Mirochove poznámky z
Teórie databázových a poznatkových systémov
Verzia 2.0
(Podklad k prednáške J. Šturca)

Upozornenie: Tento text nemá slúžiť ani ako náhrada za skriptá ani ako náhrada za prednášku. Predstavuje len spísané prepracované poznámky z tohto predmetu doplnené z literatúry o najdôležitejšie pasáže. Oproti verzii 1.01 je doplnený o pasáže týkajúce sa transakčného spracovania, QBE (bez záruky), distribuovaných databáz a fyzickej organizácie dát. Všetky riešené príklady sú bez záruky.

Miroch Chladný v1.01
E-mail: 2chladny@fmph.uniba.sk
Martin Gažák v2.00
E-mail: 4gazak@st.fmph.uniba.sk

! ! ! Dokument vo výstavbe ! ! !

Obsah

1. Historický úvod - vývin informačných systémov

- súvisí s vývojom operačných systémov

1. **Prehistorické systémy** (50.roky): aplikačný program obsahuje aj všetky spracovávané data.
2. **Systémy s monitorom** (zač. 60.rokov): umožňuje sa dávkové spracovávanie viacerých množín dát, popis fyzickej štruktúry dát je pevnou súčasťou aplikačného programu.
3. **Systémy pre spracovanie súborov** (60.roky): data sú uložené v externých súboroch, popis fyzickej štruktúry dát je súčasťou aplikačného programu, umožňuje sa prístup viacerých aplikačných programov k dátam súčasne.
4. **Systémy riadenia báz dát (SRBD) – DB-servers** (70.roky): Data sú uložené v externej databáze, ich popis nie je súčasťou aplikačného programu, t.j. programy sú nezávislé na fyzickom uložení dát, pristupuje sa k nim len skrze DB-server. Databázový systém je tvorený SRBD a databázou (obsahujúcou data). So SRBD sa komunikuje pomocou *jazyka pre definíciu dát* (Data Definition Language - DDL) - jeho účelom je správa logických štruktúr dát a *jazyka pre manipuláciu s dátami* (Data Manipulation Language - DML) - jeho účelom je špecifikovanie dotazov. SRBD obsahuje:
   a) **modul riadenia súborov**: spravuje súbory na vonkajších médiách, obdoba podobného modulu z operačných systémov
   b) **modul riadenia databázy**: rozhranie medzi súbormi a aplikačnými programami.
   c) **modul vyhodnocovania dotazov**: prekladá dotazy do vnútornej formy a optimalizuje ich.
   d) **modul preprocesora jazyka pre manipuláciu s dátami**: zaisťuje konverziu príkazov DML do procedúr hostiteľského jazyka.
   e) **modul kompilátora pre spracovanie príkazov definičného jazyka**: vytvára a spravuje potrebné štruktúry.

2. Databázové modely, typy databázových sytémov

- databázový model predstavuje všeobecnú metódu, akou sa modeluje skutočnosť
- má zmysel stanoviť isté podmienky, ktoré charakterizujú dobrý (konzistentný) stav databázy - **integritné ohraničenia** (IO):
  a) **inherentné** - sú vlastnosťami konštruktov modelu, t.j. tieto podmienky automaticky spĺňa všetko, čo je modelom namodelované.
  b) **explicitné** - sú explicitne zadané programátorom, aplikačné programy obyčajne obsahujú procedúry, ktoré overujú, či ich vstupné data spĺňajú; v prípade, že nie, vyhlásia chybu (napr. v tabuľke zamestnancov nemôžu existovať dvaja zamestnanci s rovnakým osobným číslom).
  c) **implicitné** - sú dôsledkami explicitných

Databázové modely - generácie:

1. **Primitívne databázové modely** (do konca 60.rokov): všetko ponechané vôli programátora.
2. **Klasické (hodnotovo orientované) databázové modely** (od začiatku 70.rokov): vzťahy sveta objektov sa realizujú vzťahmi medzi záznamami, t.j. jednému záznamu sú prístupné záznamy, ktoré s ním logicky súvisia (vzťah má obyčajne orientáciu), t.j. aj naďalej sú súborovo orientované, avšak na podstatne systematickejšej úrovni, navyše operácie typické pre model sú podporované SRBD. Patria sem:
   a) Sieťový DB-model
   b) Hierarchický DB-model
   c) Relačný DB-model
3. **Sémantické (objektovo-orientované) databázové modely**: Namiesto komplikovaných IO sa používajú konštrukty založené na pojmoch entita, atribút, vlastnosť, objekt (patrí sem napríklad Chenov E-R model). Cieľom je čo možno najprehľadnejšie modelovanie vzťahov reálneho sveta.

Pôvodným zámerom bolo len vytvoriť nástroj na modelovanie. V súčasnosti už existujú SRBD založené na tomto modeli (O-O databázy).

4. **Aplikačne orientované databázové modely**: špeciálne, napr. CAD systémy, systémy na spracovanie experimentálnych dát, systémy (polo)automaticky generujúce aplikačné programy (software 4. generácie: 4GL), grafické systémy.

2.1. Sieťový a hierarchický model dát - navigačné databázy

- modelujú sa vzťahy medzi záznamami(record-ami).
- databázy založené na sieťovom modeli dát sú **objektovo orientované** , t.j. každý záznam má svoju identitu, môžu existovať dva rovnaké záznamy líšiace sa polohou v sieti.
- základným typom vzťahu je C-množina: je to vzťah medzi záznamami, typu $\overleftarrow{1:N}$, t.j. jednému záznamu (owner) prislúcha viacero iných (members) - ide o inherentné IO. Je implementovaná smerníkom, ktorým owner ukazuje na spájaný zoznam memberov.
- vyhodnocovanie dotazu je charakterizované systémom smerníkov (currency pointers), ktoré určujú aktuálnu transakciu, v nej aktuálnu C-množinu a v nej aktuálne spracovávaného membera.
- priama reprezentácia relácií many:many nie je povolená, na reprezentáciu relácií many:many sa používajú tzv. spojovacie záznamy.

Príklad 1: Majme záznamy nasledovného formátu :

COURSES(DEPT, NUMBER, INSTRUCTOR)

STUDENTS(ID, NAME, ADDRESS, STATUS)

Na reprezentovanie vzťhov medzi študentami a kurzmi sa použijú záznamy formátu

ENROLL(SECTION, GRADE) : $COURSES \leftarrow ENROLL \rightarrow STUDENTS$

- dotazy v sieťovom modeli sledujú linky - smerníky, nevýhodou sieťového modelu je fakt, že výsledkom dotazu nemusí byť sieť ani časť siete, takže nemožno konštruovať siete bez použitia DDL.
- Pri hierarchickom modeli dát má graf vzťahov (C-množín) podobu sytému rovanko orientovaných disjunktných ciest formálne spojených do jedného stromu pomocou systémového uzla (koreňa). Záznamy obsahujúce data potom vytvárajú les (spojený systémovým uzlom do stromu). Príslušné cesty zodpovedajú najpravdepodobnejším dotazom. Je malá šanca, že by sa typy záznamov, ktoré tvoria uzly týchto ciest naozaj vyskytovali iba v jedinej. Konflikt možno riešiť buď použitím duplicitných záznamov (čo sťažuje update, zhoršuje konzistentnosť a zaberá zbytočné miesto) alebo zavedením virtuálnych záznamov (podobne ako virtuálne linky v systémoch súborov), t.j. v strome záznamov je uzol, označený ako virtuálny, ktorý obsahuje smerník na skutočný výskyt záznamu v inom strome.

Algoritmus 1(konverzia siete na hierarchiu):

`INPUT`: network

`OUTPUT`: hierarchy

```
procedure BUILD(n);
   make n selected;
   for each link from some node m to n do
      if m is not selected then begin
         make m a child of n;
         BUILD(m);
      end
      else /* m was previously selected */
         make virtual m be a child of n
end

/* main program */
make all nodes unselected;
while not all nodes are selected do begin
   pick an unselected node n;
      /* prefer a node n with no links to unselected nodes,
```

```
      and prefer a node with many incoming links */
    BUILD(n);
  end
```

Navigačné databázy - vyhodnocovanie dotazu: prebieha v 2 krokoch

1. LOCATE
2. GET, GET NEXT

- Tieto fázy sú typické pre všetky súborovo orientované modely. Rozdiel je v tom, kto a ako ich zabezpečuje.
- V navigačných databázach je riadenie tejto činnosti ponechané na aplikačného programátora. Realizuje sa pomocou tzv. *procedurálnych* dotazovacích jazykov.
- Navigačné databázy sú typické pre sieťový a hierarchický model dát.

2.2. Relačný model dát (RMD)

- je založený na manipulácii s tabuľkami (pôvodná stratégia bola "Nothing, but tables").
- tabuľka je istou abstrakciou súboru, preto aj RMD je klasický súborovo-orientovaný, má však veľmi dobre prepracované matematické pozadie založené na logike 1.rádu. Tabuľku možno totiž chápať ako schému (konečej) relácie a systém tabuliek ako schému (konečnej) interpretácie predikátových symbolov jazyka logiky 1.rádu. Samotnú interpretáciu tvorí obsah tabuliek.
- Binárna relácia je usporiadaná trojica $\mathbf{R}^* = (Dom, Codom, R \subseteq Dom \times Codom)$
- n-árna relácia je usporiadaná dvojica $\mathbf{R}^* = (\{D_i\}_{i=1}^n, R \subseteq D_1 \times \cdots \times D_n)$. Je teda (len formálny) rozdiel medzi n-árnou reláciou pre $n = 2$ a binárnou reláciou.
- Stĺpce tabuľky (nazývané atribúty) sú pomenované, narozdiel od matematickej relácie, kde sa zložky n-tice rozlišujú pozíciou. Množiny D_i sú definičným oborom atribútov.
- Relačná databáza $\mathbf{R}^*$ je potom množina relácií $\{R_1^*, \ldots, R_k^*\}$.
- Relačná schéma databázy $\mathbf{R}$ je množina schém relácií $\mathbf{R} = \{R_1, \ldots, R_k\}$, kde schéma R_i je $R_i(A_{i_1} : D_{i_1}, \ldots A_{i_n} : D_{i_n})$, pričom A_{i_j} sú atribúty.
- Inherentným IO je, že hodnoty atribútov sú ďalej nedeliteľné (tabuľka je automaticky v 1.NF).
- Špeciálnym typom explicitných IO sa hovorí závislosti a multizávislosti.
- Ďalším typom explicitných IO sú
  a) **kľúč schémy relácie**: minimálna množina atribútov, ktoré jednoznačne identifikujú riadok tabuľky, t.j. nemôžu existovať 2 rôzne riadky s tou istou hodnotou kľúča. Jeden z kľúčov, na ktorý sa kladú ďalšie podmienky, sa nazýva *primárny.*
  b) **obmedzenie cudzieho kľúča:** Nech schéma R_1 obsahuje atribúty kľúča K schémy R_2. Uvedená podmienka značí, že hodnota každého riadka R_1^* na množine K (K-hodnota) je K-hodnotou nejakého riadka v R_2^*

2.3. Sémantický model dát (E-R diagramy) a konceptuálne modelovanie

- podrobnejšie viď [4].
- E-R diagram je tvorený poprepájanými *entitnými množinami, vzťahovými množinami a atribútmi.*
- Entitná množina (typ) je množina entít (nazývaných aj objekty), ktoré majú podobné vlastnosti (zhodné typy atribútov). Entitné množiny sú znázorňované obdĺžnikom.
- Hodnoty atribútov charakterizujú stav entity. Atribúty sú znázorňované krúžkami pripojenými k obdĺžniku svojej entitnej množiny. Množina atribútov tvoriacich kľúč sa podčiarkuje.
- Vzťahy medzi entitami sú zachytené vzťahovými množinami. Znázorňujú sa kosoštvorcom s pripojením k participujúcim entitným množinám s vyznačením násobnosti (koľko entít jednej entitnej množiny je vo vzťahu s koľkými ostatných množín) $\overleftrightarrow{1:1}$, $\overleftarrow{1:N}$, $\overline{M:N}$, príp. ďalšími.

- Špeciálne sú slabé entitné množiny, ktoré sú v existenčnom, či identifikačnom vzťahu k nejakej inej entitnej množine. Existenčný vzťah (EX) je vzťah, ktorý určuje, že k entite jednej entitnej množiny musí existovať entita druhej entitnej množiny (viď. obmedzenie cudzieho kľúča). Identifikačný vzťah (ID) je vzťah, ktorý rozlišuje entity jednej entitnej množiny pomocou entít druhej, ktoré by inak boli nerozlíšiteľné. (t.j. príslušná entitná množina nemá kľúč). Identifikačný vzťah implikuje existenčný, naopak nie.

2.4. Objektovo-orientovaný model

Základné črty objektovo orientovaných databáz (či už navrhovaných alebo implementovaných):

1. **Komplexné objekty** - relačné databázy poskytujú len dva druhy objektov, záznamy a tabuľky. OO databázy poskytujú aj ďalšie typy, najmä polia (array) a zoznamy (list). Komplexné objekty možno skladať - objekt môže byť samostatná entita aj hodnota atribútu. Objekty môžu obsahovať metódy - procedúry a funkcie na manipuláciu s atribútmi.
2. **Identita objektov** - každý objekt má svoju identitu nezávisle na hodnote svojich atribútov. Identita nadobúda formu mena premennej resp. adresy v pamäti. Identifikátory objektov sa využívajú namiesto cudzích kľúčov pri vzájomnom referencovaní objektov.
3. **Hierarchia typov, dedenie** - OO databázy umožňujú deklaráciu typov aj pre komplexné objekty. Zároveň možno vytvárať hierarchie užívateľských typov resp. tried za účelom zdieľania atribútov a funkcií medzi typmi.
4. **Polymorfizmus a dynamické viazanie** - s OO programovaním súvisí polymorfizmus, prinajmenšom vo forme prekrývania (v rôznych typoch rôzne funkcie rovnakého mena), prípadne inkluzívny a parametrický polymorfizmus. S polymorfizmom súvisí dynamické viazanie t.j. zobrazenie mien funkcií do definícií (=implementácií) až počas behu programu. Podrobnejšie o OO programovaní viď napr.[4].
5. **Logická dátová nezávislosť** - RDBMS umožňujú pomocou pohľadov len čiastočné dosahovanie logickej dátovej nezávislosti t.j. zmenu štruktúry databázy bez nutnosti meniť aplikačné programy. Logická dátová nezávislosť predpokladá využitie princípov **ukrývania dát a verejných interfejsov**. Keďže k ukrývaným dátam sa pristupuje len prostredníctvom interfejsov, ich reprezentáciu možno zmeniť bez dopadu na aplikačné programy.
6. **Spájanie databázového a programovacieho jazyka** - v relačných databázach sa príkazy napr. SQL vyvolávajú z hosťovského jazyka. OO databázy poskytujú výpočtovo úplný jazyk (založený napr. na C++ alebo smalltalku).

2.5. Konceptuálne modelovanie

- Návrhu databázy predchádza modelovanie skutočnosti.
- Najabstraktnejšiu časť predstavuje konceptuálny návrh (návrh koncepcie). Obzvlášť výhodným jazykom je preňho jazyk sémantického modelu.
- Konceptuálny návrh sa zobrazuje do logického návrhu databázy, ktorý je v jazyku výsledne použitého DB-modelu (napr. relačného). V prípad O-O databáz môže byť konceptuálna schéma databázy zhodná s logickou.
- Databázu používajú rôzny používatelia, každý len najekú jej časť, a to navyše každý inak. Tomu zodpovedjú externé schémy databázy (je ich viac). Každej externej schéme zodpovedá na logickej úrovni pohľad (view), odvodený od centrálnej logickej schémy.
- Je zmysluplné skúmať zobrazenia schém v jednom modeli na schémy v inom modeli, dokonca aj z jedného modelu na ten istý (iterácia, optimalizácia).

3. Dotazovacie jazyky

- Rozdelenie:
  a) Jazyk pre definíciu dát
  b) Jazyk pre manipuláciu s dátami
  c) Jazyk pre riadenie dát

3.1. Jazyky pre definíciu dát

- slúžia na popis konštruktov modelu (objektov v O-O databázach, relácií v RMD, ...).

Časť SQL *(Standard query language)* **slúžiaca na definíciu dát:**

- vytvorenie tabuľky: `CREATE TABLE` <meno>(atr$_1$ typ$_1$, atr$_2$ typ$_2$, ...), kde typ$_i$ je napr.
  `INTEGER` - prirodzené číslo,
  `CHAR(n)` - reťazec pevnej dĺžky,
  `VARCHAR(n)` - reťazec premenlivej dĺžky a pod.
- vytvorenie pohľadu: `CREATE VIEW` <meno>[(atr$_1$, atr$_2$, ...)] `AS` <select-clause>
- zrušenie tabuľky: `DROP TABLE` <meno>
- zrušenie pohľadu: `DROP VIEW` <meno>
- vytvorenie indexu: `CREATE INDEX` <meno-indexu> `ON` <meno-tabulky>(<meno-atributu>)
- zrusenie indexu: `DROP INDEX` <meno-indexu>

3.2. Jazyky pre manipuláciu s dátami

Definícia 1(jazyka logiky 1. rádu):
Definícia **termu** :
1. premenná je term.
2. ak $T_1, \dots, T_n$ sú termy a F je n-árny funkčný symbol ($n \geq 0$), potom $F(T_1, \dots, T_n)$ je term.

Definícia **formuly** :
(i) Ak R je n-árny predikátový funkčný symbol ($n \geq 0$) a $T_1, \dots, T_n$ sú termy, potom $R(T_1, \dots, T_n)$ je atomická formula (špeciálne postavenie majú aritmetické preikáty $>, \geq, =, \leq, <$).
(ii) Ak P, Q sú formuly, potom $\neq P, P \wedge Q, P \vee Q$ sú formuly.
(iii) Ak P je formula, x je premenná, potom $(\forall x)P, (\exists x)P$ sú formuly.

Interpretácia jazyka logiky 1.rádu je trojica skladajúca sa z:
- neprázdnej množiny D, ktorá sa nazýva doménou interpretácie
- množiny n-árnych funkcií f pre všetky n-árne funkčné symboly F
- množiny n-árnych predikátov r pre všetky n-árne predikátové symboly R

3.2.1. Doménový relačný kalkul

- Vyjadrovanie dotazov v tvare $\{x_1, x_2, \dots, x_n | A(x_1, x_2, \dots, x_n)\}$, kde $F_{var}(A) = \{x_1, x_2, \dots, x_n\}$ a A je formula jazyka logiky 1.rádu. Takéto vyjadrovanie je vzhľadom k matematickému pozadiu prirodzené.
- Vyhodnotenie dotazu: Nech (D, I) je interpretácia predikátových symbolov jazyka logiky 1.rádu daná obsahom databázy $\mathbf{R}^*$, kde

$$D = \bigcup_{i=1}^{n} Dom(R_i).$$

Potom

$$[\{x_1, x_2, \ldots, x_n | A(x_1, x_2, \ldots, x_n)\}]^I = \{h|_{F_{var}(A)} | Val_{I,h}(A) = TRUE\},$$

kde $h : Var \rightarrow D$ je ohodnotenie premenných v interpretácii..

- Základný problém: zabezpečiť, aby výsledná množina bola konečná, aj keď D bude nekonečné.
- Najvýhodnejšie by bolo uvažovať len také dotazy, že ich vyhodnotenie nezávisí na pridaní nového prvku do D, t.j. závisí len na $\mathbf{R}^*$ (definitné, doménovo nezávislé formuly). Zistiť, či daná formula je doménovo nezávislá je však nerozhodnuteľné.
- Reálne dotazovacie jazyky postavené na DRC využívajú len časť formúl DRC, tú, kde je garantovaná doménová nezávislosť. Preto sa definujú "bezpečné" formuly ako podmnožina doménovo nezávislých. Kritériá bezpečných formúl by mali spĺňať nasledujúce vlastnosti :
  (1) Každá "bezpečná" formula je doménovo nezávislá
  (2) Z tvaru formuly možno ľahko určiť, či je bezpečná
  (3) Formuly vyjadriteľné v reálnych dotazovacích jazykoch sú bezpečné

Definícia 2(bezpečného relačného kalkulu (safe DRC)): Formula F sa nazýva bezpečná $\Longleftrightarrow$:

(1) F neobsahuje všobecný $\forall x$ kvantifikátor - $(\forall x)F$ sa nahrádza $\neg(\exists x)\neg F$
(2) pri použití operátora OR majú obe spojené formuly (povedzme $F_1 \vee F_2$) rovnakú množinu voľných premenných
(3) uvažujme maximálnu podformulu pozostávajúcu z konjunkcií formúl $F_1 \wedge F_2 \wedge \cdots \wedge F_m$. Potom všetky premenné, ktoré sú voľné v nejakej z F_i, musia byť bezpečné, t.j:
  (i) premenná A je bezpečná, ak je voľná v nejakom F_i, kde F_i nie je aritmetické porovnávanie a nie je negované
  (ii) ak F_i je tvaru $X = a$ alebo $a = X$, kde a je konštanta, tak X je bezpečné
  (iii) ak F_i je tvaru $X = Y$ alebo $Y = X$, kde Y je bezpečná, tak X je bezpečné
(4) operátor $\neg$ sa môže použiť len v rámci konjunkcie typu(3), t.j. $\neg G$ narušuje definíciu bezpečnosti, ak nie je súčasťou nejakej konjunkcie splňujúcej (3) takej, že v nej je aspoň 1 negovaná formula : $\neg G \wedge H_1 \wedge \cdots \wedge H_i$ - aspoň 1 z $H_1 \ldots H_i$ je pozitívna

- Iné kritériá "bezpečných formúl" - napr syntaktické (podľa [2]):
  (1) $A(t_1, t_2, \ldots t_n)$ je tvaru $P_1(t_1) \wedge P_2(t_2) \wedge \cdots \wedge P_n(t_n) \wedge A'(t_1, t_2, \ldots t_n)$, kde t_i sú n-ticové premenné a $P_i \in \mathbf{R}$, $i \in \{1, \ldots, n\}$
  (2) Obmedzenosť kvantifikátorov v A' - sú jedného z tvarov:
  $(\exists x \in P)Q(x) \equiv (\exists x)(P(x) \wedge Q(x))$,
  $(\forall x \in P)Q(x) \equiv (\forall x)(\neg P(x) \vee Q(x))$, kde $P \in \mathbf{R}$.

3.2.2. n-ticový relačný kalul (TRC)

- n-ticový relačný kalkul je variant relačného kalkulu, kde premenné označujú záznamy(n-tice, riadky tabuľky), takže predikáty sa stanú unárnymi vzhľadom na ne. Na referenciu i-teho komponentu n-tice μ sa používa zápis $\mu[i]$, prípadne $\mu[\textit{meno atribútu}]$ na označenie arity n-ticovej premennej zápis $\mu^{(i)}$.
- Vyjadrovanie dotazov v tvare $\{\mu | F(\mu)\}$ kde $F_{var}(F) = \{\mu\}$ a F je formula jazyka TRC (logiky 1.rádu).
- obdobne ako pri DRC sa definuje trieda bezpečných formúl ako podmnožina triedy doménovo nezávislých.

Definícia 3(bezpečného n-ticového relačného kalkulu (safe TRC)): Formula F sa nazýva bezpečná $\Longleftrightarrow$:

(1) F neobsahuje všobecný $\forall x$ kvantifikátor
(2) pri použití operátora OR majú obe spojené formuly (povedzme $F_1 \vee F_2$) len jednu voľnú n-ticovú premennú rovnkú v oboch formulách
(3) uvažujme maximálnu podformulu pozostávajúcu z konjunkcií formúl $F_1 \wedge F_2 \wedge \cdots \wedge F_m$. Potom všetky komponenty voľných n-ticových premenných, ktoré sú voľné v nejakej z F_i, musia byť bezpečné, t.j.:

(i) ak F_i nie je aritmetické porovnávanie a nie je negované a má voľnú premennú μ, potom všetky komponenty μ sú bezpečné
(ii) ak F_i je tvaru $\mu[j] = a$ alebo $a = \mu[j]$, kde a je konštanta, tak $\mu[j]$ je bezpečné
(iii) ak F_i je tvaru $\mu[j] = \nu[k]$ alebo $\nu[k] = \mu[j]$, kde $\nu[k]$ je bezpečná, tak $\mu[j]$ je bezpečné

(4) operátor $\neg$ sa môže použiť len v rámci konjunkcie typu(3)

Predpokladá sa, že arita každej n-ticovej premennej je daná a nemení sa, aj keď sú dva výskyty viazané dvoma rôznymi kvantifikátormi.

3.2.3. Relačná algebra

- Relačná algebra zodpovedá základným operáciám s tabuľkami.
- **Základné operácie** relačnej algebry sú:
  a) **Zjednotenie:**

$$R \cup S = \{x_1, x_2, \ldots, x_n | R(x_1, x_2, \ldots, x_n) \vee S(x_1, x_2, \ldots, x_n)\}$$

  b) **Rozdiel:**

$$R \setminus S = \{x_1, x_2, \ldots, x_n | R(x_1, x_2, \ldots, x_n) \wedge \neg S(x_1, x_2, \ldots, x_n)\}$$

  c) **Kartézsky súčin:**

$$R \times S = \{x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_m) | R(x_1, x_2, \ldots, x_n) \wedge S(y_1, y_2, \ldots, y_m)\}$$

  d) **Projekcia:** Ak $\Omega = \{A_1, A_2, \ldots, A_n\}$ je množina všetkých atribútov relačnej schémy R a $X \subseteq \Omega$, $X = \{A_{i_1}, A_{i_2}, \ldots, A_{i_m}\}$, $i_1 < i_2 < \cdots < i_m$, tak

$$\pi_X R = \{x_1, x_2, \ldots, x_n | (\exists r_1)(\exists r_2) \ldots (\exists r_n)(R(r_1, r_2, \ldots, r_n) \wedge x_1 = r_{i_1} \wedge \cdots \wedge x_m = r_{i_m}\}$$

  e) **Selekcia:** Definujme najprv formuly jazyka L^R, kde R je relačná schéma s množinou atribútov $\Omega = \{A_1, A_2, \ldots, A_n\}$:
  (1) A_i pre $i \in \{1, \ldots, n\}$ je term; konštanta je term.
  (2) Ak $\vartheta \in \{<, =, >, \leq, \neq, \geq\}$ a t_1, t_2 sú termy, tak $t_1 \vartheta t_2$ je atomická formula.
  (3) Atomická formula je formula jazyka L^R.
  (4) Ak F_1, F_2 sú formuly jazyka L^R, tak aj $\neg F_1$, $(F_1 \wedge F_2)$, $(F_1 \vee F_2)$ sú formuly jazyka L^R.
  (5) Jazyk L^R neobashuje formuly iného typu.

  Potom ak $F \in L^R$ a F' vznikne z F nahradením x_i za A_i, tak

$$\sigma_F R = \{x_1, x_2, \ldots, x_n | R(x_1, x_2, \ldots, x_n) \wedge F'\}$$

- Zjednocovať a odčítavať možno len kompatibilné relácie.

Definícia 4(relačného výrazu jazyka $\mathcal{A}_R$): Nech $S = (R_1, R_2, \ldots, R_n, \mathbf{I})$ je relačná schéma s integritnými ohraničeniami **I**. Potom

(1) R_i pre $i \in \{1, \ldots, n\}$ je relačný výraz.
(2) Konštantná relácia je relačný výraz.
(3) Ak E_1, E_2 sú relačné výrazy, tak $E_1 \cup E_2$, $E_1 \times E_2$, $E_1 \setminus E_2$ sú relačné výrazy.
(4) Ak X je množina atribútov schémy zodpovedajúca relačnému výrazu E, tak $\pi_X E$ je relačný výraz.
(5) Ak F je formula jazyka L^S, kde S je schéma zodpovedajúca relačnému výrazu E, tak $\sigma_F E$ je relačný výraz
(6) Relačné výrazy iného typu neexistujú.

- $\mathcal{A}_R$ nezahrňuje ÁNO/NIE dotazy.
- Relačný výraz možno reprezentovať stromom. V prípade, že nebola použítá operácia rozdielu, hovoríme o výraze pozitívnej relačnej algebry.
- **Odvodené operácie** relačnej algebry:

a) **Prienik:** Ak R, S sú kompatibilné, tak

$$R \cap S = R \setminus (R \setminus S)$$

b) ***F* - spojenie:** Ak F je formula jazyka $L^{R\times S}$, tak

$$R \bowtie_F S = \sigma_F(R \times S)$$

c) **Prirodzené spojenie (join):** Nech $C = \Omega_R \cap \Omega_S$ a F je konjunkcia atomických formúl $R.A = S.A$ pre $\forall A \in C$. Potom

$$R \bowtie S = \pi_{(\Omega_R \cup \Omega_S)}(R \bowtie_F S)$$

Teda ak $|\Omega_R| = n$, $|\Omega_S| = m$, $|\Omega_R \cap \Omega_S| = k$, tak

$$R \bowtie S = \{t_1, t_2, \ldots, t_{n+m-k} | (\exists u_{k+1}) \ldots (\exists u_n)(\exists v_{k+1}) \ldots (\exists v_m)(R(t_1, \ldots, t_k, u_{k+1}, \ldots, u_n) \wedge$$

$$\wedge S(t_1, \ldots, t_k, v_{k+1}, \ldots, v_m) \wedge t_{k+1} = u_{k+1} \wedge \cdots \wedge t_n = u_n \wedge t_{n+1} = v_{k+1} \wedge \cdots \wedge t_{n+m-k} = v_m)\}$$

d) **Polospojenie (semijoin):**

$$R \ltimes S = \sigma_{\Omega_R}(R \bowtie S)$$

e) **Delenie:** Nech $\Omega_R = \Omega_S \cup X$. Potom $R \div S = \{x | (\forall y)(S(y) \implies R(x, y))\}$, t.j.

$$R \div S = (\pi_X R) \setminus \pi_X((\pi_X R \times S) \setminus R)$$

Veta 1(hlavná): Nasledujúce jazyky sú si ekvivalentné :
(i) Výrazy relačnej algebry
(ii) Bezpečné formuly doménového relačného kalkulu (safe DRC)
(iii) Bezpečné formuly n-ticového relačného kalkulu (safe TRC)
(iv) Bezpečné DATALOGové programy s negáciou

Dôkaz: $\triangleright$ viď napr. [1] kap. 3 $\triangleleft$

- Relačná algebra sa rozširuje o *agregačné* funkcie: **min**, **max**, **count**, **sum**, **prod**, ...

Príklad 2: Nech $\Omega_R = \{A\}$. Potom

$$max(R) = R \setminus (\pi_A \sigma_{a<b}(R(a) \times R(b)))$$

3.2.4. SQL

- Základnou štruktúrou je <select-clause>:

```
SELECT [DISTINCT] [*|Attr_1, ... ,Attr_n]
FROM <Tab_1> [<Id_1>], ... ,<Tab_m>[ <Id_m>]
[WHERE <Podm_1>]
[GROUP BY B_1, ... B_k]
[HAVING <Podm_2>]
[ORDER BY C_1[ASC|DESC]], ... C_l[ASC|DESC]
```

Jej význam je nasledovný: Z tabuliek uvedených za FROM sa urobí kartézsky súčin, so vzniknutých riadkov sa vyberú iba tie, ktoré spĺňajú podmienku za WHERE. Riadky sa formálne rozdelia do skupín s rovnakou hodnotou atribútov uvedených za GROUP BY a ponechajú sa iba tie skupiny, ktoré spĺňajú podmienku za HAVING. Výsledná postupnosť riadkov sa odprojektuje na atribúty uvedené za SELECT (v prípade * sa nič neudeje). Ak je uvedené DISTINCT, odstránia sa duplicitné riadky. Výsledok sa utriedi podľa atribútov uvedených za ORDER BY. Id_1, ... , Id_m sú n-ticové premenné tabuliek.

SELECT musí viesť k jedinému riadku pre každú skupinu špecifikovanú pomocou GROUP BY. INGRES vyžaduje, aby všetky atribúty za SELECT boli uvedené aj za GROUP BY, podmienka za HAVING sa môže týkať len atribútov uvedených za GROUP BY a agregačných funkcií MAX(<Attr>), MIN(<Attr>), SUM([DISTINCT] <Attr>), COUNT([DISTINCT]<Attr>), COUNT(*). Na vyjadrenie zjednotenia sa používa špecifikácia UNION [ALL]. ALL označuje zrušenie duplicitných riadkov. Špeciálne podmienky sú:

- $x\vartheta$ **any** $Y \equiv (\exists y \in Y)(x\vartheta y)$
- $x\vartheta$ **all** $Y \equiv (\forall y \in Y)(x\vartheta y)$, kde $\vartheta \in \{<,=,>,\leq,\neq,\geq\}$.

Niektoré implementácie obsahujú aj operáciu rozdielu DIFFERENCE a podmienku byť podmnožinou CONTAINS. Okrem relačných operácií $<,>,\ldots$ kombinovaných pomocou AND, OR a NOT, možno použiť aj test na byť (resp. nebyť) prvkom IN, resp. NOT IN, ktoré sú ekvivalentné s = **any**, resp. $\neq$ **all** a test na neprázdnosť množiny EXISTS, ktorý je ekvivalentný $0 \neq$ SELECT COUNT(*) ...

Veta 2: SQL je relačne úplný (t.j. všetky operácie $\mathcal{A}_R$ sú v ňom vyjadriteľné).
Dôkaz: $\triangleright$

$\pi_X R$ - SELECT DISTINCT X FROM R;
$\sigma_F R$ - SELECT * FROM R WHERE F;
$R \times S$ - SELECT * FROM R, S;
$R \cup S$ - SELECT * FROM R UNION ALL SELECT * FROM S;
$R \setminus S$ - SELECT * FROM R DIFFERENCE SELECT * FROM S; $\triangleleft$

Ďalšie príkazy:

- vloženie dát do tabuľky: INSERT INTO <meno> (<atribúty>) VALUES <hodnoty>
- alebo INSERT INTO <meno> <select-clause>
- mazanie dát z tabuľky: DELETE FROM <meno> WHERE <podmienka>
- úprava dát v tabuľke: UPDATE <meno-updatovanej> FROM <meno-pomocnej> WHERE <podmienka>

3.2.5. QBE - Query by example

Dotazovací jazyk QBE je založený na doménovom relačnom kalkule. Dotazy sa formulujú pomocou špeciálneho obrazovkového editora - vypĺňaním **skeletonov**.

QBE skeleton :

meno relácie	meno atribútu	meno atribútu	...
			...
príkazy	príklad požadovaných riadkov		

Do stĺpcov špecifikujúcich požadované riadky sa vpisujú konštanty (číselné resp. znakové reťazce, ktoré nezačínajú znakom _) a doménové premenné, ktorých identifikátory majú tvar _meno. Premenné označené prefixom P. sú vytlačené.

Implementácia dotazu :

Pre každý vyplnený riadok nejakého skeletonu sa vytvorí n-ticová premenná. Pre k takto vytvorených n-ticových premenných sa vykoná k vhniezdených cyklov, pričom každý z cyklov spôsobí, že jednej z n-ticových premenných sa priradia postupne všetky hodnoty jej relácie. Pre každé priradenie hodnôt n-ticovým premenným sa skontroluje, či možno priradiť doménovým premenným konzistentné hodnoty. V prípade, že je to možné, vykoná sa špecifikovaná operácia - napr. sa vytlačia doménové premenné s prefixom P. (viď príkazy).

Zápis ϑc, kde c je konštanta resp. ϑv, kde v je doménová premenná (napr.$>$ 3, $\leq$ *_meno*), značí, že vyhovujú všetky hodnoty domény vyhovujúce danej relácii. QBE ďalej poskytuje agregačné funkcie SUM., AVG., MAX., MIN., CNT.. Špeciálne operátory sú ALL. a UN.. ALL. aplikované na doménovú premennú vráti zoznam všetkých hodnôt, ktoré premenná nadobúda - vrátane viacnásobných (P.AVG.ALL._x vytlačí priemer všetkých hodnôt _x). UN. eliminuje viacnásobné hodnoty. Na zadávanie zložitejších podmienok sa používa tzv. **condition box**, pričom sa povoľujú operátory AND a OR, nie však NOT ($\neg$).

Príkazy - zadávajú sa do 1. stĺpca :

I. - vkladanie špecifikovaných riadkov do tabuľky
U. - úprava dát v tabuľke. QBE rozlišuje medzi kľúčovými a nekľúčovými atribútmi. Update prebieha tak, že riadku, ktorý súhlasí so zadaným riadkom na kľúčových atribútoch, sa nastavia hodnoty nekľúčových atribútov.
D. - mazanie špecifikovaných riadkov z tabuľky
P. - vytlačenie špecifikovaných riadkov.
$\neg$ - symbol negácie v príkazovom stĺpci vyžaduje, aby žiaden riadok vyhovujúci danej špecifikácii nebol riadkom tabuľky.

Príklad 3: Nasledujúci dotaz vytlačí názov najdrahšieho tovaru :

cenník	tovar	cena
	P.	_x
$\neg$		> _x

Veta 3: QBE je relačne úplný(t.j. všetky operácie $\mathcal{A}_R$ sú v ňom vyjadriteľné).
Dôkaz: $\triangleright$ prenechaný na čitateľa, prípadne [3] str. 206 $\triangleleft$

Príklad 4: Majme relácie Dodávka(tovar, cena, dátum, dod), Dodávateľ(dod, firma, adresa). Vyjadrime dotaz: Nájsť firmy (a ich adresy), ktoré dodali všetky dodávané tovary.

(DRC)

$$\{f,a|(\exists dod)Dodávateľ(dod,f,a) \wedge (\forall dod)(Dodávateľ(dod,f,a) \implies$$

$$\implies (\forall t)((\exists c')(\exists d')(\exists dod')Dodávka(t,c',d',dod') \implies (\exists c)(\exists d)Dodávka(t,c,d,dod)))\}$$

(TRC)

$$\{\mu^{(2)}|(\exists\nu^{(3)})(Dodávateľ(\nu) \wedge \nu[firma] = \mu[1] \wedge \nu[adresa] = \mu[2]) \wedge$$

$$\wedge(\forall\nu^{(3)})((Dodávateľ(\nu) \wedge \nu[firma] = \mu[1] \wedge \nu[adresa] = \mu[2]) \implies$$

$$\implies (\forall\alpha^{(4)})(Dodávka(\alpha) \implies (\exists\beta^{(4)})(Dodávka(\beta) \wedge \beta[tovar] = \alpha[tovar] \wedge \beta[dod] = \nu[dod])))\}$$

($\mathcal{A}_R$)

$$\pi_{f,a}(((\pi_{dod,t}Dodávka(t,c,d,dod)) \div (\pi_t Dodávka(t,c,d,dod))) \bowtie Dodávateľ(dod,f,a))$$

(SQL)

```
SELECT DISTINCT firma,adresa
FROM Dodávateľ WHERE dod IN
(SELECT dod
FROM Dodávka
GROUP BY dod
HAVING COUNT(DISTINCT tovar )=
(SELECT COUNT(DISTINCT tovar)
FROM Dodávka));
```

(QBE)

Dodávateľ	dod	firma	adresa
	_dod	P.	P.

Dodávka	tovar	cena	dátum	dod
	_x			_dod
	_y			

CONDITIONS
CNT.UN.ALL._x =CNT.UN.ALL._y

3.3. Jazyk pre riadenie dát

- Predstavuje prostriedky pre zadanie autorských práv, zmenu zobrazení medzi dátami, zmenu schém apod.
- V SQL sa na úpravu prístupových práv používa príkaz `GRANT` - osobitne na tabuľky, stĺpce tabuliek, pohľady:
  Získanie prístupových práv na reláciu R : `GRANT READ|WRITE|ALL ON` R `FROM GROUP|WORLD`
  Zrušenie prístupových práv na reláciu R : `REVOKE READ|WRITE|ALL ON` R `FROM GROUP|WORLD`

4. Teória navrhovania

4.1. Závislosti

Závislosti patria medzi základné druhy IO v relačných databázach. Základné druhy závislostí sú: *funkčné závislosti, multizávislosti, závislosti na spojení* a *inkluzívne závislosti.*

4.1.1. Funkčné závislosti

Definícia 5(funkčnej závislosti): Nech Ω je množina všetkých atribútov relačnej schémy R. Nech $X, Y \subseteq \Omega$ sú množiny atribútov. Budeme hovoriť, že X určuje Y (označenie $X \to Y$) $\iff$ keď pre ľubovoľné 2 riadky r_1, r_2 prípustnej relácie R^* platí, že ak sa riadky r_1, r_2 zhodujú vo všetkých atribútoch množiny X, tak sa zhodujú aj vo všetkých atribútoch množiny Y (t.j. $\pi_X r_1 = \pi_X r_2 \implies \pi_Y r_1 = \pi_Y r_2$).

Poznámka: Ak má relačná schéma R atribúty napr. $\Omega = \{A, B, C, D\}$, tak funkčnú závislosť $AB \to C$ možno popísať formulou:

$$(\forall a)(\forall b)(\forall c_1)(\forall c_2)(\forall d_1)(\forall d_2)(R(a,b,c_1,d_1) \wedge R(a,b,c_2,d_2)) \implies c_1 = c_2)$$

pričom $AB \to C$ je skrátený zápis pre $\{A, B\} \to \{C\}$.

Definícia 6(vyplývania): Nech $\mathcal{F}$ je množina závislostí. Nech Z je závislosť. Budeme hovoriť, že závislosť Z vyplýva z množiny $\mathcal{F}$ (označenie $\mathcal{F} \models Z$) $\iff$ pre každú relačnú schému $\mathbf{R}$ platí, že ak $\mathbf{R}$ spĺňa všetky závislosti v $\mathcal{F}$, tak $\mathbf{R}$ spĺňa aj Z.

Poznámka: Uvedená definícia je v platnosti pre ľubovoľné (aj zmiešané) typy závislostí, nielen pre funkčné závislosti.

Armstrongove axiómy funkčných závislostí:

1. Ak $Y \subseteq X \subseteq \Omega$, tak $X \to Y$
2. Ak $X \to Y$ a $Z \subseteq \Omega$, tak $XZ \to YZ$
3. $\{X \to Y, Y \to Z\} \vdash X \to Z$

Označenie: $\mathcal{F} \vdash Z$ označuje, že z množiny závislostí $\mathcal{F}$ možno pomocou axióm, resp. pravidiel odvodiť závislosť Z.

Veta 4: Armstrongove axiómy funkčných závislostí sú zdravé a úplné, t.j. ak $\mathcal{F}$ je množina funkčných závislostí a Z je funkčná závislosť, tak $\mathcal{F} \models Z \iff \mathcal{F} \vdash Z$.

Poznámka: Z Armstrongových axióm vyplýva, že $X \to Y \iff \forall A \in Y : X \to A$. Preto sa ďalej stačí zoberať len funkčnými závislosťami tohto typu.

Definícia 7(uzáveru): Nech X je množina atribútov a $\mathcal{F}$ množina funkčných závislostí. Uzáverom X vzhľadom k $\mathcal{F}$ nazývame množinu $X^+ = \{A \in \Omega | X \to A\}$.

Označenie: Ak $\mathcal{F}$ je množina závislostí, tak pod $\mathcal{F}^+$ rozumieme množinu všetkých takých závislostí Z, že $\mathcal{F} \vdash Z$.

Definícia 8(pokrytia): Nech $\mathcal{F}, \mathcal{G}$ sú množiny funkčných závislostí. Hovoríme, že $\mathcal{F}$ pokrýva $\mathcal{G} \iff \mathcal{G}^+ \subseteq \mathcal{F}^+$.
Problém príslušnosti: Pre danú množinu závislostí $\mathcal{F}$ a závislosť Z určiť, či $\mathcal{F} \vdash Z$ (t.j. či $Z \in \mathcal{F}^+$).
Problém uzáveru: Pre danú množinu závislostí $\mathcal{F}$ určiť $\mathcal{F}^+$.
Problém ekvivalencie: Pre dané množiny závislostí $\mathcal{F}, \mathcal{G}$ určiť, či $\mathcal{F}^+ = \mathcal{G}^+$.

Definícia 9(minimálneho pokrytia): Nech $\mathcal{F}$ je množina funkčných závislostí. Minimálnym pokrytím množiny $\mathcal{F}$ nazývame množinu Γ takú, že $\Gamma^+ = \mathcal{F}^+$ a

1. Všetky pravé strany závislostí v Γ sú jednoprvkové.
2. Pre žiadnu $X \to A \in \mathcal{F}$ nie je $\mathcal{F} \setminus \{X \to A\}$ ekvivalentné s $\mathcal{F}$, t.j. Γ je minimálna vzhľadom na inklúziu.
3. Pre žiadnu $X \to A$ a $Z \subsetneqq X$ nie je $(\mathcal{F} \setminus \{X \to A\}) \cup \{Z \to A\}$ ekvivalentné s $\mathcal{F}$, t.j. pravá strana žiadneho pravidla nie je určená žiadnou vlastnou podmnožinou ľavej strany (v kontexte $\mathcal{F}$).

Problém minimálneho pokrytia: Pre danú množinu závislostí $\mathcal{F}$ nájsť minimálne pokrytie.

Poznámka: Množina $\mathcal{F}^+$ môže mať exponenciálne veľa prvkov, preto neexistuje polynomiálny algoritmus riešiaci problém uzáveru: Nech $\mathcal{F} = \{A_i \to B_i | i = 1, \ldots, n\} \cup \{B_i \to A_i, i = 1, \ldots, n\} \cup \{A_1 A_2 \ldots A_n \to C\}$ Potom $D_1 D_2 \ldots D_n \to C \in \mathcal{F}^+$ pre $D_i \in \{A_i, B_i\}$. Takých pravidiel je 2^n. Dokonca ľavá strana každého je kľúčom, a teda existuje exponenciálne veľa kľúčov.

Definícia 10(nadkľúča): Ak $\mathbf{R}$ je relačná schéma a Ω množina jej atribútov, tak $S \subseteq \Omega$ je nadkľúčom $\mathbf{R} \iff S \to \Omega$.

Definícia 11(kľúča): Nech $\mathbf{R}$ je relačná schéma. $K \subseteq \Omega$ nazývame kľúčom $\mathbf{R} \iff K$ je nadkľúč, ale žiadna vlastná podmnožina K nadkľúčom nie je.

Tvrdenie 1: $X \to Y \iff Y \subseteq X^+$.

Algoritmus 2(výpočet uzáveru X^+ vzhľadom k množine funkčných závislostí $\mathcal{F}$):

`INPUT:` Ω, $\mathcal{F}$, $X \subseteq \Omega$.

`OUTPUT:` X^+ vzhľadom k $\mathcal{F}$.

1. $X^{(0)} := X$;
2. $X^{(i+1)} := \bigcup\{A \in \Omega | Y \to Z \in \mathcal{F} \wedge Y \subseteq X^{(i)} \wedge A \in Z\}$;
3. `RETURN` $X^{(i)}$ `WHEN NO MORE CHANGE OCCURS;`

Algoritmus 3(pre problém príslušnosti):

`INPUT:` $\Omega, \mathcal{F}, X \to Y$.

`OUTPUT:` Odpoveď na otázku, či $\mathcal{F} \vdash X \to Y$.

1. `Compute` X^+ `with respect to` $\mathcal{F}$ (algorithm 1);
2. `RETURN YES IFF` $Y \subseteq X^+$ `(ELSE RETURN NO);`

Algoritmus 4(test pokrytia):

`INPUT:` $\Omega, \mathcal{F}, \mathcal{G}$.

`OUTPUT:` Odpoveď na otázku, či $\mathcal{F}$ pokrýva $\mathcal{G}$.

1. `For each functional dependency` $X \to Y \in \mathcal{G}$ `test whether` $\mathcal{F} \vdash X \to Y$ `holds` (algorithm 2);
2. `RETURN YES IFF TRUE (ELSE RETURN NO);`

Algoritmus 5(nájdenie minimálneho pokrytia množiny funkčných závislostí):

`INPUT:` $\Omega, \mathcal{F}$.

`OUTPUT:` Γ - minimálne pokrytie $\mathcal{F}$.

1. `LET` $\Gamma := \{X \to A | A \in \Omega \wedge (\exists Y \subseteq \Omega)(A \in Y \wedge X \to Y \in \mathcal{F})\}$;
2. `Repeatedly search for violations of rules 2, 3 from definition of minimal cover in` Γ`. When such a violation is found, delete it from` Γ.
3. `RETURN` Γ;

4.1.2. Multizávislosti a vnorené multizávislosti

Definícia 12(multizávislosti): Nech Ω je množina všetkých atribútov relačnej schémy $\mathbf{R}$; $X, Y, Z \subseteq \Omega$. Hovoríme, že X multiurčuje Y (označenie $X \twoheadrightarrow Y$) $\iff$ pre ľubovoľné 2 riadky r_1, r_2 prípustnej relácie R^* také, že $r_1[X] = r_2[X]$ (majú rovnaké X-hodnoty) platí, že v nej existujú aj riadky s_1, s_2 také, že:

1. $s_1[X] = s_2[X] = r_1[X] = r_2[X]$
2. $s_1[Y] = r_1[Y] \wedge s_1[\Omega \setminus X \setminus Y] = r_2[\Omega \setminus X \setminus Y]$
3. $s_2[Y] = r_2[Y] \wedge s_2[\Omega \setminus X \setminus Y] = r_1[\Omega \setminus X \setminus Y]$.

Poznámka: Ak má relačná schéma R atribúty napr. $\Omega = \{A, B, C, D\}$, tak multizávislosť $A \twoheadrightarrow B$ možno popísať formulou:

$$(\forall a)(\forall b_1)(\forall b_2)(\forall c_1)(\forall c_2)(\forall d_1)(\forall d_2)((R(a, b_1, c_1, d_1) \wedge R(a, b_2, c_2, d_2)) \implies R(a, b_2, c_1, d_1))$$

pričom $A \twoheadrightarrow B$ je skrátený zápis pre $\{A\} \twoheadrightarrow \{B\}$.

Axiómy funkčných závislostí a multizávislostí:

1. Ak $Y \subseteq X \subseteq \Omega$, tak $X \to Y$.
2. Ak $X \to Y$ a $Z \subseteq \Omega$, tak $XZ \to YZ$.
3. $\{X \to Y, Y \to Z\} \vdash X \to Z$
4. $\{X \twoheadrightarrow Y\} \vdash X \twoheadrightarrow (\Omega \setminus X \setminus Y)$
5. Ak $X \twoheadrightarrow Y$ a $V \subseteq W \subseteq \Omega$, tak $WX \twoheadrightarrow VY$.
6. $\{X \twoheadrightarrow Y, Y \twoheadrightarrow Z\} \vdash X \twoheadrightarrow (Z \setminus Y)$.
7. $\{X \to Y\} \vdash X \twoheadrightarrow Y$.
8. Ak $X \twoheadrightarrow Y$ a $Z \subseteq Y$ a pre nejaké $W \subseteq \Omega, W \cap Y = \emptyset$ platí $W \to Z$, tak $X \to Z$.

Veta 5: Axiómy funkčných závislostí a multizávislostí sú zdravé a úplné, t.j. ak $\mathcal{M}$ je množina funkčných závislostí a multizávislostí a Z je funkčná závislosť alebo multizávislosť, tak $\mathcal{M} \models Z \iff \mathcal{M} \vdash Z$.

Definícia 13(bázy závislostí): Nech Ω je množina všetkých atribútov a $X \subseteq \Omega$. Potom bázou závislostí pre X (označenie X^+) rozumieme množinu takých množín atribútov $Y_1, Y_2, \ldots, Y_k \in \Omega \setminus X$, že ak $Z \subseteq \Omega \setminus X$, tak $X \twoheadrightarrow Z \iff Z$ je zjednotením niektorých Y_i.

Poznámka: $X^+ = \{Z \subseteq \Omega \setminus X | X \twoheadrightarrow Z \wedge$ pre žiadnu $W \subsetneqq Z$ neplatí, že $X \twoheadrightarrow W\}$.

Dôsledok: $X \twoheadrightarrow Y \iff (Y \setminus X)$ je zjednotením nejakých Y_i z X^+.

Algoritmus 6(výpočet bázy závislostí):

`INPUT:` $\mathcal{M}, \Omega, X \subseteq \Omega$.

`OUTPUT:` X^+ - báza závislostí pre X vzhľadom ku $\mathcal{M}$.

1. $S := \{\Omega \setminus Y\}$;
2. `REPEAT UNTIL NO MORE CHANGE CAN BE MADE:`

   2.1. `Look for dependency` $V \twoheadrightarrow W \in \mathcal{M}$ `and set` $Y \in S$ `such that` $Y \cap W \neq \emptyset$ `and` $Y \cap V = \emptyset$. `Replace Y by` $Y \cap W$ `and` $Y \setminus W$ `in` S;
3. `RETURN S;`

Definícia 14(vnorenej multizávislosti): Nech Ω je množina všetkých atribútov relačnej schémy $\mathbf{R}$; $X, Y \subseteq \Omega$. Používame označenie $X \twoheadrightarrow Y|Z \iff$ pre ľubovoľné 2 riadky r_1, r_2 prípustnej relácie R^* také, že $r_1[X] = r_2[X]$ (majú rovnaké X-hodnoty) platí, že v nej existujú aj riadky s_1, s_2 také, že:

1. $s_1[X] = s_2[X] = r_1[X] = r_2[X]$
2. $s_1[Y] = r_1[Y] \wedge s_1[Z] = r_2[Z]$
3. $s_2[Y] = r_2[Y] \wedge s_2[Z] = r_1[Z]$.

Poznámka: Multizávislosť $X \twoheadrightarrow Y$ je teda špeciálnym prípadom vnorenej multizávislosti $X \twoheadrightarrow Y|(\Omega \setminus X \setminus Y)$. Vnorené multizávislosti majú význam pri dekompozícii tabuliek, pretože multizávilslosť $X \twoheadrightarrow Y|Z$ je multizávislosťou v tabuľke $\pi_{X \cup Y \cup Z} R$.

Poznámka: Ak má relačná schéma R atribúty napr. $\Omega = \{A, B, C, D\}$, tak vnorenú multizávislosť $A \twoheadrightarrow B|C$ možno popísať formulou:

$$(\forall a)(\forall b_1)(\forall b_2)(\forall c_1)(\forall c_2)(\forall d_1)(\forall d_2)((R(a, b_1, c_1, d_1) \wedge R(a, b_2, c_2, d_2)) \implies (\exists d_3) R(a, b_2, c_1, d_3))$$

pričom $A \twoheadrightarrow B|C$ je skrátený zápis pre $\{A\} \twoheadrightarrow \{B\}|\{C\}$.

Axiómy aritmetiky usporiadaných množín

1. $x \leq x$

2. $x < y \implies x \leq y$
3. $x < y \implies x \neq y$
4. $x \leq y \wedge x \neq y \implies x < y$
5. $x \neq y \implies y \neq x$
6. $x < y \wedge y < z \implies x < z$
7. $x \leq y \wedge y \leq z \implies x \leq z$
8. $x \leq z \wedge z \leq y \wedge x \leq w \wedge w \leq y \wedge z \neq y \implies x \neq y$

Veta 6: Axiómy aritmetiky usporiadaných množín sú korektné a úplné.

4.1.3. Dekompozícia relácie

- Relačnú schému $\mathbf{R}$ rozložíme na niekoľko schém $\mathbf{R}_1, \ldots, \mathbf{R}_n$, pričom tieto novovzniknuté schémy majú nejaké (potenciálne žiadne) atribúty spoločné. Nech Ω je množina všetkých atribútov. Nech $\rho = \{X_1, \ldots, X_n\}$ je množina množín atribútov $X_i \subseteq \Omega$ (vytvárame schémy $\pi_{X_i}\mathbf{R}$). Spojenie $m_\rho = \pi_{X_1}\mathbf{R} \bowtie \cdots \bowtie \pi_{X_n}\mathbf{R}$ sa nazýva **join projecting mapping**. Zrejme $\mathbf{R} \subseteq m_\rho$.

Definícia 15(bezstratovej dekompozície): Hovoríme, že dekompozícia je bezstratová $\iff m_\rho = \mathbf{R}$.

Poznámka: Ak má relačná schéma R atribúty napr. $\Omega = \{A, B, C\}$ a $R_1 = \sigma_{\{A,B\}}R$, $R_2 = \sigma_{\{B,C\}}R$, tak skutočnosť, že ide o bezstratovú dekompozíciu možno vyjadriť formulou:

$$(\forall a)(\forall b)(\forall c)((R_1(a,b) \wedge R_2(b,c)) \implies R(a,b,c))$$

Túto vlastnosť nazývame aj závisloť na spojení a označujeme aj $\bowtie (AB, BC)$.

Algoritmus 7(test, ci je dekompozícia bezstratová):

`INPUT:` $R = A_1, \ldots, A_n, \mathcal{F}, R_i, i = 1, \ldots, k$

`OUTPUT:` odpoveď na otázku, či je dekompozícia bezstratová

1. We construct a table with n columns and k rows,column j corresponds to A_j and row i corresponds to R_i. In row i, column j put the symbol a_j if A_j is in R_i. If not, put the symbol b_ij there.
2. Repeatedly consider the dependencies $X \to Y \in \mathcal{F}$, until no more changes can be made.Each time we consider $X \to Y$,we look for the rows that agree in all of the columns for the atributes of X.If we find two such rows,equate the symbols of those rows for the atributes of Y. When we equate two symbols, if one of them is a_j, make the other be a_j. If they are b_ij and b_lj, make them both b_ij or both b_lj. It is important to understand that when two symbols are equated,<u>all</u> occurences of those symbols in the table become the same.
3. If after step 2. some row become $a_1, \ldots, a_n$, then the join is lossless. If not, join is lossy(not lossless).

Dôkaz: $\triangleright$ viď napr. [1] str.394 $\triangleleft$

Definícia 16(dekompozície zachovávajúcej závislosti): Nech $\mathbf{R}$ je relačná schéma a $\rho = \{X_1, \ldots, X_n\}$ množina množín jej atribútov a $\mathcal{F}$ množina jej funkčných závislostí. Nech $\pi_{R_i}(\mathcal{F}) = \{X \to Y | X \to Y \in \mathcal{F}^+ \wedge X, Y \in X_i\}$. Hovoríme, že dekompozícia zachováva závislosti $\iff (\cup_{i=1}^n \pi_{R_i}(\mathcal{F}))^+ = \mathcal{F}^+$.

Algoritmus 8(test, či dekompozícia zachováva závislosti):

`INPUT:` $R, \mathcal{F}, R_i, i = 1, \ldots, k$

`OUTPUT:` odpoveď na otázku, či dekompozícia zachováva závislosti

```
For each X → Y ∈ F do
   Z:=X
   while changes to Z occur do
      for i:=1 to k do
         Z:=Z∪((Z∩Ri)+ ∩ Ri)
If Y ⊆ Z holds for every X → Y ∈ F, return YES (else return NO).
```

Veta 7: Nech $\mathbf{R}$ je relačná schéma. Nech $X_1 \cup X_2 = \Omega$. Nech $\mathbf{R}_1 = \pi_{X_1}\mathbf{R}$ a $\mathbf{R}_2 = \pi_{X_2}\mathbf{R}$. Potom $\mathbf{R}_1$, $\mathbf{R}_2$ je bezstratová dekompozícia $\iff \pi_{X_1 \cap X_2}\mathbf{R} \twoheadrightarrow \pi_{X_1 \setminus X_2}\mathbf{R}$.
Dôkaz: $\triangleright$ Nech $X = X_1 \setminus X_2, Y = X_1 \cap X_2, Z = X_2 \setminus X_1$.
($\Longrightarrow$): Nech platí $R(x_1, y, z_1)$ a $R(x_2, y, z_2)$. Potom platí $R_1(x_1, y)$ a $R_2(y, z_2)$. Keďže spojenie je bezstratové, tak platí $R(x_1, y, z_2)$. Preto $\pi_{X_1 \cap X_2}\mathbf{R} \twoheadrightarrow \pi_{X_1 \setminus X_2}\mathbf{R}$.
($\Longleftarrow$): Nech platí $R_1(x, y)$ a $R_2(y, z)$. Potom $(\exists z')R(x, y, z')$ a $(\exists x')R(x', y, z)$. Pretože $\pi_{X_1 \cap X_2}\mathbf{R} \twoheadrightarrow \pi_{X_1 \setminus X_2}\mathbf{R}$, tak platí $R(x, y, z)$. $\triangleleft$

4.2. Normálne formy

4.2.1. Prvá a druhá normálna forma

- Hlavným dôvodom normalizácie dát je eliminácia redudantnej informácie. Realizácia dotazu môže mať za následok zložité spájanie tabuliek, preto v istých špecifických prípadoch je možné nechať tabuľky nenormalizované.

Definícia 17(prvej normálnej formy): Relačná schéma je v prvej normálnej forme $\iff$ hodnoty atribútov sú jednoduché, t.j. ďalej nedeliteľné.

- Tabuľka je automaticky v 1.NF.

Definícia 18(primárneho atribútu): Atribút sa nazýva primárny $\iff$ je súčasťou nejakého kľúča. V opačnom prípade sa nazýva nekľúčový.

Definícia 19(druhej normálnej formy): Relačná schéma je v druhej normálnej forme $\iff$ žiaden nekľúčový atribút nie je netriviálne funkčne závislý na vlastnej časti žiadneho kľúča.

4.2.2. Tretia normálna forma

Definícia 20(tretej normálnej formy): Relačná schéma je v tretej normálnej forme $\iff$ ak pre každú funkčnú závislosť $X \to A \in \mathcal{F}^+$, kde $A \in \Omega$ platí aspoň jedno z nasledujúcich:
- Závislosť je triviálna, t.j. $A \in X$.
- A je primárny.
- X je nadkľúč.

Veta 8: Tabuľka v 3.NF je aj v 2.NF.
Dôkaz: $\triangleright$ Nech $X \to A$ je netriviálna závislosť a A je nekľúčový. Potom podľa definície 3.NF platí, že X je nadkľúč. Žiaden nadkľúč však nemôže byť vlastnou časťou žiadneho kľúča. Preto X nie je vlastnou časťou žiadneho kľúča a tabuľka je v 2.NF. $\triangleleft$

- Tabuľky sa normalizujú minimálne do 3.NF. Najlepšie by bolo dosiahnuť bezstratovú dekompozíciu, ktorá zachováva závislosti. To je však možné len do určitej úrovne normalizácie.

Algoritmus 9(dekompozície zachovávajúcej závislosti do 3.NF):

INPUT: $R, \mathcal{F}$

OUTPUT: $R_1, \ldots, R_n$ také, že zachovávajú závislosti

1. Compute minimal cover of $\mathcal{F}$.
2. If there are any attributes of R not involved in any dependency, either on the left or right, eliminate them to form the relation scheme by itself.
3. If one of the dependencies in $\mathcal{F}$ involves all the attributes of R then output R itself.
4. Otherwise, the decomposition to be output consists of scheme XA for each dependency $X \to A$ in $\mathcal{F}$.

4.2.3. Boyce-Coddova normálna forma

Definícia 21(Boyce-Coddovej normálnej formy): Relačná schéma **R** je v Boyce-Coddovej normálnej forme $\iff$ pre každú netriviálnu závislosť $X \to A$ platí, že X je nadkľúč.

Poznámka: Relačná schéma v Boyce-Coddovej normálnej forme je automaticky aj v 3.NF.
Algoritmus 10(bezstratovej dekompozície do BCNF):

INPUT: $R, \mathcal{F}$

OUTPUT: $R_1, \ldots, R_n$ také, že $R_1 \bowtie \cdots \bowtie R_n = R$ a R_i je v BCNF.

```
Main program
   Z:=R;
   repeat
      decompose Z into Z\A and XA, where XA is in BCNF and X → A ∈ F.
      add XA to the decomposition;
      until Z cannot be decomposed;
   add Z to the decomposition;

procedure DECOMPOSITION; /* see main program :  2.1, 2.2 */
   if Z contains no A and B such that A is in (Z \ AB)+ then
return that Z is in BCNF and cannot be decomposed;
   else
begin
      find one such A and B;
      Y:=Z\B;
      while Y contains A and B such that (Z \ AB)+ → A do
         Y:=Y\B;
      return the decomposition Z - A and Y;
/* Y here is XA in the main program */
end
```

- Korektnosť algoritmu je založená na nasledovnej vete:

Veta 9: Každá binárna relačná schéma je v BCNF. Ak R nie je v BCNF, tak v R existujú také atribúty A≠B, že $(R \setminus AB) \to A$.
Dôkaz: $\triangleright$ Prvá časť sa priamo preverí. V druhej časti, keďže R nie je v BCNF, tak exisuje taká netriviálna (t.j. $A \notin X$) závisloť $X \to A \in \mathcal{F}^+$, že X nie je nadkľúč, t.j. $\exists B \in \Omega : X \not\to B$. Potom $B \notin X \wedge A \neq B$. Teda $X \subseteq R \setminus \{A, B\}$ a teda $R \setminus AB \to A$. $\triangleleft$

- Otázka je, ako dosiahnuť, že dekompozícia bude aj bezstratová aj zachovávať závislosti so súčasným zachovaním normalizácie dát. Žiaľ:

Veta 10: Existuje relačná schéma, pre ktorú neexistuje dekompozícia do BCNF zachovávajúca závislosti.
Dôkaz: $\triangleright$ Stačí prešetriť všetky dekompozície schémy $R(ps\check{c}, mesto, adresa)$ s funkčnými závislosťami $ps\check{c} \to mesto$ a $mesto, adresa \to ps\check{c}$. $\triangleleft$

Určiť, či pre danú relačnú schému existuje dekompozícia do BCNF, je NP-ťažký problém.

Algoritmus 11(bezstratovej dekompozície do 3.NF zachovávajúcej závislosti):

INPUT: $R, \mathcal{F}$

OUTPUT: $R_1, \ldots, R_n$

```
1.  Compute decomposition into 3.NF preserving dependencies (alg.  6).
2.  Let X is any key of R. Add X to decomposition.
```

4.2.4. Štvrtá normálna forma

Definícia 22 (štvrtej normálnej formy) : Relačná schéma je v 4.NF $\iff$ pre každú netriviálnu

multizávislosť $X \twoheadrightarrow Y$ platí, že X je nadkľúč. (Netriviálnu v tomto prípade znamená $Y \not\subseteq X \wedge \Omega \not\subseteq XY$.)
<u>**Poznámka:**</u> Algoritmus bezstratovej dekompozície do 4.NF je podobný ako pre BCNF.

4.3. Ďalšie závislosti

- Ďalším pomerne rozšíreným typom závislostí je závislosť inkluzívna. Ide o vyjadrenie IO, že istý atribút môže nadobudnúť len hodnoty obsiahnuté v nejakom inom. Praktický význam je vo vyjadrení obmedzenia cudzieho kľúča.
- Na základe logických formúl, ktoré popisujú tú ktorú závislosť možno zaviesť pojem zovšeobecnenej závislosti zadanej takouto formulou. Riešiť problém príslušnosti potom možno špeciálnou metódou (tablá - chase procedure).

4.4. Návrh tabuliek

- V praxi sa stretávame skôr s kompozíciou ako dekompozíciou tabuliek. Kompozíciu možno nahradiť dekompozíciou tzv. univerzálnej relácie, čo je fiktívna relácia obsahujúca všetky atribúty. Na to potrebujeme 2 predpoklady:
- <u>*Predpoklad univerzálnej relácie:*</u> Univerzálna relácia existuje.
- <u>*Predpoklad jednoznačnosti vzťahov:*</u> Pod závislosťou $X \to Y$ máme pri každej zmienke na mysli tú istú funkčnú závislosť.

4.5. Prevod konceptuálneho E-R modelu na relačný

1. Pre každú entitnú množinu E definujeme schému relácie $E(A_1, \ldots, A_n)$, kde A_i sú mená atribútov definovaných na E.
2. Pre každú vzťahovú množinu R definovanú pomocou entitných množín $E_1, \ldots, E_k$ s atribútmi $B_1, \ldots, B_m$, kde $k \geq 2, m \geq 0$, definujem schému relácie $R(A_1, \ldots, A_k, B_1 \ldots, B_m)$, kde A_i je primárny kľúč entitnej množiny E_i.
3. Je nutné špecifikovať obmedzenia cudzieho kľúča, pretože hodnotou atribútu vzťahovej relácie môže byť len údaj naozaj obsiahnutý v entitnej relácii (iný stav je nekonzistentný).

5. Transakčné spracovanie

5.1. Journal

- Vykonávané operácie sa zoskupujú do pre užívateľa atomických <u>*transakcií*</u>.
- Toto zoskupovanie má niekoľko dôvodov:
    1. Vo vnútri transakcie môže dôjsť k porušeniu IO. Pre užívateľa je potom transakcia atomická operácia vedúca od jedného konzistentného stavu k druhému.
    2. Ak dôjde k poruche (výpadok zariadenia, prúdu, ...), musí sa databáza uviesť do konzistentného stavu, t.j. niektoré transakcie treba uviesť do pôvodného stavu. Z tohto dôvodu sa vedie prehľad o vykonávaných transakciách -<u>*journal*</u>.
- Je možné vykonať 2 špeciálne operácie, o ktorých sa predpokladá, že sú atomické.

1. COMMIT - úspešné ukončenie transakcie.
2. ROLLBACK - vynútenie uvedenia transakcie do pôvodného stavu. Túto operáciu môže vyvolať aj užívateľská aplikácia v prípade, že nie je možné vo vykonávaní transakcie ďalej pokračovať.

Systém rozoznáva *kontrolné body* - **checkpoints**. V týchto bodoch systém koordinuje obsah vyrovnávacích pamätí s obsahom disku. Spôsob zotavenia z chyby je závislý od spôsobu záznamu zmien na disk:

1. **Inkrementálna tvorba žurnálu s odloženými realizáciami zmien:**
   Behom prevádzania transakcie sa všetky operácie zmien WRITE odložia až do okamihu, keď transakcia čaká na potvrdenie ukončenia COMMIT. Žurnál potom behom vykonávania transakcie T obsahuje trojice <T, meno atribútu, nová hodnota>. Na začiatku sa zapíše <T, START> a na konci <T, COMMIT>. V prípade chyby pred dosiahnutím <T, COMMIT> by bolo treba uviesť transakciu do pôvodného stavu. Keďže žiadne zmeny na disku neboli realizované, možno proste údaje zo žurnálu zabudnúť - žiadaný stav je dosiahnutý. V prípade chyby po dosiahnutí <T, COMMIT> ale pred dosiahnutím kontrolného bodu je nutné transakciu znovu realizovať (REDO(T)), na čo slúžia údaje v žurnáli.
2. **Inkrementálna tvorba žurnálu s bezprostrednou realizáciou zmien:**
   Behom prevádzania transakcie sa všetky opeácie zmien WRITE realizujú priamo do vyrovnávacej pamäte. Stránky vyrovnávacej pamäte sa zapisujú na disk v kontrolných bodoch (operácia OUTPUT). V žurnále sa okrem START a COMMIT záznamu držia štvorice <T, meno atribútu, stará hodnota, nová hodnota>. V prípade poruchy systému sa databáza uvedie do konzistentného stavu nasledovne:

   Žurnál sa prehliada od posledného kontrolného bodu smerom dopredu. Všetky transakcie spracovávané v poslednom kontrolnom bode sa presunú do zoznamu UNDO. Ak sa pri prehliadaní narazí na začiatok ďalšej transakcie, zaradí sa do zoznamu UNDO. Ak sa narazí na príkaz COMMIT pre transakciu T, presunie sa T do zoznamu REDO. Systém spätným chodom za pomoci údajov v žurnále uvedie transakcie v zozname UNDO do pôvodného stavu a transakcie v zozname REDO vykoná znova.

Pri multitaskingových systémoch môže mať vyvolanie operácie ROLLBACK (t.j. započatie UNDO) za následok vyvolanie ROLLBACK aj pre ďalšie transakcie (ktoré použili výsledky pôvodnej) - tzv. *dominový efekt*.

Periodicky sa údaje z databázy zálohujú na **BACK-UP**. V prípade úplného zrútenia systému sa dá obnoviť stav z backupu a realizovať transakcie znova za pomoci žurnálu.

5.2. Paralelné spracovanie

Ako už bolo povedané, vykonávané operácie sa zoskupujú do pre užívateľa atomických *transakcií*. V multitaskingovom systéme sa vykonáva niekoľko transakcií "súčasne". Treba si uvedomiť, že získaný výsledok môže závisieť od poradia vykonávania transakcií, ak spolu súvisia. Dôležitým pojmom je **sériovateľnosť** transakcií - bolo by vhodné, aby sa konečnému užívateľovi aj paralelne bežiace transakcie javili tak, ako keby išli za sebou (v nejakom poradí). *Rozvrhom* nazývame určenie poradia vykonávania elementárnych (atomických) krokov transakcií. Rozvrh je sériovateľný $\Longleftrightarrow$ k nemu existuje ekvivalentný sériový rozvrh, t.j. transakcie možno usporiadať a vykonať postupne za sebou s rovnakým výsledkom, aký dával pôvodný rozvrh bez ohľadu na to, aké konkrétne data tabuľky obsahujú. Rozhodnúť vo všeobecnosti o sériovateľnosti daného rozvrhu môže byť dosť náročné - napríklad pri určovaní rovnosti výsldných hodnôt položiek sa môžu brať do úvahy aj algebraické zákony ($(X+10)*3=(X*3)+30$). Overovanie sériovateľnosti by sa ale potom stalo časovo veľmi zložité. Pesimistický prístup - nestane sa nič, ak sa zakáže nejaké množstvo sériovateľných rozvrhov. Jediný nesériovateľný rozvrh bz však spôsobil fatálnu chybu. Spôsobom, ako dosiahnuť sériovateľnosť transakcií, sú **uzamykacie protokoly**. Podobne ako v operačných systémoch si transakcia údaje, s ktorými bude pracovať, **uzamkne**. Zamykať možno súbory, vety, záznamy, ale obyčajne sa zamykajú stránky (čím menšia časť z hľadiska štruktúry údajov sa dá zamknúť, tým je väčšia priepustnsť systému, ale zároveň narastajú nároky na správu zámkov). Pod protokolom rozumieme "pravidlá slušného správania sa transakcií". Transakcia zamkne stránku atomickou operáciou LOCK a odomkne ju atomickou operáciou UNLOCK. Predpokladá sa, že

(i) Transakcia nezamyká objekt, ak je už touto transakciou uzamknutý.

(ii) Transakcia nikdy neodomyká objekt, ktorý nie je touto transakciou uzamknutý.
(iii) Po ukončení transakcie všetky objekty, ktoré boli uzamknuté, sú odomknuté.

V závislosti od použitého protokolu sa môže stať, že nastane LIVELOCK alebo DEADLOCK.

Definícia 23(LIVELOCK): LIVELOCK-om nazývame stav, keď nejaká transakcia T čaká na prístup k nejakej položke A, ktorá je postupne zamykaná (a odomykaná) transakciami $T_1 \dots T_n$, takže T čaká.

Definícia 24(DEADLOCK): Majme napríklad transakcie T_1 a T_2:

T_1: LOCK A; LOCK B; ... ;UNLOCK B; UNLOCK A
T_2: LOCK B; LOCK A; ... ;UNLOCK A; UNLOCK B

Deadlockom nazývame stav, keď každá transakcia z množinz transakcií S čaká na položku predtým zamknutú nejakou inou transakciou z S.

Žurnál musí obsahovať aj údaje o uzamykaní stránok, aby bolo podľa potreby možné overiť sériovateľnosť a detekovať DEADLOCK. Dosiahnuť sériovateľnosť možno nasledovne:

1. Optimistické riadenie - transakcie sa vykonávajú. Po vykonaní COMMIT sa overí, či rozvrh je sériovateľný. Ak nie, vyvolá sa ROLLBACK a transakcie sa spustia znovu. Je nutné detekovať DEADLOCK.
2. SCHEDULER - zvláštny proces, ktorý transakcie plánuje tak, aby boli sériovateľné.
3. Použitie protokolu, ktorý zaručuje, že rozvrh je sériovateľný. Ak protokol umožňuje DEADLOCK, treba ho detekovať.

Spôsoby 2 a 3 predstavujú príklad pesimistického prístupu.

Algoritmus 12(test sériovateľnosti):

`INPUT:` Rozvrh transakcií

`OUTPUT:` Odpoveď na otázku, či je sériovateľný.

Skúmanému rozvrhu priradíme graf, ktorého vrcholy sú transakcie. Ak transakcia B uzamkne stránku X hneď potom, čo ju transakcia A odomkne, pridáme hranu $A \to B$ s významom B musí nasledovať až po A. Ak je získaný graf cyklický, rozvrh nie je sériovateľný. Ak je acyklický, možno topologickým utriedením získať jeden ekvivalentný sériový rozvrh.

5.2.1. Dvojfázový protokol

Jedným z protokolov zabezpečujúcich sériovateľnosť (avšak potenciálne umožňujúcich DEADLOCK) je dvojfázový protokol prebiehajúci v 2 fázach:

1. Transakcia uzamyká potrebné stránky.
2. Od prvého odomknutia už stránky iba odomyká (operácia LOCK už nie je možná).

Keby v grafe transakcií pri dvojfázovom protokole vznikol cyklus $X_1, \dots, X_n, X_1$ existovala by transakcia X_1, ktorá zamkla stránku po transakcii X_n, ktorá zamkla nejakú stránku až potom, čo X_1 nejakú odomkla, čo je spor s dvojfázovým protokolom pre transakciu X_1.

Protokoly môžeme rozdeliť nasledovne :

a, ***Agresívne***, t.j. také, ktoré chcú napredovať čo najrýchlejšie, bez ohľadu na to, že je tu možnosť dostať sa do situácie vyžadujúcej ABORT. Transakcia žiada o zámok tesne predtým, ako potrebuje čítať z resp. písať do položky. V prípade READ a WRITE zámkov (viď nasldujúcu kapitolu) môže v prípade potreby urobiť upgrade RLOCK-u na WLOCK.Tento postup môže viesť k DEADLOCK-om.

b, ***Konzervatívne***, t.j. protokol neumožní štart transakcie, ak si nie je istý, že sa transakcia nedokončí. Konzrevatívne protokoly sú vo všeobecnosti pomalšie ako agresívne, ale bezpečnejšie.

Konzervatívny dvojfázový protokol :

1. Každá transakcia žiada všetky zámky, ktoré bude (potenciálne) potrebovať, na začiatku. Ak sú všetky zámky dostupné, SCHEDULER jej ich poskytne. Ak je čo len jeden zo zámkov nedostupný, transakcia je zaradená do fronty a čaká. Tento postup vylučuje DEADLOCK.
2. Transakcie sú ukladané do fronty v takom poradí, v akom prišli, žiadna transakcia sa nezačne vykonávať, pokiaľ sa nevykonajú transakcie vo fronte pred ňou. Tento postup vylučuje LIVELOCK.

5.2.2. Model s READ a WRITE zámkami

Zámky možno špecifikovať do viacerých úrovní :

1. READ-LOCK (zdieľaný zámok na čítanie)
   Transakcia T len čítajúca dáta zamkne položku A RLOCK-om, čo zabráni iným transakciám zapisovať do A, ale ľubovoľný iný počet transakcií môže zamknúť A R-LOCK-om.
2. WRITE-LOCK (exkluzívny zámok na zápis)
   Pokiaľ transakcia T zamkla A WLOCK-om, žiadna iná transakcia nemôže čítať ani písať do A.

Definícia 25 (ekvivalencie rozvrhov s READ a WRITE zámkami) :
Dva rozvrhy nazývame ekvivalentné $\Longleftrightarrow$

1. Výsledná hodnota každej položky je rovnaká v oboch rozvrhoch.
2. Každý R-LOCK požadovaný nejakou transakciou sa vykoná v oboch rozvrhoch v časoch, keď má zamykaná položka rovnakú hodnotu.

Algoritmus 13(test sériovateľnosti pre rozvrhy s READ- a WRITE- zámkami):

`INPUT:` Rozvrh transakcií

`OUTPUT:` Odpoveď na otázku, či je sériovateľný.

Skúmanému rozvrhu priradíme graf, ktorého vrcholy sú transakcie.Hrany sú určené nasledujúcimi pravidlami :
1. Nech transakcia A uzamkne stránku X RLOCK-om alebo WLOCK-om, transakcia B je nasledujúca transakcia taká, že uzamkne X WLOCK-om. Potom pridáme hranu $A \rightarrow B$ s významom B musí nasledovať až po A.
2. Nech transakcia A uzamkne stránku X WLOCK-om. Nech $B \neq A$ je ľubovoľná iná transakcia zamykajúca X RLOCK-om medzitým, ako A odomkne X a nejaká iná transakcia zamkne X WLOCK-om. Potom pridáme hranu $A \rightarrow B$.
Ak je získaný graf cyklický, rozvrh nie je sériovateľný. Ak je acyklický, možno topologickým utriedením získať jeden ekvivalentný sériový rozvrh.

Ako v modeli s jednoduchými zámkami, aj tu môžeme zadefinovať **dvojfázový protokol** zabezpečujúci sériovateľnosť, kde operácie RLOCK a WLOCK budú predchádzať operáciu UNLOCK.

5.2.3. Všeobecné modely zámkov

V prípade, že používame viac ako jeden typ zámkov (napr. READ a WRITE LOCK), môžeme pravidlá pre ne zhrnúť pomocou **matice kompatibility zámkov**. Riadky zodpovedajú módu požadovaného zámku, stĺpce zodpovedajú typu zámku, ktorým je položka už zamknutá. Prvkami matice sú Y (požadovaný zámok môže byť pridelený) a N (požadovaný zámok nemôže byť pridelený).

Príklad 5: Majme okrem READ a WRITE LOCK-ov aj INCR-LOCK, ktorý umožňuje držiteľovi pričítať 1 atomicky, bez prečítania položky, takže poradie, v ktorom sa inkrementácie uskutočnia, neovplyvní výslednú hodnotu. Čítanie položky a písanie do nej ale s inkrementáciou nekomutujú. Takže matica kompatibility pre READ, WRITE a INCR LOCK-y vyzerá nasledovne :

		Zámky držané inými transakciami		
		Read	Write	Incr
	Read	Y	N	N
Zámky	Write	N	N	N
požadované	Incr	N	N	Y

Algoritmus testu sériovateľnosti možno upraviť pre všeobecné modely zámkov. Konštruujeme graf sériovateľnosti tak, že pridávame hranu $A \rightarrow B$, kedykoľvek zámok položky X typu P transakciou A predchádza v rozvrhu zámku položky X typu Q transakciou B a v tabuľke kompatibility zámkov v riadku Q a stĺpci P je N. Tak, ako v predchádzajúcich prípadoch, rozvrh je sériovateľný, ak je získaný graf acyklický.

5.2.4. Kontrola paralelného spracovania pomocou časových známok

Časové známky tvoria popri zamykacích protokoloch druhú možnosť kontroly sériovateľnosti rozvrhu. Žiaden z týchto prístupov nie je dominantný, existujú rozvrhy sériovateľné zamykacím protokolom a nesériovateľné známkami a naopak (viď napr. [1] str.524). Vo všeobecnosti patrí spôsob kontroly časovými známkami k optimistickým - transakcie sa vzkonávajú, keď sa narazí na transakciu narušujúcu sériovateľnosť, musí dôjsť k Abort-u.

Kontrola sériovateľnosti pomocou časových známok :

Uvažujme Read/Write schému (podobne ako pri zamykacích protokoloch možno zaviesť systém známok aj pre viac resp. menej rôznych typov prístupu k dátam). Každá transakcia dostane od SCHEDULER-a časovú známku. V prípade, že existuje viac procesov, ktoré prideľujú známky (viacprocesorový počítač, distribuovaná databáza), treba zabezpečiť ošetrenie možných kolízií.
Každá položka databázy má priradené dva časy :

1. Read-time - čas najvyššej známky transakcie, ktorá položku čítala
2. Write-time - čas najvyššej známky transakcie, ktorá do položky zapisovala

Majme transakciu T s časovou známkou t, ktorá sa pokúša previesť operáciu X na položke s Read-time t_r a s Write-time t_w:

a, Vykonaj operáciu, ak $X = READ$ a $t \geq t_w$, alebo $X = WRITE$, $t \geq t_r$, $t \geq t_r$. V prvom prípade nastav Read-time na t, ak $t > t_r$, v druhom nastav Write-time na t, ak $t > t_w$.
b, Ak $X = WRITE$ a $t_r \leq t < t_w$, nerob nič.
c, Ak $X = READ$ a $t < t_w$, alebo $X = WRITE$ a $t < t_r$, Abort!!!

6. Distribuované databázové systémy

Distribuované DBS sú tvorené uzlami a linkami medzi nimi. Uzly buď obashujú disjunktnú informáciu (čo je nebezpečné v prípade výpadku uzla), alebo je tá istá informácia aj vo viacerých uzloch (čo môže umožňovať čítanie špinavých dát). Praktická realizácia:

1. Jeden koordinačný uzol, čo je nebezpečné pre prípad jeho výpadku.
2. Každý uzol má úplnú informáciu o sieti. V tomto prípade je ťažké zabezpečiť konzistenciu dát.
3. Každý uzol spravuje lokálne dáta a má informáciu o svojich susedoch.

Z hľadiska architektúry delíme SRBD na:

1. **Fyzicky aj logicky centralizované** - bežný model
2. **Fyzicky decentralizované, ale logicky centralizované** - užívateľ pristupuje k distribuovanej databáze bez znalosti toho, že je fyzicky decentralizovaná, javí sa mu ako centralizovaná.
3. **Logicky decentralizované systémy** , či už fyzicky decentralizované alebo nie - užívateľ k nim pristupuje ako k distribuovaným (aj keď možno o distribuovaný SRBD nejde).

V prípade sieťových SRBD sú iné nároky na optimalizáciu. Zvyčajne prevládne globálna optimalizácia (prenosy po sieti) nad lokálnou.

6.1. Distribuované zamykanie

- V prípade nedisjunktných dát sa používajú zámky R-LOCK (zamknutie konkrétnej stránky na zdieľané čítanie) a W-LOCK ALL (zamknutie všetkých kópií stránky na exkluzívny zápis). Za

istých okolností postačuje aj W-LOCK MAJOR (zamknutie väčšiny kópií stránky na exkluzívny zápis), čo môže ale umožniť čítanie špinavých dát.
Tieto dva prístupy možno zovšeobecniť do stratégie k z n,$n/2 < k \leq n$, kde n je počet kópií každej položky (na WRITE LOCK treba zamknúť k kópií položky, na READ LOCK $n - k + 1$).

- Ďalšou možnosťou je používanie tzv. primárnych kópií - každá položka má svoje primárne miesto a transakcie, ktoré žiadajú o zámok, musia žiadať o zámok primárnej kópie.
  Jednotlivé stratégie sa líšia jednak v množstve kontrolných informácií prenášaných sieťou, jednak správaním sa v prípade výpadku niektorého uzla.
- Ošetrenie DEADLOCKov vyžaduje, aby niektorý uzol mal prehľad o grafoch čakania všetkých uzlov(napr. v uzle A_1 čaká T_1 na T_2 a v uzle A_2 čaká T_2 na T_1). Tento spôsob okrem skutočných DEADLOCKov môže odhaliť aj fantómové - podtransakcia pošle graf čakania a potom sa rozhodne pre Abort, t.j. systém môže detekovať v danom okamihu už neexistujúci DEADLOCK. Pri druhom spôsobe každá transakcia po uplynutí vhodného time-out-u vykoná ROLLBACK.
- Sériovateľnosť transakcií : sériovateľné musia byť celé transakcie, nielen ich podtransakcie v rámci jednotlivých uzlov. V prípade využívanie časových známok na kontrolu sériovateľnosti treba zosynchronizovať časy jednotlivých uzlov a časové známky vydané rôznymi uzlami rozlíšiž napr. pomocou prefixu. Ďalší postup je ako pri nedistribuovaných systémoch, až nato, že pri zápise do položky treba aktualizovať známky všetkých jej kópií.

6.1.1. Distribuované dvojfázové zamykanie

- Predpokladajme, že všetky udalosti vieme zoradiť podľa času. Rozvrh nazveme **sériovým**, ak pozostáva zo všetkých akcií jednej transakcie, nasledovaných akciami ďalšej, atď., t.j. nestačí len sériovateľnosť podtransakcií v rámci každého uzla. Princíp dvojfázového protokolu ostáva nezmenený - najskôr sa len zamyká a potom len odomyká. Každá podtransakcia musí dať ostatným vedieť, či získala potrebné zámky, odomykať sa začína, až keď všetky podtransakcie potvrdia COMMIT.
- V prípade využívanie časových známok na kontrolu sériovateľnosti treba zosynchronizovať časy jednotlivých uzlov a časové známky vydané rôznymi uzlami rozlíšiž napr. pomocou prefixu. Ďalší postup je ako pri nedistribuovaných systémoch, až nato, že pri zápise do položky treba aktualizovať známky všetkých jej kópií.

6.2. Distribuované riadenie transakcií

- Princíp : Uzol, ktorý naštartuje transakciu je koordinátor, ostatné participujúce uzly sú účastníci. Každý z účastníkov sa môže rozhodnúť, či bude požadovať COMMIT alebo ABORT. Ak niektorý účastník požaduje ABORT, musia vykonať ROLLBACK všetci účastníci. Ak všetci účastníci požadujú COMMIT a koordinátor to COMMITom potvrdí, vykonajú sa zmeny a transakcia sa ukončí. To všetko ale predpokladá úplnú spoľahlivosť liniek a uzlov. V prípade zlyhania sa však môže stať, že niektorá z podtransakcií ostane odrezaná. Vtedy sa nemôže rozhodnúť samovoľne ani pre COMMIT ani pre ABORT - v oboch prípadoch to môže viesť k nekonzistencii, takže podtransakcia ostane zablokovaná a ak navyše drží nejaké zámky, môže blokovať celý systém. Existuje viac protokolov riešiacich problém blokovania, žiaden ho však nemôže vylúčiť úplne. Žurnál pre distribuované databázy musí okrem už spomínaných položiek obsahovať aj všetky odoslané a prijaté správy.

6.2.1. Dvojfázový commit protokol

Pre každého účastníka transakcie je nastavený **timeout**, vypršanie ktorého môže signalizovať poruchu v sieti.

Koordinátor : môže sa nachádzať v nasledovných stavoch :

1. **Initial** : Vyšle všetkým účastníkom signál **begin-vote**, ide do stavu **Waiting**
2. **Waiting** : Ak obdrží od všetkých účastníkov **vote-commit**,ide do stavu **Must commit**. Ak dostane aspoň jeden **vote-abort**, alebo signál **timeout**, ide do stavu **Must abort**.

3. **Must commit**: Vyšle všetkým účastníkom signál **commit**, ide do stavu **Commited**.
4. **Must abort**: Vyšle všetkým účastníkom signál **abort**, ide do stavu **Aborted**.

5.a 6. **Commited** resp **Aborted**

Účastník :

1. **Initial**: Ak dostane signál **begin-vote**, ide do stavu **Deciding**, inak po vypršaní **timeout**-u do stavu **Aborted**.
2. **Deciding**: Účastník sa rozhodne pre COMMIT (vyšle **vote-commit** a ide do stavu **Willing to commit**) alebo pre ABORT (vyšle **vote-abort** a ide do stavu **Aborted**).
3. **Willing to commit**: Účastník čaká na správu od koordinátora, pričom sú tri možnosti :
   (i) obdrží **commit** - ide do stavu **Commited**
   (ii) obdrží **abort** - ide do stavu **Aborted**
   (iii) vyprší **timeout**(výpadok siete a pod...) - ide do stavu **Blocked**
4. **Blocked**: Účastník vysiela do siete signál **help-me**. Príjemca signálu (koordinátor, iný účastník), ktorý vie, ako transakcia ďalej pokračuje, vracia **commit** resp. **abort**, na základe týchto odkazov nasleduje stav **Commited** resp. **Aborted**. V prípade, že účastníci už nie sú aktívni, signály **help-me** musí ošetriť systém.

5.a 6. **Commited** resp **Aborted**

Podobne vyzerá **trojfázový** protokol, kde sa zavádza ešte tretie kolo správ a ktorý ešte redukuje prípady blokovania (podrobnejšie viď napr.[1]).

7. Fyzická organizácia dát

7.1. Fyzický model dát

Pod fyzickou databázou môžeme rozumieť súbor viet (záznamov), pozostávajúcich z položiek elementárnych typov. Vzhľadom na rozsah databázy nemožno predpokladať, že celý obsah bude neustále v operačnej pamäti. Súbory sa zvyčajne delia do blokov veľkosti 2^9 až 2^{12} bytov (jeden blok zodpovedá veľkosti stránky virtuálnej pamäte, alebo stopy na disku atď). Výmena údajov medzi operačnou a sekundárnou pamäťou potom prebieha na úrovni blokov. Vzhľadom nato, že táto komunikácia je rádovo pomalšia ako výpočty v rámci operačnej pamäti, cena prístupu k dátam sa určuje v počtoch prístupu k blokom.
K jednotlivým záznamom sa pristupuje pomocou pointrov, či už absolútnych , alebo dvojicou **(b,k)**, kde **b** je adresa bloku a **k** je kľúč záznamu. V prípade, že na záznamy môžu ukazovať pointre, hovoríme o **pripichnutých záznamoch** (**pinned records**), inak o **nepripichnutých** (**unpinned**). Nepripichnuté záznamy možno voľne pohybovať, mazať, v prípade pripichnutých sa záznamy nemažú, len označujú ako zmazané. Záznamy môžu mať pevnú dĺžku (čo zapríčiňuje horšie využitie priestoru), alebo premennú (čo vyžaduje zložitejšie algoritmy).

7.2. Indexové súbory

Využitie indexových súborov prepokladá, že záznamy sú umiestnené v súbore utriedené podľa kľúča (bez ohľadu ma charakter kľúča, vzhľadom nato, že je reprezentovaný bitovým reťazcom, možno zaviesť reláciu usporiadania). K hlavnému súboru obsahujúcemu reťazce sa vytvorí pomocný súbor **index** pozostávajúci z dvojíc (kľúč , blok). Zvyčajne pre každý blok je v indexe odkaz naň.
Vyhľadávanie v indexovom súbore :

(i) - **lineárne** - zložitosť : v úspešnom prípade sa prehľadá v priemere polovica blokov indexu, v neúspešnom všetky

(ii) - **binárne** - vyžaduje usporiadanosť prehľadávaného súboru a pamäť s náhodným prístupom. Priemerná zložitosť je $\sigma(\log n)$, kde n je počet blokov indexu

(iii) - **interpolačné** - pracuje podobne ako binárne, ale interval sa nedelí na polovicu (ak hľadáme v telefónnom zozname napr. meno Šturc, neotvoríme ho v 50%, ale v 75% hrúbky). Predpokladá sa, že máme funkciu $f(v_1, v_2, v_3)$, ktorá pre dané v_1 vypočíta zlomok vyjadrujúci polohu v_1 medzi v_2 a v_3. Priemerná zložitosť je $\sigma(\log\log n)$, ale efektívnosť je strojovo závislá a prejavuje sa len pre veľké n.

Nakoľko index nie je nič iné ako súbor s nepripichnutými záznamami, dá sa vztvoriť index na index, atď. - (**hisam** - hierarchical indexed sequential acces method). Jednou z foriem viacúrovňových indexových štruktúr sú napr. **B-stromy**.

<u>Definícia 26(B-stromu m-tého stupňa):</u> (podľa [7])

1.- každý uzol má najviac m synov
2.- každý uzol okrem koreňa a listov má aspoň $\frac{m}{2}$ synov
3.- koreň, ak nie je list, má aspoň dvoch synov
4.- nelistový uzol s k synmi obsahuje $k-1$ kľúčov
5.- všetky listy sú na jednej úrovni
Ak uzol obsahuje kľúče $K_1 \dots K_j$, pointre na synov $p_0 \dots p_j$, potom prvok s kľúčom k, $k_i < k < k_{i+1}$, leží v podstrome p_i.

Vyhľadávanie v B-stromoch vyžaduje v priemere $O(\log_m n)$ operácií, faktor zaplnenia stránky je približne $\ln 2$. Niekedy sa používajú aj B^*-stromy, ktoré namiesto $\frac{m}{2}$ až m synov majú $\frac{2m-1}{3}$ až m synov.

7.3. Hašované súbory

Hlavná idea hašovaného súboru je rozdelenie záznamov v súbore do m bucketov, ku ktorým sa pristupuje pomocou **hašovacej funkcie h**. $h(k)$ pre každý záznam s kľúčom k určuje bucket, v ktorom sa môže záznam nachádzať. Jeden bucket zodpovedá bloku alebo skupine blokov. Základnou požiadavkou hašovacej funkcie je, aby rozmiestňovala kľúče čo možno najrovnomernejšie, t.j. $h(k)$ nadobúda všetky hodnoty so zhruba rovnakou pravdepodobnosťou. Jednotlivé metódy sa navyše líšia spôsobom riešenia kolízií (t.j. situácií, keď dôjde k preplneniu bucketu).

<u>Definícia 27(faktoru zaplnenia):</u> Nech adresár má m bucketov a je v ňom už n bucketov plných. Potom $\alpha = \frac{n}{m}$ sa nazýva faktor zaplnenia (pri hašovaní s kolíziami riešenými zreťazením môže byť aj $\alpha > 1$).

Niektoré hašovacie funkcie :

- **stred mocniny** - kľúč sa vynásobí sám sebou a ako výsledok sa zoberie niekoľko bitov zo stredu. Tieto bity totiž závisia od všetkých bitov kľúča.
- **metóda delenia** - $h(k) = k \mod m$, pričom efektívnosť závisí od m. Ak m je mocninou základu číselnej sústavy počítača, do úvahy sa zoberú len posledné bity m. Najvýhodnejší prípad je nejaké prvočíslo.
- **metóda násobenia** - $h(k) = \lfloor m((Ak) \mod 1)\rfloor$. Táto metóda je duálna k metóde delenia, odporúčaná hodnota konštanty A je hodnota *zlatého rezu*: $\frac{\sqrt{5}-1}{2}$

Metódy riešenia kolízií (t.j. preplnenia nejakého bucketu):

a, - **kolízie riešené zreťazením**: pri preplnení bucketu sa vytvára spájaný zoznam (prípadne iná štruktúra). Dá sa ukázať, že priemerný počet pokusov pri hľadaní je $1 + \frac{\alpha}{2}$ pri úspešnom hľadaní a $e^{-\alpha} + \alpha$ pre neúspešné([7]).

b, - **kolízie riešené otvorenou adresáciou**: ak bucket $h(k)$ je plný, hľadá sa ďalší voľný, t.j. hašovacia funkcia je tvaru $h'(k,i)$, kde i je číslo pokusu. Ideálny prípad je, ak každá z $n!$ permutácií je rovnako pravdepodobná ako $h'(k,1), \dots, h'(k,n)$ (tzv. **uniformné** hašovanie).
Priemerný počet krokov na vloženie (vyhľadanie) prvku je pri uniformnom hašovaní $\frac{1}{\alpha}\log\frac{1}{1-\alpha}$.
Pri tomto riešení kolízií sa nedajú prvky vymazať úplne, mohlo by sa narušiť zaplnenie nejakého bloku a tým prerušiť vyhľadávanie tých prvkov, pre ktoré hašovacia postupnosť bucketov vedie cez daný bucket, preto sa len označujú ako zmazané.
Metódy otvorenej adresácie :

(i) **lineárna**: $h'(k,i) = (h(k)+ci) \mod m$. Nevýhodou metódy je primárne zhlukovanie sa kolidujúcich prvkov(ak $h(k_1) = h(k_2)$).
(ii) **kvadratická**: $h'(k,i) = (h(k) + ai + bi^2) \mod m$
(iii) **dvojité hašovanie**: $h'(k,i) = h_1(k) + ih_2(k) \mod m$, kde h_1 a h_2 sú hašovacie funkcie. Príkladom hašovacích funkcií môžu byť napr.: $h_1(k) = k \mod m$, $h_2(k) = 1 + (k \mod (m-1))$. Funkcie h_1 a h_2 by mali byť *nezávislé*, t.j. $h_1(k) = h_2(k)$ s pravdepodobnosťou $\frac{1}{m^2}$.

7.3.1. Rozšíriteľné hašované súbory

Princíp - veľkosť adresáru sa mení podľa potrieb, nedá sa prehašovať. Adresár má v ľubovoľnom čase veľkosť 2^m pre nejaké **m**. Hašovacia funkcia vráti $h(k)$, pričom do úvahy sa zoberie len prvých m bitov. V adresári je 2^m smerníkov na buckety, pričom môže viac smerníkov ukazovať na ten istý bucket. Každý bucket má priradenú určitú hĺbku - číslo **d** z intervalu $1 \ldots m$ a bitový reťazec **s**, pričom v buckete sú tie záznamy, pre ktoré sa prvých **d** bitov zahašovaného kľúča $h(k)$ zhoduje s **s**. Ak sa nejaký bucket(**d**,**s**) prehašuje, rozdelí sa na dva buckety $(d_0$,$s_0)$,$(d_1$,$s_1)$nasledovne : $d_0 = d_1 = d+1$, $s_0 = s0$, $s_1 = s1$ - hĺbka nových bucketov je zväčšená o 1, záznamy sa do nových bucketov rozdelia podľa toho, či majú na $d+1$ - om mieste v $h(k)$ 0 alebo 1.

Ak je hĺbka nových bucketov väčšia ako **m** (hĺbka adresára), **m** sa zväčší o 1, t.j. adresár sa zväčší na dvojnásobok(na obrázku by táto situácia mohla nastať pretečením bucketu B alebo C).

Naopak, ak sú vedľa seba dva buckety rovnakej hĺbky,ktorých reťazce sa líšia len na poslednom mieste , ktoré sú zaplnené na menej ako polovicu, možno ich zlúčiť (napr. B a C).

	Adresár $m = 3$	Buckety
	$000 \to$ A	A $\boxed{d = 2, s = 00}$
	$001 \to$ A	
	$010 \to$ B	B $\boxed{d = 3, s = 010}$
kľúč$\to h(k) \to$ výber prvých 3 bitov $\to$	$011 \to$ C	
	$100 \to$ D	C $\boxed{d = 3, s = 011}$
	$101 \to$ D	
	$110 \to$ D	D $\boxed{d = 1, s = 1}$
	$111 \to$ D	

Rozšíriteľné hašovanie

7.4. Vyhľadávanie na čiastočnú zhodu a intervalové dotazy

Vyhľadávaním na čiastočnú zhodu a intervalovými dotazmi nazývame dotazy, kde sú špecifikované len niektoré položky netvoriace kľúč, resp. položky sú obmedzené intervalmi. Týmto druhom dotazov, ak majú byť efektívne, sa musí prispôsobiť aj fyzická štruktúra (napr. priamy dotaz v hašovanom súbore vedie k úplnému prehľadaniu súboru).

7.4.1. Rozkladové (partitioned) hašovacie funkcie

Nech kľúč má položky $k = k_1, k_2, \ldots, k_n$, nech hašovacia funkcia $h_i(k_i)$ vráti reťazec b_i bitov, nech adresár má veľkosť 2^b, nech $\sum_{i=1}^{n} b_i = b$. Potom zreťazením hodnôt $h_1(k_1), \ldots, h_n(k_n)$ získame hašovaciu funkciu h. Pri dotaze na čiastočnú zhodu vypočítame bity $h_(k)$ určené špecifikovanými položkami, treba prehľadať všetky buckety, ktorých adresa vyhovuje získanej maske.
Aby sa hašovacia funkcia dala použiť na intervalové dotazy, musela by nejakým spôsobom zachovávať poradie, napr. $k_1 < k_2 \implies h(k_1) < h(k_2)$, čo však odporuje filozofii hašovacích funkcií. Aby funkcia zobrazovala na každý bucket približne rovnaký počet kľúčov, musíme poznať vopred ich distribúciu. Ak $a_0, \ldots, a_m$ sú také hodnoty, že v každom z intervalov (a_i, a_{i+1}) leží približne rovnaké množstvo kľúčov, potom hašovacia funkcia zobrazujúca číslo z intervalu (a_i, a_{i+1}) do bucketu B_i vyhovuje obom požiadavkom (rovnomernej distribúcii aj usporiadanosti).

7.4.2. k-d stromy

k-d stromy vzniknú modifikáciou binárnych vyhľadávacích stromov pre kľúče pozostávajúce z viacerých zložiek tak, že v každej úrovni sa porovnáva iná zložka kľúča. Potom pri dotazoch na čiastočnú zhodu treba v uzle, kde sa porovnáva špecifikovaná zložka kľúča, ísť zodpovedajúcou cestou, v uzle, kde nie je definovaná porovnávaná zložka, treba prehľadať obe vetvy. Ak predpokladáme, že všetky listy ležia v jednej úrovni v hĺbke $\log n$, potom pri dotaze na čiastočnú zhodu,pri ktorej je z k zložiek kľúča špecifikovaných m, sa výpočet rozdelí v $\frac{k-m}{k}\log n$, t.j. dosiahne sa $n^{\frac{k-m}{k}}$ listov.

K-d stromy a rozkladové hašovacie funkcie sú pri dotazoch na čiastočnú zhodu a intervalových dotazoch približne rovnako efektívne([7]).

7.5. Redukcia dát

Redukciou dát sa znížia nároky na priestor, skráti sa čas prístupu k údajom a zníži sa množstvo prenášných dát. Nevýhodou je Zvyšujúca sa zložitosť algoritmov, manipulácia s premenlivou dĺžkou kódu … .

7.5.1. Huffmanovo kódovanie

Huffmanov kód je kód premenlivej dĺžky spĺňajúci *prefixovú* vlastnosť (žiaden kód nie je prefix iného).
Algoritmus 14(Huffmanovo kódovanie):

`INPUT:` n jednotiek(napr.znakov) na kódovanie, postupnosť pravdepodobností $p[i], 1 \leq i \leq n$, ich výkytov

`OUTPUT:` kódovanie jednotiek dané zreťazením ohodnotení hrán na cestách k jednotlivým listom binárneho stromu

```
pre každú jednotku (znak) i vytvoríme strom - list ohodnotený o(p[i]);
k := n;
while k ≥ 2 do
  vyber zo stromov dva s najmenším ohodnoteniami koreňa r,s, nech q := r + s;
  spoj tieto dva stromy o(r),o(s) cez koreň o(q) hranami (o(q),o(r)),(o(q),o(s));
  ohodnoť jednu hranu 0-ou, druhú 1-kou;
  k := k − 1;
```

7.5.2. Kompresia dát adaptívnym slovníkom

Podstatou metódy je vytvorenie určitého slovníku, ktorý sa v pamäti vytvára súbežne s kódovaným textom. Pri dekodódovaní je slovník opäť paralelne generovaný. Pre danú abecedu Σ sa vytvorí les n-árnych stromov hĺbky h. Pre daný reťazec S potom definujme **regál slovníkov** $F_h(S)$ ako zodpovedajúci les, kde každá cesta z koreňa zodpovedá podreťazcu z S a naopak (uzly v lese sú ohodnotené znakmi zo Σ). Kódovanie reťazca S spočíva v nahradzovaní jeho podreťazcov adresami zodpovedajúcimi adrese posledného znaku podreťazca existujúceho ešte v slovníku. Metóda je jednoprechodová.

Algoritmus 15(Kompresia adaptívnym slovníkom):

`INPUT:` reťazec S nad abecedou σ

`OUTPUT:` zoznam adries

```
begin
  vytvor pre Σ prázdny regál slovníkov;
    {pre každý znak zo Σ list ohodnotený daným znakom}
  k := 1;
  zakóduj S(1) ako adresu zodpovedajúceho koreňa stromu
  ( v cykle predpokladajme, že je zakódovaný S(1,k),je vytvorený F_h(S(1,k)))
  while k ≤ n do begin
    vyhľadaj v S maximálny S(k+1,k+z),z ≥ 1, taký, že je v F_h(S(1,k));
      {adresa S(k+z) nech je adr}
    zakóduj S(k+1,k+z) ako adr;
    vytvor F_h(S(1,k+z) vložením podreťazcov
           S(k−h+2,k+1),...,S(k+z−h+1,k+z) do F_h(S(1,k));
```

```
        k := k + z;
      end;
    end
```

Algoritmus dekódovania je inverzný.

8. Logické a deduktívne bázy dát

8.1. Rozšírenie relačnej databázy na deduktívnu - DATALOG

- DATALOG - formálne odvodený od PROLOGu, ktorý tiež narába s reláciami, ako interpretovanými predikátmi. Používané predikáty sú:

1. **extenzionálne:** súbor externých faktov daných obsahom databázových tabuliek; tieto predikáty sú interpretované reláciami zodpovedajúcimi databázovým tabuľkám.
2. **zabudované:** Ide o preddefinované, aj nekonečné predikáty ako $=, >, <, \leq, \geq, \ldots$.
3. **intenzionálne:** definované z ostatných (aj iných intenzionálnych, ba dokonca seba samých - rekurzia) pomocou pravidiel; sú interpretované tzv. *virtuálnymi* reláciami. Forma pravidiel je prebratá z PROLOGu, odlišný je spôsob výpočtu. Prologovský súhrn faktov je zadaný extenzionálnou databázou, použité pravidlá sú Hornove klauzuly, t.j. majú tvar:

$$r(x,y) : -p_1(x,y), p_2(x,y), \ldots, p_n(x,y)$$

kde r je nezabudovaný a neextenzionálny predikátový symbol s významom

$$(\forall)(p_1(x,y) \wedge p_2(x,y) \wedge \cdots \wedge p_n(x,y) \rightarrow r(x,y))$$

(Všetky predikáty majú skutočne len pozitívne výskyty - negácia je zakázaná, sú možné rozšírenia o negáciu za istých podmienok). Otázkou je, ako zvoliť model, t.j. akými reláciami interpratovať intenzionálne predikáty za splnenia podmienok vyplývajúcich z pravidiel. Je žiadúca ekvivalencia prístupov pri vyhodnocovaní dotazov:

1. Platí sémanticky, t.j. platí v každom modeli, ktorému vyhovuje program ($\models$).
2. Platí syntakticky, t.j. dotaz možno odvodiť ako dôkaz ($\vdash$).
3. "Platí" algoritmicky, t.j. dotaz možno algoritmicky vyhodnotiť ("vypočítať").

Snaha o ekvivalenciu týchto prístupov vedie k interpretácii intenzionálnych predikátov "najmenším" modelom za predpokladu uzavretého sveta, čo pri výpočte znamená hľadanie najmenšieho pevného bodu. Model $\mathcal{I} = (\mathcal{D}, \mathcal{R})$, kde $\mathcal{D}$ je doména a $\mathcal{R}$ interpretácia predikátov, možno minimalizovať rôznymi spôsobmi. Ideálne by bolo zaoberať sa doménovo nezávislými formulami, v tom prípade nie je nutné zaoberať sa doménou $\mathcal{D}$. Doménová nezávislosť je však nerozhodnuteľná. V skutočnosti sa $\mathcal{D}$ vôbec minimalizovať nedá. Minimalizácia $\mathcal{D}$ môže viesť k dôkazu tvrdenia, ktoré pri tej istej $\mathcal{R}$, ale inom $\mathcal{D}$ neplatí (čo je nežiadúce).

Príklad 6: Majme tvrdenia:
(1) Každý má šéfa.
(2) Existuje Katriňák.
(3) Existuje Šturc.
(4) Nikto nie je sám sebe šéfom.

Pri minimalizácii domény na $\mathcal{D} = \{\check{S}turc, Katri\check{n}\acute{a}k\}$ dostaneme, že Šturc je Katriňákov šéf a Katriňák Šturcov, čo nie je odvoditeľné pri väčšom modeli.

Preto sa doména $\mathcal{D}$ neminimalizuje, ale uvažuje sa $|\mathcal{D}| = \aleph_0$, pričom $\mathcal{D}$ obsahuje všetky konštanty zmienené v extenzionálnej databáze.

Príklad 7: Uvažujme extenzionálny predikát r(x,y) a program

```
t(X,Y):-r(X,Y)
```

`t(X,Y):-r(X,Z),t(Z,Y)`

Tento program vypočíta tranzitívny uzáver t relácie r. DATALOG teda nie je slabší ako relačný kalkul, pretože v ňom tranzitívny uzáver vyjadriť nemožno.

Predpoklad uzavretého sveta: Tvrdenie, ktoré nie je možné z dostupných údajov odvodiť ako pravdivé, je nepravdivé.

Predpoklad uzavretého sveta môže viesť k nekonzistenciám.

Príklad 8: Majme iba extenzionálny predikát r a pravidlá

`p(X):-r(X)`

`p(0):-¬p(1)` - Tento zápis má v zmysle predchádzajúceho význam $p(0) \vee p(1)$ a nie je Hornovou klauzulou.

Predpokladajme, že neplatí r(0) ani r(1). Potom nemožno odvodiť ani p(0) ani p(1) a z predpokladu uzavretého sveta vyplýva, že platí $\neg p(0)$ a $\neg p(1)$, čo je spor so semantickým významom druhého pravidla. Druhé pravidlo nemá dokonca ani najmenší model, pretože ako $\{p(0), \neg p(1)\}$, tak aj $\{\neg p(0), p(1)\}$ sú jeho modely.

V prípade, že všetky použité pravidlá sú Hornove klauzuly, k nekonzistenciám nedôjde.

Veta 11: Množina Hornových klauzúl má najmenší model rovný prieniku všetkých modelov.

8.2. Výpočtový proces - vyhodnocovanie deduktívnych dotazov

Pretože jeden intenzionálny predikát môže byť popísaný aj viacerými pravidlami líšiacimi sa ľavou stranou (či už názvami premenných alebo výrazmi typu p(x,x)), upravujú sa zavedením nových premenných tak, aby ich ľavé strany boli totožné. Tomuto procesu sa hovorí `rektifikácia ľavých strán`.

Príklad 9: Majme pravidlá

`p(a,X,Y):-r(X,Y)`

`p(X,Z,X):-r(Z,X)`

Zavedením nových premenných U, V, W urobíme rektifikáciu na:

`p(U,V,W):-U=a,V=X,W=Y,r(X,Y)`

`p(U,V,W):-U=X,V=Z,W=X,r(Z,X)`

Prípadne je možné previesť aj optimalizáciu vzniknutých pravých strán.

Je zmysluplné špecifikovať **bezpečné pravidlá**, pretože zabudované predikáty sú nekonečné a umožňujú vznik nekonečných dotazov.

Definícia 28(ohraničenej premennej): Premenná X je v pravidle P ohraničená $\iff$ platí aspoň jedno z nasledujúcich:

(1) Je v tele atomického nenegovaného predikátu na pravej strane pravidla P.
(2) Na pravej strane pravidla P sa vyskytuje X=a, kde a je konštanta.
(3) Na pravej strane pravidla P sa vyskytuje X=Y, kde Y je premenná ohraničená v P.

Definícia 29(bezpečného pravidla): Pravidlo je bezpečné $\iff$ všerky jeho premenné sú v ňom ohraničené.

Budeme sa zaoberať len bezpečnými pravidlami.

8.2.1. Výpočet bezpečných pravidiel bez rekurzie a negácie

Takéto dotazy sa priamo prevedú do relačnej algebry, pričom zabudované predikáty predstavujú selekciu a konjunkcia na pravej strane prirodzené spojenie. Výsledok sa upraví projekciami.

Príklad 10:

`p(a,Y,Y)` sa prevedie na

$$\sigma_{X=a \wedge Y=Z} p(X,Y,Z)$$

`p(X,Y):-q(X,Z),r(Z,Y)` sa prevedie na

$$\pi_{XY} q(X,Z) \bowtie r(Z,Y)$$

`p(X,Y):-q(X,U),r(V,Z),U<V` sa prevedie na

$$\pi_{XY} \sigma_{U<V} q(X,U) \times r(V,Z)$$

V prípade, že je predikát popísaný viacerými pravidlami, po prevedení rektifikácie ľavých strán možno pristúpiť k zjednoteniu pravých strán. Ak je program pre p nasledovný:
`p(X,Y):-`ϕ_1
`p(X,Y):-`ϕ_2 a formuly ϕ_1, ϕ_2 sa prevedú na relačné výrazy Φ_1, Φ_2, tak relačný výraz pre p je $\Phi_1 \cup \Phi_2$.

8.2.2. Výpočet bezpečných pravidiel s negáciami a bez rekurzie

Bezpečná negácia sa samozrejme prevedie na operáciu rozdielu. Pri prevode sa môžu vyskytnúť isté technické ťažkosti.

Príklad 11:
`p(X,Y):-r(X,Y),¬q(X,Y)` sa prevedie na

$$r(X,Y) \setminus q(X,Y)$$

`p(X,Y):-r(X,U),¬q(X,Z),t(Z,Y)` sa prevedie na

$$\pi_{XY}(((\pi_{XZ}(r(X,U) \times t(Z,Y))) \setminus q(X,Z)) \bowtie t(Z,Y))$$

8.2.3. Výpočet bezpečných pravidiel bez negácie s rekurziou

Ak máme program
`p:-E`$_1$
$\vdots$
`p:-E`$_n$ výpočet p sa prevedie na nájdenie najmenšieho pevného bodu relačnej rovnice $p = E_1 \cup \cdots \cup E_n$. V prípade, že sa na pravej strane nevyskytuje p, existuje jediný pevný bod a jeho výpočet bol popísaný už v predchádzajúcich kapitolách.

- (v ďalšom viď. aj [6])

Definícia 30(čiastočne usporiadenej množiny): Nech $A \neq \emptyset$ je množina a $\sqsubseteq$ je binárna relácia na A. Hovoríme, že $(A, \sqsubseteq)$ je čiastočne usporiadaná množina $\iff$
(1) $(\forall x \in A)(x \sqsubseteq x)$
(2) $(\forall x \in A)(\forall y \in A)(\forall z \in A)((x \sqsubseteq y \wedge y \sqsubseteq z) \implies x \sqsubseteq z)$
(3) $(\forall x \in A)(\forall y \in A)((x \sqsubseteq y \wedge x \sqsupseteq y) \implies x = y)$

Definícia 31(cpo): Hovoríme, že čiastočne usporiadaná množina $(A, \sqsubseteq)$ je **cpo** $\iff$
(1) $(\exists \perp \in A)(\forall x \in A)(\perp \sqsubseteq x)$
(2) Každý reťazec $x_0 \sqsubseteq x_1 \sqsubseteq \ldots$ má v A supremum, t.j. $(\exists y \in A)((\forall i \in \mathbb{N})(x_i \sqsubseteq y) \wedge$ y je najmenšie (v zmysle $\sqsubseteq$) také) Toto supremum označujeme $\sqcup_{i=0}^{\infty} x_i$.

Definícia 32(monotónnej funkcie): Funkcia $f : A \to B$, kde A, B sú čiastočne usporiadané, je monotónna $\iff$ $(\forall x \in A)(\forall y \in A)(x \sqsubseteq y \implies f(x) \sqsubseteq f(y))$.

Definícia 33(spojitej monotónnej funkcie): Monotónna funkcia $f : A \to B$, kde A, B sú cpo je spojitá $\iff$ pre každý reťazec $x_0 \sqsubseteq x_1 \sqsubseteq \ldots$ v A platí, že reťazec $f(x_0) \sqsubseteq f(x_1) \sqsubseteq \ldots$ v B má supremum $f(\sqcup_{i=0}^{\infty} x_i)$, t.j. $\sqcup_{i=0}^{\infty} f(x_i) = f(\sqcup_{i=0}^{\infty} x_i)$

Poznámka: Ak A je konečná množina, je každá na nej monotónna funkcia spojitá.

Definícia 34(pevného bodu): Nech $f : A \to B$ je funkcia. Potom $x \in A$ také, že $f(x) = x$ nazývame pevným bodom funkcie f.

Definícia 35(najmenšieho pevného bodu): Nech $f : A \to B$ je funkcia, kde A, B sú cpo. Pevný bod x funkcie f nazývame najmenší $\iff$ pre každý pevný bod y funkcie f platí, že $x \sqsubseteq y$.

Veta 12(Tarski): Nech $f : A \to B$ je spojitá monotónna funkcia, pričom A, B sú cpo. Potom f má najmenší pevný bod, a to

$$\sqcup_{i=0}^{\infty}(f^i(\perp))$$

kde

$$(\forall x \in A) f^0(x) = \perp,$$
$$f^1 = f,$$
$$(\forall i \in \mathbb{N}^+)(\forall x \in A) f^{i+1}(x) = f(f^i(x))$$

Dôkaz: $\rhd$ Pretože $f^0(\perp) = \perp \sqsubseteq f(\perp) = f^1(\perp)$ a f je monotónna, tak z indukčného predpokladu $f^i(\perp) \sqsubseteq f^{i+1}(\perp)$ plynie priamo $f^{i+1}(\perp) \sqsubseteq f^{i+2}(\perp)$. Tým je ukázané, že $f^i(\perp)$ je reťazec a má supremum $\sqcup_{i=0}^{\infty}(f^i(\perp))$. Pretože f je spojitá, tak $f(\sqcup_{i=0}^{\infty}(f^i(\perp))) = \sqcup_{i=0}^{\infty} f(f^i(\perp)) = \sqcup_{i=0}^{\infty} f^{i+1}(\perp) = \sqcup_{i=1}^{\infty}(f^i(\perp)) = \sqcup_{i=0}^{\infty}(f^i(\perp))$. Teda $\sqcup_{i=0}^{\infty}(f^i(\perp))$ je pevným bodom funkcie f.
Nech y je ľubovoľný pevný bod funkcie f. Indukciou dokážeme, že $f^i(\perp) \sqsubseteq y$. Platí, že $f^0(\perp) = \perp \sqsubseteq y$ a z indukčného predpokladu $f^i(\perp) \sqsubseteq y$ a monotónnosti f priamo vyplýva, že $f^{i+1}(\perp) \sqsubseteq f(y) = y$. Teda $\sqcup_{i=0}^{\infty}(f^i(\perp))$ je skutočne najmenším pevným bodom funkcie f. Q.E.D. $\lhd$

Poznámka: Dôkaz zároveň obsahuje návod, ako najmenší pevný bod vypočítať.

Tvrdenie 2: Nech A je ľubovoľná množina. Potom $(2^A, \subseteq)$ je cpo, pričom $\perp = \emptyset$. Všeobecnejšie $((2^A)^n, \sqsubseteq)$ je cpo, kde $(A_1, \ldots, A_n) \sqsubseteq (B_1, \ldots, B_n) \iff A_i \subseteq B_i, i = 1, \ldots, n$, pričom $\perp = (\emptyset, \ldots, \emptyset)$.

V prípade programu zloženého z Hornových klauzúl je funkcia daná výrazom pravej strany, totiž

$$f_p : (2^A)^n \to 2^A, f_p((E_1, \ldots, E_n)) = E_1 \cup \cdots \cup E_n$$

monotónna vzhľadom na $\subseteq$ pretože "$\cup, \bowtie, \sigma, \pi, \times$" sú monotónne. Pretože pri výpočte môžu vznikať len tvrdenia o konštantách z externej databázy a tých je konečne veľa (naša množina A), je definičný obor tejto funkcie $((2^A)^n)$ aj obor hodnôt (2^A) konečný, a preto ide o spojitú monotónnu funkciu.

Pri praktickej realizácii (vrátane zložitého vzájomného volania sa počítaných predikátov) možno postupovať takto: Zostrojíme graf, ktorého uzlami budú jednak použité extenzionálne a intenzionálne predikáty a jednak "stroje" predstavujúce pravé strany rovníc. Hranou bude spojený predikát so "strojom" v prípade, že ho "stroj" používa. Ak takýto graf je acyklický, nejde o rekurzívny prípad, topologickým utriedením dostaneme jednu možnosť ako postupovať pri výpočte (viď predchádzajúce kapitoly). Ak je cyklický postupuje výpočet po cykloch podľa Tarského vety.

Algoritmus 16(naivný algotitmus výpočtu):

`INPUT:` DATALOGovské predikáty (intenzionálne predikáty $p_1, \ldots p_m$ a extenzionálne relácie $R_1, \ldots R_k$) a pravidlá pre výpočet intenzionálnych predikátov

`OUTPUT:` Najmenší pevný bod týchto pravidiel

```
for i:=1 to m do
   P_i := ∅;
repeat
   for i:=1 to m do
      Q_i := P_i;
   for i:=1 to m do
      P_i :=EVAL(p_i, R_1, ..., R_k, Q_1, ..., Q_m);
until P_i = Q_i for all 1 ≤ i ≤ n;
Output all P_i;
```

Funkcia EVAL(p,X) tu predastavuje našu funkciu f_p. Do p_i teda postupne pribúdajú prvky. Nemá však zmysel pridávať tam prvky, ktoré tam už sú. Nový prvok však môžme dostať iba vtedy, ak v aspoň jednej E_i použijeme aspoň jeden "novovyrobený" prvok. Teda na výpočet použijeme funkciu EVAL-INCR$(p, R_1, \ldots, R_k, E_1, \ldots, E_n, \Delta E_1, \ldots, \Delta E_n)$, ktorá počíta hodnotu pre p tak, že použije aspoň jednu hodnotu z ΔE_i pre nejaké i, R_i predstavujú externé relácie.

Algoritmus 17(seminaivý algoritmus výpočtu):

INPUT: DATALOGovské predikáty (intenzionálne predikáty $p_1, \dots p_m$ a extenzionálne relácie $R_1, \dots R_k$) a pravidlá pre výpočet intenzionálnych predikátov

OUTPUT: Najmenší pevný bod týchto pravidiel

```
for i:=1 to m do begin
   ΔP_i:=EVAL(p_i, R_1, ..., R_k, ∅, ..., ∅);
   P_i:=ΔP_i;end;
repeat
   for i:=1 to m do
      ΔQ_i:=ΔP_i;
   for i:=1 to m do begin
      ΔP_i:=EVAL-INCR(p_i, R_1, ..., R_k, P_1, ..., P_m, ΔQ_1, ..., ΔQ_m);
      ΔP_i:=ΔP_i \ P_i;end;
   for i:=1 to m do
      P_i:=P_i ∪ ΔP_i;
until ΔP_i = ∅ for all i;
Output all P_i;
```

8.2.4. Výpočet bezpečných pravidiel s rekurziou a negáciou

Do predchádzajúceho modelu prináša negácia operáciu rozdielu, tá ho však narúša, pretože nie je monotónna. V skutočnosti je problém oveľa hlbší. Je totiž možné, že pevný bod vôbec existovať nebude.

Príklad 12: Uvažujme program

```
p(x):-¬p(x),r(x),
```

pričom r je neprázdna externá relácia. Výpočet predchádzajúcou metódou prináša nemotónnu nekonečnú postupnosť pre p, a to $r, \emptyset, r, \emptyset, \dots$. Nevedie teda k výsledku. Keby ale P bolo pevým bodom rovnice $p = (A \setminus p) \cap r$, kde A je množina všetkých konštánt v r, tak

1. P nie je prázdne, pretože $(A \setminus \emptyset) \cap r \neq \emptyset$
2. P nie je neprázdne, pretože keby $a \in P$, tak by muselo platiť, že $a \in (A \setminus P) \cap r$, a teda $a \notin P$, čo je spor.

Táto rovnica teda nemá žiaden pevný bod.

Riešením je teda povoliť iba niektoré negácie. Na základe predchádzajúceho už vieme, že v poriadku je výpočet s negáciami ale bez rekurzie. Negáciu teda zakážeme v rámci rekurzívneho výpočtu. Možno ju použiť iba na predikát, ktorý je buď externý alebo už plne vypočítaný. Takéto negácie sa označujú ako *stratifikované.* Výpočet treba teda vzhľadom na negácie usporiadať tak, aby sa toto pravidlo dodržalo (ak sa také usporiadanie realizovať dá). Tomuto procesu sa hovorí *stratifikácia negácií.* Negácie sú teda rozmiestnené do vrstiev, pričom negácia môže používať len predikáty z nižších vrstiev. Nultú vrstvu tvoria prirodzene extenzionálne relácie. Vždy, ak pravidlo r používa na pravej strane negáciu predikátu q, musí byť jeho ľavá strana p vysunutá do úrovne aspoň o 1 vyššej ako q. Pritom ľavá strana musí byť v aspoň takej úrovni ako predikáty pravej strany. Ak sa pri takomto postupe stane, že je potrebných viac úrovní ako predikátov, tak negované predikáty sú v rekurzívnom cykle a stratifikácia nie je možná.

Algoritmus 18(stratifikácie negácií):

INPUT: extenzionálne a intenzionálne predikáty, DATALOGovské pravidlá

OUTPUT: pole stratum[ ], ktoré pre každý predikát určuje jeho vrstvu alebo informácia o tom, že stratifikácia nie je možná

```
for each predicate p do
   stratum[p]:=1;
repeat
   for each rule r with head predicate p do begin
```

```
            for each negated subgoal of r with predicate q do
               stratum[p]:=max(stratum[p],1+stratum[q]);
            for each nonnegated subgoal of r with predicate q do
               stratum[p]:=max(startum[p],stratum[q]);end;
      until there are no more changes to any stratum or some stratum
   exceeds the number of predicates;
```

8.3. Porovanie DATALOGovského a PROLOGovského výpočtu

Hlavným rozdielom týchto výpočtov je, že DATALOG vyhodnocuje dotaz "zdola" narozdiel od PROLOGu, ktorý ho vyhodnocuje "zhora". Rozdiel teda spočíva v tom, že DATALOG najprv vypočíta celú reláciu(-ie), ktorých sa dotaz týka a potom z nich potrebnú informáciu vyselektuje. PROLOG realizuje výpočet zhora - dedukuje, t.j. realizuje len výpočty skutočne potrebné pre odvodenie informácií potrebných pre vyhodnotenie dotazu. Metódou jeho výpočtu je SLD-rezolúcia využívajúca ľavý BACK-TRACK.

Príklad 13: Majme program

```
p(X,Y):-e(X,Y).
p(X,Y):-e(X,Z),p(Z,Y).
e(b,c).
e(a,b).
```

Ak zadáme dotaz, či platí p(a,c), PROLOG sa najprv pokúsi pomocou prvého pravidla zistiť, či platí e(a,c). To je "extenzionálny" predikát, a tak sa ľahko zistí, že podcieľ e(a,c) sa nepodarilo splniť (subgoal failed). Nasleduje BACK-TRACK. Pomocou 2. pravidla sa najprv stanoví podcieľ e(X,Z). Ten sa podarí splniť pre X=b, Z=c. Preto sa ďalej stanoví podcieľ p(c,c). Po niekoľkých BACK-TRACKových krokoch sa zistí, že cieľ nemožno splniť. Nasleduje BACK-TRACK v 2.pravidle pre podcieľ e(X,Z), ten stanoví X=a, Y=b a nový podcieľ p(b,c), ten sa po niekoľkých krokoch podarí splniť.

PROLOG postupuje pri vytváraní podcieľov pravej strany zľava doprava. PROLOG je teda pri výpočte šetrnejší, počíta len to, čo je nutné. Tento postup je síce v čase exponenciálny, ale pri obrovskej externej databáze môže byť rýchlejší ako DATALOGovský výpočet zdola, ktorý je síce polynomiálny, ale vo veľkosti databázy. PROLOG však pristupuje k prvkom externej databázy po jednom zázname. Ak je externá databáza uložená na vonkajšom pamäťovom médiu, je prístup zdola, ktorý naraz načítava veľké množstvo dát, rýchlejší. Jednou z nevýhod SLD-rezolúcie s ľavým BACK-TRACKom je, že je potenciálne nekonečná (problémy vyvoláva okrem iného ľavá rekurzia). Ošetrenie je ponechané na programátora.

Príklad 14: Ak program z predchádzajúceho príkladu pozmeníme na

```
p(X,Y):-e(X,Y).
p(X,Y):-p(X,Z),p(Z,Y).
e(b,c).
e(a,b).
```

čo je logicky ten istý predikát p a DATALOG aj dospeje k tomu istému výsledku, PROLOG bude pri spracovávaní 2. pravidla vyvolávať vždy nový podcieľ p(X,Z) a výpočet bude nekonečný.

9. Optimalizácia

Optimalizovať možno

1. Fyzické uloženie dát v extérnej pamäti
2. Vyhodnocovanie základných relačných operácií
3. Spôsob vyhodnocovania dotazu
4. Spôsob vyhodnocovania dotazu v distribuovanej databáze

5. Návrh tabuliek - tejto časti sme sa už venovali

9.1. Optimalizácia fyzických štruktúr

Táto kapitola sa možno objaví v niektorej z ďalších verzií. Pre bližsie informácie viď [5].

9.2. Optimalizácia realizácie základných relačných operácií

Projekcia: Projekcia sa realizuje jednoducho. Problémom je, že môžu vzniknúť totožné riadky. Ich odstránenie požaduje triedenie (O(n.log n)). Je teda potrebné zvoliť vhodnú stratégiu - kedy triediť. Dotazovacie jazyky (SQL) obsahujú prostriedky, ktoré umožňujú programátorovi triedenie ovplyvniť.

Selekcia: Optimalizácia realizácie selekcie súvisí s použitými dátovými štruktúrami, či už je to index, B-strom, hashovanie, dynamické hashovanie alebo "mriežky" pre range queries.

Zjednotenie: Zjednotenie prináša problémy len v úpravách zložitých vyhľadávacích štruktúr. Je to miesto, kde môže vznikať veľmi výrazná duplicita riadkov.

Rozdiel: Pri realizácii $p \setminus q$ je pravdepodobne najvýhodnejšie prechádzať p a vyhľadávať tieto prvky v q. Pre q sa teda hodí štruktúra podporujúca dotaz na úplnú zhodu.

θ-spojenie:

Algoritmus 19(realizácia θ-spojenia):

`INPUT:` R, S-relácie

`OUTPUT:` $R \bowtie_\theta S$

```
for all r from R do
   for all s from S do
      if θ then create(r,s);
```

Algoritmus má zložitosť $O(|R|.|S|)$, čo v najhoršom prípade (kartézsky súčin) zlepšiť nemožno. Pre prirodzené spojenie však možno zlepšiť priemerný prípad. Ak realizujeme $R(X,Y) \bowtie S(Y,Z)$, tak R aj S utriedime podľa Y a potom balancovane čítame podľa Y. Priemerný čas výpočtu je $O(|R|.log|R| + |S|.log|S| + |R| + |S| + |F|)$, kde $|F|$ je veľkosť získaného výsledku.

Špeciálnym spôsobom realizácie prirodzeného spojenia je *multiway join (multijoin)*. Touto metódou pracuje napr. QUEL. Joiny sa "backtrackujú". Pri nie príliš špeciálnych spojeniach je táto metóda pomerne dobrá.

Príklad 15: Majme relácie $R_1(X,Y), R_2(Y,Z), R_3(Z,W)$, a to $R_1 = \{(1,2),(2,4),(3,6),(4,8)\}$, $R_2 = \{(1,2),(2,3),(2,4)\}$, $R_3 = \{(1,2),(2,4),(3,6),(4,8)\}$.

Výpočet $R_1 \bowtie R_2 \bowtie R_3$ prebehne nasledovne: R_1 pošle relácii R_2 dvojicu $(1,2)$. R_2 ju spojí s dvojicou $(2,3)$ na $(1,2,3)$. Tá sa spojí v R_3 pomocou $(3,6)$ na $(1,2,3,6)$. Iná možnosť v R_3 neprichádza do úvahy, preto nasleduje back-track a vytvorenie trojice $(1,2,4)$, ktorá sa spojí na $(1,2,4,8)$. Opäť back-track, tentokrát dvojnásobný a pristúpi sa k dvojici $(3,6)$. Tú spojiť však nemožno, rovnako ako $(4,8)$. Teda $R_1 \bowtie R_2 \bowtie R_3 = \{(1,2,3,6),(1,2,4,8)\}$.

9.3. Optimalizácia relačných dotazov

Relačný výraz podobne ako aritmetický možno reprezentovať binárnym (nie striktne) stromom. Výpočet prebieha potom tak, že najprv sa zrealizujú operácie vo všetkých synoch a až potom v otcovi. Je možné vyhľadávať a tým optimalizovať výpočet elimináciou spoločných podvýrazov (spoločný podvýraz sa počíta len raz.) Na úpravu relačného výrazu na ekvilantný, ale "lepší", sa používajú nasledovné vlastnosti:

(1) Komutativita spojenia a kartézskeho súčinu

$$E_1 \bowtie_\theta E_2 \equiv E_2 \bowtie_\theta E_1$$
$$E_1 \bowtie E_2 \equiv E_2 \bowtie E_1$$
$$E_1 \times E_2 \equiv E_2 \times E_1$$

(2) Asociativita spojenia a kartézskeho súčinu:

$$(E_1 \bowtie_\theta E_2) \bowtie_\theta E_3 \equiv E_1 \bowtie_\theta (E_2 \bowtie_\theta E_3)$$
$$(E_1 \bowtie E_2) \bowtie E_3 \equiv E_1 \bowtie (E_2 \bowtie E_3)$$
$$(E_1 \times E_2) \times E_3 \equiv E_1 \times (E_2 \times E_3)$$

(3) Kaskáda projekcií:

$$\pi_{A_1,\dots,A_n}(\pi_{B_1,\dots,B_n}(E)) \equiv \pi_{A_1,\dots,A_n}(E)$$

(4) Kaskáda selekcií:

$$\sigma_{F_1}(\sigma_{F_2}(E)) \equiv \sigma_{F_1 \wedge F_2}(E)$$
$$\sigma_{F_1}(\sigma_{F_2}(E)) \equiv \sigma_{F_2}(\sigma_{F_1}(E))$$

(5) Komutativita selekcií a projekcií: Za predpokladu, že F obsahuje iba atribúty $A_1,\dots,A_n$:

$$\pi_{A_1,\dots A_n}(\sigma_F(E)) \equiv \sigma_F(\pi_{A_1,\dots,A_n}(E))$$

Všeobecnejšie, ak F navyše obashuje atribúty $B_1,\dots,B_n$, ktoré nie sú medzi $A_1,\dots,A_n$:

$$\pi_{A_1,\dots A_n}(\sigma_F(E)) \equiv \pi_{A_1,\dots A_n}\big(\sigma_F(\pi_{A_1,\dots,A_n,B_1,\dots,B_n}(E))\big)$$

(6) Komutativita selekcie a kartézskeho súčinu: Ak všetky atribúty v F sú atribútmi v E_1:

$$\sigma_F(E_1 \times E_2) \equiv \sigma_F(E_1) \times E_2$$

(7) Komutativita selekcie a zjednotenia:

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

(8) Komutativita selekcie a rozdielu:

$$\sigma_F(E_1 \setminus E_2) \equiv \sigma_F(E_1) \setminus \sigma_F(E_2)$$

(9) Komutativita selekcie a prirodzeného spojenia: Ak F obsahuje iba atribúty, ktoré sú obsiahnuté aj v E_1 aj E_2, tak

$$\sigma_F(E_1 \bowtie E_2) \equiv \sigma_F(E_1) \bowtie \sigma_F(E_2)$$

(10) Komutativita projekcie a kartézskeho súčinu: Ak spomedzi atribútov $A_1,\dots,A_n$ sú v E_1 $B_1,\dots,B_m$ a v E_2 $C_1,\dots,C_k,(B_i \neq C_j)$, tak

$$\pi_{A_1,\ldots A_n}(E_1 \times E_n) \equiv \pi_{B_1,\ldots,B_m}(E_1) \times \pi_{C_1,\ldots,C_k}(E_2)$$

(11) Komutativita projecie a zjednotenia:

$$\pi_{A_1,\ldots,A_n}(E_1 \cup E_2) \equiv \pi_{A_1,\ldots,A_n}(E_1) \cup \pi_{A_1,\ldots,A_n}(E_2)$$

Algoritmus 20(optimalizácie relačného výrazu):

`INPUT:` Relačný výraz

`OUTPUT:` Program pre výpočet dotazu

1. Použitím pravidla (4) rozložiť každú selekciu $\sigma_{F_1 \wedge \cdots \wedge F_n}(E)$ na kaskádu $\sigma_{F_1}(\ldots(\sigma_{F_n}(E))\ldots)$.
2. Pre každú selekciu použitím pravidiel (4)-(9) presunúť selekcie v strome tak nízko, ako je to možné.
3. Pre každú projekciu použitím (3), (10), (11) a zovšeobecneného (5) presunúť projekcie v strome tak nízko, ako je to možné. Projekcie, ktoré projektujú na všetky atribúty treba odstrániť.
4. Použitím (3)-(5) kombinovať kaskády selekcií a projekcií do jedinej selekcie, jedinej projekcie alebo selekcie nasledovanej projekciou.
5. Skombinovať vnútorné uzly stromu do skupín, ktorých centrom sú binárne operácie $\times, \cup, \setminus$.
6. Vytvoriť program postupne vyhodnocujúci jednotlivé skupiny.

9.3.1. Optimalizácia pomocou hypergrafov

Túto metódu používa QUEL. Optimalizujú sa ňou dotazy typu $\sigma_{F_1 \wedge \cdots \wedge F_n}(R_1 \times \cdots \times R_k)$, teda $\sigma\times$-dotazy. Dotaz sa reprezentuje hypergrafom: Jeho vrcholmi sú v použitých relačných schémach existujúce atribúty. Plnou hyperhranou sú spojené atribúty patriace tej istej relačnej schéme. Čiarkovanou hyperhranou sú spojené atribúty patriace tomu istému F_i, okrem F_i, ktoré sú tvaru $X = Y$ a $X = a$, kde X, Y sú atribúty a a je konštanta. Pre každú rovnosť $A = B$, ktorú možno odvodiť z rovností typu $X = Y$ obsiahnutými medzi F_i, vrcholy A, B stotožníme a novovzniknutý vrchol pomenujeme A alebo B ľubovoľne. Pre každé F_i tvaru $A = a$ pripojíme konštantu a k atribútu A obyčajnou hranou.

Algoritmus 21(optimalizácie redukciou hypergrafu):

`INPUT:` dotaz reprezentovaný hypergafom

`OUTPUT:` Funkcia PROG($\mathcal{G}$), ktorá predstavuje podstatnú časť optimálneho vyhodnotenia dotazu reprezentovaného hypergrafom $\mathcal{G}$ a výsledok uloží do RESULT($\mathcal{G}$)

1. Ak hypergraf $\mathcal{G}$ má disjunktné komponenty $\mathcal{H}_1, \ldots, \mathcal{H}_k, k \geq 2$, tak PROG($\mathcal{G}$) je

   PROG($\mathcal{H}_1$); ... ;PROG($\mathcal{H}_k$);
   RESULT($\mathcal{G}$):=RESULT($\mathcal{H}_1$)$\times$ ... $\times$ RESULT($\mathcal{H}_k$);

2. Ak z hypergrafu $\mathcal{G}$ odstránime prerušovanú hyperhranu E (vrcholy sa neodstraňujú), pričom dostaneme hypergraf $\mathcal{H}$, tak PROG($\mathcal{G}$) je

   PROG($\mathcal{H}$);
   RESULT($\mathcal{G}$):=$\sigma_{C(E)}$(RESULT($\mathcal{H}$);

3. Ak z hypergrafu $\mathcal{G}$ odstránime plnú hyperhranu E a dostaneme pritom hypergrafy $\mathcal{H}_1, \ldots,$ $\mathcal{H}_k$, $k \geq 1$, tak PROG($\mathcal{G}$) je

   `for each relation hyperedge F that intersects E do`
   R(F):=R(F)$\ltimes$R(E);
   PROG($\mathcal{H}_1$); ... ;PROG($\mathcal{H}_k$);
   RESULT($\mathcal{G}$):=R(E) $\bowtie$ RESULT($\mathcal{H}_1$) $\bowtie \cdots \bowtie$ RESULT($\mathcal{H}_k$);

   Program výpočtu dostaneme tak, že najprv zrealizujeme selekcie predstavujúce test na konštantu a reprezentované obyčajnými hranami a potom vykonáme PROG($\mathcal{G}$). V skutočnosti nejde o algoritmus, pretože poradie krokov 2, 3 nie je jednoznačne špecifikované a môže byť vykonané opakovane ľubovoľne. Dobrou heuristikou je vykonávať selekcie čo najskôr, t.j. (pretože výpočet je rekurzívny) dávať prednosť kroku 3.

9.3.2. Optimalizácia pomocou acyklických hypergrafov

Definícia 36(ucha): Nech $\mathcal{G}$ je hypergraf. Jeho hranu E nazveme *uchom* $\iff$ žiaden vrchol hrany E nepatrí do žiadnej inej hrany hypergrafu $\mathcal{G}$ alebo existuje taká hrana F hypergrafu $\mathcal{G}$, že vrcholy $E \setminus F$ patria iba do hrany E.

Definícia 37(acyklického hypergrafu): Hypergraf $\mathcal{G}$ nazveme *acyklickým* (v Grahamovom zmysle) $\iff$ možno ho postupným odstraňovaním uší upraviť na prázdny hypergraf (t.j. hypergraf bez hrán). Odstraňovanie hrán chápeme tak, že ich vrcholy sa neodstraňujú.

Acyklické joiny možno optimalizovať nasledovne ($\pi \bowtie$ - dotazy):

Algoritmus 22(optimalizovaného výpočtu $\pi \bowtie$ - dotazov):

`INPUT:` Hypergraf $\mathcal{G}$ pre pre výraz $\pi_X(R_1 \bowtie \cdots \bowtie R_k)$, ktorého vrcholy sú atribúty obsiahnuté v $R_1, \ldots, R_k$ a hrany zodpovedajú relačným schémam $R_1, \ldots, R_k$, pričom $\mathcal{G}$ je súvislý a acyklický.

`OUTPUT:` hodnota relácie zodpovedajúcej vstupu

`1. Zostaviť parsovací strom dotazu:`
Vrcholmi tohto stromu sú hrany hypergrafu $\mathcal{G}$. Ak v procese odstraňovania uší vedúcemu k prázdnemu hypergrafu (GYO-redukcia) je hrana E uchom, pretože vrcholy $E \setminus F$ patria iba do E, tak E je synom F.
`2. Prechádzať parsovacím stromom bottom-up, t.j. najprv navštíviť synov a až potom otca (poradie navštívenia synov je však ľubovoľné).`
`Pri návšteve uzla obsahujúceho reláciu` R`, ktorého otcom je uzol obsahujúci reláciu` S`, nahradiť` S `reláciou` $\pi_{S\cup(X\cap R)}(R \bowtie S)$.
`3. Reláciu v koreni odprojektovať na X.`

Príklad 16: Pri optimalizácii výrazu $\pi_{AG}(ABC \bowtie BF \bowtie BCD \bowtie CDE \bowtie DEG)$ možno z hypergrafu dotazu dostať tento parsovací strom (zapísaný LISPovsky ako "(otec 1.syn 2.syn 3.syn …)"):
(CDE (BCD ABC BF) DEG).

Algoritmus potom zoptimalizuje dotaz na

$$\pi_{AG}(\pi_{ACDE}(\pi_{ABCD}((ABC \bowtie BCD) \bowtie BF) \bowtie CDE) \bowtie DEG)$$

9.3.3. Optimalizácia konjunktívnych dotazov

Konjunktívne dotazy sa najčastejšie objavujú pri DATALOGovských programoch. Pravá strana pravidla predstavuje konjunkciu, ktorú často možno zjednodušiť. Napríklad ak `e` je externý predikát, tak program `f(X):-e(X,Y),e(X,Z)`, ktorý zodpovedá formule $f(X) \iff (\exists Y)e(X,Y) \wedge (\exists Z)e(X,Z)$ ekvivalentnej s $f(X) \iff (\exists Y)e(X,Y)$, možno zjednodušiť na `f(X):-e(X,Y)`.

Zjednodušovanie však môže byť podstatne komplikovanejšie.

Príklad 17: Program

$\mathtt{f}_p\mathtt{(X,Y):-e(X,Y).}$
$\mathtt{f}_p\mathtt{(X,Y):-e(Y,X),f}_p\mathtt{(X,Z).}$

možno rozvinutím 2. pravidla zjednodušiť na

$\mathtt{f}_p\mathtt{(X,Y):-e(X,Y).}$
$\mathtt{f}_p\mathtt{(X,Y):-e(Y,X),e(X,Z).}$

čo je nerekurzívny (!) program.

Z hľadiska relačnej algebry predstavujú DATALOGovské pravidlá $\pi\sigma \bowtie$ - dotazy (π je projekcia na ľavú stranu, σ len s podmienkami $X = Y$ alebo $X = a$, ktoré sa môžu vyskytnúť na pravej strane a $\bowtie$

prestavuje čiarku z pravej strany). Popíšeme tablovú metódu optimalizácie dotazov, ktorá je vhodná aj na principiálne zložitejšie dotazy. V prípade $\pi\sigma \bowtie$ - dotazov je však zvlášť jednoduchá:

Dotaz nad reláciou budeme reprezentovať **tablom**. Celá metóda je schopná pracovať len s jedinou reláciou. Využijeme teda predpoklad univerzálnej relácie, že totiž existuje univerzálna relácia u obsahujúca všetky použité atribúty, pričom jednotlivé relácie sú jej projekciami. Metóda potom pracuje s jedinou reláciou u. Tablo je tabuľka s (nepovinne - ak je to jasné z kontextu) pomenovanými stĺpcami. Stĺpec zodpovedá atribútu a nesie jeho meno. Prvý riadok obshuje tieto mená a na nič ďalšie sa nepoužíva. Poslednou operáciou dotazu býva najčastejšie projekcia (nevýznamné atribúty sa odprojektujú). Druhý riadok tabla, nazývaný hlavička, obsahuje názvy premenných pre *významné* stĺpce (ktoré sa vo výslednej relácii objavia) a prázdne miesta pre ostatné *nevýznamné* stĺpce. Premenné vyskytujúce sa v hlavičke sa nazývajú *významné*, ostatné sú *nevýznamné*. Prázdne miesto v tabuľke sa používa namiesto nevýznamnej premennej, ktorá sa vyskytuje len na tomto jedinom mieste (je rôzna aj od nevýznamných premenných, ktoré sú na iných prázdnych miestach). Ostatné riadky predstavujú podciele (subgoals) relácie vstupujúce do prirodzeného spojenia, resp. pri konjunktívnych dotazoch (čo je ekvivalentné) do konjunkcie. Selekcia na rovnosť na konštantu sa vyjadruje umiestnením tejto konštanty do riadku, podobne rovnosť dvoch atribútov sa prejaví umiestnením rovnakých premenných do ich stĺpcov.

Príklad 18: Dotaz pre program

```
f(X):-u(a,X,Y),
```

ktorý zodpovedá výrazu $f := \pi_X \sigma_{A=a} u(A, X, Y)$ možno reprezentovať tablom

E	D	M
	X	
a	X	Y

kde E(employee), D(department), M(master) sú názvy atribútov relácie u.

Príklad 19: Dotaz pre program

```
f(X):-u(a,X,Y),u(W,X,Z),
```

ktorý zodpovedá výrazu $f := \pi_X((\sigma_{A=a} u(A, X, Y)) \bowtie u(W, X, Z))$
možno reprezentovať tablom

E	D	M
	X	
a	X	Y
W	X	Z

Algoritmus 23(konštrukcie tabla pre $\pi\sigma \bowtie$ - dotaz):

`INPUT:` $\pi\sigma \bowtie$ - dotaz prípúšťajúci selekciu len na rovnosť

`OUTPUT:` tablo ekvivalentné vstupnému dotazu

Báza: Tablo pre relačný výraz $R(A_1, \ldots, A_n)$ má hlavičku a jediný riadok, ktoré sú identické. Obsahujú významné premenné pre atribúty $A_1, \ldots, A_n$ a prázdne miesta inde.

Prípad 1: Tablo T$'$ pre relačný výraz $\pi_{A_1,\ldots,A_n}(E)$ zostrojíme takto: Ak T je tablo pre E, tak T$'$ dostaneme tak, že v hlavičke nahradíme premenné, ktorých atribúty nie sú medzi $A_1, \ldots, A_n$, medzerami.

Prípad 2: Tablo T$'$ pre relačný výraz $\sigma_F(E)$ zostrojíme takto: Nech T je tablo pre E. Ak F je tvaru $A = c$, kde c je konštanta, tak T$'$ dostaneme tak, že všetky výskyty významnej premennej A nahradíme konštantou c. Ak F je tvaru $A = B$, kde A, B sú významné premenné, tak T$'$ dostaneme tak, že všetky výskyty jednej z nich nahradíme druhou z nich (resp. všetky výskyty A, B nahradíme novou premennou C).

Prípad 3: Tablo T pre relačný výraz $E_1 \bowtie E_2$ zostrojíme takto: Nech T$_1$, T$_2$ sú tablá pre E_1, E_2. Bez ujmy na všeobecnosti predpokladajme, že ak majú obe tieto tablá pre atribút A významnú premennú v hlavičke, tak je to tá istá premenná, inak však ďalšie spoločné premenné nemajú. Potom T má riadky z T$_1$ a T$_2$ a hlavičku určenú nasledovne:

1. Ak majú obe tablá T$_1$, T$_2$ pre istý stĺpec blank (prázdnu pozíciu), tak ho má aj T.
2. Ak jedno z T$_1$, T$_2$ má pre istý stĺpec blank, tak T má na tejto pozícii to, čo druhé z T$_1$, T$_2$.

3. Ak obe T_1, T_2 majú pre istý stĺpec premennú, musí to byť podľa predpokladu tá istá a tak ju má aj T.
4. Ak jedno z T_1, T_2 má pre istý stĺpec konštantu a a druhé premennú X, treba nahradiť všetky výskyty X na a. T má pre tento stĺpec v hlavičke a.
5. Ak obe z T_1, T_2 majú pre istý stĺpec tú istú konštantu a, má ju aj T.
6. Ak obe z T_1, T_2 majú pre istý stĺpec rôzne konštanty, join je prázdny, čo vyjadríme tablom s prázdnou hlavičkou (takéto tablo bude vyjadrovať prázdnu reláciu definitoricky).

Definícia 38(vnorenia): : Nech Var_1, Var_2 sú množiny premenných a $Const_1, Const_2$ množiny symbolov konštánt. Zobrazenie $h : (Var_1 \cup Const_1) \rightarrow (Var_2 \cup Const_2)$ také, že $(\forall a \in Const_1) h(a) = a$ nazývame *vnorenie* (containment mapping).

Veta 13: Tablá T_1, T_2 sú ekvivalentné $\iff$ existuje vnorenie h_1, ktoré mapuje symboly T_1 do symbolov T_2, hlavičku T_1 na hlavičku T_2 a každý riadok T_1 zobrazí na nejaký riadok T_2 a zároveň existuje vnorenie h_2, ktoré mapuje symboly T_2 do symbolov T_1, hlavičku T_2 na hlavičku T_1 a každý riadok T_2 zobrazí na nejaký riadok T_1.

Príklad 20: V predchádzajúcicj dvoch príkladoch uvedené tablá T_1, T_2 sú ekvivalentné. Vnorenie $h_1 : T_1 \rightarrow T_2$ je identita a vnorenie $h_2 : T_2 \rightarrow T_1$ je určené $h_2(a) = h_2(W) = a, h_2(X) = X, h_2(Y) = h_2(Z) = Y$.

Podstatou optimalizácie je zostrojenie "jednoduchšieho" ekvivalentného tabla. Špeciálnym prípadom je vynechávanie nepotrebných riadkov.

9.4. Optimalizácia v rámci distribuovaného databázového systému

Najväčšou záťažou pre distribuovaný systém je komunikácia po linkách. Odhliadnuc od chybových stavov, globálna optimalizácia môže viesť k lokálnemu zhoršeniu (čo môže spôsobiť kritické predĺženie doby odozvy systému). Najnáročnejšou operáciou na komunikáciu je θ-spojenie, či prirodzené spojenie, pri ktorom sú operandy umiestnené v rôznych uzloch. Stratégiu presunu dát treba dobre uvážiť. Môže sa totiž stať, že po náročnej realizácii $\bowtie$ bude ešte treba presúvať výsledok. Veľký význam nadobudne aj čo najskoršie realizovanie projekcií. Na urýchlenie výpočtu prirodzeného spojenia sa používa technika semijoinov:

Ak sa v uzle **A** nachádza relácia R(X,Y) a v uzle **B** S(Y,Z), komunikácia sa uskutoční nasledovne: **A** pošle **B** $\pi_Y R(X,Y)$, teda hodnoty, cez ktoré sa bude spájať. Tieto údaje postačujú na určenie riadkov z S, ktoré budú spojené. **B** teda pošle späť $S \ltimes R = (\pi_Y R(X,Y)) \bowtie S(Y,Z)$. **A** dokončí spojenie a dostane $R \bowtie S = R \bowtie (S \ltimes R)$. Výsledok je teda v **A**. Samozrejme **A** a **B** si mohli vymeniť úlohy. Závisí to od toho, kde sa má nachádzať výsledok a koľko času je potrebné na transakcie v jednotlivých prípadoch. Ide teda vlastne o metódu podobnú multijoinu, ale "urýchľovanú".

10. Chase procedure na test príslušnosti závislostí

Ide o metódu formálne podobnú tablovej metóde na optimalizáciu konjunktívnych dotazov. Ku každému druhu skúmaných závislostí možno zostaviť popisujúcu formulu. Napr. v relačnej schéme R(A,B,C,D) pre $A \twoheadrightarrow B$ je to

$$(\forall a)(\forall b_1)(\forall b_2)(\forall c_1)(\forall c_2)(\forall d_1)(\forall d_2)\big((R(a,b_1,c_1,d_1) \wedge R(a,b_2,c_2,d_2)) \implies R(a,b_2,c_1,d_1)\big)$$

To vedie k pojmu zovšeobecnenej závislosti zadanej pomocou nejakej logickej formuly. Skutočnosť popisovanú formulou možno zjednodušene zachytiť tabuľkou - **tablom** s významom: Ak relácia obsahuje n-tice nad čiarou, tak obsahuje aj n-tice pod čiarou. To predstavuje symbolickú manipuláciu. Ak nejaké tablo obsahuje riadky zodpovedajúce predpokladom závislosti Z, ktorá v relačnej schéme platí, možno pripísať riadky predstavujúce záver pravidla Z. Pre $A \twoheadrightarrow B$, $A \twoheadrightarrow B|C$ je to postupne

$$\begin{array}{cccc} \mathtt{a} & \mathtt{b}_1 & \mathtt{c}_1 & \mathtt{d}_1 \\ \mathtt{a} & \mathtt{b}_2 & \mathtt{c}_2 & \mathtt{d}_2 \\ \hline \mathtt{a} & \mathtt{b}_2 & \mathtt{c}_1 & \mathtt{d}_1 \end{array} \qquad\qquad \begin{array}{cccc} \mathtt{a} & \mathtt{b}_1 & \mathtt{c}_1 & \mathtt{d}_1 \\ \mathtt{a} & \mathtt{b}_2 & \mathtt{c}_2 & \mathtt{d}_2 \\ \hline \mathtt{a} & \mathtt{b}_2 & \mathtt{c}_1 & \mathtt{d}_3 \end{array}$$

kde d_3 je nová premenná. Podmienka $A \rightarrow B$ je špeciálna a nie je týmto spôsobom jednoducho vyjadriteľná, pretože jej záverom je $b_1 = b_2$. Zodpovedajúca symbolická manipulácia je: ak existujú dva riadky, ktoré majú rovnakú symbol pre A, tak symboly pre B treba stotožniť. Pre relácie $R_1(A,B,C), R_2(B,C,D)$ je podmienka bezstratovosti spojenia $\bowtie(R_1,R_2)$ vyjadrená tablom

$$\begin{array}{cccc} \mathtt{a}_1 & \mathtt{b} & \mathtt{c} & \mathtt{d}_1 \\ \mathtt{a}_2 & \mathtt{b} & \mathtt{c} & \mathtt{d}_2 \\ \hline \mathtt{a}_1 & \mathtt{b} & \mathtt{c} & \mathtt{d}_2 \end{array}$$

Odvodiť $\bowtie(R_1,R_2)$ značí dokázať bezstratovosť spojenia $R_1 \bowtie R_2$.

Pri odvodzovaní zovšeobecnenej závislosti Z z množiny zovšeobecnených závislostí $\mathcal{F}$ postupujeme tak, že na tablo T, ktoré obsahuje len predpoklady závislosti Z aplikujeme symbolické manipulácie vyplývajúce z významu tabiel pre závislosti v $\mathcal{F}$ (ak obsahuje tablo riadky nad čiarou, obsahuje aj riadky pod čiarou, a preto ich možno do T pripísať). V prípade odvodenia rovnosti $a = b$ nahradíme všetky výskyty a v T za b (resp. naopak resp. a aj b nahradíme úplne novým symbolom c).

Príklad 21: Dokážte, že $\{AB \rightarrow C, A \rightarrow B\} \vdash A \rightarrow C$.

Dôkaz: $\triangleright$ Tablo T obsahuje najprv predpoklady dokazovaného $A \rightarrow C$:

$$\begin{array}{ccc} \mathtt{a} & \mathtt{b}_1 & \mathtt{c}_1 \\ \mathtt{a} & \mathtt{b}_2 & \mathtt{c}_2 \\ \hline \end{array}$$

Cieľom je dokázať, že $c_1 = c_2$. S využitím symbolickej manipulácie pre $A \rightarrow B$, pretože riadky sa zhodujú v A, dostaneme, že $b_1 = b_2$ a T upravíme tak, že b_1 nahradíme za b_2. Dostaneme tak:

$$\begin{array}{ccc} \mathtt{a} & \mathtt{b}_1 & \mathtt{c}_1 \\ \mathtt{a} & \mathtt{b}_1 & \mathtt{c}_2 \\ \hline \end{array}$$

Pretože sa teraz riadky zhodujú v AB použitím posledného pravidla dostáveme, že $c_1 = c_2$ Q.E.D. $\triangleleft$

Príklad 22: Dokážte, že $\{B \twoheadrightarrow A\} \vdash \bowtie(AB, BC)$.

Dôkaz: $\triangleright$ Postupne dostávame:

$$\begin{array}{ccc} \mathtt{a}_1 & \mathtt{b} & \mathtt{c}_1 \\ \mathtt{a}_2 & \mathtt{b} & \mathtt{c}_2 \\ \hline \end{array} \qquad\qquad \begin{array}{ccc} \mathtt{a}_1 & \mathtt{b} & \mathtt{c}_1 \\ \mathtt{a}_2 & \mathtt{b} & \mathtt{c}_2 \\ \hline \mathtt{a}_1 & \mathtt{b} & \mathtt{c}_2 \end{array}$$

Q.E.D. $\triangleleft$

Tvrdenie 3: Ak $\mathcal{F}$ neobsahuje vnorené multizávislosti, procedúra sa zastaví (teda ide o algoritmus). Ak $\mathcal{F}$ obsahuje vnorené multizávislosti, zastavenie nie je zaručené.

Dôkaz: $\triangleright$ Bez účasti vnorených multizávislostí nepribúdajú žiadne nové symboly, a preto po konečnom počte krokov už nebude možné žiaden ďalší riadok pridať. Ak vzniknuté tablo nedokazuje Z, tak Z odvodiť nemožno. V prípade účasti vnorených multizávislotí nové symboly pribúdajú. $\triangleleft$

Literatúra

[1] Jeffrey D. Ullman: Principles of Database and Knowledge-base Systems, Volume I, II, *Computer Science Press 1988*

[2] Jaroslav Pokorný: Databázové systémy a jejich užití v informačních systémech

[3] Jaroslav Pokorný: Databázové systémy

[4] Ľubor Šešera, Aleš Mičkovský: Objektovo-orientovaná tvorba systémov a jazyk C++, *Perfekt 1994*
[5] Prednášky zo Štruktúr súborov (Kravčík) a Algoritmov a dátových štruktúr (Procházka)
[6] Igor Prívara: Matematická teória programov, prednáška
[7] D.E.Knuth: The Art of Computer Programming,Vol. 3 :Searching and Sorting