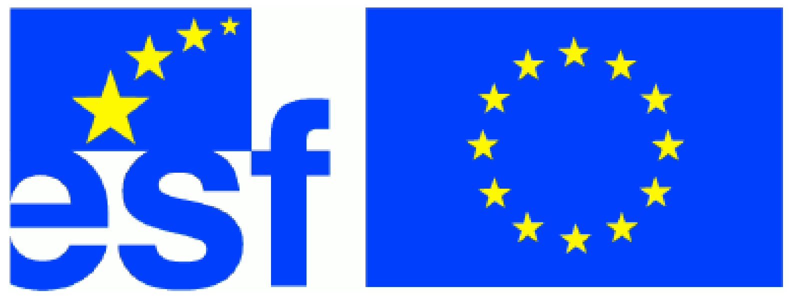
KATEDRA INFORMATIKY
PŘÍRODOVĚDECKÁ FAKULTA
UNIVERZITA PALACKÉHO

DATABÁZOVÉ SYSTÉMY

JIŘÍ HRONEK



VÝVOJ TOHOTO UČEBNÍHO TEXTU JE SPOLUFINANCOVÁN
EVROPSKÝM SOCIÁLNÍM FONDEM A STÁTNÍM ROZPOČTEM ČESKÉ REPUBLIKY

Olomouc 2007

Abstrakt

Tento učební text seznamuje čtenáře se základy databázové technologie. Obsahuje stručný úvod do problematiky s definicí základních pojmů, architektury, návrhu a použití databázových systémů. Postupně je čtenář seznámen se specifikou databázových dat a modelů, databázovými jazyky pro definici dat, manipulaci s daty a hlavně pro dotazování, souběžné zpracování transakcí a zajištění konzistence při poruchách. Dále jsou popsány některé postupy, související s modelováním dat a návrhem databáze. Z problematiky fyzické implementace jsou vybrány partie týkající se optimalizace uložení, přístupu k datům a implementace operací s daty. Kde je to účelné, jsou použity ilustrativní příklady. Text nemůže nahradit rozsáhlé monografie a měl by usnadnit orientaci při detailnějším studiu.

Cílová skupina

Text je primárně určen pro posluchače třetího ročníku bakalářského studijního programu Aplikovaná informatika na Přírodovědecké fakultě Univerzity Palackého v Olomouci.. Text předpokládá znalosti základních matematických pojmů, logiky 1. řádu, teorie grafů a algoritmizace, základy z operačních, distribuovaných a paralelních systémů.

Obsah

1 Základní pojmy ... 9
1.1.1 Pojem databáze ... 9
1.1.2 Úrovně abstrakce ... 14
1.1.3 Historický vývoj zpracování dat ... 15
1.1.4 Systém Řízení Báze Dat – SŘBD ... 18
1.1.5 Architektura DBS ... 19
2 Konceptuální modelování ... 24
2.1 Analýza a návrh IS ... 25
2.2 ER Model ... 25
2.3 Konceptuální model HIT ... 31
3 Logické modely dat ... 33
3.1 Síťový databázový model ... 33
3.2 Hierarchický databázový model ... 35
3.3 Relační model ... 36
3.3.1 Relační struktura dat ... 37
3.3.2 Integritní omezení v relačním modelu ... 38
3.3.3 Doporučení pro transformaci ER diagramu do relačního modelu ... 39
3.4 Objektově-relační model ... 43
3.5 Objektový model ... 43
3.5.1 Objektový koncept ... 43
3.5.2 Úvod do ODL ... 45
3.6 Další databázové modely ... 47
4 Relační databázové a dotazovací jazyky ... 50
4.1 Relační algebra ... 50
4.2 N-ticový relační kalkul ... 54
4.3 Doménový relační kalkul ... 55
4.4 Datalog ... 56
4.5 Jazyk SQL ... 57
4.5.1 Příkazy pro definici dat ... 57
4.5.2 Příkazy pro modifikace dat ... 59
4.5.3 Výrazy a funkce, agregované funkce ... 61
4.5.4 Agregační funkce se skupinou dat ... 62
4.5.5 Podotázky ... 63
4.5.6 Pohledy ... 63

4.5.7 Další možnosti SQL 64
4.6 Jazyk QBE 67
4.7 Úvod do OQL 68
5 Fyzická organizace dat 72
5.1 Databázový přístup k datům 72
5.2 Organizace souborů 74
5.2.1 Sekvenční soubory 75
5.2.2 Setříděné sekvenční soubory 76
5.2.3 Indexování 76
5.2.4 Soubory s přímým adresováním - hašování 80
5.2.5 Shlukování – clustering 81
5.2.6 Indexování pomocí binární matice 81
5.2.7 Soubory s proměnnou délkou záznamu 82
6 Zpracování dotazu 85
6.1 Základní etapy 85
6.2 Optimalizace dotazu 86
6.3 Implementace operací, metody pro výpočet spojení 90
6.3.1 Hnízděné cykly (Nested-loop join) 90
6.3.2 Indexované hnízděné cykly 91
6.3.3 Hašováné spojení (Hash Join) 91
6.3.4 Vícenásobné spojení 92
6.3.5 Další operace 93
6.3.6 Vyhodnocení stromového výrazu dotazu 93
7 Formalizace návrhu relační databáze, normalizace 95
7.1 Problémy návrhu schématu relační databáze 95
7.2 Funkční závislosti 95
7.2.1 Armstrongovy axiomy 96
7.2.2 Určení uzávěru FZ pro podmnožiny atributů relace 98
7.2.3 Určení příslušnosti funkční závislosti k uzávěru množiny FZ 99
7.2.4 Určení neredundantního pokrytí pro množinu elementárních funkčních závislostí. 99
7.3 Dekompozice relačních schémat 99
7.4 Normální formy relací 101
7.5 Postup návrhu schématu relační databáze 104
8 Transakční zpracování, paralelismus a zotavení po poruše 108
8.1 Koncept transakce 108
8.2 Ochrana proti porušení konzistence dat 109
8.3 Řízení paralelního zpracování transakcí 112

8.3.1 Problémy paralelního přístupu ... 113
8.3.2 Uzamykací protokoly ... 116
8.3.3 Uváznutí ... 118
8.3.4 Řešení problému uváznutí ... 120
9 Distribuované databázové systémy ... 123
9.1 Vlastnosti distribuovaných databází ... 123
9.1.1 Klasifikace DDBS ... 124
9.2 Rozmístění dat v DDBS ... 126
9.3 Paralelní zpracování v distribuovaných databázích ... 128
10 Závěr ... 131
11 Seznam literatury ... 132
12 Seznam obrázků ... 133
13 Rejstřík ... 134

1 Základní pojmy

Studijní cíle: Po zvládnutí kapitoly bude studující schopen charakterizovat databázový systém a jeho roli v informačních systémech, popsat historii vývoje databázové technologie a výhody databázového přístupu a architekturu ANSI/ SPARC, vysvětlit základní pojmy, typy, modely a databázové jazyky DBS.

Klíčová slova: Data, informace, databáze, datový model, SŘBD, databázové jazyky, architektura.

Potřebný čas: 4 hodiny.

Při studiu databázových systémů narazíme často na problematiku, kterou systematicky zpracovávají jiné předměty informatiky, ale většinou se jedná o specifické rozšíření a integrující pohled. Jsou to například oblasti a předměty jako teoretická informatika a algoritmizace, operační a distribuované systémy, programovací jazyky, projektování a vývoj softwaru, matematické disciplíny – algebra, logika, práce s množinami. V textu se předpokládá alespoň orientační přehled mimo jiné o organizaci dat, metodách přístupu k datům, datové a funkční analýze, HW (disky, paměti) počítače a jeho řízení atd..

Průvodce studiem

Studium databázových systémů (DBS) můžeme členit do tří širokých oblastí podle úhlu uživatelského nebo vývojářského pohledu a hloubce a rozsahu řešených problémů.
***1.Návrh databáze** – Řeší problém, jak navrhnout strukturu informací, jaký model pro popis vztahů, typů, jaké hodnoty ukládat.*
***2.Programování databáze** - Řeší problém, jak konkrétně, jakým programovacím jazykem, manipulovat s daty a získávat informace, pracovat s transakcemi, atd. v aplikacích i v návaznosti na konvenční programovací jazyky.*
***3.Implementace databáze** - Řeší problém, jak navrhnout program, který efektivně řeší všechny důležité databázové procesy a operace, jako je fyzická struktura uložení dat, rychlý přístup k datům, efektivní zpracování dotazu, transakcí, atd..*

Abychom poznali specifiku databázové technologie, seznámíme se s vybranými pojmy.

1.1 Úvod do databázové technologie

1.1.1 Pojem databáze

Informací se data a vztahy mezi nimi stávají vhodnou interpretací pro uživatele vytvořením struktur, které odhalují uspořádání, vzory, tendence a trendy.

Moderní pojem informační technologie představuje unifikovaný soubor metod a nástrojů pro vývoj software informačních systémů, které zpracovávají data (realizují sběr, uložení a uchování, zpracování, vyhledávání) a většinou ve své architektuře používají databázovou technologii. Od historicky prvních síťových a hierarchických databázových systémů se přešlo na relační systémy. Na relačních komerčních systémech v průběhu několika desetiletí vývoje a využití došlo k odstranění hlavních problémů, většina postupů byla správně definována i úspěšně a efektivně implementována. Tím se tato klasická technologie stala nesmírně robustní a

pro jistou třídu aplikací i do budoucna perspektivní. Další vývoj pokračuje paralelně ve formě čistě objektových systémů, které se prosazují hlavně ve speciálních aplikacích (CAD, grafické systémy) a největší perspektiva se dává evolučnímu pokračování relačních systémů - relačně-objektovým systémům. I klasická relační technologie se dále rozvíjí – příkladem jsou paralelní architektury pro zvýšení výkonu SŘBD, deduktivní databáze a expertní systémy.

S rozvojem technologie krystalizují za klasickými IS s databázovými aplikacemi (katalogy, bankovnictví, knihovny, sklady, doprava, …) další aplikace se složitě strukturovanými daty :

- Multimediální databáze – informace ve formě (i kombinované) dokumentů - textů, obrázků, zvuků, videa
- Geografické informační systémy – data ve formě map
- Podnikové systémy pro podporu analýzy, řízení a rozhodování, využívající technologii datových skladů a OLAP s možností dolování dat (data minig)
- Komerční obchodování na internetu , práce s XML daty
- Řízení podnikových procesů – workflow

Rozvoj databázové technologie je reakcí na potřebu efektivně, za pomoci počítačů, zpracovávat různé rozsáhlé agendy, se zaměřením na vyhledávání a aktualizaci dat. Vyhledávání představuje nalezení takových informací - záznamů, které vyhovují podmínkám na požadovaná data. Podmínky jsou formulovány ve formě dotazu ve vhodném dotazovacím jazyce a odpovědí je typicky podmnožina z uložených záznamů (případně ještě zpracovaných – výpočtem z uložených dat získáme odvozené informace). Výsledné údaje můžeme třídit dle různých kritérií, prezentovat ve formě tištěných výstupních sestav. Aktualizace zajišťuje změnu hodnot vlastností objektů nebo zrušení či přidání nového objektu (záznamu) tak, aby informace korespondovaly s realitou. Aby popsané operace byly efektivní, musí být data vhodně uspořádaná a organizovaná na vhodném médiu.

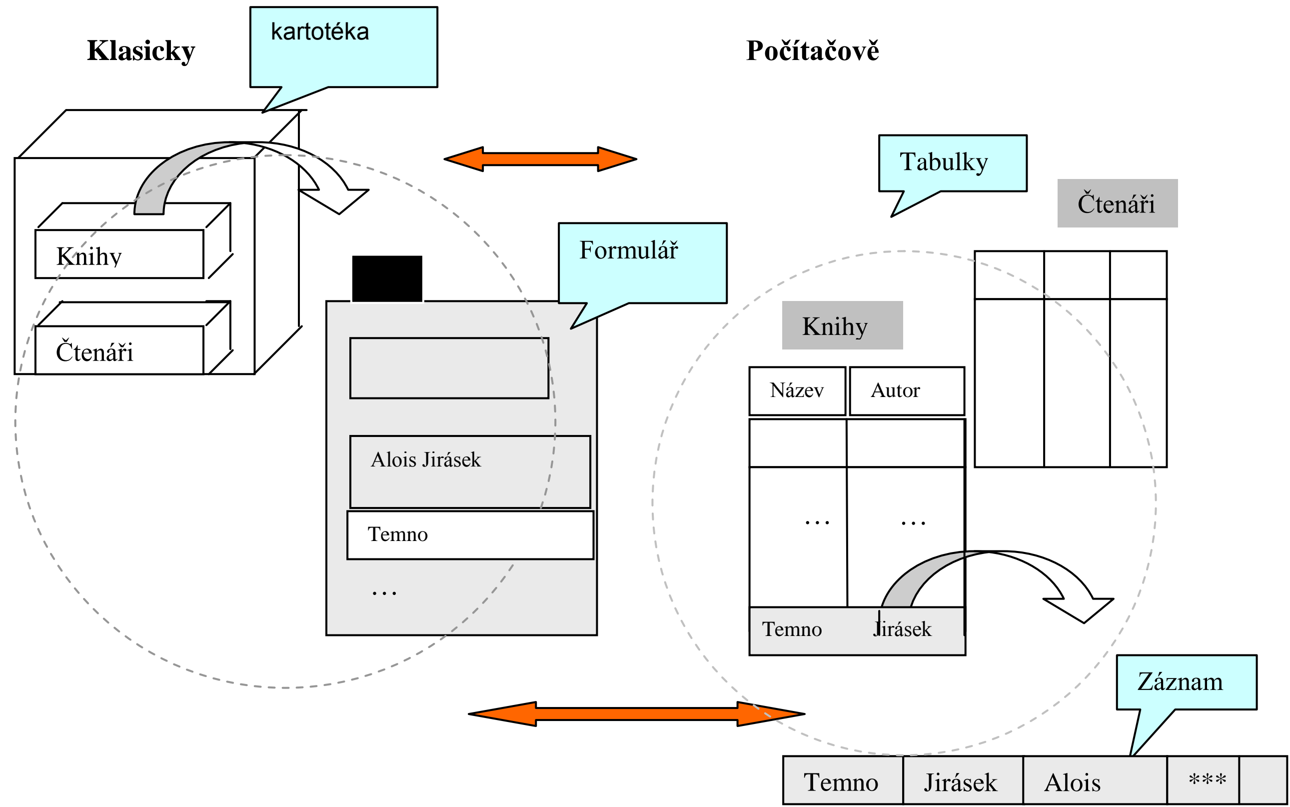
Databáze – registr v elektronické podobě

Existuje mnoho způsobů, jak definovat pojem databáze, například: Databáze je úložiště informací, udržované v čase, v počítačově zpracovatelné formě.

Průvodce studiem

Databáze – sdílená kolekce logicky souvisejících dat i s popisem své datové struktury, organizovaná pro optimální manipulaci s perzistentními daty a získávání informací pro potřeby informačního systému.

Pro základní představu porovnejme zjednodušené analogie klasické a elektronické verze na příkladu kartotéky části knihovny (je použit relační model dat, který data uchovává ve formě tabulek):



Obr. 1 Porovnání klasické a elektronické technologie

Výše uvedený příklad je typický svou „plochou“ datovou strukturou, přirozeně transformující data aplikace do dvourozměrných tabulek. Takto pojatá data – tabulky- jsou častou základní logickou datovou strukturou počítačem podporovaných informačních technologií. Programové vybavení, zajišťující perzistentní uložení a bezchybnou údržbu dat na médiích, programové rozhraní pro bezpečný přístup více uživatelů nebo aplikací k manipulacím s daty, získávání informací z kolekcí dat vhodným dotazovacím jazykem, správu transakcí a další funkce, se nazývá systém řízení báze dat – SŘBD. Vše podstatné se v databázové technologii točí kolem dat.

Data v databázi si můžeme představit jako známá fakta, která nás zajímají, s poměrně pevnou strukturou, uložená trvale v počítači. Mezi nejdůležitější charakteristiky dat v databázích patří

Databázová technologie se zabývá řízením velkého množství perzistentních, spolehlivých a sdílených dat

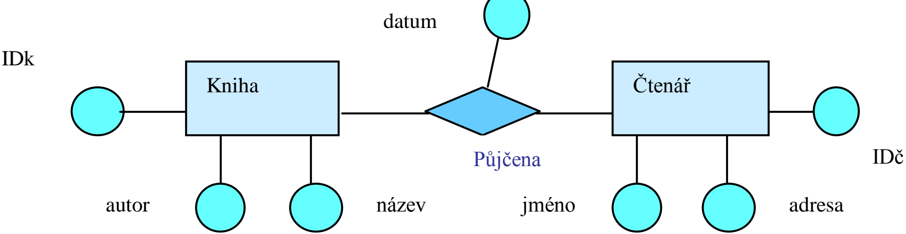
- Perzistence – data přetrvávají dlouhodobě od jedné operace ke druhé, nezávisle na použitých programech
- Velké množství – operace typicky nevystačí s vnitřní pamětí, proto použití sofistikovaných algoritmů při manipulaci s daty
- Správnost, nerozpornost – snaha odhalením nejrůznějších chyb v datech při vkládání nebo úpravě databáze zachovat korespondenci s realitou, vztaženou ke konkrétnímu času, ne nutně k nejaktuálnějšímu (realizováno pomocí integritních omezení)
- Spolehlivost – data je možné po poruše počítače zrekonstruovat
- Sdílení – s daty pracuje typicky více uživatelů
- Bezpečnost – možnost omezit přístup k datům a operacím s nimi
- Integrace – spojení několika požadovaných pohledů do komplexní datové struktury
- Konzistence – identická data mohou být dočasně nebo trvale uložena na více místech, ale musí mít stejnou hodnotu

S efektivitou – rychlostí operací – je spojena organizace dat. Klasicky je základem zpracování na fyzické úrovni soubor, každý objekt reality je popsán záznamem souboru, vlastnost objektu je položkou záznamu, která je uložená ve formátu vybraného předdefinovaného typu. Množinu datových souborů, uchovávajících data o nějakém vymezeném úseku reality, nazýváme *databází*. *Instance databáze* je kolekce informací uložených v databázi, nerozporná v konkrétním čase, definuje stav.

Schéma konkrétní *databáze* – informace o metadatech (data o datech, uložena odděleně v katalogu dat), popisuje strukturu dat v databázi - je definováno prostředky použitého datového modelu, se kterým pracuje SŘBD.

Datový model je soubor prostředků a konceptů, popisujících data (sémantiku, strukturu, vztahy, integritní omezení) na určité úrovni abstrakce. Obvykle rozlišujeme tři komponenty – strukturální, operační a specifikace integritních omezení. Konkrétní datový model souvisí s úrovní abstrakce, s pohledem na data v procesu vývoje aplikace – od vymezení požadované části reality ze zadání až po fyzické uložení v počítači. V průběhu vývoje IS se prakticky může použít mnoho různých modelů v závislosti na metodologiích, informačních technologiích, architektuře IS, atd., typická je potřeba přechodů z jednoho typu modelů do druhého v průběhu vývoje softwaru IS s použitím vhodných transformačních pravidel. Možné rozdělení databázových modelů :

- *konceptuální* (data na úrovni pohledů a konceptů) - založené na objektech (ER model, sémantický model, OO model, funkcionální datový model), na vysoké úrovni abstrakce, bez bližší specifikace budoucí implementace. Je výsledkem datové analýzy, prostředníkem mezi zadavatelem a analytikem v procesu formulace a zpřesňování zadání. Spolu s funkčními závislostmi mezi atributy rozhodujícím způsobem určují interpretaci dat v databázi – jejich sémantiku



Obr. 2 Příklad ER diagramu

- *logické* - založené na záznamech (znalosti seznamů vlastností - atributů <a1, a2, …, an >), které tvoří logický celek (n-tici) jako obraz vlastností abstraktního objektu, prostředky a formu určuje typ datového modelu použitého SŘBD. Mezi historicky první řadíme modely

  hierarchický, ***síťový*** – vztah mezi záznamy je v implementaci definován pomocí ukazatelů a data tvoří skupiny záznamů s topologií příslušných grafů – stromu, nebo obecnější sítě.

  Stále nejrozšířenější je následující model

  relační – Záznamy stejného entitního typu jsou logicky organizovány ve formě dvoudimenzionálních tabulek, vztah mezi záznamy je definován hodnotami vazebních atributů (cizích klíčů), obecně v samostatných tabulkách. Relační databázi tvoří jedna, nebo několik tabulek. Tabulka uchovává informace o skupině podobných objektů reálného světa, např. o knihách. Informace o jednom objektu je na jednom řádku tabulky. Pořadí řádků v tabulce není důležité, nenese žádnou informaci. Sloupec tabulky uchovává informace o jedné nestrukturované vlastnosti objektu.

Př. Definice relací – tabulek (představuje databázové relační schéma, vzniklé transformací z předchozího ER diagramu)

Kniha (IDk : int, autor : char(20), název : char(20))

Půjčena (IDk : int, IDč : int, datum : date)

Čtenář (IDč : int, jméno : char(20), adresa_ulice : char(20), adresa_číslo_pop : char(20))

Kniha

IDk	autor	název
65	Němcová	Babička
3	Jirásek	Temno

Půjčena

IDk	IDč	datum
3	103	1.3.1999
…	…	….

Čtenář

IDč	jméno	adresa_ulice	adresa_číslo_pop
6	Krátká	Okružní	3
103	Novák	Zelená	26

Základní operace v databázi, manipulující s daty, jsou například:

- vložení informací o nové knize (INSERT INTO Kniha VALUES (6, ‘Čapek’, ‘Matka’)
- odstranění informací o vyřazené knize (DELETE FROM Kniha WHERE IDk = 5)
- oprava, aktualizace údaje existující položky (UPDATE Kniha SET stav = ‘zapůjčen’ WHERE IDk = 63)
- dotaz na výběr knih s jistou vlastností – např. rok vydání (SELECT * FROM Kniha WHERE vydání = 1992)

    - **fyzické** (data na fyzické úrovni, struktura uložení v paměti – na disku, pomocné podpůrné vyhledávací struktury – indexy, …)

Průvodce studiem

Databázové systémy můžeme rozdělit na klasické, souborově orientované s navigací pomocí ukazatelů – hierarchické a síťové, pracující s tabulkami – relační a na nové směry a přístupy v databázové technologii, což mohou reprezentovat rozšířené relační systémy (relačně- objektové), čistě objektově orientované, XML databáze, deduktivní databáze (Datalog) a distribuované databáze

Databázový systém (DBS) zahrnuje:

- technické prostředky – spolu s dalšími faktory a požadavky uživatele limitují možnou složitost architektury IS nebo častěji je HW návrhem určen. Komerční DBS pokrývají širokou škálu možností s různým stupněm úplnosti a efektivity splnění požadavků, kladených na SŘBD, výkonem, cenou, charakterem aplikace, atd.. Setkáváme se s jednoduššími souborovými systémy (např. dBASE, FoxPro, Microsoft Access) na jedné straně až po komplexní (a nákladné) systémy (DB2, Oracle, Microsoft SQL server)
- programové vybavení (SŘBD, vývojové nástroje)
- data uložená v databázi (DB)
- uživatele – ty můžeme klasifikovat podle různých kritérií (oprávnění k operacím, znalost a úroveň řízení DBS i aplikace, …) do typových skupin např.
    1. administrátor, správce dat - koordinuje všechny aktivity v databázovém systému, zakládá, modifikuje uživatele, rozhoduje o tom, která data a jak budou v bázi uložena – definuje schéma databáze a integritní omezení, určuje

schéma uložení dat a metody přístupu k datům, pokud je to nutné, realizuje požadované změny, modifikuje struktury dat, přiděluje přístupová práva k datům i operacím, sleduje výkon a chování DB serveru, zálohuje, rekonstruuje databáze v případě jejího poškození, …

2. aplikační programátor (tvůrce aplikací) - programuje aplikační programy nad definovanými datovými strukturami, složitější dotazy a transakce použitím DML v hostitelském jazyku nebo jazyky 4. generace.
3. příležitostný uživatel - umí prostřednictvím dotazovacího jazyka formulovat vlastní specifický dotaz nebo jinak manipuluje s daty
4. naivní uživatel - (obvykle neprogramátor), který prostřednictvím aplikačních programů pracuje s databází a používá databázi jako informační systém pro ukládání, zpracování a vyhledávání informací

Pro úplnost - jedno z dalších kritérií dělení DBS je na jedno-uživatelské (hlavně dříve na PC) a více-uživatelské.

Schematicky zkracujeme definici jako spojení SŘBD a dat uložených v databázi

databázový systém = systém řízení bází dat + databáze

DBS = SŘBD + DB

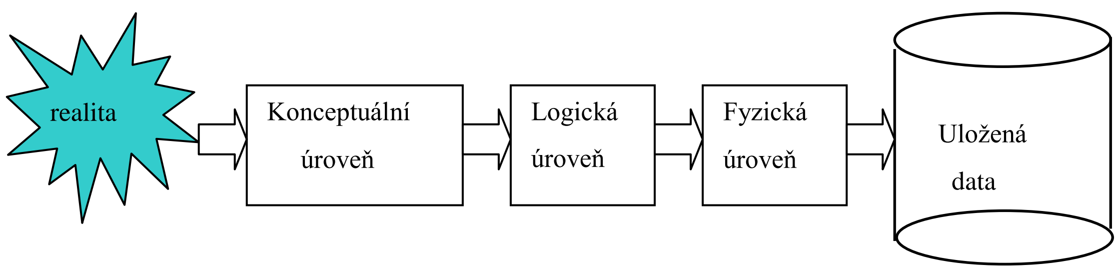
Základní paradigma – existence dat v databázi je nezávislá na aplikačních programech. To umožňuje na aplikaci nezávislý popis dat v datovém slovníku (např. v systémových tabulkách relačních systémů)

1.1.2 Úrovně abstrakce

Při vývoji softwaru IS modelujeme datové struktury způsobem, který nejlépe vyhovuje příslušné fázi životního cyklu programu a míře abstraktnosti. Nejvyšší úroveň abstrakce tvoří reálný svět, ze kterého vyčleňujeme takové typy objektů a údaje o objektech, které souvisí s fakty, jež chceme zahrnout do informačního systému. Tuto úroveň definuje zadavatel na základě integrace dílčích uživatelských pohledů a upřesňuje se v procesu analýzy iterací s vývojem funkční analýzy IS. Pohledy jednotlivých uživatelů tvoří **externí schémata** Výsledkem datové analýzy a použitím konceptuálního modelu vznikne informační struktura zvaná **konceptuální schéma** databáze, jež je nezávislé na pozdějším logickém databázovém schématu. **Databázové schéma** je realizací, výsledkem transformace konceptuálního schématu prostřednictvím konstruktů příslušného datového modelu a představuje logickou – databázovou úroveň abstrakce. Popisuje definice datových struktur a jejich vazeb pomocí prostředků použitého SŘBD. Externí schémata se na databázové úrovni mohou realizovat ve formě virtuálních pohledů - většinou majících podporu SŘBD v jazycích definujících data (např. databázový objekt *view* v SQL relačních systémů) ve formě datových struktur, optimálně navržených pro typovou skupinu uživatelů, využívajících IS podobným způsobem.

Abstraktnější pohled na data v databázi, který poskytuje SŘDB, umožňuje skrýt detaily uložení a správy dat.

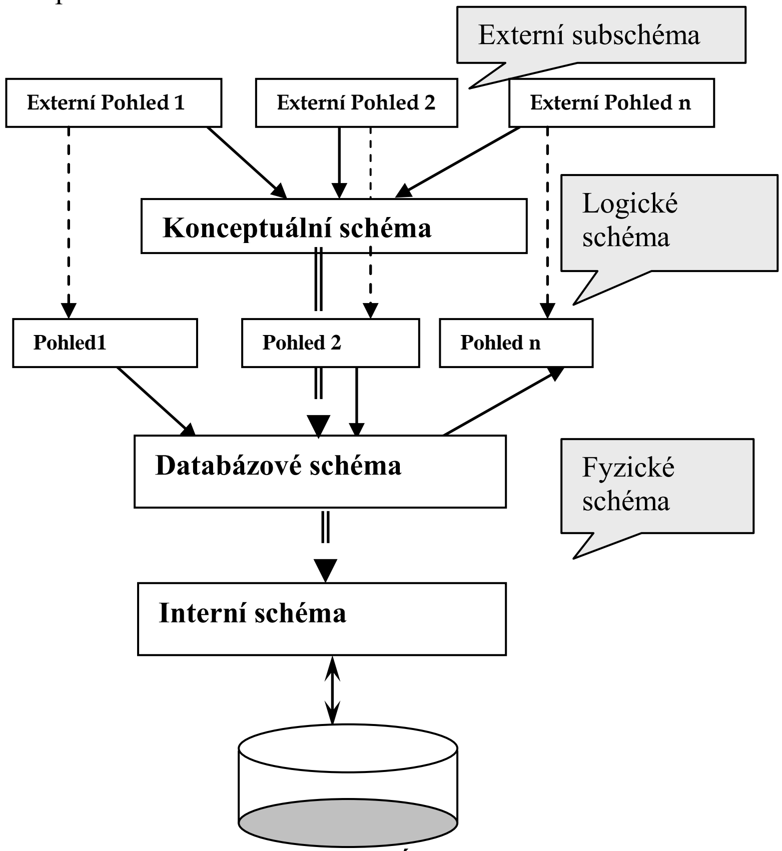
Interní schéma databáze popisuje nejnižší fyzickou úroveň abstrakce - uložení dat na médiu počítače. Definuje fyzické záznamy, fyzickou reprezentaci jejich položek, sdružování záznamů do souborů, charakteristiky těchto souborů. Souvisí bezprostředně s použitým SŘBD a možností explicitně určovat fyzické uložení nastavením jeho parametrů nebo použitím příslušných příkazů. Schematicky :



Průvodce studiem

ANSI/SPARC architektura definuje tři úrovně:
1.externí úroveň nabízí uživatelský pohled do databáze a popisuje části reality z pohledu každého uživatele zvlášť(mnoho pohledů).
2.konceptuální úroveň integruje data do společného pohledu, určuje jaká data jsou uložena v databázi a jaké jsou mezi nimi vztahy (jedno konceptuální schéma)
3.interní úroveň popisuje fyzickou reprezentaci dat v počítači (jedno fyzické schéma)

- Původně vychází z ANSI/SPARC architektury (1976), definuje tři úrovně pohledu na data



Objekty reálného světa

Pohledy popisují data tak, jak je vidí jednotlivé typy uživatelů, aplikace odstiňují detaily datových typů – ***jaké objekty***

Logická struktura(tabulky, pohledy) popisuje všechna data a vztahy mezi nimi v databázi – ***jaká data***

Soubory

Tabulkové prostory

Indexy

– ***jak jsou data uložena*** *na médiu*

Obr. 3 Úrovně abstrakce

1.1.3 Historický vývoj zpracování dat

Ilustrativní z pohledu funkce a vývoje programové vrstvy SŘBD je historický přehled používaných metod, který také úzce souvisí se stupněm rozvoje hardwaru a architektury výpočetních systémů.

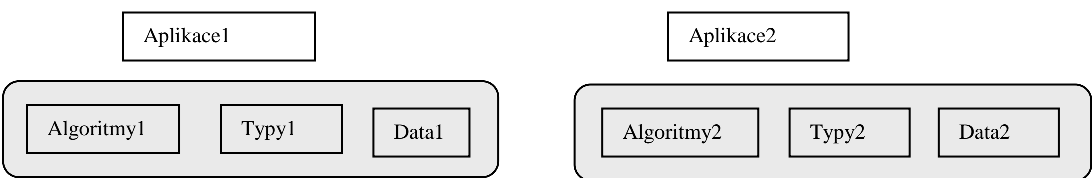
Průvodce studiem

Souborový systém – kolekce aplikačních programů, které zajišťují služby pro uživatele. Každý program definuje a udržuje svá vlastní data.

50. léta :

Počátečním etapám programového řešení úloh tohoto typu se říká agendové zpracování dat. Vybrané charakteristiky tohoto přístupu:

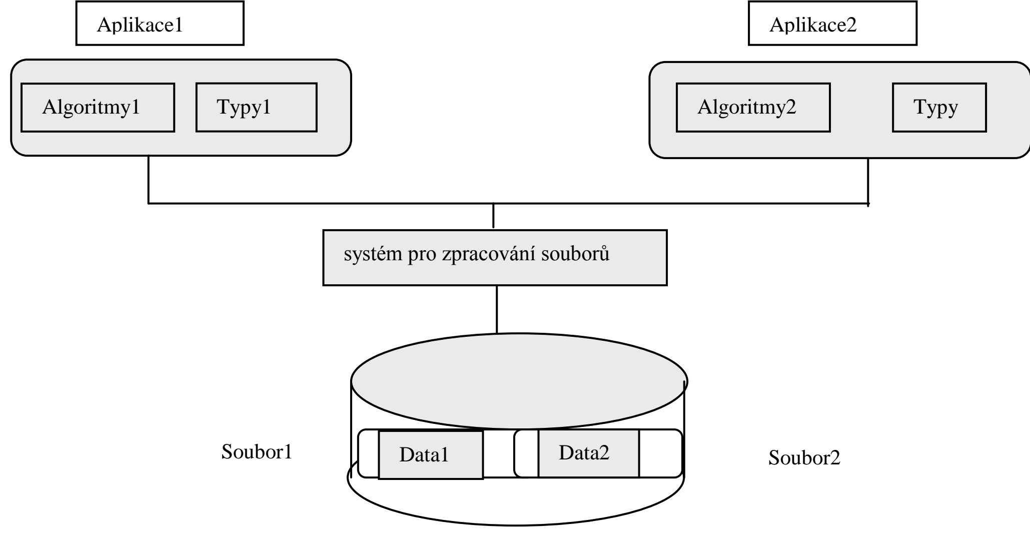
- aplikační programy řeší jednotlivé úlohy - uložení dat na médium, zpracování dat, tisk sestav, …
- soubor programů tvoří ucelenou agendu
- plná závislost dat a programů (každý program řeší nejen vlastní aplikační problém, ale i otázky fyzického uložení dat na médiu; navazující úlohy musí respektovat již vytvořené fyzické struktury dat)
- nízká efektivnost datových struktur i programů
- zpracování v dávkách : - data se ručně zapisují do formulářů
  - z formulářů se zaznamenávají na vstupní médium pro počítač
  - formou primárního zpracování se data načtou do počítače
  - řadou sekundárních zpracování se pak nad daty provádějí výpočty, výběry, tisky sestav …
- řešení ucelených problémových oblastí v jedné agendě (data se sbírají speciálně pro tuto agendu)
- mezi různými agendami nejsou žádné nebo jen minimální vazby
- typická architektura aplikace (vše v programu)
- Použití specializovaných jazyků (Pl /1, COBOL)



Obr. 4 Struktura dat agendového typu

První polovina 60. let

Systémy pracující s interními organizacemi dat s přímým přístupem a interaktivním stykem s uživatelem, vytvoření **systémů pro zpracování souborů,** závazná doporučení CODASYL



Obr. 5 Struktura dat systémů pro zpracování souborů

Nevýhody dosavadních řešení:

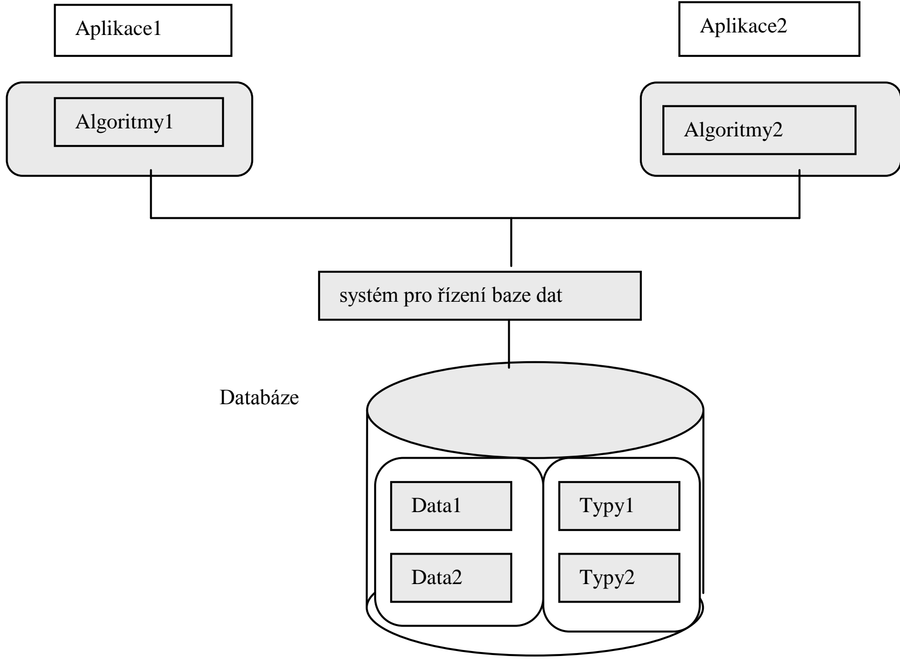
- Soubory navrženy podle potřeb konkrétních programů – malá flexibilita, nízká úroveň abstrakce při pohledu na data – jednoduché datové modely
- Velká redundance dat (aplikační programy vytvářené různými programátory způsobí opakování informací ve více souborech)
- Nekonzistence dat (při změnách hodnot se oprava položky neprovede na všech místech, kde je opakující se informace zapsána)
- Problémy se zabezpečením integrity dat (uložená data musí být většinou aktuální, vyjadřovat skutečnost z reálného světa, vztaženou k jistému časovému okamžiku, implementace integritních omezení)
- Špatná dosažitelnost dat – izolovanost dat (data rozptýlena v různých agendách , různé formáty ekvivalentních dat, problémy s neplánovanými dotazy …)
- Problémy se specifikací a zajištěním ochrany dat (proti získání informací z vnějšku systému, ale i mezi uživateli), souběžného přístupu více uživatelů

Reakcí ve vývoji je základní princip - tendence oddělení dat a jejich definic od aplikačních programů.

Druhá polovina 60. let a 70. léta

vytvoření prvních systémů pro řízení baze dat – SŘBD, vývojem ze souborových systémů. DBS podporovaly různé datové modely - síťový a hierarchický, později relační model dat (Edgar Codd)

- data centralizovaná
- jednotný přístup k informacím, rozvoj speciálních jazyků a rozhraní
- popis dat oddělen od aplikačních programů



Obr. 6 Struktura dat SŘBD

1.1.4 Systém Řízení Báze Dat – SŘBD

SŘBD je kolekce programů, které tvoří rozhraní mezi aplikačními programy a uloženými daty.

Základní funkce:

- na základě použitého datového modelu:

- umožňuje vytvořit novou databázi a definuje její schéma a data (popis souborů, záznamů, položek, typů, velikostí, vztahů mezi záznamy, indexů),

- provádí validaci dat (kontrola typu, rozsahu, konzistence, nerozpornosti),

- případně modifikuje schéma, strukturu dat (vytváří, modifikuje slovník dat)

- určuje strukturu uložení (i pro velké množství) dat
- manipuluje s daty, hlavně umožňuje dotazování, volí metody přístupu (optimalizace), zajišťuje výkonnost
- zajišťuje autorizaci a bezpečnost
- souběžný přístup
- zajišťuje zotavení po poruše
- kontroluje integritu dat
- zajišťuje správu transakcí

Průvodce studiem

SŘBD – softwarový systém, umožňující definovat, vytvořit, udržovat a řídit přístup do databáze.

Hlavní výhody a požadavky na SŘBD

1. Vyšší datová abstrakce – manipulace s formalizovanými strukturami na vyšší, logické úrovni abstrakce
2. Nezávislost dat – schopnost modifikovat definici schématu bez vlivu na schéma vyšší úrovně abstrakce. Analogie s abstraktními datovými typy, detaily implementace jsou skryty.

Průvodce studiem

Nezávislost dat:
Fyzická nezávislost dat - změna fyzického schématu neovlivní aplikační programy (sem patří rovněž např. přidání a zrušení indexů, změna klastrů)
Logická nezávislost dat - změna logického schématu neovlivní aplikační programy (sem patří rovněž např. přidání, modifikace a zrušení entitních typů nebo vazeb)

3. Centralizovaná administrace dat a popis struktury.
4. Možnost formulovat ad hoc dotazy mimo aplikační programy.

Většina SŘBD má vlastní speciální *databázové jazyky*, které zajišťují funkčnost prostřednictvím příkazů a předdefinovaných standardních funkcí. V relačních systémech je prakticky standardem jazyk SQL. Možné dělení jazyků (případně příkazů komplexního jazyka) do kategorií :

1. jazyk pro definici dat (**JDD**) - definice, modifikace a rušení entitního typu, vazby mezi entitními typy a atributů, s použitím logických jmen a datových typů nebo domén (předdefinovaných nebo uživatelských), definuje některá systémová integritní omezení
2. jazyky pro manipulaci dat (**JMD**) - manipulace s atributy, entitami a jejich vazbami, množinami entit, realizují operace typu INSERT, UPDATE, DELETE na logické úrovni. Výběr dat z databáze (operace SELECT) zajišťuje dotazovací jazyk, který je buď součástí JMD, nebo funguje samostatně. Je většinou neprocedurální (formulujeme jen požadavky dotazu, ne postup, algoritmus získání informací)
3. jazyk pro řízení přístupu k datům – prostředky pro realizaci změn schématu a dat databáze se zaměřením na definici oprávnění operací pro každého uživatele
4. jazyk pro řízení transakcí
5. jazyky pro zápis algoritmu
    - v hostitelském jazyce (Cobol, C, Java, ...), pak jsou výše uvedené JDD a JDM vytvořeny jako procedury v hostitelském jazyce a celý SŘBD tvoří nadstavbu tohoto jazyka;
    - vlastní speciální jazyk SŘBD, obsahující příkazy JDD a JMD a navíc programové struktury pro provádění algoritmů - příkazy pro větvení a cykly, někdy podporující i vývoj prezentační vrstvy aplikace.
6. jazyky čtvrté generace (4GLs) – se stávají pravidelnou součástí hlavně aplikačních vývojových prostředků. Patří sem generátory celých aplikací, formulářů, dotazů, výstupních sestav, grafických výstupů, ale i datových tabulek.

Původně byly SŘBD velké a drahé programové systémy na rozsáhlých a vybavených počítačích. Vývoj sekundárních pamětí a jejich cena dovoluje nasazení vhodných SŘBD na všechny třídy počítačů – od nejmenších systémů, většinou nad jednoduchými datovými soubory, až po nejvýkonnější paralelní architektury, zpracovávající TB informací.

Průvodce studiem

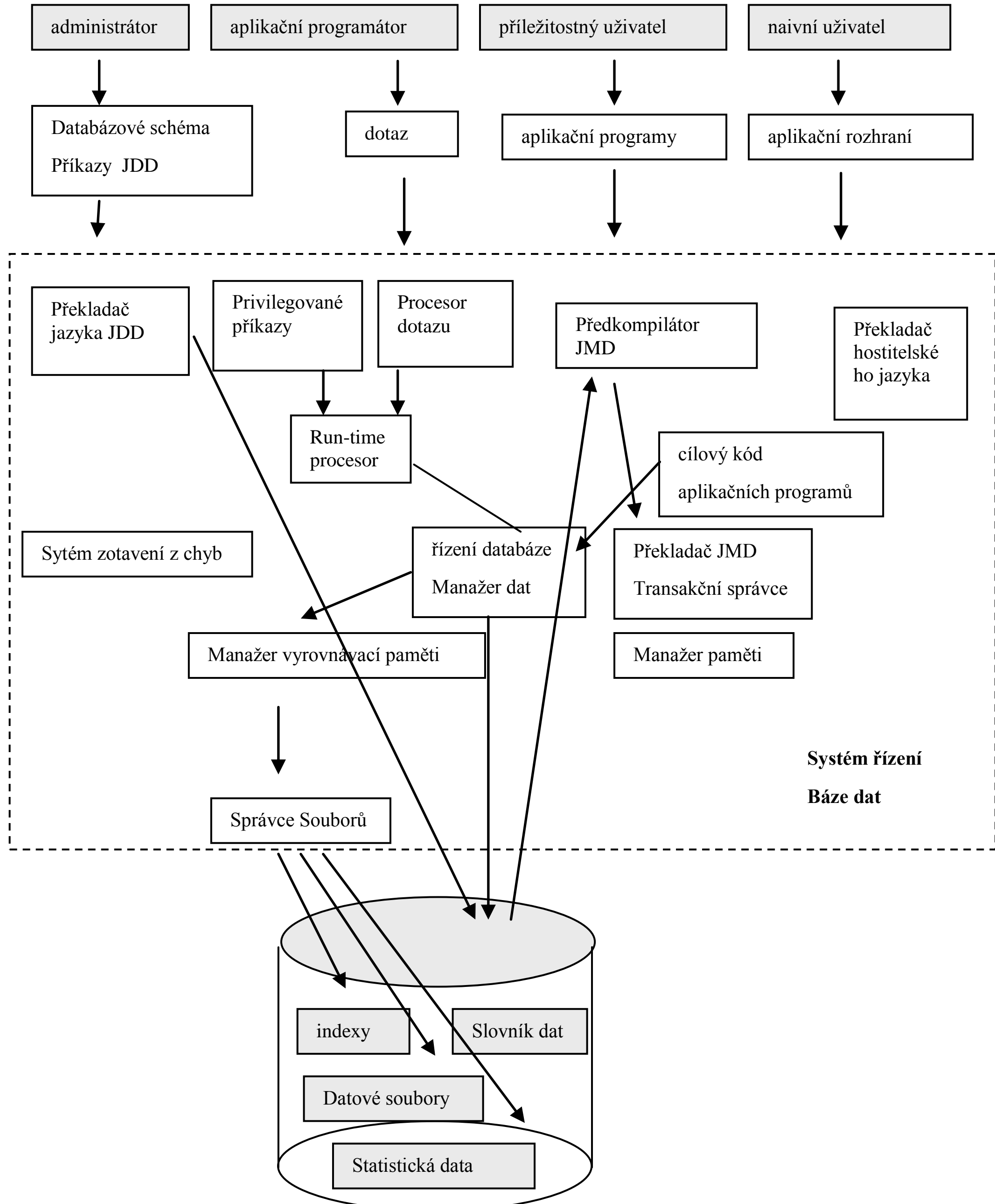
Proč použít databáze?
- pro datovou nezávislost a efektivní přístup
- urychlují a standardizují návrh aplikací
- integrují data a zajišťují bezpečnost
- zajišťují snadnou administraci a minimální redundanci
- umožňují souběžný přístup a zotavení po poruše

1.1.5 Architektura DBS

Pojem architektura DBS, případně IS zahrnuje mnoho možných úhlů pohledu v závislosti na kontextu a etapě návrhu. Tradiční je některá varianta centralizovaného modulárního funkčního schématu relačního databázového stroje, ve které můžeme rozlišit vrstvy - pro optimalizaci a

provedení dotazu, relační operace, metody přístupu k souborům, správce vyrovnávací paměti a správce disku.

Například:



Obr. 7 Funkční architektura DBS

Překladač JDD zpracovává definici a změny schématu databáze a ukládá je do katalogu dat. Run-time procesor pracuje s databází při běhu programu. Manažer dat spolupracuje s operačním systémem případně podsystémy vnitřní a vyrovnávací paměti a řízení disků na přenosu dat. Procesor dotazu interpretuje nebo překládá dotaz do optimalizované podoby a předá ho na vyhodnocení. Nejnižší úroveň tvoří subsystém pro ovládání souborů. Zahrnuje fyzickou organizaci datových souborů, vlastní uložení dat na vnějším médiu a realizaci přenosů dat s pamětí prostřednictvím příslušných manažerů. Na disku jsou uloženy informace čtyř kategorií

1. Data – obsah vlastní databáze
2. metadata ve slovníku dat – popis schématu databáze a integritních omezení

3. statistiky – informace o vlastnostech uložených dat, jako je velikost, charakter hodnot, vzájemné vazby
4. indexy – podpora efektivního přístupu k datům

Mezi nejdůležitější patří architektura z hlediska topologie rozdělení základních typů služeb ve vrstvách programového vybavení IS. Služby můžeme rozdělit na :

1. prezentační – vstup / výstupní zařízení zobrazuje informace(určuje co a jak uživatel vidí), reakce myši, klávesnice, …
2. prezentační logika – interakce uživatele s aplikací (reprezentuje hierarchii formulářů a menu, logiku jejich vztahů)
3. logika aplikace – realizuje aplikační operace a funkce(výpočty, rozhodování) , „prostředky (jazykem) aplikace“
4. logika dat – podpora logiky aplikace operacemi, které mají být prováděny s databází, vyjádřená jazykem SŘBD(SQL – SELECT, INSERT, UPDATE, DELETE)
5. datové služby – operace s databází vně logiky dat, např. definice dat, transakce
6. zpracování souborů – operace na fyzické úrovni, získání dat z disku, práce s vyrovnávací pamětí, … (většinou poskytuje operační systém)

Průvodce studiem

Typický SŘBD se skládá s jednotlivých vrstev – např. 1. Optimalizace a provádění dotazů, 2. Provádění relačních operátorů, 3. Správa souborů a přístupové metody, 4. Správa vyrovnávacích pamětí, 5. Správa disku. Vrstvy 2-5 musí umožnit paralelizmus řízení a zotavení po poruše.

Typické architektury mají historický kontext s návazností na stupeň vývoje HW i SW včetně operačních a síťových systémů. Některé vybrané příklady:

- *Centrální architektura:*V/V (neinteligentní) terminál (služby **1**) – sálový počítač (služby **2-6**): sdílení systémových prostředků, ale primitivní textové rozhraní, problematická rozšiřitelnost o další klienty, zátěž sítě o prezentační data
- *File-server, databáze jako soubory*: stanice, např. PC (služby **1-5**) - file-server (služba **6**) : umožňují rozšiřitelnost o nové klienty, ale velké zatížení sítě, neefektivní souběžný přístup, citlivé na změnu logiky dat.

Klient – server má několik variant, je nejpoužívanější, znamená dekompozici funkcionality a jistou distribuci dat a databázového softwaru mezi databázovým serverem a jeho klientem. To umožňuje aplikacím škálování zdrojů – horizontální (aplikaci lze zpřístupnit více DB serverů) nebo vertikální (nasazení levnějšího méně výkonného počítače pro klienta a výkonného pro server). Komunikace mezi klientem a serverem probíhá pomocí příkazů SQL s odpovědí typicky ve formě relačních dat. Centralizace architektury podporuje ochranu dat před ztrátou, nebo zneužitím.

- Klient – server : stanice (služby **1-4**)- DB server (služby **5-6**) – mnoho variant architektury, ve které požadavek jednoho procesu (klienta) je poslán k provedení na druhý proces (server), problémy s efektivitou při souběžném přístupu více uživatelů : typicky heterogenní prostředí

- Klient – server se třemi vrstvami : stanice (služby **1-2**) - aplikační server (služba **3**)- DB-server (služby **4-6**)
- Klient – server s více vrstvami : tenký web klient (služby **1**) – web-server (služba **2**) - aplikační server (služba **3**)- DB-server (služby **4-6**)

Dále existuje mnoho možností, jak služby rozdělit nebo netypicky přesunout (např. služba **3** implementována v uložených procedurách na DB serveru nebo na web serveru). Samostatnou kapitolou jsou distribuované DBS, které umožňují fyzické rozdělení nebo replikaci dat na více uzlech sítě.

Shrnutí Databázové systémy tvoří většinou základ pro datovou úroveň v architektuře informačních systémů. Předchůdci moderních DBS byly souborové systémy s aplikačními programy se službami zajišťujícími například tvorbu výstupních sestav, s izolovanými vlastními daty v každém aplikačním programu. Problémy tohoto řešení – např. velká redundance a izolovanost dat, závislost na fyzické struktuře uložení, nedostatečná podpora paralelního transakčního zpracování, nemožnost jednoduše modifikovat menší části dat, atd. vedly postupně k vývoji systému řízení báze dat. SŘBD je programová vrstva, která podporuje efektivní správu velkých perzistentních dat, umožňuje získání informací z databáze prostřednictvím dotazovacích jazyků, s možností pracovat souběžně v transakcích s maximální vzájemnou podporou nezávislosti, izolovanosti a konzistence. Uživatel komunikuje se SŘBD prostřednictvím databázových jazyků, ve kterých jsou jednotlivé příkazy podle účelu děleny na části jazyka definující data (JDD) s možností vytvořit a modifikovat databázové objekty a jazyka manipulující s daty (JMD), který umožňuje provádění operací insert, update, delete a select. Do kontextu DBS můžeme zahrnout hardware – počítač na kterém DBS pracuje, software – SŘBD, operační systém, aplikační programy, dále data a procedury a nakonec různé skupiny uživatelů – administrátor databáze, aplikační programátor, běžný uživatel a podobně. Mezi nejdůležitější moduly SŘBD patří manažer paměti a transakcí a modul zpracování a optimalizace dotazu. Historicky první generaci databázových, souborově orientovaných systémů reprezentují hierarchické a síťové – podle CODASYL modely dat. Nejrozšířenější datový model databázových systémů je relační model, který informace organizuje ve formě tabulek a pro programování nejčastěji používá jazyk SQL, který umožňuje definovat a modifikovat datovou strukturu databáze, manipulovat s daty a administrovat databázový systém s možností vytvářet i modifikovat všechny databázové objekty a určovat oprávnění k operacím. Perzistence dat je zajištěna uložením databáze na disku. Nejčastější architektura databázových systémů je typu klient – server, s podporou uživatelského rozhraní pro přístup do databáze na obou stranách, na klientovi i na serveru.

Pojmy k zapamatování

- Databázový systém, vlastnosti databázových dat, historický přístup
- Systém řízení báze dat, transakce
- Datový model, úrovně abstrakce
- Databázové jazyky, SQL
- Architektura systému

Kontrolní otázky

1. *Jaké specifické vlastnosti mají data v databázích?*
2. *Co je databáze?*
3. *Jaké jsou základní databázové operace?*
4. *Jaký byl historický vývoj hromadného zpracování dat?*

5. *Jaké základní funkce podporuje SŘBD a z jakých logických modulů se skládá?*
6. *K čemu slouží databázové programovací jazyky?*
7. *S jakými datovými modely se setkáme na jednotlivých úrovních abstrakce?*
8. *Jak je možné charakterizovat jednotlivé skupiny uživatelů?*
9. *Jaké typy architektury DBS znáte?*

Úkoly k textu

Kontaktujte uživatele databázových systémů nebo administrátora a pokuste se o charakteristiku jednotlivých komerčních systémů, získání přístupu do databázového systému a seznamte se podle možností s prostředím a ovládáním DBS.

2 Konceptuální modelování

Studijní cíle: Po prostudování kapitoly by studující měl porozumět modelování datové struktury databázové aplikace pomocí ER modelu, popsat základní koncept ER modelu. Měl by definovat a na jednoduchých úlohách použít konstrukty ER modelu při návrhu datové struktury ze zadání, popsat integritní omezení v ER modelu. Měl by vysvětlit pojem slabá entita a ISA hierarchie.

Klíčová slova: ER model, entita, atribut, vztah (relace), identifikační (primární) klíč, kardinalita, povinné členství ve vztahu, ISA hierarchie.

Potřebný čas: 2.hodiny

Rozsáhlejší projekt IS i malá aplikace prochází při vývoji typickými fázemi životního cyklu:

- Analýza (datová a funkční) – odpovídá na otázky : Proč vyvíjíme aplikaci? Kdo ji bude používat? Jaký bude mít přínos pro uživatele? Jaké funkce a potřeby bude splňovat? Kompletní funkčnost systému získáme z analýzy požadavků, z formalizovaného zadání, konzultacemi s uživateli.
- Návrh – je nejdůležitější fáze. Návrh databáze je prováděn modelováním datové struktury použitím ER diagramu tak, aby vyhovoval funkčním požadavkům IS, logické schéma databáze se nakonec transformuje do fyzických struktur databázového systému.
- Vývoj, Implementace – v této fázi se tvoří programy a datové struktury podle návrhu, na návrh a vývoj databáze navazuje vývoj aplikačních programů. Vytváří se prototypy z návrhu, po ověření správnosti následuje implementace aplikace.
- Testování – je důležitá fáze ověřování očekávaných funkcí systému v reálných podmínkách. Případné chyby jsou opraveny a znovu je prováděno testování.
- Údržba - je fáze, ve které se sledují vlastnosti systému, dolaďuje se, v průběhu času se mění požadavky a systém se modifikuje, reorganizuje.

S rozvojem informačních technologií se mění metodologie i metody a prostředky. Konceptuální návrh databáze začíná analýzou, ze které vyplyne, jaké informace je nutno uložit v databázi a v jaké struktuře. Komplexnější přístup k problematice analýzy a návrhu softwaru je jistě náplní speciálních předmětů, následující řádky jsou úvodem k typicky databázovým prostředkům pro konceptuální modelování formou ER-modelu.

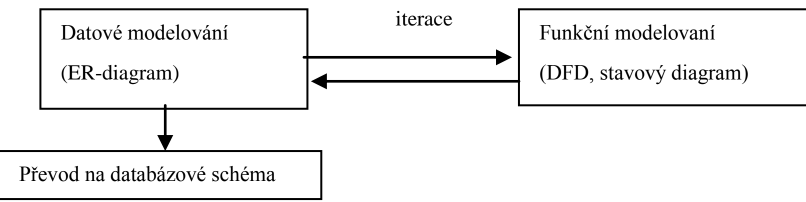
Dá se říci, že v databázových aplikacích (soustředíme se na řešení datové struktury na úrovni relačních SŘBD) se často setkáváme s klasickými přístupy – konceptuální ER model nebo rozšířený EER model se transformuje do relační databáze, ale rozšiřuje se i objektově orientovaný návrh, používající UML a nové metody návrhu (diagram tříd, …). Výhodné je transformovat objektový návrh do objektového nebo objektově-relačního databázového systému. Důvody obliby ER modelu hledejme hlavně v převažujících plochých typech aplikací, optimálně využívající relační systémy a setrvačnosti v používání osvědčených postupů, atd.. V této kapitole se soustředíme na první dvě fáze, které řeší základní problémy – jak v databázi popsat sémantiku dat a jak data strukturovat. Jednoznačnost a smysluplnost sémantiky konceptuálního schématu určuje jeho korektnost.

konceptuální schéma databáze je implementačně nezávislý popis části reality, která se týká IS, prostřednictvím modelového pohledu

2.1 Analýza a návrh IS

Výstupem datové analýzy je konceptuální schéma databáze, jako výsledek iteračního procesu, jehož vstupem jsou požadavky různých (skupin) uživatelů (charakterizují objekty aplikační domény, parciální data a funkce). V průběhu procesu se prvotní návrhy datové struktury integrují a modifikují podle toho, jak vyhovují požadavkům funkční analýzy. Klasickými strukturovanými nástroji funkční analýzy jsou například diagram datových toků (DFD) a stavový diagram.

.



Principy konceptuálních modelů:

- oddělení konceptuální a interní úrovně
- orientace na objekty, entity ne na záznamy a soubory
- bohatší koncept, v relačním modelu jsou relace využívány na „všechno", reprezentují entity, vícehodnotové atributy, asociace, agregace, dědičnost, …
- možnost využít úrovně abstrakce v komplexních objektech k zakrytí detailů, možnost modelovat přímo aplikační objekty.
- funkcionální podstata vztahů (atribut nebo funkce je jediným konstruktem)
- ISA hierarchie (práce s nadtypy a podtypy)
- Hierarchický mechanismus (objekty lze konstruovat z jiných objektů, formou agregace, seskupováním do množin, tříd)

Typ relace je množina smysluplných asociací mezi entitními typy v informačním systému

Průvodce studiem

Konceptuální návrh řeší, prostřednictvím ER modelu otázky:
Jaké entity (objekty) a v jakých vztazích a struktuře jsou v analyzovaném systému ?
Jaké informace o těchto entitách, případně vztazích se mají uložit do databáze ?
Jakým integritním omezením musí vyhovovat data v databázi ?
Také prakticky - jak navrhnout ER diagram tak, aby byl srozumitelný a přehledný (aby na některých úrovních zakrýval příliš velké detaily), aby jeho transformace do relačního modelu byla optimální ?

Výskyt vztahu je identifikovatelná asociace mezi jednotlivými entitami entitních typů, zúčastněných ve vztahu.

2.2 ER Model

ER model chápe realitu, případně její sledovanou část, jako množinu objektů (entity) a vztahů mezi nimi (relationship). To představuje přirozený, ale zjednodušený pohled na svět. Model pracuje s těmito konstrukty a pojmy:

Entity odpovídají objektům reálného světa (osoba, věc, …)a jsou popsány pomocí hodnot svých vlastností. *Entita* musí být rozlišitelná od ostatních entit a existovat nezávisle na nich.

Stupeň relace je počet zúčastněných entitních typů ve vztahu

Relace – vztah, případně typ vztahu, je vazba mezi dvěma nebo více entitami. Vztahová množina R je určena:

$R \subset E_1 \times E_2 \times \dots \times E_n = \{ (e_1, e_2, \dots, e_n) \mid e_1 \in E_1, e_2 \in E_2, \dots, e_n \in E_n \}$, kde e_i je entita, E_i je entitní typ, n je stupeň relace.

Atribut je vlastnost entity nebo vztahu.

Popisný typ (doménu atributu) definujeme jako jednoduchý datový typ (množina přípustných hodnot, množina operací).

Doménu atributu tvoří přípustné hodnoty atributu.

Atribut je funkce, která přiřazuje entitám nebo vztahům hodnotu vlastnosti (popisného typu, domény), je charakteristikou entity. Je zadán svým názvem (identifikátorem) a datovým typem. Každá entita je reprezentována množinou dvojic {atribut, hodnota dat}. Rozlišujeme několik typů atributů, např.

- Jednoduché – jedna atomická hodnota a skupinové (strukturované, kompozitní, složené) - struktura nemusí být jednoúrovňová, ale může vytvářet obecně celou hierarchii. Skupinový atribut vznikne vytvořením složitější struktury z jednotlivých položek. V obecných úvahách je užitečné užívat skupinové atributy, pokud potřebujeme někdy přehledně popisovat celou skupinu, jindy dáme přednost podrobnější reprezentaci pomocí jednotlivých složek a zploštěním víceúrovňové struktury.

např. ADRESA proti {ULICE, ČÍSLO POPISNÉ, MĚSTO, PSČ, STÁT}

- Vícehodnotové - atributy obsahují opakující se stejné položky, tedy jsou představovány množinou hodnot.

Např. AUTOR KNIHY – {J. Ullman, J. Widom}

- Odvozené atributy – požadovaná hodnoty atributů, které nejsou uloženy v tabulkách, vypočteme z hodnot jiných atributů, uložených v tabulkách.

Např. Známe POČET OBYVATEL a PLOCHA STÁTU a vypočteme HUSTOTA OBYVATEL

HUSTOTA OBYVATEL = POČET OBYVATEL / PLOCHA STÁTU

- S nedefinovanou hodnotou (NULL), případně předdefinovanou hodnotou (default).

Atribut -

jednoduchý obsahuje atomickou hodnotu

kompozitní obsahuje strukturu hodnot

vícehodnotový obsahuje množinu hodnot

odvozený obsahuje hodnoty odvozené z jiných atributů

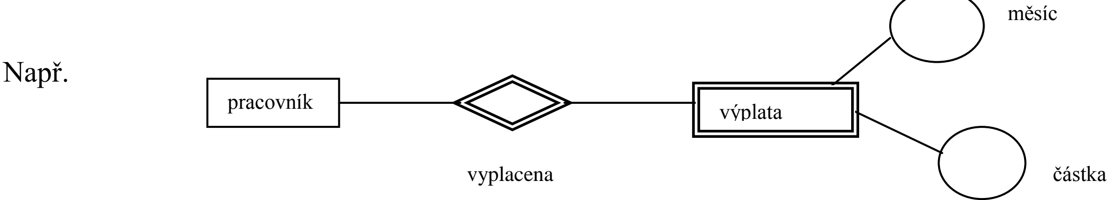
Typ entity - množina objektů stejného typu, abstrakce popisující typ objektu. Je definován jménem a množinou atributů. Jednotlivé entity nazýváme také výskyty, nebo instance objektů entitního typu.

Silný entitní typ – Entitní typ existenčně nezávislý na jiném entitním typu.

Slabý entitní typ – Někdy nejsou dvě instance jednoho entitního typu rozlišitelné pomocí svých atributů, jsou rozlišitelné až pomocí toho, že jsou povinně v identifikačním vztahu k další entitě jiného typu (silné, regulární). V identifikačním klíči takového slabého entitního typu musí mýt i vazební atribut, případně atributy (cizí klíč) identifikačního vlastníka. Graficky v ER diagramu se slabý entitní typ často znázorňuje dvojitým obdélníkem, identifikační vztah dvojitým kosočtvercem.

Entitní typ je skupina objektů se stejnými vlastnostmi, identifikované v informačním systému s nezávislou existencí

Např.

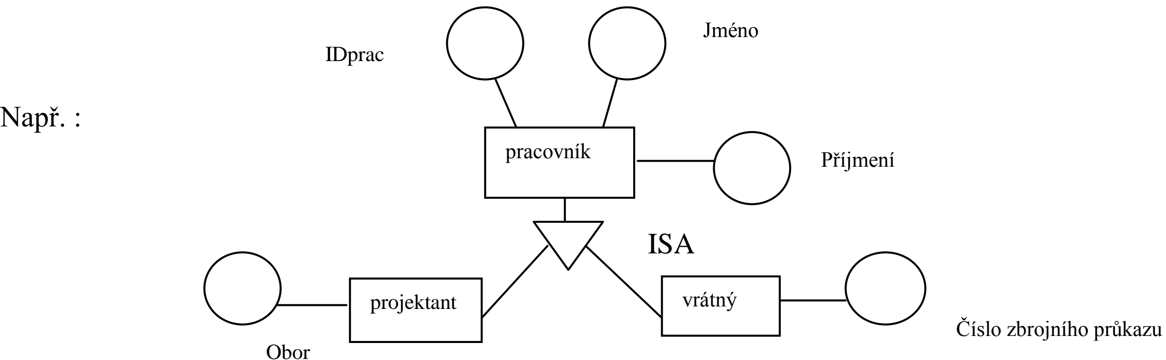


Výskyt (instance) entity je jednoznačně identifikovaný objekt entitního typu.

Slabý entitní typ je existenčně závislý na jiném entitním typu

ISA hierarchie. Někdy se mezi sledovanými objekty vyskytují objekty podobného typu, sémanticky vyjadřující asociace – generalizaci (jedním směrem) nebo specializaci (opačným směrem), popsané řadou stejných atributů a lišících se jen v některých atributech. Specializace je proces maximalizující rozdíly mezi podobnými entitními typy identifikací rozdílných rysů. Naopak generalizace je proces minimalizující rozdíly mezi podobnými entitními typy identifikací jejich společných rysů. Jestliže při většině manipulací s daty obou typů entit se provádějí akce nad společnými údaji, bude vhodné definovat společný typ entity, který bude mít atributy společné a speciální typy se speciálními atributy, které rozlišují odvozené entity. Někdy hovoříme o nadtřídě, respektive podtřídě. Takový vztah definuje ISA hierarchii.

Např. :



IO:

Primární klíč je vybraný kandidátní klíč entitního typu, identifikující každou entitu.

Agregace reprezentuje vztah 'je částí', 'obsahuje' mezi dvěma entitními typy, z nichž jeden představuje celek a druhý jeho část. Zvláštní případ agregace je *kompozice* pro případy silného, nesdíleného vlastnictví části celkem, se stejnou délkou života obou entit.

Průvodce studiem

Integritní omezení ER modelu se týkají identifikačních klíčů, speciálně primárního klíče, referenční integrity, domény atributů, kardinality a členství ve vztahu.

Kardinalita vazby popisuje maximální počet entit, zúčastněných entitních typů v typu relace

Účast ve vztahu určuje, zda se ve vazbě zúčastní všechny, nebo jen některé entity

Integritní omezení (IO) ER modelu jsou logická omezení na typy a hodnoty atributů, entit a vazeb taková, aby konceptuální schéma co nejlépe a nerozporně odpovídalo zobrazované realitě.

Jeden atribut nebo množinu atributů, které jednoznačně určují entitu v množině entit, nebo vztah v množině vztahů, nazveme *identifikačním klíčem,* obecně nadmnožinou klíče (superkey). Množinu všech minimálních podmnožin atributů entity, které ji identifikují, nazýváme kandidátní klíče. Jeden z kandidátních klíčů zvolíme za primární klíč. Vybíráme ten, který je z hlediska zpracování dat nejefektivnější, nebo přirozeně identifikující. Často se volí i uměle dodefinovaný identifikační atribut (automaticky generované přirozené číslo), jako klíč pro efektivnější provádění operací. V ER diagramu se značí obvykle podtržením.

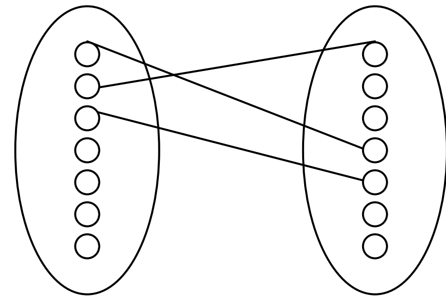
Na hodnoty atributů mohou být kladeny omezující podmínky rozličného charakteru, které respektují meze dané sémantikou dat a které představují doménové integritní omezení.

Další forma integritních omezení na úrovni ER diagramu se týká vztahů.

1. *Kardinalita vztahu* - Binární vztah typů entit E1 a E2 může mít jeden ze tří poměrů:

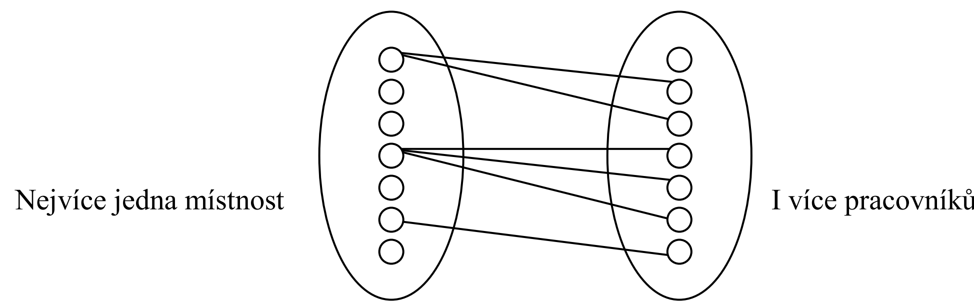
1:1 - jedné $e1 \in E1$ odpovídá ve vztahu nejvýše jedna $e2 \in E2$ a naopak, jedné $e2 \in E2$ odpovídá nejvýše jedna $e1 \in E1$;

např. vztah *vedoucí představitel státu* (stát – prezident)



1:N, nebo s opačnou orientací **N:1** - jedné e1 ∈ E1 odpovídá ve vztahu obecně několik e2∈ E2, ale jedna e2 ∈ E2 má vztah pouze k jedné entitě e1 ∈ E1

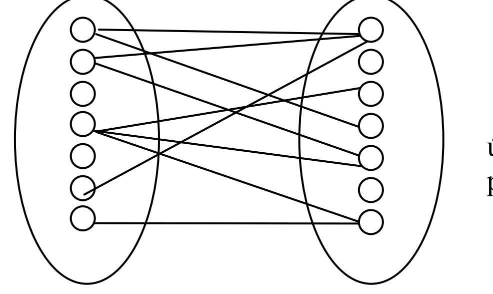
např. vztah *sedí v* (místnost- pracovník)



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.

např. vztah *pracuje na* (pracovník – úkol)

Pracovník řeší i více úkolů



úkol je řešen i více pracovníky

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

Rekurzívní relace je speciálního typu, ve kterém jsou zúčastněny entity stejného typu, ale v různých rolích.

2. Členství ve vztahu – vyjadřují možnost samostatné existence entity (nepovinné, fakultativní členství)

- případ, kdy jsou entity existenčně svázány ve vztahu (povinné, obligatorní členství)

Přitom může mít jedna entita povinné členství, druhá nepovinné. Typ povinnosti členství alternativně zaznamenáváme graficky v ER diagramu značkou(např. plným kroužkem na straně příslušného entitního typu, nepovinnost prázdným kroužkem), nebo společně s kardinalitou

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

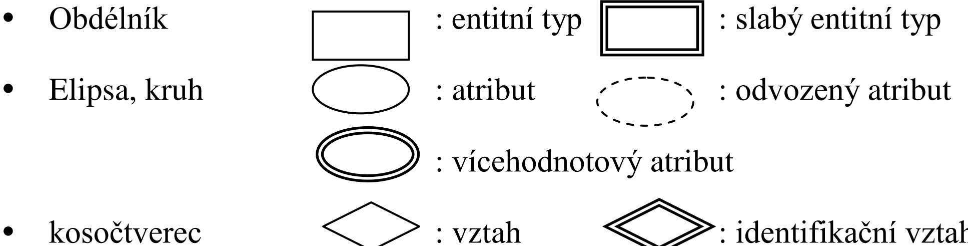
kde min je hodnota 0 (nepovinné) nebo 1 (povinné), max je hodnota 1 nebo M dle kardinality vztahu.

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

Mezi další typy integritních omezení, které částečně souvisí a používají se v relačním modelu dat, do kterého je ER diagram většinou transformován, můžeme zahrnout omezení na unikátní = neopakující se hodnoty v odpovídajících jednotlivých atributech nebo skupinách atributů. Dále může být použita referenční integrita, která zabezpečuje vztahy mezi objekty tak, aby případné odkazy nemohly odkazovat na objekt, který se v databázi nevyskytuje.

Obr. 8 Dr. Peter Chen, autor ER modelu (1970)

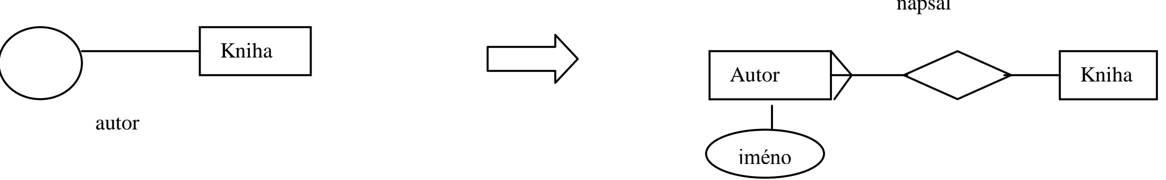
ER diagram (Chenova notace)



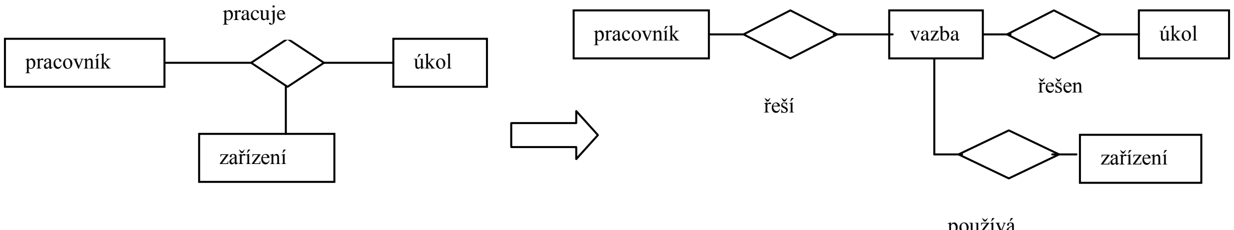
V současnosti se stále častěji využívá rozšířený (enhanced) ER model – EER s podporou agregace a kompozice, používající UML.

Vymezení pojmů entita, vztah a atribut je velmi volné, při modelování se podle zkušeností analytika mohou z různých důvodů konstrukty transformovat, většinou se záměrem zvolit podobu ER diagramu ve formě vhodné pro převod do relačního databázového modelu. Jednoznačná pravidla pro klasifikaci neexistují, výsledná podoba schématu závisí na subjektivním chápání zadání analytikem. Následují ukázky některých typových situací

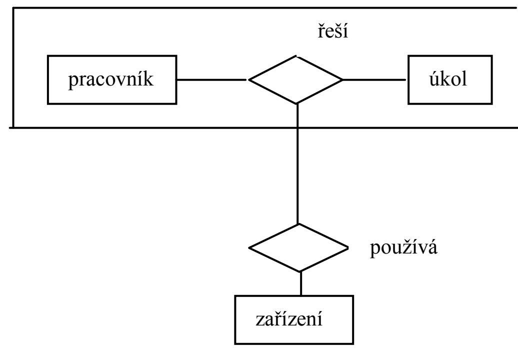
1. Převod vícehodnotového atributu na vztah mezi entitami



2. Převod ternárního vztahu na binární – pracovník používá k řešení úkolu zařízení



3. Jinak ER agregace předcházejícího případu – vytvoří se agregovaná entitní množina



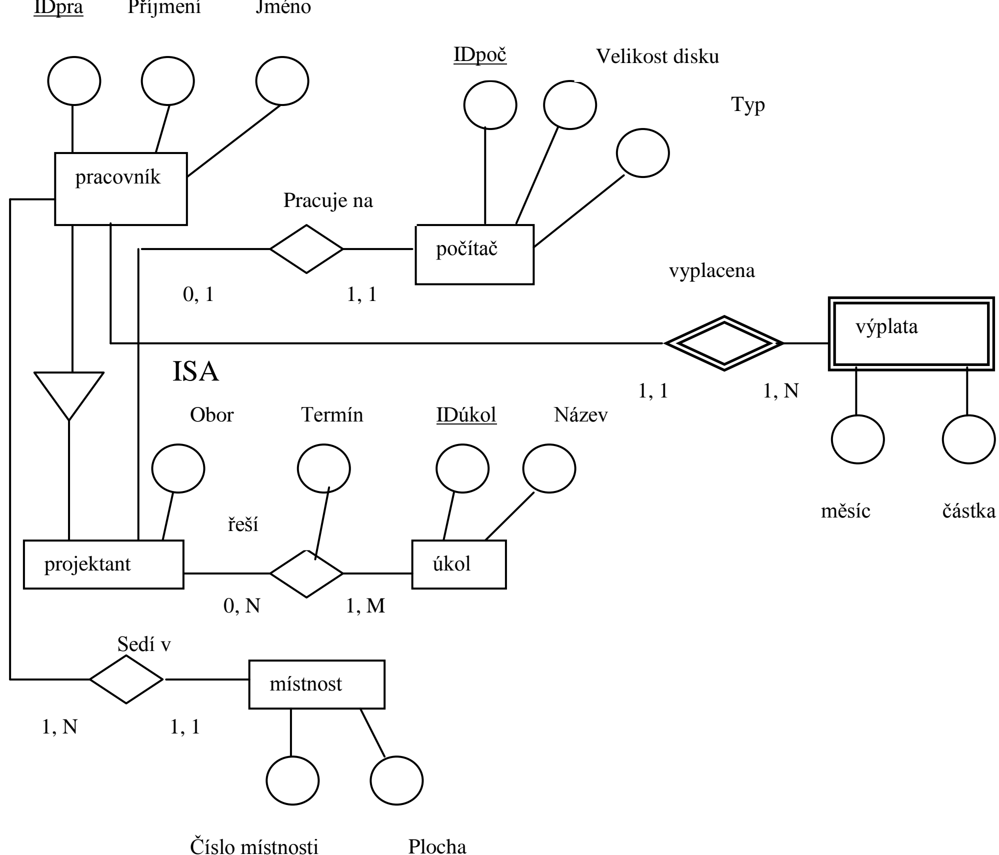
Průvodce studiem

Při návrhu ER diagramu musíme vyřešit dilema:
Modelovat koncept jako entitu, nebo atribut?
Modelovat koncept jako entitu, nebo vztah?
Modelovat koncept jako vztah binární, nebo n-ární, použít agregaci, ISA hierarchii?

Vlastní návrh ER diagramu můžeme provádět principiálně několika způsoby (velice schématicky) :

1. shora – dolů : popíšeme typy entit, typy vztahů a jejich atributy na určité úrovni podrobnosti, zformulujeme IO, vyzkoušíme funkčnost navržené struktury, …(zjemníme pojmy, iterujeme)
2. zdola – nahoru : vycházíme z jednotlivých typů objektů na nejnižší úrovni, vytvoříme množinu všech požadovaných informací – potenciálních atributů jako výchozí univerzální relaci, definujeme a pracujeme s funkčními závislostmi mezi atributy, často i s pomocí počítače vhodnými algoritmy seskupíme atributy do entitních typů a vztahů. Vznikají tak dílčí nezávislé pojmy, integrují se do komplexnějších celků,
3. zevnitř ven : Zaměříme se na nejdůležitější, jasné pojmy. Potom přes pojmy blízké výchozím se propracujeme ke vzdálenějším (jakoby všemi směry, ne jen zdola nahoru), postupným spojováním vytvoříme celek.
4. kombinovaně – vhodná kombinace předchozích způsobů

Příklad ER diagramu :



Obr. 9 Příklad ER diagramu

2.3 Konceptuální model HIT

Původním českým produktem je konceptuální model HIT (Homogenita-Integrovanost-Typy). Je založen na teorii typovaného Lambda-kalkulu, která vychází z predikátové logiky vyšších řádů. Umožňují pracovat s celou hierarchií typů (objekty, třídy objektů, třídy tříd ap.). Tím jsou jeho vyjadřovací schopnosti silnější a lépe postihnou složitost reálných objektů a jejich chování.

Na rozdíl od ER modelu, který používá pojmů entita a vztah, používá model HIT pojmy objekt a funkce, vztahy definuje jako funkce. Definuje vlastní grafickou reprezentaci základních datových typů a datových funkcí.

Shrnutí

ER diagram slouží pro modelování datové struktury ve formě popisu entitních množin a jejich vztahů a také vlastností jednotlivých entit. Existují různé notace pro kreslení ER diagramu, klasická Chánova používá obdélník pro entitu, kroužek pro atribut a kosočtverec pro vztah. Kardinalita vztahu může být 1:1. 1: N a N:M. Podmnožina atributů entity tvoří identifikační klíč, pokud kombinace hodnot těchto atributů určuje jediný výskyt entity. Primární a unikátní klíč, spolu s kardinalitou vazby, povinným a nepovinným členství ve vztahu, určení domén jednotlivých atributů a referenční integrity patří mezi integritní omezení ER modelu.

Specializace a generalizace je v ER diagramu zachycena pomocí ISA hierarchie. Modelování a transformace v ER diagramu umožňují variantně modifikovat schéma například přechodem z atributu na entitu, modelováním vazeb se změnou jejich počtu a stupně, agregováním struktury entit a vztahů.

Pojmy k zapamatování

- ER model, entita, slabá entita, atribut, vztah, asociace, agregace
- Integritní omezení - primární klíč, kardinalita vztahu, členství ve vztahu, referenční integrita
- ISA hierarchie, specializace, generalizace

Kontrolní otázky

10. *Jaké konstrukty a s jakou charakteristikou má ER model?*
11. *Jaká integritní omezení má ER model?*
12. *Co je a jak určíme v konkrétním případě kardinalitu vazby?*
13. *Co je slabý entita a jak se s ní pracuje v ER diagramu?*
14. *Co je a k čemu slouží ISA hierarchie?*

Úkoly k textu

Vytvořte z vlastního jednoduchého zadání ER diagram s několika vazbami. Určete kardinalitu a účast ve vztahu, modifikujte ER diagram ve dvou funkčních variantách, formulujte a znázorněte integritní omezení.

Navrhněte ER diagram se slabou entitou, ISA hierarchií.

3 Logické modely dat

Studijní cíle: Po prostudování kapitoly by student měl znát historii a charakteristiku jednotlivých logických databázových modelů, jejich integritní omezení. Na příkladech nakreslit jednoduché datové struktury ve zvoleném modelu transformací z ER diagramu. Student by měl popsat relační strukturu dat a její vlastnosti, vysvětlit základní termíny, definovat relaci, kategorie relací a n-tice, formulovat pravidla pro transformaci z ER diagramu do relačního modelu. Charakterizovat objektový model a jazyk ODL.

Klíčová slova: Síťový databázový model, Bachmanův diagram, C-množina, hierarchický databázový model, relační model, integritní omezení, klíč, referenční integrita objektový model, ODL

Potřebný čas: 4.hodiny

Pro uživatele při konkrétní práci s DBS je nutno zvládnout příkazy JDD a JMD příslušného SŘBD a rozumět datovému modelu, který podporuje. V současnosti se v praxi setkáme v převážné většině případů s relačním nebo relačně-objektovým modelem, ale začneme přehledem historicky prvních modelů.

3.1 Síťový databázový model

V roce 1971 byla skupinou DBTG (Data Base Task Group) při sdružením CODASYL (Conference on Data System Languages) definována architektura SŘBD síťového modelu, založená na souborech a vztazích mezi záznamy. Vztahy mezi záznamy se modelují pomocí spojek. Logickému modelu databáze se říká schéma a má například s pomocí grafové struktury formu Bachmanova diagramu. Při definici schématu se typ entity nazývá *typ záznamu - Record* a může obsahovat položky – jména atributů čtyř typů – *jednoduché, opakující se, složené nebo opakující se složené.* Jednotlivým záznamům s nějakou kombinací hodnot odpovídajících položek říkáme *výskyty záznamu* příslušného typu. Mohou existovat dva identické záznamy. Tyto výskyty jsou rozlišeny pouze hodnotou identifikačního *databázového klíče*, který je systémem automaticky přidělován každému výskytu záznamu.

Síťový model definuje pouze funkcionální binární vztahy typů 1:1 a 1:N mezi dvěma typy záznamů a tento vztah se nazývá set, C-množina, případně CS-typ. Je definována pomocí svého typu vlastníka a typu člena nebo členů, jako pojmenovaná uspořádaná dvojice. Realizace vztahu je potom ve *výskytech CS-typu.* Výskyt CS-typu obsahuje právě jeden výskyt záznamu vlastníka a právě ty výskyty záznamů člena C-množiny, které jsou s vlastníkem výskytu setu v příslušném vztahu. Výskyt setu může obsahovat pouze výskyt záznamu vlastníka (prázdný výskyt setu). Příklady operací

- vytvoř databázové schéma,

- vytvoř nový záznam daného typu, zruš daný záznam, změň daný záznam,

- vlož člen do výskytu C-množiny daného vlastníka,

- vyřaď člen z daného výskytu C-množiny

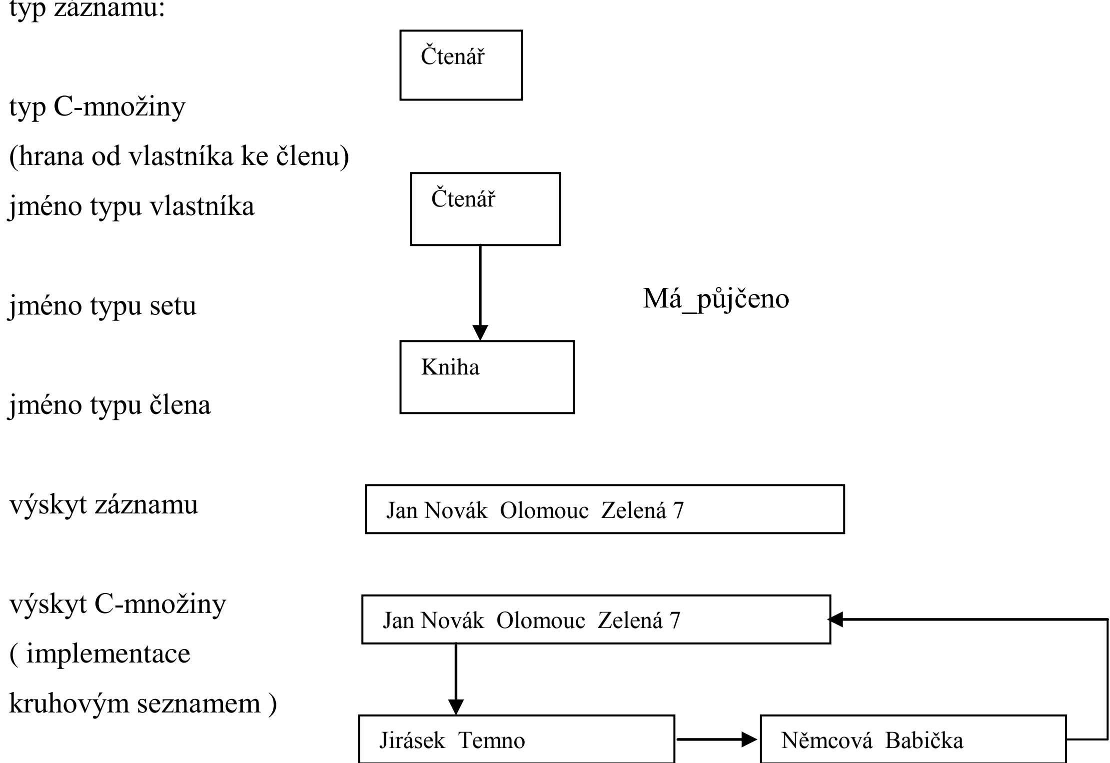
- najdi první člen ve výskytu C-množiny daného vlastníka,

- najdi následníka ve výskytu C-množiny daného vlastníka pro daný člen,

- najdi vlastníka ve výskytu C-množiny, známe-li člen

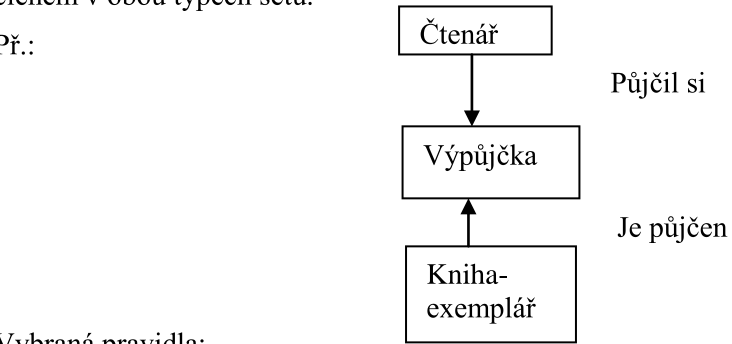
Schéma je tedy tvořeno zadáním typu záznamů a CS-typů. Implementace je realizována pomocí ukazatelů formou kruhových seznamů.

Grafické znázornění modelu (Bachmanův diagram):



Vztah typu M:N není možno realizovat přímo a realizuje se (již na úrovni konceptuálního schématu) pomocí dvou vazeb typu 1:M prostřednictvím průnikového typu záznamu. Ten je členem v obou typech setů.

Př.:



Vybraná pravidla:

- Tentýž typ záznamu může být současně vlastníkem jedné C-množiny a členem v jiné C-množině.
- Záznam libovolného typu může být členem, případně vlastníkem libovolného počtu C-množin.

Vybraná omezení (integritní):

- Nejsou povoleny rekurzívní typy vztahů.
- Vlastník a člen C-množiny nemohou být záznamy téhož typu. Unární vztah 1:N se realizuje prostřednictvím pomocného typu záznamu
- Nemůže existovat člen C-množiny bez vlastníka

Příklad deklarace databáze :

Typy záznamů :

Čtenář(jméno, adresa)

Výpůjčka (dat_půjčeno, dat_vráceno)

Kniha-exemplář (autor, název)

Typy C-množin :

Půjčil si(Čtenář, Výpůjčka)

Je půjčen(Výpůjčka, Kniha-exemplář)

Známé implementace – IDMS, ADABAS, DMS 1100

3.2 Hierarchický databázový model

Hierarchický databázový model můžeme pojmout jako speciální případ síťového modelu. Diagram datové struktury tvoří strom (graf bez cyklů) nebo les stromů. Prakticky to znamená, že záznam může patřit maximálně do jednoho setu. Typy záznamů se podobají síťovému modelu, ale jsou jednodušší, obsahují jen jednoduché atributy. Při popisování hierarchického modelu se mění terminologie. Místo vlastník se užívá pojmu otec(rodič), místo člen pojmu syn(dítě).

v hierarchickém modelu má každý syn právě jednoho rodiče, v síťovém modelu jich může mít více

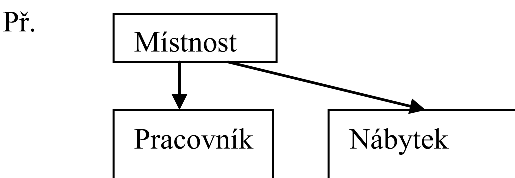
Databázové schéma je tvořeno zadáním typu záznamů a hierarchické struktury *definičních stromů.* Záznamy stromů jsou uspořádané. Lze definovat pohledy. Databázi lze chápat jako jediný strom se systémovým kořenem. Pro ilustraci při manipulaci s daty postupujeme následovně:

1. Nastavení na kořen stromu DB
2. přechod na požadovaný strom
3. přechod mezi úrovněmi hierarchie
4. přechod mezi položkami na jedné úrovni
5. požadovaná operace se záznamem na specifikované pozici

Problém se vztahem M:N můžeme řešit podobně, jako u síťového modelu, rozložením typu vztahu M:N na dva 1:N – více definičních stromů, nebo pomocí duplikovaných záznamů – k záznam typu dítě se vytvoří duplicitní záznamy se vztahy ke všem požadovaným rodičovským.

Známé implementace – IMS firmy IBM

Př.



Průvodce studiem

Hierarchický i síťový model závisí na struktuře dat IS, informace tvoří příslušný graf a implementují je kolekce různých typů záznamů, nejčastěji ve vztahu 1: N.Pro pohyb datovou strukturou se používá odkazů – to vystihuje termín navigace při vyčíslování dotazů. Při změnách v datových strukturách se typicky musí měnit i aplikační programy.

3.3 Relační model

Databázový relační model dat navrhl v roce 1970 pracovník firmy IBM - Dr. E. F. Codd. Základem je matematické zobecnění pojmu soubor pomocí silného formalizmu matematické relace a využívání operací relační algebry.

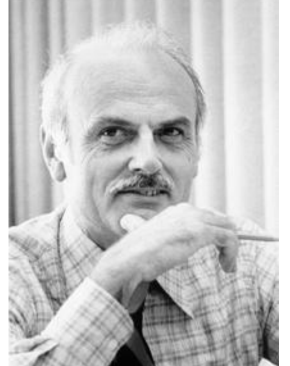
Hlavní pravidla relačního modelu :

- Databázi, popisující úsek reálného světa, tvoří na logické úrovni konečná množina relací - tabulek.
- Transparentnost při manipulaci s daty – nezajímáme se o přístupové mechanizmy k datům, obsaženým v relacích. Pro manipulaci s daty je silná matematická podpora, důsledné oddělení logické úrovně dat od implementace.
- Informace v databázi jsou vyjádřeny explicitně na logické úrovni jediným způsobem - hodnotami v tabulkách, data jsou přístupná pomocí JMD, parametrizovaném kombinací logického jména tabulky, logických jmen sloupců a jejich výrazů, nejčastěji s hodnotami všech typů klíčů.
- Systematická podpora zpracování nedefinovaných hodnot. Umožňuje práci s neúplnými daty.
- Dynamický on-line katalog založený na relačním modelu. Schéma databáze je vyjádřeno na logické úrovni stejným způsobem, jako uživatelská databáze, ve formě systémových tabulek. Správce databáze může používat stejný relační jazyk pro dotazy na strukturu databáze, jako uživatel při práci s daty aplikace.
- Nezávislost IO na aplikaci - integritní omezení jsou definovatelná jazykovými prostředky SŘBD, jsou uložena v katalogu, ne v aplikačním programu.
- Omezení redundance dat v relační databázi - jsou navrženy postupy, umožňující normalizovat relace, tedy navrhovat potřebné relace s minimální strukturou uložení na disku.
- Vazby mezi entitami jsou reprezentovány opět relacemi. Formálně se s nimi pracuje stejně jako s entitními relacemi.

Relace je tabulka se sloupci a řádky

Atribut je pojmenovaný sloupec

Doména je množina přípustných atomických hodnot pro jeden nebo několik atributů



Obr. 10 Dr. Edgar Frank Codd

3.3.1 Relační struktura dat

RELACE

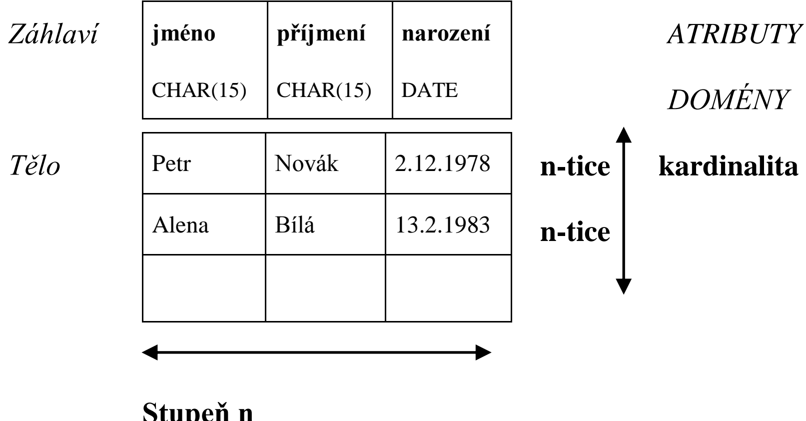


Schéma relace **R** je výraz tvaru **R**(A_1 : D_1, A_2 : D_2, … A_n : D_n) $A_i \neq A_j$ pro $i \neq j$, kde **R** je jméno schématu, A = {A_1, A_2,..., A_n} je konečná množina jmen atributů, f(A_i) = D_i je zobrazení přiřazující každému jménu atributu A_i neprázdnou množinu, kterou tradičně nazýváme doménou atributu D_i, . V terminologii relačních jazyků odpovídá pojmu relace praktičtější pojem tabulka, která se skládá ze záhlaví a těla. Záhlaví odpovídá schématu relace, tělo je tvořeno množinou n-tic (a_1, a_2,..., a_n), což je konečná podmnožina kartézského součinu domén D_i příslušejících jednotlivým atributům A_i -

n-tice je jeden řádek relace

stupeň relace je počet jejích atributů

kardinalita relace je počet jejích datových řádků

$R \subset D_1 \times D_2 \times ... \times D_n$, , tedy hodnoty atributů jsou z příslušných domén a vyhovují všem integritním omezením. Hovoříme o přípustné relační databázi, nebo o konzistentní množině relací.

Př. doména pro jméno : {Alena, Petr, Jiří, Jana, …}

Počet n-tic určuje kardinalitu relace. Číslo n se nazývá stupněm relace (aritou). Stupeň relace je relativně konstantní(umožněna modifikace pomocí ALTER TABLE), kardinalita se v čase mění (INSERT, DELETE). Relacím se schématem R říkáme, že jsou jeho instancí, nebo jinak také jsou typu R .

Na tabulku můžeme také nahlížet tak, že většinou (po transformaci z ER diagramu) každý řádek odpovídá jedné entitě, každý sloupec jednomu atomickému atributu. Na rozdíl od matematických relací jsou databázové relace proměnné v čase. Aktualizace databáze, která umožňuje zachytit v databázi změny nastávající v reálném světě, tedy spočívá ve změně aktuálních relací přidáváním, rušením prvků relací nebo změnou hodnot některých atributů.

Relační databáze je kolekce normalizovaných, unikátně pojmenovaných relací.

Vlastnosti relací a tabulek

- Pořadí n-tic v relaci (řádků v tabulce) je nevýznamné
- Pořadí atributů v relaci (sloupců v tabulce) je nevýznamné
- V relaci neexistují duplicitní n- tice (je to množina), v tabulce se obecně mohou duplicity vyskytovat, pokud se o jejich odstranění explicitně nepostaráme (primární klíč, … distinct …)
- Jména atributů jsou v relaci, tabulce unikátní,
- každý údaj (hodnota atributu ve sloupci) je atomickou položkou z jedné domény (vyhovuje 1NF)
- V praktických aplikacích je každý řádek tabulky jednoznačně identifikovatelný hodnotami jednoho nebo několika atributů (primárního klíče) – nepřipouští se duplicitní řádky.

Relační schéma je pojmenovaná relace, definovaná množinou dvojic atribut – doména

Schéma relační databáze je množina schémat relací

Schéma relační databáze je dvojice (**R**, **I**), kde **R** je konečná množina relačních schémat {R1(A1), R2(A2),. . . , Rm(Am)} a **I** je množina IO.

Relace můžeme kategorizovat na:

- Pojmenované

  - *bázové, reálné* jsou fyzicky existující (CREATE TABLE …)

  - *pohledy* jsou virtuální, odvozené z bázových (CREATE VIEW …)

  - *snímky* jsou odvozené, statické, ale existující (CREATE MATERIALIZED VIEW / SNAPSHOT …)

- Nepojmenované

  - dočasné (CREATE GLOBAL TEMPORARY TABLE …)

  - výsledky a mezivýsledky dotazů (SELECT …)

3.3.2 Integritní omezení v relačním modelu

Integritní omezení se dělí na

- Specifická – typicky implementována v prostředí procedurálního jazyka, jako rozšíření SQL, ve formě triggeru, uložené procedury. Jsou unikátní, určena specifikou aplikace. Dříve byla implementována v aplikaci.
- Obecná – jednodušší, odvozená z principů relačního modelu, specifikovaná příkazy DDJ při definici tabulek, ověřovaná při manipulaci s daty.

  1. Klíč relace (tabulky) je podmnožina atributů relace, která identifikuje n-tici, tedy splňuje tyto časově nezávislé vlastnosti :

- unikátnost (neexistují dvě n-tice se stejnými hodnotami klíčových atributů)
- Minimálnost (z množiny atributů klíče nelze vynechat žádný atribut, aniž by se tím neporušila unikátnost)
- platnost (hodnoty všech klíčových atributů musí být definované)

Všechny takové podmnožiny nazveme *kandidátními klíči*, z nich jeden vybraný je *primárním klíčem*, zbývajícím kandidátním klíčům někdy říkáme *alternativní*(použití klauzule UNIQUE*)*. Zbývající podmnožiny atributů relace, které nejsou kandidátními označujeme jako *sekundární* (obecně jejich hodnoty určují více n-tic v relaci) a uplatní se například při třídění a spojení relací.

Cizí klíč (jeho hodnoty jsou hlídány referenční integritou) je podmnožina atributů relace, které zároveň tvoří primární (kandidátní) klíč v jiné relaci.Oba klíče by měly být ze stejné domény. Referenční integrita popisuje asymetrický vztah mezi dvěma (tabulka hlavní a závislá), nebo více tabulkami prostřednictvím cizího a kandidátního klíče, je typickou realizací vazby mezi entitami z ER diagramu. SŘBD připustí v cizím klíči jen hodnoty plně nezadané (nedefinované), nebo ty které jsou aktuálně v kandidátním klíči druhé tabulky. Databáze nesmí obsahovat nesouhlasnou hodnotu cizího klíče. Na to reaguje DMJ v příkazech INSERT omezením vložení n-tice do závislé tabulky a DELETE omezením odstranění n-tice z hlavní tabulky, nebo kaskádovým odstraněním n-tic z hlavní relace s propagací do závislé (závislých) relací formou odstranění celých n-tic, či jen nahrazením hodnot cizího klíče prázdnou nebo implicitní hodnotou. Cizí klíč se může odkazovat i na svou vlastní relaci.

Nadklíč relace je množina atributů relace, identifikující jednu n-tici.

Kandidátní klíč je nadklíč relace, jehož žádná podmnožina není nadklíčem.

Primární klíč je jeden vybraný kandidátní klíč.

Cizí klíč tvoří množina atributů v jedné relaci, která má vazbu na kandidátní klíč jiné, nebo téže relace.

NULL reprezentuje neznámou,nedefino vanou hodnotu

2. Doménové IO, testuje hodnoty vkládané do databáze podle oboru hodnot - domén atributů v tabulce, při dotazování testuje smysluplnost operací porovnání.

- Nejjednodušší IO, definuje přípustnost nedefinované hodnoty atributu(klauzule NULL proti NOT NULL)
- Definice implicitní hodnoty atributu – náhrada nedefinované hodnoty v některých příkazech (klauzule DEFAULT)
- Zúžení předdefinovaných datových typů, pomocí logických podmínek v klauzuli CHECK.
- V SQL 92 se setkáme s ASSERTION, což je predikát, jehož podmínky musí databáze splňovat (CREATE ASSERTION …CHECK (…))

Průvodce studiem

Entitní integrita – v bázové relaci nesmí být v žádné složce primárního klíče nedefinovaná hodnota NULL.

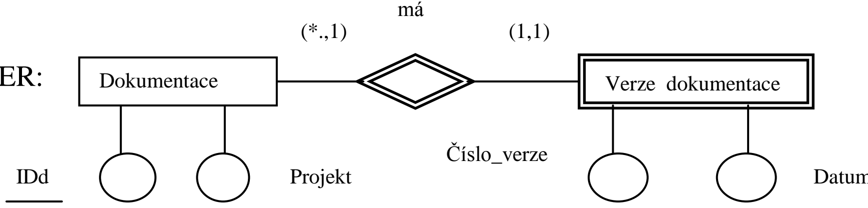
Referenční integrita – hodnota cizího klíče každé n-tice v relaci musí odpovídat hodnotě nadklíče v odkazované zdrojové relaci nebo může obsahovat ve všech složkách klíče nedefinovanou hodnotu NULL.

Mezi IO relačního modelu patří také funkční závislosti mezi atributy. Funkční závislost mezi množinami atributů X a Y (značíme obvykle $X \rightarrow Y$) znamená, že ke každé hodnotě atributů z množiny X existuje nejvýše jedna hodnota atributů z množiny Y. Funkční závislosti se definují již na konceptuální úrovni z kontextu a zadání aplikace a podílí se na upřesnění sémantiky. Podrobněji v dalším textu.

3.3.3 Doporučení pro transformaci ER diagramu do relačního modelu

Cílem je vytvoření (výchozího) schématu databáze v relačním modelu dat (definice záhlaví tabulek) z výchozího konceptuálního schématu aplikace v ER modelu. Nejdůležitějším úkolem je zamezit nebo minimalizovat ztráty spojené s přechodem do nižší úrovně abstrakce, tedy zajistit jakousi informační ekvivalenci. Proto ke schématu relací připojujeme i integritní omezení. Vyberme některá doporučení, společná většině metodologií:

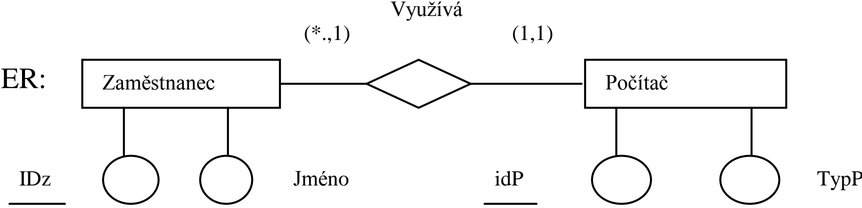
1. Vytvoření relací z regulárních (silných) entitních typů, atributy relací odpovídají jednoduchým atributům entitních typů, u složených typů atributů provedeme dekompozici na jednoduché atributy. Pokud je struktura složitější, transformujeme příslušné subschéma ER diagramu do pro transformaci použitelné podoby. Převezmeme, nebo určíme primární klíč.
2. K transformaci slabého entitního typu potřebujeme znát relaci identifikačního vlastníka. Provedeme transformaci jako v předešlém případě. Dostaneme pouze parciální klíč. Doplníme relaci o atributy identifikačního klíče relace identifikačního vlastníka (tvoří cizí klíč) a přidáním k parciálnímu klíči definujeme primární klíč.



Relace : Dokumentace(IDd, Projekt), Verze_dokumentace(IDd, Číslo_verze, Datum)

3. Typu vztahu odpovídá schéma relace, zahrnující referenční integritní omezení.Obecně pro všechny transformace existuje více variant a záleží na dalších vazbách a IO (povinné a nepovinné členství ve vztahu, …), předpokládaných datech (rozsah, počet (relativní) propojených entit) a převažujících dotazech. Problematiku naznačíme na příkladech.

Reprezentace vztahů 1:1



a) Při vztahu (1,1)-(1,1) , každý pracovník má právě jeden počítač se nabízí možnost spojit entitní typy do jedné relace, zvolit primární klíč z jednoho entitního typu, sloupec původního primárního klíče druhé relace (idP) doplníme o IO UNIQUE. Sémanticky z pracovního počítače uděláme charakteristiku zaměstnance. Toto řešení nepředpokládá další vazby na entitní typ počítač.

aa) Zaměstnanec(IDz, jméno, idP, TypP)

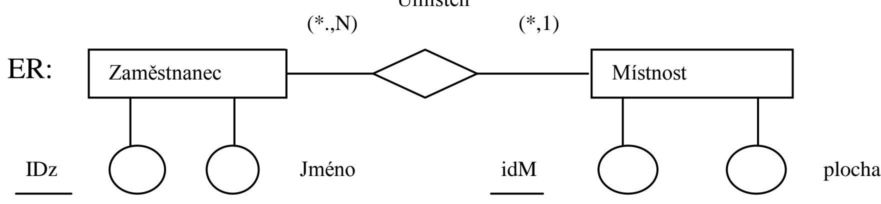
Jinak transformaci provedeme převedením na dvě relace. Do záhlaví jedné (Zaměstnanec) přidáme jako vazební položku primární klíč druhé (idP), připojíme IO – referenční integritu, UNIQUE, případně NOT NULL. Nový sloupec je cizím klíčem, NOT NULL kontroluje povinné členství ve vztahu. Tedy pokud by počítač neměl každý zaměstnanec – vztah (0,1)(1,1), potom NOT NULL nepoužijeme.

ab) Zaměstnanec(IDz, jméno, idP), Počítač(idP, TypP)

Pro některé případy (velká záhlaví) a převažující užívání speciálních dotazů(např. na atributy vazby) může být výhodné provést transformaci do tří relací. Dvě entitní relace jsou definovány entitními typy a záhlaví třetí vztahové relace (pro typ vztahu) vytvoříme z primárních klíčů ve vazbě zúčastněných entitních typů. Ty jsou opět cizími klíči, UNIQUE a NOT NULL. V kombinaci potom tvoří primární klíč. Toto řešení je sémanticky srozumitelné třeba v situaci, kdy vztahový typ má další atributy.

ac) Zaměstnanec(IDz, jméno), Využívá(IDz, idP), Počítač(idP, TypP)

b)Podobně můžeme analyzovat reprezentace vztahů 1:N, N:1.



Vztah (1,N)(1,1) – všechny místnosti jsou obsazeny, všichni zaměstnanci jsou umístěni řešíme nejčastěji vytvořením dvou relací. Do záhlaví relace(Zaměstnanec), která je ve vazbě s kardinalitou 1 přidáme primární klíč z entitního typu(Místnost), který ve vazbě reprezentuje kardinalitu N. Ten je cizím klíčem s dalším integritním omezením NOT NULL (povinné členství ve vztahu), ale již bez UNIQUE.

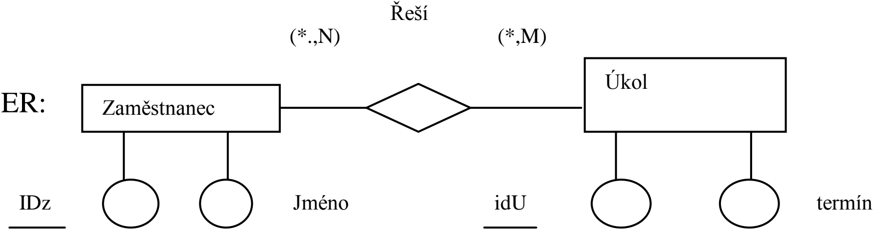
ba) Zaměstnanec(IDz, jméno, idM), Místnost(idM, plocha)

Z podobných důvodů, které byly analyzovány v předchozím případě ac), můžeme provést transformaci do tří relací.

bb) Zaměstnanec(IDz, jméno), Umístěn(IDz, idM), Místnost(idM, plocha)

c)Podobně řešíme reprezentace vztahů N:M. Pokud každý zaměstnanec pracuje nejméně na jednom úkolu a každý úkol je řešen nejméně jedním zaměstnancem, vztah lze popsat jako (1,N) (1,M).

ER:

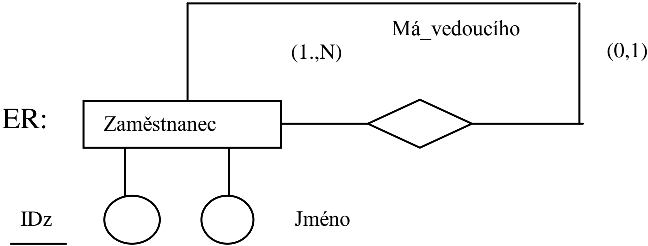


Řešení je dáno postupem v ac) :

Zaměstnanec(IDz, jméno), Řeší(IDz, idU), Úkol(idU, termín)

d) Zvláštními případy, které se ale řeší analogicky jsou unární (rekurzívní) vazby. Např. N : 1

ER:

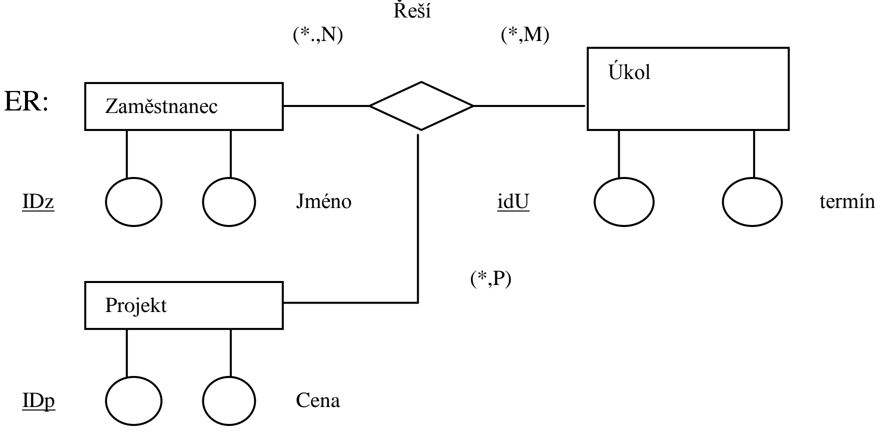


Do relace Zaměstnanec přidáme opět vazební sloupec (IDvedoucí), jehož doména je identická s IDz entitního typu Zaměstnanec.

Zaměstnanec(IDz, jméno,IDvedoucí),

e) Naopak n-ární vztah při povinném členství pro různé kombinace kardinalit transformujeme tak, že definujeme jednu vztahovou relaci se záhlavím složeným z primárních klíčů všech v relaci zúčastněných entitních typů.

ER:



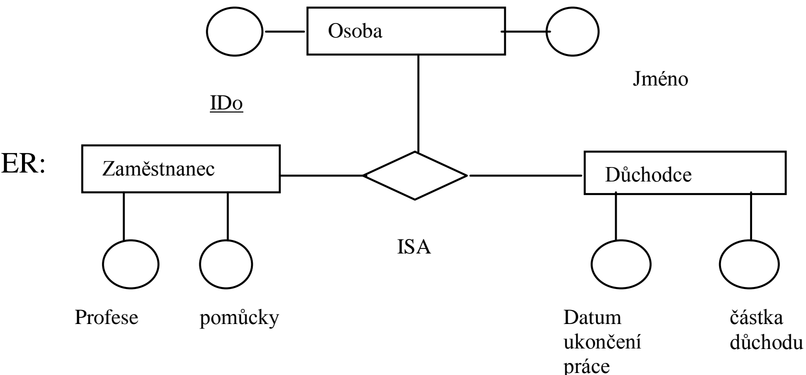
Řešení :

Zaměstnanec(IDz, jméno), Úkol(idU, termín), Projekt(IDp, cena), Řeší(IDz, idU, IDp),

Nebo relaci rozložíme na binární vazby.

e) Se stejnými principy se setkáme u transformace ISA hierarchie, jde vlastně o speciální vztahy.

ER:



Příklady řešení

ea) Všechny entitní typy z ISA hierarchie transformujeme do jedné relace. V záhlaví jsou všechny atributy entitních typů, primárním klíčem relace je klíčový atribut (IDo) zdroje hierarchie(Osoba). Výhoda – ušetříme čas na spojení, nevýhoda – řídká tabulka s mnoha nedefinovanými hodnotami. Řešení:

Osoba(Ido, Jméno, Profese, pomůcky, Datum_ukončení_práce, částka_důchodu)

Eb) Všechny entitní typy transformujeme do samostatných relací. Součástí primárního klíče relací jsou sloupce, do záhlaví přidané propagací primárních klíčů z nadřazených úrovní hierarchie, zároveň jsou to cizí klíče a vazební položky pro získání sdílených sloupců (společných vlastností) pomocí spojení. Řešení:

Osoba(Ido, Jméno),

Zaměstnanec(Ido, Profese, pomůcky),

Důchodce(Ido, Datum_ukončení_práce, částka_důchodu)

Při konverzi se můžeme setkat (a musíme vyřešit) s těmito problémy:

1. Stejná logická jména sloupců se stejnou (ve vazbách) nebo různou sémantikou. Proto používáme výstižné, úplné unikátní pojmenování, případně tečkové notace.

2. Pokud mnoho vazeb v ER modelu tvoří sekvence nebo cykly, může dojít k nežádoucím efektům, které v konečném důsledku představují neúplnost, zkreslení, omezení nebo chybu.

Tabulka ekvivalentních položek ER modelu a relačního modelu

ER model	**Relační model**
Entitní typ .Entita	relace
1:1 nebo 1:N typ vztahu	cizí klíč (nebo vztahová relace)
M:N typ vztahu	vztah. relace a dva cizí klíče
n-ární typ vztahu	vztah. relace a n cizích klíčů
jednoduchý atribut	atribut
kompozitní atribut	množina atributů
vícehodnotový atribut	relace a cizí klíč
množina hodnot	Doména
klíčový atribut	Primární, kandidátní klíč

3.4 Objektově-relační model

Tlak nových typů aplikací s bohatě strukturovanými daty a přirozený evoluční vývoj vedl k rozšíření relačních systémů o softwarovou vrstvu dodávající relačnímu SŘBD objektový charakter. Rozšiřuje relační model o bohatší typový systém. Takový databázový model dat nazýváme objektově-relační. Mimo jiné jsou podporovány zanořené relace (není vyžadována 1. NF) – jinak formulováno atributy entit – objektů mohou být strukturované, složité datové typy. Dále je možné definovat uživatelské ADT, kolekce typů jako např. množina, multimnožina, seznam, pole, … , reference na objekty, konstruktory složitých objektů. Podpora objektových rysů umožňuje definovat metody objektů, aplikovat dědičnost. V objektově-relačních systémech přebírají n-tice relací roli objektů a proto mají povinně své identifikátory ID, většinou nepřístupné uživateli. Doporučení a vlastnosti OR SŘDB jsou normalizovány v SQL 99 nebo v SQL 3 a představují proces sbližování s čistě objektovými systémy. Proto jsou některé společné rysy podrobněji rozebrány v následující kapitole.

3.5 Objektový model

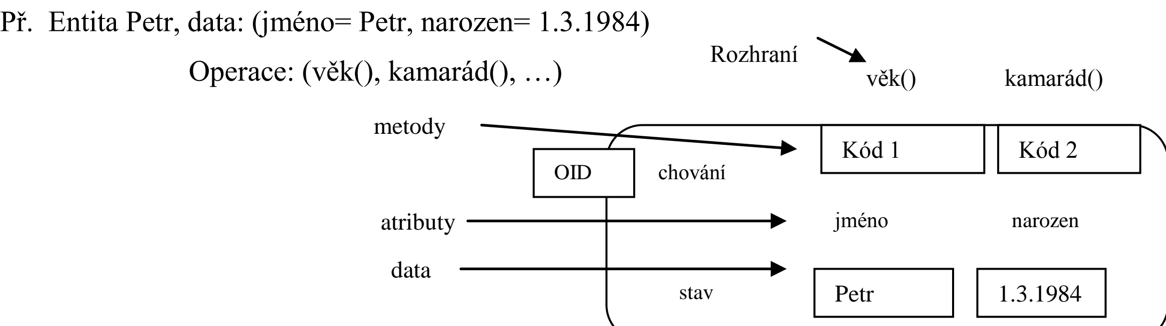
Datové modely, vyhovující aplikacím s plochou jednoduchou datovou strukturou (krátké záznamy pevné délky s atomickými hodnotami atributů) jsou prakticky nepoužitelné v aplikacích s bohatě strukturovanými daty(hierarchie složitých objektů, často se správou vývoje verzí), jako jsou systémy CAD, multimédia, grafické aplikace, hypertextové a dokumentografické systémy. Takové aplikace požadují novou koncepci uložení komplexních dat a efektivnější a abstraktnější operace s nimi – nové metody indexování a dotazování. Vývoj softwaru jasně směřuje k maximálnímu využití výhod objektově orientovaného paradigmatu a technologie ve všech fázích vývoje programu – objektově orientovaná analýza a návrh, modelování procesů, implementaci (s rysy jako zapouzdření – část funkcí, dříve v aplikačních vrstvách, se přenáší do objektu a stává se součástí databáze , použití abstraktních datových typů, atd. s rozšířením o perzistenci objektů) a testování. Objektová technologie v databázových systémech přináší na konceptuální úrovni bezprostřední vztah mezi objektem reálného světa a jeho reprezentací ve formě složitého objektu.

3.5.1 Objektový koncept

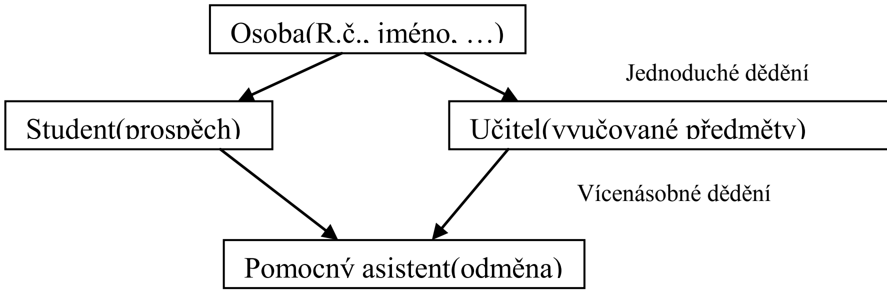
Objekty v sobě kombinují a zapouzdřují data a operace nějaké entity z reálného světa. Data odpovídají atributům a reprezentují stav objektu, operace(metody) definují chování objektu, z hlediska programování představují rozhraní pro zprávy. Metody jsou programy, napsané většinou v univerzálních jazycích (C++, Java). Vlastní data objektu jsou přístupná přímo, na data ostatních objektů jsou odkazy ve formě zaslání zpráv. Velice důležitá je identita objektu (OID), nezávislá na stavu objektu. To vede například k obecnějšímu chápání rovnosti a ekvivalence objektů. Stav dvou objektů může být stejný, ale nejsou si rovny, protože je rozlišuje OID.

Př. Entita Petr, data: (jméno= Petr, narozen= 1.3.1984)

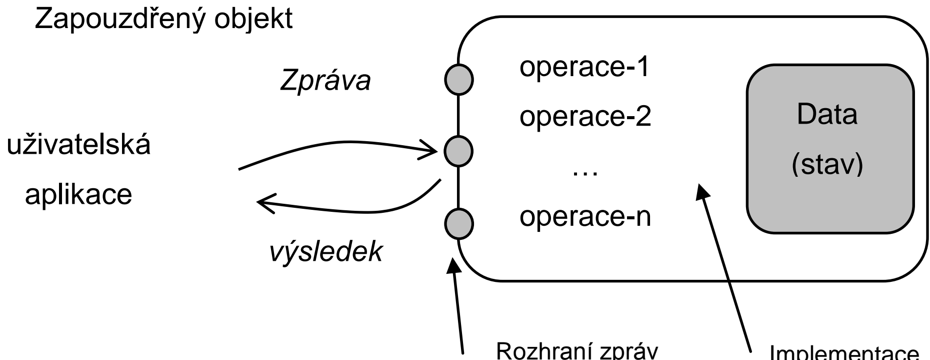
Operace: (věk(), kamarád(), …)



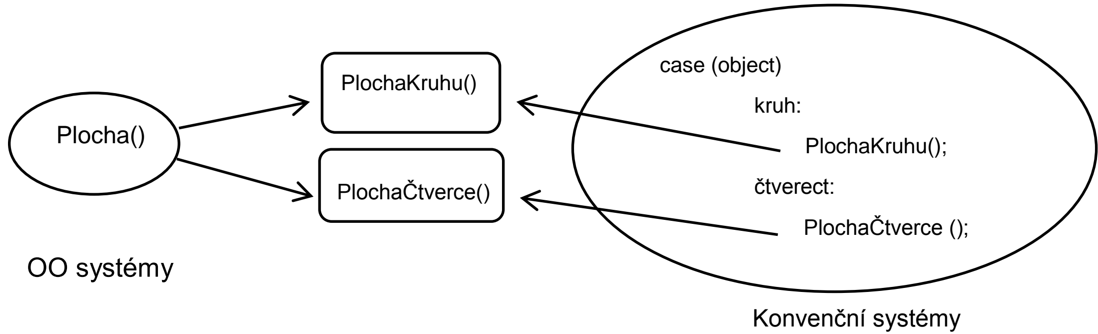
Objektový Typ popisuje společné struktury (atributy a metody) množiny objektů a vztahuje se více ke konceptuální fázi. *Třída* v sobě zahrnuje navíc i možnost vytvářet nové objekty, rušit instance sdružovat objekty do kontejnerů (extent, extenze třídy) a pracovat tam s nimi, vztahuje se více k implementaci. Objektově orientované jazyky nabízí tedy kromě atomických typů možnost vytvářet vlastní komponentní typy (structures), složené z několika obecných typů, dále různé kolekce typů (set, bag, list, array, dictionary) a referenčních typů – hodnota tohoto typu určuje umístění odkazovaného typu. Schopnosti odvozovat z existujících tříd (superclass) nové třídy(subclass), které ke svým speciálním atributům a metodám automaticky získají atributy a metody svého předka se nazývá *dědění*. Obecně může vzniknout hierarchie tříd i s více nadtřídami (jednoduché proti vícenásobnému dědění) pro jednu úroveň podtřídy.



Zapouzdření – princip viditelnosti - zpřístupnění objektu jen přes jeho rozhraní, ne jiným způsobem. Zlepšuje se tím zabezpečení dat (data jsou modifikovatelná jen veřejnými operacemi). Mezi výhody patří podpora modularity – rozsáhlý problém se rozdělí na menší izolované části s jasnou zodpovědností bez nadbytečných závislostí na ostatních částech aplikace a zvětšuje se znovupoužití kódu v nových aplikacích.



Metody mohou být redefinovány pro různé typy, tj. identická zprava v různém kontextu vyvolá různé reakce na různých objektech (late binding). Schopnost operací fungovat na více než jednom typu objektů se nazývá *polymorfizmus*. Příklad



ODMG 1.0 Standard (1993)

ODMG 2.0 Standard (1997)

ODMG 3.0 Standard (2000)

Výsledkem vývoje a potřeb různých aplikací je řada objektových modelů, lišících se často v důležitých koncepčních aspektech. Snaha o dosažení kompatibility čistě objektových systémů vedla ke schválení prvního standardu ODMG v roce 1993. ODMG definuje dva typy produktů :

- ODBMS (Object Database Management Systems) ukládají na disk přímo objekty
- ODMs (Objekt to Database Mapings) transformují objekty a ukládají je ve formě relací nebo XML dokumentů.

Průvodce studiem

Hlavní rysy objektově orientovaného přístupu:
Abstrakce – složitá komplexní realita může být reprezentována zjednodušenými konstrukty modelu.
Zapouzdření – znamená uzavření operací a dat dohromady v objektovém typu s přístupem přes veřejné rozhraní.
Dědičnost – představuje hierarchii objektových typů, kdy následník převezme část nebo všechna data a metody předka
Znovupoužitelnost – schopnost znovu využít objektový typ během návrhu nebo implementace systému.
Komunikace – objekty spolu komunikují prostřednictvím zpráv, které jsou určeny metodám příjemce.
Polymorfizmus – umožňuje psát kód čitelněji, když objekty dědí a modifikují stejnojmenné metody, metoda je aplikována na několik objektových typů.

3.5.2 Úvod do ODL

ODL (Object Definition Language) je standardizovaný jazyk pro definici struktury objektů v objektových databázových systémech. Vychází a rozšiřuje IDL (Interface Description Language) součást objektového standardu CORBA.. Podrobná syntaxe jazyka jde za rámec tohoto textu, ale pro ilustraci je uvedeno několik typických příkladů. Základní prvky jsou objekty (Olomouc, Petr Novák, …) a literály („Zelená 30“, „zedník“, …). Literály jsou jednoduché hodnoty, nemají ID, stav ani metody. Podobné objekty jsou sdruženy do tříd, jsou generovány pomocí konstruktorů. Když navrhujeme třídy pomocí ODL, popisujeme tři druhy složek:

- Stav - je určen

atributy – mohou být atomického, výčtového (seznam literálů) i strukturovaného typu

Např. 1. atomické atributy Attribute string povolání;

2. strukturované atributy Attribute struct Adresa { string obec; string ulice; int číslo } adr1;

3. vícehodnotové atributy Attribute set<int> pokusy;

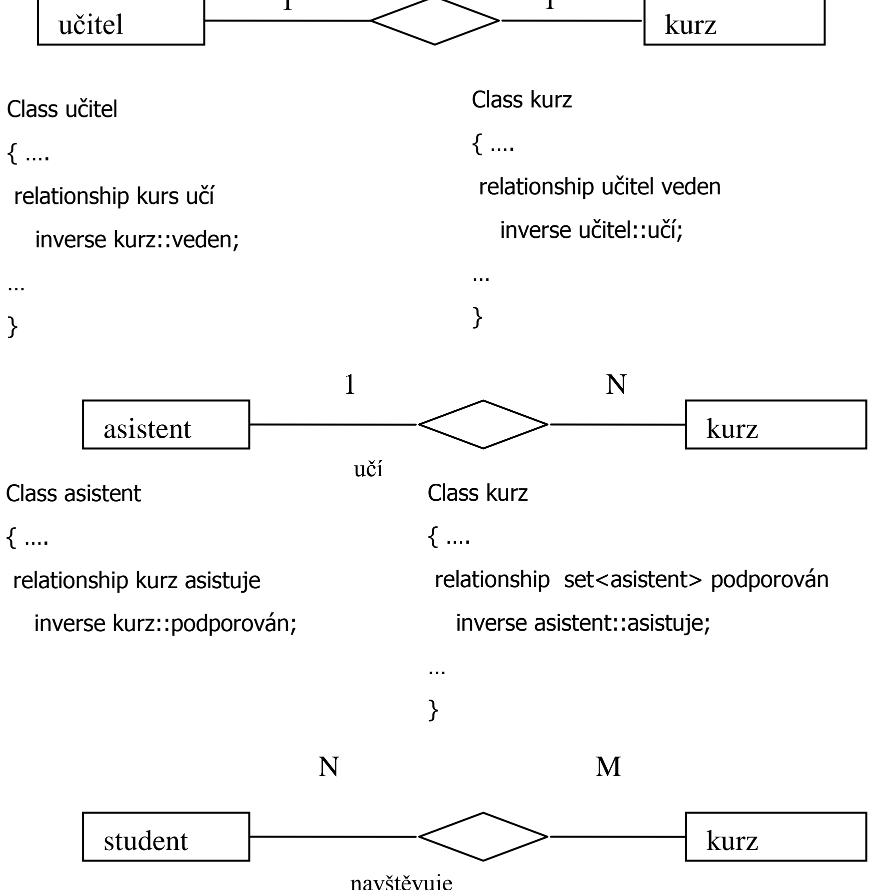
4. odvozené atributy int věk(); -- metoda

dalšími objekty

Hodnoty se mohou měnit v čase. Několik objektů může mít v jednom okamžiku stejné hodnoty atributů.

učí

Vztahy mezi objekty – s rozlišením kardinality a dalších vlastností vztahu, se zavedením inverzního vztahu pro řízení referenční integrity. Příklad vazby 1:1, 1:N a N:M




```
Class učitel
{ ....
 relationship kurs učí
    inverse kurz::veden;
...
}
```

```
Class kurz
{ ....
 relationship učitel veden
    inverse učitel::učí;
...
}
```

```
Class asistent
{ ....
 relationship kurz asistuje
    inverse kurz::podporován;
```

```
Class kurz
{ ....
 relationship  set<asistent> podporován
    inverse asistent::asistuje;
...
}
```

Hierarchie tříd sémanticky realizuje vztahy generalizace nebo opačně specializace. Při dědičnosti z více předků ODL neřeší případné konflikty. Příklad jednoduchého dědění

```
Class dílo
  (extent díla)
{ ....
  attribute string titul;
  attribute int  vydání;
...
}
```

```
Class román extends dílo
  { ....
    relationship  Set<hrdina> kladní;
    ...
  }
```

```
Class detektivka extends román
  { ....
    attribute string zbraň;
    ...
  }
```

- Metody – deklarace signatury se specifikací parametrů vstupních (in), typicky předávaných hodnotou, výstupních (out) a kombinovaných (inout), předávaných referencí a modifikovatelných. Nepovinně může následovat seznam vyjímek, které metoda umí ošetřit.

```
Class učitel
    (extent učitels)
{
...
Int praxe();
void plat(out m_plat, in měsíc)
raises(chyba);
}
```

Objekty a literály jsou kategorizovány podle svých typů nebo tříd, které mohou tvořit hierarchie. Typy v ODL můžeme rozdělit na základní:

1. Atomické – např. bolean, integer, fload, string, …
2. Třídy - např. učitel, kurz, …

Kombinací základních typů můžeme vytvářet

Structure N {T1 F1, T2 F2, …, Tn Fn}, kde Ti jsou typy a Fi jsou jména položek

nebo kolekce s různými vlastnostmi:

1. Set <T> je konečná neuspořádaná množina prvků typu T
2. Bag <T> je konečná neuspořádaná multimnožina prvků typu T (vícenásobný výskyt prvků)
3. List <T> je konečná uspořádaná množina prvků typu T
4. Array <T,I > je pole prvků typu T , jejichž počet je I
5. Dictionary <T,S > je konečná množina dvojic prvků klíčového typu T a typu S s vlastnostmi rychlého nalezení hodnoty S podle klíče T

Kolekce existují ve dvou verzích. V první verzi jsou jen hodnoty bez OID, v druhé jsou objekty i se svými metodami. Množina aktuálně uložených objektů z jedné třídy tvoří extent třídy.

3.6 Další databázové modely

Do praktického života databázových aplikací se stále častěji prosazují aplikace, pro které jsou dříve uvedené klasické modely z různých důvodů méně vhodné. Prosazují se nové přístupy popisu, formy uložení a přenosu dat v souvislosti například s internetem, kde se často používají semistrukturovaná data. Podrobnější popis těchto modelů jde za rámec tohoto textu, ale alespoň krátká charakteristika:

V semistrukturovaných datových modelech jsou data uložena ve formě grafu. Uzly grafu popisují strukturované objekty a hodnoty atributů a hrany zachycují vazby mezi objekty nebo objekty a atributy. Takové modely, například založené na XML dokumentech, se používají pro integraci databází napříč různými technologiemi a platformami, pro transformaci struktur dat a nové metody prezentace, hlavně v kontextu nových technologií na Internetu. Model obsahuje informace jak o své hierarchické struktuře, tak vlastní data. Strukturu dat lze explicitně formálně

popsat (DDT, XMLSchema) a tento popis použít pro kontrolu jednotlivých datových dokumentů. Do praktického použití se prosazuje dotazovací jazyk Xquery.

Shrnutí

Historicky první databázové modely jsou síťový a hierarchický, založené na souborech a vztazích mezi záznamy, které se implementují kruhovými seznamy, pomocí odkazů. Graf datové struktury se skládá z entitních typů a hran, které zobrazují vazby 1:1 nebo 1:N mezi entitními typy a jsou definovány pomocí C-množin – vlastník : člen nebo podobně otec : syn.
Relační model : Relace je při praktickém pohledu tabulka, kde sloupce představují atributy s přiřazením množiny přípustných hodnot ve formě datového typu nebo domény. Řádky tabulky tvoří n-tice a představují informace o jednom výskytu entity. Schéma relace tvoří její logické jméno společně s logickými jmény atributů a doménami. Schéma databáze je kolekce všech relačních schémat aplikace, instanci databáze tvoří aktuální uložená data. Konverze z ER diagramu do relačního modelu v prvé fázi probíhá tak, že entitě odpovídá tabulka, atributy entity převedeme na atomické položky a ty tvoří sloupce tabulky. U slabých entitních typů přidáme identifikační atributy z regulární entity, se kterou tvoří identifikační vazbu. Vazby mezi entitami se realizují pomocí atributů v jedné tabulce, korespondujících s primárními klíči v druhé tabulce nebo tabulkách, které figurují ve vztahu. Podobně se řeší transformace ISA hierarchie.
Objektový model: Vychází z ověřených zásad objektového konceptu, využívající zapouzdření, polymorfizmus a třídy. Pro formální popis schématu databáze se používá jazyk ODL, pomocí kterého definujeme třídy, jejich atributy, metody a vazby. Vztahy jsou pouze binární, mezi dvěma třídami, vyjádřené pojmenováním přímé i inverzní vazby v obou třídách. Kardinalita může být libovolná, což umožňuje použití různých typů kolekcí objektů. Každý objekt má své OID pro jednoznačnou identifikaci, které generuje systém a není uživatelsky modifikovatelné. Objekt může obsahovat i klíčovou položku.

Pojmy k zapamatování

- Bachmanův diagram, Entitní typ, C-množina
- Klíč identifikační, primární, kandidátní, sekundární, cizí
- Referenční integrita, doménové integritní omezení
- Objektový koncept, ODL

Kontrolní otázky

15. *Kam se transformují atributy z ER diagramu při transformaci do hierarchického databázového modelu?*
16. *Jak se implementuje vazba N:M v síťové modelu?*
17. *Jaká jsou integritní omezení v logických modelech dat?*
18. *Jaké vlastnosti musí splňovat kandidátní klíč?*
19. *Jaké jsou vlastnosti relací?*
20. *Jaké jsou kategorie relací v databázových systémech?*
21. *Jak se realizuje vazba mezi entitami v relačním modelu dat, jaká jsou transformační pravidla?*
22. *Jak se definují základní struktury objektů pomocí ODL?*
23. *Jaké vlastnosti mají další databázové modely?*

Úkoly k textu

Navrhněte ER diagram jednoduché databázové aplikace, obsahující vazby 1:N a N : M a vytvořte schéma databáze, definujte kandidátní, primární a cizí klíče, vyznačte referenční integritu. Vytvořte dané schéma databáze pro relační, hierarchický model dat .

4 Relační databázové a dotazovací jazyky

Studijní cíle: Po prostudování kapitoly by student měl charakterizovat a popsat základní dotazovací jazyky a jejich vztahy. Na příkladech vysvětlit operace relační algebry, kategorizovat operaci spojení, porovnat relační kalkuly formou zápisu dotazu pomocí jednoduchých výrazů. Student by měl zvládnou základy syntaxe jazyka SQL na základních příkazech pro definici dat a manipulaci s daty. Seznámit se s přístupem k dotazování pomocí QBE, DATALOG, OQL.

Klíčová slova: Operace relační algebry, n-ticový a doménový relační kalkul, bezpečný výraz, SQL, DDJ, DMJ, agregované funkce, QBE, DATALOG, OQL

Potřebný čas: 5 hodin

Obecné požadavky na dotazovací jazyk můžeme formulovat jako

- blízkost – výsledek dotazu musí být reprezentovatelný v konceptech datového modelu,
- kompletnost – jazyk musí zajišťovat operace datového modelu – např. u OOJ konstruktory, selektory složek, dereference, operace s kolekcemi, …
- ortogonalitu – možnost různě kombinovat a vnořovat operace.

Hlavní sílou relačního modelu je matematicky formalizovaná podpora jednoduchého dotazování. Jazyky SŘBD můžeme rozdělit podle různých kritérií, například na procedurální proti neprocedurálním, nebo podle úrovně na „čisté“(matematické, interní), které tvoří základ pro vyšší uživatelské dotazovací jazyky. Dva reprezentanti interních jazyků relačního modelu :

1. jazyky založené na *relační algebře*, kde jsou výběrové požadavky vyjádřeny jako posloupnost operací prováděných nad relacemi (tím je definován algoritmus vyhodnocení dotazu – procedurální jazyk), vhodné pro reprezentaci prováděcích rozvrhů při optimalizaci.
2. jazyky založené na *predikátovém kalkulu*, které požadavky dotazu zadávají jako predikát **P** charakterizující požadovanou relaci - {a, P(a)}. Jedná se o neprocedurální jazyk. Takové jazyky dále dělíme na
   - *n-ticové relační kalkuly*
   - *doménové relační kalkuly.*

Porozumění relační algebře a relačnímu kalkulu umožní pochopení dotazování v SQL

Jazyky vyšší úrovně, např.

SQL- postavený na n-ticovém relačním kalkulu a vybraných algebraických konstruktech

QBE : využívá doménové relační kalkuly

QUEL : využívá n-ticové relační kalkuly

4.1 Relační algebra

Relační algebra je důležitou podporou relačního modelu, představuje vyjadřovací sílu a sémantiku dotazu. Formálně je RA definována jako dvojice (R,O), kde nosičem R je množina relací a O je množina operací. Síla prostředku je dána tím, že nepracuje s jednotlivými n-ticemi, ale s celými relacemi. Operátory relační algebry se aplikují na relace, výsledkem jsou opět relace. V průběhu vývoje se ustálila skupina základních a odvozených(redundantních) operací – některé se již obvykle nezmiňují (podíl), jiné odvozené se využívají pro jednoduší

formulaci a efektivnější implementaci (spojení) Protože relace jsou množiny n-tic, můžeme operace relační algebry rozdělit na množinové nad relacemi s ekvivalentním záhlavím(sjednocení, průnik, rozdíl), množinovou bez omezení (kartézský součin) a speciální (projekce, selekce, spojení) . Ekvivalencí záhlaví rozumíme stejný počet atributů relací a existence vzájemně jednoznačného přiřazení atributů z jedné a druhé relace, omezené na dvojice s odpovídajícími doménami. Operace jsou popsány formálně i s grafickou ilustrací.

Základní množinové operace relační algebry pro relace R a S :

sjednocení relací ekvivalentního typu $R \cup S = \{x \mid x \in R \vee x \in S\}$,

ekvivalence je definována nad R, S i V :

R1↔S1↔V1; R2↔S2↔V2

R

R1	R2
Petr	5
Ivan	8

S

S1	S2
Ivan	8
Alena	16

V = R ∪ S

V1	V2
Alena	16
Petr	5
Ivan	8

SQL: SELECT R1 as V1, R2 as V2 FROM R as V UNION SELECT * FROM S

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

R

R1	R2
Petr	5
Ivan	8

S

S1	S2
Ivan	8
Alena	16

V = R ∩ S

V1	V2
Ivan	8

SQL: SELECT R1 as V1, R2 as V2 FROM R as V INTERSECT SELECT * FROM S nebo

SELECT R1 as V1, R2 as V2 FROM R as V JOIN S on R1=S1 AND R2=S2

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

R

R1	R2
Petr	5
Ivan	8

S

S1	S2
Ivan	8
Alena	16

V = R - S

V1	V2
Petr	5

SQL: SELECT R1 as V1, R2 as V2 FROM R as V MINUS SELECT * FROM S

kartézský součin relace R stupně m a relace S stupně n

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

R

R1	R2	R3
Petr	5	A
Ivan	8	B

S

S1	S2
Olomouc	1.9.2001
Brno	5.3.1995

V = R x S

VR1	VR2	VR3	VS1	VS2
Petr	5	A	Olomouc	1.9.2001
Petr	5	A	Brno	5.3.1995
Ivan	8	B	Olomouc	1.9.2001
Ivan	8	B	Brno	5.3.1995

SQL: SELECT * FROM R ,S nebo

SELECT * FROM R CROSS JOIN S

Speciální operace:

projekce relace R stupně n na atributy $A = \{A_{i1},...A_{im}\}$, kde $1 \le im \le n$, $ij \ne ik$ pro $j \ne k$

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

Jiné označení π_A (R)

R

R1	R2
Petr	5
Ivan	8

V = R[R2]

V1
5
8

SQL: SELECT R2 AS V1 FROM R AS V

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

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

Kde P je formule ve tvaru <atribut> = <atribut> | <konstanta>, spojené případně logickými operátory AND, OR, NOT, používající další relační operátory < , <= , > , >= , ≠.

Jiné značení σ_P (R)

R

R1	R2
Petr	5
Ivan	8

V = R(R2 > 6)

V1	V2
Ivan	8

SQL: SELECT * FROM R WHERE R2 > 6

(*vnitřní*) *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.

Jiné značení $R \bowtie S$

Nejčastěji se používá spojení relací *na rovnost* s použitím 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.

R

R1	R2	R3
Petr	5	Brno
Ivan	8	Olomouc

S

S1	S2
Olomouc	1.9.2001
Ostrava	5.3.1995

V = R [R3 = S1] S

VR1	VR2	VR3	VS2
Ivan	8	Olomouc	1.9.2001

SQL: SELECT * FROM R ,S WHERE R3 = S1 nebo

SELECT * FROM R INNER JOIN S ON R3 = S1

- *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(nebo sémantikou) jako atributy z B a C jsou ostatní atributy z A i B; ze součinu RxS se vyberou ty prvky, které mají stejné hodnoty(spojení na rovnost) na maximální množině společných atributů.

Nedefinované hodnoty představují neznámá, nebo neexistující data. Pokud operace spojení využívá i prázdné hodnoty (NULL) jedná se o *vnější spojení* . Motivací je snaha ve výsledku operace získat i n-tice, které s ničím spojit nelze. Podle lokalizace těchto n-tic definujeme *levé*

vnější spojení – ve výsledku jsou všechny n-tice levé vstupní relace (v části se záhlavím, odpovídající levé relaci), analogicky *pravé vnější spojení* poskytne všechny n-tice pravé vstupní relace a *symetrické(úplné) vnější spojení* vznikne sjednocením levého a pravého vnějšího spojení. Ve výsledku operace je podmnožina n-tic, odpovídající vnitřnímu spojení a ve zbylých n-ticích jsou na nedefinovaných místech hodnoty NULL. Pro levé spojení platí symbolicky

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

R

R1	R2	R3
Petr	5	Brno
Ivan	8	Olomouc

S

S1	S2
Olomouc	1.9.2001
Ostrava	5.3.1995

$V = R*_{Full} S$

VR1	VR2	VR3	VS1	VS2
Petr	5	Brno	NULL	NULL
Ivan	8	Olomouc	Olomouc	1.9.2001
NULL	NULL	NULL	Ostrava	5.3.1995

SQL: SELECT R1 AS VR1, R2 AS VR2, R3 AS VR3, S1 AS VS1, S2 AS VS2

FROM R AS V FULL OUTER JOIN S ON R3 = S1

Hlavně v distribuovaných systémech se setkáme s odvozenou operací *polospojení (levé a pravé)*, pomocí níž jsou omezeny prvky jedné relace podle prvků druhé relace v závislosti na spojovací podmínce. Například levé polospojení <* je definováno

$R <* S = (R*S)[A_R]$

R

R1	R2	R3
Petr	5	Brno
Ivan	8	Olomouc

S

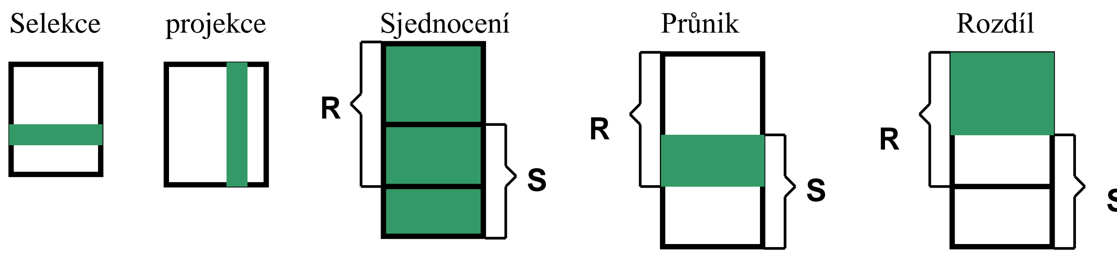
S1	S2
Olomouc	1.9.2001
Ostrava	5.3.1995

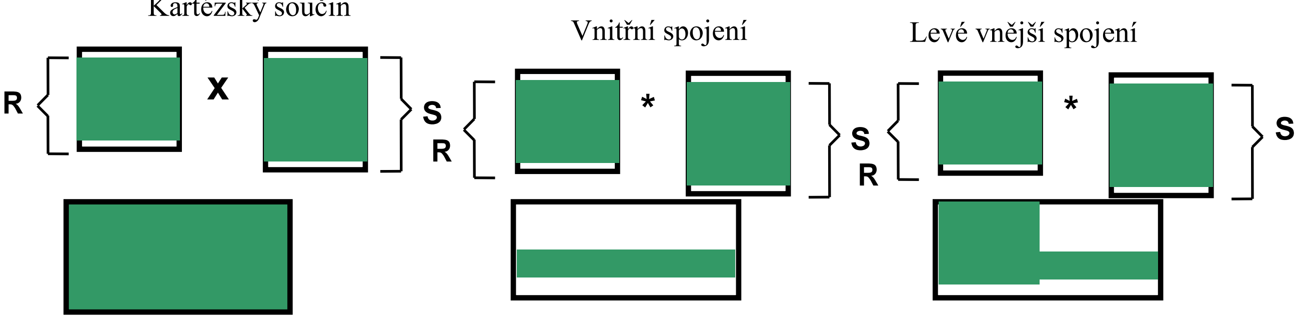
V = R <* [R3 = S1] S

VR1	VR2	VR3
Ivan	8	Olomouc

SQL: SELECT R.* FROM R ,S WHERE R3 = S1 nebo

SELECT R.* FROM R INNER JOIN S ON R3 = S1





Operace relační algebry

Do relační algebry se zahrnuje i operace *přejmenování* (atributu, relace), využitelná například - spojení jedné relace sama se sebou, při stejných jménech atributů v různých relacích v jednom výrazu, atd. .

Další, praxí vynucené, rozšíření relační algebry se týká zvýšení síly dotazování o aritmetické operace (agregační funkce), rozšíření o procedurální prvky, aby bylo možné pracovat s rekurzí.

4.2 N-ticový relační kalkul

Jako jazyk pro výběr informací z relační databáze lze využít jazyk logiky, predikátového kalkulu 1. řádu. 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.

Abecedu n-ticového relačního kalkulu tvoří :

- atomické konstanty (hodnoty atributů), např. ’Novák’
- 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), odkazy 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.

Výraz n-ticového relačního kalkulu je výraz tvaru

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

kde x je jediná volná proměnná ve formuli F.

Výraz n-ticového relačního kalkulu určuje relaci tvořenou všemi možnými hodnotami proměnné x, které splňují formuli F(x). x definuje seznam komponent proměnných, které definují schéma výstupní relace. Je to buď již definovaná entita, seznam entit nebo seznam komponent volných proměnných.

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 | R(x) OR S(x)

$R \cap S$ => x | R(x) AND S(x)

R - S => x | R(x) AND NOT S(x)

R x S => x, y | R(x) AND S(y)

$R[a_1,a_2,...,a_k]$ => $x.a_1, x.a_2,..., x.a_k$ | R(x)

R(P) => x | R(x) AND P

R[A*B]S => x, y | R(x) AND S(y) AND x.A * y.B

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

t | NOT R(t)

Výrazy relačního kalkulu se proto omezují jen na tzv. bezpečné výrazy, které definování relací nekonečných nedovolují. *dom*(*P*) označme doménu formule P, tedy množinu všech hodnot explicitně se vyskytujících v P nebo v relacích, které jsou součástí P. Bezpečný výraz t | P(t) musí splňovat podmínku, že všechny hodnoty, které se vyskytují v n-ticích výrazu P jsou z *dom*(*P*) .

Příklady jednoduchých dotazů:

Seznam pracovníků, jejichž plat je menší než 15 000 Kč.

{P | Pracovníci (P) $\wedge$P.plat < 15 000 }

Seznam jmen a příjmení pracovníků, kteří sedí v místnosti s plochou větší, než 20 m^2.

{P.jméno, P.příjmení | Pracovníci (P) $\wedge(\exists$ M)(Místnost(M) $\wedge$ (P.místnost = M.číslo) $\wedge$ (M.plocha > 20) }

4.3 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 abeceda kalkulu:

- 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.

Výraz doménového relačního kalkulu je výraz tvaru

x1 x2 ...xn | 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.

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

Věta o ekvivalenci

Ke každému výrazu relační algebry existuje ekvivalentní bezpečný výraz relačního kalkulu a opačně, oběma prostředky je možno vyjádřit stejné třídy dotazů.

Příklady jednoduchých dotazů:

Seznam pracovníků, jejichž plat je menší než 15 000 Kč.

{pjméno, ppříjmení, pplat, pmístnost | (Pracovníci (pjméno, ppříjmení, pplat, pmístnost)) ∧ (pplat < 15 000) }

Seznam jmen a příjmení pracovníků, kteří sedí v místnosti s plochou větší, než 20 m^2.

{pjméno, ppříjmení | (∃pplat, ∃pmístnost) (Pracovníci (pjméno, ppříjmení, pplat, pmístnost)) ∧ (∃mčíslo, ∃mplocha) (Místnost(mčíslo,mplocha)) ∧ (pmístnost = mčíslo) ∧ (mplocha > 20) }

4.4 Datalog

Relační algebra umožňuje vyjádřit mnoho užitečných operací, ale existují výpočty, založené na zpracování rekurzívních sekvencí podobných výrazů, které nemohou být vyjádřeny. Reakcí je použití logicky orientovaného datového modelu v deduktivních databázích, s typickým reprezentantem v systému Datalog, který vychází . Datalog je založen na logickém paradigmatu programování. Skládá se z části extenzionální, která obsahuje základní data ve tvaru tvrzení, tj. logických formulí a intenzionální, která osahuje odvozovací pravidla, pomocí nichž se definují virtuální relace. Pravidla mají na levé straně *hlavu* a pravou tvoří *tělo*. Virtuální relace se tvoří pomocí pravidel se stejnou hlavou, pomocí odkazů na další pravidla i rekurzívně na sebe. Zápis tvrzení a pravidel má formu:

Lx :- Ly, …, Lz,

kde L jsou atomické formule – literály,ve tvaru P(t1, …, tk). P je predikátový symbol, t je proměnná nebo konstanta. Základní literály obsahují jen konstanty a definují tvrzení, tedy data databáze. Pro pravidla, podobně jako v relačním kalkulu, se zavádí omezující podmínky, aby virtuální relace byly konečné a obsahovaly pouze tvrzení.

Příklad rekurze   Rozvíjí (x,y) :- PřímoVychází(x,y)

Rozvíjí (x,y) :- PřímoVychází(x,z) AND Rozvíjí (z,y)

Pro formulaci dotazu v DATALOGU musíme znát sémantiku programu. Výsledkem programu je materializace virtuálních relací pomocí odvozovacího stromu. Základní tvrzení jsou v listech,

vnitřní uzly jsou mezivýsledky aplikace pravidel. Princip dotazování ukážeme na jednoduchém příkladu:

Učitel(U1, Petr, Novák, 21000, informatika)

Učitel(U2, Alena, Malá, 19000, čeština)

…

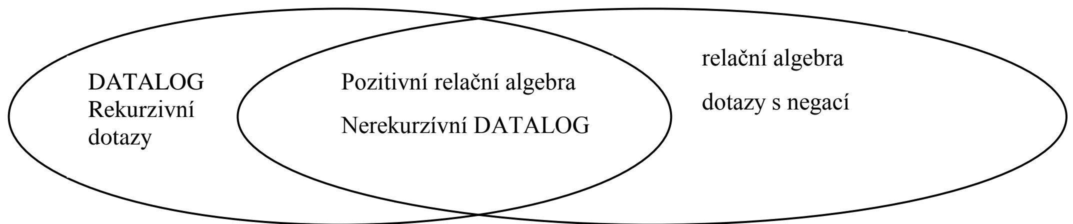
Virtuální relace :

DobbřePlacenýUčitel(jméno, příjmení) := Učitel(jméno, příjmení, plat, předmět), plat > 20000

reprezentuje dotaz :

?? := Učitel(jméno, příjmení, plat, předmět), plat > 20000, kterému vyhovují data <Petr, Novák>

Porovnání síly DATALOGu a relační algebry ukazuje následující obrázek.



Obr. 11 Vztah vyjadřovací síly DATALOGu a relační algegry

4.5 Jazyk SQL

Jazyk SQL byl původně navržen v roce 1975 u firmy IBM jako dotazovací jazyk (původní název Sequel v systémech R). V roce 1986 definován standard ANSI (American National Standard Institute), standard ISO – SQL/86, v roce 1989 přidán integritní dodatek –SQL/89. Přelomovým rokem byl rok 1992, standard SQL/92 se třemi úrovněmi souladu – Entry, intermediate, full. Další standardy (SQL3, SQL1999) evolučně směřují k relačně-objektovým databázím a různým rozšířením. Prakticky existují firemní dialekty SQL/92 s rozšířením.

Jazyk obsahuje příkazy pro definici databáze a vytvoření jejích objektů a integritních omezení, interaktivní jazyk pro manipulaci s daty – včetně komplexních dotazů, řízení přístupových práv, řízení transakcí, které z SQL dělají mnohem silnější prostředek, než jen dotazovací jazyk.

4.5.1 Příkazy pro definici dat

Následující přehled vybraných příkazů nemůže nahradit obsáhlé firemní manuály nebo normy, cílem je naznačit syntaktická pravidla nejpoužívanějších tvarů, bez nároků na kompletnost. V úplná syntaxe příkazů je pro svou obsáhlost pro začínajícího uživatele nepřehledná.

Administrátor nebo uživatel má podle stupně oprávnění možnost definovat a vytvořit řadu databázových objektů – od uživatelsky nejběžnějších až po objekty, které definuje administrátor – např. TABLE, VIEW, INDEX, PROCEDURE, FUNCTION, TRIGGER, …, DATABASE, USER, … . Skupiny příkazů mají společnou syntaxi pro vytvoření objektu – CREATE objekt…, zrušení objektu – DROP objekt …, modifikaci objektu – ALTER objekt … .

Náznak základní syntaxe definice tabulky (obsahuje zjednodušení) :

CREATE TABLE [IF NOT EXISTS] tabulka_jméno

(sloupec_deklarace, [sloupec_deklarace]*, [TabIntegritníOmezení_deklarace], ...)

sloupec_deklarace ::= sloupec_jméno type [ŘádIntegritníOmezení_deklarace], ...

ŘádIntegritníOmezení_deklarace ::= [DEFAULT výraz] [NULL | NOT NULL] [PRIMARY KEY | UNIQUE] [CHECK (výraz)] [REFERENCES tabulka_jméno [(sloupec_jméno)] ...]]

type ::= INTEGER | SMALLINT | NUMERIC | DECIMAL | FLOAT | REAL | CHAR | VARCHAR | DATE | TIME | BOOLEAN | …

TabIntegritníOmezení_deklarace :: = [CONSTRAINT IOomezení_jméno]

[PRIMARY KEY (sloupec_jméno1, sloupec_jméno2, ...)] |

[FOREIGN KEY (sloupec_jméno1, sloupec_jméno2, ...) REFERENCES tabulka_jméno [(sloupec_jméno1, sloupec_jméno2, ...)] [ON UPDATE t_akce] [ON DELETE t_akce]] |

[UNIQUE (sloupec_jméno1, sloupec_jméno2, ...)] | [CHECK (výraz)]

t_akce :: = NO ACTION | SET NULL | SET DEFAULT | CASCADE

Tabulka má logické jméno a nejméně jeden sloupec. Každý sloupec je logicky pojmenován, má definovaný jednoduchý datový typ (případně zúžený IO check()) a případnou kombinaci řádkových integritních omezení. Po definici sloupců tabulky může následovat definice tabulkových IO pro jedno i více složek, sloupců.

Př.:

```
CREATE TABLE pracovník (
      IDpra char (10)  NOT NULL ,
      Příjmení char (20)  NULL ,
      Jméno char (15)  NULL ,
      Číslo_místnosti char (10)  NULL
      )
```

Některé modifikace struktury tabulky(tento příkaz se často v některých frázích liší na různých firemních dialektech):

ALTER TABLE tabulka_jméno

[ADD [[sloupec_jméno] sloupec_deklarace] | [CONSTRAINT omezení_jméno IntegritníOmezení_deklarace]]|

[ALTER COLUMN sloupec_deklarace_nová]|

[DROP [CONSTRAINT omezení_jméno]| TabIntegritníOmezení | COLUMN sloupec_jméno]

Př.:

```
ALTER TABLE pracovník ADD
      CONSTRAINT PK_pracovník PRIMARY KEY  CLUSTERED
      (
            IDpra
      )
ALTER TABLE pracovník ADD
      CONSTRAINT FK_pracovník_místnost FOREIGN KEY
      (
            Číslo_místnosti
      ) REFERENCES místnost (
            Číslo_místnosti
      ) ON UPDATE CASCADE
```

Zrušení tabulky včetně definice

DROP TABLE tabulka_jméno

Př.

```
if exists (select * from sysobjects where id = object_id(N'pracovník')
and OBJECTPROPERTY(id, N'IsUserTable') = 1)
drop table pracovník
```

Vytvoření a rušení indexu :

CREATE [UNIQUE] INDEX index_jméno ON tabulka_jméno (seznam_sloupců)

DROP INDEX index_jméno

Klauzule UNIQUE znamená požadavek jednoznačnosti indexu

4.5.2 Příkazy 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 tabulka_jméno [(sloupec_jméno1, sloupec_jméno2, ...)]   VALUES (výraz_1, výraz_2, ...)

Př.

```
insert into pracovník values ('A','Novák','Petr','10')
```

Pomocí příkazu INSERT je možno také naplňovat řádky tabulky hod-notami 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.

INSERT INTO tabulka_jméno [(sloupec_jméno1, sloupec_jméno2, ...)]

SELECT ...

Změny hodnot v řádcích tabulky

UPDATE tabulka_jméno   SET sloupec_jméno1 = výraz1, sloupec_jméno2 = výraz2, ...

[WHERE výraz]

Př.

```
update pracovník set Příjmení = 'Nováková' where IDpra like 'B%'
```

Rušení řádků tabulky

DELETE [FROM] tabulka_jméno   [WHERE výraz]

Př.

```
delete pracovník where IDpra like 'A%'
```

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 tvar příkazu je

SELECT [DISTINCT | ALL]

sloupec_výraz1, sloupec_výraz2, ...

[FROM from_clause]

[WHERE where_výraz]
[GROUP BY výraz1, výraz2, ...]
[HAVING having_výraz]
[ORDER BY order_sloupec_výraz1, order_sloupec_výraz2, ...]

sloupec_výraz ::= s_výraz [AS] [sloupec_alias]
s_výraz ::= *| seznam_sloupců | …

from_clause ::= select_tabulka1, select_tabulka2, ...
from_clause ::= select_tabulka1 LEFT [OUTER] JOIN select_tabulka2 ON výraz ...
from_clause ::= select_tabulka1 RIGHT [OUTER] JOIN select_tabulka2 ON výraz ...
from_clause ::= select_tabulka1 [INNER] JOIN select_tabulka2 ...

select_tabulka ::= tabulka_jméno [AS] [tabulka_alias]
select_tabulka ::= (sub_select_statement) [AS] [tabulka_alias]

order_sloupec_výraz ::= výraz [ASC | DESC]
Použití znaku * místo seznamu sloupců znamená výpis všech sloupců tabulky.

Příkaz 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 relačního kalkulu

$x.A_1, x.A_2, \ldots, x.A_k \mid R(x)$ AND podm

Jednoznačnost prvků výsledné relace nezajišťuje jazyk SQL automaticky, ale musí se zadat v příkazu klauzulí DISTINCT.

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

konstanty, jména sloupců,

relační operátory : = <> < <= > >=

logické operátory : NOT AND OR

další operátory : BETWEEN dolní mez AND horní mez
IN(seznam_prvků_množiny)
IS NULL | NOT NULL
LIKE vzor ... pro porovnání řetězců podobně jako u hvězdičkové konvence :
% ... odpovídá skupině znaků
_ ... podtržítko zastupuje jeden znak

Př.

```
SELECT     pracovník.*, projektant.Obor AS [obor projektanta],
místnost.Plocha AS [plocha místnost]
FROM         pracovník INNER JOIN
             projektant ON pracovník.IDpra = projektant.IDpra INNER
           JOIN
                      místnost ON pracovník.Číslo_místnosti =
místnost.Číslo_místnosti
where pracovník.příjmení like 'D%'
order by pracovník.příjmení, pracovník.jméno
```

Setřídění výsledných řádků podle třídicího klíče, ne podle pořadí uložení v souboru zajistí klauzule ORDER BY. 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 variantně klasicky i 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 oddělením tečkou(tečková notace).

SELECT {seznam_sloupců | *}

FROM seznam_tabulek

[WHERE podm_spojení]

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.

FROM tabulka_jméno1 [AS] P1, tabulka_jméno2 [AS] P2, . . .

4.5.3 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, zapíšeme ho po mezeře za výrazem. Pokud se nadpis skládá z více slov, uzavírá se do úvozovek.

Dále jazyk SQL používá funkce

aritmetické: POWER(číslo,mocnitel)

ROUND(číslo,poč_des_míst)

ABS(číslo)

SIGN(číslo)

SQRT(číslo)

SIN(číslo) …

řetězcové: LEN (řetězec)

SUBSTRING(řetězec,pozice,délka) výběr podřetězce

UPPER(řetězec)

LOWER(řetězec)

LTRIM(řetězec,množ_znaků) vypustí zleva

RTRIM(řetězec,množ_znaků) vypustí zprava …

datumové a časové:	DATEDIFF (částdatum , stdatum , koDatum) …	
agregované:	AVG ({[DISTINCT] sez_výr|*})	průměr
	SUM ({[DISTINCT] sez_výr|*})	součet
	MIN ({[DISTINCT] sez_výr|*})	minimum
	MAX ({[DISTINCT] sez_výr|*})	maximum
	COUNT({[DISTINCT] sez_výr|*})	počet

Př.

```
select count(*) as [celkový počet pracovníků] from pracovník
```

4.5.4 Agregační funkce se skupinou dat

Tabulku můžeme uspořádat tak, že vzniknou skupiny řádků se stejnou hodnotou třídicího klíče. Také pro tyto skupiny můžeme provádět operace (částečné počty, součty ap.). Vytvoření skupin se provede klauzulí GROUP BY klíč

SELECT {seznam_výrazů | *}

FROM seznam_tabulek

GROUP BY seznam_sloupců

Př.

```
select count(*) as [počet pracovníků v místnosti],Číslo_místnosti
as [číslo místnosti] from pracovník group by Číslo_místnosti
```

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 sez-výr

FROM sez_tab

[WHERE podm_pro_řádek]

[GROUP BY sez-klíčů

[HAVING podm_pro_skup]]

Př.

```
select count(*) as [počet pracovníků v místnosti],Číslo_místnosti
as [číslo místnosti] from pracovník group by Číslo_místnosti
having Číslo_místnosti > 10
```

Syntaxe příkazu SELECT pro množinové operace: např.

SELECT … UNION [ALL] SELECT …

SELECT … INTERSECT SELECT …

SELECT … MINUS SELECT …

Př.

```
select * from pracovník1
union
select * from pracovník2
```

4.5.5 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)
- pokud je výsledkem poddotazu n-tice hodnot (relace o 1 řádku), pak s relačními operátory = a <> místo n-tice hodnot:
  výraz rel_oper (příkaz SELECT)
- je-li výsledkem poddotazu množina hodnot (relace o 1 sloupci):
  výraz [NOT] IN (příkaz SELECT)

  výraz rel_oper {[ANY | ALL]} (příkaz SELECT)

  kde pro ANY je minimální a ALL maximální prvek výsledné množiny.

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

Př.

```
select pracovník.příjmení from pracovník where
pracovník.Číslo_místnosti in (select Číslo_místnosti from
místnost where plocha > 100)
```

4.5.6 Pohledy

Pohled představuje virtuální tabulku - tabulku přímo v databázi neexistující, ale definovatelnou některým příkazem 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 …
```

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.

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

Př.

```
CREATE VIEW Projektanti_úkoly

AS

SELECT     TOP 100 PERCENT projektant.*,
Projektant_řeší_úkol.Termín AS [Termín projektanta], úkol.*

FROM        projektant INNER JOIN

                      Projektant_řeší_úkol ON projektant.IDpra =
Projektant_řeší_úkol.IDpra INNER JOIN

                      úkol ON Projektant_řeší_úkol.IDúkol =
úkol.IDúkol

ORDER BY projektant.IDpra
```

4.5.7 Další možnosti SQL

Již jsme uvedli, že SQL není jen dotazovací jazyk, ale obsahuje i příkazy další. 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 pak reprezentuje mocný příkaz SELECT.

SQL zahrnuje také příkazy, sloužící správě databáze pro přidělování přístupových práv na různé úrovni různým uživatelům:

GRANT privileges ON database_objekt TO (PUBLIC | user_list)

[WITH GRANT OPTION]

Odebrání práv:

REVOKE privilegia_seznam ON database_object

FROM (PUBLIC | user_list)

Zakázání práv:

DENY privilegia_seznam ON database_object

TO (PUBLIC | user_list)

privilegia_seznam ::= privilegia1, privilegia2, ...

privilegia::= ALL [PRIVILEGES] | SELECT | INSERT | UPDATE |

DELETE | …

database_objekt ::= [TABLE] tabulka_jméno | SCHEMA schéma_jméno

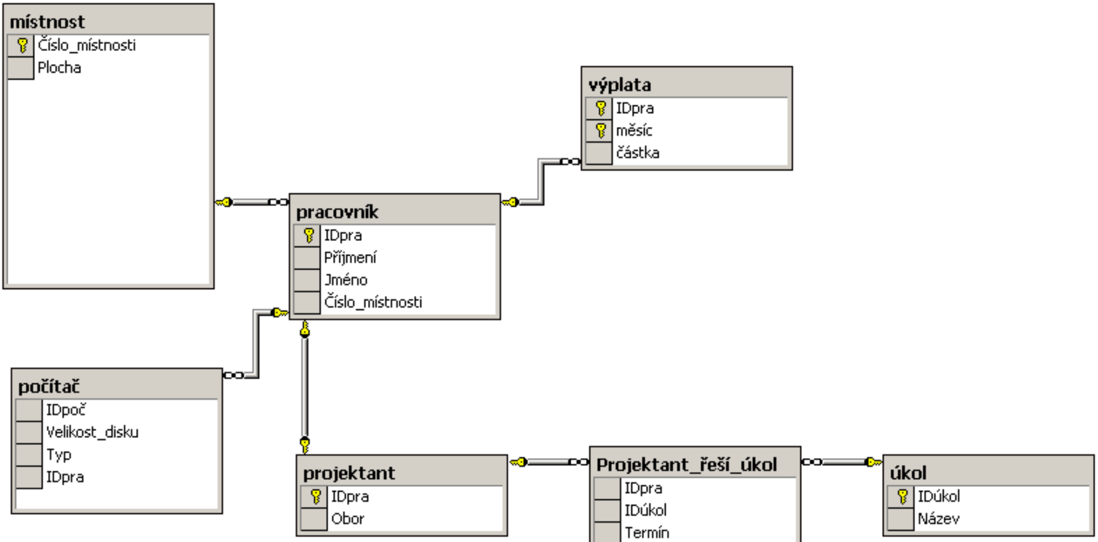
seznam_uživatelů ::= PUBLIC | uživatel1, uživatel 2, ...

Příkazy pro řízení transakcí,

BEGIN TRANSACTION, COMMIT, ROLLBACK

pro záznam některých IO, pro vytváření hierarchických struktur dat, pro práci se systémovým katalogem ap..

Příklad ukazující databázové schéma z ER diagramu předešlé kapitoly



Obr. 12 Schéma relací na SQL Serveru

Příklad skriptu

```
CREATE TABLE místnost (
	Číslo_místnosti char (10)  CONSTRAINT PK_místnost PRIMARY KEY  CLUSTERED ,
	Plocha numeric(18, 0) NULL
)
GO
CREATE TABLE pracovník (
	IDpra char (10)  CONSTRAINT PK_pracovník PRIMARY KEY  CLUSTERED ,
	Příjmení char (20)  NULL ,
	Jméno char (15)  NULL ,
	Číslo_místnosti char (10)  NULL,
	CONSTRAINT FK_pracovník_místnost FOREIGN KEY
	(
		Číslo_místnosti
	) REFERENCES místnost (
		Číslo_místnosti
	) ON UPDATE CASCADE
)
GO
CREATE TABLE počítač (
	IDpoč char (10)  NULL ,
	Velikost_disku char (10)  NULL ,
	Typ char (15)  NULL ,
	IDpra char (10)  NOT NULL,
	CONSTRAINT FK_počítač_pracovník FOREIGN KEY
	(
		IDpra
	) REFERENCES pracovník (
		IDpra
	) ON UPDATE CASCADE
)
GO
CREATE TABLE projektant (
	IDpra char (10)  NOT NULL CONSTRAINT PK_projektant PRIMARY KEY  CLUSTERED,
	Obor char (20)  NULL,
	CONSTRAINT FK_projektant_pracovník FOREIGN KEY
	(
		IDpra
	) REFERENCES pracovník (
```

```
                IDpra
        ) ON DELETE CASCADE  ON UPDATE CASCADE
)
GO
CREATE TABLE výplata (
        IDpra char (10)  NOT NULL ,
        měsíc char (10)  NOT NULL ,
        částka char (10)  NULL,
        CONSTRAINT PK_výplata PRIMARY KEY  CLUSTERED
        (
                IDpra,
                měsíc
        ),
        CONSTRAINT FK_výplata_pracovník FOREIGN KEY
        (
                IDpra
        ) REFERENCES pracovník (
                IDpra
        ) ON DELETE CASCADE
)
GO
CREATE TABLE úkol (
        IDúkol char (10)  NOT NULL ,
        Název char (20)  NULL,
CONSTRAINT PK_úkol PRIMARY KEY  CLUSTERED
        (
                IDúkol
        )
)
GO
CREATE TABLE Projektant_řeší_úkol (
        IDpra char (10)  NOT NULL,
        IDúkol char (10) NOT NULL ,
        Termín datetime NULL ,
        CONSTRAINT PK_Projektant_řeší_úkol PRIMARY KEY  CLUSTERED
        (
                IDpra,
                IDúkol
        ),
        CONSTRAINT FK_Projektant_řeší_úkol_projektant FOREIGN KEY
        (
                IDpra
        ) REFERENCES projektant (
                IDpra
        ),
        CONSTRAINT FK_Projektant_řeší_úkol_úkol FOREIGN KEY
        (
                IDúkol
        ) REFERENCES úkol (
                IDúkol
        )
)
GO
CREATE VIEW Projektanti_úkoly
AS
SELECT TOP 100 PERCENT projektant.*, Projektant_řeší_úkol.Termín AS [Termín
projektanta], úkol.*
FROM         projektant INNER JOIN                  Projektant_řeší_úkol ON
projektant.IDpra = Projektant_řeší_úkol.IDpra INNER JOIN
                      úkol ON Projektant_řeší_úkol.IDúkol = úkol.IDúkol
ORDER BY projektant.IDpra
GO
CREATE VIEW Projektanti_místnost
```

```
AS
SELECT     pracovník.*, projektant.Obor AS [obor projektanta], místnost.Plocha AS
[plocha místnost]
FROM         pracovník INNER JOIN
                      projektant ON pracovník.IDpra = projektant.IDpra INNER JOIN
                      místnost ON pracovník.Číslo_místnosti = místnost.Číslo_místnosti

GO
insert into místnost values ('10',102)
insert into místnost values ('20',24)

insert into pracovník values ('A','Novák','Petr','10')
insert into pracovník values ('B','Dlouhá','Aneška','10')
insert into pracovník values ('C','Vít','Viktor','20')

insert into projektant values ('A','stavební')
insert into projektant values ('B','stavební')
insert into projektant values ('C','stavební')

insert into úkol values ('1','škola')
insert into úkol values ('2','radnice')

insert into Projektant_řeší_úkol values ('A','1','1/3/2004')
insert into Projektant_řeší_úkol values ('A','2','2/3/2004')
insert into Projektant_řeší_úkol values ('B','1','1/3/2004')
insert into Projektant_řeší_úkol values ('C','2','1/3/2004')

insert into počítač values ('1001','160G','aa1','A')
insert into počítač values ('1002','160G','aa1','B')
insert into počítač values ('1003','80G','aa0','C')

select * from Projektanti_úkoly
select * from Projektanti_místnost

drop VIEW Projektanti_úkoly
drop VIEW Projektanti_místnost
drop TABLE Projektant_řeší_úkol
drop TABLE úkol
drop TABLE výplata
drop TABLE projektant
drop TABLE počítač
drop TABLE pracovník
drop TABLE místnost
```

4.6 Jazyk QBE

Jazyk QBE (Query By Example) byl původně vytvořen v Yorktown Heights firmou IBM (Zloof, 1975) pro pohodlné zadávání výběrových podmínek pro naivní uživatele. Vytvořil se z něj standard, užívaný v modifikacích u řady SŘBD. Mezi nejznámější implementace patří Paradox. Je to grafický dotazovací jazyk, založený na doménovém relačním kalkulu. Základem je voudimenzionální syntaxe. Dotazy jsou vyjadřovány interaktivně pomocí příkladů (odtud název jazyka). Tabulky, z nichž se mají informace čerpat, se formou prázdných tabulkových schémat zobrazují na obrazovce. Dotaz se zapíše tak, že do příslušných sloupců prázdné tabulky se vypíší ty hodnoty, které se ve sloupcích hledají. Duplicity jsou většinou implicitně odstraněny.

Výpis všech sloupců se zadá znakem P. (print) pod jméno relace.

Osoba.db	Číslo osoby	jméno	adresa

P.			

Dotaz v DRK : {č,j,a | osoba(číslo_osoby: c, jméno: j, adresa: a)}

Projekce se zapíše znakem P. do příslušného sloupce a vyjádřením proměnné (začíná podtržítkem) jako příkladu, např. P._Novák (odtud název jazyka)

Osoba.db	Číslo osoby	jméno	adresa
		P._Novák	

Dotaz v DRK : {j | osoba(jméno: j)}

Protože _Novák je proměnná, má sejný účinek např. _X.

Osoba.db	Číslo osoby	jméno	adresa
		P._X	P._Y

Dotaz v DRK : {č,j,a | osoba(číslo_osoby: c, jméno: j, adresa: a)}

Selekce se zapíše hledanými hodnotami ve sloupcích s případným použitím relačního operátoru.

Osoba.db	Číslo osoby	jméno	adresa
		P._X	Olomouc
		P._Y	Brno

Dotaz v DRK : {j,a | osoba(jméno: j, adresa: a) AND (a = %Olomouc% OR a = %Brno%) }

Osoba.db	Číslo osoby	jméno	adresa
		P._X	Olomouc
		_X	Brno

Dotaz v DRK : {j,a | osoba(jméno: j, adresa: a) AND (a = %Olomouc% AND a = %Brno%) }

Dále existují možnosti formulace složitějších podmínek, spojení(i vnější) více tabulek, třídění, výpočtu agregovaných hodnot, nedefinovaných hodnot, pohledů ap. Jsou podporovány další příkazy DMJ ekvivalent INSERT, DELETE, UPDATE a DDJ - CREATE TABLE(INDEX), , DROP TABLE (INDEX). To vše dělá jazyk QBE podobně silným prostředkem, jako je jazyk SQL.

QBE neumožňuje hnízdění jako SQL.

4.7 Úvod do OQL

OQL je dotazovací jazyk, který je standardem pro OOSŘBD, část ODMG standardu. Důvody pro jeho vznik jsou například pro uživatele v jednodušším neprocedurálním formulováním interaktivních ad hoc dotazů, zjednodušeném programování aplikací, možnosti optimalizace, nezávislost na fyzických datech, použití trigerů a integritních omezení a hlavně v možnosti odstranění nevýhody klasických relačních SQL – což je neschopnost vypočítat libovolnou vyčíslitelnou funkci, tj. nejsou výpočetně úplné. Problém síly jazyka se řeší začleněním SQL do procedurálního jazyka, který rozšiřuje sílu dotazu, ale vytvoří se sémanticky i strukturálně nesourodý prostředek (impedance mismatch). OOSŘBD nabízí možnost skloubit výpočetní úplnost a homogenitu konstruktů. Manipulačním jazykem je často Smalltalk, C++ nebo Java. Přesto zůstávají některé principiální problémy otevřené. Například otázka porušení zapouzdření objektu při dotazování, formulace dotazu pouze na data nebo i metody. Důležité je rozhodnutí, jak chápat odpověď na dotaz. Zda výsledkem dotazu budou pouze hodnoty – ve formě datových

struktur, složených z literálů (např. tabulky), nebo nové objekty. OQL vychází ze zaběhnutého dotazu v SQL se syntaxí SELECT … FROM … WHERE …s rozšířením o rysy, které přináší ODL, jako jsou metody, strukturované a vícehodnotové atributy, vztahy definované ve třídách. Využití tečkové notace a pojmenování objektů, atributů, metod a vztahů umožňuje formulovat výrazy, které tvoří u složitých objektů *cesty*.

Např. u.adresa.ulice nebo u.praxe()

Výsledkem dotazu může být kolekce struktur (structs) nebo objektů, využitelná jako extenze.

Základní struktura dotazu je

SELECT < výraz >

FROM <extent>

WHERE < podmínka >

Podobnost s SQL je mimo jiné v použití

- DISTINCT , EXISTS
- Agregačních funkcí COUNT(), SUM(), MAX(), …
- Množinové operace UNION, INTRSECT, EXCEPT (MINUS)
- Vyhodnocení podmínky X ANDTHEN Y, X ORELSE Y

{FOR ALL | EXISTS } x IN (poddotaz) : (podmínka)

Další rysy – např. kompatibilita typů, ekvivalence a porovnávání, dědičnost, kolekce různých typů, pohledy

Pro ilustraci uveďme několik jednoduchých příkladů -

1. Dotaz s metodou :

SELECT u

FROM učitels AS u

WHERE u.praxe() > 5

2. Se strukturovaným atributem :

SELECT u, u.adresa.ulice

FROM učitels AS u

WHERE u.adresa.obec = „OLOMOUC“

3. S vazbou N:M, kde se v SELECT frázi vyčíslí obecně množina jmen a výstupem je tabulka se záhlavím (jméno : string, pomocník : SET<string>) :

SELECT u.jméno, (SELECT a.name

FROM u.učí.podporován AS a)

AS pomocník

FROM učitels AS u

Průvodce studiem

OQL je podobný SQL základní konstrukcí SELECT – FROM – WHERE, s rozšířením o komplexní objekty s identitou OID, výrazy v tečkové notaci pro vyjádření cesty ve struktuře objektu, možností využít metod, polymorfizmu a pozdní vazby. OQL může být vnořen do OO programovacích jazyků - Smalltalk, C++ nebo Java.

Shrnutí

Relační algebra tvoří základ pro většinu relačních dotazovacích jazyků.Je to procedurální jazyk vyšší úrovně. Základní operace jsou množinové – sjednocení, průnik, rozdíl, kartézský součin a speciální – projekce, selekce, přirozené i vnější spojení a přejmenování. Selekcí získáme část výchozí relace, jejíž n-tice vyhovují selekční podmínce. Projekcí se redukuje záhlaví výchozí relace na vyjmenované položky – sloupce. Obecné spojení je podmnožina kartézského součinu relací, porovnávají se všechny kombinace n-tic zúčastněných relací a vybere se dvojice, která vyhovuje spojovací podmínce. Pokud v podmínce porovnáváme všechny sdílené atributy v jedné i na druhé relaci, hovoříme o přirozeném spojení. Vnější spojení – levé, pravé nebo úplné umožní ve výsledku získat i data z n-tic příslušných (levé, pravé nebo obou) relací, které nevyhovují spojovací podmínce. V takovém případě je v druhé části výsledné dvojice n-tic nedefinovaná hodnota NULL. Pro výběr informací z relační databáze lze využít jazyk logiky - predikátového kalkulu 1. řádu ve formě relačního kalkulu. Je to neprocedurální jazyk, který je možné použít pro porovnání síly relačních dotazovacích jazyků. Jazyk, který dokáže dotazem získat libovolnou relaci odvoditelnou relačním kalkulem se nazývá relačně úplný. Dotaz má formu výrazu s volnými proměnnými a formulí kalkulu s predikáty. Odpovědí jsou hodnoty, které po dosazení do volných proměnných splňují formuli. Bezpečný výraz určuje pouze konečnou množinu dat v odpovědi. Ke každému výrazu relační algebry existuje ekvivalentní bezpečný výraz relačního kalkulu a opačně, oběma prostředky je možno vyjádřit stejné třídy dotazů. SQL jazyk je nejdůležitější uživatelský databázový jazyk relačních databází ve verzi SQL-92 a objektově-relačních databází ve verzi SQL-99 nebo SQL3. Pro dotazování slouží příkaz SELECT – FROM – WHERE, pomocí kterého můžeme použít všechny operace relační algebry, například použitím operátorů UNION, INTERSECT, EXCEPT, JOIN, LEFT JOIN, … . Do frází FROM nebo WHERE můžeme vnořit poddotaz (= další SELECT) například pomocí operátorů EXISTS, IN, ALL, ANY nebo pomocí relačních operátorů =, <>, <, >, … . Výsledky některých operací nemusí tvořit množiny n-tic – např. při použití operátoru UNION ALL. Pokud chceme odstranit duplicity, používáme ve správném kontextu operátor DISTINCT. Výstupní n-tice můžeme setřídit operátorem GROUP BY. Pro aritmetické výpočty slouží agregované funkce SUM, COUNT, MAX, MIN, AVG s možností aplikace na skupiny n-tic pomocí operátoru GROUP BY a možností výběru skupin pomocí HAVING. Další příkazy JMD jsou INSERT, DELETE, UPDATE. Příkazy JDD jsou například CREATE TABLE, CREATE VIEW, CREATE INDEX, … , podobně pro modifikaci schématu ALTER TABLE, ALTER VIEW, …, DROP TABLE, DROP VIEW, … . OQL nabízí podobně jako SQL výraz SELECT … FROM … WHERE. V klauzuli FROM mohou být deklarovány proměnné pro libovolné kolekce objektů nebo atributů, extenty tříd, atd. Operátory jsou v plné šíři převzaty z SQL. Je možné definovat uživatelské a referenční typy.

Pojmy k zapamatování

- Operace relační algebry - sjednocení, průnik, rozdíl, kartézský součin, projekce, selekce, spojení a přejmenování
- n-ticový a doménový relační kalkul, bezpečný výraz
- jazyk SQL, OQL

Kontrolní otázky

24. *Jaké jsou operace relační algebry a jak působí?*
25. *Jaké jsou druhy spojení a jak se liší?*
26. *Jak jsou definovány operace relační algebry pomocí relačního kalkulu?*
27. *Jaký je vztah relační algebry a relačního kalkulu?*
28. *Jaká je syntaxe hlavních příkazů jazyka SQL?*
29. *Kde v příkazu SELECT najdeme jednotlivé operace relační algebry?*
30. *Jak se formulují selekční podmínky?*
31. *Jak se vnořují podotázky v SQL?*
32. *Čím se liší OQL od SQL?*

Úkoly k textu

Ověřte si prakticky syntaxi příkazu SELECT jazyka SQL na všech operacích relační algebry na dostupném DBS.
Navrhněte a vytvořte na relačním systému jednoduchou databázi a vyzkoušejte příkazy pro vytvoření tabulek a manipulaci s daty.

5 Fyzická organizace dat

Studijní cíle: Student by se měl seznámit se s hierarchií, vlastnostmi a typy pamětí v procesu přenosu mezi hlavní pamětí a vnější pamětí, se strukturou a organizací uložení dat v souborech s pevnou i proměnnou délkou. Měl by vysvětlit metody přístupu k datům a související podpůrné datové struktury.

Klíčová slova: Položka, záznam, organizace souboru, primární sekundární a terciární paměť, hustý index, řídký index, víceúrovňový index, B-strom, ISAM, přímé hašování, nepřímé hašování, statické hašování, dynamické hašování, Faginovo hašování, hašovací funkce, shlukování

Potřebný čas: 3 hodiny

5.1 Databázový přístup k datům

Fyzická reprezentace datové struktury, implementace databázových operací a strategie organizace přenosu dat mezi diskovou pamětí a operační pamětí rozhoduje o efektivitě a úspěšnosti SŘBD. Databáze je typicky uložena jako kolekce souborů. Při popisu struktury uložení se setkáme s pojmy jako:

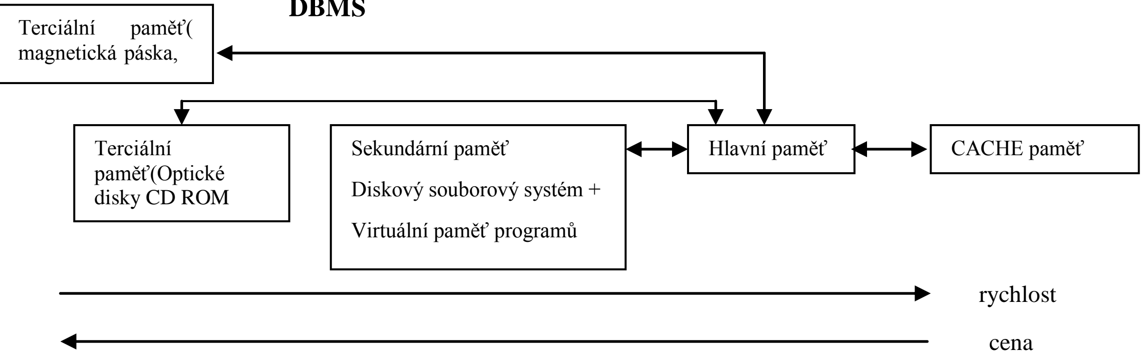
Záznam (věta)

- logický (kolekce logicky souvisejících položek, hodnot atributů)
- fyzický (doplnění o oddělovače a další režijní položky, definice délek, …)

Soubor - homogenní (záznamy stejného typu, pevné délky) a *nehomogenní* - je identifikovatelná kolekce logicky souvisejících záznamů, seskupená do bloků. Soubory se záznamy s proměnnou délkou mohou obsahovat záznamy různých typů, nebo typ záznamu s proměnnou délkou některých položek, ale také s opakujícími se položkami. K dalším datovým strukturám patří datový slovník (jinak systémový katalog) a indexy. Systémový katalog obsahuje mimo jiné metadata schématu databáze, tj. logická jména relací a jejich atributů, domény, IO, pohledy, indexy, …. Dále statistické informace o uložené databázi.

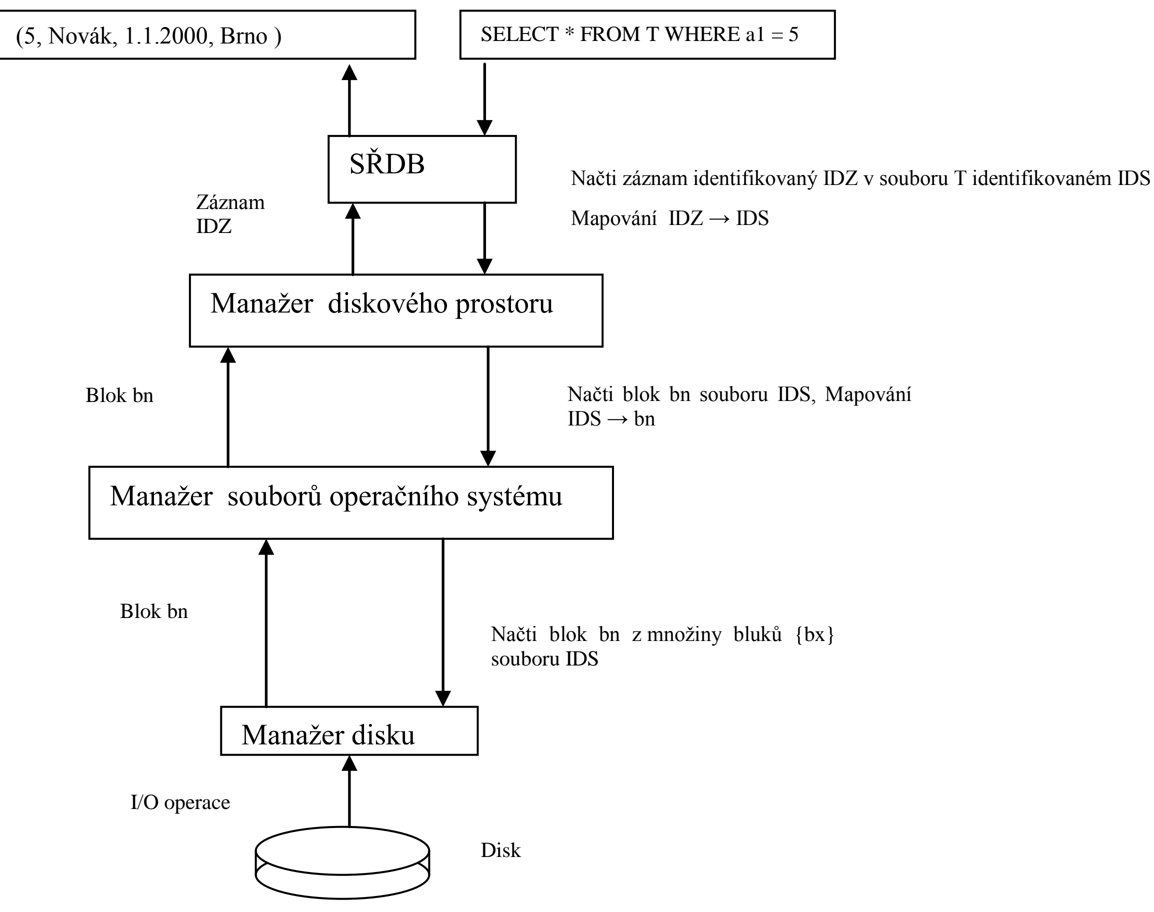
Záznamy tedy mohou být v databázi uloženy fyzicky různým způsobem a v průběhu zpracování se nachází v různých typech pamětí s klasifikačními parametry, jako jsou rychlost přístupu, cena za jednotku dat, spolehlivost a chování při ztrátě napájení nebo poruše zařízení.

Hierarchie pamětí v DBS je na následujícím obrázku



Terciální paměť je velkokapacitní (TB), často se sekvenčním přístupem, s příznivým poměrem cena/ byte, s použitím například pro zálohování systému. Sekundární disková paměť je typicky rychlejší, s přímým přístupem, zabezpečena proti ztrátě informace při poruše disku (disková pole RAID s úrovněmi 4-6 se samoopravnými kódy), se standardními konstrukčními prvky. Proto u disků můžeme dobu přístupu (čas od požadavku na čtení nebo zápis a počátkem přenosu dat) rozdělit na čas nastavení hlavičky disku na požadovanou stopu (Seek time ~ 1-20 ms) a čas, kdy se disk otočí do polohy, kdy hlavička je u požadovaného sektoru na stopě (Rotational latency ~ 0-10ms). Další důležité parametry jsou rychlost přenosu dat (Data-transfer rate ~ 1ms/4kB) a průměrná doba mezi poruchami disku (Mean time to failure). *Diskovou adresu* na fyzické úrovni proto tvoří označení diskové jednotky, číslo cylindru, stopy a sektoru.
Blok je souvislá sekvence sektorů z jedné stopy a zároveň je základní jednotkou pro přenos dat mezi pamětí vnitřní a vnější. Rychlost přenosu se měří i u rychlých systémů v ms, rychlost operací ve vnitřní (hlavní) paměti je řádově měřena v μs a vyšší. Tedy rychlost zpracování dat záleží na počtu přenosů dat mezi diskem a primární pamětí a prakticky nezáleží na počtu operací v hlavní paměti , případně paměti typu CAHE (statická paměť typu CACHE je nejrychlejší, ale i nejdražší a stejně jako hlavní paměť je energeticky závislá). Dalším typem paměti, jež zatím není tolik rozšířen, je paměť FLASH. Ta je energeticky nezávislá a srovnatelně rychlá, jako operační paměť, ale počet přepisovacích cyklů paměti je limitován.

Ilustrační příklad vyhledání záznamu na disku :



Na základě logické podmínky pro záznam určíme logickou adresu záznamu IDZ, tj. dle okolností pořadové číslo záznamu pro soubor s pevnou délkou nebo adresu v rámci souboru. V obou případech lze pak určit číslo bloku bn v souboru IDS, ve kterém je záznam uložen a určit začátek záznamu v bloku. Dále odvodíme z čísla bloku fyzickou adresu záznamu, tedy číslo válce, stopy a sektoru.

První etapu řeší SŘBD: v průběhu zpracování aplikace nebo zadáním dotazu pomocí dotazovacího jazyka je zadána logická podmínka, se kterým záznamem se bude pracovat.

Druhou 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 pořadí záznamů v bloku určí hledaný záznam.

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. Volí se různé strategie výměny bloků (LRU - least recently used – uvolňuje se blok, se kterým se nejdéle nepracovalo,FIFO, MRU- Most recently used s fixací bloků (Pinned block), které se nevrací na disk). Snaha o předvídání potřeby následujících bloků z analýzy využití předešlých bloků i modelu chování a jejich přesunutí do vyrovnávací paměti. Implementace vyhodnocení dotazů má obvykle předvídatelně definované modely chování, které se dají po získání informací z DBS o uživatelském dotazu použít jako předloha pro předpověď využitelných následujících bloků. LRU například selhává u dotazů, kdy opakovaně pracujeme a vracíme se k již použitým datům. Manažér vyrovnávací paměti může využít statistické informace k odhadu pravděpodobnosti použití bloků na disku, proto jsou ve vyrovnávací paměti bloky systémového katalogu a indexů.

5.2 Organizace souborů

Počet přístupů závisí také na organizaci uložení dat v diskových souborech. Při popisu následujících technik budeme předpokládat soubory s pevnou délkou záznamu. Jazyky třetí generace nabízely sekvenční, index-sekvenční a indexované organizace, vhodné spíše pro statické - neaktualizované databáze. S počtem aktualizací narůstají problémy s výchozím, pevně vymezeným diskovým prostorem a řešením jsou neefektivní techniky, vedoucí na řetězení záznamů v přetokových oblastech. Odpovědí jsou dynamické organizace dat, kdy i při aktualizacích databáze je rychlost vyhledání konstantní, nebo logaritmická. Přesto se v DBS můžeme stále setkat se všemi uvedenými postupy. Záznamy mohou být obecně organizovány několika základními způsoby v mnoha alternativách, z nichž každý může být optimální v jisté situaci a méně vhodný v jiné. Nejčastější formy:

Formy organizace souborů –

Hromada je vhodná pro vkládání velkého počtu záznamů.

Hašováné se hodí pro výběr podle určitých hodnot klíče.

Indexované se hodí pro univerzální a intervalové výběry.

- Hromada/ nesetříděné – záznamy mohou být umístěny v souboru kdekoliv je volné místo, bez jakéhokoliv uspořádání. Organizace vhodná pro případ zpracování všech nebo většiny záznamů.
- Sekvenční - záznamy jsou ukládány sekvenčně v uspořádání podle pořadí vkládání záznamů nebo
- Setříděné - podle vyhledávacího klíče. Vhodné pro případ, kdy zpracování informace probíhá při jistém uspořádání záznamů nebo pro jistý interval hodnot.
- Indexované – pomocná datová struktura (index) urychluje přístup do hlavního datového souboru
- Hašované – hašovací funkce z klíčového atributu vypočte fyzickou adresu bloku ukládaného nebo uloženého záznamu. Vhodné pro vyhledávání na rovnost.
- Shlukované (Clustering) – záznamy různých typů jsou uloženy v jednom souboru, související záznamy jsou uloženy ve stejném bloku. Vhodné pro efektivní spojování zúčastněných relací.

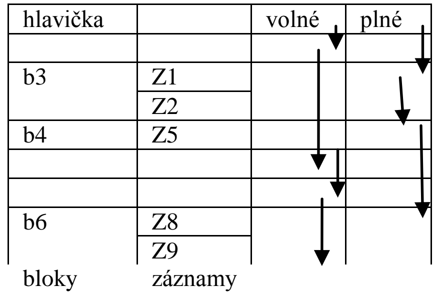
Průvodce studiem

Fyzický návrh databáze je postup popisující implementaci databáze na discích počítače. Určuje bázové relace a jejich uložení na disku a přístupové metody tak, aby systém pracoval efektivně i při splnění podmínek integritních omezení a bezpečnostních hledisek, vždy v návaznosti na podporu poskytovanou SŘBD. V prvním kroku se provede transformace integrovaného logického schématu do formy implementovatelné SŘDB, dále se navrhuje organizace souborů, přístupové metody, volba indexů, odhaduje prostor na disku, potřebný k implementaci.

5.2.1 Sekvenční soubory

Nejjednodušší organizací, vycházející z přirozeného uspořádání záznamů podle pořadí jejich uložení, jsou sekvenční soubory. Věty jsou uloženy v souboru v libovolném pořadí v blocích, které mohou následovat fyzicky za sebou(v následujícím cylindru, stopě) a díky eliminaci nebo minimalizaci mechanických přesunů hlavičky a předvídané načítání několika bloků najednou do vyrovnávací paměti (pre-fetching) je sekvenční přístup rychlejší než náhodný. Pokud bloky nenásledují fyzicky za sebou (sekvence je na vyšší logické úrovni) , jsou propojeny ukazateli (obsazené i uvolněné bloky), nebo jsou adresy bloků souboru někde uloženy - obvykle řeší OS. Implementace sekvenčních souborů je nejjednodušší.

Z1	b1
Z2	
Z3	
Z4	b2
Z5	
Z6	
záznamy	bloky



Provádění databázových operace - nový záznam se uloží jednoduše na konec souboru.Pro vyhledání záznamu je nutno prohledávat datový soubor sekvenčně : každý záznam postupně načíst a otestovat, zda vyhovuje podmínce. Vyhledávání sekvenční potřebuje průměrně n/2 porovnání nebo n/(2*f_R) přístupů na disk. Číslo n znamená počet záznamů a číslo f_R je blokovací faktor, znamená počet záznamů v bloku. Modifikace záznamu znamená tyto operace : nalézt záznam, načíst, opravit a na stejnou adresu znovu zapsat. Zrušení záznamu u sekvenčních souborů se obvykle provádí označením jeho neplatnosti, ne vymazáním. 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 postupně zabírají v souboru místo. Je možné zbavit se těchto položek a soubor setřást. 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 klíče, musí se prohledat celý soubor a zkontrolovat jedinečnost klíče vkládané věty.

5.2.2 Setříděné sekvenční soubory

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. Znamená to po každé změně (vložení nové věty nebo modifikaci klíče) znovu soubor setřídit. Pak se dá vyhledávat podle klíče mnohem rychleji (např. metodou půlení intervalu nebo některou její modifikací). Počet přenosů pro binární hledání je průměrně log n.

Někdy se metoda vylepšuje tak, že provádíme interpolaci na základě znalosti statistického rozložení hodnot klíče v tabulce.

Jestliže vyžadujeme, aby záznamy byly nějakým způsobem setříděny, je možno použít také 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, seznamy mohou realizovat uspořádání dle libovolného kritéria.

5.2.3 Indexování

Datové záznamy jsou v indexovaném sekvenčním souboru, ke kterému existuje jedna nebo několik dalších pomocných struktur, uložených v indexovém souboru, pomocí nichž můžeme v datovém souboru rychleji hledat. Indexové soubory jsou podstatně menší než datové.

Indexování je založeno na principu rychlého nalezení dvojice - vyhledávací klíč, (fyzická) adresa záznamu, která je **uložena** v pomocné struktuře – indexu. Ukazatele nemusí být tedy součástí záznamů (jako u zřetězených organizací), ale mohou být uloženy zvlášť.

Indexový soubor - index

IDP(klíč)	Adresa, ukazatel
3	
12	
15	

Datový soubor

IDP(klíč)	Jméno	profese
12	Petr	dělník
3	Jana	kuchařka
15	Karel	dělník

Správa pomocných indexových struktur při manipulaci s daty zabírá čas. Proto mezi sledovaná kritéria při použití indexů počítáme nejenom zrychlení při vyhledávání, ale i zpomalení při vkládání a mazání dat. Indexem nemusí být jen primární klíč, ale kterákoliv položka souboru nebo seznam několika položek.

Primární index – jeho vyhledávací klíč(obsahuje klíčové atributy) určuje uložení záznamu.

Sekundární index – jeho vyhledávací klíč neurčuje uložení záznamu.Realizován mapováním do datového soboru nebo do hodnot primárního klíče. Může být i více sekundárních indexů k jednomu datovému. Pro sekundární index se také používá název *invertovaný soubor.*

sekundární index

profese (klíč)	Adresy, ukazatelé, PK
dělník	12, 15
kuchařka	3

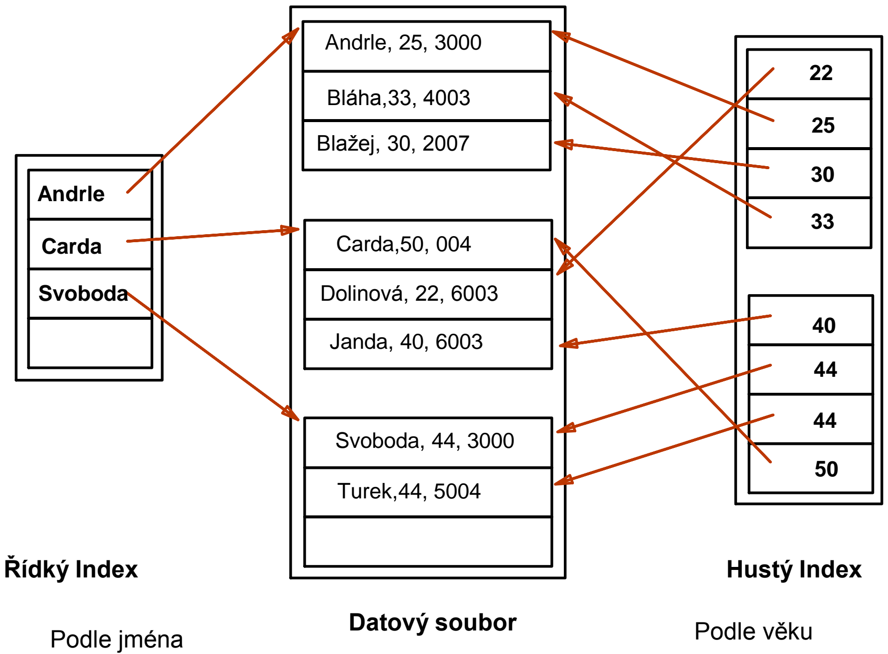
Datový soubor

IDP(PK)	Jméno	profese
12	Peter	dělník
3	Jana	kuchařka
15	Karel	dělník

Jednoduché indexování :

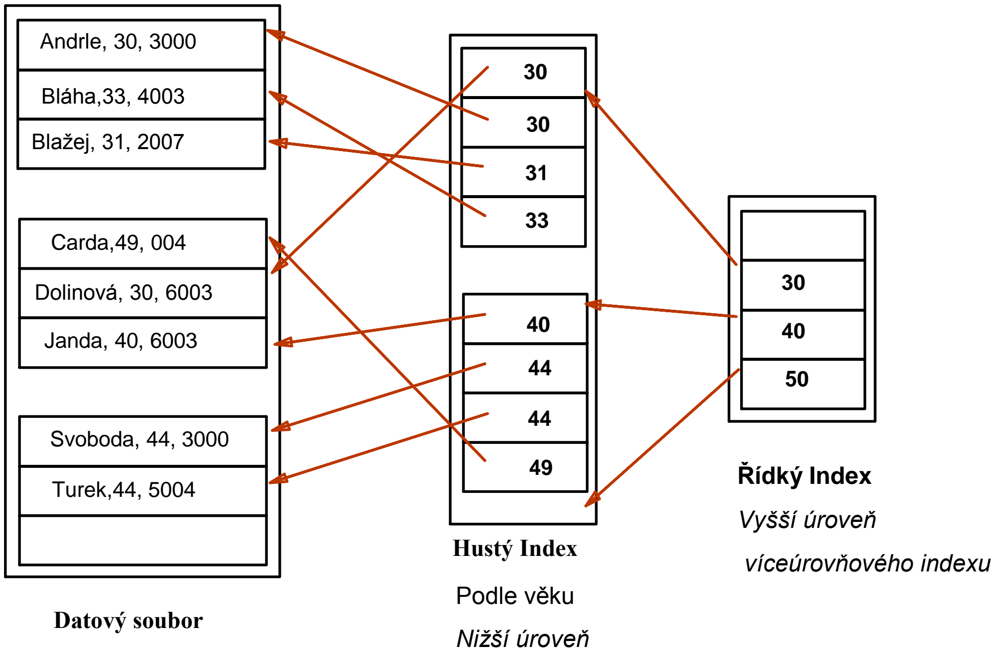
Hustý index – v indexu jsou sekvenčně uloženy a setříděny **všechny** vyhledávací klíče datového souboru s příslušnými odkazy IDZ. Datový soubor je typicky **nesetříděný** podle vyhledávacího klíče. Jednomu indexovému záznamu odpovídá jedna hodnota vyhledávacího klíče v datovém souboru.

Řídký index – v indexu jsou sekvenčně uloženy a setříděny **některé** vybrané vyhledávací klíče datového souboru s příslušnými odkazy. Datový souboru je **setříděný** podle vyhledávacího klíče. Jednomu indexovému záznamu odpovídá řada hodnot vyhledávacího klíče v datovém souboru. Typicky odkaz indexovému záznamu určuje právě jeden datový blok.



V některých aplikacích mohou být klíče velmi dlouhé. Pak se také místo v paměti zvětšuje a 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.

Jednoduché indexování (lineární, jeden index na záznam) může vést k velkým a neefektivním indexovým souborům. Proto se používají sofistikovanější postupy. Přechodem k nim je: *Víceúrovňový index* – struktura „index na index“, je možno pro hledání v indexovém souboru použít opět indexový soubor a sestrojit tak celou hierarchii indexových souborů. Typicky je na vyšších úrovních použit řídký index a na nejnižší úrovni je použit hustý index.



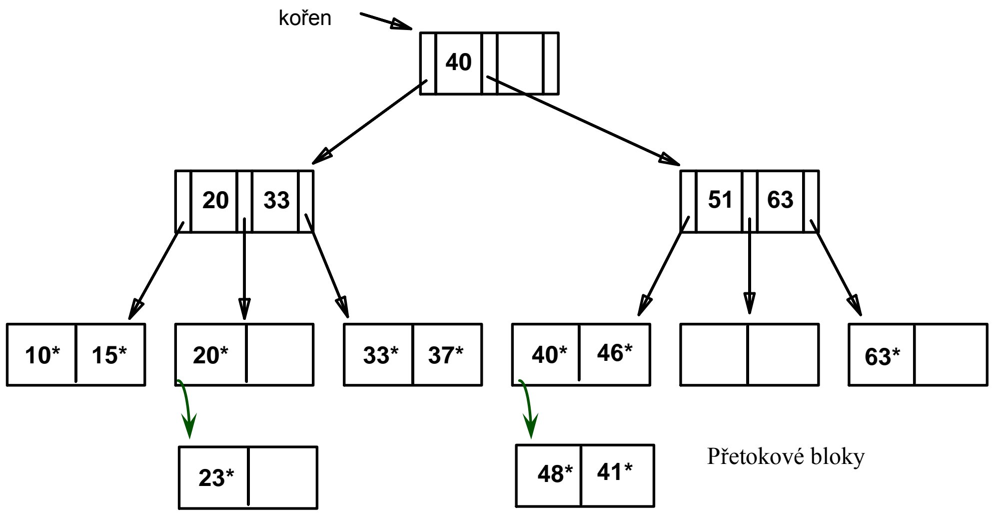
Dalším vylepšením je použití stromových struktur, které podporují dotazy na rovnost i na interval hodnot. Pro statičtější případy databází se používají struktury ISAM (Indexed Sequential Access Method) se sekvenčním i náhodným, indexem podporovaný přístupem(index-sekvenční soubor), pro dynamičtější případy jsou používané B^+ stromy. Základem obou přístupů jsou varianty stromových struktur. Nejefektivnější jsou varianty B stromu , s požadavkem vyváženosti(všechny cesty od kořene stromu do libovolného listu jsou stejně dlouhé), nelistový uzel je chápán jako blok se strukturou (po, K1, p1, K2, p2, …, Kn, pn). Kde pi jsou ukazatele na následníky, Ki jsou vyhledávací klíče, pro které platí K1< K2 < K3 < … < Kn .Každý uzel má nejvýše m a nejméně m/2 následníků. Levý podstrom klíče Ki obsahuje klíče menší nebo stejné než Ki, pravý podstrom naopak klíče větší. V listových uzlech jsou všechny vyhledávací klíče s odkazy do datových stránek (B^+ stromy), doplněné o ukazatel na následující listový uzel (blok) pro rychlé sekvenční čtení dat. Počet přístupů na disk při vyhledávání odpovídá výšce stromu ~ $\log_m n$, kde n je počet klíčů primárního datového souboru. Vyváženosti při manipulaci s daty se u B-stromů dosahuje štěpením a sléváním uzlů, u ISAM, s nemodifikovatelnou strukturou vnitřních uzlů stromu, pomocí přetokových bloků (stránek).

Při vyhledávání datového záznamu 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 v podstromu. Modifikace záznamu je z hlediska vyhledávání triviální. Ilustračně popíšeme zbylé manipulace.

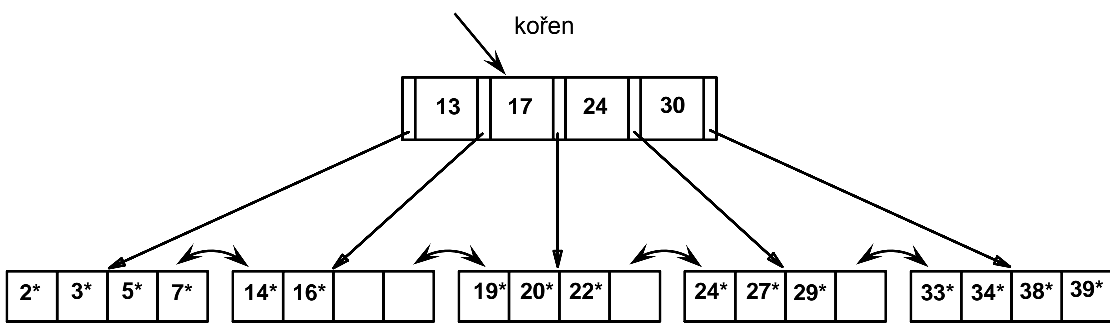
Při vkládání nového záznamu najdeme příslušný blok a mohou nastat dvě možnosti : buď v nalezeném bloku je dostatečný prostor, takže můžeme přidat vkládaná data, nebo nalezený blok je plný, takže musíme vytvořit nový blok; z původního plného bloku vytvoříme dva bloky. Do vyšší úrovně musíme nový blok 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 další úroveň v indexové struktuře.

Rušení záznamů se provádí opačně, než vkládání. Při zrušení posledního záznamu bloku se zruší i odkaz na něj, totéž se promítne do vyšších úrovní, případně se v krajním případě může hierarchie indexů o jednu úroveň snížit – rozhoduje o tom obsazení uzlů.

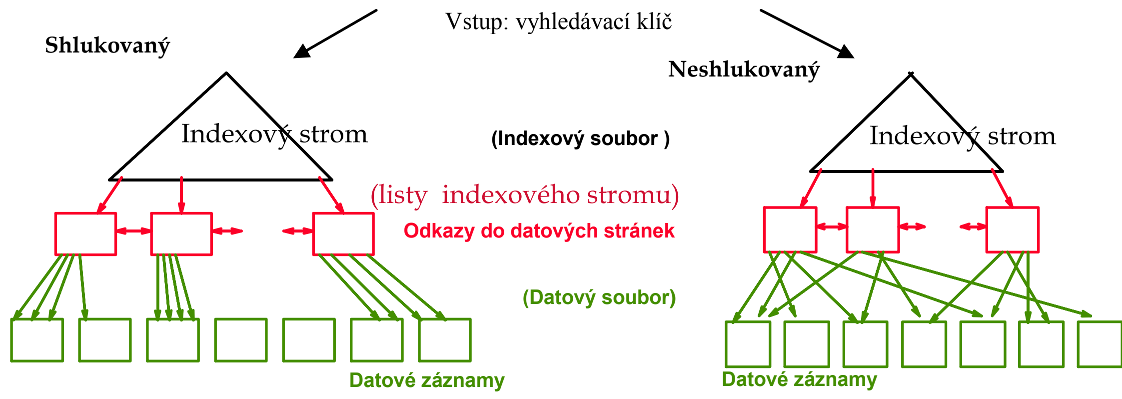
Příklad ISAM stromu po několika vloženích a smazáních datových záznamů.



příklad B^+ pro m = 4:



Nakonec si ještě na příkladu ukážeme shlukované a neshlukované organizace stromových indexových struktur. *Shlukované indexy* organizují data v datovém souboru tak, že záznamy se stejným nebo blízkým vyhledávacím klíčem jsou uloženy ve stejném, nebo blízkém bloku.



.

Průvodce studiem

Index je pomocná struktura, ve které je efektivním způsobem zpřístupněna uložená dvojice <klíč, adresa záznamu>. Hašování pomocí hašovací funkce, parametrizované klíčem, určí adresu uložení, použitelnou i při vyhledávání záznamu výpočtem.

5.2.4 Soubory s přímým adresováním - hašování

Velmi rychlý přístup k záznamům prostřednictvím klíčů zajišťuje metoda přímého adresování. Teoretický princip metody je tento - jednoznačný klíč záznamu se pomocí hašovací funkce transformuje do jednoznačného čísla, které určuje adresu záznamu v souboru. Tak je možné jediným přístupem na disk záznam načíst, případně zapsat. Požadavkem na hašovací funkci je zajištění rovnoměrného rozložení obsazených míst adresového prostoru i pro náhodné aktuálně zpracovávané hodnoty klíče. Hašování může být:

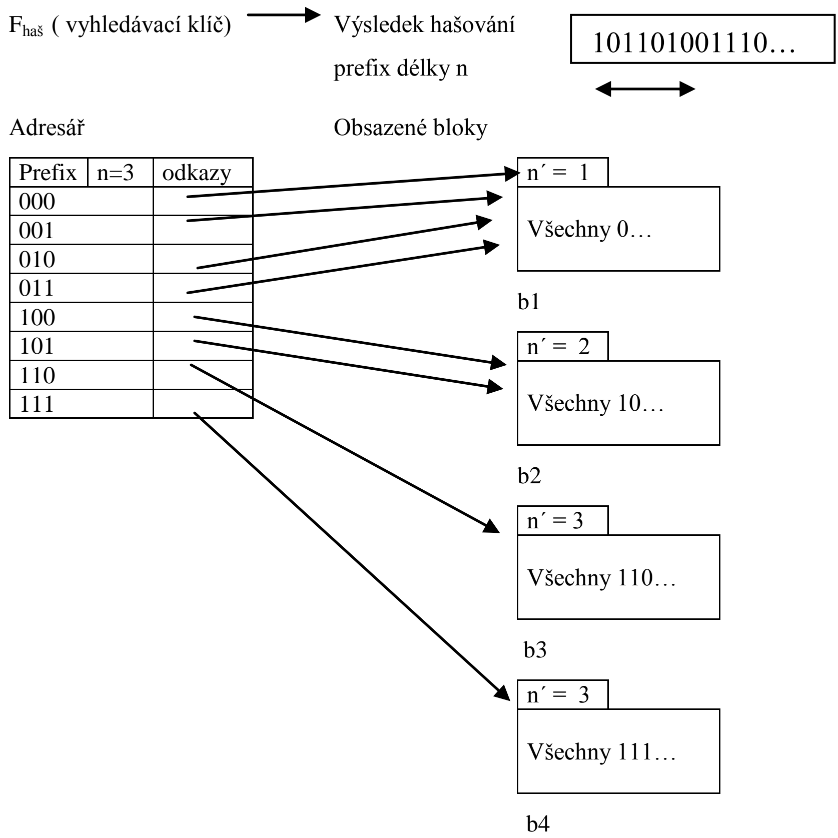
- přímé (výsledkem hašování je adresa v primárním datovém souboru)
- nepřímé(výsledkem hašování je adresa sektoru ukazatelů)
- statické (velikost adresového prostoru je konstantní)
- dynamické(velikost adresového prostoru se přizpůsobuje potřebám)

Problémem statického hašování jsou kolize a velikost adresového prostoru. Kolize jsou důsledkem vlastnosti každé běžně používané hašovací funkce, kdy několika vyhledávacím klíčům odpovídá stejná adresa - $F_{haš}(K1) = F_{haš}(K2)$.

Na jedné adrese ale nemůže být více záznamů. Situace se pak řeší různými technikami. Mezi nejjednodušší způsoby patří postup, kdy všechny záznamy, určené hodnotou hašovací funkce do jednoho adresového prostoru se zřetězí do seznamu záznamů. Když se do databáze přidává mnoho nových záznamů, adresový prostor je více zaplněn a dochází často ke kolizím a zřetězení záznamů vede ke zmenšování rychlosti vyhledání – proces se blíží sekvenčnímu vyhledávání. To vede k opakované reorganizaci – nová velikost adresového prostoru, nová hašovací funkce. Další možností je použití přetokové oblasti, nebo dynamicky modifikované vícenásobné hašovaní a další strategie.

Dynamické hašování nabízí mnoho sofistikovaných postupů, jak se problémům statického hašování vyhnout. Typickým příkladem je Faginovo hašování:

Hašovací funkce transformuje vyhledávací klíč do binárního řetězce. Je použita pomocná dynamická struktura – adresář. Ten může být modelován jako tabulka s jedním sloupcem prefixů zahašovaných klíčů a druhým sloupcem odkazů na paměťové bloky. Podle aktuální potřeby bloků paměti se mění velikost adresáře na 2^n, kde n je délka prefixu (v našem případě n=3, velikost je 8). Délka prefixu (hloubka adresáře) je zaznamenána v hlavičce adresáře. Podobně je umístěna odvozená veličina, tzv. lokální hloubka - n´, v každé paměťové stránce. Pro vložení a vyhledání záznamu platí stejné postupy. Z bitového řetězce po hašování je oddělen prefix s aktuální délkou n. Ve sloupci adresáře, kde jsou všechny kombinace binárních řetězců dané délky, je nalezen řádek s odpovídající hodnotou prefixu a zároveň odkazem na blok paměti, kde je nový záznam umístěn, nebo starý nalezen. Pokud při vkládání je určen plný blok a pro jeho n´ platí n´ < n, tak operační systém dodá nový paměťový datový blok, zvětší se n´ na n´ = n´ + 1 a záznamy z původní stránky se rozdělí do obou bloků podle nového prefixu n´. Zároveň se opraví příslušný odkaz v adresáři na nový blok. Pokud ale n´ = n, zvětší se prefix n na n = n + 1, adresář se zdvojnásobí a dále se postupuje stejně jako v předešlém případě. Při mazání záznamů se kontroluje obsazení bloků a pokud to podmínky dovolí, dojde ke slévání stránek a případně zmenšení adresáře opačným postupem.



Obr. 13 Faginovo dynamické hašování

5.2.5 Shlukování – clustering

Techniky shlukování umožňují uživateli ovlivnit fyzické uložení záznamů tak, že v jednom paměťové bloku (nebo z hlediska přístupu blízkém) jsou umístěny záznamy různých typů se stejnou hodnotou v atributech spojovací podmínky, figurující v nejfrekventovanějších dotazech uživatelů. Schématicky pro vazbu 1:N mezi relací **R** a **S** s vazebním atributem **a** se organizace souboru dá znázornit

Blok: b1 (R1.a = S1.a = S3.a = S4.a, …)								b2 (R3.a=S7.a, R4.a=S8.a=S9.a)							
R1	S1		S3	S4	R2	S5	S6	R3	S7		R4	S8	S9		

Existují dva typy shluků - indexované a hašované. Zhruba platí, že u spojení dvou tabulek pro vytvoření jednoho spojeného záznamu je třeba dva přístupy na disk při klasickém uložení a jen jeden při použití shlukování. Protože se jedná o fyzické uložení, je možné takto preferovat jenom jednu skupinu dotazů, pro ostatní to neplatí – vazby na ostatní tabulky, prostřednictvím jiných vazebních atributů.

5.2.6 Indexování pomocí binární matice

Pro sekundární indexování se někdy pro ušetření kapacity používá jiného způsobu - 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 různých hodnot.

atribut	**Hodnota atributu**	**pořadí záznamů**	1	2	3	
profese	dělník		1	0	1	
	kuchařka		0	1	0	
Jméno	Petr		1	0	0	
	Jana		0	1	0	
	Karel		0	0	1	

Binární matice jsou zvlášť výhodné v případech, že se hodnoty sekundárních atributů nebo záznamy nemění. Hlavní výhodou binárních matic je rychlá realizace kombinovaných dotazů pomocí logických operátorů negace, konjunkce a disjunkce.

5.2.7 Soubory s proměnnou délkou záznamu

Dosud jsme předpokládali 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 známým počtem i předem neznámým počtem, o skupinové položky, dále o dlouhé texty různé délky, o záznamy obrázků, zvuků (datový typ BLOB, CLOB, …) 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. Využití pevné délky 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 sní samostatně.
- 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; další problém je rozpoznání prázdného prvku
- místo opakující se položky uvedeme odkaz na seznam jejích prvků, ten může být součástí jiného souboru;

Př.

Ve stejném souboru

Z1	A100	500	
Z2	A60	330	
Z3	A200	978	
	A130	60	
	A90	500	⊥
	A210	0	⊥

V jiném souboru

Z1	A100	500	
Z2	A60	330	
Z3	A200	978	

	A130	60	
	A90	500	⊥
	A210	0	⊥

- 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.

Př.

Z1	A100	500	A130	60	A90	500
Z2	A60	330	⊥	⊥	⊥	⊥
Z3	A200	978	A210	0	⊥	⊥

2. Proměnná délka záznamu v sekvenčním souboru

Při sekvenční organizaci souborů s proměnnou délkou záznamu je nutno od sebe jednotlivé záznamy rozlišit. Používají se tyto metody :

- systém oddělovačů: záznamy jsou odděleny oddělovačem, např. ⊥ (viz např. textové soubory), uvnitř záznamu se atributy oddělují jiným typem oddělovače, opakující se položky dalším typem ap.

Př.

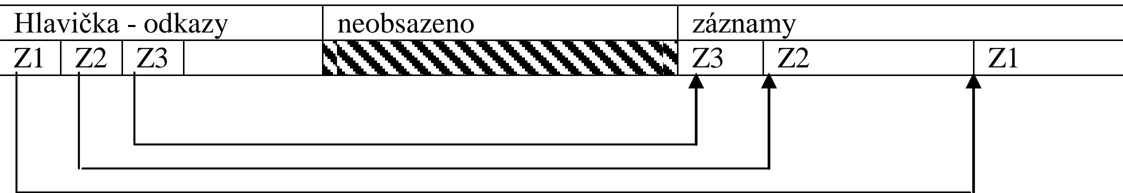
Z1	A100	500;	A130	60;	A90	500;	⊥
Z2	A60	330;	⊥				
Z3	A200	978;	A210	0;	⊥		

- zaznamenání délky aktuálního záznamu na začátku záznamu (pro jednosměrný průchod souborem), či na začátku i konci záznamu (pro obousměrný průchod souborem)

3. Různé typy záznamů v jednom souboru

4. 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 Využití různých typů paměti počítače databázovým systémem je ovlivněno parametry jako rychlost, kapacita, cena za bit, stálost uložení. V pořadí od menších/ dražších k větším/ levnějším se setkáváme s pamětí typu cache, operační pamětí, sekundární diskovou pamětí a

terciární páskovou nebo CD ROM pamětí. Pro rychlé zpracování dat je rozhodující práce manažeru dat a disku se strukturou dat na disku, organizací sektorů a bloků, spolupráce operační paměti, vyrovnávací paměti a disku. Existuje mnoho způsobů, jak zrychlit přístup k datům. Mezi často používané patří metoda rozdělení dat mezi několik diskových jednotek s využitím souběžného přístupu, použití zrcadlových disků s udržováním několika kopií dat rovněž s možností souběžného přístupu. Dále se klasicky využívá konstrukce disku a organizace dat pro souběžný přístup na stopy nebo cylindry disku s uchováním maximální velikosti dat ve více vyrovnávacích pamětích a sofistikovaných algoritmech predikce následně použitých bloků dat. Disková zařízení musí být schopná maximálně eliminovat případné chyby. K odhalení slouží klasické cyklické součty, parita, atd., ale užitečnější jsou postupy umožňující opravy případných chyb. RAID úrovně 1 využívá zrcadlení, RAID úrovně 4 používá disk, jehož obsah je paritním součtem korespondujících bitů ostatních disků. RAID úrovně 6 umožňuje použití samoopravné korekce chyb a zvládne odstranit chyby i u několika souběžných poruch ve skupině disků. Položky nebo záznamy tvoří základ struktury uložení. Mohou být obecně pevné (standardní atomické typy – INT, CHAR(15), atd.), nebo proměnné délky. Záznamy mohou obsahovat hlavičku s informací o struktuře položek, délce a podobně. Tyto informace jsou nezbytné u organizací s různou délkou položek nebo záznamů. Záznamy jsou organizované v blocích, které rovněž ve své hlavičce udržují informace o datové struktuře bloku. Velké záznamy informací ve formě obrázků, zvuků atd. jsou typu BLOB (binary large object) a typicky zaberou několik bloků, nejlépe na jednom cylindru. Záznamy mohou být obecně organizovány ve formě hromady, sekvenční, setříděné, indexované, hašované a shlukované. Hustý index představuje pomocnou strukturu, ve které jsou uloženy dvojice klíčová hodnota – adresa v datovém souboru pro všechny n-tice v nesetříděném datovém souboru. Řídký index naopak ukládá dvojici klíčová hodnota – adresa na první položku v bloku setříděného datového souboru. Víceúrovňový index vytváří další index na existující index nebo indexy. Efektivnější podpory přístupu je docíleno stromových organizací ISAM a hlavně dynamického vyváženého B-stromu, jehož nelistové uzly obsahují n klíčů a n+1 odkazů a listy stromu odkazy do datového souboru. Obsah uzlu kolísá mezi polovičním a úplným obsazením. Pro dotazy, ve kterých figuruje požadavek na rozsah hodnot je výhodné použít B+ stromy se zřetězením listů podle požadovaného uspořádání. Pro dotazy na izolované hodnoty se naopak hodí hašování, kde hašovací funkce podle klíčového parametru určí adresu uložení i vyhledání v datovém souboru. Dynamické hašování umožňuje přizpůsobovat velikost adresového prostoru aktuálním požadavkům. Velice efektivní podpora rychlosti zpracování souvisejících dat spočívá ve shlukování, tj. uložení souvisejících dat do jednoho prostoru – bloku nebo sousedících bloků.

Pojmy k zapamatování

- Sekvenční soubor, soubory s proměnnou délkou záznamu
- hustý index, řídký index, víceúrovňový index, B-strom, ISAM
- statické a dynamické hašování

Kontrolní otázky

33. *Jaké jsou vlastnosti jednotlivých druhů paměti?*
34. *Jaké jsou formy organizace záznamů?*
35. *Co je to index a jaké druhy indexů znáte?*
36. *Jaké vlastnosti má B-strom?*
37. *Čím se odlišují indexování a hašování?*
38. *Jak pracuje Faginovo hašování?*
39. *Jak jsou organizovány soubory s proměnnou délkou záznamu?*
40. *Co je to shlukování*

6 Zpracování dotazu

Studijní cíle: Po zvládnutí kapitoly bude studující schopen popsat jednotlivé fáze zpracování a optimalizaci dotazu a odpovídající softwarové moduly, popsat přepisovací pravidla algebraické optimalizace, způsoby reprezentace výrazu dotazu a optimalizační heuristiky. Seznámí se s vybranými statistikami databáze a jejich využitím pro optimalizaci, dále s možnostmi implementace klíčových databázových operací

Klíčová slova: Optimalizace dotazu, algebraické přepisování, heuristika optimalizace dotazu, databázová statistika, implementace operací,

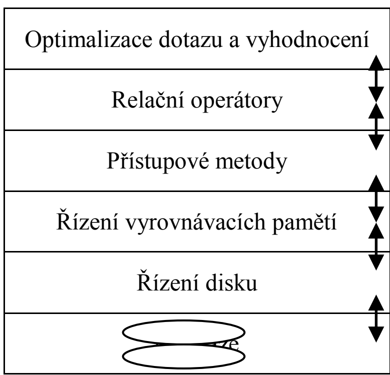
Potřebný čas: 2 hodiny

Zpracování dotazu je rozdílné podle typu SŘBD. Hierarchické a síťové databáze mají DMJ nižší úrovně a optimalizaci řeší aplikační programátoři. Relační systémy mají dotazovací jazyk vyšší úrovně a optimalizaci provádí SŘBD. Modul pro zpracování dotazu – dotazový procesor zahrnuje komponenty, které optimalizaci provádí ve dvou fázích – logické a fyzické. V logické části se transformuje dotaz v interní formě (relační algebra) na ekvivalentní výraz s efektivnějším vyhodnocením. Výraz dotazu určuje posloupnost databázových operací a ve fyzické fázi optimalizace se určuje, jakým způsobem se provedou operace v závislosti na fyzické struktuře uložení a podpoře. Optimalizace se podílí rozhodujícím způsobem na komerčním úspěchu SŘBD, protože rozhoduje o rychlosti odezvy DBS. Ve výrazu dotazu jsou použita logická jména tabulek a sloupců, pohledů, … , čili se předpokládá uživatelova znalost schématu databáze.

6.1 Základní etapy

Při zpracování dotazu probíhají procesy a aktivity související se získáním informace z dat v databázi

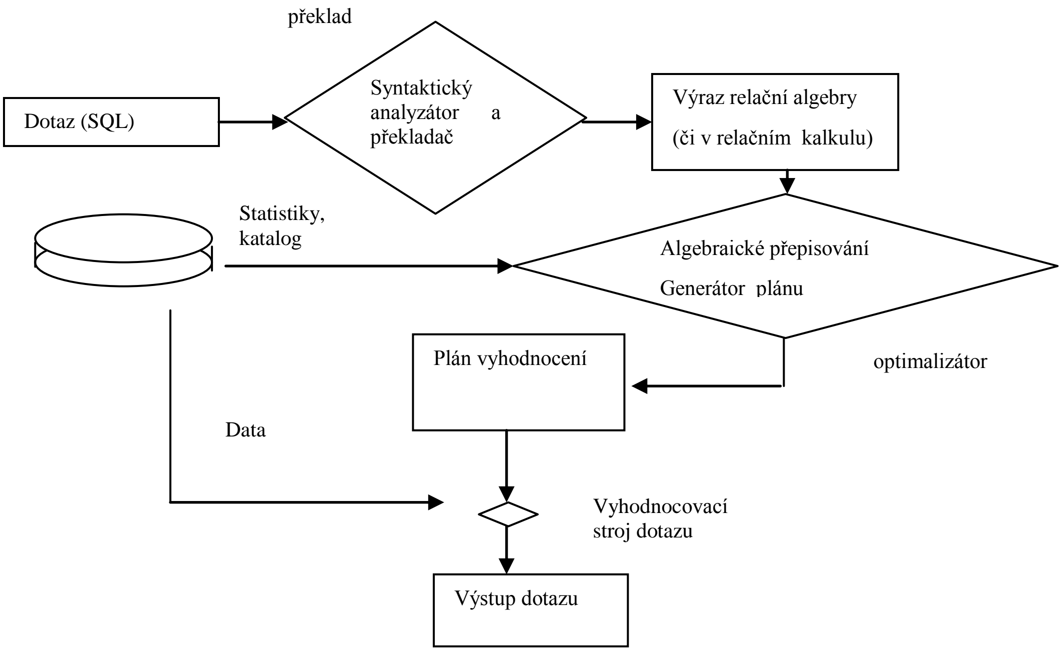
Základní postup zpracování dotazu:



1. analýza a překlad dotazu – SQL dotaz se po kontrole syntaxe vhodným lexikálním a syntaktickým analyzátorem a po ověření sémantiky dotazu - logických jmen (relací, atributů, typů …) preprocesorem pomocí informací v katalogu, se přeloží do interní formy nižšího dotazovacího jazyka (relační algebry nebo relačního kalkulu). Obvykle se provede konverze do kanonického tvaru. Forma lineárního výrazu dotazu se obvykle transformuje do stromové struktury – parse tree (listy stromu jsou zúčastněné relace, vnitřní uzly představují operace relační algebry, kořen pak dává výsledek dotazu). Pokud výraz projde všemi kontrolami, vygeneruje se z něj výchozí logický plán dotazu.

2. optimalizace – hledání nejrychlejšího plánu vyhodnocení s dosažitelnými zdroji (statistiky o databázi, omezení hlavní a vyrovnávacích pamětí, podpůrné indexy, …), jinak formulováno – systém hledá optimální pořadí operací, které provádí dotaz a vybírá nejvhodnější metodu přístupu do databáze – např. existující primární nebo sekundární indexy na jedno nebo vícesloupcové položky (případně dočasně vytvořené pro účel optimalizace přístupu), hašování, … . Pro optimalizaci se využívají i statistiky, získané z katalogu ze subsystémů uložení dat na disku. Optimalizace má tedy dvě fáze: logickou – algebraické přepisování a fyzickou – volba algoritmů.
3. vyhodnocení – vyhodnocovací stroj převezme prováděcí plán vyhodnocení – optimální pořadí operací relační algebry a na základě nejvýhodnější implementace operací vyhodnotí dotaz. Faktory ovlivňující rychlost vyhodnocení jsou např.:

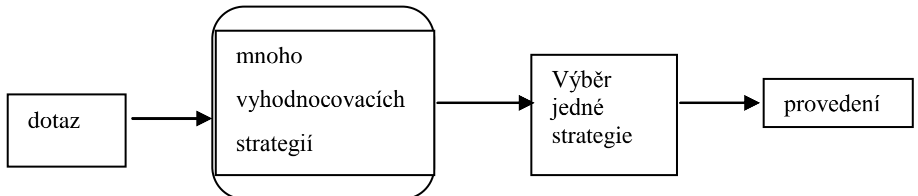
- Fyzická organizace uložených záznamů v relaci
- Zda diskový blok obsahuje záznamy jedné relace nebo několika (shlukování)
- Použité algoritmy operací (hlavně časově náročného spojení)



Obr. 14 Etapy optimalizace dotazu

6.2 Optimalizace dotazu

Výraz relační algebry se dá vyhodnotit obecně mnoha způsoby. V první fázi se typicky generují ekvivalentní výrazy k výrazu, vygenerovanému překladačem vstupního SQL dotazu. Ekvivalentní výrazy jsou různé tvary výrazu dotazu v relační algebře, které po vyhodnocení vygenerují stejná výstupní data. Podle konceptu by se měly systematicky na každý nový podvýraz opakovaně aplikovat všechna dále uvedená pravidla, aby se postupně vygenerovaly všechny ekvivalentní výrazy. Tento postup je velmi časově i prostorově náročný. Prostor se dá ušetřit sdílením podvýrazů, čas se šetří tím, že se negenerují všechny výrazy.



Ekvivalentní výrazy se generují v přepisovacích systémech pomocí ekvivalencí – přepisovacích pravidel, jako např. :

Komutativita umožňuje optimální pořadí v sekvencích operací

1. Komutativita operací - spojení a kartézského součinu, sjednocení, průniku

$E_1 \bowtie E_2 = E_2 \bowtie E_1 \qquad E_1 \cup E_2 = E_2 \cup E_1$

$E_1 \times E_2 = E_2 \times E_1 \qquad E_1 \cap E_2 = E_2 \cap E_1$

2. Asociativita operací - spojení a kartézského součinu, sjednocení, průniku

$(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$

$(E_1 \times E_2) \times E_3 = E_1 \times (E_2 \times E_3)$

$(E_1 \cup E_2) \cup E_3 = E_1 \cup (E_2 \cup E_3)$

$(E_1 \cap E_2) \cap E_3 = E_1 \cap (E_2 \cap E_3)$

Kaskáda projekcí umožňuje heuristiku ‘ co nejdříve’ projekci v sekvencích operací

3. Kaskáda projekcí

$\pi_{A1, A2, \ldots, An} (\pi_{B1, B2, \ldots, Bm} (E)) = \pi_{A1, A2, \ldots, An} (E)$

Předpoklad : $\{B_1, B_2, \ldots, B_m\} \supseteq \{A_1, A_2, \ldots, A_n\}$

Kaskáda selekcí umožňuje heuristiku ‘ co nejdříve’ selekci v sekvencích operací

4. Kaskáda selekcí

$\sigma_{F1} (\sigma_{F2} (E)) = \sigma_{F1 \wedge F2} (E)$

5. Komutace selekce a projekce

$\pi_{A1, \ldots, An} (\sigma_F (E)) = \pi_{A1, \ldots, An} (\sigma_F (\pi_{A1, \ldots, An, B1, \ldots, Bm} (E)))$

Při splnění předpokladu může projekce předcházet selekci

Předpoklad : *F obsahuje všechny $B_1, \ldots, B_m$*

6. Komutace selekce a kartézského součinu

$\sigma_F (E_1 \times E_2) = \sigma_{F1} (E_1) \times \sigma_{F2} (E_2)$

Předpoklad : $F = F_1 \wedge F_2$ *F_1 obsahuje jen atributy E_1*

F_2 obsahuje jen atributy E_2

7. Komutace selekce a sjednocení

$\sigma_F (E_1 \cup E_2) = \sigma_F (E_1) \cup \sigma_F (E_2)$

8. Komutace selekce a rozdílu

$\sigma_F (E_1 - E_2) = \sigma_F (E_1) - \sigma_F (E_2)$

9. Komutace projekce a kartézského součinu

   $\pi_{A1, \ldots, An} (E_1 \times E_2) = \pi_{B1, \ldots, Bm} (E_1) \times \pi_{C1, \ldots, Ck} (E_2)$

   B_i : Atributy E_1 $\{B_i\} \cup \{C_i\} = \{A_i\}$

   C_i : Atributy E_2

10. Komutace projekce a sjednocení

   $\pi_{A1, \ldots, An} (E_1 \cup E_2) = \pi_{A1, \ldots, An} (E_1) \cup \pi_{A1, \ldots, An} (E_2)$

Průvodce studiem

Jsou dvě základní techniky optimalizace dotazu, které se v praxi často kombinují. První technika využívá různých formulací téhož dotazu a určení optimální strategie pomocí minimalizace relativní cenové funkce, která zohledňuje časové nároky a využití zdrojů. Druhá technika využívá formulací osvědčených heuristických pravidel pro uspořádání operací v plánu dotazu.

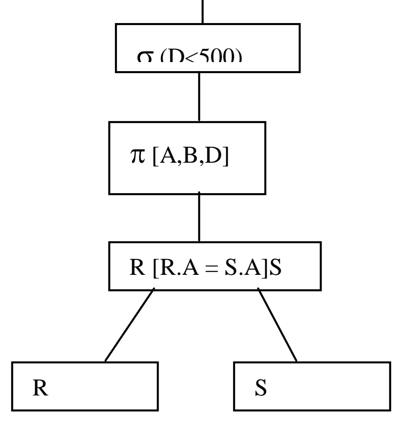
Často se aplikují doporučení, která téměř vždy vedou k efektivnějšímu výrazu, které nazýváme heuristikou. Heuristika optimalizace výrazu relační algebry – např. aplikuj pravidla tak, aby operace selekce a projekce byly provedeny co nejdříve. Pokud v dotazu je eliminace duplikátů, proveď co nejdříve.

Příklad:

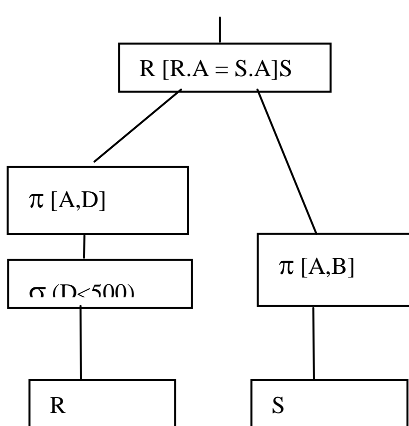
SELECT S.A S.B, R.D FROM R JOIN S ON S.A = R.A WHERE R.D < 500

se dvěma řešeními po algebraickém přepisování.

Dle heuristiky horší



Dle heuristiky lepší



Uspořádání výrazů podle efektivity vyhodnocení určuje cena dotazu – kriteriální funkce, která zohledňuje hlavně cenu I/O přístup bloků dat na disku (počet přístupů na disk), nebo jemněji čas procesoru (databáze v operační paměti), nebo cena komunikace u distribuovaných systémů. Cena algoritmů je ovlivněna i velikostí vyrovnávacích pamětí a operační paměti, která je

procesu k dispozici – větší vyrovnávací paměť zmenšuje potřebu diskových operací. Pro nalezení optimálního prováděcího plánu se používají oba přístupy -

1. Naleznou se všechny ekvivalentní plány a podle ceny se vybere nejlepší
2. Podle heuristik se odvodí nejlepší plán

Často se na odhadu ceny podílí i statistiky z katalogu dat o velikosti relací a indexů i vybrané parametry z fyzického uložení dat. Různé podmínky jsou určeny pro sloupce v selekčních a spojovacích predikátech, pokud jsou primárním, sekundárním nebo hašovacím klíčem, existuje obyčejný index, nebo typu cluster. Předpokladem je rovnoměrné rozložení hodnot A v R[A]. Formulace cenových kritérií může probíhat v několika krocích, nejprve se vyčíslí selektivity každého predikátu, potom se postupně určují odhady kardinality dílčích mezivýsledků až nakonec určíme cenu dotazu. Mezi sledované veličiny patří například :

n_R - počet n-tic v relaci R

Optimalizace v ideálním případě hledá nejlepší plán dotazu, prakticky ale většinou odmítá plány špatné.

p_R - počet stránek na disku, které obsahují n-tice relace R

b_R - počet bloků na disku, které obsahují n-tice relace R

s_R - velikost n-tice v bytech

f_R - blokovací faktor- počet n-tic relace R v bloku

V(A,R) – počet různých hodnot atributu A v relaci R

M – počet stránek volného prostoru v RAM

p_{RA} - počet listových stránek B^+ stromu indexu pro R.A

I_{AR} - počet úrovní B^+ stromu indexu pro R.A

…

Příklad odhadu velikosti výsledku pro dotaz typu

SELECT A1, …,Am FROM $R_1,\ldots, R_N$ WHERE term1 AND … AND term k

Maximální počet n-tic n_{max} je dán součinem kardinalit relací $R_1,\ldots, R_N$. Počet n-tic výsledku odhadneme vynásobením n_{max} a všech redukčních faktorů. Typy redukčních faktorů podle tvaru termů :

- A= hodnota   redukční faktor F :   1/V(AR)
- A1= A2   redukční faktor F :   1/ max(V(A1 R_1) , V(A2 R_2))
- A1>A2   redukční faktor F :   1/ 3
- A1>hodnota F :(max hodnota A1 - hodnota)/ (max hodnota A1 - min hodnota A1) pro aritmetické typy, jinak   1/ 3

Pro další typy výrazů můžeme odvozovat podle pravidel:

1) Term 1 AND Term 2

   (předpokládáme, že hodnoty různých sloupců jsou nezávislé)

   F = F (Term 1) * F (Term 2)

2) Term 1 OR Term 2

   F = F (Term 1) + F (Term 2) – F (Term 1) * F (Term 2)

3) NOT Term 1

   F = 1 – F (Term 1)

Průvodce studiem

Cenová funkce využívá k odhadu ceny statistických informací ze systémového katalogu. Typické statistiky zahrnují informace o kardinalitě a stupni relací, počtu bloků, počtu úrovní indexů, počtu různých hodnot v doméně atributu a podobně.

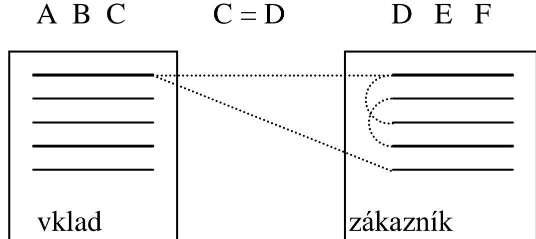
6.3 Implementace operací, metody pro výpočet spojení

Relační SŘBD nabízí typicky implementace několika algoritmů každé operace a je na dotazovém procesoru, kterou variantu v rámci optimalizace vybere. Předpokládejme uložení relací jako nezávislých záznamů ve fyzických stránkách, s podporou přístupu pomocí indexů nebo hašování.

Operace selekce, kdy predikát má tvar A = hodnota, při sekvenčním vyhledávání má cenu p_R pro nejhorší případ, průměrně p_R /2 je-li A primární klíč. Pro binární vyhledávání při uspořádání R podle A , když A je primárním klíčem je cena rovna $\log_2 (p_R)$. Pokud existuje primární index, je cena rovna I(A) + 1, …, konečně pro hašovaný atribut A stačí přístup jeden. Podobně můžeme definovat ceny pro predikát ve tvaru A< hodnota a další složitější kombinace a tvary.

Pro ilustraci použijeme některé metody pro výpočet časově nejnáročnější operace - spojení.

6.3.1 Hnízděné cykly (Nested-loop join)



Algoritmus můžeme symbolicky popsat

```
for each záznam v in vklad  do
       begin
       for each záznam z in zákazník do
              begin
              test páru (v, z) na shodu hodnot ve sloupcích C a D
              pokud ano, vloží se (v,z) do výsledku
              end
       end
```

Příklad odhadu ceny spojení pro hnízděné cykly s ilustrací záměny pořadí relací. Zvolme parametry databáze :

vklad : 10 000 n-tic, 500 bloků (f_R = 20)

zákazník: 200 n-tic, 10 bloků (f_R = 20)

Cena při použití relace vklad ve vnějším cyklu a zákazník ve vnitřním cyklu

- Čtení relace vklad : 500 přečtených bloků
- Pro každou n-tici relace vklad, čteme celou relaci zákazník :

  10 bloků × 10 000 n-tic = 100 000 přečtených bloků
- Celková cena je 100 500 přečtených bloků

Cena při použití relace zákazník ve vnějším cyklu a vklad ve vnitřním cyklu

- Čtení relace zákazník : 10 přečtených bloků
- Pro každou n-tici relace zákazník, čteme celou relaci vklad :

  200 × 500 = 100 000 přečtených bloků

  Celková cena je 100 010 přečtených bloků

Ke snížení ceny vede cesta přes využití všech n-tic v načteném bloku.

```
for each blok Bv relace vklad do
       begin
       for each blok Bz relace zákazník do
              begin
                     [hnízděné cykly s n-ticemi v Bv a Bz]
              end
       end
```

Cena při použití relace vklad ve vnějším cyklu a zákazník ve vnitřním cyklu

čtení bloků relace vklad : 500 přečtených bloků

pro každý blok relace vklad čteme celou relaci vnitřního cyklu :

500 × 10 = 5000 přečtených bloků

Celková cena je 5500 přečtených bloků

6.3.2 Indexované hnízděné cykly

Použití indexů na jednu nebo obě relace může výrazně snížit cenu. Místo procházení celé relace ve vnitřním cyklu získáme požadované n-tice efektivně. Často používaná strategie je

Setřídění-slévání (Sort-Merge Join)

Postup můžeme formulovat následovně

1. setřiď první relaci (vklad) podle sloupce ve spojovací podmínce (C)
2. setřiď druhou relaci (zákazník) podle sloupce ve spojovací podmínce (D)
3. Postupně pro každou n-tici z první relace připoj všechny n-tice z druhé relace se stejnou hodnotou ve sloupci spojovací podmínky.

Cena : setřídění první a druhé relace

+ jedno načtení bloků první a druhé relace

6.3.3 Hašováné spojení (Hash Join)

Složitost algoritmu je O(n), kde n je počet n-tic.Princip: hašovací funkce s parametrem sloupce spojovací podmínky rozdělí n-tice spojovaných relací na disjunktní podmnožiny a pouze n-tice v odpovídajících si podmnožinách se propojí, tj. platí $h(c) = h(d)$, když $c = d$, kde c, d jsou

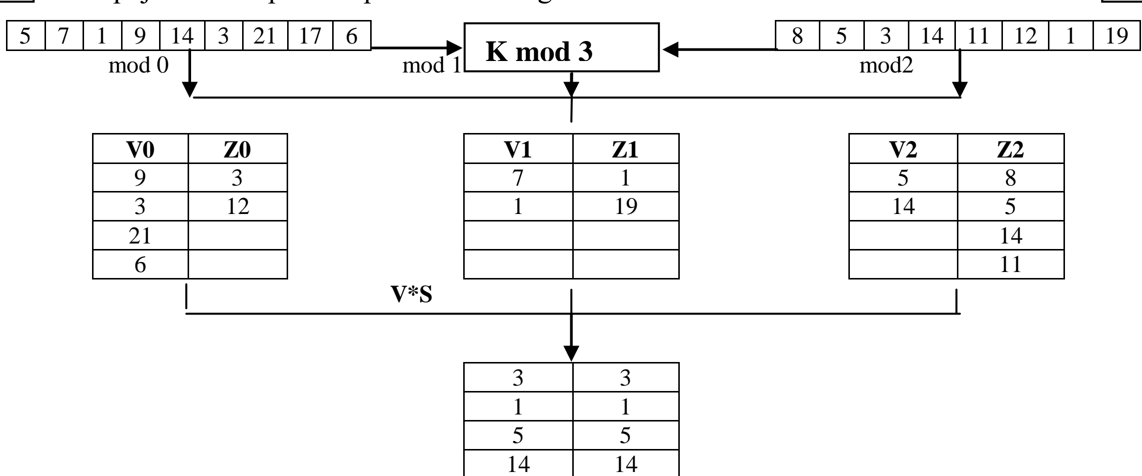
hodnoty spojovacích atributů. Spojení se potom realizuje pomocí mnoha menších spojení v rámci podmnožin, jejichž velikost zaručí celé provedení v operační paměti.

Postup pro klasické hašování (relace se vejde do operační paměti)

1. Zahašuj první relaci do vnitřní paměti(relaci vklad podle sloupce C).
2. Čti druhou relaci (zákazník) sekvenčně. Hašuj podle spojovacího sloupce (D) a přímým přístupem najdi n-tici nebo n-tice první relace.
3. Zkontroluj rovnost spojovacích sloupců (C= D), pokud vyhovuje, vlož spojené n-tice do výsledku.

Pro případ, kdy se relace nevejde do operační paměti, postupujeme podobně, ale ve dvou fázích

1. Rozděl první i druhou relaci pomocí vhodné hašovací funkce s parametrem spojovacích sloupců. Vzniknou disjunktní podmnožiny obou relací.
2. Sobě odpovídající podmnožiny obou relací, které se již vejdou do operační paměti, spojíme třeba pomocí předchozího algoritmu. Příklad



6.3.4 Vícenásobné spojení

Hledejme optimální pořadí spojení množiny S spojení S1 * S2 * … * Sn. Existuje

(2(n – 1))!/(n – 1)! možných kombinací. Pro nalezení optimálního pořadí se používají metody dynamického programování.

Uvažujeme všechny možné plány ve formě S1 * (S – S1), kde S1 je libovolná neprázdná podmnožina S. Rekurzívně počítáme cenu spojení podmnožin . Vybereme nejlepší z $2^n – 1$ alternativ. Algoritmus zjednodušeně popisuje následující procedura:

```
procedure najdinejplan(S)
if (nejplan[S].cena≠ max

       return nejplan[S]
// else nejplan[S] když není vyčíslen
for each non-empty subset S1 of S such that S1 ¹ S
       P1= najdinejplan(S1)
       P2= najdinejplan(S - S1)
       A = best algorithm for joining results of P1 and P2
       cena = P1.cena + P2.cena + cena A
       if cena < nejplan[S].cena
              nejplan[S].cena = cena
              nejplan[S].plan = „execute P1.plan; execute P2.plan;
                                  join results of P1 and P2 using A”
return nejplan[S]
```

– některé realizace pořadí spojování relací(A,B,C,D)

1) sekvence binárních spojení

((A * B) * C) * D
((A * B) * D) * C
((A * C) * B) * D

} 4 ! = 24 možností

2) strom binárních spojení (A * B) * (C * D)

Průvodce studiem

Hlavní strategie při implementaci spojení jsou hnízděné cykly, indexované hnízděné cykly, setříděné – slévané cykly a hašovaná spojení.

6.3.5 Další operace

Eliminace duplikátních záznamů se provádí pomocí hašování nebo tříděním. Využívá se seskupení záznamů a ve vhodné fázi algoritmu se ruší duplikáty. Podobně se zpracovávají agregační funkce. Pro count, min, max, sum se agregovaná hodnota získá postupným procházením záznamů ve skupině, pro avg se počítá sum a count a nakonec se tyto hodnoty podělí.

6.3.6 Vyhodnocení stromového výrazu dotazu

Používají se dvě alternativy:

1. Materializace (Materialization) – vytváří výsledek dotazu z výrazu dosazením relací do listů stromu a postupným prováděním operací od listů ke kořenu. Aktuálně běží vždy jen jedna operace a mezivýsledky se ukládají do dočasných tabulek v paměti nebo na disku. Při odhadu ceny dotazu se musí vzít v úvahu i přístupy na disk při manipulaci s mezivýsledky. Další možností je použití dvou vyrovnávacích pamětí pro každou operaci, když je jedena paměť plná, uloží se na disk a pracuje se s druhou.
2. Pipelining – vyhodnocuje simultánně několik operací tak, že výsledek jedné je vstupem další. Tento způsob je efektivnější, ale nedá se použít vždy(při třídění, …).

Průvodce studiem

Při materializaci je výstup jedné operace uložen do dočasné relace pro následné zpracování v další operaci – výhodné při znovupoužití mezivýsledků. Alternativně, při přístupu pipeline je výsledek jedné operace použit přímo v druhé operaci, bez vytvoření dočasných relací, nebo mezivýsledků, což šetří čas na vytváření těchto relací a čtení z disku.

Shrnutí Dotaz je při zpracování přeložen z SQL do interní formy logického plánu dotazu s použitím výrazů relační algebry nebo kalkulu. Následuje optimalizace a potom provedením fyzického plánu - operací relační algebry z výrazu dotazu se vyhodnotí. Data ze souborů je možné zpřístupnit mnoha způsoby. Celé tabulky se jednoduše načítají sekvenčně po blocích, při vyhledávání v intervalu dat nebo třídění je výhodné použít indexy, pro jednotlivé vyhledávané hodnoty potom hašování. Výhodnost použité metody určuje cena, kterou měříme většinou počtem přístupů na disk při vyhodnocení dotazu. Metoda hnízděných cyklů je použitelná při provádění operací s argumenty v operační paměti. Při práci s relacemi na disku je výhodná implementace s podporou indexů nebo hašování.

Pojmy k zapamatování

- algebraické přepisování, heuristika optimalizace dotazu
- databázové statistiky

Kontrolní otázky

41. *Jakými procesy prochází zpracování dotazu v relačních DBS?*
42. *Jaká jsou pravidla algebraického přepisování?*
43. *Jaké veličiny tvoří databázové statistiky?*
44. *Jak se může implementovat operace spojení?*

Úkoly k textu

Vytvořte několik ekvivalentních zápisů dotazu podle přepisovacích pravidel. Odhadněte pořadí výhodnosti jednotlivých výrazů podle ceny dotazu.

Vytvořte stromovou reprezentaci několika dotazů a aplikujte typická heuristická pravidla.

Vytvořte malou databázi a vyčíslete všechny její užitečné statistiky.

7 Formalizace návrhu relační databáze, normalizace

Studijní cíle: Student by po nastudování kapitoly měl porozumět problematice návrhu schématu relační databáze, vysvětlit na příkladech anomálie při nevhodném návrhu. Dále vysvětlit pojem a využití funkčních závislostí v algoritmech návrhu databáze. Student by měl vysvětlit pojem a důvody normalizace relací, definovat jednotlivé normální formy

Klíčová slova: Armstrongovy axiomy, neredundantní pokrytí, normalizace relací, bezeztrátová dekompozice schématu, Boyce-Coddova normální forma, multifunkční závislost

Potřebný čas: 2 hodiny

Převod konceptuálního schématu zapsaného v nějakém konceptuálním modelu není jediným způsobem, jak navrhnout relační schéma databáze. Současně se vznikem teorie relačního modelu dat vznikla také metoda návrhu relačních schémat pomocí funkčních závislostí, jejíž postupy se hodí i pro úlohy optimalizace relačního schématu, což můžeme chápat jako proces nahrazení schématu $R_1=\{R_i\}$ jinou množinou relací ve schématu $R_2 = \{ R_n\}$

tak, aby R_2 mělo dobré vlastnosti. Nejprve ukážeme některé problémy, plynoucí ze špatného návrhu.

7.1 Problémy návrhu schématu relační databáze

Motivační příklad: Uvažme relaci **R** (úloha, pracovník_ID, pracovník_jméno, pracovník_příjmení, místnost_číslo,místnost_plocha) s částí dat:

úloha	**Pracovník_ID**	**pracovník_jméno**	**pracovník_příjmení**	**místnost_číslo**	**místnost_plocha**
A1	5	Petr	Novák	3	87
A3	5	Petr	Novák	3	87
A9	5	Petr	Novák	3	87
B6	8	Jan	Holý	3	87
A3	8	Jan	Holý	3	87
C9	8	Jan	Holý	3	87
A1	8	Jan	Holý	3	87
…		…	…	…	…

Nevhodný návrh signalizuje výskyt opakujících se položek v datech, ale snadno odhalíme následující potíže :

- **redundance**, pro každého pracovníka se opakují hodnoty o místnosti, …
- nebezpečí vzniku **nekonzistence** při modifikacích jako důsledek redundance (v řádku změníme číslo a nezměníme plochu)
- **anomálie při vkládání** záznamů : nemůžeme vložit úlohu bez pracovníka, který ji řeší, neboť by nebyly obsazeny klíčové atributy,
- **anomálie při vypouštění** záznamů : přestanou-li řešit úlohy všichni pracovníci z jedné místnosti, ztratíme informaci i o její ploše.

Problém vyřešíme dekompozicí relace za pomoci funkčních závislostí.

7.2 Funkční závislosti

Funkční závislost je definována mezi dvěma podmnožinami atributů v rámci jednoho schématu relace. Jde tedy o vztahy mezi atributy, nikoliv mezi entitami.

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

Př.:Rozborem kontextu (pracovník sedí v jedné místnosti, …) můžeme určit příklady funkčních závislostí z předcházejícího příkladu

F : {místnost_číslo → místnost_plocha,

pracovník_ID → místnost_číslo,

pracovník_ID → pracovník_jméno, pracovník_příjmení, …}

Nechť X, Y jsou podmnožiny atributů schématu **R** s množinou závislostí F. Říkáme, že *Y plně funkčně 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.

$$X \rightarrow Y \qquad \Rightarrow \quad (t1[x] = t2[x] \Rightarrow t1[y] = t2[y])$$

Průvodce studiem

Funkční závislosti používáme :

- při testování relace - zda vyhovuje množině funkčních závislostí F. O takové relaci r říkáme, že splňuje F .

- při specifikaci integritních omezení nad množinou relací, které splňují F.

- při normalizaci relací a návrhu schématu databáze

- při určování kandidátních klíčů

Pro účely odvozování nových závislostí byl Armstrongem navržen formální systém, který se nazývá Armstrongovy axiomy.

7.2.1 Armstrongovy axiomy

Nechť A je množina atributů daného relačního schématu, F množina funkčních závislostí mezi atributy A. V následujících pravidlech označujeme sjednocení X ∪Y jako XY. 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+). Následující odvozovací pravidla nejsou v minimálním tvaru :

A1 : jestliže Y⊂ X⊂ A, pak F logicky implikuje X → Y (reflexivita, triviální závislost)

A2 : jestliže X → Y a Z⊂ A, pak XZ → YZ (rozšíření)

A3 : jestliže X → Y a Y → Z, pak X → 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 :

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

Zavedeme pojem *uzávěr* F+ jako množinu všech funkčních závislostí, odvoditelných Armstrongovy axiomy. Souvislost funkčních závislostí a primárního klíče je dána definicí.

Nechť **R** (A1,A2,...,An) je relační schéma s množinou funkčních závislostí F, nechť $X \subset \{A1,A2,...,An\}$. Řekneme, že X je klíč schématu **R**, jestliže

$X \rightarrow A1...An \subset F+$

pro každou vlastní podmnožinu $Y \subset X$ je $Y \rightarrow A1...An \not\subset F+$.(podmínka minimality)

Zřejmě můžeme klíč schématu definovat také jako takovou $X \subset A$, že A je úplně závislá na X.

Platí, že F lze nahradit závislostmi, které vzniknou dekompozicí pravých stran závislostí na jednotlivé atributy. Závislost, která má na pravé straně pouze jeden atribut, nazýváme *elementární*. 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'. Říkáme, že **závislost f je redundantní** v F', jestliže platí

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

Odstraněním všech redundantních závislostí z F' vznikne tzv. neredundantní pokrytí F. Neredundantní pokrytí není dáno jednoznačně, závisí na pořadí, ve kterém odebíráme neredundantní závislosti. Obecně tedy nemusí být podmnožinou původní množiny F, pokud vycházíme z F+, ne z F.

Průvodce studiem

Pro odvození uzávěru množiny funkčních závislostí postačí následující Armstrongovy axiomy:

1. *jestliže $Y \subseteq X$, potom $X \rightarrow Y$* ***(reflexivita)***
2. *jestliže $X \rightarrow Y$, potom $ZX \rightarrow ZY$* ***(rozšíření)***
3. *jestliže $X \rightarrow Y$ a $Y \rightarrow Z$, potom $X \rightarrow Z$* ***(transitivita)***

Příklad : Určete neredundantní pokrytí množiny funkčních závislostí F :

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

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 redundantní, vyloučíme je v uvedeném pořadí a dostaneme výsledek:

$F1: AB \to C, C \to A, BC \to D, ACD \to B, D \to E, D \to G, BE \to C, CG \to D, CE \to G$

Jestliže zvolíme jiné pořadí při odstraňování redundantních závislostí v pořadí $CE \to A$, $CG \to D$, $ACD \to B$, obdržíme :

$F2: AB \to C, C \to A, BC \to D, D \to E, D \to G, BE \to C, CG \to B, CE \to G$

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 $X \subset A$ vzhledem k F. Tento uzávěr tvoří množina všech atributů funkčně závislých na X a označíme jej X+.

Jestliže $X \to Y$ a pro nějaké $C \in X$ platí $(X - C)+ = X+$, říkáme, že atribut C je redundantní pro zadanou závislost. Pokrytí, v jehož závislostech neexistují žádné redundantní 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 : Uvažme opět neredundantní pokrytí F1 z dřívějšího příkladu :

$F1 : AB \to C, C \to A, BC \to D, ACD \to B, D \to E, D \to G, BE \to C, CG \to D, CE \to G.$

Z $C \to A$ lze odvodit $CD \to AD$ a $CD \to ACD$. Protože $ACD \to B$, platí dále $CD \to B$. Tak získáme minimální pokrytí

$Fmin : AB \to C, C \to A, BC \to D, CD \to B, D \to E, D \to G, BE \to C, CG \to D, CE \to G$

Při eliminaci redundantních 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

$$F+ = F1+ = F2+ = Fmin+$$

Obě transformace (odstranění redundantních závislostí a redundantních atributů) nelze provádět v libovolném pořadí. Pro získání minimálního pokrytí je nutno odstranit nejprve redundantní 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 $X \to Y$ k uzávěru F+ není nutné spočítat celý uzávěr F+, ale stačí výpočet X+.

7.2.2 Určení uzávěru FZ pro podmnožiny atributů relace

Algoritmus Určení uzávěru X+.

Vstup : F nad množinou atributů A relace R(A), kde $|F| = m$, $|A| = n$, podmnožina $X \subset A$

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 k do
                                       begin
                                          if (LS[i] ⊂ UZX) and (PS[i] ⊄  UZX)
                                          then begin
                                              UZX := UZX ∪ PS[i];
                                              POKR := false
                                               end
                                        end
                                     end
                            end;
```

7.2.3 Určení příslušnosti funkční závislosti k uzávěru množiny FZ

Algoritmus 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;
```

7.2.4 Určení neredundantního pokrytí pro množinu elementárních funkčních závislostí.

Algoritmus určení neredundantního pokrytí pro množinu elementárních funkčních závislostí.

Vstup : F' nad množinou atributů A relace R(A)

Výstup : neredundantní pokrytí G

```
Algoritmus :    begin
                    G := F';
                    foreach  f ∈  F do
                    if  f  ∈ (G - {f})+   then G := G - {f}
                end
```

7.3 Dekompozice relačních schémat

Dříve uvedené nepříjemné vlastnosti relačního schématu plynou z toho, že některé atributy relací jsou funkčně závislé nejen na klíči, ale i na jeho podmnožině.Této vlastnosti relačního schématu se zbavíme vhodným rozdělením relačního schématu, tzv. dekompozicí.

Dekompozice relačního schématu R(A) je množina relačních schémat

$$R' = \{R_1(A_1), ..., R_k(A_k)\}, \quad \text{kde } A = A_1 \cup A_2 \cup \ldots \cup A_k.$$

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 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.

Formálně:

Nechť R(A) je relační schéma, R'={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 :

$$R = R[A_1] \, [*] \, R[A_2]$$

Odpověď na otázku, jak poznat binární dekompozici, při níž nedojde ke ztrátě informace nám dává následující věta o ztrátě informace:

Nechť R' = {R1(B),R2(C)} je dekompozice relačního schématu R(A), tedy $A = B \cup C$ 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 : Mějme schéma R(A,B,C), kde A,B,C jsou disjunktní podmnožiny atributů a funkční závislost F : $B \rightarrow C$. Rozložíme-li R na schémata R1(B,C) a R2(A,B), je provedená dekompozice bezeztrátová. Naopak, je-li dekompozice R1(B,C) a R2(A,B) bezeztrátová, musí platit buď $B \rightarrow C$ nebo $B \rightarrow A$.

Důsledek : platí-li $B \rightarrow C$, dekompozice R1(B,C) a R2(C,A) není bezeztrátová. Neplatí totiž ani $C \rightarrow B$, ani $C \rightarrow A$.

Dalším důležitým požadavkem na proces rozkladu je zachování funkčních závislostí. Ty představují integritní omezení původní relace a v zájmu zachování reality musí být zachovány.

Nechť R(A) je relační schéma, R' = {R1(B), R2(C)} je jeho dekompozice s množinou závislostí F. Projekcí F[A] množiny funkčních závislostí F na množinu atributů A nazveme množinu prvků $X \rightarrow Y$ z F+ takových, že $X \cup Y \subseteq A$. Řekneme, že dekompozice R' *zachovává množinu funkčních závislostí F* (zachovává pokrytí závislostí), jestliže množina závislostí $(F[B] \cup F[C])$ logicky implikuje závislosti v F, tj. $F+ = (F[B] \cup F[C])+$.

Příklad : Uvažme relační schéma ADRESA(Město, Ulice, PSČ) se závislostmi $F = \{MU \rightarrow P, P \rightarrow M\}$ a jeho rozklad R' = {UP(U,P), MP(M,P)}.

Při rozkladu R‘ nedochází ke ztrátě informace vzhledem k daným závislostem, protože $\{U,P\} \cap \{M,P\} \rightarrow \{M,P\} - \{U,P\}$, avšak rozklad R‘ nezachovává množinu závislostí F. Projekce F[U,P] obsahuje pouze triviální závislosti, projekce F[M,P] závislost $P \rightarrow M$ a triviální závislosti neimplikují $MU \rightarrow P$, 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 $MU \rightarrow P$, tj.jednoznačnost PSČ pro dané město a ulici.

S dekompozicí jsou spojeny i problémy:

Některé dotazy jsou po dekompozici časově náročnější

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: Schéma R(A,B,C,D), rozklad RO = {R1(A,B), R2(C,D)} se závislostmi $F = \{A \rightarrow B, C \rightarrow D\}$:

Některé závislosti se dají kontrolovat až po spojení dílčích relací.

$$\{A,B\} \cap \{C,D\} = \emptyset$$

$$\{A,B\} - \{C,D\} = \{A,B\}$$

$$\{C,D\} - \{A,B\} = \{C,D\}$$

Uvedené vlastnosti dekompozicí použijeme nyní k takovým rozkladům, aby vzniklá relační schémata měla optimální vlastnosti.

7.4 Normální formy relací

Normalizace směřuje k návrhu množiny relací s požadovanými vlastnostmi, ovlivněný souvislostmi mezi daty

relační model dat pracuje pouze s relacemi v 1NF.

Jestliže relační schéma obsahuje pouze atomické atributy říkáme, že je normalizovanou relací nebo že je v *první normální formě* (1NF).

Př.R1 není v 1NF - R1(A,B, R2(C,D)) $\Rightarrow$ R(A,B,C,D)

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. 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.

Relační schéma je ve *druhé normální formě* (2NF), jestliže je v první normální formě a každý neklíčový atribut je úplně závislý na každém klíči schématu R. Jinak – žádný neklíčový atribut není závislý na podmnožině žádného klíče R.

Př. R není ve 2.NF - FZ: $AB \rightarrow C$, $B \rightarrow D$; R(A, B, C, D) $\Rightarrow$ R1(A, B, C), R2(B, D)

Schémata ve 2NF však mohou mít další typ anomálií, podobných předchozím, avšak z poněkud jiných příčin. Uvažme příklad :

TŘÍDA(ČísMístnosti, ČísBudovy, Patro, PočetLavic, Plocha)

Relační schéma UČITEL(ČU, Jméno, Plat, Funkce) s jediným klíčem ČU je ve 2NF. Uvažme další, z reálného světa odpozorovanou, funkční závislost Funkce $\rightarrow$ 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 např. 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.

Nechť X,Y,Z jsou podmnožiny množiny atributů A schématu R, nechť platí Y X, X Z = O, Y Z = O a nechť F je množina závislostí. Jestliže $X \rightarrow Y$ F, $Y \rightarrow Z$ F+ a $Y \rightarrow X$ F+, pak také $X \rightarrow Z$ F+ a $Z \rightarrow X$ F+ a říkáme, že Z je *tranzitivně závislá* na X.

Relační schéma je ve *třetí normální formě* (3NF), jestliže je ve 2NF a žádný atribut, který není primární, není tranzitivně závislý na žádném klíči schématu R. Relace ve 2NF, které nejsou ve 3NF, se mohou rozložit vhodnou dekompozicí do 3NF, přičemž nedochází ke ztrátě informace a dekompozice zachovává množinu závislostí.

Př. R není ve 3.NF - FZ: $A \rightarrow BC$, $C \rightarrow D$; R(A, B, C, D) $\Rightarrow$ R1(A, B, C), R2(C, D)

Dalším omezením třídy relací ve 3NF - vyloučením všech netriviálních funkčních závislostí mezi atributy kromě závislosti na klíči - dostaneme relace v Boyce-Coddově normální formě.

BCNF zaručuje potlačení redundance

Relační schéma R(A) s množinou funkčních závislostí F je v *Boyce-Coddově normální formě* (BCNF), jestliže vždy, když $X \rightarrow Y$ a $Y \notin X$, pak X obsahuje klíč schématu R.

Př. R není ve BCNF - FZ: $AB \rightarrow CD$, $D \rightarrow B$; R(A, B, C, D) $\Rightarrow$ R1(A, D, C), R2(D, B)

Každé schéma v BCNF je zároveň ve 3NF. Jinak by existovala buď závislost $Y \rightarrow Ai$, kde Y je vlastní podmnožinou klíče a Ai je sekundární atribut, nebo tranzitivní závislost $X \rightarrow Y \rightarrow Ai$, přičemž nepatí $Y \rightarrow X$. V obou případech Y neobsahuje klíč a to je ve sporu s tím, že schéma je v BCNF.

Příklad : Schéma ADRESA(Město, Ulice, PSČ) je ve 3NF, ale není v BCNF, protože existuje závislost $P \rightarrow M$. V této relaci nemůžeme např. uvést údaj o tom, že dané PSČ patří k danému městu, aniž uvedeme ulici. Přitom existence závislosti $P \rightarrow M$ plyne z reality. Tento problém se opět řeší dekompozicí, ovšem s jistými výhradami.

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;

```
3NF(R, F)
      Fc je kanonické pokrytí F;
i := 0;
for each  funkční závislost X → Y v Fc do
       if žádná schémata Rj, 1 ≤ j  ≤ i neobsahují XY
             then begin
                          i := i  + 1;
```

```
                    Ri  := XY
                end
if žádná schémata Rj, 1 ≤ j  ≤ i neobsahují kandidátní klíč R
    then begin
                i := i  + 1;
                Ri := kandidátní klíč R;
            end
return (R1, R2, ..., Ri)
```

- 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. Algoritmus je popsán v další kapitole.

Příklad : Uvažujme schéma ADRESA(M,U,P). Platí zde dvě netriviální funkční závislosti :

$$MU \rightarrow P,\ P \rightarrow M$$

Protože neplatí ani $U \rightarrow P$, ani $M \rightarrow P$, je {M,U} klíčem schématu, klíčem je také {U,P}. Platí totiž $P \rightarrow M$, ale neplatí $P \rightarrow U$, proto musíme připojit k PSČ i ULICI. Schéma tedy nemá žádný neklíčový atribut a je ve 3NF, není však v BCNF. Nemůžeme evidovat města a PSČ, aniž bychom znali nějakou ulici ve městě. Adresář se musí nahradit schématy

ULICE(U,P),   MĚSTA(P,M).

Podmínkou pro dekompozici do BCNF však je, aby dekompozice obsahovala i samotné schéma ADRESA (pro vyjádření $MU \rightarrow P$), které není v BCNF, tedy řešení neexistuje. Jinak - pokud dekompozice schéma ADRESA neobsahuje - pak projekce závislostí schématu ADRESA na množinách atributů schémat rozkladu neimplikují závislost $MU \rightarrow P$ schématu ADRESA.

Závěrem uvedeme ještě jiný typ závislosti, než funkční závislost, tzv. multizávislost.

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 $X \rightarrow> Y$.

Příklad : Mějme relaci s katalogem automobilových modelů AUTA s atributy MODEL, ZEMĚ PŮVODU, POČET VÁLCŮ:

AUTA	MODEL	ZEMĚ_PŮVODU	POČET_VÁLCŮ
	Škoda	ČR	4
*	Mustang	USA	6
*	Mustang	Kanada	4
**	Mustang	USA	4
*	Mustang	Kanada	6
	Monaco	Kanada	6

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 ZEMI PŮVODU, ale multizávisí na MODELu,ovšem nezávisle na zemi výroby. Model Mustang 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 :

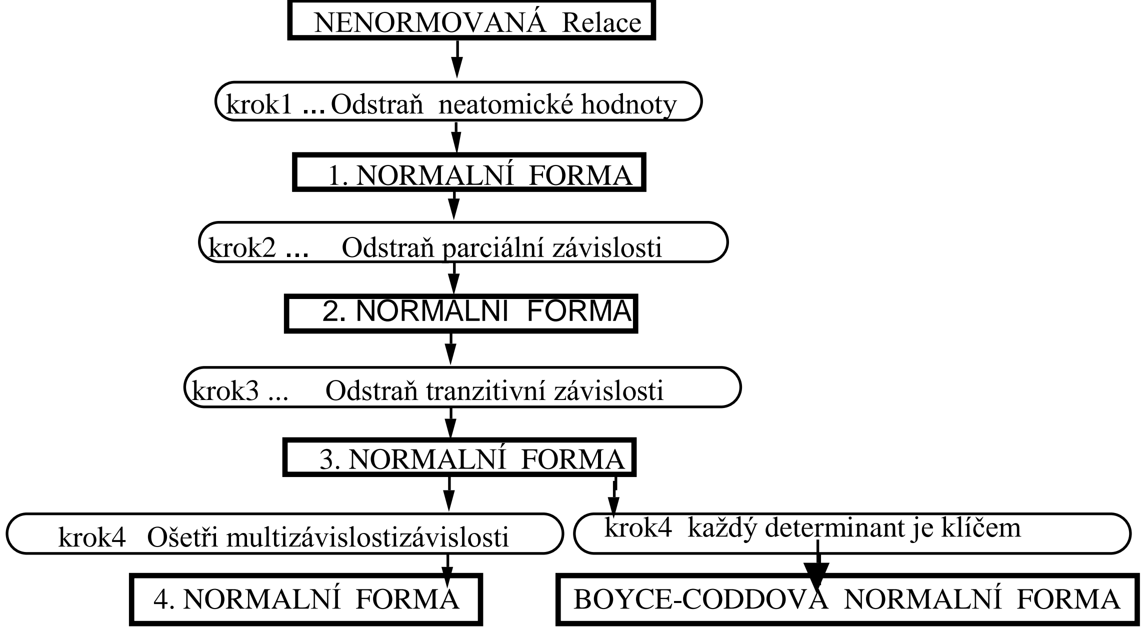
AUTA (MODEL, POČET_VÁLCŮ)

ODKUD (MODEL, ZEMĚ_PŮVODU)

Jakmile přestanou v USA vyrábět Mustangy se 4 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.

Multifunkční závislost X $\rightarrow>$ Y je netriviální, když žádné $A_i \in Y$ není částí X a atributy v X a Y nepokrývají celou relaci R.

Pokud je relace R v BCNF, říkáme, že je ve *čtvrté normální formě (4NF),* když multifunkční závislosti ve tvaru X $\rightarrow>$ Y je netriviální a X je nadklíč relace R .



Obr. 15 Postup normalizace relací

7.5 Postup návrhu schématu relační databáze

Závěrem si popíšeme dva algoritmy návrhu schématu relační databáze:

- algoritmus dekompozice relačních schémat (též shora dolů)
- algoritmus syntézy relačních schémat (též zdola nahoru).

V obou případech jde vlastně o dekompozici univerzálního schématu relace. První algoritmus postupně nahrazuje jedno schéma dvěma, druhý vytváří relační schémata 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. Např. je-li atribut ZNAMKA vyhrazen pro známku studenta u zkoušky, nesmí se použít např. také pro hodnocení kvalifikace učitele. Existuje-li tedy ve více relacích stejné jméno atributu, mají všechny tyto atributy stejnou doménu.
- 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. Např. relace VEDOUCÍ_PROJEKTU (UČITEL, STUDENT) a UČÍ (UČITEL, PŘEDNÁŠKA, 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 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 R = {Ri(Ai,Fi)}, 1 <= i <= n

Struktury dat : VYSLED obsahuje po skončení algoritmu schéma R, POKR je typu Boolean

```
Algoritmus :  begin
                   vysled:={R};
                   POKR:=false;
                   Vytvoř F+;
                   while (not POKR) do
                      if v VYSLED existuje Ri, které není v BCNF
                      then begin
          pro netriviální funkční závislost X → Y∈ Ri(Ai), že X → Ai ∉ F+
proveď
        VYSLED := (VYSLED - Ri(Ai)) ∪ Ri(Ai - X) ∪ Rj(XY)
                           end
                       else POKR:=true
                       end
```

Další algoritmus syntézy (Bernsteinův) vychází stejně jako dekompozice z dané množiny atributů a funkčních závislostí :

1. vytvoří se minimální pokrytí
2. závislosti se roztřídí do skupin tak, aby v každé skupině byly závislosti se stejnou levou stranou

3. atributy závislostí každé skupiny vytvoří schéma jedné relace, vzniklého syntézou; atributy levé strany tvoří klíč vzniklého schématu
4. 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
5. atributy, které se nevyskytují v žádné z funkčních závislostí F buď umístíme do samostatné relace, nebo je připojíme k některému ze schémat.

Výsledkem je databázové schéma ve 3NF se zachováním závislostí a lze zajistit i bezeztrátovost.

Protože i na syntézu se můžeme dívat jako na jakousi dekompozici univerzálního schématu, otázkou dále je, zda lze zajistit bezeztrátovost výsledného rozkladu. K zajištění bezeztrátovosti stačí připojit k výsledku další relační schéma s množinou atributů K - klíčem příslušného univerzálního schématu. Je-li K podmnožinou některého z dosavadních schémat, nové schéma se nevytváří.

Shrnutí Při nevhodném návrhu databázového schématu se v datech objevují opakující se položky (redundance), což může při nevhodné manipulaci s daty vést k nekonzistenci, nebo ke ztrátě informací. Funkční závislost mezi atributy relace vyjadřuje situaci, kdy pro dvě n-tice relace platí, že pokud se rovnají jejich hodnoty v jedné podmnožině atributů (levá strana pravidla), potom se rovnají i hodnoty v druhé podmnožině atributů (pravá strana pravidla). Funkční závislosti můžeme odvozovat podle Armstrongovych axiomů a dopracovat se až k tranzitivnímu uzávěru množiny funkčních závislostí. Prakticky je výhodné udržovat minimální množinu FZ, ze kterých se dá odvodit stejný tranzitivní uzávěr. FZ mohou sloužit pro hledání kandidátních klíčů, formulaci sémantiky dat a integritních omezení a v kontextu této kapitoly mohou sloužit pro návrh schématu databáze dosažením co nejvyšší normální formy relací, čehož je dosaženo vhodnou dekompozicí schématu relací. Předpokladem správné dekompozice je požadavek jednoznačného vytvoření původní relace z dekomponovaných a zároveň zachování množiny funkčních závislostí. Splnění 1.NF zaručuje atomicitu dat, 2.NF zaručuje **úplnou** funkční závislost neklíčových atributů na klíčových (neexistuje podmnožina primárního klíče, která by funkčně určovala neklíčový atribut). Splnění 3.NF zaručuje odstranění tranzitivních FZ (nepřipouští FZ mezi neklíčovými atributy). Splnění BCNF zaručuje minimální redundanci a dá se formulovat jako požadavek na existenci FZ jediného typu v relaci, kdy nadklíčkové atributy určují neklíčové. Funkční multizávislost je vlastnost relace, kdy dvě podmnožiny atributů obsahují množiny hodnot, které se vyskytují v relaci ve všech možných kombinacích. 4.NF řeší redundanci pro relace s multizávislostmi. Definice je podobná, jako BCNF, jen jsou přípustné multifunkční závislosti, kde na levé straně pravidel figurují nadklíčkové atributy.

Pojmy k zapamatování

- normalizace relací, bezeztrátová dekompozice schématu
- funkční a multifunkční závislost
- uzávěr množiny funkčních závislostí, minimální pokrytí
- 1.NF, 2.NF, 3.NF, Boyce-Coddova normální forma, 4.NF

Kontrolní otázky

45. Jaké problémy mohou nastat při manipulaci s daty v nesprávně navržené relaci, jaké jsou důvody k normalizaci?
46. Co je funkční závislost?
47. Jaké jsou a k čemu slouží Armstrongovy axiomy?
48. Jaké algoritmy se používají při práci s funkčními závislostmi?
49. Co je to multifunkční závislost?
50. Jaké znáte normální formy relací a jak jsou definovány?

Cvičení

1. Je dána množina FZ {AC → B, CB → D}, množina atributů je {A, B, C, D}. Patří do uzávěru FZ závislost AC → D?

Úkoly k textu

Pokuste se ve dříve uvedených i svých příkladech ER diagramů, nebo v nově formulovaných zadáních, určit z kontextu aplikace funkční závislosti, odvodit uzávěr a minimální pokrytí množiny FZ. Transformujte jednodušší ER diagramy do relačního schématu a určete, v jakých normálních formách se relace nacházejí a proveďte dekompozici do co nejvyšší normální formy.

Řešení

1. Podle 1. pravidla Armstrongových axiomů platí: AC → C.
2. Podle 2. pravidla Armstrongových axiomů platí: AC → BC.
3. Podle 3. pravidla Armstrongových axiomů platí: AC → D.

8 Transakční zpracování, paralelismus a zotavení po poruše

Studijní cíle: Student po prostudování kapitoly by měl popsat vlastnosti a stavy transakce a souvislosti se zajištěním konzistence databázového systému po poruchách, využití žurnálu. Měl by rozumět paralelnímu zpracování transakcí s možnými variantami rozvrhů a anomálií, způsoby zamykání, vysvětlit problém uváznutí, jeho detekce a odstranění.

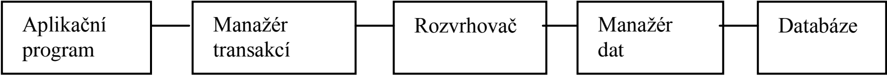
Klíčová slova: Transakce, konzistence, stavový diagram transakce, model transakce, žurnál, algoritmus UNDO – REDO, anomálie,izolační úroveň, optimistický a pesimistický systém, uspořadatelný a sériový rozvrh transakce, uzamykací protokol, uváznutí

Potřebný čas: 4 hodiny

8.1 Koncept transakce

Transakční zpracování dat je reakcí na nejdůležitější úkol každého SŘBD zajistit ochranu a správnost uchovávaných dat a zároveň poskytnout maximální výkon – umožnit korektní a rychlý asynchronní přístup více uživatelů. Proto v každém SŘBD jsou moduly řízení souběžného zpracování a zotavení po poruše, které každému uživateli poskytují data v konzistentním stavu databáze(vyhovují všem integritním omezením ve schématu databáze), i když se stejnými daty paralelně pracuje několik uživatelů nebo došlo k poruše či chybě v softwaru, hardwaru, výpadku napájení atd..

Transakce je posloupnost akcí nebo specifikace operací (jako jsou čtení a zápis dat, výpočty), které se buď provedou všechny, nebo se neprovedou vůbec. Z hlediska aplikace je to logická i programová jednotka práce, odpovídající nějakému procesu v realitě. Př.: Převod peněz z jednoho účtu na jiný při platbě faktury. Její provádění je zabezpečeno a řízeno tak, že jsou splněny následující podmínky. Při čtení potřebných dat musí být databáze konzistentní, během operací transakce je dočasně i v nekonzistentním stavu, ale nakonec při potvrzení transakce musí být databáze opět konzistentní. Po výpadku se databáze uvede do stavu na počátku transakce. Aby systém mohl splnit tyto úkoly, udržuje vlastními prostředky historii změn dat v přiměřeném časovém intervalu v žurnálu(log file). Architektura softwaru, která zajišťuje tuto činnost, se skládá z následujících vrstev :



Jazyk SQL má pro řízení transakcí uživatelem explicitní příkazy, které mohou mít v komerčních systémech různou formu. Microsoft SQL Server používá BEGIN TRANSACTION pro definici začátku transakce – závorkuje posloupnost operací transakce, ROLLBACK [TRANSACTION] pro přerušení transakce a návrat hodnot databáze do stavu na začátku transakce a COMMIT [TRANSACTION] pro její ukončení. Pro zachování integrity a konzistence dat musí transakce splňovat vlastnosti ACIT:

- Atomicita(Atomicity) – operace transakce musí úspěšně proběhnout všechny(je-li transakce potvrzena), nebo se neprovedou vůbec. Údržba atomicity transakce provede příkaz ROLLBACK po chybě dat nebo po uváznutí.
- Konzistence(Consistency) – izolované provádění transakce chrání konzistenci databáze.Transakce převádí databázi z jednoho konzistentního stavu do jiného.
- Izolovanost(Isolation) – dílčí efekty jedné transakce nejsou viditelné jiným, transakce mohou pracovat souběžně, ale každá transakce musí být odstíněna od výsledků operací

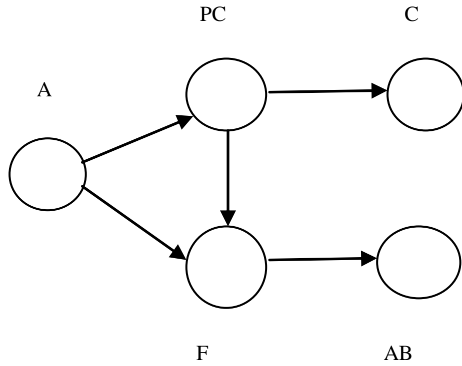
ostatních neukončených nebo následujících transakcí. Databáze prochází takovými stavy, jako by souběžné transakce probíhaly sériově za sebou s vhodným uspořádáním

- Trvalost(Durability) – výsledky potvrzené transakce jsou perzistentně uloženy

Průvodce studiem

Transakce je akce nebo série akcí, vykonávaná jediným uživatelem nebo aplikačním programem, která čte nebo modifikuje obsah databáze a tvoří logickou jednotku práce.

Stavový diagram transakce



Obr. 16 Stavový diagram transakce

se stavy:

A – aktivní, stav od počátku provádění transakce

PC – částečně potvrzený, stav po provedení poslední operace transakce

F – chybný, stav po výskytu problému, který nedovolí pokračování transakce

C- potvrzený, nastane po potvrzení operací COMMIT, znamená úspěšné provedení transakce

AB – zrušený, nastane po skončení operace ROLLBACK, která uvede databázi do stavu před transakcí

8.2 Ochrana proti porušení konzistence dat

Konzistence dat znamená, že každý uživatel má zajištěn konzistentní pohled na data, která jsou vidět včetně změn, které provedly transakce uživatele samotného nebo transakce ostatních uživatelů. Uveďme si nejprve, ve kterých situacích může k nekonzistenci systému dojít. Datové soubory jsou uloženy na disku po blocích, které jsou jednotkou přenosu dat mezi vnější diskovou pamětí a operační pamětí počítače. Operace SŘBD při zpracování transakcí čtou data z disku po blocích do vyrovnávací a operační paměti, někdy je modifikované opět po blocích zapisují zpět na disk. Pokud se v důsledku poruchy nepodaří aktualizovaná data z vyrovnávací paměti uložit na disk, dojde většinou k porušení konzistence. K porušení konzistence může dojít také při souběžném zpracování sdílených dat. Pro podrobnější analýzu si definujeme nové pojmy.

Zotavení databáze je proces uvedení databáze do korektního konzistentního stavu po poruše nebo výpadku systému

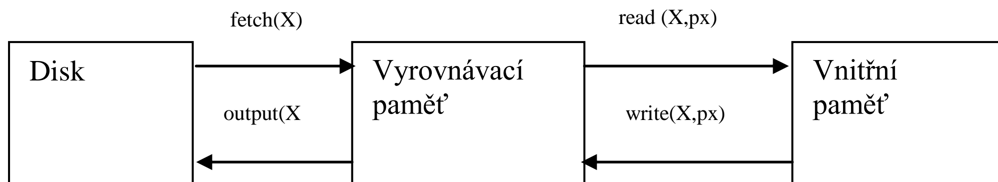
Při modelování transakcí pracujeme s transakčním modelem, který operuje nad strukturou objektů (jednoduchý, složitý objekt, ADT), se strukturou transakcí a jednoduchým modelem chyb (pouze dvě možnosti, úspěch - neúspěch). Struktura transakce může být plochá nebo hnízděná (obsahující dílčí transakce), či dlouhotrvající (workflow). Dále budeme předpokládat jednoduchý model transakce, nezávislý na databázovém modelu a jednoduché objekty, operace *fetch* a *output* pro komunikaci disku s vyrovnávací pamětí.

V modelu definujeme operace:

- *read (X,px)*, která načítá databázovou hodnotu X do lokální proměnné px; není-li blok, ve kterém je uložena hodnota X na disku, právě umístěn v operační paměti, provede se operací fetch(X) přesun bloku z disku do vyrovnávací paměti; hodnota X se z vyrovnávací paměti přenese do px.

- *write (X,px)*, která hodnotu lokální proměnné px zapíše do databázové položky X, která se nalézá ve vyrovnávací paměti; pokud blok, ve kterém je X umístěna, není v operační paměti, provede se nejprve operací fetch(X) jeho přesun z disku do vyrovnávací paměti. Potom se provede přenos hodnoty proměnné px do X.

Žádná z obou operací nevyžaduje zápis na disk. Blok s daty je na disk zapsán jen v některých situacích – například když správce vyrovnávací paměti potřebuje v operační paměti místo pro jiný blok, nebo program končí práci s datovým souborem příkazem close, pak správce vyrovnávací paměti zapíše všechny bloky tohoto souboru na disk, nebo program dá příkaz k zápisu změn provedených ve vyrovnávací paměti na disk - provádí instrukci output(X). Tato operace tedy nemusí vždy bezprostředně následovat za operací write(X,px), protože v bloku, kde se X nalézá, mohou být umístěny další položky, se kterými chce systém pracovat. Když dojde k chybě mezi operací write(X,px) a output,(X), ztratí se data uložená ve vyrovnávací paměti, dosud nezapsaná na disk.



Chyby a poruchy mají globální (vliv na více transakcí) nebo lokální charakter (vliv na jednu transakci). Příklady globálních poruch :

- výpadek systému serveru, ztráta dat z vyrovnávacích pamětí (ztráta napájení)
- systémové chyby, uváznutí komunikace mezi klientem a serverem, (ovlivňují jednotlivé transakce, ale ne celou databázi)
- chyby médií

a lokálních : logické chyby v transakci(nenalezená data), dělení nulou

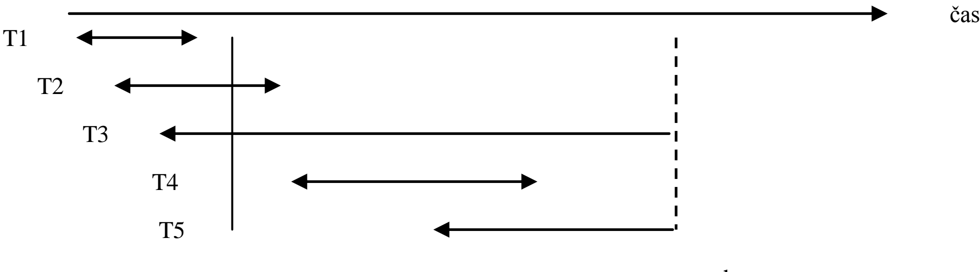
Společnou vlastností všech metod, které se snaží tento problém eliminovat, je to, že pracují s kopiemi dat tak dlouho, dokud není jasné, že transakce proběhla bezchybně celá nebo že je nutné ji zopakovat.

Metoda zpožděné aktualizace zapisuje data do diskového souboru teprve když je jisté, že transakce proběhla celá úspěšně. Výsledky transakce nezapisuje přímo do databázového souboru, ale do žurnálu. Žurnál obsahuje historii všech změn na stabilním paměťovém médiu ve vhodném časovém intervalu. Formát záznamů v žurnálu obsahuje informace <TRANSAKCE T_i,START>, <TRANSAKCE T_i,COMMIT>, <TRANSAKCE T_i,ATRIBUT, Původní_HODNOTA, Nová_HODNOTA > všech probíhajících transakcí, informaci o kontrolním bodě a aktuálních transakcích <Tc, T_1,T_2, …T_i > .

Pokud dojde při transakci k chybě, může se celá provádět znovu, protože původní data nebyla změněna. Přitom se obsah pomocného souboru začne vytvářet znovu, původní je ignorován. Skončí-li transakce úspěšně, příslušný obsah žurnálu se překopíruje do skutečného datového souboru. Pokud by došlo k chybě při kopírování, může se spustit znovu tolikrát, dokud neskončí tato druhá etapa úspěšně. Operace, které je možno spouštět opakovaně, aniž by došlo k porušení konzistence dat, nazýváme operacemi typu REDO.

Metoda přímé aktualizace provádí zápis do výsledného datového souboru přímo, avšak pro případ neúspěšného ukončení transakce zaznamenává souběžně do žurnálu změny vůči původním hodnotám dat Dojde-li v průběhu transakce k chybě, odvodí se pomocí hodnot v žurnálu původní hodnoty datového souboru. Operace, která obnoví původní hodnoty dat v souboru se nazývá operací typu UNDO.

Aby se omezilo prohledávání žurnálu a zefektivnila činnost systému řízení, zadávají se tzv. kontrolní body tc (checkpoint), které způsobí pravidelné ukládání informací z paměti na disk. Pak při obnově databáze není nutné se vracet na začátek žurnálu, ale jen k transakcím posledního kontrolnímu bodu. Možné typy stavu transakce ke kontrolnímu bodu a poruše :



Kontrolní bod (checkpoint) je okamžik , kdy se synchronizuje obsah databáze a žurnálu.

Obr. 17 Stavy transakce ke kontrolnímu bodu a poruše

T1 je ukončena před Tc

T2 začala před tc a skončila před poruchou

T3 začala před tc a neskončila před poruchou

T4 začala po tc a skončila před poruchou

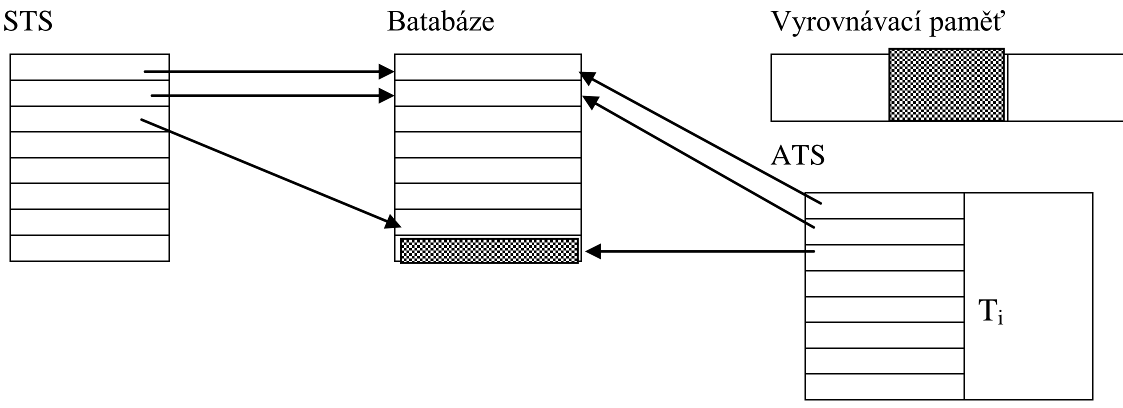
T5 začala po tc a neskončila před poruchou

Po restartu systému po poruše probíhá procedura :

1. Vytvoří se dva seznamy – UNDO a REDO. Do UNDO se vloží rozpracované transakce z posledního kontrolního bodu před poruchou.
2. Od tohoto tc se prochází dopředu žurnál a každá nová transakce je přidána do UNDO, pokud je nalezen COMMIT nějaké transakce, je tato převedena z UNDO do seznamu REDO.
3. Na konci žurnálu obsahuje UNDO typy T3, T5 a REDO typy T2, T4.
4. Žurnál se prochází zpětně a transakce z UNDO realizují ROLLBACK
5. Žurnál se opět prochází dopředu a transakce z REDO se zpracují.

Metoda stínového stránkování používá k popisu uložení dat v databázi na disku dvě pomocné tabulky stránek – aktuální(ATS) a stínovou(STS), kde jsou udržovány diskové adresy bloků databáze. Před zahájením transakce mají obě pomocné tabulky stejný obsah - aktuální adresy obsazených stránek. Stínová tabulka se během transakce nemění. Pokud se během transakce mění obsah některé stránky, nechá se na disku původní stránka beze změny, pokud není stránka ve vyrovnávací paměti, provede se INPUT a stránka s novým obsahem se zapíše na prázdné

místo na disku. V aktuální tabulce stránek se změní odkaz na novou stránku, ve stínové tabulce odkazů zůstává odkaz na starou stránku. Skončí-li transakce bezchybně, platí aktuální tabulka a místo se starým obsahem stránky na disku se uvolní pro další použití. Skončí-li transakce poruchou, platí stínová stránka, která obsahuje odkazy na stránky před zahájením transakce. Uvolní se naopak stránky se změněnými hodnotami, transakce se může zopakovat. Problémem při tomto způsobu práce je evidence uvolněných stránek (garbage collection) a uvolňování pro další použití. To zvyšuje složitost a režii celého systému.



Obr. 18 Metoda stínového stránkování

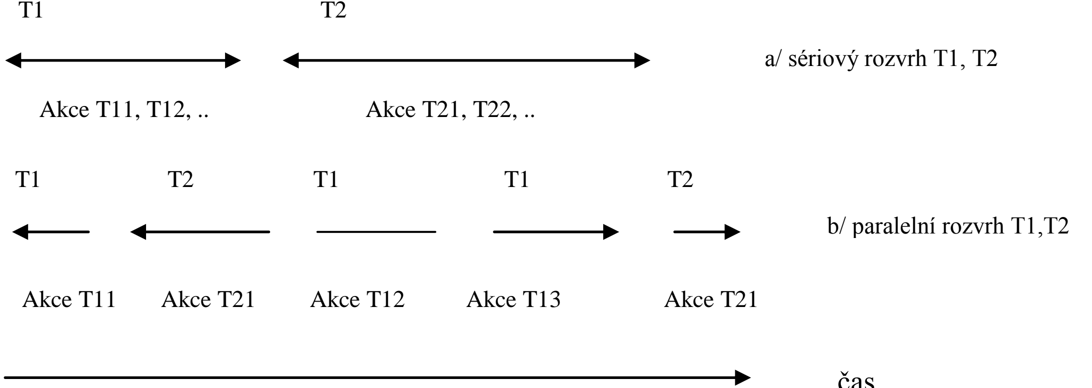
Dosud popisované metody chrání data před ztrátou informace v paměti počítače. Ke ztrátě informace však může dojít také na disku. Základní metodou ochrany diskových dat je důsledné a pravidelné kopírování obsahu disku (na magnetické pásky, na záložní disk ap.). Poruchy operace "zapiš blok" můžeme rozdělit na dva případy - chyba je zjištěna při přenosu, blok na disku zůstane nezměněn; přenos se může opakovat nebo je zjištěna později a blok na disku obsahuje chybná data. Pro pojištění této situace zapisuje systém blok do dvou fyzických bloků na disku a po úspěšném přenosu porovnává jejich obsah. V případě shody předpokládá bezchybný přenos.

Je zřejmé, že všechny tyto postupy zvětšují paměťové i časové nároky databázového systému.

8.3 Řízení paralelního zpracování transakcí

Rozvrh je posloupnost databázových operací, kterou souběžně provádí několik transakcí s tím, že je zachováno pořadí operací v každé individuální transakci.

Existuje však mnoho případů, kdy je nutné, aby databáze byla současně přístupná více uživatelům a aby (často se stejnými daty) s ní pracovalo současně (paralelně) více aplikací. Typickými příklady jsou velké systémy pro rezervaci místenek, jízdenek či letenek. V těchto případech nastává problém jak zajistit, aby při paralelním zpracování dat v databázi nedocházelo vlivem špatné koordinace operací k chybám, k porušení konzistence a integrity. Pokud se data jen čtou, je výhodné zajistit co největší míru paralelismu. Hodnoty dat se nemění a nemůže tedy vzniknout nekonzistence. U operací, které databázi modifikují (vkládají, ruší nebo aktualizují data) je však nutné zajistit, aby souběžné transakce nad sdílenými daty zaručovaly uspořádatelnost rozvrhu. Pro vysvětlení problému si zavedeme další pojmy. *Rozvrh* je stanovené pořadí prováděných klíčových akcí, operací (zápis, čtení) jedné nebo více transakcí v čase.



Sériový rozvrh je takový, ve kterém jsou operace jednotlivých transakcí prováděny za sebou, bez prokládání.

Uspořádatelný rozvrh je takový, ke kterému existuje sériový rozvrh ze stejných transakcí takový, že oba rozvrhy mají v databázi stejný efekt.

Speciální kategorií jsou *sériové rozvrhy*, kdy transakce probíhají v čase v libovolném pořadí za sebou. Úkol vytvářet efektivní rozvrhy souběžného zpracování transakcí má rozvrhovač. Jde tedy o zajištění maximálního paralelismu v manažeru transakcí s garancí uspořádatelnosti. Jde o přirozený požadavek (a kritérium korektnosti), aby výsledek po paralelním provedení řady transakcí byl stejný, jako když by byly provedeny celé transakce postupně za sebou v libovolném sériovém rozvrhu. V tomto případě mluvíme o uspořádatelném rozvrhu.

8.3.1 Problémy paralelního přístupu

Připusťme nyní paralelní zpracování transakcí, tj. takové, že se operace obou transakcí mohou střídat. Takových schémat paralelního zpracování je zřejmě mnoho. Ukážeme si, že některá z nich vedou k porušení konzistence. Úkolem je zřejmě najít schémata, která splňují požadavek uspořádatelnosti.

Anomálie ukážeme na příkladu rezervačního systému. Mějme dvě transakce T1 a T2, příslušející dvěma paralelně běžícím terminálům. K prezentaci rozvrhu použijeme tabulku.

Ztráta aktualizace (při nevhodném „prokládání“ transakcí)

čas ↓

Transakce T1	Transakce T2	stav
read(X)		X = 200
X:=X-50		Zrušení 50 rezervací
	read(X)	X = 200
write(X)		X = 150
	X:=X+10	Přidání 10 rezervací
	write(X)	X = 210

Správnou hodnotou X by mělo být 160, ale je chybných 210, transakce T1 se neuplatnila.

Dočasná aktualizace (přepis nepotvrzené hodnoty při chybě systému)

čas ↓

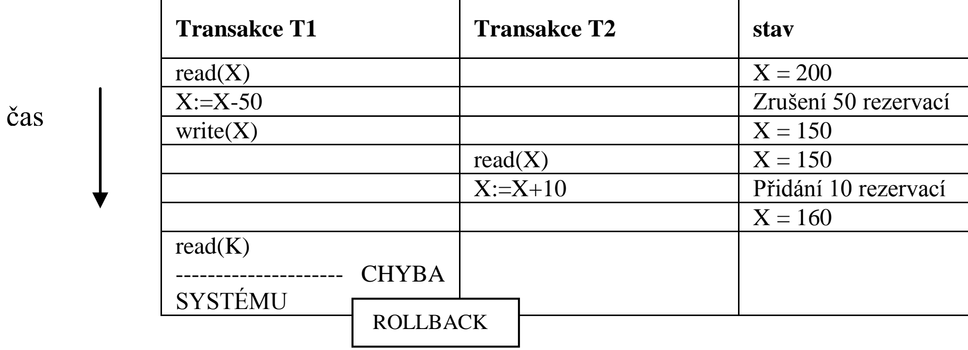
Transakce T1	Transakce T2	stav
read(X)		X = 200
X:=X-50		Zrušení 50 rezervací
write(X)		X = 150
	read(X)	X = 150
	X:=X+10	Přidání 10 rezervací

	write(X)	X = 160
read(K) --------------------- CHYBA SYSTÉMU		
		X = 200

ROLLBACK

Po provedení ROLLBACKU transakce T1 bude aktualizace provedená v transakci T2 přepsána a ztracena. Tento typ chyby bývá někdy tolerován pro dlouhotrvající transakce.

Závislost na potvrzení aktualizace (zpracování nepotvrzené hodnoty při chybě systému)



čas

Transakce T1	Transakce T2	stav
read(X)		X = 200
X:=X-50		Zrušení 50 rezervací
write(X)		X = 150
	read(X)	X = 150
	X:=X+10	Přidání 10 rezervací
		X = 160
read(K) --------------------- CHYBA SYSTÉMU		

ROLLBACK

Po provedení ROLLBACKU transakce T1 bude načtená data X v transakci T2 znehodnocena

V SQL se kromě ztráty aktualizace(Lost Updates) uvažují tři typy porušení uspořádatelnosti:

Dirty read

T1	T2
read(X) … write(X) … rollback	… read(X)

Transakce T2 načte a pracuje s nepotvrzenými daty T1

Nonrepeatable read

T1	T2
read(X) … read(X)	… write(X)

Transakce T1, používající kurzor, se snaží znovu přečíst řádek, který již jednou četla. Ten však již obsahuje jiné hodnoty, aktualizované T2

Phantoms

T1	T2
read(X) … read(X)	… INSERT(X)

Transakce T1 přečte množinu řádků (SELECT) a zpracovává je. T2 změní řádky operacemi INSERT nebo DELETE. Použije-li T1 znovu SELECT, dostane jinou množinu řádků.

V SQL 92 se definují čtyři izolační úrovně, které určují různé chování systému zpracování transakcí, připouští různé kombinace anomálií. Teprve poslední úroveň *Serializable* zaručuje uspořádatelnost transakcí, pro ostatní úrovně by měl SŘBD poskytnout příkazy pro řízení souběžného přístupu. Microsoft SQL Server a podobně další nabízí příkaz SET TRANSACTION ISOLATION LEVEL pro nastavení izolační úrovně.

	anomálie		
Izolační úroveň	Dirty read	Nonrepeatable read	Phantom
Read uncommitted	ano	ano	ano
Read committed	ne	ano	ano

Repeatable read	ne	ne	ano
Serializable	ne	ne	ne

Pro n současně běžících transakcí existuje n! možností sériového pořadí. V reálných podmínkách se zřídka setkáme s perfektní izolovaností transakcí při souběžném zpracování, protože je častěji preferován kompromis omezující částečně izolovanost a podporující výkon systému.

V paralelním rozvrhu jsou operace různých transakcí v rozvrhu prokládány. Pro ilustraci příklady souběžných rozvrhů

Uspořádatelný paralelní rozvrh

T1	T2
read(X) … write(X)	
	read(X) … write(X)
read(Y) … write(Y)	
	read(Y) … write(Y)

Neuspořádatelný paralelní rozvrh

T1	T2
read(X) … write(X) read(Y) … write(Y)	read(X) … write(X) read(Y) … write(Y

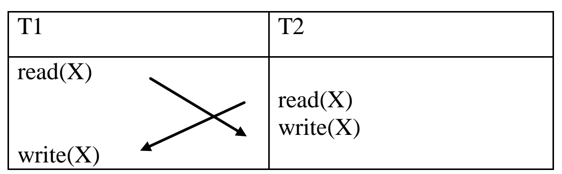
Je možno vysledovat, že důležité jsou operace read () a write () a jejich pořadí, ostatní operace na výsledek paralelního zpracování z hlediska konzistence nemají vliv. Ekvivalence rozvrhů je založena na komutativitě těchto operací. Dvě operace jsou kompatibilní, jestliže výsledky jejich různého sériového provádění jsou ekvivalentní. V tomto případě jsou kompatibilní operace READ-READ a naopak konfliktní ostatní kombinace operací WRITE-READ, READ-WRITE a WRITE-WRITE. Dají se formulovat podmínky, za kterých jsou dva rozvrhy S1 a S2 ekvivalentní, které vycházející z nutnosti zachovat relativní pořadí konfliktních operací v S1 i S2. Pro systém, kde každá transakce nejprve přečte objekt operací READ a teprve potom do něj zapíše operací WRITE, je možno sestrojit precendenční graf, pomocí kterého můžeme testovat uspořádatelnost. Je to orientovaný graf {U,H}, jehož uzly jsou transakce – U={ Ti | i = 1, … , n} a jehož hrany H = {h_{ik}(A)} ,vzhledem k manipulaci s objektem A, jsou orientovány Ti → Tj, jestliže

Efektivní metoda zjišťování uspořádatelnosti rozvrhu využívá precedenčních grafů

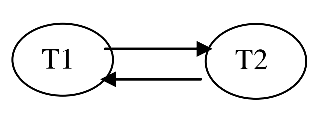
(1) Ti provede WRITE(A) dříve, než Tj provede READ(A)

(2) Ti provede READ(A) dříve, než Tj provede WRITE(A).

Př 1 :



T1	T2
read(X) write(X)	read(X) write(X)

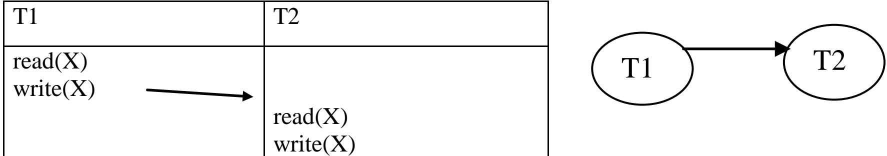


Platí, že

T1 provede READ(A) dříve než T2 provede WRITE, dle (2) platí T1→ T2

T2 provede READ dříve než T1 provede WRITE, dle (2) platí T2→ T1

Př. 2 :



T1	T2
read(X) write(X)	read(X) write(X)

Platí, že

T1 provede WRITE dříve než T2 provede READ, dle (1) platí T1 → T2

T2 neprovede dříve než T1 nic

Jestliže získaný orientovaný graf obsahuje cyklus, pak testované schéma paralelního zpracování transakcí nesplňuje požadavek uspořádatelnosti. Ekvivalentní rozvrhy mají stejný precedenční graf.

Pro systém, kde transakce zapisuje pomocí WRITE(A), aniž by předtím četla operací READ(A) lze ukázat, že neexistuje žádný efektivní algoritmus, který by rozhodl, zda dané schéma paralelního zpracování transakcí splňuje požadavek uspořádatelnosti. Testování libovolného rozvrhu vede k neakceptovatelným časovým nárokům. Proto se používají tzv. protokoly – transakce se vytváří podle jistých pravidel, které zaručí, že za jistých předpokladů jsou rozvrhy takových transakcí uspořádatelné.

Databázové systémy v konkrétních podmínkách mohou být provozovány podle různých strategií transakčního zpracování. Optimistická strategie předpokládá relativně malé paralelní zpracování s minimem nekonzistentních rozvrhů. Transakce se provádí s operacemi jednoduchého modelu a testuje se konzistence systému. Pokud dojde k nekonzistentnímu stavu, kritické transakce se operací ROLLBACK vrátí do výchozího stavu a provedou znovu. U pesimistických systémů se silným souběžným zpracováním transakcí by bylo nekonzistencí tolik, že by efektivita zpracování klesla pod únosnou mez. V pesimistických systémech se do transakčního modelu přidávají operace LOCK a UNLOCK pro zamykání a odmykání vhodných databázových objektů, pomocí kterých se požadavek uspořádatelnosti dá snáze vynutit.

Průvodce studiem

SŘBD mohou zajistit, že souběžné zpracování transakcí T1, …, Tn je ekvivalentní některému sériovému zpracování týchž transakcí, což je záruka konzistentního stavu databáze. Musí ale splňovat požadavek uspořadatelnosti – tj., že efekt paralelního zpracování transakcí je stejný jako podle nějakého sériového rozvrhu.

8.3.2 Uzamykací protokoly

Způsob řízení zámků určují uzamykací protokoly, které mohou být statické nebo dynamické a omezují množinu možných rozvrhů.

Uzamykací protokoly jsou založeny na statickém nebo dynamickém zamykání a odemykání objektů v databázi. Úkolem zámku je zablokovat přístup ostatních transakcí ke sdíleným datům tak, aby tato data byla vždy k dispozici jen jediné transakci. Když jedna transakce získá k údaji výlučný (exklusivní) přístup, pak tento údaj nemůže modifikovat jiná transakce dříve, než první transakce skončí a uvolní přístup k údaji - a to i v případě, že byla při paralelním zpracování

několikrát přerušena. Říkáme, že údaje jsou uzamčeny. *Statické protokoly* uzamknou všechny objekty, se kterými transakce pracuje hned na začátku a uvolní je všechny těsně před koncem transakce. Proto je možné paralelní zpracování jen takových transakcí, které nesdílí žádné objekty. Uzamykací protokoly požadují práci v podmínkách, které definuje pojem legální rozvrh

- objekt může být uzamčen nejvýše jednou transakcí

- objekt je v transakci uzamknutý, kdykoliv k němu transakce přistupuje, transakce má právo provádět s objektem operace

- transakce, která se pokouší zamknout objekt uzamčený jinou transakcí čeká na uvolnění zámku.

Dále budeme předpokládat přirozené požadavky na chování transakce (dobře formované transakce)

- Transakce vždy zamyká objekt, když s ním bude pracovat, ale ne opakovaně, když jej již zamkla
- Transakce neodemyká objekt, který nezamkla, ale na konci jsou odemčeny všechny objekty, které transakce uzamkla

Existuje několik úrovní zamykání údajů v závislosti na tom, co se zamyká. V principu se jedná o viditelné uživatelské objekty (tabulky, řádky) a o neviditelné systémové objekty (mezivýsledky operací, některé informace z katalogu dat). Dalším pohledem je jemnější rozlišení objektů - granularita (logická a fyzická):

Granularita je velikost částí dat, určených jako jednotka zpracování a ochrany v protokolu paralelního řízení transakcí.

1. Soubor - na úrovni operačního systému definujeme soubor typu read-only a tak zakážeme zápis a modifikaci všem.

   Na úrovni SŘBD v aplikačním programu definujeme svůj pracovní soubor jako soubor s výlučným přístupem (exclusive). Tak zamezíme přístup všem ostatním procesům, dokud náš program neskončí a neuvolní soubor. Použijeme příkaz k uzamčení a uvolnění souboru, říkáme, že soubor zamykáme explicitně. V SŘBD existují příkazy pro práci se souborem, které vyžadují výlučný přístup k souboru a tak si uzamykají soubor automaticky.
2. Tabulka, blok (stránka), záznam - v aplikačním programu stačí často zamknout na logické úrovni tabulku, na fyzické blok nebo jen jeden či několik záznamů (ne celý soubor), aby tak byly ostatní záznamy přístupné ostatním uživatelům. Zamykání záznamů může být explicitní nebo automatické.
3. Položka - některé SŘBD umožňují zamykat dokonce jen jednotlivé položky záznamu.

Dále rozlišujeme zámky dvou základních druhů v souvislosti s operacemi, které budeme s databázovým objektem provádět:

1. zámky pro sdílený přístup (shared), umožňují údaje jen číst více transakcím současně, ne však do nich zapisovat,
2. zámky výlučné (exclusive), umožní čtení i zápis vždy pouze jediné transakci.

Pokud má jedna transakce údaj (soubor, záznam) uzamčený a další transakce jej chce uzamknout také, může dojít u většiny kombinací ke kolizi. Proto v SŘBD existují funkce testující, zda je údaj volný. Pokud není, systém testuje typ zámku a situaci programově řeší (použití zámku, čekání na uvolnění, zrušení transakce ap.). K tomuto účelu se používá tabulka kompatibility zámků

Kompatibilita zámků	Požadované další zámky v módu	
	shared	exclusive

Objekt drží zámky v módu	shared	ano	ne
	exclusive	ne	ne

Zaveďme si následující označení pro žádosti transakcí o uzamčení :

LS(A) ... zamkni položku A pro sdílený přístup (Lock Shared)

LX(A) ... zamkni položku A pro výlučný přístup (Lock eXclus)

UN(A) ... uvolni položku A (UNlock)

Z tabulky plyne, že je možné nasadit mnoho sdílených zámků na objekt, ale výlučný zámek drží jen jediná transakce. Tedy žádosti LS(A) lze vyhovět vždy, není-li na A zámek typu LX(A). Žádosti LX(A) lze vyhovět pouze tehdy, je-li položka A ve stavu po provedení UN(A) nebo bez zámku.

Ale i s používáním zámků mohou nastat problémy. Samotné zamykání a odmykání nezaručuje uspořádatelnost rozvrhů.

Příklad : mějme dvě transakce T1 a T2 s počátečními hodnotami X = 100, Y = 50.

příklady souběžných rozvrhů

Uspořadatelný paralelní rozvrh S1

T1	T2	
LX(X)		
read(X)		X1 = 100
X:=X+10		X1 = 110
write(X)		
UN(X)	LS(X)	
LX(Y)	read(X)	X2 = 110
read(Y)	UN(X)	Y1 = 50
Y:=X+Y-20		
write(Y)		Y1 = 140
UN(Y)		
	LX(Y)	Y2 = 140
	read(Y)	
	Y:= Y-X	
	write(Y)	Y2 = 30
	UN(Y)	

Neuspořadatelný paralelní rozvrh S2

T1	T2	
LX(X)	LS(X)	
read(X)	read(X)	X2 = 100
X:=X+10	UN(X)	X1 = 110
write(X)	LX(Y)	
UN(X)	read(Y)	Y2 = 50
	Y:= Y- X	Y2 = -50
	write(Y)	
	UN(Y)	
LX(Y)		
read(Y)		Y1 = -50
Y:= X +Y-20		Y1 = 40
write(Y)		
UN(Y)		

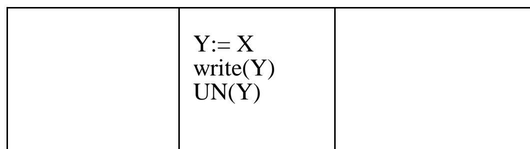
S1 a S2 nejsou ekvivalentní.

8.3.3 Uváznutí

Analyzujme rozvrh S3. Jedná se o typický případ, kdy dvě transakce pracují se dvěma objekty a operace jsou v čase seřazeny tak, že každá z transakcí uzamkne jeden objekt a zámek neuvolní a následně se křížem pokusí uzamknout druhý objekt, ale ten není k dispozici.

Uváznutí je bezvýchodná situace, která nastane když například dvě transakce čekají vzájemně na uvolnění zámků, které drží křížem právě druhá zúčastněná transakce

T1	T2	
LX(X)		
read(X)		
	LX(Y)	
	read(Y)	
LX(Y)		T1 čeká na T2
read(Y)	**LX(X)**	T2 čeká na T1,
Y:= X +Y-20	read(X)	Nastalo uváznutí
write(Y)	UN(X)	
UN(Y)		
UN(X)		



	Y:= X write(Y) UN(Y)	

Takovéto situaci, kdy transakce čekají a nelze žádný požadavek uspokojit a celý proces zpracování se zastaví říkáme *uváznutím (deadlock).* Problém tedy je v tom, že pokud používáme zámků málo, hrozí nekonzistence, používáme-li mnoho zámků nevhodným způsobem, hrozí uváznutí.. Jednoduchou metodou pro sestavení takového protokolu, který vynucuje uspořádatelnost, je *dvoufázový protokol*. Spočívá v tom, že v první fázi objekty transakce co nejdříve jen zamykáme a zámky neuvolňujeme, ve druhé fázi, která začíná prvním odemknutím, naopak zámky jen uvolňujeme a nezamykáme. Jestliže všechny transakce v rozvrhu jsou dobře formované a dvoufázové, potom jejich legální rozvrh je uspořádatelný, není však vyloučena možnost uváznutí. Řešením uváznutí je použití metod detekce nebo prevence uváznutí. Pro prevenci uváznutí existuje více technik.

Transakce vyhovuje dvoufázovému protokolu, když operace zamykání všech objektů transakce předchází prvnímu uvolnění zámku.

Nejjednodušší metodou prevence uváznutí je uzamčení všech položek, které transakce používá, hned na začátku transakce ještě před operacemi a jejich uvolnění až na konci transakce (statické zamykání). Tak se transakce nezahájí dříve, dokud nemá k dispozici všechny potřebné údaje a nemůže dojít k uváznutí uprostřed transakce. Tato metoda však má dvě velké nevýhody :

1. využití přístupu k položkám je nízké, protože jsou dlouhou dobu zbytečně zamčené,
2. transakce musí čekat až budou volné současně všechny údaje, které chce na začátku zamknout, a to může trvat velmi dlouho.

V SŘBD řeší problém uváznutí synchronizací paralelních transakcí plánovače, s programovými moduly :

- Modul řízení transakcí - je to fronta, na kterou se transakce obracejí se žádostí o vykonání operací READ(X) a WRITE(X). Každá transakce je doplněna příkazy BEGIN TRANSACTION a END TRANSACTION.
- Modul řízení dat realizuje čtení a zápis objektů dle požadavků plánovače a dává plánovači zprávu o výsledku a ukončení.

Plánovač zabezpečuje synchronizaci požadavků z fronty podle realizované strategie a řadí požadavky do schémat. Schéma pro množinu transakcí je pořadí, ve kterém se operace těchto transakcí realizují. Plánovač při dvoufázovém zamykání vykonává tyto operace
- řídí zamykání objektů
- operace čtení a modifikace objektů povoluje jen těm transakcím, které mají příslušné objekty zamknuté
- sleduje, jestli transakce dodržují protokol dvoufázového zamykání. Pokud zjistí jeho porušení, transakci zruší.
- předchází uváznutí nebo ho detekují a řeší zrušením transakce.

Jiný způsob řízení paralelních transakcí je pomocí časových razítek . Časové razítko je číslo přidělené transakci nebo objektu databáze. Čísla tvoří rostoucí posloupnost (např. jsou odvozena od systémového času), jsou jednoznačná pro všechny transakce a platí pro všechny operace transakce. Čísla používá plánovač pro řízení konfliktních operací READ(A) a WRITE(A). Všechny páry konfliktních operací se provádějí v pořadí jejich časových razítek. Princip základního plánovače založeného na časových razítkách :

Časové razítko je unikátní identifikátor, generovaný systémem, který reprezentuje relativní čas

Tento typ plánovače eviduje pro každý objekt A databáze dvě čísla:

1. největší časové razítko, které měla operace READ(A), již provedená nad objektem A, označme jej R/ČR(A)
2. největší časové razítko, které měla operace WRITE(A) provedená nad A, označme jej W/ČR(A).

Když plánovač obdrží požadavek s nějakým časovým razítkem ČR na čtení hodnoty objektu A, je-li ČR < W/ČR(A), odmítne požadavek a zruší transakci, která požadavek zaslala. Jinak vyhoví požadavku a aktualizuje hodnotu R/ČR(A) = max(ČR, R/ČR(A))

Když plánovač obdrží požadavek s nějakým časovým razítkem ČR na zápis hodnoty objektu A, je-li splněna podmínka (ČR < W/ČR(A) OR ČR < R/ČR(A)), odmítne požadavek a zruší transakci, která požadavek zaslala. Jinak vyhoví požadavku a aktualizuje hodnotu W/ČR(A) = ČR. Zrušené transakce se znovu spustí s novou (vyšší) hodnotou ČR. Tento základní plánovač může způsobovat časté rušení transakcí. Existují jeho modifikace nebo jiné strategie plánovačů, které snižují počet zrušení transakcí.

Příklad: Transakce T1 a T2 provádějí čtení a zápis údajů v tomto pořadí

T1: read(A); read(B);write(A);write(B);

T2: read(B);write(B);

Přidělování časových razítek

R/ČR(A) W/ČR(A) R/ČR(A) R/ČR(A)

0 0 0 0

T1: read(A) pro T1 je ČR=1 R/ČR(A)=0 → 1 1

T2: read(A) pro T2 je ČR=2 R/ČR(B)=0 → 2

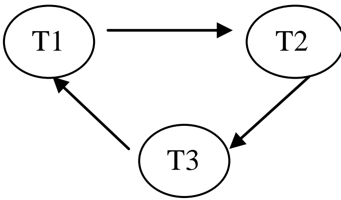
T2: write(B) W/ČR(B)=0 → 2

T1: read(A) W/ČR(A)=0 X

8.3.4 Řešení problému uváznutí

Jestliže systém nepoužívá prevenci uváznutí, musí mít prostředky pro *detekci (testování) uváznutí* a obnovu činnosti umrtvených transakcí. Detekce se provádí obvykle použitím grafu čekání (wait for graf) Je to graf, jehož uzly jsou aktivní transakce Ti a orientované hrany dynamicky zachycují situaci, kdy libovolná transakce Ti čeká, aby mohla uzamknout objekt X, který je ale uzamčen jinou transakcí Tj – vznikne hrana (Ti→Tj) . Analýzou grafu čekání se rozpoznává uváznutí, je-li v grafu detekován cyklus. V okamžiku uváznutí se podle zvolené strategie priorit přeruší a restartuje jedna transakce z cyklu, čímž se zablokovaný přístup k datům (pro tuto transakci) odblokuje a umožní provést ostatní transakce .

Příklad čekacího grafu:



T1 čeká na T2, T2 čeká na T3, T3 čeká na T1

Obnovení činnosti se provádí pomocí žurnálu. V případě potřeby je možno kteroukoliv transakci vrátit. Jde jen o to, kdy a které transakce se mají provést znovu. Systém vybírá takové transakce, aby s celým postupem byly spojeny co nejmenší náklady, k tomu bere v úvahu :

- jaká část transakce již byla provedena,

- kolik dat transakce použila a kolik jich ještě potřebuje pro dokončení,
- kolik transakcí bude třeba celkem vrátit.

Podle těchto kriterií by se mohlo dále stát, že bude vracena stále tatáž transakce a její dokončení by bylo stále odkládáno. Je vhodné, aby systém měl evidenci o vracených transakcích a při výběru bral v úvahu i tuto skutečnost.

Shrnutí Transakční manažer řídí dvě základní úlohy – uložení informací o transakci do žurnálu (začátek transakce, změny datových položek, konec nebo odmítnutí transakce) pro minimalizaci problémů při korektním zotavení po poruše a dále se podílí na organizaci souběžného zpracování transakcí. Data, zapsaná do hlavní paměti nebo na disk nepotvrzenou transakcí nazýváme *Dirty*. Taková data, pomocí informací v žurnálu (log file), můžeme operací ROLLBACK vrátit do výchozího stavu. Obecněji, po poruše, je žurnál využit pro opravu databáze tak, aby zůstala v konzistentním stavu. Toho lze dosáhnout metodou UNDO - REDO. Databáze je v konzistentním stavu, když data v ní uložená vyhovují integritním omezením a reprezentují stav reality. Operace s daty musí být prováděné tak, že databáze přechází z jednoho konzistentního stavu do druhého. SŘBD umožňují najednou několika transakcím přístup ke sdíleným datům, ale musí zajistit izolované provádění operací, zajišťující konzistenci. Za tuto činnost zodpovídá rozvrhovač. Transakci modelujeme pomocí posloupnosti zjednodušených základních akcí, jako je čtení nebo zápis z /do databáze a takto pojatý model nazýváme rozvrhem. Transakce prováděné v čase jedna za druhou tvoří sériový rozvrh. Pokud souběžně běžící transakce dosáhnou v databázi stejného výsledku jako tytéž transakce provedené nějakým sériovým rozvrhem, hovoříme o uspořádatelném rozvrhu. V uspořádatelném rozvrhu nemohou nastat poruchy konzistence, jejichž typy definují anomálie, jako je ztráta aktualizace, dočasná aktualizace, neopakovatelné čtení nebo fantom. Detekci uspořádatelnosti můžeme provádět pomocí precedenčních grafů, které pomohou odhalit případné konflikty. Uzly grafu korespondují s transakcemi a hrana T1 → T2 reprezentuje situaci, kdy akce v T1 je v konfliktu s pozdější akcí v T2. Rozvrh je potom uspořádatelný, když odpovídající precedenční graf je acyklický. Osvědčenou technikou pro podporu uspořádatelnosti je zamykání sdílených databázových objektů před prováděním operací a odemykání těchto objektů po provedení operací. Zámek znemožní přístup k objektu ostatním transakcím. Použití zámků neodstraní všechny problémy a nezajistí uspořádatelnost. Uváznutí nastává například v momentě, kdy jedna transakce čeká na objekt zamčený druhou transakcí a zároveň druhá transakce čeká na objekt, zamčený první transakcí. Minimalizaci problému uváznutí zajistíme použitím dvoufázového uzamykacího protokolu. Transakce v první fázi zamkne všechny požadované objekty a v druhé fázi provádí operace a odemykání zámků. Zámky se rozlišují na sdílení (čtení) a exkluzivní (pro zápis), DBS se řídí při jejich používání tabulkou kompatibility. Dále je možno u zámků rozlišovat, jaký objekt, nebo jeho logická, či fyzická část se zamyká – např. tabulka, n-tice, položka, blok (stránka), soubor, atd.. Uváznutí se dá detekovat jako cyklus v grafu čekání, který v uzlech zachycuje aktuálně běžící transakce a hrany T1→ T2 představují situaci, kdy transakce T1 čeká na objekt uzamčený T2.

Pojmy k zapamatování

- Transakce, vlastnosti ACID
- Konzistence, žurnál, algoritmus UNDO – REDO
- Anomálie, izolační úroveň
- Rozvrh transakce, uspořadatelnost rozvrhu, uzamykací protokol, uváznutí
- optimistický a pesimistický systém

Kontrolní otázky

51. *Co je transakce a jaké má vlastnosti?*
52. *Co je stavový diagram transakce?*
53. *Co je konzistence a jak je zajišťována?*
54. *Co je žurnál a jaké obsahuje informace?*
55. *Co je precedenční graf a k čemu slouží?*
56. *Jak pracuje algoritmus UNDO – REDO?*
57. *Jaké znáte anomálie, úrovně izolace při souběžném provádění transakcí?*
58. *Co je dvoufázový protokol?*
59. *Co je čekací graf?*
60. *Co je optimistický a pesimistický systém?*

Úkoly k textu

Vytvořte rozvrhy jednoduchých paralelních transakcí, které povedou k jednotlivým typům nekonzistence databáze.

Vytvořte rozvrhy jednoduchých paralelních transakcí, které povedou k uváznutí. Nakreslete příslušný čekací graf.

9 Distribuované databázové systémy

Studijní cíle: Po prostudování kapitoly by měl student definovat vlastnosti distribuované databáze a porovnat ji s lokální databází. Měl by popsat možnosti rozložení dat v síti počítačů, problémy spojené se zajištěním transakčního zpracování, udržování schématu databáze.

Klíčová slova: Autonomie, topologie, horizontální, vertikální a kombinovaná fragmentace, replikace dat, distribuovaná optimalizace dotazu, polospojení, dvoufázový potvrzovací protokol

Potřebný čas: 2 hodiny

9.1 Vlastnosti distribuovaných databází

Technické možnosti komunikačních systémů a vzájemného propojování počítačů umožňují spojovat i původně samostatně pracující počítače do distribuovaných systémů. Speciálně u Distribuovaných databázových systémů (DDBS) se s komunikační technologií spojí databáze. Obecná charakteristika systémů:

Distribuovaná databáze je logicky související kolekce sdílených dat a jejich popisů, fyzicky rozmístěná v síti počítačů.

- Tvoří je množina uzlů(míst), propojených komunikačními kanály
- Každý uzel je autonomní SŘBD, ve všech službách se nespoléhá na centrální uzel
- Uživatel nic neví o síťové struktuře systému, ale systém umožní transparentně zpřístupnit data z libovolného uzlu a opačně zpracovává vlastní data pro distribuované dotazy ostatních. Práce jednoho uzlu nenarušuje celý systém, který nepřetržitě pracuje.
- Nezávislost na hardware, operačním systému, síti i SŘBD.

To má řadu výhod, jako

Distribuované SŘBD jsou systémy umožňující řízení distribuovaných databází s tím, že distribuce je pro uživatele transparentní.

1. Vysoká efektivita zpracování a racionalizace provozu – data jsou umístěna v místech, kde jsou nejčastěji využívána, vyšší výkonnost - zátěž zpracování je díky paralelismu rozdělena na více počítačů, větší dostupnost - možnost sdílení společných informačních fondů, DDBS se snadno škáluje – rozšíření konfigurace.
2. Snižuje se cena komunikace vhodným rozmístěním dat v uzlech.(data, která uzel používá má lokálně k dispozici).
3. Větší spolehlivost a tolerance k chybám (zotavení po chybách díky replikám), možnost zálohování počítačů.

Ale i nevýhod

1. Větší složitost – návrhu databáze, katalogu dat a jeho správy, transakčního zpracování s problémy uváznutí, optimalizaci dotazů, územní distribucí dat, šíření aktualizace při replikaci, heterogennost dat na jednotlivých uzlech vede k transformacím, komunikace v počítačových sítích – minimalizace počtu a velikost zpráv, výpadky.
2. Obtížnější dosažení bezpečnosti a utajení.
3. Distribuce řízení.
4. Relativní nedostatek standardů, nástrojů a zkušeností.

Jsou to komplikované a vzájemně propojené problémy a dosavadní distribuované báze jsou prozatím jen unikátními řešeními. Dosud neexistuje obecný model či návod pro jejich tvorbu.

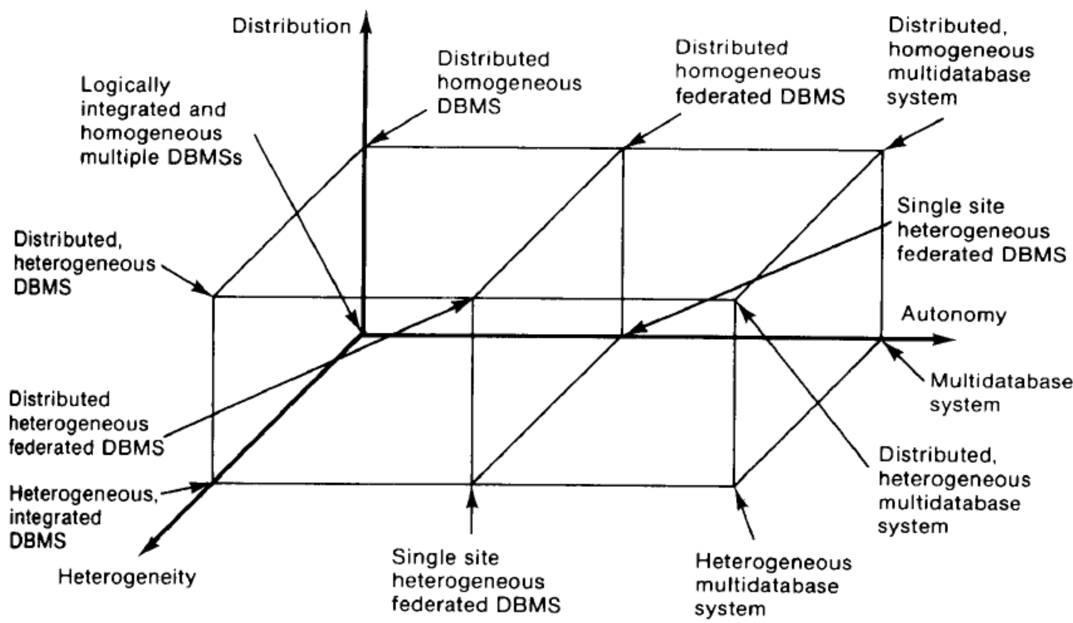
Jednotné řízení distribuované báze může být realizováno různými formami na různém stupni centralizace řízení.

Důležitou vlastností DDBS je to, že přes rozložení dat do jednotlivých uzlů sítě se jeví celá distribuovaná báze uživateli tak, jako by byla lokální. Distribuce je uživateli skryta a z hlediska jeho potřeb není podstatná. Celá distribuovaná báze dat je popsána v *globálním schématu DDBS*. To je popis báze nezávislý na distribuci dat a stejný pro všechny uživatele ve všech uzlech sítě. Nejčastěji je globální schéma popsáno pomocí relačního modelu dat.

9.1.1 Klasifikace DDBS

Hlavní hlediska, která se mohou použít při klasifikaci a návrhu distribuovaných databázových systémů jsou stupně :

- autonomie – distribuce řízení : těsná integrace, poloautonomie, úplná izolace
- distribuce dat : centralizovaná, decentralizovaná
- heterogennosti hardware, OS, SŘBD, databázových modelů : homogenní, nehomogenní
- modely dat, datová komunikace
- těsnost spojení



Obr. 19 Klasifikace DDBS

Lokální báze dat je část celé (distribuované) báze, která je umístěna v jednom uzlu (počítači) počítačové sítě; je řízená lokálním SŘBD, který pracuje ve spolupráci s ostatními složkami distribuovaného databázového systému,

Distribuovaná báze dat je sjednocení všech lokálních bází dat distribuovaného databázového systému,

Distribuovaný databázový systém (DDBS) je tvořen distribuovanou bází dat a programovým vybavením, skládajícím se z lokálních SŘBD a dalších programů potřebných k jejich koordinaci a řízení, k zabezpečení systémových úloh spojených s přístupem uživatelů k DDBS, s udržováním jeho integrity a provozuschopnosti v prostředí počítačové sítě.

Při *těsném spojení* má každé místo znalost o datech v celém DDBS a může řídit a zpracovávat požadavky, jejichž data leží kdekoliv v síti uzlů. *Centralizované DDBS* mají popis a řízení DDBS soustředěno na jeden centrální počítač. Toto centrum DDBS nemusí být v centru příslušné počítačové sítě. V centru DDBS jsou soustředěny popisy všech dat tvořících DDBS a centrálně se tu řídí

- přístup k distribuované bázi dat,
- provádění změn ve struktuře distribuované báze dat,
- provádění a synchronizace transakcí v DDBS,
- všechny další činnosti systému.

Výhodou centralizovaných DDBS je poměrná jednoduchost řízení všech činností systému. Správa má soustavný přehled o aktuálním stavu všech částí systému a má možnost v nevyhnutelném případě přesně a jednoznačně zasahovat.

Nevýhodou těchto DDBS jsou vysoké celkové nároky na komunikaci v systému. Každý přístup k datům, každá změna musí být povoleny a řízeny centrem. Tato skutečnost může značně zpomalit a prodražit provoz DDBS. Jen v lokálních počítačových sítích s vysokými přenosovými rychlostmi se tato nevýhoda nemusí projevit tak výrazně. Dalším nebezpečím je možnost poruchy počítače s centrálním řízením databáze, protože při výpadku hrozí pracná obnova celého DDBS.

Ve *volnější autonomii* je sdílena jen část dat, při *úplné izolaci* se nesdílí žádná data. *Decentralizované DDBS* jsou tvořeny počítačovou sítí, kde žádný uzel nemá privilegované postavení. Všechny počítače mají stejné informace o DDBS a stejnou zodpovědnost za dodržování pravidel vedoucích k zachování integrity systému. Je zřejmé, že algoritmy pro řízení transakcí v takovémto distribuovaném prostředí, bez centrálního řízení, budou složitější. Decentralizované systémy však proti centralizovaným vynikají svou stabilitou. Výpadek žádného počítače nemá za následek větší ztrátu, než ztrátu přístupu k vlastním datům. Tu je navíc možno duplikováním kritických dat v několika uzlech sítě podle potřeby zmírňovat.

Transparence distribuce určuje míru viditelnosti distribuce dat v uzlech jednotlivým aplikacím a uživatelům. Možnosti:

- Distribuce *základních relací* - jednotkou je tabulka, celá umístěna na jednom místě
- Distribuce *odvozených dat* - jednotkou je fyzická tabulka, vzniklá z výsledku dotazu, umístěna na jednom místě bez napojení na základní relace. Podobně u typu *momentka*, ale tady dochází k periodické aktualizaci dat.
- *Fragmentované relace* – Horizontálním(jen podmnožina řádků), vertikálním (jen podmnožina sloupců - atributů) či kombinovaným (obě techniky) logickým rozdělením tabulek vzniknou fragmenty dat, fyzicky uložené – alokované nebo replikované v různých uzlech.
- *Alokované relace* - každý fragment relace je uložen v uzlu s optimálními vlastnostmi distribuce, které určují také informace o počtu
- *Replikované relace* – kopie fragmentu nebo jedné tabulky jsou v různých uzlech.

Průvodce studiem

Transparence distribuce umožňuje uživateli vidět distribuovanou databázi jako logicky jednotnou, lokální databázi. Jednotlivé úrovně:

Transparentnost fragmentace – přístup do databáze je založen na globálním schématu, uživatel nepotřebuje znát jména a rozmístění fragmentů.

Transparentnost umístění – uživatel musí používat pojmenované fragmenty relací, ale nezná jejich umístění. Typicky se používá operátor UNOIN v dotazech.

Mapování umístění – uživatel musí explicitně určit jméno fragmentu i umístění.

Jména atributů musí být unikátní v celé distribuované databázi.

Jednotlivé lokální báze dat mohou být realizovány s použitím jednoho nebo různých datových modelů. Použití jednoho datového modelu samozřejmě snižuje složitost architektury DDBS a tento případ se užívá u mnoha existujících DDBS. Pokud z nějakých důvodů architektura DDBS připouští různé modely dat v jednotlivých lokálních databázových systémech, používá se obvykle jeden společný datový model pro komunikaci a popis hlavních struktur dat v rámci celého distribuovaného systému. Tímto společným datovým modelem je nejčastěji relační model. Pokud lokální SŘBD podporují jiné modely dat, musí být doplněny programovým vybavením, které umí požadované funkce relačního modelu alespoň emulovat. Standardem pro komunikaci lokálních SŘBD bývá většinou jazyk SQL.

V klasických SŘBD se přirozeně požaduje, aby se aktualizační akce provedly okamžitě a aktualizované hodnoty byly ihned nato přístupné všem uživatelům databáze. V distribuovaných systémech se tento přirozený požadavek zabezpečuje podstatně hůř a je příčinou složitosti programového řešení DDBS a tím i vyšších nákladů na jeho provoz. Podle konkrétních aplikačních úloh však není nutné přísně dodržovat tyto požadavky vždy a tak je možno někdy celý systém zjednodušit a zlevnit.

Často používaná data jsou v ní rozmístěna v kopiích v každé lokální bázi. Aby se zjednodušilo řízení a provoz tohoto DDBS, během dne se aktualizační operace jen zaznamenávají a provedou se najednou ve večerních hodinách, kdy počítače nejsou vytížené a volnější jsou i komunikační linky. Samozřejmě se s tímto řešením aktualizace muselo počítat již při celkovém návrhu informačního systému, aby dovoloval počítat s dočasně nepřesnými údaji. Takovéto DDBS nazýváme také volně spojené systémy.

9.2 Rozmístění dat v DDBS

U klasických SŘBD jsme za nejslabší místo zpracování (z hlediska času) považovali diskové operace, případně přesuny bloků mezi operační pamětí a diskem. V distribuovaných databázích mají největší časovou náročnost operace přesunu dat po komunikační lince mezi uzly sítě. Tím vznikají některé problémy, které jsme u klasických systémů nepoznali. Pro řešení distribuce dat v DDBS existuje několik optimalizačních technik. Obecně platí, že je data nutno umisťovat co nejblíže místům jejich vzniku nebo využívání, pokud tomu nebrání např. parametry počítačů v síti ap.

Vzhledem k vyšší možnosti poruch komunikačních linek a uzlových počítačů musí být relace uloženy tak, aby byla zajištěna jejich dostupnost i při výpadku některých částí sítě. Relací zde rozumíme logický celek, ale fyzicky může být implementována nejen jako jeden datový soubor. Jak již bylo uvedeno výše r*eplikace* spočívá v uchování kopií relací v různých uzlech. Důvodem je minimalizace problémů při poruchách v jednom uzlu, které by znemožnily přístup k lokálním

relacím. Tak jsou navíc data v lokálních bázích připravena k použití okamžitě, bez nutnosti přenosu.

Průvodce studiem

Používají se i systémy s kompletní replikací, kdy v každém uzlu je kompletní kopie celé databáze, což maximalizuje spolehlivost, dosažitelnost a výkon při dotazování, ale maximální je i cena paměťového prostoru a komunikace - například při aktualizacích dat.

Horizontální fragmentace obsahuje n-tice relace

Vertikální fragmentace obsahuje podmnožinu atributů

Kombinovaná fragmentace obsahuje vertikální fragment s horizontální fragmentací. nebo horizontální fragment s vertikální fragmentací

Urychlení výběru dat však má za následek ztížení aktualizace, protože všechny kopie musí být aktualizovány současně a v průběhu aktualizace je nutno uzamknout aktualizovaná data ve všech uzlech sítě. Proto se v DDBS ukládají v kopiích především ta data, ke kterým je potřebný rychlý přístup a která nejsou často aktualizována, příp. která jsou pro systém mimořádně důležitá. Jsou to např. různé číselníky, registry ap. Obecně tedy stupen replikace záleží na četnosti změn v databázi. Pro menší četnost změn je lepší použít většího počtu kopií. Replikace zvyšuje výkon při operaci READ, ale snižuje výkon pro operace UPDATE.

Fragmentace znamená rozložení relace na části (fragmenty), které jsou umístěny v různých uzlech sítě. Může jít o horizontální fragmentaci, kdy se v různých uzlech ukládají části relace rozložené do skupin řádků, nebo o vertikální fragmentaci, kdy se v uzlech ukládají různé projekce relace. Fragmentace se provádí tak, aby bylo možno z fragmentů získat původní relaci standardními operacemi nad relační databází, např. sjednocením nebo spojením.

Obě metody se často kombinují tak, že se v distribuované bázi uchovávají kopie fragmentů v různých uzlech. V distribuované databázi musí být unikátní jména všech položek, tedy ani dvě lokální relace nesmí používat stejná jména pro různé položky. Jednou z možností, jak tento problém řešit, je použití *centrálního systémového katalogu*. To má však nevýhody v tom, že katalog se stává nejužším článkem systému, protože požadavky na paralelní činnost mohou být vysoké. Má-li centrální katalog poruchu, havaruje celý systém. Samostatná práce nad lokálními databázemi je velmi omezena.

Alokace je proces vytváření fyzicky uložených dat fragmentů na optimálním uzlu sítě

Jiná možnost spočívá v používání prefixů odvozených ze jména uzlového počítače. Tím se zase snižuje nezávislost označení dat na implementaci. Kromě unikátních jmen položek musí mít také každá kopie (replika) a každý fragment své unikátní jméno : UZEL.RELACE.FRAG.REPL

Replikace je proces vytváření a reprodukování kopií dat na několika místech sítě

Není možno požadovat od uživatele distribuované databáze, aby sám rozhodoval, kterou kopii relace bude používat. To je úlohou systému a systém se musí také postarat o modifikaci všech kopií, byla-li provedena modifikace relace. Stejná zásada platí pro použití fragmentace. Existence, druh a jména fragmentů jsou vnitřní záležitostí systému. Stejně tak rozhodnutí o tom, zda pro požadovanou operaci je nutné sestavení celé relace, nebo zda lze operaci provést efektivněji nad fragmenty.

K tomu existuje tabulka přezdívek (alias), kde jsou uvedeny uživatelské názvy objektů i názvy objektů ve vnitřní reprezentaci systému. K tabulce existuje algoritmus, který určuje fragmenty a kopie. Pro zjištění nákladů na zpracování akce se musí brát v úvahu rozmístění kopií a fragmentů, možnost paralelního zpracování částí akce, délky komunikačních linek mezi uzly, které se na zpracování podílejí a množství informace, kterou je nutno přesouvat.

Ukažme si dále strategie, které můžeme v distribuované databázi použít pro zpracování dotazu obsahujícího operaci spojení.

Př. : Máme tři relace R1, R2, R3, které nejsou ani replikovány, ani fragmentovány. Každá z relací je umístěna v jiném uzlu U1, U2, U3 a v dalším uzlu U4 položíme dotaz, který vyžaduje provést operaci spojení R1 + R2 + R3.

Můžeme použít např. následující strategie, nejčastěji s využitím polospojení :

1. Kopie všech tří relací přemístíme do uzlu U4 a zpracujeme dotaz v U4.
2. Kopii R1 pošleme do R2 a v U2 zpracujeme spojení R1 + R2. Výsledek pošleme do R3 a v U3 zpracujeme (R1 + R2) + R3. Výsledek potom pošleme do U4.
3. Kopie zpracováváme podobně jako při strategii 2, ale použijeme jiné pořadí.

Nebo pro 4 fragmenty je možno spustit paralelně spojení (R1 + R2) v uzlu U1, (R3 + R4) v uzlu U3 a pak výsledek spojit do uzlu U5.

Obecně není možno rozhodnout, která strategie je lepší. K tomu je třeba vzít v úvahu

- množství dat, které je třeba přesunout,
- cenu za přenos bloku dat mezi jednotlivými uzly,
- rychlost zpracování v jednotlivých uzlech.

a vhodným optimalizačním algoritmem zvolit nejlepší strategii. Řeší se problémy typu:

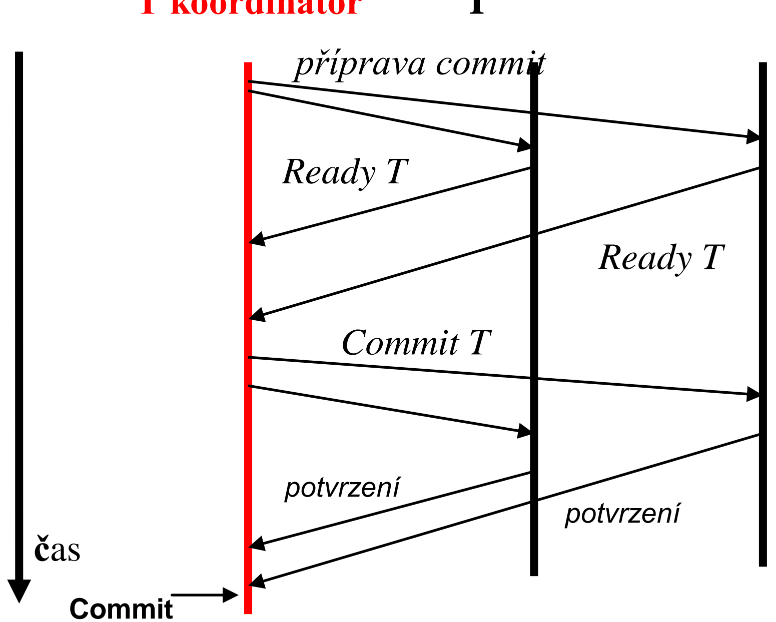
Pokud existují kopie dat pro zpracování dotazu, z kterého místa se načtou? Kdo bude koordinovat provádění dotazu? Kde se provede vyčíslení?

9.3 Paralelní zpracování v distribuovaných databázích

Zajišťování atomického charakteru transakcí v distribuované bázi je řádově složitější, než u lokálních sítí, protože možné poruchy jsou mnohem složitější. V centralizovaných DDBS řídí pořadí vykonávání operací jediný plánovač umístěný v centru DDBS. Protože plánovač má kontrolu nad všemi daty v DDBS, mohou se bez problémů použít typy plánovačů pro klasické SŘBD. Připomeňme, že každá transakce může číst a měnit hodnoty objektů v libovolných lokálních bázích dat. Proto případné zrušení transakce, například při distribuovaném uváznutí, je spojeno s většími problémy. Zrušená transakce musí vrátit původní hodnoty objektům, které již změnila. V distribuované bázi to může být zdlouhavá a nákladná operace, proto budou výhodnější takové plánovače, které pracují bez rušení transakcí. U plánovačů pracujících s časovými razítky je nutno sladit lokální hodiny - tak, aby časová razítka byla navzájem různá, i když pocházejí z různých uzlů. V případě decentralizovaných DDBS jsou plánovače v každém lokálním SŘBD. I v decentralizovaných DDBS je možno použít typy plánovačů z klasických SŘBD. Při zamykání objektů zamyká a odemyká každý lokální plánovač objekty ve své bázi. Problémem je ale kontrola korektního dodržování dvoufázového protokolu zamykání. Řešením může být pozdržení odemknutí všech objektů zamknutých pro transakci, dokud transakce neskončí a pak vyslat všem plánovačům zprávu o skončení transakce. Pak je možno objekty uvolnit.

Hojně využívanou možností je použití protokolu s dvoufázovým potvrzením. Předpokladem je autonomní činnost uzlů s použitím lokálních žurnálů pro případné použití operací ROLLBACK. Jeden z uzlů převezme roli koordinátora a určuje, kdy je transakce potvrzena nebo odmítnuta a navrácena do výchozího stavu. Komunikace probíhá prostřednictvím zasílání zpráv mezi koordinátorem a ostatními uzly.V první fázi koordinátor pošle všem zúčastněným uzlům zprávu příprava na potvrzení T. Každý autonomní uzel se rozhodne, zda T potvrdí, nebo odmítne svou část T. Pokud potvrdí commit např. < Ready T>, nemůže již přejít do stavu abort. Jinak pošle zprávu < Do not commit T> . Druhá fáze začíná po získání všech odpovědí koordinátorem. Pokud odpovědi nedojdou v odhadnutém čase, předpokládá se zamítnutí. Koordinátor pošle zprávu < Commit T> nebo < Abort T> na všechny zúčastněné uzly. Následně uzly provedou požadovanou akci a informují koordinátora o provedení operace.

Příklad pro úspěšně potvrzené přípravy potvrzení v protokolu s dvoufázovým potvrzením:



Obr. 20 Příklad průběhu dvoufázového potvrzovacího protokolu

V decentralizovaném systému se těžko zjišťuje uváznutí, to nemůže detekovat žádný lokální plánovač. Proto je výhodné uváznutím předcházet, např. použitím plánovačů pracujících s časovými razítky, kde k uváznutí nedochází.

Průvodce studiem

Transparentnost transakcí – DDBS musí zajistit atomičnost globální transakce pro zachování integrity a konzistence. Globální transakce se rozdělí na subtransakce – agenty, jednu na každém uzlu sítě. DDBS musí zajistit synchronizaci subtransakcí, ale zároveň se nesmí ovlivňovat lokální a globální transakce a v případě ztráty zprávy, poruchy sítě, atd. musí systém umět zareagovat na tyto poruchy.

Shrnutí V distribuovaných databázích mohou být data rozmístěna a rozložena v uzlových SŘBD počítačové sítě horizontálně – relace má své n-tice rozděleny do podmnožin na několika uzlech, vertikálně – relační schéma se dekomponuje do několika subschémat, která jsou umístěna v jednotlivých uzlech nebo kombinovaně oběma způsoby (fragmentace). Pro odolnost proti poruchám se typicky fragmenty kopírují na další uzly sítě – provádí se replikace. V distribuovaných databázích se jedna logická transakce skládá z částí, které se zpracovávají v různých uzlech. Obtížná synchronizace transakcí se provádí často podle dvoufázového potvrzovacího protokolu, který testuje shodu mezi uzly, které se zúčastňují transakce. V první fázi uyel-koordinátor zahájí přípravu potvrzení a čeká na zprávy od ostatních uzlů. V druhé fázi vyšle zprávu o provedení operace commit nebo abort podle toho, jak vyhodnotí odpovědi z první fáze – provedení commit předpokládá předchozí shodu všech uzlů na provedení commit.

Rovněž zamykání musí být podobně koordinováno i s ohledem na replikaci dat. Distribuovanost databází zvyšuje jejich složitost, náchylnost k chybám i cenu celého systému. Výrazně se zvyšuje podíl režie systému na celkovém zpracování. Řadu problémů zpracování dat ovšem nelze řešit efektivně jiným způsobem. Důležitým požadavkem je, aby se problémy týkající se programového vybavení nedotýkaly uživatele a jeho způsobu práce s distribuovanou databází a aby byl distribuovaný systém zabezpečen pro případ poruchy některého z uzlových počítačů, nebo některého spojovacího vedení, musí mít zabudován systém detekce chyb a možnost rekonfigurace.

Pojmy k zapamatování

- Distribuovaný systém, autonomie, topologie
- Horizontální a vertikální fragmentace, alokace, replikace
- Centrální systémový katalog
- Globální transakce, dvoufázový potvrzovací protokol

Kontrolní otázky

61. *Jaké vlastnosti a definici má distribuovaná databáze?*
62. *Jaká může být topologie a rozložení výpočetního výkonu v DDS?*
63. *Co znamená fragmentace, alokace a replikace dat?*
64. *Co je to protokol s dvoufázovým potvrzením?*

10 Závěr

V tomto textu jsou formulovány základní partie z databázové technologie se zaměřením na relační SŘDB. Text je zpracován tak, aby upozornil na všechny podstatné rysy jednotlivých témat, ale nemůže v daném rozsahu textu detailně rozebrat celou problematiku do šířky se všemi souvislostmi a prakticky využitelnými dovednostmi. Na publikaci navazují další texty věnující se problematice informačních systémů.

Monografie, věnující se táto problematice jsou obvykle desetkrát rozsáhlejší Předpokládám proto, že po nastudování tohoto textu sáhne studující po dalších materiálech, ve kterých si doplní potřebné detaily a hlavně na dostupném databázovém systému ověří a zdokonalí své teoretické znalosti.

11 Seznam literatury

[Garcia-Molina02] GARCIA-MOLINA, H., ULLMAN, J. D., WIDOM, J. *Databáse Systems: The Komplete Book*. Prentice-Hall, Inc., Upper Saddle River, New Persey 07458, 2002. ISBN 0-13-031995-3.

[Connoly02] CONNOLLY, T., BEGG, C. *Databáse Systems: A Practical Approach to Design, Implementation, and Management*.3. vyd. Pearson Education Limited, Edimburgh Gate Harlow Essen CM20 2JE England, 2002. ISBN 0-201-70857-4.

[Pokorný92] POKORNÝ, J. *Databázové systémy a jejich použití v informačních systémech* . 1. vyd. Akademia Praha, Praha, 1992. ISBN 80-200-0177-8.

[Pokorný99] POKORNÝ, J. *Konstrukce databázových systémů*. 1. vyd. ČVUT, Zikova 4 Praha 6, Praha, 1999. ISBN 80-01-01935-8.

12 Seznam obrázků

Obr. 1 Porovnání klasické a elektronické technologie ... 11
Obr. 2 Příklad ER diagramu ... 12
Obr. 3 Úrovně abstrakce ... 15
Obr. 4 Struktura dat agendového typu ... 16
Obr. 5 Struktura dat systémů pro zpracování souborů ... 17
Obr. 6 Struktura dat SŘBD ... 17
Obr. 7 Funkční architektura DBS ... 20
Obr. 8 Dr. Peter Chen, autor ER modelu (1970) ... 29
Obr. 9 Příklad ER diagramu ... 31
Obr. 10 Dr. Edgar Frank Codd ... 36
Obr. 11 Vztah vyjadřovací síly DATALOGu a relační algegry ... 57
Obr. 12 Schéma relací na SQL Serveru ... 65
Obr. 13 Faginovo dynamické hašování ... 81
Obr. 14 Etapy optimalizace dotazu ... 86
Obr. 15 Postup normalizace relací ... 104
Obr. 16 Stavový diagram transakce ... 109
Obr. 17 Stavy transakce ke kontrolnímu bodu a poruše ... 111
Obr. 18 Metoda stínového stránkování ... 112
Obr. 19 Klasifikace DDBS ... 124
Obr. 20 Příklad průběhu dvoufázového potvrzovacího protokolu ... 129

13 Rejstřík

. multizávislost, 103

Agregace, 27

agregované funkce, 61

Algoritmus určení neredundantního pokrytí, 99

Algoritmus Určení příslušnosti, 99

Algoritmus Určení uzávěru, 98

anomálie při vkládání, 95, 101

anomálie při vypouštění, 95, 102

Armstrongovy axiomy, 96

autonomie, 124

B^+ stromy, 78

Boyce-Coddova normální forma, 102

cena dotazu – kriteriální funkce, 88

Časové razítko, 119

Členství ve vztahu, 28

čtvrtá normální forma, 104

Dekompozice, 104

Dekompozice relačního schématu, 100

Distribuce, 125

Doménový relační kalkul, 55

druhá normální forma, 101

dvoufázový protokol, 119

Dynamické hašování, 80

EER, 29

Ekvivalentní výrazy při optimalizaci dotazu, 87

entitní typ

typ entity, 26

ER model, 24, 25, 29

Fragmentace, 127

Funkční závislosti, 95

granularita, 117

hašovací funkce, 91

Hašováné spojení, 91

hašování, 80

Heuristika optimalizace výrazu, 88

Hnízděné cykly, 90

Hustý index, 77

Indexované hnízděné cykly, 91

ISA hierarchie, 27

ISAM, 78

Jazyk SQL, 57

Kardinalita vztahu, 27

kontrolní bod, 111

konzistence dat, 109

Materializace, 93

Metoda stínového stránkování, 111, 112

minimální pokrytí, 98

nekonzistence, 95

Normální formy relací, 101

N-ticový relační kalkul, 54

optimalizace dotazu, 86

OQL, 68

Organizace souborů, 74

paměti typu CAHE, 73

Pipelining, 93

Pohled view, 63

Primární index, 76

protokol s dvoufázovým potvrzením, 128

první normální forma, 101

QBE, 67

RAID, 73

redundance, 95, 101

Relační algebra, 50

Replikace, 127

Řídký index, 77

Sekundární disková paměť, 73

Sekundární index, 76

Sekvenční soubory, 75
Shlukované indexy, 79
Shlukování, 81
Silný entitní typ, 26
Slabý entitní typ, 26
Soubor, 72
soubor s proměnnou délkou záznamu, 82
Soubory s proměnnou délkou záznamu, 82
SŘBD, 17, 18
Terciální paměť, 73
transakce, 108
třetí normální forma, 102
UNDO REDO, 111
Určení uzávěru FZ, 98
uspořádatelnost, 115
uváznutí, 118, 120
Uzamykací protokoly, 116
Víceúrovňový index, 77
Ztráta aktualizace, 113