální fční závislost $X \rightarrow Y \in Ri(Ai)$, že $X \rightarrow Ai \notin F+$ proveď
VYSLED:= (VYSLED - Ri(Ai)) $\cup$ Ri(Ai - Y) $\cup$ Rj(XY)
end
else POKR:=true
end

Příklad 4.100.

Mějme opět množinu funkčních závislostí

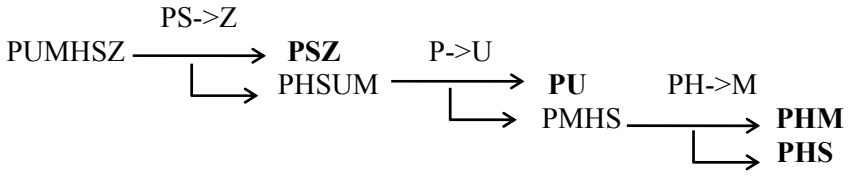
$$F= \{P \rightarrow U, HM \rightarrow P, HU \rightarrow M, PS \rightarrow Z, HS \rightarrow M, PH \rightarrow M\}.$$

Najděte dekompozici do BCNF.



Výsledkem rozkladu R tedy je RO1 = {PSZ, PU, PHM, SHM}

Jiným pořadím použití funkčních závislostí dostaneme jiný rozklad



Výsledkem je RO2 = {PSZ, PU, PHM, PHS}

Podobně můžeme dekompozicí dostat ještě další rozklady. Z hlediska algoritmu jsou výsledné rozklady rovnocenné, ovšem z hlediska uživatele dáme zřejmě přednost rozkladu RO1 *před* RO2, *neboť* HSM *sleduje, ve které místnosti a kdy jsou studenti, což nemusí uživatele zajímat.*

♦

Dekompozice je tedy proces návrhu schématu databáze, který by měl být řízen uživatelem.

Výsledky jsou příkladem, že dekompozice nemusí zachovávat pokrytí závislostí, neboť se mohou ztrácet některé závislosti.

❑ Bernsteinův algoritmus syntézy

Algoritmus syntézy vychází stejně jako dekompozice z dané množiny atributů a funkčních závislostí. Je dáno relační schéma **R** a množina funkčních závislostí F.

1. určí se klíč K univerzálního schématu a vytvoří se minimální pokrytí;
2. závislosti se roztřídí do skupin tak, aby v každé skupině byly závislosti se stejnou levou stranou; atributy závislostí každé skupiny vytvoří schéma jedné relace, vzniklého syntézou; atributy levé strany tvoří klíč vzniklého schématu;
3. jsou-li v takto vzniklých schématech schémata s ekvivalentními klíči, je vhodné je sloučit v jedno schéma, protože pravděpodobně popisují stejný objekt; sloučení ovšem může vést k tranzitivitám a tak i k vyšší složitosti algoritmu;
4. atributy, které se nevyskytují v žádné z funkčních závislostí F buď umístíme do samostatné relace, nebo je připojíme k některému ze schémat;
5. neobsahuje-li žádné schéma klíč univerzálního schématu, připojíme k zajištění bezeztrátovosti k výsledku další relační schéma s množinou atributů K tvořících klíč univerzálního schématu

Výsledkem je databázové schéma ve 3NF se zachováním všech závislostí a bodem 5. lze zajistit i bezeztrátovost. Je-li univerzální klíč K podmnožinou některého z dosavadních schémat, nové schéma se nevytváří.

Příklad 4.101.

Je opět dáno schéma **R(U, P, H, M, S, Z)**

a množina funkčních závislostí **F = {P → U, HM → P, HU → M, PS → Z, HS → M}**

Navrhněte strukturu databáze v BCNF.

Řešení algoritmem syntézy:

1. *klíčem schématu je* HS; *množina funkčních závislostí neobsahuje redundandní ani částečné závislosti a tvoří tedy minimální pokrytí*
2. *skupiny dle levých stran obsahují vždy jednu závislost, syntézou obdržíme schéma:*

RO4 = {PU, HMP, HUM, PSZ, HSM}

3. *protože* HM ↔ HU, *je možné sloučit schémata* HMP a HUM *a výsledkem je schéma*

RO5 = {PU, PUHM, PSZ, HSM}

4. *schéma* PU *je možno vyloučit, protože tvoří projekci* PUHM*; zde je jediným neklíčovým atributem* P *a ten není tranzitivně závislý na žádném ze dvou klíčů* HM, HU
5. *klíčem univerzálního schématu je* HS *a to je obsaženo v* HSM*, výsledkem je*

RO6 = {PUHM, PSZ, HSM}.

♦

Obecně syntézou bez bodu 5. může vzniknout schéma databáze, které nezachovává bezeztrátovost.

Příklad 4.102.

Mějme R(A,B,C) a F = {B→C, A→C}.

Řešení: Klíčem schématu je BA,

syntézou vzniknou schémata RO = {R1 (B,C), R2 (A,C)},

podle věty o ztrátě informace bezeztrátovost není zachována. Proto k rozkladu RO musíme připojit schéma R3(A,B).

Výsledek: RO = {R1 (B,C), R2 (A,C), R3(A,B)}

♦

Příklad 4.103.

Určete klíč a normální formu schématu R(A, B, C)
s funkčními závislostmi F={A→BC, C→A}.

Řešení: A+ = {A, B, C}
B+ = {B}
C+ = {C, A, B}

celkem 2 klíče A *a* C,

atributy A,C *jsou primární*, B *je sekundární;*

schéma je ve 3NF, není v BCNF, schéma nelze do BCNF převést (žádný rozklad by nepokryl závislost A→BC).

♦

Příklad 4.104.

Určete klíč a normální formu schématu R(A, B, C, D, E)
s funkčními závislostmi F={AB →C, C → A, ACD → B, BE → C, C → BD}

Řešení:

AB+ = {A,B,C,D}	A+ = {A}
C+ = {C,A,B,D}	B+ = {B}
ACD+ = {A,C,D,B}	D+ = {D}
BE+ = {B,E,C,A,D}	E+ = {E}

klíč: BE,

atributy B,E ... *primární*, A,C,D ... *sekundární*

minimální pokrytí: F={BE→C, C→A, C→B, C→D, AB→C}

skupiny a rozklad: RO1 = {BEC, CABD, ABC}

sloučení: RO2 = {R1(B, E, C), R2 (A, B, C, D)} *s klíči* BE *pro* R2, C nebo AB *pro* R2♦

♦

Příklad 4.105.

Navrhněte strukturu databáze pro IS ***Veřejné knihovny****. Knižní titul je zadaný ISBN, názvem, jedním autorem a vydavatelem; jednotlivé exempláře knih jsou evidovány pod přírůstkovým číslem. V knihovně může být více exemplářů téže knihy s různými přírůstkovými čísly. IS musí dávat přehled o stavu knižního fondu - o cenách knih, knihách vypůjčených, vybírat knihy pro upomínky po 1 měsíci od výpůjčky, tisknout upomínky, příp. umožnit prodloužení výpůjční doby knihy. Dále IS musí umožnit provádění rešerší - výběr knih podle klíč. slov, autorů a názvů.*

Řešení:

RU (isbn, nazev, autor, vydav, prir, cena, id-cten, jmeno, ulice, obec,psc,dat-od,dat-do,vrac, klic)

F = {isbn → nazev, autor, vydav, cena, klic;
prir → isbn;
id-cten → jmeno, ulice, obec, psc;
prir,id-cten, dat-od → dat-do, vrac}

F *už je ve tvaru min-nered pokrytí.*

Najděte dekompozici pomocí algoritmu dekompozice.

isbn→nazev,autor,vydav,cena,klic

vše ⟶ **isbn,nazev,autor,vydav,cena,klic** prir→isbn

isbn, prir, id-cten, …, dat-od, … ⟶ **prir, isbn**

prir,id-cten,jmeno,ulice,…,dat-od,…

id-cten→jmeno,ulice,obec,psc

prir, id-cten, jmeno, ulice, …, dat-od,… ⟶ **id-cten,jmeno,ulice,obec,psc**

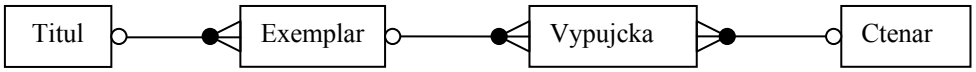
prir,id-cten,dat-od,dat-do,vrac

Výsledkem rozkladu je

R1 (isbn, nazev, autor, vydav, cena, klic)	... Titul
R2 (prir, isbn)	... Exemplar
R3 (id-cten, jmeno, ulice, obec, psc)	… Ctenar
R4 (prir, id-cten, dat-od, dat-do, vrac)	… Vypujcka

Tato malá databáze může být navržena zkušeným analytikem dobře i intuitivně.

Do odpovídajícího ERD je zakreslíme také povinnosti členství a kardinality.



♦

Příklad 4.106.

Navrhněte databázi pro evidenci projektů projekční firmy. Je potřeba evidovat jméno projektu, vedoucího projektu, zaměstnance pracující na projektu, počet plánovaných hodin na projekt pro každého zaměstnance, cenu projektu, datum zahájení, plat zaměstnance, oddělení, vedoucího oddělení a ohodnocení zaměstnance po ukončení projektu.

Platí: každý projekt má jednoznačný název,
každý projekt má jednoho vedoucího,
každé oddělení má jednoho vedoucího, oba vedoucí mohou být různí;
zaměstnanec může pracovat na více projektech, na projektu pracuje více zaměstnanců
jména oddělení jsou jednoznačná, zaměstnanec patri jednomu odděleni

Řešení: **RU** (nazev_pr, ved_pr, jmeno_zam, hod_zam, cena_pr, dat_zah, plat, oddel, ved_oddel, ohod_zam, id_zam)

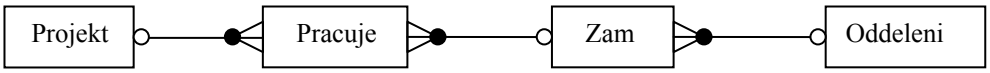
F = {nazev_pr → ved_pr, cena_pr, dat_zah;
oddel → ved_oddel;
id_zam → jmeno_zam, oddel;
id_zam, nazev_pr → hod_zam, ohod_zam }

Klic: id_zam, nazev_pr

F min,nered = F převedeno na elementární závislosti, bez tranzitivit

Alg. syntezy: R1 (nazev_pr, ved_pr, cena_pr, dat_zah)	… Projekt
R2 (oddel, ved_oddel)	… Oddeleni

R3 (id_zam, jmeno_zam, *oddel*) … Zam
R4 (*id_zam, nazev_pr*, hod_zam, ohod_zam) … Pracuje (vazebni)



♦

❑ Rozdíl mezi relační vazbou a funkční závislostí

Jak vazba relací, tak funkční závislosti mezi množinami atributů jsou integritní omezení, tedy plynou z reality a ze zadání. Podívejme se na oba pojmy znovu ve vzájemném kontextu.

Jde-li o vztah mezi atributy uvnitř jednoho schématu, jde o funkční závislosti. Někdy – dle modelované reality nebo po dekompozici – se však mohou tytéž atributy z funkční závislosti vyskytnout ve dvou schématech a pak již jde o relační vazbu.

Mezi oběma typy IO může tedy být jakási souvislost.

Příklad 4.106.

1. Člověk (jméno, ..., adresa, ...) adresa je atribut J → A fční závislost
2. Student (jméno, ..., adresa, kolej) J → KA fční závislost

   nebo

   Student (jméno, ..., adresa, číslo-koleje) J → AČ
   Kolej(číslo-k, k-adresa) Č → K
   } vazba typu 1 : M
3. Dům (adresa, počet bytů, počet pater, ...) A → vše
   Majitel (jméno, ...) J → vše
   Vlastní (jméno, adresa) JA → nic
   } vazba typu M: N, realizovaná vazební tabulkou

♦

CD-ROM

Na CD-ROMu s tímto výukovým textem jsou animované příklady na právě probraný algoritmus syntézy. Soubory s animacemi jsou pojmenovány

- Animace\relacni model\synteza_spoj2tab.exe
- Animace\relacni model\synteza_ztrataINF.exe

Shrnutí pojmů 4.3.4.

Předpoklad univerzální relace, předpoklad jednoznačnosti vztahů.
Algoritmus dekompozice.
Algoritmus syntézy.
Porovnání vztahů mezi atributy a vztahů mezi entitami.

Otázky 4.3.4.

1. Vyjmenujte předpoklady pro použití algoritmů návrhu struktury databáze.
2. Popište algoritmus dekompozice a charakterizujte jeho vlastnosti – jaký výsledek zaručuje.
3. Popište algoritmus syntézy a charakterizujte jeho vlastnosti – jaký výsledek zaručuje.
4. Charakterizujte rozdíl mezi funkční závislostí a relační vazbou.

Úlohy k řešení 4.3.4.

1. Je dáno relační schéma **R** (X, Y, Z, U, V) s funkčními závislostmi F = {XY→U, Y→V, U→Z}. Převeďte schéma do 3NF.

2. Je dána množina funkčních závislostí F = {AB → C, A → D, D → B}

   a) aplikujte Bernsteinův algoritmus syntézy a zkonstruujte odpovídající schéma relační databáze ve 3NF

   b) proveďte možné dekompozice

   c) je výsledné schéma z a) bezeztrátové ?

3. KNIHKUPEC

   Je dáno schéma relace Prodej(Knihkupec, ISBN, Adresa, Cena)

   a závislost F ={ISBN → Cena } (tj. cena knihy je stejná u všech knihkupců.)

   a) Ukažte, že Prodej není ve 3NF. Nahraďte toto schéma množinou schémat ve 3NF.

   b) Co se stane, změníme-li sémantiku tak, že zrušíme dané IO?

4. TŘÍDA

   Je dán seznam atributů pro databázi Třída, která má zaznamenat vztah mezi studenty a zapsaný předmět:

   Třída(čp, čst, jménost, čzam, rozvrh, místnost, známka)

   Přednášku má vždy jen jeden učitel; hodnoty atributu rozvrh mají zakódován den a hodinu v týdnu.

   Jsou dále dány tyto funkční závislosti :

   F = { čp → čzam, rozvrh, místnost
   čst → jménost,
   čp, čst → jménost, čzam, rozvrh, místnost, znamka.}

   a) určete klíč schématu Třída

   b) najděte částečné závislosti na klíči a sestrojte schéma relační databáze ve 2NF

   c) zjistěte, zda jsou výsledné relace ve 3NF

   d) zjistěte, zda byly použity všechny závislosti plynoucí z reality;

   e) je provedená dekompozice bezeztrátová?

   f) vytvořte k realitě Třída E-R diagram.

5. PROJEKTY

Databáze obsahuje atributy

JMPR = jméno projektu, JMVED = jméno vedoucího projektu, ČZAM = číslo zaměstnance, HOD = počet plánovaných hodin na projekt pro daného zaměstnance, JMÉNO = jméno zaměstnance, CENA = cena projektu, DATUM = datum zahájení, PLAT = plat zaměstnance, JMODD = jméno vedoucího oddělení, ODD = oddělení a OHOD = ohodnocení zaměstnance po ukončení projektu.

Platí :

- každý projekt má jednoznačné jméno, jména zaměstnanců a jména vedoucích nejsou jednoznačná;
- každý projekt má jednoho vedoucího, každé oddělení má jednoho vedoucího, oba vedoucí mohou být různí;
- mezi zaměstnanci a projekty je vztah typu M:N;
- jména oddělení jsou jednoznačná.

a) určete závislosti vyplývající za zadání

b) dekompozicí najděte schéma relační databáze ve 3NF

c) algoritmem syntézy vytvořte schéma relační databáze ve 3NF a porovnejte výsledek S b)

6. VELKOOBCHOD S KNIHAMI

Je dáno univerzální schéma relace obsahující informace o knižním velkoobchodě
RU(titul, autor, ISBN, jm_nakl, adr_nakl, objed, sklad, dat_vyd, kateg, cena_nákup, cena_prod)
Dále platí :

- ISBN jednoznačně identifikuje titul, sám titul není jednoznačný
- další vydání téhož titulu mají jiné ISBN
- eviduje se pouze 1.autor knihy
- jeden autor mohl napsat více titulů
- nakladatel má jednoznačné jméno a adresu – jm_nakl a adr_nakl
- objed udává počet objednaných kopií daného titulu
- sklad udává počet ještě neprodaných kopií titulu na skladě
- každá kniha má jedno datum vydání dat_vyd
- k určení kategorie (např. SCIFI, POEZIE,...) nestačí název knihy
- cena_prod je cena, kterou si velkoobchod účtuje za knihu, je o 20% vyšší než cena_nákup, kterou
- zaplatil knihu od nakladatele.

a) zformulujte netriviální funkční závislosti

b) najděte klíč schématu U

c) dekompozicí určete rozklad schématu do 3NF.

4.4. Pravidla dr. Codda pro RMD

Dr. Codd definoval hlavní pravidla relačního modelu:

- Databázi, popisující úsek reálného světa, tvoří konečná množina relací.
- SŘBD s relačním modelem dat obsahuje minimálně operace selekce, projekce a spojení bez jakýchkoliv předpokladů kladených na realizaci těchto operací. Při manipulaci s daty se nezajímáme o přístupové mechanizmy k datům obsaženým v relacích.

- Informační pravidlo: všechny informace v databázi jsou vyjádřeny explicitně na logické úrovni jediným způsobem - hodnotami v tabulkách.
- Pravidlo jistoty: Všechna data v databázi jsou přístupná pomocí kombinace jména tabulky, hodnotami primárního klíče a jména sloupce.
- Systematické zpracování nedefinovaných hodnot: SŘBD podporuje reprezentaci nedefinovaných hodnot nezávisle na datovém typu.
- Dynamický on-line katalog založený na relačním modelu: Popis databáze je vyjádřen na logické úrovni stejným způsobem, jako uživatelská databáze. Správce databáze může používat stejný relační jazyk pro dotazy na strukturu databáze, jako uživatel při práci s daty.
- Datový podjazyk: SŘBD může podporovat několik jazyků. Musí však být nejméně jeden jazyk s dobře definovanou syntaxí, který podporuje definici dat, definici pohledů, manipulaci s daty jak interaktivně tak dávkově, integritní omezení, autorizovaný přístup k databázi, zpracování transakcí.
- Pravidlo vytvoření pohledů: všechny teoreticky možné pohledy jsou definovatelné.
- Operace vkládání, rušení a vytváření dat zachovávají relační pravidla.
- Fyzická datová nezávislost: aplikační programy jsou nezávislé na fyzické datové struktuře.
- Logická datová nezávislost: aplikační programy jsou nezávislé na změnách v logické struktuře databáze.
- Integritní nezávislost: integritní omezení jsou definovatelná prostředky definice databáze nebo jejím jazykem pro manipulaci, musí být schopna uložení v katalogu, ne v aplikačním programu.
- Nezávislost distribuce: relační SŘBD musí být schopny implementace na jiných počítačových architekturách.
- Pravidlo přístupu do databáze: jestliže má SŘBD jazyk nízké úrovně, pak tato úroveň nemůže být použita k vytváření integritních omezení; pro práci s databází je nutno použít relačního jazyka vyšší úrovně. Pro to existují v RMD dva silné prostředky založené na relačním kalkul nebo relační algebře.
- Omezení redundance dat v relační databázi: jsou navrženy postupy umožňující normalizovat relace, tj. vhodně navrhovat potřebné databázové struktury.
- Realizace vazeb: vazby mezi entitami jsou reprezentovány opět relacemi; formálně se s nimi pracuje stejně jako s datovými relacemi. Předem v databázi nemusí být definovány žádné vazby, všechny mohou být v případě potřeby vytvořeny dynamicky prostřednictvím libovolných kritérií; po dokončení realizace požadavku vazba zaniká; pro opakované vytvoření téže vazby není nutné zachování stejné struktury datových objektů.

Zadání projektu

Podle odsouhlaseného zadání zpracujte **nový konceptuální datový model** tak, aby všechny relace byly alespoň ve 3NF. Použijte k tomu kompletní postup popsaný v kapitolkách 4.3. – 4.7. To znamená: definujte univerzální schéma relace, definujte existující funkční závislosti mezi atributy, určete k nim minimální neredundandní pokrytí a některým ze 2 algoritmů navrhněte schéma databáze ve 3NF nebo BCNF. Závěrem porovnejte oba výsledky (intuitivní a tento nový) a okomentujte případné rozdíly.

Pro navrženou strukturu databáze zformulujte nejprve slovně 5 různých dotazů na výběr informací a pak je napište pomocí jazyka SQL.

Výsledný projekt odevzdejte ve formátu DOC nebo PDF prostřednictvím podatelny na internetové adrese barborka.vsb.cz/podatelna/

5. METODY FYZICKÉ ORGANIZACE DAT

Čas ke studiu kapitoly: 3 hodiny

Cíl Po prostudování této kapitoly budete

- umět vyjmenovat základní databázové operace
- vědět, proč existují různé způsoby ukládání dat do databází
- vědět, jakými způsoby ukládají SŘBD data do databáze
- umět pro každý takový fyzický datový model popsat jeho podstatu a provádění základních databázových operací

Výklad

5.1. Vnější paměti

❑ Proč existují různé organizace ukládání dat v databázích

Připomeňme si, že hlavní důvodem existence databází je evidovat v nich rozsáhlá data. Ale samotná evidence by nebyla k velkému užitku, kdyby se uložené informace nedaly podle okamžitých potřeb vyhledávat a zpracovávat. U rozsahem malých databází (stovky až tisíce záznamů) v době rychlých počítačů příliš na způsobu uložení dat možná nezáleží. Ale z rozsáhlých databází ani rychlé počítače nevyhledají potřebná data dostatečně rychle, pokud jejich uložení nebude vhodně organizováno.

Příklad 5.1.

Představme si evidenci 1000 zaměstnanců velké firmy, jsou-li do tabulky zapisováni v pořadí, jak byli do firmy přijati. Potřebujeme-li vyhledat záznam o panu Kovářovi, musí se procházet postupně (sekvenčně) tabulkou shora dolů, až podle sloupce `Jmeno` *hledaný záznam najdeme.*

Ještě horší je představa sekvenčního vyhledání telefonního čísla v telefonním seznamu, pokud by ten nebyl seřazen podle abecedy. V setříděném seznamu hledáme informovaněji – kdo zná abecedu, začne hledat Kováře asi uprostřed, podle jmen na náhodně otevřené straně se rozhodne, zda bude pokračovat dopředu nebo dozadu v seznamu – to tak dlouho, až najde stranu se jmény Kovář (může jich být více) a pak sekvenčně dohledá toho svého (podle doplňkových údajů - křestního jména, adresy).

Zdálo by se tedy, že problém vyřeší abecední setřídění údajů o zaměstnancích. Ale jak budeme příště hledat zaměstnance s minimálním platem, zaměstnance se svářečskou zkouškou, zaměstnance dojíždějící z Opavy apod.? Opět sekvenčně.

♦

❑ Základní databázové operace, problém vyhledání informace

Výkon a tím i efektivita používání databáze je velmi závislá na implementaci operací na fyzické úrovni. Komunikaci s databází na logické úrovni zajišťují čtyři základní databázové operace:

- vyhledávání záznamu podle hodnoty jedné nebo více položek **SELECT** nebo FIND,
- vložení nového záznamu **INSERT**,
- modifikace záznamu **UPDATE** nebo MODIFY,
- rušení záznamu **DELETE** nebo ERASE.

Záznamy mohou být v databázi uloženy fyzicky různým způsobem. Jde nyní o to zvolit při implementaci SŘBD takovou organizaci uložení záznamů v databázi, aby výše uvedené čtyři databázové operace probíhaly co nejrychleji.

Ještě si rozeberme, jak která ze 4 operací bude na rychlost náročná. Již tušíme z příkladů, že vyhledávání informací bude klíčové. Při modifikaci nebo rušení záznamu ale také musíme nejprve příslušný záznam vyhledat a potom s ním provést příslušnou operaci. Uložení nových záznamů bude u různých organizací různě rychlé, ale na čas obvykle méně náročné. Tedy klíčovou operací z hlediska rychlosti je vyhledání záznamu.

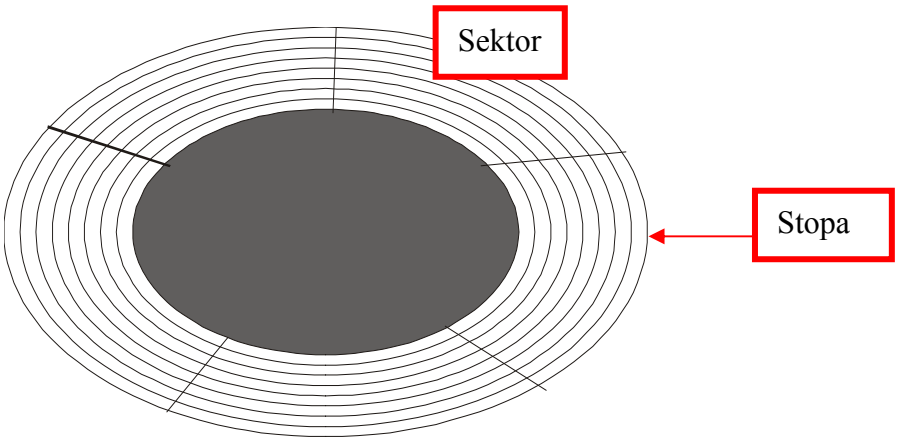
❑ Fyzické uložení dat na vnějších médiích

Diskem nazýváme jednu nebo celý svazek kruhových desek pokrytých magnetickou vrstvou. Každou desku pokrývají soustředné kružnice nazývané stopy, do těchto stop se zapisují data. Deska je dále rozdělena na kruhové výseče. Část stopy v jedné výseči se nazývá sektorem a je to nejmenší adresovatelná jednotka na disku. Pokud disk tvoří několik desek nad sebou, všechny stopy nad sebou nazýváme válec či cylindr.

Diskovou adresu na fyzické úrovni tedy tvoří označení diskové jednotky, číslo cylindru, stopy a sektoru.

Blok (stránka) je základní jednotku informace pro přenos dat mezi pamětí vnitřní a vnější. Rychlost takového přenosu i u rychlých systémů je řádově 1000x pomalejší, než rychlost operací ve vnitřní paměti. Tedy rychlost zpracování dat záleží na počtu přenosů dat mezi diskem a pamětí a prakticky nezáleží na počtu operací v paměti. Počet přenosů dále velmi závisí na organizaci uložení dat v diskových souborech.

Velikost bloku a délka záznamu jsou obecně různé. Blok může obsahovat několik záznamů nebo může záznam přesahovat délku několika bloků. Tedy počet přenosů mezi diskem a pamětí není totožný s počtem přenesených záznamů, jen je úměrný. Počet přenosů je roven číslu n/B, kde B je tzv. blokovací faktor - počet záznamů v bloku.



Obr. 5.1.: Struktura disku

❑ Softwarová podpora práce s databází

Vyhledat záznam na disku tedy znamená tři etapy zpracování:

- Uživatel zadá **logickou podmínku** na hledaný záznam v rámci aplikačního programu nebo pomocí dotazovacího jazyka *(hledá zaměstnance se jménem Novák, faktury ze dne 14.5. apod.)*.
- Na základě logické podmínky pro záznam určí SŘBD **logickou adresu záznamu**, tj. dle okolností pořadové číslo záznamu pro soubor s pevnou délkou nebo relativní adresu v rámci souboru. V obou případech SŘBD na základě znalosti své organizace dat spočítá číslo bloku, ve kterém je záznam uložen a určí začátek záznamu v bloku.
- Z čísla bloku se určí číslo válce, stopy a sektoru, tedy určit **fyzickou adresu záznamu**. Tuto etapu řeší obvykle součást **operačního systému** - subsystém ovládání souborů. Ten na základě zadaného čísla bloku v souboru spočítá absolutní číslo válce, stopy a sektoru. Pak na základě adresy záznamu v bloku a jeho délky najde hledaný záznam.

Proto v následujících odstavcích budeme popisovat jen vnitřní organizace na logické úrovni, určení fyzické adresy je již rychlá rutinní funkce.

S rychlostí přístupu k datovým souborům souvisí také využívání vyrovnávací paměti, její velikosti a ovládání. Ovládání se řeší na několika úrovních: v rámci OS, nastavením velikosti CACHE paměti, případně si vyrovnávací paměť řídí SŘBD sám.

Při popisu následujících technik budeme předpokládat soubory s pevnou délkou záznamu, soubory se záznamy s proměnnou délkou probereme až v závěru kapitoly.

5.2. Sekvenční soubory

❑ Podstata sekvenčních souborů

Nejjednodušší organizací pro implementaci, vycházející z přirozeného uspořádání záznamů podle pořadí jejich uložení, jsou sekvenční soubory.

Věty jsou uloženy v souboru v libovolném pořadí v blocích, ty mohou následovat fyzicky za sebou. Pokud bloky nenásledují za sebou, jsou propojeny ukazateli, nebo jsou adresy bloků souboru někde uloženy, to obvykle řeší OS.

❑ Základní databázové operace u sekvenčních souborů

Implementace sekvenčních souborů je jednoduchá. Základní databázové operace se provádějí následovně.

Nový záznam se uloží na konec souboru, k tomu stačí jeden přenos záznamu z paměti na disk.

Pro **vyhledání záznamu** se zadaným pořadím se určí z pořadí a délky záznamu adresa záznamu přímo. Ovšem v databázi se obvykle vyhledává pomocí hodnot jednotlivých položek, ne podle čísel záznamů. Pak je nutno prohledávat datový soubor sekvenčně: každý záznam postupně načíst a otestovat, zda vyhovuje podmínce. Pokud ano, je nalezen, pokud ne, načítá se další záznam v pořadí.

Vyhledávání sekvenční potřebuje průměrně n/2 porovnání nebo n/(2*B) přístupů na disk. Číslo n znamená počet záznamů a číslo B je blokovací faktor, znamená počet záznamů v bloku.

Modifikace záznamu znamená provést tyto operace: nalézt záznam, načíst, opravit a na stejnou adresu znovu zapsat.

Zrušení záznamu u sekvenčních souborů se obvykle provádí ne vymazáním záznamu, ale pouze označením jeho neplatnosti. K označení neplatnosti se obvykle vyhradí místo v záznamu (bit, byte) a vlastní položky záznamu zůstanou zachovány. Při zpracování se záznamy označené jako neplatné nezpracovávají. "Díry" po zrušených záznamech tak postupně zabírají v souboru místo. Je vhodné občas soubor překopírovat do nového souboru jenom s platnými záznamy. Jiná možnost je využít prázdných míst při vkládání nové věty - záznam se uloží do prvé díry po vypuštěné větě, nebo se uloží na konec souboru, pokud díra neexistuje. Ovšem pak se i operace vložení věty provádí průměrně pomocí n/2 přístupů na disk.

Mají-li věty primární klíče, musí se při vkládání nové věty a při modifikaci klíče prohledat celý soubor a zkontrolovat jednoznačnost klíče.

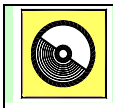
Příklad 5.2.

Evidence dat o zaměstnancích v tabulce. Zaměstnanci jsou zapisováni v pořadí, jak byli do firmy přijati.

adresa	delete	**osob**	**jmeno**	**…**
1		3456	Dudek Jindřich	
2		1243	Kovář František	
3	*	3333	Novák Karel, ing.	
4		5734	Horák Ivo	
5		2578	Sedlák Jiří	
6		9999	Forman Zdeněk	
7		3579	Anděl Gabriel	
…				
n		7766	Nováčková Iva	
n+1		8765	Gavendová Miriam	

Při hledání Dudka jde o jeden přenos záznamu, při hledání Nováčkové o n přenosů, průměrně při hledání obecného záznamu o n/2 přenosů z disku do paměti.

♦



CD-ROM

Na CD-ROMu s tímto výukovým textem jsou animované příklady na provádění čtyř základních databázových operací u souborů s použitím sekvenčních souborů. Jsou to soubory:

- Animace\01-Sekvencni\01-sekv_insert.exe
- Animace\01-Sekvencni\02-sekv_select.exe
- Animace\01-Sekvencni\03-sekv_select_vice_zaznamu.exe
- Animace\01-Sekvencni\04-sekv_update.exe
- Animace\01-Sekvencni\05-sekv_delete.exe

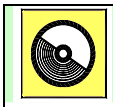
5.3. Setříděné sekvenční soubory

Popis setříděné organizace

Pokud se v souboru často vyhledává podle některého klíče (zde myslíme vyhledávací klíč, což nemusí být vždy primární klíč) a provádí se relativně málo změn těchto klíčů, je vhodné uchovávat soubor v setříděném tvaru podle vyhledávacího klíče. Znamená to po každé změně (vložení nové věty nebo modifikaci klíče) znovu soubor setřídit, což výrazně zpomaluje tyto operace.

Pak se ale dá vyhledávat podle klíče mnohem rychleji (např. metodou půlení intervalu nebo některou její modifikací). Počet přenosů pro binární hledání je průměrně $\log_2 n$. S výhodou se tedy tento způsob uložení používá u souborů s velmi řídkými změnami (např. u tzv. číselníků ap.).

Někdy se metoda vyhledání záznamu vylepšuje tak, že provádíme interpolaci na základě znalosti statistického rozložení hodnot klíče v tabulce.



CD-ROM

Na CD-ROMu s tímto výukovým textem jsou animované příklady na provádění čtyř základních databázových operací u souborů s použitím setříděných sekvenčních souborů. Jsou to soubory:

- Animace\02-Sekvencni_Setridene\06-sekv_setr_insert.exe
- Animace\02-Sekvencni_Setridene\07-sekv_setr_select.exe
- Animace\02-Sekvencni_Setridene\08-sekv_setr_select_vice_zaznamu.exe
- Animace\02-Sekvencni_Setridene\09-sekv_setr_update.exe
- Animace\02-Sekvencni_Setridene\10-sekv_setr_delete.exe

5.4. Zřetězené organizace záznamů

Popis zřetězené organizace

Jestliže vyžadujeme, aby záznamy byly nějakým způsobem setříděny, je možno použít také techniku zřetězení záznamů. Proti sekvenčnímu souboru obsahuje každý záznam navíc jednu položku. V ní je ukazatel na následující záznam v souboru podle daného uspořádání. V souboru tak je vytvořen seznam či řetěz vzájemně propojených záznamů, řetězy mohou realizovat uspořádání dle libovolného kritéria.

Základní databázové operace u zřetězených souborů

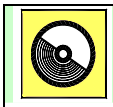
Operace **SELECT** se provádí tak, že se seznam prohledává postupně pomocí ukazatelů a testuje, zda záznam vyhovuje vyhledávací podmínce. Z povahy zřetězené organizace plyne, že vyhledávání v seznamu bude vyžadovat častější přechody mezi bloky souboru a proto více přenosů mezi diskem a pamětí. Proto zřetězené organizace je vhodné používat jen u "krátkých" seznamů, kdy je možno případně celý seznam načíst do paměti.

Operace **INSERT** se provede tak, že se záznam fyzicky zapíše kamkoliv, pak se v seznamu vyhledají sousední záznamy dle udržovaného pořadí a přesměrují se ukazatele.

Operace **DELETE** se provede tak, že se vyhledá umístění záznamu v seznamu a přesměrují se ukazatele.

Operace **UPDATE** znamená jen vyhledání záznamu a po modifikaci jeho zápis zpět. Jen při modifikaci položek, které mají vliv na uspořádání seznamu, se provede modifikace jako DELETE a INSERT.

Obvykle se zřetězených organizací užívá v kombinaci s některou jinou metodou. Seznamy pak jsou dostatečně krátké, aby byly zachovány výhody zřetězení. Některé metody popíšeme níže.



CD-ROM

Na CD-ROMu s tímto výukovým textem jsou animované příklady na provádění čtyř základních databázových operací u souborů s použitím zřetězených souborů. Jsou to soubory:

- Animace\03-Zretezeni\11-zret_insert.exe
- Animace\03-Zretezeni\12-zret_select.exe
- Animace\03-Zretezeni\13-zret_select_vice_zaznamu.exe
- Animace\03-Zretezeni\14-zret_update.exe
- Animace\03-Zretezeni\15-zret_delete.exe

5.5. Soubory s přímým adresováním

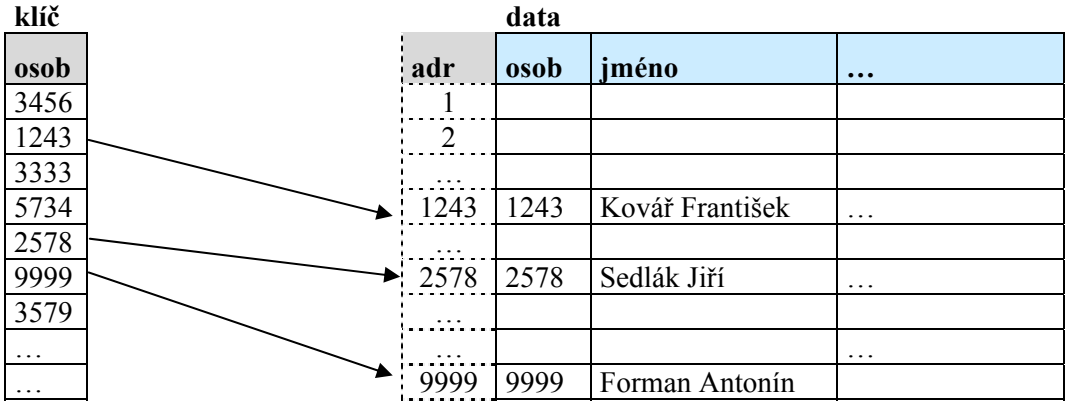
Popis organizace souborů s přímým přístupem

Velmi rychlý přístup k záznamům prostřednictvím klíčů zajišťuje metoda přímého adresování. Teoretický princip metody je tento: jednoznačný klíč záznamu se zakóduje do jednoznačného čísla, toto číslo pak přímo určuje adresu záznamu v souboru. Tak je možné jediným přístupem na disk záznam načíst, příp. druhým přístupem záznam po modifikaci zapsat.

Interval získaných adres však může být ohromný a velmi řídce obsazený. Tak by docházelo k velkému plýtvání kapacitou paměti.

Příklad 5.3.

Čtyřciferné osobní číslo je klíčem záznamu zaměstnance, navíc není nutné ho kódovat číselně. Znamená tedy přímo adresu záznamu. Při hledání podle osobního čísla stačí jediný přenos z diskové adresy totožné s osobním číslem. Potřebný prostor pro asi 100 zaměstnanců zde je 9999 adres.



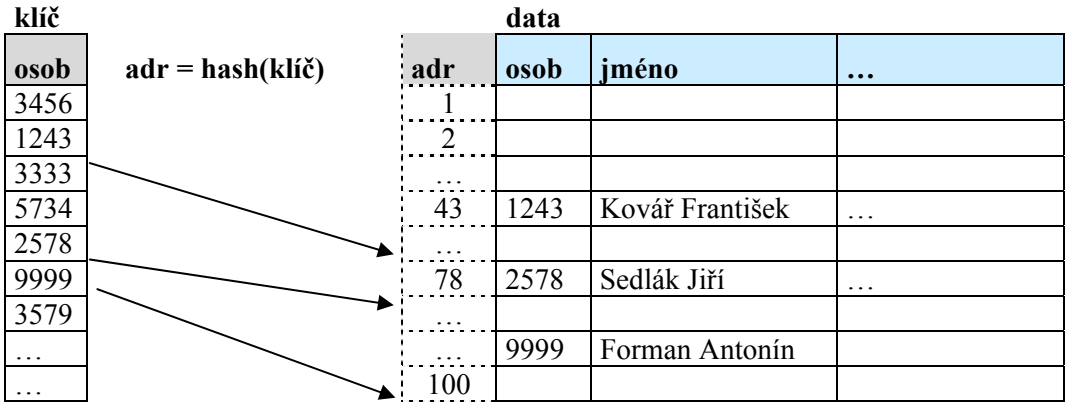
Proto se užívá speciální funkce zvané hašovací (hash function), která transformuje původní interval adres do číselného intervalu požadované velikosti. Velikost výsledného intervalu je zvolena tak, aby zhruba odpovídala skutečnému počtu záznamů.

Příklad 5.4.

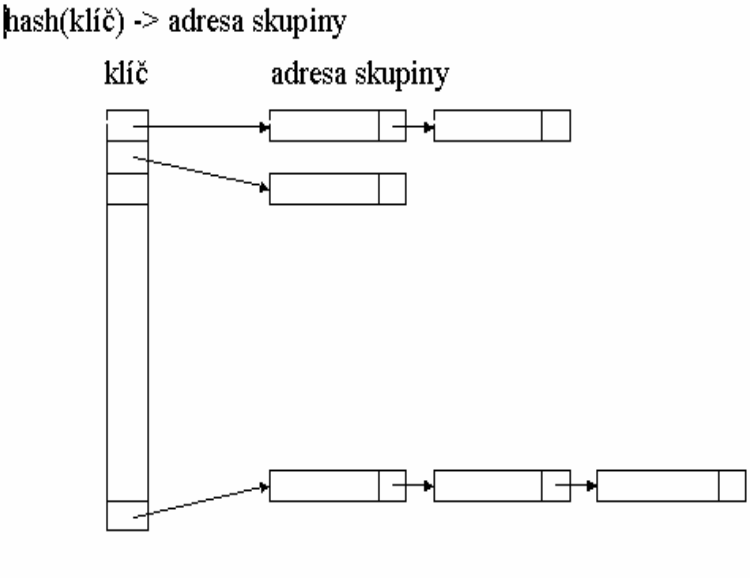
Tabulka zaměstnanců s klíčem `osob` *(osobní číslo) používá jednoduchou hašovací funkci*

adr = (osob MOD 100) + 1

tedy transformuje čtyřciferné osobní číslo na adresu z intervalu <1,100> funkcí Modulo 100, výsledkem je tedy poslední dvojčíslí osobního čísla. Získáme tak prostor jen pro 100 zaměstnanců.



♦



Použitím hašovací funkce však může dojít k nejednoznačnosti výsledné adresy. Tak několika původním klíčům odpovídá stejná adresa. Situace se pak řeší takto: nejednoznačná adresa znamená adresu skupiny všech záznamů se stejnou hodnotou hašovací funkce, záznamy se v rámci skupiny ukládají zřetězeně. Protože již jde o krátké seznamy, je přístup k nim rychlý.

❑ Základní databázové operace u souborů s přímým přístupem

Vyhledání záznamu podle klíče je tedy velmi rychlé a odtud jsou rychlé i ostatní databázové operace. Čas potřebný k výpočtu adresy pomocí hašovací funkce je zanedbatelný: z klíče se vypočte adresa skupiny záznamů, na ní se začne prohledávat zřetězený seznam až po hledaný záznam.

Vyhledávání podle neklíčové hodnoty se však naopak prodlužuje, protože je potřeba sekvenčně procházet prázdná místa a seznamy. Také požadavek na setřídění záznamů znamená komplikaci.

Pro **nový záznam** se spočítá adresa skupiny záznamů, v ní se prohledají záznamy (pro kontrolu jednoznačnosti klíče), nový záznam se pak uloží na první volné místo ve skupině.

Rušení se provede vyhledáním, označením neplatnosti a přesměrováním ukazatelů z.předchůdce na následníka.

Modifikace znamená vyhledání, modifikaci a uložení zpět. Jen při modifikaci klíče se provede nejprve zrušení a pak nový záznam.

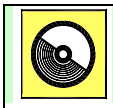
❑ Hašování nejen podle primárního klíče

Jestliže je potřeba vyhledávat záznamy nejen podle klíče, ale podle více položek, je možno použít např. následující postup:

1. položky pro vyhledávání považujeme za řetězec bytů, rozdělíme je do skupin po 2 bytech = 16 bitech
2. každou skupinu považujeme za celé číslo, všechna čísla sečteme a dostaneme klíč pro ukládání záznamů,
3. na součet použijeme hašovací funkci a dostaneme adresu skupiny záznamů.

Pak je ovšem pro hledání nutno zadat všechny hodnoty těchto položek.

Jestliže počet záznamů souboru trvale roste, je potřeba občas upravit hašovací funkci a provést reorganizaci souboru.



CD-ROM

Na CD-ROMu s tímto výukovým textem jsou animované příklady na provádění čtyř základních databázových operací u souborů s použitím hašování. Jsou to soubory:

- Animace\04-Hash\16-hash_insert.exe
- Animace\04-Hash\17a-hash_select.exe
- Animace\04-Hash\17b-hash_select_sekvencni.exe
- Animace\04-Hash\18-hash_select_sekvencni_dohledani.exe
- Animace\04-Hash\19-hash_update.exe
- Animace\04-Hash\20-hash_delete.exe

5.6. Indexované a indexové soubory

❑ Popis indexování

Nejčastější organizací dat v relačních SŘBD je použití některého způsobu indexování.

Základem indexované organizace je sekvenční datový soubor, ke kterému existuje jedna nebo několik dalších pomocných tabulek, pomocí nichž můžeme v datovém souboru rychleji hledat.

Základní datový soubor, který je doplněn takovou pomocnou tabulkou, nazýváme **indexovaným souborem**, pomocnou tabulku nazýváme **indexovým souborem**.

Do pomocné tabulky se umístí hodnota vyhledávacího klíče (**indexu**) a adresa (např. pořadové číslo) záznamu. Tabulka je organizována tak, aby **vyhledání klíčové hodnoty** proběhlo rychle pro libovolnou část datového souboru. Například při setřídění pomocné tabulky dle indexu se hledá binárně hodnota klíče, na vyhledaném řádku v indexovém souboru se zjistí hodnota adresy v datovém souboru a jediným přístupem do dat se načte hledaný datový záznam. Často je indexová tabulka dost malá na to, aby mohla být při prohledávání celá umístěna v operační paměti. Tím se hledání opět výrazně zrychlí.

Hlavní myšlenkou indexování tedy je, že ukazatele nemusí být vždy součástí záznamů (jako u zřetězených organizací), ale mohou být uloženy ve zvláštním (indexovém) souboru.

Indexem nemusí být jen primární klíč, ale kterákoliv položka souboru nebo seznam několika položek, pokud se podle nich často vyhledává. Pokud je indexem klíč, mluvíme o **primárním indexování**, v ostatních případech o **sekundárním indexování**. Při sekundárním indexování musíme počítat s tím, že stejné hodnoty indexu v datech nabývá více záznamů a tak jedné hodnotě položky odpovídá více adres.

V kapitole o relačním datovém modelu si uvedeme další využití indexových souborů pro spojování tabulek při realizaci vazby.

Příklad 5.5.

Mějme jednoduchý soubor zaměstnanců s osobním číslem, platem a procentem daně.

indexovaný datový soubor

adr	OSOB	PLAT	PROC
1	106	850	20
2	121	870	10
3	124	2400	30
4	100	1230	10
5	133	1340	30
6	110	1900	10
7	120	740	20
8	117	1400	30
9	130	1600	20
10	111	1600	30

primární index

OSOB	Adresa
100	4
106	1
110	6
111	10
117	8
120	7
121	2
124	3
130	9
133	5

sekundární indexy

PROC	Adresy
10	2,4,6
20	1,7,9
30	3,5,8,10

nebo

PROC	Adresa
10	2
10	4
10	6
20	
…	

♦

Pro sekundární indexování se také používá název **invertovaný soubor**. Jestliže jsou tabulky indexů vytvořeny pro všechny atributy říkáme, že je soubor **úplně invertovaný**. V tomto případě je zapotřebí více než dvakrát tolik paměti, než potřebuje sám datový soubor.

❑ Základní databázové operace u indexových souborů

Základní databázové operace se tedy provádějí takto:

při **vkládání** nového záznamu se záznam uloží na konec datového souboru, přidá se do všech indexových souborů a ty se setřídí.

Vyhledávání podle některého existujícího indexu se provádí výše popsaným způsobem - binárním vyhledáním hodnoty indexu v příslušném indexovém souboru, odtud adresy záznamu v datovém souboru. Pak jediným přenosem z datového souboru.

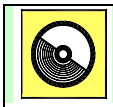
Odtud i **modifikace** položek a **rušení** záznamu, vyhledávaného pomocí indexu je rychlá. Vyhledají se a po modifikaci zapíší zpět, případně označí za neplatné. Při modifikaci některé indexované položky je nutné přetřídit příslušný indexový soubor.

Vyhledávání, modifikace a rušení záznamu podle neindexovaných položek se provádí jako u sekvenčního souboru. Ale při změně indexovaných položek je opět nutné upravit indexové soubory.

V některých aplikacích mohou být klíče velmi dlouhé. Pak se také místo v paměti zvětšuje. Se zvětšováním délky klíče se prodlužují seznamy záznamů, odkazující na shodné hodnoty podklíče. Pak se prodlužuje i hledání v takových indexech. Proto se někdy ukládají klíče v indexech zkráceným způsobem tak, že je v tabulce uložena předpona klíče a je připojen seznam přípon s adresami, potom další předpona atd. Stejná metoda je používána v jazykových slovnících.

Celkově můžeme konstatovat, že technika indexování umožňuje rychlejší přístup k tabulkám pomocí indexovaných položek, ale na druhé straně spotřebuje více místa na disku pro indexové soubory a také je zapotřebí vyšší režie na údržbu indexových tabulek při změnách datového souboru. Proto je indexování výhodné především pro soubory s malým objemem změn a pro malé a střední rozsahy dat (aby nebyly indexy příliš velké).

Analýza toho, které indexové soubory se budou používat, patří k důležité součásti vývoje informačního systému. Tento problém budeme podrobněji diskutovat později.



CD-ROM

Na CD-ROMu jsou animované příklady na provádění čtyř základních databázových operací u souborů s použitím jednoduchého indexování. Jsou to soubory:

- Animace\05-Indexovani\21-index_index.exe
- Animace\05-Indexovani\22-index_insert.exe
- Animace\05-Indexovani\23-index_select.exe
- Animace\05-Indexovani\24-index_update.exe
- Animace\05-Indexovani\25-index_delete.exe

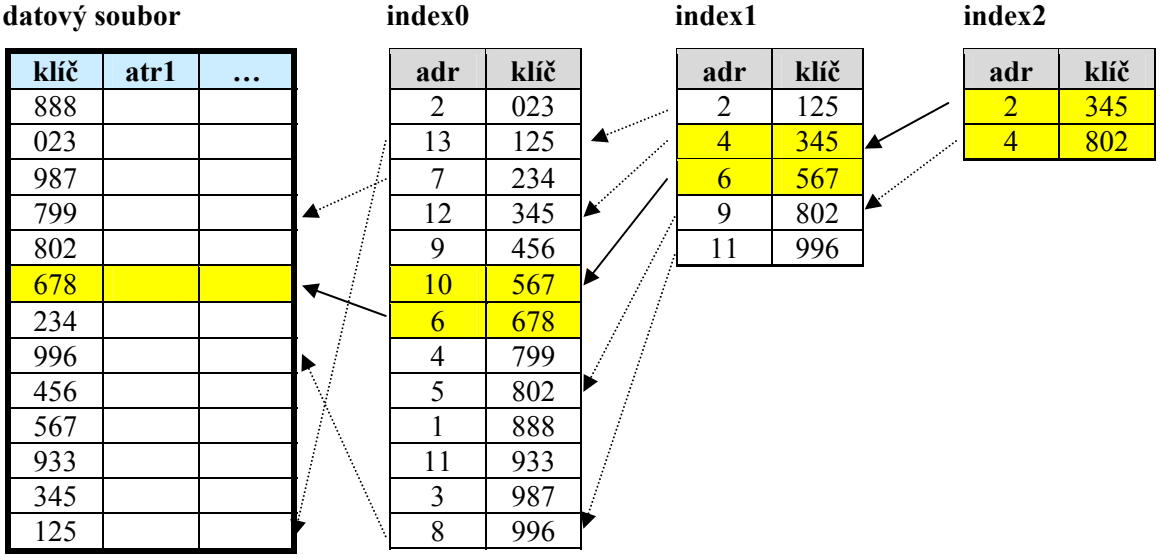
5.7. Hierarchické indexování, B-stromy

❑ Hierarchické indexování

Pro zvláště velké soubory je možno pro hledání v indexovém souboru použít opět indexový soubor a sestrojit tak celou **hierarchii indexových souborů**. Pak se na vyšších úrovních indexace objevují indexy skupin indexů nižší úrovně. V rámci nalezené skupiny se dohledávají záznamy sekvenčně nebo opět binárně. Na indexy 1. úrovně mohou odkazovat indexy 2. úrovně atd.

Hledá se od nejvyšší úrovně. V indexech vyšších úrovní se nevyhledává přesná hodnota indexovaného atributu, ale interval, do kterého padne.

Příklad 5.6. *V hierarchii indexů zaznamenává vyšší index dolní mez intervalu, v němž je na nižší úrovni odkaz na interval s hledaným záznamem. Nejnižší úroveň indexu ukazuje již do datového souboru. Vyhledání záznamu s klíčem 678: v indexu2 se najde nejbližší menší hodnota klíče (=345), její adresa ukazuje do indexu1 na interval mezi 2.a 4. záznamem. Tam se opět najde nejblíže nižší klíč (=567), ten ukazuje do indexu0 na interval mezi 6. a 9. záznamem. V něm se dohledá přesná hodnota klíče 678 a adresa vedle něj ukazuje na 6. záznam v datovém souboru.*

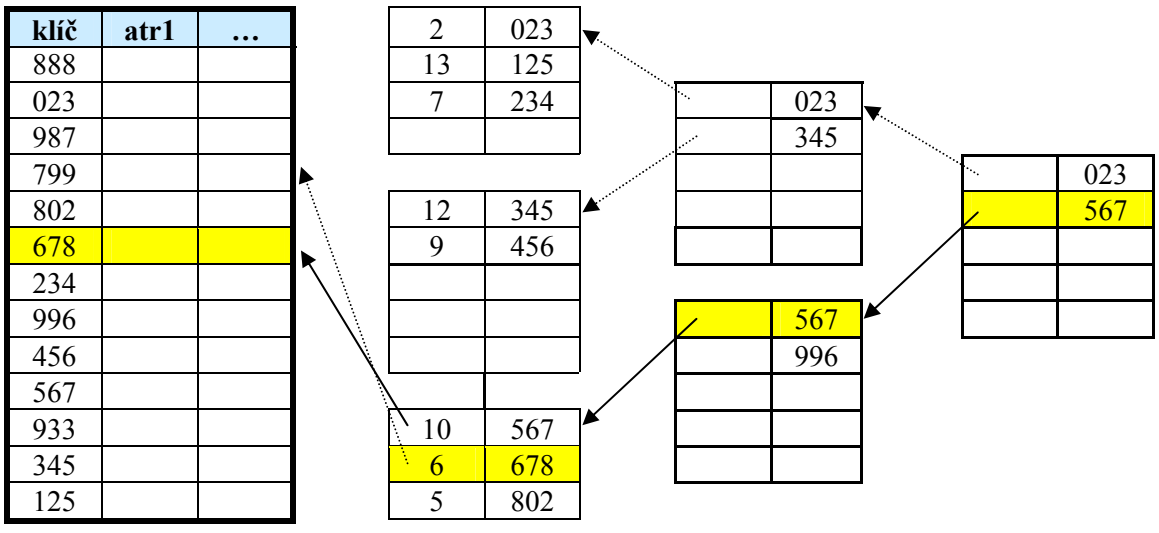


♦

❑ Popis indexování pomocí B-stromů

Pro uspořádání úrovní indexů se velmi často používá varianta hierarchického indexování pomocí tzv. B-stromů (**B**alanced-tree). Listy stromu vzniknou tak, že se indexové soubory všech úrovní (mimo kořen) rozsekají na části. Listy odpovídají velikosti bloku a jsou tak načteny jediným přenosem z disku. V rámci malého a setříděného uzlu se najde nejbližší hodnota (nejblíže nižší nebo nejblíže vyšší podle konstrukce stromu) hledaného vyhledávacího klíče velmi rychle. Platí, že všechny cesty od kořene stromu do libovolného listu jsou stejně dlouhé – stejným počtem přenosů se dojde k datům.

Příklad 5.7. *Tentýž příklad pomocí B-stromu, index každé úrovně je rozdělen na stejně velké části – uzly stromu.*



♦

❑ Základní databázové operace při užití B-stromů

Při **hledání** najdeme cestu ve stromě od kořene až k listu, v němž by měl být hledaný záznam (pokud v souboru existuje). V každém uzlu najdeme správnou větev porovnáním hledaného klíče s klíči v uzlu. Klíče v uzlu mohou udávat minimální (příp. maximální) hodnotu klíče, která je příslušnou větví dosažitelná.

Modifikace záznamu je z hlediska vyhledávání jednoduchá. Záznam se vyhledá, změní a zapíše zpět. Při modifikaci klíče se provede zrušení původního záznamu a zavedení nového v indexu0 (případně vyšších, viz vkládání).

Při **vkládání** nového záznamu se najde příslušný blok a mohou nastat dvě možnosti: bud v nalezeném bloku je místo, takže se může přidat vkládaný záznam, nebo nalezený blok je plný, takže se musí vytvořit nový blok. Z původního plného bloku se vytvoří dva bloky, každý obsahuje polovinu záznamů a zbylá místa prázdná.

Do vyšší úrovně musíme nový blok nižší úrovně zaznamenat a opět mohou nastat dva případy. Proces se opakuje až do kořene stromu a případně se musí kořen rozdvojit. Pak se přidá nový kořen a vznikne o úroveň víc v indexové struktuře.

Rušení záznamů se provádí opačně, než vkládání. Při zrušení posledního záznamu listu stromu se zruší i odkaz na něj, totéž se promítne do vyšších úrovní, případně se v krajním případě může hierarchie indexů o jednu úroveň snížit.

CD-ROM

Na CD-ROMu s tímto výukovým textem jsou animované příklady na provádění čtyř základních databázových operací u souborů s použitím indexování pomocí B-stromů. Jsou to soubory:

- Animace\06-B-stromy\26-B_stromy_index.exe
- Animace\06-B-stromy\27-B_stromy_insert.exe
- Animace\06-B-stromy\28-B_stromy_select.exe
- Animace\06-B-stromy\29-B_stromy_update.exe
- Animace\06-B-stromy\30-B_stromy_delete.exe

5.8. Indexování pomocí binární matice

❑ Popis indexování pomocí binární matice

Pro sekundární indexování se někdy z důvodů ušetření kapacity používá jiného typu indexů - binárních matic. Poloha záznamu se bude zaznamenávat polohou jedničkového bitu v posloupnosti, která má tolik bitů, kolik má soubor záznamů. První bit odpovídá prvnímu záznamu, druhý druhému atd. Pro každou hodnotu sekundárního atributu je zaznamenána nová posloupnost. Zřejmě je tato metoda vhodná pro takové atributy, které nabývají jen několika málo různých hodnot.

Příklad 5.8.

Mějme soubor zaměstnanců s osobním číslem, platem a procentem daně z příkladu 5.5.

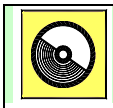
atribut	hodnota	pořadí záznamů 1	2	3	4	5	6	7	8	9	10	11	12	…
proc	10	0	1	0	1	0	1	0	0	0	0	0	1	
	20	1	0	0	0	0	0	1	1	0	0	0	0	
	30	0	0	1	0	1	0	0	0	1	1	1	0	
plat	2000	0	0	0	0	1	0	0	0	0	0	0	0	
	3000	1	0	0	0	0	1	0	0	1	1	0	0	
	4000	0	1	1	1	0	0	0	0	0	0	1	1	
	5000	0	0	0	0	1	0	1	1	0	0	0	0	

proc = 30 ∧ plat = 4000	1		1

♦

Binární matice jsou zvlášť výhodné v případech, že se hodnoty sekundárních atributů nemění, když se nemění záznamy, případně se jen přidávají sériově na konec souboru. Další výhodou binárních matic je snadná realizace kombinovaných dotazů pomocí logických operátorů negace, konjunkce a disjunkce.

V případech, které nevyužijí jmenovaných výhod, se používají normální indexy.



CD-ROM

Na CD-ROMu s tímto výukovým textem jsou animované příklady na provádění čtyř základních databázových operací u souborů s použitím indexování pomocí binární matice Jsou to soubory:

- Animace\07-Binarni_Matice\31-binarni_matice_index.exe
- Animace\07-Binarni_Matice\32-binarni_matice_insert.exe
- Animace\07-Binarni_Matice\33-binarni_matice_select.exe
- Animace\07-Binarni_Matice\34-binarni_matice_update.exe
- Animace\07-Binarni_Matice\35-binarni_matice_delete.exe

5.9. Datové soubory s proměnnou délkou záznamu

Dosud jsme předpokládali pevnou délku datových typů všech atributů a tedy i pevnou délku záznamů. Jejich implementace je výrazně jednodušší a mnohé SŘBD jinou možnost nepřipouští. Ovšem z reality plyne často požadavek na složitější strukturu. Jde např. o již zmiňované opakující se položky (pole) předem známý počet-krát i předem neznámý počet krát, o skupinové položky (recordy), dále o dlouhé texty různé délky, o záznamy obrázků, zvuků a mnohé jiné datové typy.

Používání souborů s proměnnou délkou záznamu vede k řadě nových problémů. Často se úvahy o datových typech vedoucích k proměnné délce záznamu vyskytují jen v logickém modelu a implementace se provádí pomocí záznamů pevné délky. Novější SŘBD však stále častěji připouštějí různé datové typy proměnné délky a také je tak implementují.

Hlavní metody těchto implementací jsou následující:

1. Pseudoproměnná délka záznamu

Takto nazveme případ, kdy se logicky proměnná délka záznamu implementuje jako záznam s pevnou délkou. Používají se k tomu následující způsoby:

- pole se známým počtem opakování: pole atomických položek se rozloží na jednotlivé položky, každá dostane vlastní jméno a pracuje se s ní samostatně; pak při zpracování to ale komplikuje přístup k opakovaným položkám jako celku;

  Příklad 5.9.

  Konceptuální schéma: Typ_entity (atr1, atr2, atr3, atr4:multi)

  Atibut atr4 má vždy známý počet složek (zde 4). Transformace do databázového schématu je možná rozepsáním pomocí samostatně pojmenovaných atributů a navrhne se struktura tabulky s pevnou délkou:

atr1	atr2	atr3	atr4_1	atr4_2	atr4_3	atr4_4
...	...	...				

  ♦

- pole s neznámým počtem opakování: odhadne se shora počet výskytů prvků pole a tak se převede na předcházející případ; může se stát, že značná část prvků pole bude nevyužitá; dalším problémem je pak rozpoznání prázdného prvku – může se řešit například další položkou „počet“.

  Příklad 5.10.

  Konceptuální schéma: Typ_entity (atr1, atr2, atr3, atr4:multi)

  Transformace do databázového schématu pomocí pseudoproměnné délky: odhadne se nejvyšší počet opakování (zde 7) a navrhne struktura tabulky s pevnou délkou:

atr1	atr2	atr3	počet	atr4_1	atr4_2	atr4_3	atr4_4	atr4_5	atr4_6	atr4_7
...	...	...	1							
			5							
			2							
			4							
			...							

  ♦

- místo opakující se položky uvedeme odkaz na seznam jejích prvků, ty mohou tvořit záznamy jiného souboru; soubor proměnné délky se převede na dva soubory pevné délky:

  Příklad 5.11. *tentýž příklad*

atr1	atr2	atr3	počet	ukaz
...	...	...	1	
			5	
			2	
			4	
			...	

atr4	ukaz
hod1	nil
hod2	ukaz
hod3	
hod4	
...	

  ♦

- pro záznamy s alternativními skupinami položek: buď se proměnná část "překrývá" a záznam zabírá velikost nejdelší z proměnných částí, avšak v záznamu se musí rozlišovat typ proměnné části a implementace je složitější; nebo se všechny rozdílné atributy zaznamenají "za sebou" a pro každý typ se vyplňují jen odpovídající atributy; implementace je jednodušší, záznam však obsahuje vždy řadu prázdných položek.

2. Proměnná délka záznamu v sekvenčním souboru

Při sekvenční organizaci souborů s proměnnou délkou záznamu je nutno od sebe jednotlivé záznamy rozlišit. Používají se např. tyto metody:

- systém oddělovačů: záznamy jsou odděleny oddělovačem (viz např. textové soubory), uvnitř záznamu se atributy oddělují jiným typem oddělovače, opakující se položky dalším typem ap.
- zaznamenání délky aktuálního záznamu na začátku záznamu (pro jednosměrný průchod souborem), či na začátku i konci záznamu (pro obousměrný průchod souborem)

3. Proměnná délka záznamu s jinou organizací

Zřetězené organizace, přímé adresování, indexování, příp.další organizace je možno implementovat podobně, jako při pevné délce záznamu. Rozdíl je pouze v tom, že místo pořadového čísla záznamu je nutno zaznamenávat skutečnou adresu záznamu v souboru, což obecně zabírá více místa.

Shrnutí pojmů 5.

Organizace ukládání dat v databázi.
4 základní databázové operace. INSERT, SELECT, UPDATE, DELETE.
Realizace databázových operací pomocí pomocných souborů.
Indexové soubory, B-stromy.
Realizace databázových operací pomocí pomocných výpočtů.
Adresy záznamu vypočtené z klíče, hašování.
Průběžné udržování dat, setříděné nebo zřetězené soubory.

Otázky 5.

1. Proč jsou v různých SŘBD používány různé organizace ukládání dat do databáze?
2. Které operace se provádějí s daty v databázi?
3. Jaké techniky pro urychlení práce s daty se používají?
4. Co znamená sekvenční organizace dat?
5. Jak se provádějí základní databázové operace při sekvenční organizaci dat?
6. Co znamená setříděná organizace dat?
7. Jak se provádějí základní databázové operace setříděných souborů?
8. Popište princip jednoduchého indexování a provádění základních databázových operací při této organizaci dat.
9. Popište princip indexování pomocí B-stromů a provádění základních databázových operací při této organizaci dat.

10. Popište princip indexování pomocí binární matice a provádění základních databázových operací při této organizaci dat.
11. Popište princip přímého adresování a provádění základních databázových operací při této organizaci dat.
12. Popište princip zřetězené organizace a provádění základních databázových operací při této organizaci dat.
13. Kdy se používá zřetězených organizací?
14. Které z probraných organizací dat používají pomocné datové soubory?
15. Které z probraných organizací dat nepoužívají pomocné datové soubory a co používají k urychlení vyhledání dat?

Domácí úkol

Indexování je velmi často používaná organizace a je vhodné ji pochopit detailně. Proto váš hlavní úkol bude naprogramovat (v libovolném programovacím jazyce, ne v SŘBD) tuto úlohu:

Zvolte si jednoduchý datový soubor s několika atributy (např. adresář). Napište program, který bude provádět 4 základní operace nad daty, a to ve 2 režimech: sekvenčně a s použitím indexového souboru vytvořeného k jednomu atributu. Režimy bude možno přepínat, výpisy dat se podle toho zobrazí buď sekvenčně nebo setříděně.

Program bude uživatelem řízen pomocí jednoduchého menu s položkami

1. nový záznam
2. modifikace záznamu
3. zrušení záznamu
4. vyhledání záznamu
5. výpis souboru
6. přepnutí režimu sekvenční / indexovaný přístup
7. konec

6. SÍŤOVÝ DATOVÝ MODEL

Čas ke studiu: 3 hodiny

Cíl Po prostudování celé kapitoly budete umět

- rozumět pojmům síťového datového modelu
- vysvětlit základní rozdíl mezi relačním a datovým modelem
- definovat data v tomto modelu
- vysvětlit princip práce s daty v síťovém modelu
- vyhledávat informace v síťové databázi pomocí procedurálního jazyka

Výklad

6.1. Základní pojmy síťového datového modelu

❑ Historie síťového datového modelu

Již víme, že historicky prvním datovým modelem v databázové technologii byl model hierarchický a pak jeho zobecnění, model síťový. Nejprve vznikaly samostatné implementace síťových SŘBD. V roce 1971 pak byl definován pracovní skupinou pro databáze DBTG (Data Base Task Group) při sdružení CODASYL (Conference on Data System Languages) návrh architektury SŘBD síťového modelu. Od té doby se stal standardem pro prakticky všechny implementace.

U nás bylo v té době velmi málo počítačů vůbec, většinou jen ve velkých výrobních podnicích a pro zpracování dat se převážně používal jazyk Cobol. Síťový model se příliš nerozšířil. Mohli jsme se setkat například se síťovými databázovými systémy

IDMS na počítačích řady IBM 360, IBM 370, EC 1045, EC 1055,
DBMS na počítačích řady SMEP, DEC, VAX,
IMAGE na počítačích Hewlet-Packard, ADT.

Standard CODASYL se týkal především implementace síťového SŘBD: deklarací - příkazů pro definice dat a JMD - způsobu práce a příkazů pro manipulace s daty.

❑ Význam síťového modelu v současnosti

Proč vlastně ještě probíráme síťový model? I když se v současnosti příliš nepoužívá, snad jen ve starších dosud užívaných IS, nejsou principy tohoto modelu zastaralé. Některé metody zabudované do standardu je velmi užitečné znát i dnes. Například použití jednoznačného databázového klíče, v SŘBD s datovými typy ukazatel realizace vazeb pomocí setů atd. jsou metody aktuální i u současných objektově-relačních databází. Mohou tak využít výhod relačního modelu a současně některých metod modelu síťového.

Základní pojmy síťového datového modelu

Slovník síťového modelu vychází z programovacího jazyka Cobol. Standard zahrnuje tyto hlavní pojmy a zásady:

Logickému modelu databáze se říká **schema**, externím schématům **subschemata**. Na úrovni definice schématu se typ entity nazývá **typ záznamu** (RECORD), jeho atributy se nazývají **komponenty**. V databázi samé se jednotlivé entity nazývají **výskyty záznamu** příslušného typu.

Databáze se skládá z řady výskytů záznamů každého z deklarovaných typů.

Výskyty záznamů dokonce nemusí být různé, tj. jedna entita může být v databázi reprezentována více výskyty záznamů. Tyto výskyty jsou rozlišeny pouze hodnotou **databázového klíče**, který je systémem automaticky přidělován každému výskytu záznamu v okamžiku jeho uložení do databáze. Jeho hodnota jednoznačně identifikuje každý výskyt záznamu vůči jiným výskytům záznamů i jiného typu, tedy v rámci celé databáze.

Vyskytují se i deklarace typů záznamu bez komponent. Zdánlivě prázdné výskyty těchto záznamů pak ve skutečnosti obsahují ukazatele na jiné výskyty záznamů.

Realizace vazeb v síťovém datovém modelu

Datový model síťový se liší od relačního hlavně způsobem realizace vztahů mezi entitami.

Síťový model definuje pouze binární vztahy typů 1:1 a 1:N mezi dvěma typy záznamů R a S. Ostatní typy vztahů se předem. na úrovni konceptuální rozloží (v tom je shoda s RDM). Tento vztah se nazývá množina neboli **set**. Set je definován pomocí svého vlastníka a členů. Typ záznamu R se nazývá vlastníkem (OWNER) setu S, S je členem (MEMBER) setu S. Ve schématu je pro každý vztah definován **typ setu**. Je mu přiděleno jméno, definován typ záznamu, který je vlastníkem a typ záznamu, který je členem setu.

Síťový datový model realizuje vztah pomocí ukazatelů na vazební entity. Ke každé tabulce je připojena systémová část s tolika odkazy, ke kolika jiným typům záznamů je záznam vázán (= kolika setů je účastníkem, vlastníkem nebo členem).

V databázi je vztah reprezentován řadou **výskytů setu**. Výskyt setu obsahuje právě jeden výskyt záznamu vlastníka a právě ty výskyty záznamů člena setu, které jsou s vlastníkem výskytu setu v příslušném vztahu. Výskyt setu může obsahovat pouze výskyt záznamu vlastníka (prázdný výskyt setu) a množiny členů dvou výskytů téhož typu setu jsou disjunktní.

Výskyty setů se realizují zřetězeným seznamem: z vlastníka na prvního člena, členové setu mezi sebou, z posledního člena zpět na vlastníka. Pak procházení celé množiny je velmi rychlé, bez zbytečných přenosů záznamů mezi diskem a pamětí. Pokud jsou členy setu navíc umisťovány „blízko sebe“, může být počet přenosů ještě menší, pokud je více členů množiny umístěno v jednom fyzickém bloku.

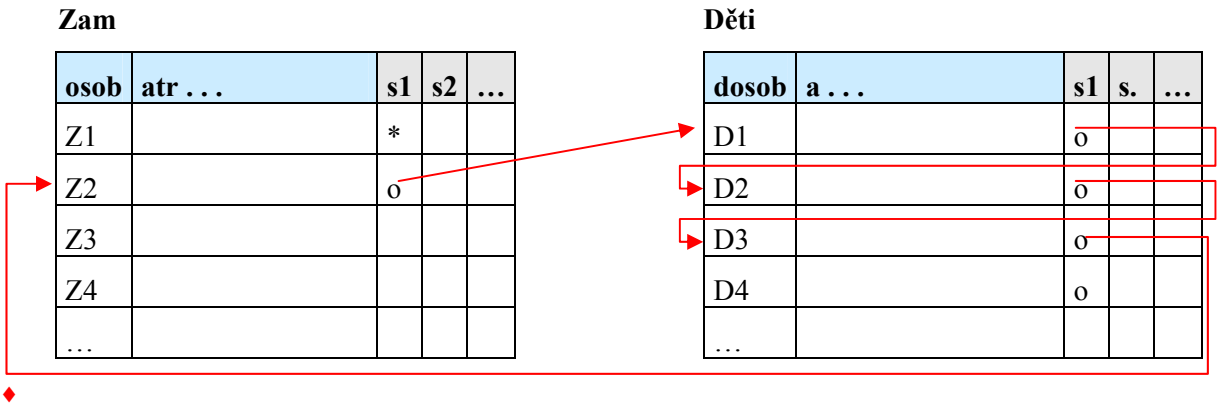
Pro zařazení záznamů do setů platí:

- výskyt setu obsahuje právě jeden výskyt záznamu vlastníka a právě ty výskyty záznamů členů setu, které jsou s vlastníkem výskytu setu v příslušném vztahu,
- výskyt setu může obsahovat pouze výskyt záznamu vlastníka (prázdný výskyt setu),
- množiny členů dvou výskytů téhož typu setu jsou disjunktní

Příklad 6.1.

Na následujícím obrázku jsou dvě tabulky **Zam** a **Děti** *ve vztahu 1:M, tedy* Zam *je vlastníkem,* Děti *členem setu zde nazvaného* **rod**. *První systémový sloupec v tabulkách* Zam i Děti *je deklarován pro set* rod. *První zaměstnanec nemá žádné děti, druhý má 3 děti , a to D1, D2, D3.*

Další systémové sloupce jsou určeny pro další typy setů, kterých se oba typy záznamů mohou účastnit.



❑ Zobrazení záznamů, vazeb a jejich typů – konceptuální schéma

Graficky se znázorňují jednotlivé prvky modelu takto (jde o jiný způsob zakreslení konceptuálního schématu, než pomocí ERD):

typ záznamu: Katedra

typ setu (hrana od vlastníka ke členu)

jméno typu vlastníka — Katedra

jméno typu setu — ČLEN-KATEDRY

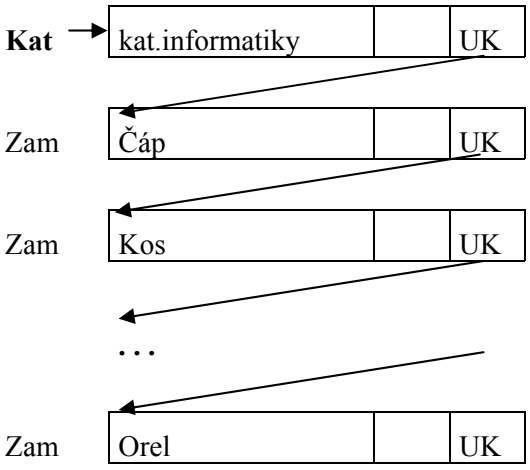
jméno typu člena — Zaměstnanec

výskyt záznamu

Katedra

kat. informatiky ...	UK

výskyt setu

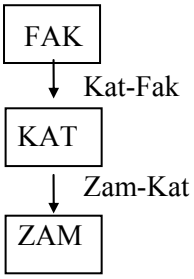


❑ Pravidla pro definování setů

Pro vlastníka a členy setu platí:

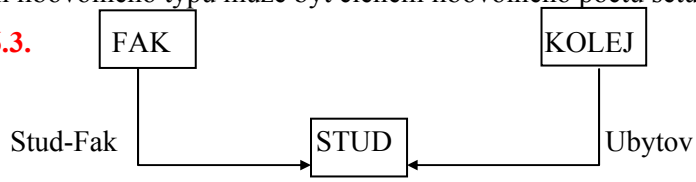
1. Tentýž typ záznamu může být současně vlastníkem jednoho setu a členem v jiném setu.

Příklad 6.2.



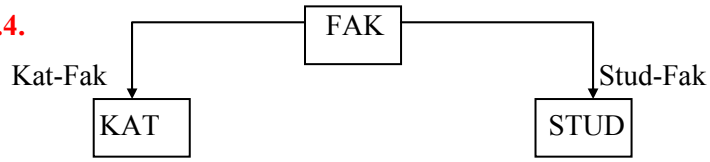
2. Záznam libovolného typu může být členem libovolného počtu setů.

Příklad 6.3.



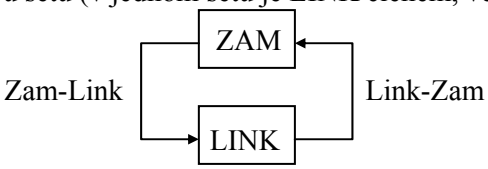
3. Záznam libovolného typu může být vlastníkem libovolného počtu setů.

Příklad 6.4.



4. Vlastník a člen setu nemohou být záznamy téhož typu. Unární vztah 1:N se realizuje prostřednictvím dalšího pomocného typu záznamu (který má pouze spojovací roli a nazveme jej LINK) a pomocí dvou typů setů (v jednom setu je LINK členem, ve druhém vlastníkem).

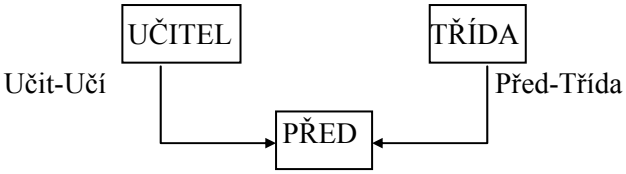
Příklad 6.5.



Pro vztah „přímý nadřízený“ mezi záznamy typu Zaměstnanec se zavede typ záznamu LINK bez uživatelských komponent. Každý výskyt záznamu příslušející vedoucímu pracovníkovi je vlastníkem výskytu setu Zam-Link s právě jedním členem typu LINK a ten je pak vlastníkem výskytu setu Link-Zam, jehož členy jsou podřízení dotyčného vedoucího pracovníka.

5. Vztah typu M:N není možno realizovat přímo a realizuje se (již na úrovni konceptuálního schématu) pomocí dvou vazeb typu 1:M prostřednictvím spojovacího typu záznamu. Ten je členem v obou typech setů.

Příklad 6.6.



Mezi Učitelem a Třídou má vztah „Učí“ kardinalitu M:N. Zavedeme pomocný spojovací typ záznamu Předmět, který na rozdíl od LINKu z minulého případu má vlastní komponenty Název a Hodin. Vazbu

pak realizují dva typy setů Učit-Učí a Před-Třída. Vlastníky setů jsou Učitel a Třída, v obou je Předmět členem. Existence vztahu mezi Učitelem a Třídou je vyjádřena přítomností výskytu záznamu Předmět (ten předmět, který příslušný učitel učí v příslušné třídě).

Takto je možno realizovat i vztah typu M:N mezi záznamy téhož typu.

6.2. Jazyk pro definici dat

❑ Deklarace schématu a subschémat

Je-li provedena datová analýza, tj. navrženy typy entit a vazeb, převádí se konceptuální datový model celé databáze na databázový model pomocí deklarací (odpovídají JDD).

Deklarace schématu (= databázového schématu celé databáze) má strukturu:

```
SCHEMA NAME IS jméno_schématu
   RECORD SECTION
      deklarace_záznamů
   SET SECTION
      deklarace_setů
```

Z daného schématu je možno deklarovat libovolný počet subschémat (= dílčích pohledů na databázi, odpovídajících aplikačním programům). Subschéma může být sdíleno více aplikacemi, různá subschémata se mohou v databázi překrývat.

V deklaraci se uvede jméno subschématu a schématu, z něhož je odvozeno. Dále platí

- v deklaraci záznamů můžeme vynechat některé komponenty původních záznamů, některé záznamy nebo některé sety.
- komponentám záznamů je možno přiřadit jiný datový typ, než měly ve schématu, můžeme změnit pořadí komponent uvnitř záznamu, případně můžeme objektům ze schématu přiřadit jiná jména.
- při deklaraci záznamů a setů již neuvádíme jejich vlastnosti stanovené ve schématu.

Deklarace subschématu má strukturu:

```
SUBSCHEMA jméno_subschématu WITHIN jméno_schématu
   RECORD SECTION
      deklarace_záznamů
   SET SECTION
      deklarace_setů
```

❑ Deklarace typů záznamů

Při definování struktury databáze se používá syntaxe z jazyka Cobol. Datové typy jsou definovány pomocí typu a délky (například PIC 9(n) nebo PIC X(n)), jsou povoleny úrovně definice (01, 02, … před komponentou znamenají hloubku úrovně atributu), tedy neatomické položky.

Součástí definice mohou být vlastnosti, které upřesňují způsob umístění nových záznamů do databáze. Mimo automatického umisťování samotným SŘBD je možné volit umístění záznamů některého typu pomocí hašovací funkce nebo volit umisťování vlastníka a členů setů blízko sebe, aby jejich vyhledávání a přenos byly minimalizovány.

Deklarace typu záznamu má tvar:

```
RECORD NAME IS jméno_typu_záznamu
  LOCATION MODE IS SYSTEM                    ... umístí sám systém
                   DIRECT                    ... přímo dle DB-klíče
                   CALC proc USING sez-pol   ... přímo dle vybrané položky
                   VIA                       ... členy setu "blízko"
vlastníka
        01 <komponenta 1> PIC  9(n)
        01 <komponenta 2> PIC X(n)
        02 <komponenta 3> PIC X(n)
        ...
        01 <komponenta 5> PIC X(n)
        ...
```

Příklad 6.7.

Deklarace typu záznamu s neatomickým atributem adresa

Zaměstnanec(osob, jmeno, adresa(ulice, obec, PSČ), plat)

a s hašováním podle osobního čísla osob. *Hašovací procedura dává adresu umístění záznamu:*

adr = proc_zam(osob)

```
RECORD NAME IS Zaměstnanec
   LOCATION MODE IS CALC proc_zam USING osob
        01 osob          PIC  9(6)
        01 jmeno         PIC  X(20)
        01 adresa        PIC X(n)
           02 ulice      PIC  X(20)
           02 obec       PIC  X(20)
           02 PSC        PIC  9(5)
        01 plat          PIC  9(8)
```

♦

❑ Deklarace typů setů

Také set musíme předem deklarovat. Přitom je opět možno definovat (mimo vlastníka a člena setu) řadu vlastností setu, jako způsob zařazování nového člena do setu (= uspořádání setu), povinnost členství ve vztahu, zda může člen měnit vlastníka apod.

```
SET NAME IS jméno_typu_setu
   OWNER IS jméno_typu_vlastníka
      ORDER IS FIRST                              ... uložení v setu jako první člen
               LAST                               ... poslední člen
               NEXT                               ... před aktuální člen
               PRIOR                              ... za aktuální člen
        SORTED BY DEFINES KEYS                    ... uložení zatříděné podle klíče
        PERMANENT                                 ... není možno měnit pořadí člena
   MEMBER IS jméno_typu_člena MANDATORY           ... povinný člen, může měnit vlastníka
                              FIXED               ... nemůže měnit vlastníka
                              OPTIONAL            ... nepovinný člen
                              AUTOMATIC           ... začlenění provede systém automat.
                              MANUAL              ... začlenění výskytu záznamu provede
                                                      ručně uživatel v aplikaci
                    DUPLICATE IS NOT ALLOWED      ... jednoznačnost třídicího klíče
```

Příklad 6.8.

Deklarace databáze FAKULTA *s následujícími entitami a vazbami.*

Typy záznamů: Učitel (čU, jméno, funkce, plat)
Předmět (čp, název, ročník)
Uvazek (čU, čp, hodin)
Student (jméno, město, ulice)
Kolej (název)

Typy setů: UČITEL_UČÍ (Učitel, Úvazek)
PŘEDMĚT_UČÍ (Předmět, Úvazek)
BYDLÍ (Kolej, Student)

Hašovací procedury pro přímé adresování tabulek Učitel *a* Předmět *se jmenují* hash_čU (čU) *a* hash_čp (čp).

```
SCHEMA NAME IS FAKULTA
  RECORD SECTION
      RECORD NAME IS Učitel
         LOCATION MODE IS CALC hash_čU USING čU IN Učitel
            02 čU       PIC XXX
            02 jméno    PIC X(15)
            02 funkce   PIC X(10)
            02 plat     PIC 99
      RECORD NAME IS Předmět
         LOCATION MODE IS CALC hash_čp USING čP IN Předmět
            02 čp       PIC XXX
            02 název    PIC X(15)
            02 ročník   PIC 9
      RECORD NAME IS Uvazek
         LOCATION MODE IS SYSTEM-DEFAULT
            02 čU       PIC XXX
            02 čp       PIC XXX
            02 hodin    PIC 99
            . . .
  SET SECTION
      SET NAME IS Učitel_Učí
         OWNER IS Učitel
         ORDER IS PERMANENT SORTED BY DEFINED KEYS
         MEMBER IS Uvazek MANDATORY AUTOMATIC
         KEY IS ASCENDING čp IN Uvazek
         DUPLICATE ARE NOT ALLOWED
         SET SELECTION IS THRU Učitel_Učí
          IDENTIFIED BY CALC-KEY
      SET NAME IS Učí_Předmět
         OWNER IS Předmět
         ORDER IS PERMANENT SORTED BY DEFINED KEYS
         MEMBER IS Uvazek MANDATORY AUTOMATIC
         KEY IS ASCENDING čU IN Uvazek
         DUPLICATE ARE NOT ALLOWED
         SET SELECTION IS THRU Učí_Předmět
         IDENTIFIED BY CALC-KEY
            . . .
```

Příklad 6.9.

Část deklarace subschématu pro mzdovou agendu.

```
SUBSCHEMA PENIZE WITHIN FAKULTA
    RECORD SECTION
      01 učitel
         02 ču              PIC X(3)
         02 honorář PIC 99
      01 úvazek
         02 čp              PIC X(3)
         02 ču              PIC X(3)
         02 hodin    PIC 99
   SET SECTION
       SET Učitel_Učí
            . . .
```

♦

6.3. Jazyk pro manipulaci s daty

❑ Pracovní oblast uživatele

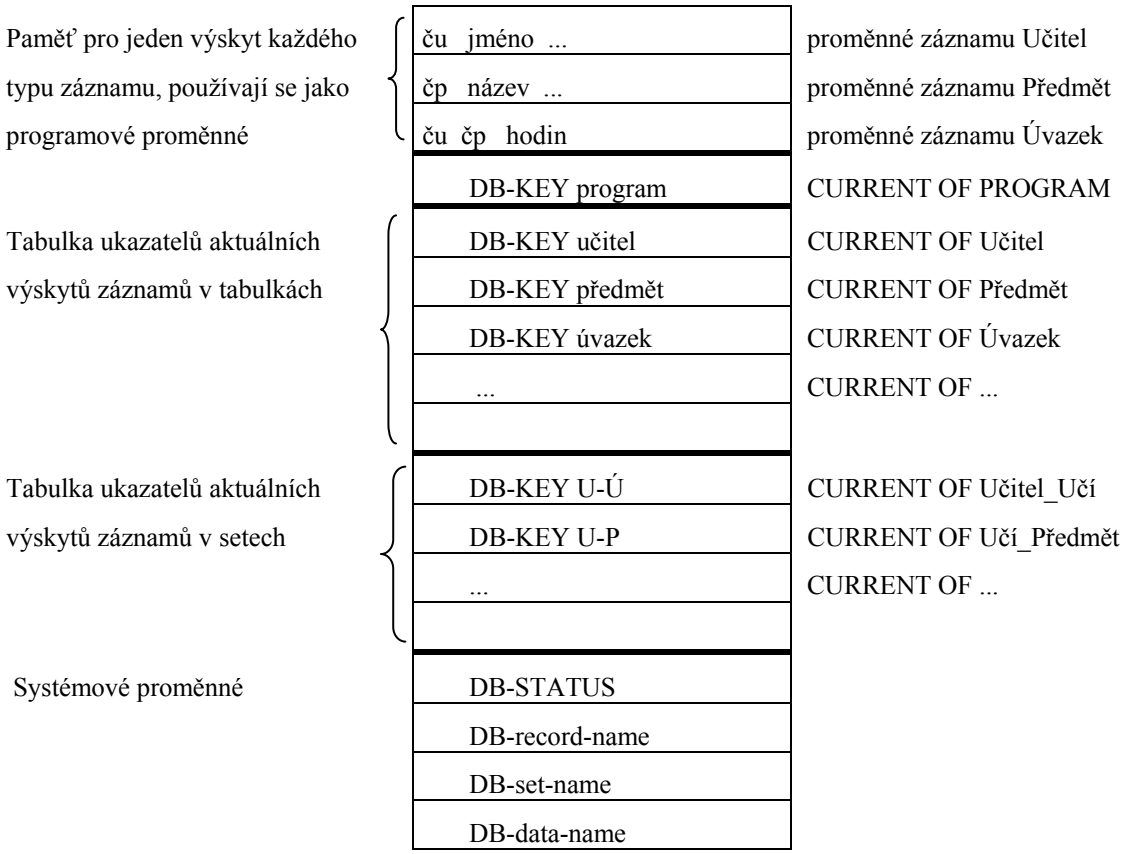
Jazyk pro manipulaci s daty je koncipován jako množina příkazů vnořených do hostitelského jazyka. Často to býval COBOL nebo PL/1. V další části textu budeme pro lepší srozumitelnost předpokládat, že hostitelským jazykem je Pascal, i když to zpravidla tak není.

Pro každý aplikační program (v paměti jich může současně být více) je v paměti počítače zvláštní oblast, která se nazývá **pracovní oblast uživatele** (user work area). V každé uživatelské oblasti jsou umístěny:

- prostor pro uchování jednoho výskytu každého typu záznamu, který uživatelský program používá; v programu se na tento prostor odkazuje jmény záznamů, případně jejich komponent podle deklarací v subschématu;
- ukazatele na aktuální objekty (currency pointer) ve formě databázového klíče, ke kterým patří: CURRENT OF typ_záznamu
- ukazatele na naposledy zpracovávané výskyty záznamů pro každý typ záznamu: CURRENT OF SET typ_setu
- odkazy na naposledy zpracovávané výskyty záznamů v každém setu: CURRENT OF PROGRAM
- odkaz na naposledy zpracovávaný výskyt záznamu celkově bez ohledu na jeho typ nebo příslušnost k setu;
- stavové proměnné, obsahující popis stavu,v němž se systém právě nachází; jsou to např.
  db-status obsahuje 0 po bezchybně provedené operaci nebo číslo chyby,
  db-set-name \ **db-record-name** lokalizují místo, kde nastala chyba,
  db-data-name obsahuje název chyby

Příklad 6.10.

Pracovní oblast pro databázi FAKULTA:



♦

❑ Seznam příkazů JMD

❑ Základní 4 databázové operace

FIND vyhledá výskyt záznamu v DB a definuje jej za aktuální záznam programu, aktuální záznam příslušného typu záznamu a případně aktuálním záznamem setů, v nichž je záznam zařazen; jinými slovy databázový klíč vyhledaného výskytu záznamu je uložen do položek CURRENT OF PROGRAM, příslušného CURRENT OF typ a příslušných CURRENT OF SET

GET přesune do uživatelské oblasti (na místo rezervované pro tento typ záznamu) nalezený aktuální záznam programu

STORE uloží nový výskyt záznamu do DB, definuje jej za aktuální záznam programu, aktuální záznam příslušného typu záznamu a za aktuální záznamy setů, do nichž je případně automaticky zařazen

MODIFY modifikuje aktuální záznam programu

ERASE vymaže aktuální záznam programu z DB

- **Základní operace se sety**

CONNECT	vloží aktuální záznam programu do výskytu setu
DISCONNECT	vyjme aktuální záznam programu z výskytu setu
RECONNECT	přesune aktuální záznam do jiného setu
ORDER	setřídí set

- **Další databázové příkazy**

READY	otevře pracovní uživatelskou oblast pro práci s DB
FINISH	ukončí práci s pracovní uživatelskou oblastí
IF	možnost testu v posledním vyhledaném záznamu
USE	nastavení reakce při výjimečném stavu DB
ACCEPT	přesun hodnot systémových DB proměnných do položek v programu
KEEP	uzamkne výskyt záznamu pro ostatní uživatele
FREE	odemkne výskyt záznamu

Příkazy FIND a STORE tedy mění obsah tabulky běžných ukazatelů, ostatní příkazy pracují s aktuálním záznamem jako s operandem.

Provádění základních databázových operací

Dále probereme nejdůležitější příkazy JMD poněkud podrobněji. Pro lepší srozumitelnost použijeme v těchto příkazech poněkud zjednodušené syntaxe JMD i upravených tvarů hostitelského jazyka.

Označíme: T ... typ záznamu
S ... typ setu
DBK ... db-klíč
POL ... položky záznamu

Prakticky na příkladech uvidíme, že často nebude nutné přenášet datové záznamy do paměti, protože pomocí ukazatelů je možno přecházet mezi záznamy jen prostřednictvím DBK. Proto jsou příkazy pro vyhledání záznamu (zatím jsme tuto operaci chápali jako vyhledání a přenos do paměti současně) rozděleny na vlastní vyhledání FIND a přenos GET. Příkazem GET se tak přenáší jen skutečně potřebná data., která se předcházejícím použitím příkazu FIND nalezla.

- **Načtení záznamu z databáze do paměti**

Příkaz GET má formát

GET T

Pokud aktuálním záznamem programu je výskyt záznamu typu T (předtím nalezený některou variantou příkazu FIND), přenese příkaz GET tento výskyt záznamu do uživatelské pracovní oblasti a tím jej zpřístupní programu ke zpracování.

- **Vyhledání záznamu**

Příkaz FIND je základním příkazem JMD, umožňuje různými způsoby vyhledávat výskyt záznamu v DB a definovat jej aktuálním záznamem programu.

Příkaz FIND má několik variant:

- **Vyhledání záznamu podle hodnoty databázového klíče:**

  FIND T RECORD BY DB-KEY DBK

Příklad 6.11.

Do proměnné typu databázový klíč si uschováme hodnotu klíče z tabulky ukazatelů:

```
klíč:=CURRENT OF učitel;
...
FIND učitel RECORD BY DB-KEY klíč;
GET učitel;
```

♦

- **Vyhledání záznamu podle CALC-klíče:**

  FIND T RECORD BY CALC_KEY

Před provedením této varianty příkazu FIND je nutné dosadit hodnotu položkám definovaným ve schématu jako CALC-klíče pro záznam typu T.

Příklad 6.12.

V záznamu UČITEL *je jako CALC-klíč definována položka* ČU. *Načtení záznamu učitele s* ČU="U1" *provedeme příkazy*

```
učitel.ČU:="U1";
FIND učitel RECORD BY CALC-KEY;
GET učitel;
```

Příkaz vezme v pracovní oblasti hodnoty definované jako CALC, z nich spočítá CALC-KEY a najde příslušný výskyt záznamu.

♦

- **Vyhledání dalších výskytů záznamů se stejnými hodnotami CALC- klíčů:**

  FIND DUPLICATE T RECORD BY CALC-KEY

Příklad 6.13.

Předpokládejme, že při deklaraci záznamu Úvazek jsme použili způsob umístění pomocí CALC-KEY, povolili více výskytů záznamů daného typu se stejnými hodnotami CALC-KEY a ČU *je definováno jako CALC-KEY tohoto záznamu a* ČP *je definováno jako CALC-KEY záznamu* Předmět. *Pak názvy všech předmětů, které učí učitel U1 zjistíme takto:*

```
Úvazek.ČU:="U1";
FIND Úvazek RECORD BY CALC-KEY;
WHILE (DB-STATUS=0) DO
   BEGIN
      GET Úvazek;
      Předmět.ČP:= Úvazek.ČP;
      FIND Předmět RECORD BY CALC-KEY;
      GET Předmět;
      PRINT (Předmět.ČP, Předmět.název);
      FIND DUPLICATE Úvazek RECORD BY CALC-KEY
  END;
```

Všimněme si, že cyklus hledání v DB zastavujeme testem na DB-STATUS, kam příkaz FIND DUPLICATE vloží jedničku, jestliže se další výskyt hledaného záznamu nenašel. ♦

➢ **Selektivní vyhledávání podle předem zadaných hodnot položek:**

FIND T RECORD USING POL

Příklad 6.14.

V záznamu STUDENT *hledáme ulici, kde bydlí student NOVÁK*

```
Student.jméno:="NOVÁK";
FIND Student RECORD USING jméno;
GET Student;
PRINT(Student.ulice);
```

♦

➢ **Vyhledání dalších záznamů se stejnými hodnotami zadané položky**

FIND DUPLICATE T RECORD USING POL

Příklad 6.15.

Chceme vypsat všechna jména studentů, kteří bydlí na koleji.

```
Student.město:="Opava";
FIND Student RECORD USING město ;
WHILE (DB-STATUS=0) DO
  BEGIN
    GET Student ;
    PRINT(Student.jméno) ;
    FIND DUPLICATE Student RECORD USING město
  END;
```

♦

CD-ROM

Na CD-ROMu s tímto výukovým textem jsou animované příklady na právě probrané případy použití příkazu FIND a GET pojmenované:

- Animace\sitovy_model\find by db - key.exe
- Animace\sitovy_model\find by CALC - key.exe
- Animace\sitovy_model\find using pol.exe
- Animace\sitovy_model\find duplicate using pol.exe

❑ Vyhledání záznamů uvnitř výskytu setu

Pokud při hledání výskytu záznamu víme, do kterého setu záznam patří, hledání se výrazně urychlí. Hledat můžeme buď v právě aktuálním setu (CURRENT SET) nebo v setu daném vlastníkem (OWNER). Při hledání se pak prochází jen krátký zřetězený seznam vlastníka a členů setu.

➢ **Vyhledání vlastníka aktuálního výskytu zadaného setu:**

FIND OWNER OF CURRENT S SET

➢ **Vyhledání prvního člena aktuálního výskytu zadaného setu:**

FIND FIRST T RECORD IN CURRENT S SET

- **Vyhledání následujícího výskytu záznamu uvnitř aktuálního výskytu setu**

FIND NEXT T RECORD IN CURRENT S SET

Příklad 6.16.

Vypište seznam všech předmětů, které učí učitel Horák.

Nejprve je vyhledán výskyt záznamu Učitel *pro učitele Horáka a tím se zároveň stal aktuálním výskytem setu* Učitel_Učí *ten výskyt setu, jehož vlastníkem je učitel Horák. Vyhledáme první členský záznam tohoto výskytu setu* Učitel_Učí *a v cyklu do vyčerpání všech členů provádíme tyto akce:*

(1) vyhledáme vlastníka lokalizovaného záznamu Učitel_Učí *v rámci setu* Učí_Předmět*; vlastníkem je záznam Předmět;*

(2) vytiskneme název předmětu;

(3) vyhledáme následující výskyt záznamu Učitel_Učí *v aktuálním výskytu setu* Učí_Předmět.

```
Učitel.jméno:="Horák";
FIND Učitel RECORD USING jméno;
FIND FIRST Úvazek RECORD IN CURRENT Učitel_učí SET
WHILE (DB-STATUS=0) DO
   BEGIN
     FIND OWNER OF CURRENT Předmět_učí SET;
     GET Předmět;
     PRINT (Předmět.název);
     FIND NEXT Úvazek RECORD IN CURRENT Účitel_učí SET
   END;
```

♦

- **Selektivní vyhledávání záznamů uvnitř aktuálního výskytu setu:**

FIND T RECORD IN CURRENT S SET USING POL

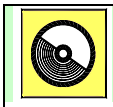
FIND DUPLICATE T RECORD IN CURRENT S SET USING POL

Příklad 6.17.

Vypište čísla všech učitelů, kteří učí předmět „Logika“ 4 hodiny týdně.

```
Předmět.název:= „logika“;
FIND Předmět RECORD USING název;
Úvazek.hodin:= 4;
FIND Úvazek  RECORD IN CURRENT Před-učí SET USING hodin;
WHILE (DB-STATUS = 0) DO
   BEGIN
     GET Úvazek;
     PRINT (Úvazek.ČU);
     FIND DUPLICATE Úvazek RECORD IN CURRENT Před-učí SET USING hodin;
   END;
```

♦



CD-ROM

Na CD-ROMu s tímto výukovým textem jsou animované příklady na právě probrané případy použití příkazu FIND uvnitř setu pojmenované:

- Animace\sitovy_model\find owner.exe
- Animace\sitovy_model\find in current set.exe

➢ **Vyhledání záznamů podle hodnot z tabulky aktuálních ukazatelů**

Následující příkazy nahrazují dosazovací příkaz, prováděný s aktuálními ukazateli záznamů a setů v rámci pracovní oblasti uživatele.

Nastavení aktuálního záznamu nějakého typu záznamu za aktuální záznam programu:

FIND CURRENT OF T RECORD

Nastavení aktuálního záznamu některého typu setu za aktuální záznam programu:

FIND CURRENT OF S SET

Příklad 6.18.

Máme zjistit, zda učitel U1 učí předmět s názvem LOGIKA a pokud ano, v jakém rozsahu.

```
Učitel.ČU:="U1";
FIND Učitel RECORD BY CALC-KEY;
neučí:=TRUE;
FIND FIRST Úvazek RECORD IN CURRENT Učitel-učí SET;
WHILE (DB-STATUS=0) AND neučí DO
   BEGIN
     FIND OWNER OF CURRENT Učí_předmět SET;
     GET Předmět;
      IF Předmět.název = "LOGIKA"
        THEN BEGIN
                FIND CURRENT OF Úvazek RECORD;
                GET Úvazek;
                PRINT(Úvazek.hodin);
                neučí:=FALSE
             END
       ELSE FIND NEXT Úvazek RECORD IN CURRENT Učitel-učí SET
   END; ♦
```

❑ **Ukládání nových záznamů**

Příkaz STORE

Při ukládání výskytu záznamu do DB nejprve vytvoříme výskyt záznamu v pracovní uživatelské oblasti a odtud ho příkazem STORE do DB uložíme. Příkaz má tvar

STORE T

Ukládaný záznam se stane aktuálním záznamem programu a je zařazen do všech setů, v nichž je deklarován jako člen typu AUTOMATIC.

Příklad 6.19.

Do databáze FAKULTA přidáme další výskyt záznamu učitele do záznamu UČITEL:

```
učitel.jméno:="NOVOTNÝ";
učitel.ČU:=123;
učitel.funkce:="Docent";
STORE učitel ; ♦
```

❑ Modifikace záznamů

Příkaz MODIFY má formát

MODIFY T

Celý postup při modifikaci komponent výskytu záznamu znamená nejprve záznam v DB najít příkazem FIND, načíst do pracovní oblasti příkazem GET, tam opravit příslušné komponenty záznamu a konečně pomocí příkazu MODIFY zapsat opět zpátky.

Příklad 6.20.

Student Novotný se přestěhoval do ulice Zelené 23.

```
Student.jméno:="NOVOTNÝ";
FIND Student USING jméno ;
GET Student ;
Student.ulice:="Zelená 23" ;
MODIFY Student ; ♦
```

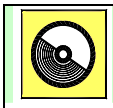
❑ Zrušení záznamů

Příkaz ERASE má formát

ERASE T [PERMANENT]
[SELECTIVE]
[ALL]

vymaže výskyt záznamu z DB.

ERASE T zruší aktuální záznam programu, pokud není vlastníkem neprázdného výskytu nějakého setu. Je-li uvedeno PERMANENT a aktuální záznam programu je vlastníkem výskytů nějakých setů, pak jsou spolu s uvedeným záznamem zrušený také všechny členské záznamy, jejichž druh členství je MANDATORY. SELECTIVE navíc zruší všechny členské záznamy s druhem členství OPTIONAL, pokud nejsou zařazeny v žádném jiném setu. ALL zruší všechny členské záznamy bez ohledu na druh členství a zařazení v jiných setech.



CD-ROM

Na CD-ROMu s tímto výukovým textem jsou animované příklady na právě probrané případy použití příkazů STORE, MODIFY, ERASE pojmenované**:**

- Animace\sitovy_model\store.exe
- Animace\sitovy_model\modify.exe
- Animace\sitovy_model\erase.exe

❑ Manipulace se záznamy v rámci setu

Kromě modifikace dat ve výskytech záznamů je možno také **modifikovat struktury setů,** připojovat nové členské záznamy, rušit je (ze setu, ne z DB), případně přesouvat z jednoho setu do jiného. Mechanizmus připojování záznamů však je složitější. Při definici setu můžeme zadat, zda se budou záznamy připojovat

AUTOMATIC . .. příkazem STORE
MANUAL ... příkazem CONNECT

Mimo to můžeme předepsat, na kterou pozici do setu se nově vkládaný záznam umístí: FIRST, LAST, NEXT, PRIOR, SORTED

➢ Zařazení aktuálního záznamu programu za aktuální výskyt zadaného setu

Příkaz CONNECT má formát

CONNECT T TO S

Příkaz nelze použít pro druh členství MANDATORY AUTOMATIC.

Příklad 6.21.

Uvažme set BYDLÍ *se záznamem typu Kolej (CALC-klíč je název) jako vlastníkem a se záznamem typu* Student *(CALC-klíč je jmeno) jako členem. Máme zařadit studentku se jménem Kosová Helena na kolej Opletalovu.*

```
Kolej.název:="Opletalova";
FIND Kolej RECORD USING název;
Student.jméno:="Kosová Helena";
FIND Student RECORD USING jméno;
CONNECT Student TO Bydlí;
```

➢ Vyřazení aktuálního záznamu programu z výskytu zadaného setu

Příkaz DISCONNECT má formát

DISCONNECT T FROM S

Druh členství v setu musí být OPTIONAL.

Příklad 6.22.

Vyloučení studenta Jana Skřivánka z koleje Mánesova

```
Kolej.název:="Mánesova";
FIND Kolej RECORD USING název;
Student.jméno:="Skřivánek Jan";
FIND Student RECORD USING jméno;
DISCONNECT Student FROM bydlí;
```

➢ Přesunutí aktuálního výskytu záznamu z původního výskytu do zadaného výskytu setu.

Příkaz RECONNECT má formát

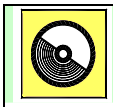
RECONNECT T TO S

Příkaz přesune aktuální výskyt záznamu z původního výskytu setu do výskytu setu aktuálního výskytu setu zadaného typu.

Příklad 6.23.

Student Novotný se přestěhoval z koleje Opletalovy do koleje Mánesovy.

```
Student.jméno:="Novotný";
FIND Student RECORD USING jméno;
Kolej.název:="Mánesova";
FIND Kolej RECORD USING název;
RECONNECT Student TO bydlí
```



CD-ROM

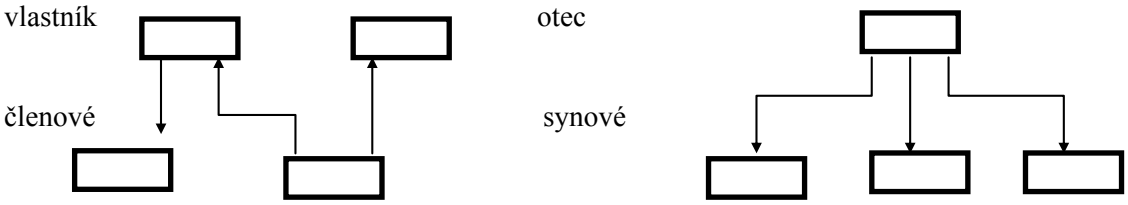
Na CD-ROMu s tímto výukovým textem jsou animované příklady na právě probrané případy použití příkazů CONNECT, DISCONNECT a RECONNECT pro setu pojmenované:

- Animace\sitovy_model\connect.exe
- Animace\sitovy_model\disconnect.exe
- Animace\sitovy_model\reconnect.exe

6.4. Hierarchický datový model

Hierarchický model byl používán hlavně v počátcích rozvoje databázových systémů. Je to vlastně zjednodušený síťový model. Zobrazíme-li záznamy jako uzly grafu a vazby realizované pomocí odkazů jako hrany grafu, odpovídá databázi síťového modelu obecný graf. Databázi sestavené na principech hierarchického modelu odpovídá graf zvaný strom (graf bez cyklů) nebo les (množina stromů).

Prakticky to znamená, že záznam nemůže patřit do více než jednoho setu. Při popisování hierarchického modelu se používá rodinné terminologie. Místo vlastník se užívá pojmu otec, místo člen pojmu syn.



síťový model hierarchický model

Rozdíl mezi síťovým a hierarchickým modelem v terminologii modelu hierarchického můžeme tedy formulovat takto: v hierarchickém modelu má každý syn právě jednoho rodiče, v síťovém modelu jich může mít více.

V literatuře je často uváděn hierarchický model jako samostatný třetí typ (vedle relačního a síťového). My jej nechápeme jako nový typ a nebudeme se jím podrobněji zabývat.

Shrnutí pojmů 6

Standard pro práci se síťovým datovým modelem CODASYL.
Schéma, subschéma, typ záznamu, výskyt záznamu, typ setu, výskyt setu. Databázový klíč.

Realizace vazeb v SDM.
Jazyk pro definici dat, deklarace databáze, typů záznamů, typů setů.
Pracovní oblast uživatele.
Jazyk pro manipulaci s daty. Příkaz pro vyhledání záznamu a jeho typy. Načtení záznamu.
Uložení, modifikace a rušení záznamu.
Práce s prvky setu. Uložení, přemístění a zrušení člena setu.

Otázky 6.

1. Jaký je základní rozdíl mezi databází v relačním a síťovém datovém modelu?
2. Jak se realizují vazby v síťovém modelu?
3. Co vše je možno definovat pomocí JDD?
4. Co je a k čemu je pracovní oblast uživatele?
5. Jak se v síťovém datovém modelu manipuluje s daty na rozdíl od relačního modelu?
6. Jaké možnosti vyhledávání dat jsou v síťovém modelu?
7. Jak je možno urychlit vyhledávání pomocí setů?
8. Co je hierarchický datový model a jak se liší od síťového?

Úlohy k řešení 6.

5. V databázi FAKULTA deklarované v odstavci 6.2.

   Typy záznamů: Učitel (ču, jméno, funkce, plat)
   Předmět (čp, název, ročník)
   Úvazek (ču, čp, hodin)
   Student (jméno, město, ulice)
   Kolej (název)

   Typy setů: UČITEL_UČÍ (Učitel, Úvazek)
   PŘEDMĚT_UČÍ (Předmět, Úvazek)
   BYDLÍ (Kolej, Student)

   vyhledejte následující informace:

   a) Jméno a funkci učitele s číslem Č17.
   b) Seznam učitelů, kteří jsou docenty.
   c) Seznam předmětů 3. ročníku.
   d) Seznam všech studentů bydlících na koleji „Studentská“.
   e) Úvazek učitele Horáka – předmět a počet hodin.
   f) Seznam učitelů, kteří učí předmět Databáze.
   g) Jména a adresy všech studentů.

KLÍČ K ŘEŠENÍ VYBRANÝCH ÚLOH

Kapitola 1.1.

Řešení úlohy 1.

Zamestnanec(RČ, jmeno, fce, mzda, ucet_banky, pojistovna, social, zdrav, jazyk:multi, vzdelani)

Řešení úlohy 2.

1. evidence domácích kazet (audio, video) a programů na nich
   Kazeta(cislo_kaz, typ_aud_vid, zanr, delka, vypujcil, datum_vypujcky, datum_vraceni)
   Program (cislo_kaz, zacatek, konec, obor, autor, nazev, interpret)
2. evidence knih ve veřejné knihovně
   Titul (ISBN, autor, nazev, vydavatel, cena)
   Exemplar (cislo_exemplare, ISBN, datum_vypujcky, ctenar, datum_vraceni)
3. evidence aut v pujcovne
   Auto (SPZ, vyrobce, typ, obsah, počet_sedadel, cena_den)
   Zákazník (RČ, jmeno, adresa)
   Vypujcka (RČ, SPZ, datum_od, datum_do)

Řešení úlohy 3.

1. Najdi kazety, které si vypůjčil František Černý,
   na kterých je nahraná dechovka.
   Najdi nahrávky (programy) Karla Černocha,
   všechny skladby J.S. Bacha,
   skladby o délce 25 minut.

Řešení úlohy 4.

1. Kazety, které si vypůjčil František Černý: v tabulce **Kazeta** hledám ve sloupci **vypujčil** jméno **František Černý**, ve vyhledaných řádcích vypíši číslo kazety a datum výpůjčky.

   Kazety s dechovkou: v tabulce **Kazeta** hledám ve sloupci **zanr** text **dechovka**, ve vyhledaných řádcích vypíši číslo kazety.

   Nahrávky Karla Černocha: v tabulce **Program** hledám ve sloupci **interpret** jméno **Karel Černoch**, ve vyhledaných řádcích vypíši číslo kazety, název skladby, začátek a konec skladby.

   Skladby J.S. Bacha: v tabulce **Program** hledám ve sloupci **autor** jméno **J.S. Bach**, ve vyhledaných řádcích vypíši číslo kazety, název skladby, začátek a konec skladby.

   Skladby délky 25 min: v tabulce **Program** hledám ve sloupcích **začátek** a **konec** rozdíl 25 min, ve vyhledaných řádcích vypíši číslo kazety, název skladby, začátek a konec skladby.

Kapitola 3.1.

Řešení úlohy 1.

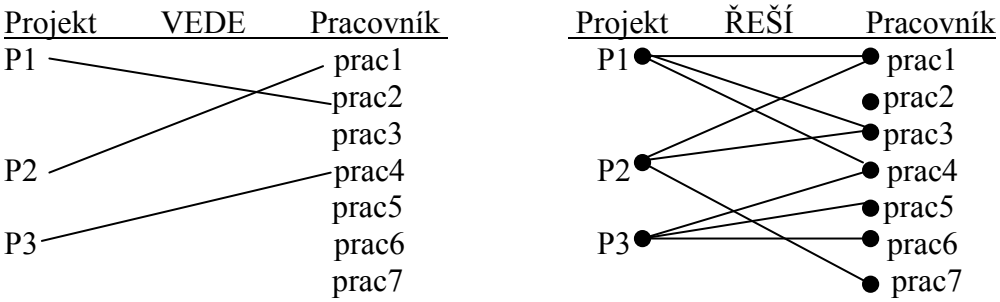
a) Bude-li potřeba pracovat (třídit, vyhledávat, formátovat ap.) samostatně s křestním a příjmením nebo s tituly (například setřídit učitele podle titulů, vybrat všechny PhD. ap.), pak je vhodné rozdělit úplné jméno na 3 nebo více atributů: krestni, prijmeni, titul_pred, titul_za). Pokud stačí jen

identifikace člověka bez práce s částmi jména, zvolíme jediný atribut: jmeno. K němu bude v datovém slovníku popsán způsob vyplnění: prijmeni, mezera, křestní, čárka, titul_před, čárka, titul_za.

Řešení úlohy 2.

Zvažte, zda přidání nebo ubrání nějakého atributu změní podstatu objektu. Dále zvažte, zda dva objekty (entity) stejného typu mohou mít totožnou množinu hodnot atributů. Víme, že každý konkrétní objekt musí být rozlišitelný od ostatních a tedy instance objektu musí mít podle schématu jednoznačný obraz. Přidávání nebo ubírání atributů jen zvyšuje nebo snižuje přesnost obrazu objektu.

Řešení úlohy 3.



Řešení úlohy 4.

a)

b)

c)

Řešení úlohy 5.

a)

Ne zcela jednoznačná situace, pokud má odrážet realitu. Dům má mít vlastníka, ale mohou být domy s neznámým vlastníkem. Osobou se zde chápeme fyzickou nebo právnickou osobu. Zde bude záležet na přesnějším zadání domů v uvažované databázi. Osoba nemusí být vlastníkem domu. Tedy: dům má povinné členství ve vztahu VLASTNÍ, osoba nepovinné.

b)

Dům nemusí být obýván, ale může být obýván více nájemníky. Nájemník je proto nájemník, že má najmutý byt v nájemním domě.

c)

Dům nemusí být obýván, ale může být obýván více nájemníky. Osoba (libovolná) nemusí obývat žádný dům, ale může obývat více domů.

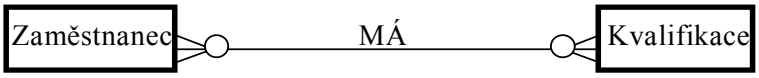
d)

Objednávka bez formulace toho, co se objednává (položky objednávky) nemá smysl, položka objednávky musí příslušet konkrétní objednávce.

e)

Zákazník může mít více účtů nebo žádný, jeden účet patří jedinému zákazníkovi.

f)



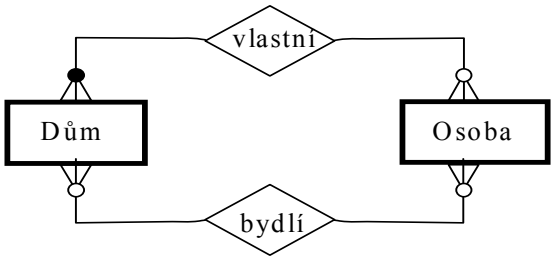
Zaměstnanec může mít žádnou, jednu nebo více kvalifikací, konkrétní kvalifikaci může mít žádný nebo více zaměstnanců.

Řešení úlohy 6.

a) Ano, může. Z výskytového diagramu (viz. str. 27) je patrný vznik vztahové entity.

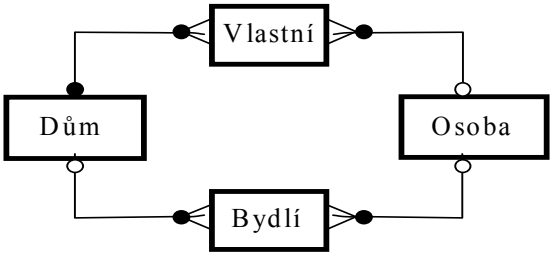
b) Obecně to není pravda, oba vztahy mezi X a Y, Y a Z mohou mít naprosto odlišný význam, například: Katedra – Zaměstnanec a Zaměstnanec – Projekt.

Řešení úlohy 7.



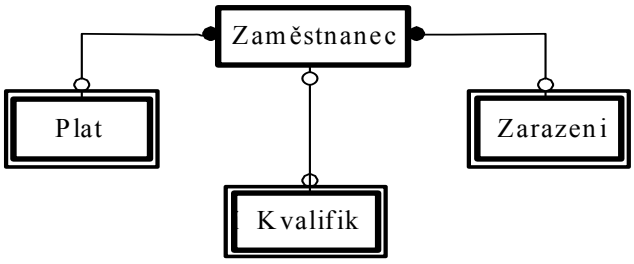
Dům (id_domu, adresa, počet_bytů)
Osoba (rod_cis, jméno)
VLASTNÍ (id_domu, rod_cis, podíl)
BYDLÍ (id_domu, rod_cis, číslo_bytu)

Po dekompozici vztahů typu M:N:



Řešení úlohy 8.

Zaměstnanec (rodne_cis, krestni, prijmeni, adresa, mesicni_plat**: multi**, rocni_plat, kvalifik**:multi**, zarazeni (datum_zaraz, funkce)**: multi**)
⇓
Zaměstnanec (rodne_cis, krestni, prijmeni, adresa, rocni_plat)
Plat (rodne_cis, mesic, plat)
Kvalifik (rodne_cis, kval_stupen)
Zarazeni (rodne_cis, datum_zaraz, funkce)



Kapitola 4.1.

Řešení úlohy 1.

Příklad 4.3.

Učitel (ČU, jmeno, fce, plat)
Předmět (ČP, název)
UČÍ (Učitel, Předmět) ⇒ Učí (ČU, ČP, hodin)
Protože jde o vazbu M:N, nelze vazební tabulku Učí spojit s některou z původních.

Příklad 4.4.

Učitel (ČU, jmeno, fce, plat)
Katedra (ČK, název_kat)
ČLEN_KAT(Učitel, Katedra) ⇒ Člen_kat (ČU, ČK)
Vazební tabulku Člen_kat je možno sloučit s tabulkou Učitel, ztotožnit sloupec ČU a připojit sloupec ČK jako cizí klíč:

Učitel

ČU	**jmeno**	**fce**	**plat**	**ČK**
U1	Adam ...	OA	3000	K2
U2	Beneš ...	prof	4000	K1
U3	Douda ...	doc	3500	K1
...				

Katedra

ČK	**název_kat**
K1	
K2	
K3	
...	

Příklad 4.5.

Učitel (ČU, jmeno, fce, plat)
Předmět (ČP, název)
Třída (kod_třídy)
Místnost (kod_místn)
Hodina (kod_hod)

ROZVRH(Učitel,Předmět,Třída,Místnost,Hodina) ⇒Rozvrh(ČU,ČP, kod_třídy, kod_místn, kod_hod)

Učitel

ČU	jmeno	...
U1	Adam ...	
U2	Beneš ...	
U3	Douda ...	
U4	Fousek	
...		

Předmět

ČP	nazev
P1	Matematika
P2	Fyzika
P3	Český jazyk
P4	Dějepis
...	

Třída

kod_třídy
1.A
1.B
2.A
3.A
...

Místnost

kod_místn
A101
A102
A103
B201
...

Hodina

kod_hod
Po-8
Po-9
Po-10
Po-11
...

Rozvrh

ČU	ČP	kod_tridy	kod_mistn	kod_hod
U1	P4	3.A	A102	Po-8
U1	P2	4.B	A103	Po-10
U3	P2	4.A	B201	Po-8
U3	P1	3.A	A102	Ut-10
...				

Kapitola 4.2.1.

Řešení úlohy 1.

a) Učí [ČU]
b) Učitel [ČU]- Učí [ČU]
c) Učí (ČP='Překladače')[ČU]
d) Učí [ČU] – Učí (ČP = 'Překladače')[ČU]
e) Učitel [ČU] – Učí (ČP = 'Překladače')[ČU]
f) Učí (ČP <> 'Překladače')[ČU]
g) Učí [ČU]-V(ČP <> 'Překladače')[ČU]

Řešení úlohy 2.

a) Lékař [specializace]
b) Lékař (specializace = ‚ortopédie‘) [jménoL]
c) Pacient (dat_narození < 1.1.1939) [jménoP]
d) (Návštěva [*] Pacient) (jménoP = ‘Nová Marie’) [číslo_licence]
e) (Návštěva [*] Lékař) (typ = ‘doma‘) [jménoL]
f) (Pacient[*]Návštěva[*]Lékař)(jménoL=‘Lom‘∧datum =’23.5.2003‘) [jménoP,adresa]
g) (Návštěva [*] Pacient) (diagnóza =‘HIV‘) [jménoP, adresa]
h) (Návštěva [*] Lékař) (diagnóza = ‘vřed_na_dvanáct‘) [jménoL, specializace]
i) (Pacient[*]Návštěva[*]Lékař)(jménoL=‘Čermák‘)[jménoP,adresa] **-**
(Pacient[*]Návštěva[*]Lékař)(jménoL <> ‘Čermák‘)[jménoP,adresa]

Poznámka: některé příklady mají více než jedno řešení, zde uvedené. Najděte další, lepší řešení a zdůvodněte, která a proč jsou lepší.

Kapitola 4.2.3.

Řešení úlohy 1.

```
a) SELECT jméno, adresa FROM Čtenář
b) SELECT jméno, adresa
      FROM Čtenář
      ORDER BY jméno
c) SELECT *
      FROM Exemplář
      WHERE koupeno > '20.3.1992'
   nebo s úplným výpisem knihy
   SELECT K.*, PŘÍR, koupeno, cena
      FROM Exemplář E, Knihy K
      WHERE E.SIGN = K.SIGN AND koupeno > '20.3.1992'
d) SELECT DISTINCT autor
     FROM Knihy K, Rezervace R
     WHERE K.ISBN=R.ISBN AND rezervováno < '31.12.92'
e) SELECT A.jméno, B.jméno
      FROM Čtenář A, Čtenář B
      WHERE A.adresa=B.adresa AND A.číslo > B.číslo
f) SELECT E.ISBN, PŘÍR, název, cena*10/8
      FROM Exemplář E, Knihy K
      WHERE E.ISBN = K.ISBN AND země = 'Slovensko'
g) SELECT COUNT(*)
      FROM Čtenář
h) SELECT COUNT(DISTINCT číslo)
      FROM Rezervace
i) SELECT COUNT(*)
       FROM Knihy
       WHERE země = 'Slovensko'
j) SELECT SUM(cena)
      FROM Exemplář E, Knihy K
      WHERE E.ISBN = K.ISBN AND země = 'Slovensko'
k) SELECT jméno,COUNT(číslo)
      FROM Výpůjčky V, Čtenář Č
      WHERE V.číslo = Č.číslo
      GROUP BY  V.číslo
l) SELECT jméno,COUNT(číslo)
      FROM Výpůjčky V, Čtenář Č
      WHERE V.číslo = Č.číslo
      GROUP BY V.číslo
      HAVING COUNT(V.číslo) > 5
m) SELECT ISBN, název, země
      FROM Knihy
      WHERE země IN ("SR","Rusko","MR")
n) SELECT DISTINCT číslo
      FROM Rezervace
      WHERE rezervováno<"31.12.1992" AND
            číslo IN (SELECT číslo FROM Výpůjčky)
o) SELECT jméno
     FROM Čtenář
     WHERE číslo IN (SELECT číslo
```

```
FROM Rezervace
WHERE ISBN = (SELECT ISBN
                FROM Knihy
                WHERE název="Babička"))
```

Kapitola 4.4.

Řešení úlohy 1.

RU(RČ, jmeno, adresa, poj, tel, kodvyk, nazvyk, cena, datum ,cas)
F = {RČ → jmeno, adresa, poj
kodvyk → nazvyk,cena
datum, cas → RČ, kodvyk}

Pacient (RČ, jmeno, adresa, poj)
Vykon (kodvyk, nazvyk, cena)
Objed (RČ, datum, cas) ⇒ Provedeno (RČ, datum, cas, kodvyk)
Kalendar (datum, cas)

Řešení úlohy 5.

a) triviální: W → W ... W+ = {W}
dáno: W→ Z ... W+ = {W, Z}
dáno: WZ → Y ... W+ = {W, Z, Y}
dáno: YZ → X ... W+ = {W, Z, Y, X} ⇒ W je klíč ⇒ WZ je nadklíč

triviální: YZ → YZ ... YZ+ = {Y, Z}
dáno: YZ → X ... YZ+ = {Y, Z, X} ... na levé straně daných závislostí se již další podmnožiny z YZ+ nevyskytují.

c) X+ = {X} ... X není na levé straně žádné závislosti, není na něm závislý žádný další atribut, tedy ani YZ.

Řešení úlohy 6.

F+ : W+ = {W, Z,Y, X}
Z+ = {Z}
Y+ = {Y}
X+ = {X} a další varianty zápisu těchto závislostí.

a) (F – {W → Z})+ : W+ = {W, Y} ... různé od F+, proto W → Z není redundandní
b) (F – {W → Y})+: W+ = {W, Z, Y, X}, Z+ = {Z}, Y+ = {Y}, X+ = {X} ... stejné s F+, proto W → Y je redundandní.
c) dáno F‘ ={W→Z, YZ→X, WZ→Y}
F‘+ = F+
(F‘-{YZ→ X})+ : W+ = {W, Z, Y} ... různé od F‘+, proto YZ→X není redundandní
obdobně pro WZ→Y.

Řešení úlohy 7.

A→C, A→D, D→B

Řešení úlohy 10.

a) redundandní je BE→A
b) redundandní je AD→E, BD→A
c) redundandní je AD→B

Kapitola 4.5.

Řešení úlohy 1.

a) klíče: CE, DE

b) rozklad není správný, chybí atribut E

c) rozklad není správný, není pokryta závislost C → A

Řešení úlohy 2.

Rozklad je správný, protože

1. XYU ∩ YZUV = YU, XYU - YZUV = X, YZUV - XYU = ZV,
   do F+ patří YU → ZV
2. každá FZ patří do některé projekce R1 nebo R2

Kapitola 4.6.

Řešení úlohy 1.

a) A+ = {AC}, B+ = {BD}, C+ = {C}, D+ = {D}, AC+ = {ACBD}, AB+ = {ABCD}, AD+={ADC},

klíče: AC, AB
primární atributy: A,B,C
sekundární: D
schéma je v 1NF (existuje závislost na podklíči B→ D a A→ C mezi primárními atributy)

b) A+ = {A}, B+ = {B}, C+ = {CAE}, D+ = {DAB}, E+ = {E}, CD+={CDAEB},

klíče: CD
primární atributy: C,D
sekundární: A,B,E
schéma je v 1NF (existuje závislost na podklíči C→ E a další)

c) A+ = {AE}, B+ = {B}, C+ = {CD}, D+ = {D}, E+ = {E}, AC+={ACED}, ACB+={ACBED},

BCD+={ABCDE},

klíče: ACB, BCD
primární atributy: A,B,C,D
sekundární: E
schéma je v 3NF, není v BCNF (existuje závislost mezi primárními atributy C→ D)

d) klíč je DE, schéma je v BCNF

e) klíč je AB, schéma je v 2NF (existuje závislost mezi sekundárními atributy C→ DE)

Kapitola 4.7.

Řešení úlohy 1.

R1(X, Y, U), R2 (Y, V), R3 (U, Z)

Řešení úlohy 2.
R1(A, B, C), R2 (A, D), R3 (D, B)

Řešení úlohy 3.

Dáno: Prodej (knihkupec, ISBN, adresa, cena) a další závislost **F** = {ISBN → cena}

Cena je atribut tranzitivně závislý na klíči, protože:
knihkupec,ISBN → ISBN → cena
což je v rozporu s definicí 3NF.

Vhodná dekompozice:
Prodejna (knihkupec, ISBN, adresa), Ceník (ISBN, cena)

Zrušení zadané funkční závislosti o ISBN a ceně způsobí, že nebude existovat závislost ceny na podklíči, původní schéma Prodej bude ve 3NF a dekompozice nebude potřeba. Bude platit jen
knihkupec, ISBN → adresa, cena
což znamená, že každý knihkupec prodává každou knihu za svou cenu.

Kapitola 6.

Řešení úlohy 1.

a)
```
Učitel.ČU:="Č17";
FIND Učitel RECORD USING jméno;
GET Učitel;
PRINT(Učitel.jméno, Učitel.funkce);
```

b)
```
Učitel.funkce:="docent";
FIND Učitel RECORD USING funkce;
   WHILE (DB-STATUS=0) DO
     BEGIN
     GET Učitel;
          PRINT(Učitel.jméno);
     FIND DUPLICATE Učitel RECORD USING funkce
     END;
```

c)
```
Předmět.ročník:="3";
FIND Předmět RECORD USING ročník;
   WHILE (DB-STATUS=0) DO
     BEGIN
     GET Předmět;
          PRINT(Předmět.název);
     FIND DUPLICATE Předmět RECORD USING ročník
     END;
```

LITERATURA

1. DATE, C. J.: *An Introduction to Datebase Systems.* Addison-Wesley Publishing Company, USA, 1990.
2. DATASEM'92 - DATASEM'99. *Sborníky seminářů*. CS-Compex, Brno, 1992-1999.
3. DATAKON'2000 - DATAKON'2003. *Sborníky seminářů*. Brno, 2000 – 2003
4. FARANA, R. - VOJÁČEK, M.: *Databázové systémy. Microsoft Access*. VŠB-TU, Ostrava, 1994.
5. HAVLÁT, T. - BENEŠOVSKÝ, M.: *Úvod do databázových systémů.* PF MU, Brno, 1987.
6. KROHA P.: *Báze dat*. FE ČVUT, Praha, 1990.
7. LUKASOVÁ, A. *Úvod do databázové technologie*. Ostrava: Ostravská Univerzita 1993
8. MODERNÍ DATABÁZE. *Sborníky seminářů*, Dům techniky, Ústí nad Labem, 1995-2003.
9. POKORNÝ, J.: *Databázové systémy a jejich použití v informačních systémech.* Academia, Praha, 1992.
10. POKORNÝ, J.: *Databázová abeceda*. Science, Praha, 1998.
11. POKORNÝ, J.: *Dotazovací jazyky*. Science, Praha, 1994.
12. POKORNÝ, J.: *Počítačové databáze*. Kancelářské stroje, Praha, 1991.
13. POKORNÝ, J.: *Učíme se SQL*. PLUS, Praha, 1993.
14. POKORNÝ, J. - HALAŠKA, I.: *Databázové systémy. Vybrané kapitoly a cvičení*. FE ČVUT, Praha, 1998.
15. RICHTA, K. - SOCHOR, J.: *Projektování programových systémů*. FE ČVUT, Praha, 1994.
16. SCHREBER, A.: *Databázové systémy*. Alfa, Bratislava, 1988.
17. STAUDEK, J. - POKORNÝ, J. - BRODSKÝ, J.: *Operační a databázové systémy.* FE VUT, Brno, 1991.
18. ŠARMANOVÁ, J. *Teorie zpracování dat.* Ostrava: Vysoká škola báňská 1997
19. TIETZE, P.: *Strukturální analýza, úvod do projektu řízení*. Grada, Praha, 1992.
20. TSICHRITZIS, D.C. - LOCHOVSKY, F.H.: *Databázové systémy*. SNTL, Praha, 1987.
21. WIRTH, N.: *Algoritmy a štruktúry údajov.* Alfa, Bratislava, 1989.