Jaroslav Pokorný

Techniky zpracování dat v databázích

Zápisy z přednášky zpracoval:
Jan Šerák

24. dubna 1995

Obsah

Seznam obrázků

1 Úvodní exkurs do problematiky databází

1.1 Pojem databáze

Dříve než se budeme zabývat databázemi *naostro*, pokusme se definovat pojem databáze.

Japonský zákon o autorsko – právní ochraně z roku 1986 definuje databázi jako „soubor informací[1], tj. znaků, čísel, diagramů apod., jehož systematická struktura umožňuje, aby tyto informace mohly být vyhledávány pomocí počítače."

V roce 1988 pan *Bancilhan* šel na definici databáze *od lesa.* Definoval databázi *implicitně* těmito slovy: „*databázová technologie* se zabývá řízením velkého množství[2] *persistentních, spolehlivých* a *sdílených* dat."

Co je tím myšleno?

- „Velkým množstvím dat" se rozumí fakt, že k jejich zpracování nestačí použít vnitřní paměť.
- *Persistencí* se rozumí nezávislost dat na programu, který je zpracovává. Data přetrvávájí od jednoho zpracování k následujícím.
- *Spolehlivostí* se rozumí fakt, že lze po chybě počítače (výpadek napájení, zhroucení systému, ...) tato data zrekonstruovat.
- A konečně *sdílená* data jsou data, která jsou přístupná více uživatelům *současně.*

Na tomto místě již nedočkavý čtenář očekává *naši* definici databáze. Z pragmatických důvodů[3] se elegantně a velikým obloukem exaktní definici databáze vyhneme a budeme předpokládat, že laskavý čtenář se již s databázemi setkal a čtenář, který slyší o takových věcech poprvé, se s nějakou databází ihned seznámí nebo bezprostředně po dočtení této věty použije tento výtisk zcela prozaičtěji než ke čtení.

1.2 Základní témata databázové technologie

Při promýšlení problematiky databází musíme počítat, že narazíme na *témata,* která hraničí s jinými oblastmi informatiky, jako jsou operační systémy, distribuované systémy, programovací jazyky, logika, modelování apod. Jsou to zejména:

- organizace dat, metody přístupu k datům,
- transakční zpracování,
- zotavení z chyb,
- paralelní přístup,
- databázové jazyky,
- optimalizace,
- konzistence dat,
- projektování dat.

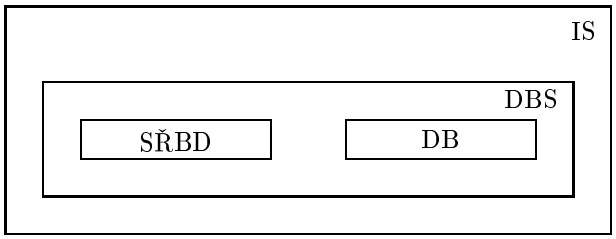
Nyní se na chvíli ještě vraťme k předchozímu odstavci. Abychom nepřišli k pojmu *databáze* tak lacino a abychom předešli tomu, že si na dalších stranách přestaneme rozumět, popišme si, co budeme rozumět pod základními pojmy v oblasti databází, bez toho, abychom je definovali. Sledujme přitom obrázek 1.

SŘBD je zkratka od *Systém Řízení Báze Dat,* což je skoro již archaické pojmenování toho, čemu se dnes častěji říká *databázový stroj,* nebo *database engine.* Jsou to rutinky, které dokáží manipulovat s daty, jež reprezentuje označení

[1] Čtenář, který se rozhodl tento úvodní exkurs také přečíst, si jistě povšimnul, že už v tomto slově definice databáze jaksi *nesedí.* Žádná databáze totiž neobsahuje *informace.* Jediné, co databáze obsahuje jsou *data,* která potom uživatel nebo nějaký databázový software interpretuje jako informace.

[2] I tato definice nám *nesedí.* Jedná se o tuto specifikaci „velkého množství". V době, kdy tato definice vznikla znamenalo velké množství dat více než 1MB dat. Dnes však takové množství můžeme několikrát umístit do operační paměti, takže zpracování takového množství dat není pro nás nijak zajímavé. Naslouchejme tedy panu *Bancilhanovi* dále, třeba to bude ještě poučné.

[3] Jinými slovy, abychom neutržili obdobnou ostudu, jako japonský zákonodárce, případně pan Bancilhan.



Obrázek 1: Znázornění základních pojmů problematiky databází

DB – databáze. Při vědomí jistých nepřesností můžeme konstatovat, že SŘBD a DB tvoří společně *databázový systém – DBS.* Poslední zkratka na obrázku – IS – znamená informační systém, jejichž problematika přesahuje rámec této přednášky.

Základní paradigma databázové technologie bylo stanoveno v době, kdy se databáze začaly v podobě prvních ideí oběvovat v myslitelských hlavách, tj. v průběhu 60. let. Jeho myšlenka se dá shrnout do věty: Existence dat v databázích je nezávislá na programech, které s nimi pracují. Při posouzení hloubky tohoto paradigmatu si musíme uvědomit, že do této doby byla data pevně svázána s programem, jediná možnost, jak je uchovávat, spočívala v jejich uložení na pásku před ukončením programu a jejich načtení bezprostředně před zahájením vlastní činnosti programu. Jak jsou tato data uložena a jak je tedy číst, věděl pouze autor daného programu.

V souvislosti se vznikem databází přichází na svět tyto pojmy:

- *Jazyk pro definici dat* (JDD, častěji (anglofonně) DDL), který do jisté míry abstrahuje[4] od fyzického uložení dat a někdy úplně odděluje *logické* schéma databáze od jeho *fyzického* schématu. Jazyk DDL je určen k vytváření tzv. *slovníku (katalogu) dat.*

- *Jazyk pro manipulaci s daty* (JMD, častěji DML) je jazyk, který slouží k manipulaci s daty a je konstruován na základě nějakého *dotazovacího jazyka.*

- *Databázový model* je kolekce pojmů (konstruktů) podporucících DDL a DML. Tento pojem je poněkud nešťastný, protože výsledkem modelování není model, nýbrž schéma[5].

- *Subschéma* je jakýsi pohled[6] na část schématu.

Abychom ještě lépe porozuměli uvedeným databázovým pojmům, podívejme se na následující tabulku.

Úroveň	**Programování**	**Databáze**
teoretická	teorie programovacích jazyků	teorie databázových modelů
struktury dat	programovací jazyk	databázový model
typy dat	program	schéma a operace nad daty
instance	stav programu	databáze

Na závěr úvodního povídání o databázích si ještě povězme na jakých třech úrovních abstrakce se data specifikují. Jsou to (v pořadí od nejnižší):

- fyzická úroveň

- konceptuální úroveň

[4] Jako příklad nám může posloužit jazyk SQL. Tento jazyk na první pohled zcela abstrahuje od fyzického uložení dat. Jenže je to pouze zdání. Fakt, že existují v SQL příkazy `CREATE INDEX` a `DROP INDEX` svědčí o tom, že uživatel SQL může ovlivnit zpracování dat a potažmo i fyzické uložení dat.

[5] Ale v tomto je čeština typická, že *vojenský inženýr* nemusí být ještě *inženýr* a *milosrdná sestra* nemusí být zdaleka *milosrdná* ba dokonce ani *sestra.*

[6] V SQL je subschéma skutečně nazýváno pohled, view.

- logická úroveň

O fyzické a logické úrovni jsme se již zmiňovali. *Konceptuální úroveň* abstrakce dat je úroveň zhruba uprostřed mezi předešlými dvěmi a spočívá v návrhu vztahů mezi daty, jejich konzistencí apod. Prostředky, které tuto úroveň podporují, jsou zejména *entitové modely,* které jsou nejrozšířenější, ale mnohé další.

2 Relační model dat

2.1 Relační model

S relačním modelem přišel v roce 1970 pan *Codd*. Relační model dat je formální abstrakcí nejjednodušších souborů. *Databázové relace* (dále jen relace) budou pro nás znamenat matematické relace[7], které budou mít pojmenované komponenty.

Terminologie, kterou budeme používat, obsahuje pojmy *relace, jméno atributu, doména atributu, atribut, jméno relace, schéma relace, n-tice* a *hodnota atributu* a je uvedena v [4]. Pro úplnost jen uveďme zevrubný přehled:

- *Relace* je (jak jsme již uvedli) matematická relace s pojmenovanými komponentami. Tyto komponenty budeme nazývat *atributy* a jejich jména budou *jména atributů*. Dále se na atributy budeme odvolávat jejich jmény.

  Příklad 2.1

  Atributy relace `STUDENT(F_NAME:'Josef',L_NAME:'Novák',YEAR:5)` jsou jméno, příjmení a ročník, jména atributů jsou `F_NAME, L_NAME` a `YEAR`.

- *Doména atributu* je množina hodnot, kterých může atribut nabývat. Např. doména atributu `F_NAME` z příkladu 2.1 je množina všech řetězců, jejichž délka nepřesahuje 20 znaků, doménou atributu `YEAR` jsou celá čísla menší než 10. *Hodnota atributu* `YEAR` v relaci z příkladu 2.1 je 5.

- *Schéma relace* se rozumí zápis $R(A_1 : D_1, \ldots, A_n : D_n)$, kde R je *jméno relace,* A_i jsou jména atributů a D_i jsou domény atributů pro $i \in \{1, \ldots, n)$.

- *n-tice* je prvek relace.

- *Schéma relační databáze* je uspořádaná dvojice $(\mathbf{R}, \mathbf{I})$, kde $\mathbf{R}$ je množina schémat relací a $\mathbf{I}$ je množina tzv. *integritních omezení.*

- *(Přípustná) relační databáze se schématem* $(\mathbf{R}, \mathbf{I})$ je množina relací $R_1, \ldots, R_k$, jejichž všechny prvky splňují všechny integritní podmínky z množiny $\mathbf{I}$.

2.2 Příklad schématu relační databáze

Mějme schéma relační databáze pro zpracování databází knihoven. Nechť toto schéma obsahuje tato schémata relací:

```
KNIHA(ISBN:char(3),AUTOR:char(20),TITUL:char(40),ZEMĚ_VYD:char(5)),
EXEMPLÁŘ(ISBN:char(3),INV_Č:char(4),CENA:number(4,2)),
ČTENÁŘ(ČT_Č:char(5),JMÉNO:char(40),ADRESA:char(40)),
VÝPŮJČKA(INV_Č:char(4),DAT_VR:date)),
REZERVACE(ČT_Č:char(5),ISGN:char(3))
```

kde `char(x)` je množina řetězců s nejvýše `x` znaky, `date` je množina všech dat[8] a `number(x,y)` je množina desetinných čísel, která mají nejvýše `x` míst před a nejvýše `y` míst za desetinnou čárkou.

V tomto příkladu lze najít tato integritní omezení:

- Stanovení primárních klíčů.

- Knihu lze rezervovat, jen když není k dispozici (všechny exempláře této knihy jsou půjčeny). Toto omezení lze ve standardu SQL 92 definovat již na úrovni databázového stroje; u starších verzí musí tyto kontroly provádět aplikační program.

- Půjčit knihu si může pouze čtenář, „který je prvkem" relace `ČTENÁŘ`.

- Každému čtenáři lze půjčit maximálně 8 knih (exemplářů knih).

Poslední dvě omezení lze definovat i ve starších verzích standardu SQL prostředky `foreign key, references`.

[7] Čtenáři málo zběhlému v matematice připomínáme, že matematická relace je libovolná podmnožina kartézského součinu nějakých množin.

[8] Čeština je v tomto velice obtížný jazyk. Toto slovo je genitiv plurálu od slova *datum,* nikoli genitiv od slova *data.*

2.3 Proč relační model dat?

V tomto odstavci si shrňme důvody, proč je relační model dat tak převratnou věcí.

1. Abstrakce dat je nezávislá na jejich uložení. Tento fakt není až tak převratný. Další důvody jsou daleko podstatnější.
2. V relačním modelu existují silné prostředky pro manipulaci s daty, které jsou jednak *neprocedurální* (relační kalkul, DATALOG[9]) a *operace vysoké úrovně* (relační algebra).
3. Existují metody návrhu relací, které zajišťují, aby relace měly „dobré" vlastnosti, jako je nalezení *třetí normální formy*.

2.4 Relační algebra

Z jazyků pro manipulaci s daty v relačním modelu dat (RMD) budeme diskutovat zejména dotazovací prostředky. Nejdůležitější dotazovací prostředky jsou *relační algebra* a *doménový relační kalkul*. V tomto odstavci se budeme zabývat prvním z nich, druhému je věnován odstavec 2.6.

Intuitivně si relaci můžeme představit jako tabulku, v jejíchž řádcích jsou její prvky a v jejíchž sloupcích jsou hodnoty jednotlivých atributů. Tato představa však v některých detailech pokulhává, např. se zde mohou opakovat řádky a tabulka definuje pořadí řádků, relace je však množina, a tudíž obě možnosti nebere v potaz.

Definice 2.1
Nechť R a S jsou relace...

1. Nechť $B \subseteq A$. Pak *projekcí* nazveme operaci, která vytvoří relaci $R[B]$ tak, že z relace $R(A)$ „odřízne sloupce z $A - B$" (uvažujeme-li na úrovni tabulek, musí ještě zahodit ze vzniklé tabulky duplikované řádky).
2. Nechť Φ je booleovská podmínka. Pak *selekcí* nazveme operaci, která vytoří relaci $R(\Phi)$ takovou, že z relace R ponechá pouze ty prvky, které splňují Φ.
3. *Spojením* relací $R(A)$ a $S(B)$ rozumíme operaci, která vytvoří maximální relaci $R * S$ s atributy z množiny $A \cup B$ takovou, že pokud $u \in R * S$, pak $u[A] \in R(A) \wedge u[B] \in S(B)$. V řeči SQL to znamená spojení přes sloupce z množiny $A \cap B$.
4. Další množinové operace jako kartézský součin $\times$, sjednocení $\cup$, průnik $\cap$ a rozdíl $-$, přičemž se musí dbát na aritu operandů a kompatibilitu domén[10].
5. Operace Θ-*spojení* je spojení přes dva atributy, kteřé se však jinak jmenují (což je rozdíl oproti spojení). Θ-spojení relací $R(A, B, C)$ a $S(G, H)$ přes atributy $R.A$ a $S.H$ se zapíše

$$R[R.A = S.H]S$$

Jak lze již z definice vidět, mohou se některé z těchto operací definovat pomocí druhých z nich. Např. Θ-spojení lze zapsat jako selekce kartézského součinu

$$R[R.A = S.H]S \equiv (R \times S)(R.A = S.H),$$

ale také je přirozené spojení lze nadefinovat složením operací kartézského součinu, příslušné selekce a následné projekce (z důvodu zastínění duplicitních sloupců).

Definice 2.2
Relační algebra je tedy množina výrazů zkonstruovaných z **R**, operací relační algebry, závorek, konstant, relačních a logických symbolů pro tvorbu booleovských podmínek.

Věta 2.3
Minimální množinu operací relační algebry, pomocí nichž lze nadefinovat libovolný výraz relační algebry, tvoří sjednocení, kartézský součin, rozdíl, selekce a projekce.

Důkaz: Toto tvrzení necháváme čtenáři k věření a pedantickému čtenáři k ověření. □

[9] Databázovému systému DATALOG je věnována samostatná kapitola 3.
[10] Ve standardu SQL 92 lze tvořit domény atributů (ovšem pouze jako podtypy standardních typů

2.5 Formalizace některých pojmů

Databázový dotaz nad relačním schématem S s odpovědí s relačním schématem T je funkce s vlastnostmi:

- definiční obor tvoří všechny databáze se schématem S,
- obor hodnot tvoří struktury databázového schématu T,
- tato funkce je částečně rekurzívní,
- data v odpovědi jsou z databáze[11],
- odpověd nezávisí na uložení dat v databázi.

Dotazovací jazyk je množina výrazů nad abecedou Σ s významovou funkcí μ, která výrazům jazyka přiřadí dotaz. Takový jazyk je většinou omezený, protože se v něm nedá vyjádřit nic, co by nemělo sémantiku (tj. to, čemu by funkce μ nedokázala přiřadit dotaz)[12]. Dotazovací jazyk je *úplný,* jestliže v něm lze vyjádřit všechny dotazy. *Vyjadřovací síla* dotazovacího jazyka je množina v něm vyjádřitelných dotazů.

Abychom se ještě na chvíli vrátili k relační algebře, poznamenejme, že existuje celkem užitečný dotaz, který nelze prostředky relační algebry vyjádřit. Mějme relaci `MANAGER(EMP,MANAGER)`, která pracovníku `EMP` přiřazuje pracovníka `MANAGER`, který je jeho vedoucím (obvykle se takto v relačních databázích ukládá strom). Zcela logický dotaz na všechny podřízené nějakého pracovníka (dotaz na tranzitivní uzávěr relace) nelze v relační algebře vyjádřit. Důvodem je, že předem neznáme maximální počet úrovní podřízených v relaci `MANAGER` je uloženo a tudíž neznáme počet selekcí, které musíme k vyjádření tranzitivního uzávěru použít[13].

Rozšíření dotazovacích jazyků Dotazovací jazyky (jak jsme právě nakousli) lze obohacovat např. o agregační funkce, jednoduchou aritmetiku, lze je začleňovat do hostitelských jazyků (jazyky 4GL, např. `Pro*C, Pro*Ada, Pro*FORTRAN` atd.) nebo rozšířit o while-cykly, tranzitivní uzávěr, pevný bod atd.

2.6 Doménový relační kalkul

Doménový relační kalkul (DRK) je druhým dotazovacím jazykem, o kterém si budeme povídat.

Definice 2.4
Doménový relační kalkul je tvořen:

1. termy, které mohou být z důvodu absence funkčních symbolu pouze proměnné a konstanty,
2. atomickými formulemi, které jsou tvořeny z termů pomocí $R_i \in \mathbf{R}$, $\leq$, $=$, $\neq$, ...
3. formulemi, které jsou tvořeny z atomických formulí pomocí kvantifikátorů $\exists$ a $\forall$ a logických spojek $\wedge$, $\vee$, $\Longrightarrow$, ...

Přesněji DRK je množina výrazů $\{x_1, \ldots, x_n \mid A(x_1, \ldots, x_n)\}$, kde A je formule DRK.

Odpověď na dotaz vyjádřený pomocí DRK je pro $k/ge1$ k-ární relace, která obsahuje všechny prvky, které se při dosazení za volné proměnné formule A vyhodnotí na $TRUE$. Např. dotaz, zda se v knihovně nachází kniha s titulem „Babička" se vyjádří takto:

$$\exists \text{ isbn } \exists \text{ autor } (\text{KNIHA(isbn,autor,'Babička')})$$

Tato formule je uzavřená a vyjadřuje tzv. *ano/ne* dotaz.

Dotaz vyjádřený formulí:

$$\text{VÝPŮJČKA(ič,'J60',dz)}$$

což je otevřená formule, vrátí relaci (ič,dz), která vyjadřuje všechny knihy, které si vypůjčil čtenář 'J60' s daty, kdy mají být vráceny.

[11] Toto tvrzení nemusí být vždy pravdou. Například v SQL agregační funkce (suma, průměr, počet, atd.) se nemusí trefit do hodnoty, která je v databázi uložena.

[12] Jako protipříklad nám může posloužit opět SQL, v němž můžeme operátory `UNION` a `INTERSECT` pouze jako vedoucí operátory výrazů, jinak je to nepřípustné už z hlediska syntaxe.

[13] Tento problém vkládá do dotazovacího jazyka prvky rekurzivnosti, které sice byly řešeny *přísernými* prostředky minimálního pevného bodu, které však zřejmě nejde efektivně implementovat. Proto je zde namístě ptát se, zda by nebylo jednodušší zavést do dotazovacího jazyka procedurální prvky (while-cyklus apod.).

2.7 Syntaktické konvence DRK

Jména atributů Je-li $R(A, B, C)$, pak místo $R(x, y, z)$ budeme pro názornost psát $R(A : x, B : y, C : z)$.

Implicitní existenční kvantifikátor Formuli DRK

$$\exists x\, R(\ldots, A : x, \ldots)$$

budeme nahrazovat formulí

$$R(\ldots, \ldots),$$

jestliže se proměnná x nemá v této formulí více výskytů.

Příklad 2.2
Zapište v DRK dotaz: Najdi isbn všech půjčených anglických knih. Řešení:

$$\{isbn \mid \exists ic(EXEMPLAR(INV\_C : ic, ISBN : isbn, ZEME\_VYD :' GB') \wedge VYPUJCKA(INV\_C : ic))\}$$

Příklad 2.3
Zapište dotaz: Najdi všechny autory, kteří mají všechny své tituly rezervovány.

$$\{a \mid \forall t(\exists i(KNIHA(ISBN : i, TITUL : t, AUTOR : a) \Longrightarrow \neg REZERV(ISBN : i)))\}$$

Příklad 2.4
Zapište dotaz: Najdi knihy rezervované všemi čtenáři, kteří mají něco rezervováno.

$$\{i \mid \forall c(REZERV(C\_CT : c) \Longrightarrow REZERV(C\_CT : c, ISBN : i))\}$$

Napíšeme-li však

$$\{i \mid \forall c(REZERV(C\_CT : c, ISBN : i))\}$$

získáme většinou $\emptyset$, což je v souladu s predikátovou logikou 1. řádu.

Hledáme-li čtenáře, které mají rezervovány všechny knihy, a nepoužijeme-li v tomto příkladě uvedeny typ formule s implikací, hrozí nám, že mám-li prázdnou relaci knih, obdržím takovým dotazem celou relaci čtenářů.

2.8 Problémy ve zpracování DRK-dotazů

1. Co když je výsledkem vyjádření dotazu v DRK nekonečná relace? Například výrazu

   $$\{y \mid \exists x(y > x \wedge R(x))\}$$

   neodpovídá žádny dotaz. Výsledná relace by totiž musela růst přes všechny meze.

2. Co když je výsledkem vyjádření dotazu v DRK konečná relace s hodnotami atributů mimo databázi? Například:

   $$\{a \mid \neg KNIHA(Titul : "Úvoddo\ldots", Autor : a)\}$$

   znamená množinu všech prvků domény atrubutu $Autor$ místo množiny autorů z knihovny.

Tyto problémy se objevují u logických operací $\neg$ a $\vee$. Řešení těchto problémů jsou dvě:

- omezení interpretace (k dané formuli přiřadíme $\wedge KNIHA(Autor : a)$
- omezení syntaxe (na tzv. bezpečné výrazy, o kterých budeme hovořit dále[14]).

[14] Taky si zde ukážeme jeden důležitý poznatek, který už objevil pan Codd.

2.9 Vztah DRK a relační algebry

Věta 2.5
Každý dotaz vyjadřitelný v relační algebře $\mathcal{A}_R$ je vyjádřitelný v DRK.

Důkaz: Důkaz povedeme vzhledem k počtu operátorů ve výrazu E:

1. Neobsahuje-li výraz E žádný operátor, pak buď $E = R$ a příslušný DRK-výraz je $\{x_1, \ldots, x_n \mid R(x_1, \ldots, x_n)\}$, nebo E je konstantní relace (katalog, číselník apod.) a příslušný DRK-výraz sestrojíme primitivně výčtem:
$$\begin{aligned}\{x_1, \ldots, x_n \mid \quad & x_1 = a_1 \wedge \cdots \wedge x_n = a_n \\ \vee \quad & x_1 = b_1 \wedge \cdots \wedge x_n = b_n \\ \vee \quad & \cdots\end{aligned}$$

2. Je-li $E = E_1 \cup E_2$, pak podle IP existuje DRK-výrazy e_1 a e_2, které jsou vyjádřením výrazů relační algebry E_1 a E_2. Hledaný DRK-výraz bude vypadat takto:
$$\{x_1, \ldots, x_n \mid e_1(x_1, \ldots, x_n) \vee e_2(x_1, \ldots, x_n)\}$$
U dalších případů už budeme jen uvádět příslušné tvary DRK-výrazů.

3. $E = E_1 - E_2$:
$$\{x_1, \ldots, x_n \mid e_1(x_1, \ldots, x_n) \wedge \neg e_2(x_1, \ldots, x_n)\}$$
Upozorněme u tohoto DRK-výrazu, že uvedený výraz, i když obsahuje negaci, je bezpečný.

4. $E = E_1[i_1, \ldots, i_k]$:
$$\{x_{i_1}, \ldots, x_{i_k} \mid \exists x_{j_1}, \ldots, x_{j_{n-k}} e_1(x_1, \ldots, x_k)\},$$
kde $(i_1, \ldots, i_k, j_1, \ldots, j_{n-k}) \in \mathbf{S}_n$).

5. $E = E_1 \times E_2$:
$$\{x_1, \ldots, x_m, x_{m+1}, \ldots, x_{m+n} \mid e_1(x_1, \ldots, x_m) \wedge e_2(x_{m+1}, \ldots, x_{m+n})\}.$$

6. $E = E_1(\varphi)$, kde φ je tvaru $A\Theta B$ nebo $A\Theta a$ a $\Theta \in \{\leq, =, \neq, \geq, \ldots\}$, pak DRK-výraz má jeden ze tvarů
$$\{x_1, \ldots, x_n \mid e_1(x_1, \ldots, x_n) \wedge x_A \Theta x_B\} \tag{1}$$
$$\{x_1, \ldots, x_n \mid e_1(x_1, \ldots, x_n) \wedge x_A \Theta a\} \tag{2}$$
Následující lemma nám zajistí převod libovolného boolovského výrazu do několika jednoduchých, které mají tvar jako φ.

□

Lemma 2.6
Je-li φ booleovský výraz pro selekci, pak k výrazu $E(\varphi)$, kde E je výraz relační algebry, existuje výraz relační algebry, který obsahuje jen jednoduché booleovské výrazy a je ekvivalentní s $E(\varphi)$.

Důkaz: Musíme prodiskutovat všechny logické spojky, které mohou jednoduché booleovské výrazy „zesložiťovat" do φ.

- Negaci $\neg$ můžeme propagovat až do nejnižší úrovně, kde se s ní vypořádáme na úrovni negovaných relačních operátorů. To znamená, že negace se dokážeme bez náhrady zbavit.
- Disjunkce $\vee$ vypadá takto: Máme-li selekci s disjunkcí, pak zřejmě platí rovnost:
$$E(\varphi_1 \vee \varphi_2) = E(\varphi_1) \cup E(\varphi_2)$$
a tedy umíme se zbavit disjunkce.
- U konjunkce $\wedge$ platí taková rovnost:
$$E(\varphi_1 \wedge \varphi_2) = (E(\varphi_1))(\varphi_2)$$
a umíme se zbavit i konjunkce.

Indukcí přes strukturu booleovského výrazu φ s použitím předchozích tří pravidel jsme dokázali, co bylo třeba.
□

2.10 Relační úplnost SQL

Jak bylo již výše uvedeno v nějaké poznámce pod čarou, SQL není relačně úplné. Důvodem je způsob, jak omezuje užití množinových operátorů. Toto však můžeme v SQL obejít způsoby, které ukazuje následující tabulka:

Operace rel. algebry	Nahrazení příkazy SQL

$T := R \cup S$	`INSERT INTO T`
	`  VALUES (SELECT * FROM R)`
	`INSERT INTO T`
	`  VALUES (SELECT * FROM S)`

$T := R - S$	`INSERT INTO T`
	`  VALUES (SELECT * FROM R)`
	`DELETE FROM T`
	`  WHERE A1,...,AN IN (SELECT * FROM S)`

$T := R \times S$	`INSERT INTO T` `  VALUES (SELECT * FROM R,S)`
$T := R(\Phi)$	Tímto výrazem je definována klauzule `WHERE`
$T := R[B]$	Tímto výrazem je definován příkaz `SELECT`

Toto obcházení množinových operací je však pouhý teoretický výsledek. Pokud bychom toto obcházeli i v praxi, ztrácíme jakoukoli možnost optimalizací dotazů.

2.11 Bezpečné výrazy

Definice 2.7

Mějme $\{x_1, \ldots, x_n \mid A(x_1, \ldots, x_n\}$, kde A je DRK-formule. Označme $\Re(A)$ množinu hodnot z relací obsažených v A a konstant obsažených v A.

Řekneme, že A je *definitní* (doménově nezávislá), jestliže pro nějaký prvek $(a_1, \ldots, a_k)$ je $A(a_1, \ldots, a_k) = TRUE$, pak $a_i \in \Re(A)$ pro $\forall i \in \{1, \ldots, k\}$. Tedy taková formule nezávisí na doménách atributů.

Ovšem již v roce 1969 vyslovil pan DiPaola následující větu:

Věta 2.8

Definitnost formule A není rozhodnutelná.

Definitnost nám tedy od nechtěných výsledků dotazů vyjádřených v DRK nepomůže. Proto si nadefinujeme tzv. bezpečné formule, které jsou definitní, ale navíc ještě syntakticky charakterizovatelné.

Definice 2.9

Řekneme, že formule A je *bezpečná*, jestliže splňuje všechny následující podmínky:

1. A má eliminovaný všeobecný kvantifikátor.
2. Je=li v A disjunkce $\vee$, pak je tvaru
$$\varphi_1(x_1, \ldots, x_s) \vee \varphi_2(x_1, \ldots, x_s)$$
a φ_1 a φ_2 mají tytéž volné proměnné.
3. Je-i ve formuli A maximální konjunkce $\wedge$, tj. $\varphi = \varphi_1 \wedge \cdots \wedge \varphi_p$, pak všechny volné proměnné musí být omezené, tj. musí splňovat jednu z podmínek:
   (a) Existuje-li φ_i taková, že není negovaná ani aritmetické porovnání, proměnná x je volná ve φ_i, pak x je omezená.
   (b) Je-li $\varphi_i \equiv x = a$ nebo $\varphi_i \equiv a = x$, pak x je omezená.
   (c) Je-li $\varphi_i \equiv x = y$ nebo $\varphi_i \equiv y = x$ a y je omezená, pak x je omezená.
4. Negaci $\neg$ lze použít jen ve tvaru 3.

Příklad 2.5
$R(x,y,z) \wedge \neg(P(x,y) \vee Q(y,z))$ je definitní (výsledky jsou z množiny $\Re(A)$), ale není bezpečná, protože porušuje podmínku 2.
$R(x,y,z) \wedge \neg P(x,y) \wedge \neg Q(y,z)$ je bezpečná.

Věta 2.10
Ke každé definitní formuli existuje ekvivalentní bezpečná formule.

Důkaz: Viz literatura. □

Bohužel však není znám algoritmus, který převod řeší. Závěrem kapitoly si uveďme tvrzení o vztahu jednontlivých druhů DRK-výrazů:

Tvrzení 2.1
Nechť A je třída všech DRK-formulí, B je třída všech definitních formulí a C je třída všech bezpečných formulí. Pak platí:

$$C \subset B \subset A.$$

2.12 Vyhodnocování a optimalizace dotazů

Při vyhodnocování výrazů se používá tohoto postupu:

- Převod do vnitřní formy, tedy z SQL do výrazu relační algebry ($\mathcal{A}$). Po té binární výraz do stromu (viz dále).
- Konverze do kanonického tvaru (algebraické optimalizace)
- Optimalizace na syntaktické úrovni
- Volba strategie vyhodnocení
- Generování kódu

2.12.1 Algebraické optimalizace

Mezi algebraické optimalizace patří všelijaká *kouzla* s výrazy relační algebry. Tato kouzla jsou definovány jakýmisi ekvivalencemi výrazů ($\approx$) a pro použití v optimalizacích je potřeba najít nějaký směr této ekvivalence, který je zrovna výhodný. Jsou to:

1. Komutativní zákony

$$\begin{aligned} E_1[\Theta]E_2 &\approx E_2[\Theta]E_1 \\ E_1 * E_2 &\approx E_2 * E_1 \\ E_1 \times E_2 &\approx E_2 \times E_1 \end{aligned}$$

2. Asociativní zákony

$$\begin{aligned} (E_1[\Theta_1]E_2)[\Theta_2]E_3 &\approx E_1[\Theta_1](E_2[\Theta_2]E_3) \\ (E_1 * E_2) * E_3 &\approx E_1 * (E_2 * E_3) \\ (E_1 \times E_2) \times E_3 &\approx E_1 \times (E_2 \times E_3) \end{aligned}$$

   Tato pravidla jsou v celku zřejmá, ale teď již přejdeme k zajímavějším kouzlům.

3. Pokud jsou všechny atributy ze selekční podmínky Φ v množině $\{A_1, \ldots, A_n\}$, pak

$$E[A_1, \ldots, A_n](\Phi) \approx E(\Phi)[A_1, \ldots, A_n]$$

   Jestliže selekční podmínka Φ obsahuje pouze atributy z $\{A_1, \ldots, A_k\}$ a pokud neobsahuje žádný atribut z množiny $\{B_1, \ldots, B_s\}$, pak

$$E[A_1, \ldots, A_k, B_1, \ldots, B_s](\Phi)[A_1, \ldots, A_k] \approx E(\Phi)[A_1, \ldots, A_k]$$

4. Jsou-li všechny atributy ze selekční podmínky Φ z relace E_1, pak

$$(E_1 \times E_2)(\Phi) \approx E_1(\Phi) \times E_2$$

5.
$$(E_1 \cup E_2)(\Phi) \approx E_1(\Phi) \cup E_2(\Phi)$$

6.
$$(E_1 - E_2)(\Phi) \approx E_1(\Phi) - E_2(\Phi)$$

7. Označme $\overline{A}$ množinu atributů, které jsou výsledkem projekce $[A_1, \ldots, A_n]$. Pokud $\overline{B} \cup \overline{C} = \overline{A}$, B_i jsou atributy pouze relace E_1 a C_i jsou atributy pouze relace E_2, pak

$$(E_1 \times E_2)[A_1, \ldots, A_n] \approx E_1[B_1, \ldots, B_k] \times E_2[C_1, \ldots, C_m]$$

8.
$$(E_1 \cup E_2)[A_1, \ldots, A_n] \approx E_1[A_1, \ldots, A_n] \cup E_2[A_1, \ldots, A_n]$$

Poznámka 2.11
SPJ-dotaz je dotaz, v němž jsou použity pouze operace selekce (S), projekce (P) a spojení (J – join). Zda lze optimalizovat SPJ-dotaz ve smyslu minimalizace počtu spojení je NP-úplný problém. Tím spíš, když používáme i další operace.

2.12.2 Heuristiky pro optimalizaci

V tomto odstavečku si uvedeme seznam heuristik, které lze při optimalizaci dotazu použít, a to setříděný podle významu (od nejvyššího).

1. Selekci provádět co neblíže k relaci. Při této úpravě se budou provádět úpravy podle pravidel: kaskády selekcí (spojení selekcí do jedné selekce: $E(\Phi)(\Psi) \succ E(\Phi \wedge \Psi)$), komutativita selekce s projekcí a kartézským součinem, distributivita nad množinovými operacemi. Tím přiblížíme selekce, které nám zmenšují objemy zpracovávaných dat, co nejvíce k listům dotazu.

2. Projekci provádět co nejblíže k relaci. Pravidla: kaskády projekcí, komutativita s kartézským součinem, distributivita nad množinovými operacemi. Tím přibližujeme projekci k listům dotazu, případně zbytečné projekce vyhazujeme.

3. Transformace kartézského součinu ($\times$) na přirozené spojení ($*$), tj. $(R \times S)(R.A > S.C) \succ R[R.A > S.C]S$, protože některé implementace databázových strojů umějí lépe pracovat s přirozenými a Θ-spojeními.

4. Posloupnosti projekcí a selekcí zpracovávat současně. Např. u dotazu $R(A > 3)[A, C]$ je lepší při vygenerování n-tice selekcí provést rovnou na ní projekce na dva atributy A a C.

   Zpracovávat přirozené spojení ($*$) a kartézský součin ($\times$) následované selekcí a projekcí opět dohromady, např. u dotazu
   $$(R * S)(R.A > 4 \wedge S.B < 6)$$
   ihned po vygenerování n-tice z přirozeného spojení na ni aplikovat selekční podmínku a rozhodnout tak ihned o generování této n-tice.

5. U dotazu $R(\varphi_1 \wedge \varphi_2 \wedge \varphi_3)$ vzít nejprve to φ_i, které zmenší relaci R co nejvíce, což zjistíme vyhodnocením selektivit těchto selekčních podmínek φ_i (blíže viz 2.12.3).

6. Ukládání společných podvýrazů (což je známé, sice poněkud s jiným cílem, z překladačů).

2.12.3 Optimalizace selektivit

Označení Pod označením $V(A, R)$ budeme rozumět počet prvků aktivní domény relace $R[A]$ (tj. počet navzájem různých prvků sloupce A v tabulce R). Počet prvků tabulky R budeme značit n_R.

Selektivita Pod pojmem selektivita budeme rozumět podíl

$$F = \frac{R(\varphi)}{n_R},$$

přičemž čím menší F, tím větší bude selektivita selekce.

Strategií vyhodnodnocování selektivit je několik. My uvedeme dvě. Jedna je definována na dané relační operátory *natvrdo.*

Operátor	F [%]
$=$	20
$>$	40
$<$	40

Již na první pohled je patrné, že – zůstaneme-li u našeho klasického příkladu s knihovnou – selekční podmínka $EXEMPLAR.CENA > 5700$ je zcela jistě daleko větší, než selekční podmínka $EXEMPLAR.CENA > 80$. Proto si zde uvedeme strategii, kterou používá Informix – OnLine.

V následující tabulce budeme pod symboly `iatribut` rozumět takový atribut, na který je vytvořen index a pod označením max_2 resp. min_2 druhý maximální resp. druhý minimální prvek aktivní domény.

Selekce	F
`R.iatribut = k`	$(V(R.iatribut, R))^{-1}$
`R.iatribut IS NULL`	$(V(R.iatribut, R))^{-1}$
`R.iatribut=S.iatribut`	$(\max(V(R.iatribut, R), V(S.iatribut, S))^{-1}$
`iatribut > k`	$(max_2 - k)/(max_2 - min_2)$
`iatribut < k`	$(k - min_2)/(max_2 - min_2)$
`atribut = výraz`	10^{-1}[15]
`atribut IS NULL`	10^{-1}
`atribut LIKE výraz`	5^{-1}
`atribut > výraz`	3^{-1}
`atribut < výraz`	3^{-1}
`EXISTS poddotaz`	1, pokud je odhad true, jinak 0
`NOT selekce`	$1 - F(selekce)$
`sel1 AND sel2`	$F(sel1) \cdot F(sel2)$
`sel1 OR sel2`	$F(sel1) + F(sel2)$ - $F(sel1) \cdot F(sel2)$

Další situace, např. `atribut IN (k1,...,kn)` se převede na případ `atribut=k1 or ... or atribut=kn`, případ `atribut` Θ `ANY poddotaz` se převede na `atribut` Θ `k1 or ... atribut` Θ `kn`, přičemž je nutné odhadnout velikost poddotazu.

V moderních DBS jsou z důvodů optimalizací vytvářeny přechodné indexy, které, pokud se *osvědčí,* se mohou stát i trvalými.

Systém R Jak vypadá optimalizace jednonduchého dotazu s více podmínkami nad jednou relací v systému R[16] si ukážeme nyní.

Předpokládejme, že některé atributy jsou indexované a známe $V(A, R)$ pro všechny atributy A s indexem. Navíc, index může být:

- *klastrovaný,* tedy $R(A = C)$ je $\approx$ v minimálním počtu bloků.
- *neklastrovaný,* tedy $R(A = C)$ je $\approx$ ve $V(A, R)$ blocích.

Lidsky (tedy příkladem) řečeno, mějme relaci s atributy J (jméno) a P (plat), která je setříděna podle jména. Pak index na atribut J je klastrovaný a index na P je neklastrovaný.

Dále předpokládejme, že n_R je počet prvků relace R a p_R je počet bloků, v nichž je R uložena.

Metoda optimalizace v systému R je taková, že se vybere jedna z osmi následujících strategií a na její výsledek se použijí ostatní podmínky jednoduchého dotazu. Výběr strategie se řídí nejmenší cenou strategie, tj. nejmenším počtem přečtených stránek.

[16] Systém R je jedním z prapůvodních relačních databázových systémů, který se v praxi ani moc nepoužíval. Na jeho základech IBM zkonstruovala svůj DB2, který dodávala jako součást softwaru svého počítače AS 400.

1. Zvol podmínku $A = C$, kde A je klastrovaný index. Cena: $\frac{p_r}{V(A,R)}$.
2. Zvol $A \Theta C$, kde $\Theta \in \{<, \leq, >, \geq\}$ a na atributu A existuje klastrovaný index. Pro $\neq$ použij pravidlo (5). Cena: $\frac{p_r}{2}$.
3. Zvol $A = C$, kde A má neklastrovaný index. Cena: $\frac{n_R}{V(A,R)}$.
4. Je-li R pouze sekvenční soubor, čte se celý. Cena: p_R.
5. Je-li R „smíchané" s jinými relacemi a existuje klastrovaný index na libovolném atributu (skupině atributů), čte se celá přes tento index. Cena: p_R.
6. Zvol $A = C$, kde na A existuje neklastrovaný index. Cena: $\frac{n_R}{2}$.
7. Pokud existuje neklastrovaný index, čte se celá přes tento index. Cena: n_R.
8. Pokud neplatí nic žádná podmínka pro uplatnění pravidel (1) – (7), pak se čtou bloky, které eventuálně obsahují relaci R. Cena: $\geq n_R$.

2.12.4 Syntaxí řízená optimalizace

Některé *písíčkové* databázové systémy, zejména je tento rys typický pro FoxPro a MS Access[17], které z důvodu držení kroku s vyspělým databázovým světem poskytují SQL. Tyto implementace SQL se však chovají tak podezřele, že např. dotaz

```
SELECT * FROM EXEMPLAR
    WHERE CENA > 40
      AND ZEME VYDANI = 'GB'
```

je méně efektivně vyhodnocen než dotaz

```
SELECT * FROM EXEMPLAR
    WHERE ZEME VYDANI = 'GB'
      AND CENA > 40
```

Takže to potom dopadá u těchto DBS tak, že člověk si musí s těmito produkty několik měsíců důkladně hrát a pozorovat jejich chování a pak teprve může psát efektivně vyhodnocované dotazy.

Když již jsme u těchto nečistých triků, uveďme si, že lze v některých systémech obejít optimalizátor. Jak na to? Vezměme si např. dotaz

```
SELECT * FROM EXEMPLAR
    WHERE (DNAKUP > '3/20/88' AND ZEME VYDANI = 'GB')
       OR ISBN = 456
```

a abstrahujme od jeho *smysluplnosti*. U některých DBS se nám může stát, že při vyhodnocování takto *složitého* dotazu vypínají optimalizátor. Proto zadáním ekvivalentního dotazu

```
SELECT * FROM EXEMPLAR
    WHERE DNAKUP > '3/20/88' AND ZEME VYDANI = 'GB'
UNION
SELECT * FROM EXEMPLAR
    WHERE ISBN = 456
```

kdy není optimalizátor vypnut, můžeme dosáhnout lepších výsledků.

[17] Jinými slovy (autor zde nechtěl být zaujatý, ale nedovedl si tuto poznámku odpustit) produkty nechvalně známé firmy Mrkvosoft (redžistrid trejd márk).

2.12.5 Statisticky řízená optimalizace

Tuto optimalizaci implementuje například DBS Ingres. Počívá v tom, že v systémovém katalogu jsou udržovány statistiky ve formě histogramů, podle nichž lze lépe odhadovat selektivity. Například exempláře knih z knihovny lze kategorizovat podle ceny po 100 takto:

CENA	\%
100	77
200	12
300	7
400	3
>400	1

2.12.6 Optimalizace v distribuovaných DBS

V distribuovaných DBS se musí provádět optimalizace nejen vzhledem k počtu přístupu na disk, ale, což je závažnější, zejména podle počtu přenosů dat mezi počítači.

3 Datalog

Než se začneme bavit o Datalogu, uveďme si příklad databáze v notaci Prolog like.

Příklad 3.1
Nechť `F(x,y)` znamená, že `x` je otec `y`, `M(x)` fakt, že `x` je muž; `S(x,y)` znamená, že `x` je sourozenec `y` a `B(x,y)`, že `x` je bratr `y`.

Pak formule

- `F(Oldřich,Pavel).`
- `F(Oldřich,Jaroslav).`
- `F(Jaroslav,Veronika).`

nazveme *extenzionální databáze (EDB)*. Extenzionální databázi jsme schopni popsat atomickými formulemi.

Mimo extenzionální databáze však můžeme ještě tvořit tato pravidla:

- `M(x) :- F(x,y).`
- `S(y,w) :- F(x,y),F(x,w).`
- `B(x,y) :- S(x,y),M(x).`

Tato pravidla tvoři *intenzionální databázi (IDB)*, neboli *virtuální relace*. Můžeme ji chápat jako program, který má spočítat relace `M`, `S` a `B`. Také zde můžeme nalézt paralelu s pohledy (views), rozdíl je však v tom, že v intenzionální databázi lze použít rekurzi.

Vyhodnocování intenzionální databáze potom probíhá buď jako tvorba modelu teorie dané pravidly intenzionální databáze nebo jako prologovské vyhodnocování po řádcích.

3.1 Minimální pevný bod

V tomto odstavci si budeme povídat takovým trošku *suchým* matematickým stylem o základě rekurze v databázích (v Datalogu).

Definice 3.1
Kompozici binárních relací R a S na doméně D definujeme takto:

$$R \circ S = \{(a,b) \mid \exists c \in D,\ (a,c) \in R \wedge (c,b) \in S\}$$

Mějme funkci f, přiřazující binární relaci R binární relaci R_i, kde R i R_i jsou nad D. Pak můžeme formulovat následující definici.

Definice 3.2
Nejmenší pevný bod rovnice $R = f(R)$, kde R je relace, je relace $R^\star$ splňující podmínky:

- $R^\star = f(R^\star)$,
- $S^\star = f(S^\star) \Longrightarrow R^\star \subseteq S^\star$.

Je zřejmé, že nejmenší pevný bod existovat může a nemusí. Pan Tarski přišel s následujícím výsledkem.

Věta 3.3
Je-li f monotonní, pak existuje nejmenší pevný bod f. Monotonií se v tomto případě rozumí vlastnost

$$R_1^\star \subseteq R_2^\star \Longrightarrow f(R_1^\star) \subseteq f(R_2^\star)$$

Tuto vlastnost lze přeformulovat takto: f je monotonní, právě když

$$f(R_1^\star \cup R_2^\star) \supseteq f(R_1^\star) \cup f(R_2^\star)$$

Monotonie f však není nutná podmínka existence nejmenšího pevného bodu. Stačí, aby f byla aditivní.

Nejmenší pevný bod pro konečnou relaci R se získá pro monotonní f opakovanou aplikací funkce f. Je-li totiž např. $\emptyset$ výchozí relace, pak víme, že platí $f^{i-1}(\emptyset) \subseteq f^i(\emptyset)$. Musí tedy existovat $n_0 \geq 1$:

$$\emptyset \subset f(\emptyset) \subseteq \cdots \subseteq f^{n_0}(\emptyset) = f^{n_0+1}(\emptyset)$$

Tvrzení $f^{n_0}(\emptyset)$ je nejmenší pevný bod. *Důkaz:* Důkaz tohoto tvrzení je uveden v [6]. Je sice veden pro potřeby λ-kalkulu, ovšem je naprosto analogický s důkazem našeho tvrzení. □

Tvrzení Tranzitivní uzávěr binární relace $R^\star$, definované na doméně D je nejmenší pevný bod rovnice

$$S = S \circ R^\star \cup R^\star,$$

kde S je relační proměnná. *Důkaz:* Důkaz je zřejmý, když si uvědomíme, že

$$f^n(\emptyset) = \bigcup_{i=1}^{n} \underbrace{R^\star \circ \cdots \circ R^\star}_{i}$$

□

Tvrzení Každý výraz relační algebry, který neobsahuje operátor $-$, je aditivní. *Důkaz:* necháváme čtenáři jako cvičení. □

Na závěr odstavce tři poznámky.

- Mohou existovat nemonotonní výrazy, které mají nejmenší pevný bod.
- Je-li ve výrazu relační algebry operátor $-$, může být tento výraz monotonní.
- Nejmenší pevný bod nemusí existovat. Může však existovat větší počet minimálních pevných bodů, vzájemně neporovnatelných.

3.2 Datalog

Konečně se začínáme dostávat k jádru věci. *Datalog* je jakási databázová verze Prologu. Logický program může obsahovat pravidla a tvrzení (viz příklad na začátku kapitoly), ovšem sémantika logického programu bude definována jinak. Syntaktické konstrukce Datalogu jsou:

- datalogické termy jsou proměnné a konstanty
- datalogická tvrzení jsou atomické formule s konstantami
- datalogická pravidla jsou hornovy klauzule
- v hlavě datalogického pravidla jsou pouze virtuální relace
- datalogické predikáty jsou buď relace (pravé predikáty), virtuální relace (nepravé predikáty) a vestavěné predikáty ($=, <, \leq, \ldots$)

Pokud bychom povolili v datalogických programech libovolná pravidla, mohli bychom se dočkat nečekaných problémů (např. bychom dostávali výčet celých domén, místo doplňků relací apod.). Musíme se proto omezit na tzv. *bezpečná pravidla*, jejichž definice následuje.

Definice 3.4
Bezpečná pravidla jsou taková pravidla, v nichž jsou všechny proměnné omezené. Proměnná je *omezená*, pokud splňuje některou z následujících podmínek:

(i) Je v literálu s pravým predikátem v těle pravidla.

(ii) Je v literálu `x=a` nebo `a=x` v těle pravidla.

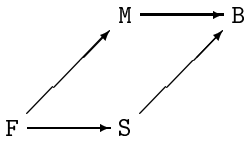
(iii) Je v literálu `x=y` nebo `y=x` a `x` je omezená nebo `y` je omezená.

Příklad 3.2
Pravidla `GREATER(x,y) :- x>y` a `FRIEND(x,y) :- M(x)` nejsou bezpečná a `S1(y,w) :- F(x,y),F(x,w),y` $\neq$ `w` je bezpečné.

Vyhodnocení datalogického programu závisí na jeho rekurzivitě. Ta se dá rozpoznat grafem závislosti.

Definice 3.5
Graf závislosti s predikáty z **R** a IDB jako uzly, a hranami (U, V), kde pro U, V existuje pravidlo V `:-` $\ldots U \ldots$

Příklad 3.3
Obrázek 2 ukazuje příklad grafu závislosti na výše zmíněných pravidlech.



Obrázek 2: Příklad grafu závislosti

Tvrzení Existuje-li cyklus v grafu závislosti, program je rekurzivní.

3.3 Vyhodnocování datalogických programů

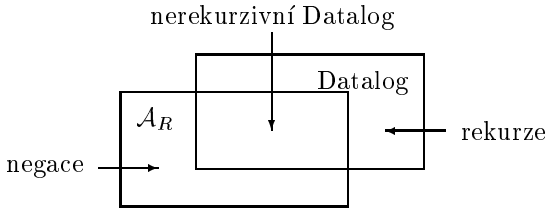
Vyhodnocování datalogických programů se provádí podle grafu závislosti. Při tom se používají tato pravidla:

- `F(x,y),F(x,w)` je spojení relací
- `F(x,y),x=w` je selekce
- `S(y,w) :- ...` je projekce
- posloupnost pravidel

  ```
  S(y,w) :- ...
  S(y,w) :- ...
  ```

  reprezentuje operaci UNION.

Z výše uvedených pravidel plyne, že nerekurzivní Datalog nepřináší vůbec nic jiného, než co přináší relační algebra. Přesněji tento důsledek zobrazuje obrázek 3



Obrázek 3: Vztah vyjadřovací síly relační algebry $\mathcal{A}_R$ a Datalogu

Pro převod datalogických nerekurzivních programů je třeba ještě vyřešit některé technické věci, např.

- Formuli `P(a,x)`, kde `a` je konstanta, převedeme na výraz relační algebry `P(1=a)[2]`.

- Podobně `R(x,x,y)` převedeme na `R(1=2)[1,3]`.

Před zahájením vlastního vyhodnocování datalogického programu je nutná taková transformace pravidel, která zaručí, aby hlavy se stejným predikátem měly stejně pojmenované proměnné, tzv. *rektifikace pravidel.*

Příklad 3.4
Rektifikujme pravidla

```
P(a,x,y) :- R(x,y).
P(x,y,x) :- R(y,x).
```

Nejprve zavedeme proměnné `u`, `v` a `w` a nasadíme je do hlav pravidel.

```
P(u,v,w) :- R(x,y),u=a,v=x,w=y.
P(u,v,w) :- R(y,x),u=x,v=y,w=x.
```

Vidíme, že jsou zde nadbytečné proměnné, tedy pravidla přepíšeme do tvaru:

```
P(u,v,w) :- R(v,w),u=a.
P(u,v,w) :- R(v,u),w=u.
```

Lemma 3.6

1. Je-li pravidlo bezpečné, pak po rektifikaci je opět bezpečné.
2. Původní a rektifikované pravidlo jsou ekvivalentní.

Tvrzení Vyhodnocený program poskytuje pro každý predikát z IDB množinu tvrzení, která

1. tvoří množinu těch tvrzení, která jsou dokazatelná z EDB aplikací pravidel z IDB;
2. tvoří minimální model pravidel z IDB.

Důkaz: indukcí podle pořadí pravidel. □

3.4 Rekurze v Datalogu

Typickým příkladem datalogického programu je tranzitivní uzávěr (již víme, že dotaz na tranzitivní uzávěr je prostředky relační algebry nevyjadřitelný).

Příklad 3.5
Mějme stromovou strukturu uloženu v relaci `otec(x,y)` (tj. otec `x` je `y`). Program na tranzitivní uzávěr může vypadat takto:

```
predek(x,y) :- otec(x,y).
predek(x,y) :- otec(x,z),predek(z,y).
```

Zřejmě platí `otec` $\subseteq$ `predek` a `otec * predek[1,3]` $\subseteq$ `predek`. Máme tedy rovnici

$$\texttt{(otec * predek)[1,3]} \cup \texttt{otec} = \texttt{predek}$$

Obecně máme soustavu rovnic pro IDB

$$E_i(R_1, \ldots, R_n) = R_i$$

pro $i = 1, \ldots, n$. Řešení takové soustavy závisí na EDB a nazýváme je *pevný bod.*

Vyhodnocování rekurzivního datalogického programu pak probíhá tak, že vyhodnocují postupně tyto rovnice, dokud se do virtuálních relací přidávají nové prvky. Takto jednoduché vyhodnocování má však za následek vytváření duplicitních n-tic (z matematického hlediska nám nevadí, ale zabírají místo a čas vyhodnocování) a zbytečně velkých relací, chceme-li ve výsledku pouze selekci. Proto se zavádějí všalijaké rafinované metody, které tyto nešvary odstraňují (např. metoda diferencí), ale na principu vyhodnocování nic nemění.

3.5 Rozšíření Datalogu o negaci

Bez negace se v některých případech nemůžeme obejít. Například chceme-li vytvořit datalogický program, který vypočítá relaci všech příbuzných, kteří nejsou bratři, musíme umožnit pravidlo:

`NSR(x,y) :- R(x,y), ¬S(x,y)`

a budeme to chápat jako spojení relací `R` a $\overline{\texttt{S}}$, kde $\overline{\texttt{S}}$ je doplněk relace `S` do *nějakého vhodného universa*. K tomu musíme v bezpečných formulích zakázat proměnné, které jsou v negativním literálu a nejsou omezené ve smyslu původní definice.

Příklad 3.6
`R(x,y) :- ¬S(x,y)` není bezpečné pravidlo, ale `R(x,y) :- ¬S(x,y),T(x,y)` již bezpečné je.

Opět tu však nastává problém, že řešením datalogického programu nemusí být nejmenší pevný bod, ale několik minimálních pevných bodů, viz následující příklad.

Příklad 3.7
Mějme datalogický program:

```
NUDNY(x) :- ¬ZAJIMAVY(x),MUZ(x).
ZAJIMAVY(x) :- ¬NUDNY(x),MUZ(x).
```

Nechť universum proměnné `x` je `{Lojza}`. Pak dostáváme dvě řešení, podle pořadí vyhodnocení pravidel:

- Model 1: `{NUDNY={Honza}, ZAJIMAVY=∅}`
- Model 2: `{ZAJIMAVY={Honza}, NUDNY=∅}`

Tento problém řeší tzv. *stratifikace*. Intuitivně stratifikace omezí negace tak, že nejprve musíme nějakou relaci definovat bez použití negace a další relaci můžeme definovat pomocí této bez negace nebo i s ní.

Definice 3.7
Definice virtuální relace S je množina všech pravidel, která mají S v hlavě.

Definice 3.8
Řekneme, že S se vyskytuje *pozitivně*, resp. *negativně*, právě když se vyskytuje v pozitivním, resp. negativním, literálu.

Definice 3.9
Program P je *stratifikovaný*, existuje-li dělení $P = P_1 \cup \cdots \cup P_n$ takové, že pro $i \in \{1, \ldots, n\}$ platí:

1. Vyskytuje-li se relační symbol pozitivně v klauzuli z P_i, pak jeho definice je obsažena ve sjednocení
$$\bigcup_{j \leq i} P_j$$
2. Vyskytuje-li se relační symbol negativně v klauzuli P_i, pak jeho definice je obsažena ve sjednocení
$$\bigcup_{j < i} P_j$$

Označení Dělení $P_1 \cup \cdots \cup P_n$ nazveme *stratifikací* a P_i *stratem* (nominativ zní *stratum*).

Co bude vyhodnocením negativního literálu? Je-li $P^\star$ spočítaná pro P nebo je P dáno z EDB, pak $\overline{P^\star}$ pro $\neg P(x_1, \ldots, x_n)$ je rozdíl:

$$\underbrace{(\mathcal{D} \times \cdots \times \mathcal{D})}_{n} - P^\star,$$

kde D je sjednocení konstant z EDB a pravidel.

Příklad 3.8

Program

```
P(x) :- ¬Q(x).
R(1).
Q(x) :- Q(x),¬R(x).
```

je stratifikovaný (strata jsou tři, první stratum je tvořeno prostřední formulí, druhé poslední a třetí stratum je tvořeno první formulí), zatímco program

```
P(x) :- Q(x).
Q(x) :- ¬P(x).
```

není stratifikovaný (obsahuje negativní definici cyklem).

Definice 3.10

Nechť (U, V) je hrana závislostního grafu. Pak (U, V) je *pozitivní*, existuje-li klauzule, kde V `:-` $\ldots$`,`U`,`$\ldots$ a U je v literálu pozitivní. Analogicky *negativní* hrana.

Věta 3.11

Program P je stratifikovaný, právě když jeho závislostní graf neobsahuje cyklus s negativní hranou.

Důkaz: Důkaz není ideově náročný, ale je třeba si vyhrát s detaily. Tudíž jej ponecháváme jako cvičení. □

Věta 3.11 dává návod na vyhodnocování stratifikovaných datalogických programů. Pokud totiž nahradíme všechny cykly grafu závislosti uzly, obdržíme acyklický graf, na kterém lze definovat topologické uspořádání. *Velké* uzly vyhodnocujeme podobně jako v Datalogu bez negace a ostatní pomocí domény $\mathcal{D}$.

3.6 Vyjadřovací síla

Resumé z této kapitoly je poznání, že stratifikovaný Datalog co do vyjadřovací síly pokrývá vyjadřovací sílu relační algebry a také rekurzivního Datalogu. Jeho vyjadrovací síla se tedy již rovná vyjadřovací síle Turingova stroje.

4 Dokumentografické informační systémy

Podíváme-li se do historie, vidíme, že s rozvojem prvních počítačů na konci 40. a začátku 50. let se rozvíjejí *klasické systémy sekundárních informací*, tedy tzv. *rešeršních středisek*, která schraňovala – jak již název napovídá – pouze výtahy z primárních dokumentů. Na základě nich se pak ručně v systému knihoven vyhledávaly primární informace, které byly mimo počítač. Až v době, kdy počítače začínaly mít rozumný výkon a velikost, kdy nebylo nutné při pořízení počítače stavět malou elektrárnu a zvláštní budovu, tedy na počátku 80. let se začaly rozvíjet *systémy pro zpracování úplných textů (full text)*.

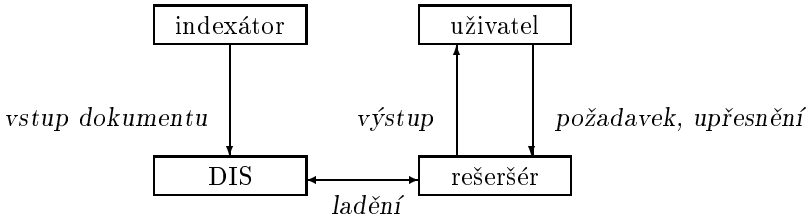
Proč full-textové systémy vznikaly? Jednak proto, že počítače již umožňovaly vznik textů přímo na nich a proto, že vznikala potřeba vyhledávat efektivněji než listováním a ne vždy bylo možné ručně indexovat[18] a jednak v důsledku rozvoje velkých pamětí (CD ROM, WORM) a vyšších výkonů počítačů.

4.1 Specifikace problému

Dokumentografický informační systém (DIS) lze obecně rozdělit do dvou subsystémů.

- subsystém dodání (pořízení) textu
- subsystém zpřístupnění textu

Podrobněji popisuje činnost DIS obrázek 4.



Obrázek 4: Schéma dokumentografického informačního systému

Podobně jako v ostatních oblastech databázové problematiky, není ani zde jednotná terminologie. Proto budeme používat vždy jedno z několika běžně užívaných označení pro daný termín.

A nyní již popišme obrázek 4.

Indexátor provádí popis *(indexaci)* dokumentu pomocí zařazení dokumentu do *kategorie* a jeho ohodnocení seznamem *klíčových slov* (užívají se také označení *termy*, *deskriptory* apod.).

Uživatel formuluje v přirozeném jazyce dotaz na DIS.

Rešeršér na základě uživatelovy specifikace formuluje dotaz ve formálním jazyce DIS.

Ladění dotazu je[19] proces, kdy uživatel spolu s rešeršérem upravují svůj dotaz do takové podoby, aby výsledek dotazu byl optimální (zejm. aby zahrnoval maximum informací, které uživatel požaduje a aby množství získaných dokumentů bylo *rozumné*, tedy aby uživatel nepotřeboval výsledek zpracovat dalším DIS.

[18] Pojem *indexace* v kontextu full-textových systémů je něco diametrálně odlišného od indexace ve smylu organizace dat na vnějších pamětech. Blíže se o ní zmíníme později, zde uveďme pouze to, že *indexace* je opatřování dokumentu *klíčovými slovy* (o nich bude také později řeč).

[19] Ladění dotazu je opět něco zcela jiného, než se obvykle pod pojmem *ladění* myslí. Je to způsobeno zejména tím, že rešeršní systémy vznikaly u velkých knihoven a jejich navrhovatelé byli více spjati s informatikou v knihovní podobě a základ jejich řešení nestál na žádné matematické teorii. V současnosti, kdy se informatika ubírala poněkud jiným směrem, stala se z těchto knihovních informatiků komunita ještě uzavřenější než je matematická komunita (matematikové se uzavírají před světem do matematické abstrakce, kdežto knihovní informatici se uzavírají nejen před světem, ale ještě také před matematiky). Dokumentografické informační systémy z rešeršních systémů vycházejí (z důvodu kompatibility s knihovní praxí) a přebírají také knihovnickou terminologii, což způsobuje, že význam mnoha pojmů v DIS staví klasický význam těchto pojmů v matematické informatice *na hlavu*.

4.2 Problém indexace

V našem zeměpisném prostoru vzniká jeden nezanedbatelný problém s indexací, který anglosaské národy nepoznaly. Uveďme si to na příkladu. Mějme dokument *Otazníky kolem boolského modelu* uložený v DIS. O tomto dokumentu je DISem udržován index, uvedený na obr. 5.

```
1. Základní vlastnosti
            Boolský model vyhledávání, rozšíření boolského modelu na úplné texty
2. Nedostatky boolského vyhledávání
3. Možnosti vylepšení
```

Obrázek 5: Příklad indexu

Zadá-li rešeršér dotaz na klíčové slovo `boolské vyhledávání`, dokument *Otazníky kolem boolského modelu* nalezen nebude, protože se v indexech toto klíčové slovo vyskytuje v jiném pádě.

Z toho je patrné, že problém automatizované indexace dokumentů je netriviální problém. Jsou však již DIS, které toto řeší, např. ByllBase.

4.3 Dotazy na DIS

V dotazech většiny DIS se vyskytují operátory `OR, AND` a `NOT`. V některých implementacích jsou dotazy omezeny[20] na konjunktivní, resp. disjunktivní, normální formu.

Slušné DIS také podporují *divoké karty (wild cards)*, známé z jiných oblastí databází. Leckdy si implementátor ulehčuje práci např. omezením operátoru `*` jen na pravostrannou formu.

Lepší DIS také rozšiřují použitelné operátory o funkce vzájemné polohy jednotlivých termů (tzv. *proximity*). Jsou to např.:

- `A & B`, tedy `A, B` jsou ve stejném kontextu (věta, odstavec).
- `AB`, tj. `A, B` bezprostředně za sebou.
- `A..B` znamená, že mezi `A, B` je cokoli libovolné délky.
- `A.`n`.B` znamená, že mezi `A, B` je právě n slov.
- `<A,B>`n znamená, že mezi `A, B` nebo `B, A` je právě n slov.
- V předchozích dvou bodech může být n zadáno také intervalem, např. `(x,y)`.

Jako příklad DIS se podívejme na izraleský `Responsa Retrieval Project`. V tomto DIS můžeme formulovat dotaz `NEDOSTATKY (1,2) VYHLEDÁVÁNÍ`. Jako odpověď jsou vyhledány všechny indexy dokumentů, které obsahují slovní spojení tvaru: `<w, kde w je synonymum ke slovu NEDOSTATKY>`, žádné nebo jedno slovo, `<w, kde w je synonymum ke slovu VYHLEDÁVÁNÍ>`.

Protože Responsa Retrieval Project dokáže pracovat se synonymy, nevystačí s klasickým slovníkem (popř. se slovníkem s možnými tvary slov). Musí mít daleko propracovanější slovník, který bude zahrnovat také synonimní vztahy mezi slovy. Nazývá se *thesaurus*.

4.4 Nedostatky boolského vyhledávání

- Formulace dotazu je více umění než věda, často je nutný rešeršér a dlouhé ladění dotazu.
- Nelze přímo řídit velikost výstupu.
- Nelze ohodnotit vystupující záznamy podle relevance (všechny jsou stejně dobré, což není v praxi pravda).
- Termy v dotazu jsou chápány jako stejně důležité (což také nevždy odpovídá skutečnosti).

[20] To je tak, když je implementátor *lenošný*.

- Nevždy je intuitivně zřejmý výsledek. Např.

  (i) V dotazu t_1 `OR ... OR` t_n vrátí ve výsledku záznamy, které obsahují všechny termy t_i, ale i ty, které obsahují jen jeden z nich.

  (ii) V dotazu t_1 `AND ... AND` t_n nevrátí ty, které neobsahují všechny t_i, ale nevrátí ani ty, které neobsahují právě jeden term t_i.

- Úspěch vyhledávání lze ovlivnit stylem práce (zde se naskýtá paralela s SQL, kdy efektivitu dotazu v SQL chápeme jako relevanci dotazu v DIS).

4.5 O relevanci

Relevance dokumentu je míra rozsahu, kterým se vybraný dokument shoduje s požadavky na něj kladenými. Pro posuzování relevance dokumentu budeme používat tyto koeficienty:

- Koeficient úplnosti R (`recall`):
$$R = \frac{m}{n},$$
kde m je počet vybraných releventních záznamů a n je počet všech relevantních záznamů v DIS.

- Koeficient přesnosti P (`precision`):
$$P = \frac{m}{o},$$
kde m je počet vybraných relevantních záznamů a o je počet všech vybraných záznamů dotazem.

Naším požadavkem na optimální dotaz jsou maximální možné hodnoty koeficientů R a P. A to je také největší zádrhel boolského vyhledávání, protože vztah mezi hodnotami koeficientů R a P ukazuje obrázek 6.

Americký systém `STAIRS`, který sloužil k ukládání dokumentů o soudních procesech a sloužil advokátům k vedení soudních pří[21]. Obsahoval bázi cca 40 000 dokumentů o celkovém rozsahu cca 350 000 stran právních textů. Pro správné využívání informací, získaných z báze systému STAIRS, byl kladen požadavek vysokého koeficientu R (alespoň 75%).

U standardního DIS lze empiricky dosáhnout hodnot 80% pro P a 20% pro R. Tuto situaci se pokoušel vyřešit pan Blaire, jehož postup si popíšeme v dalším odstavci.

4.5.1 Blairovo (uživatelské) řešení

Zlepšit nepříjemné empirické výsledky koeficientů P a R lze tímto postupem:

1. Najdeme nejprve záznam s vysokou relevancí, kterou známe, rovnou podle autora, názvu apod., pokud existuje v databázi.

2. Začneme se dotazovat s jeho klíčovými slovy.

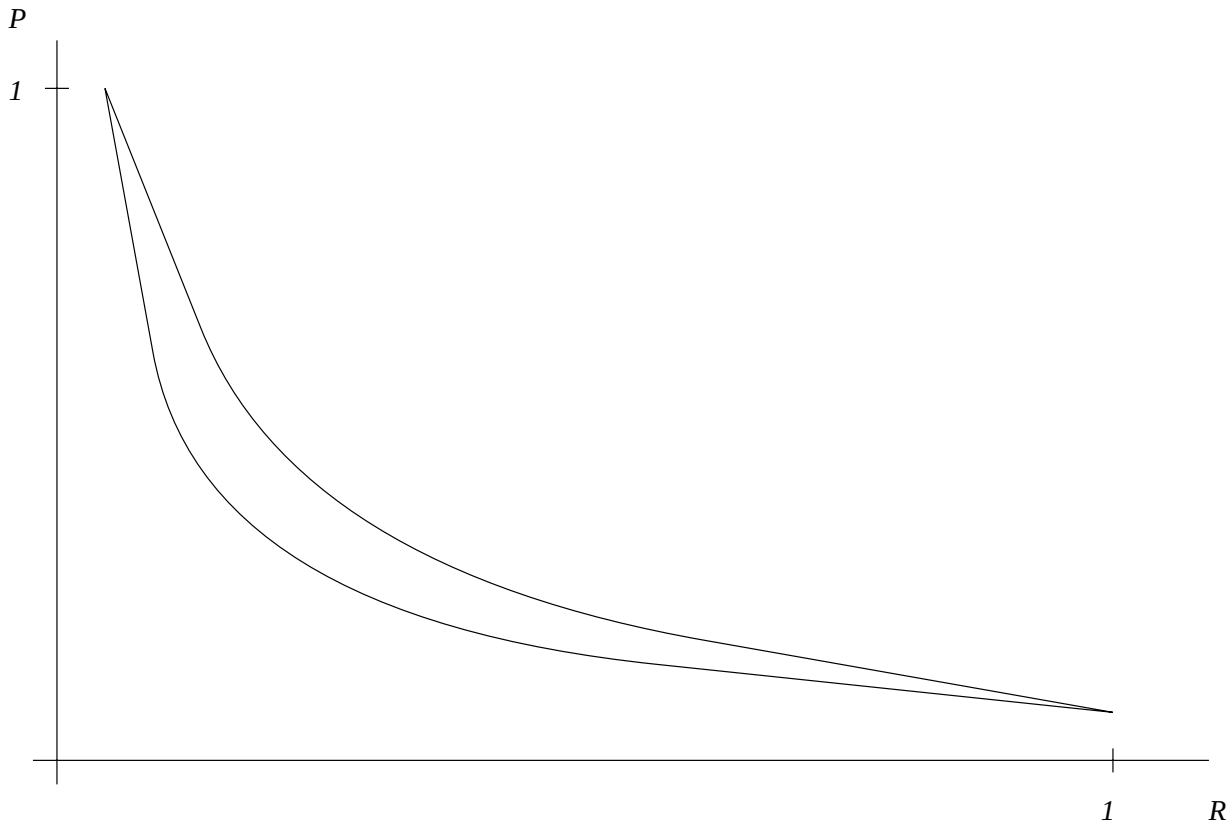
3. Odstraňujeme deskriptory resp. je nahrazujeme disjunkcemi.

Tato metoda spočívá ve zmenšování neurčitosti tazatele (to je trošku něco jiného než ladění).

4.5.2 Jednostrannost uživatele

Tento problém vyplývá z psychologie uživatelů. Uživatel se totiž dopouští několika nešvarů, které zavlékají do dotazů subjektivní pohled uživatele. Ten totiž při ladění dotazu hledá chybu v kritériích uvedených na konci dotazu, čímž přeceňuje pravděpodobnost výskytu dříve uvedených kritérií až do pozice dogmatu.

[21] Na rozdíl od našeho práva stojí americké právo na zcela odlišných základech. Kromě známé presumpce viny je již poměrně málo známým faktem, že americké právo a americké soudnictví zakládá velmi často na precedentech. To advokátům ztěžuje práci, protože soudních sporů, v nichž lze hledat precedens pro vedený spor, je daleko více než zákonů a dalších právních předpisů. U nás je to jednodušší. S trochou nadsázky a zjednodušeně lze říci, že české soudy sahají k precedentům *až když není zbytí.*



Obrázek 6: Vztah koeficientů R a P

4.5.3 Vážení dotazů

Řešením většiny problémů, které jsme výše nastínili, je vážení dotazů. Všechna kritéria dotazu ohodnotíme pravděpodobností, kterou lze chápat jako důraz na splnění daného kritéria. Pro příklad mějme dotaz:

```
Autor:     Lucký                    0.3
Rok:       1980 - 1985              0.7
Časopis:   Čs. informatika          0.2
           Inf. systémy             0.5
           MAA                      0.2
Hesla:     On-line                  0.6
           Selekční jazyk           0.7
           Dokumentografický systém 0.9
```

Máme 8 kritérií, tedy můžeme z nich sestavit 255 možných konjunkcí ($2^8 - 1$ – každé z nich tam je nebo není a musí tam alespoň jedno být.)

Algoritmus 4.1 Vyhodnocení váženého dotazu

1. Vygenerování všech kombinací termů
2. Maximální váhy pro skupiny kombinací 1 kritéria, 2 kritérií, ...
3. Splnění kritéria maxima
4. Nabídka uživateli

4.5.4 Další problémy

Řešení vážením dotazů však neodstraňuje problémy, které už jsou spíše na úrovni filozofické. Jsou to:

- Může být reference (záznam o dokumentu) relevantní a full-text nerelevantní.
- Závisí relevance na pořadí předkládaných textů (to co pro mne bylo relevantní při zadávání dotazu už může přestávat být relevantní v průběhu četby vyhledaných dokumentů).
- Není relevance zcela subjektivním soudem?

4.6 Vektorový model boolského vyhledávání

Nechť $a_1, \ldots, a_n$ jsou všechny termy, n jsou řádově desetitisíce. Dále nechť $D_1, \ldots, D_m$ jsou všechny dokumenty v DIS. Pak můžeme definovat matici $W = (w_{ij})$ typu (m, n) jejíž prvky $w_{ij} \in \langle 0, 1 \rangle$. Je-li $w_{ij} = 0$, pak se term a_j v dokumentu D_i nevyskytuje, je-li $w_{ij} = 1$, pak se term a_j v dokumentu D_i vyskytuje, nebo je maximálně relevantní v D_i (záleží na interpretaci). Čím je však w_{ij} větší, tím důležitější postavení zaujímá term a_j v dokumentu D_i. Matice W tedy představuje jakousi strukturu, dávající obraz o databázi DISu.

Dotaz je pak vektor vah $\langle q_1, \ldots, q_n \rangle$ (v praxi by to nebylo moc pohodlné, proto dotazem bude seznam uspořádaných dvojic `(<jméno termu>, <hodnota váhy>),...`). Dotazem ovšem může být dokument, což bude mít sémantiku: *nalezni takové dokumenty, které jsou podobné tomuto dokumentu.*

Pro vyhodnocování takových dotazů budeme potřebovat koeficient podobnosti S (jako similarity), který je definován:

$$S(Q, D_i) = \sum_i q_j \cdot w_{ij},$$

kde Q je dotaz (vektor vah) a D_i je vektor vah, jimiž je ohodnocen dokument. Koeficient podobnosti tedy není nic jiného než skalární součin vektorů. Čím je větší koeficient $S(Q, D_i)$, tím je D_i relevantnější vzhledem k dotazu.

Získaný výsledek dotazu lze také uspořádat podle relevance. Provede se to tak, že pro $\forall D_i$ se vypočte $S(Q, D_i)$. Dostaneme tedy seznam $S(Q, D_1), \ldots, S(Q, D_m)$, který setřídíme a vrátíme požadovaný počet dokumentů. U některých systémů lze tento počet implicitně zadat prahovou hodnotou koeficientu podobnosti.

Nevýhodou takového přístupu je, že nebere v úvahu operace `AND/OR`. Ovšem dá se tímto způsobem zvýšit koeficienty R i P až o 20%, ale na jistých souborech dat může být efekt nulový.

4.6.1 Váhy

Pro zlepšení výkonu vektorového modelu si můžeme hrát z různými vahami.

- `TF`: frekvence termu v dokumentu. Nevýhodou této váhy je, že s vysokým `TF` v mnoha textech klesá P (neboť se mj. může stát, že týmž termínem se označují různé věci).
- `IDF`: inverzní frekvence dokumentů se mění inverzně s počtem dokumentů k, ke kterým je daný term přiřazen v souboru m záznamů. Vhodná funkce je např. $\log \frac{m}{k}$.
- `TD`: rozlišení pomocí termů. Nejlepší termy jsou ty, když jsou schopny rozlišit jisté dokumenty od zbytku (tedy je-li vysoké `TF` i `IDF`. Vhodná funkce je $\texttt{TF} \cdot \texttt{IDF}$.
- Nejlepší (vyzkoušený) systém vah je tzv. normované `TD`, tj.

  $$\frac{\texttt{TD}}{\sqrt{\sum_j \texttt{TD}_j^2}}$$

  pro dokument a pro dotaz

  $$\left(\frac{1}{2} + \frac{\frac{1}{2} \cdot \texttt{TF}}{\max \texttt{TF}_i}\right) \cdot \log \frac{m}{k}$$

  Vektorový model

4.7 Možnosti zlepšování boolského DIS

- generování množin vztažených termů (např. podle četnosti vztahů),
- formace frází,
- vytváření thesaurů (nadpojem, podpojem, synonymum, antonymum,...),
- vytváření bází znalostí (vztahy mezi pojmy, zobecnění thesaurů),
- konstrukce hypertextů,
- fuzzy logika,
- nadstavba vektorovým modelem.

Hypotéza Bez přídavných informací nelze boolský model výrazněji zlepšit.

4.8 Signatury

Metoda signatur[22] spočívá v zakódování textu do řetězce nul a jedniček menší délky, než je text samotný. Tento řetězec nazveme *signaturou*.

4.8.1 Řetězené signatury

Záznam o dokumentu `(autor, abstrakt, název, rozsah)` délky z se zakóduje do řetězců délek d_1, d_2, d_3 a d_4 tak, že $d = \sum d_i \ll z$. Toto kódování je podobné hašování, protože nutně není prosté.

Dotaz v řetězených signaturách spočívá v zadání bitového vzorku, např. `0011101011001???0???00??10???00101`. Databáze se prohledává sekvenčně.

[22] Tato metoda se používá zejména na CD ROM.

4.8.2 Vrstvené signatury

Dokument je rozdělen na logické bloky, které jsou obecně různě dlouhé, ale obsahují vždy stejný počet významových slov.

Pro každé slovo vytvoříme binární reprezentaci pro slova vždy stejné délky. k je konstantní počet jedniček takové reprezentace a udává váhu reprezentace.

Pro každý blok logicky sečteme (operací `OR`) binární reprezentace všech významových slov v bloku, čimž vznikne signatura bloku. Soubor signatur je množina signatur bloků (a z důvodu proměnné délky bloků je nutné do souboru signatur také uvést adresy bloků).

Mějme dotaz `w`. Jeho binární reprezentaci označíme `br(w)`. Nechť má nějaký blok signaturu `111101`.

- Je-li `br(w)` nějakého dotazu `w` rovna `100001`, pak je náš blok adeptem na zařazení mezi relevantní bloky.
- Je-li však `br(w)=110010`, pak náš blok není relevantní pro dotaz `w`. Důvodem je právě jednička druhá od konce v `br(w)`. Jelikož signatura bloku je dána logickým součtem binárních reprezentací významových slov, term z dotazu `w` tedy v daném bloku není (v signatuře bloku je druhá od konce `0`).

Lze navrhnout konstanty d a k tak, že lze ovlivnit průměrný počet chybných bloků na výstupu (chybný blok je takový, který je vyhodnocen jako adept relevance, ale relevantní není). V celkové úspěšnosti se také uplatňují ještě konstanty n – počet bloků a Q – zadaný počet slov v dotazu. Např. je-li $d = 217$ a $k = 16$, pak počet chybných je 4%.

Jak implementovat soubor signatur? Při implementaci je nejvýhodnější tzv. transponovaný soubor, který má d sloupců, v nichž je však jen k jedniček, takže stačí procházet jen k sloupců souboru.

5 Organizace dat

Celá tato kapitola duplikuje informace, získané z přednášky *Organizace dat*. Nezbývá než se odkázat na [7]. Aby bylo lze se připravit na zkoušku z Technik zpracování dat v databázích, uvedeme si zde seznam pojmů, který do této kapitoly naší přednášky spadají: *sekvenční soubory, index-sekvenční soubory, indexované soubory, hašování, perfektní hašování, rozšířitelné hašování, grid file (mřížka), B-stromy, vícerozměrné B-stromy.*

6 Transakční zpracování a zotavení z chyb

Teoreticky transakce začal popisovat Jim Gray. Ten definoval *transakci* jako logickou jednotku práce s databázovým systémem, která

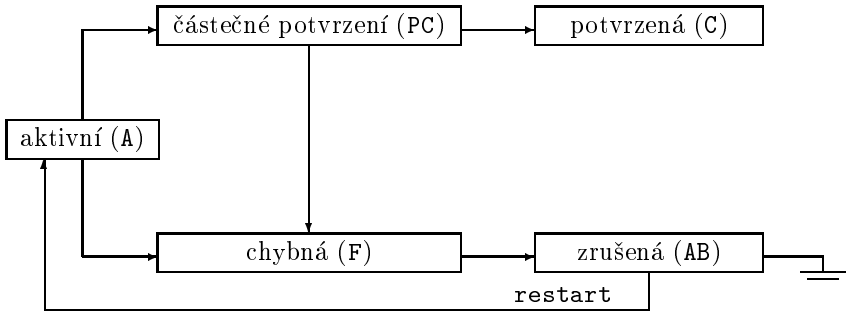
- je vyjádřena posloupností operací (ty nemusejí být nutně manipulacemi s databází),
- zachovává konzistenci databáze.

Příklad 6.1
Změníme-li číslo čtenáře v tabulce `ČTENÁŘ`, musíme také provést změny v tabulkách `VÝPŮJČKY` a `REZERVACE`, aby byla konzistence databáze zachována. Proto všechny tři operace budou tvořit jednu transakci.

Transakci lze také definovat jako posloupnost operací, která se buď provede celá nebo se neprovede vůbec. Pokud není transakce provedena do konce, je třeba obnovit původní stav databáze.
Pro obsluhu transakcí jsou definovány následující operace:

- `COMMIT` – signalizace úspěšnosti transakce,
- `ROLLBACK` – signalizace návratu do původního stavu databáze,
- `BEGIN_T` – označení začátku transakce,
- `END_T` – označení konce transakce,
- `UNDO` – provede návrat do původního stavu databáze,
- `REDO` – provede transakci znovu ze spočítaných hodnot.

Transakce se vždy nachází v jednom z pěti stavů, mezi kterými lze přecházet podle přechodového grafu z obr. 7.



Obrázek 7: Stavy transakce a přechody mezi nimi

Transakce je ukončena, je-li ve stavu `C` nebo `AB`. Transakce jsou jednoduché a krátké (vychází z bankovní transakce), nelze je zahnizďovat (tj. v průběhu neukončené transakce nelze zahájit novou transakci[23]). Nic však nebrání aplikaci

[23] U některých DBS toto nepřipadá v úvahu vůbec (např. u DBS Oracle), protože po zavolání `COMMIT` nebo `ROLLBACK` se automaticky zahajuje nová transakce, což je jediný způsob, jak transakci zahájit (již z definice oráklovského DML). Jsou však také jiné DBS, které transakci zahajují explicitně (např. Informix příkazem `BEGIN WORK`) a kompilátor není schopen poznat, zda nejsou transakce vnořovány. Pokud v době běhu dojde k zavolání příkazu `BEGIN WORK` v rámci neukončené transakce, je ohlášena běhová chyba.

dlouhých transakcí, což má ovšem za následek prudké zhoršení chování DBS (protože transakční mechanismy jsou postaveny na zamykání záznamů atd.).

Transakce je také jednotkou zotavení z chyb. Je zakládán tzv. *žurnál*[24].

6.1 Transakční žurnál

V následujícím textu budeme uvažovat následující operace jako definované:

- `READ(X,x)` – přečte z bufferu `X` hodnotu do proměnné `x`.
- `WRITE(X,x)` – zapíše do bufferu `X` hodnotu proměnné `x`.
- `OUTPUT(X)` – zapíše buffer `X` na disk.
- `FETCH()` – se použije k načtení bloku do vnitřní paměti.

Zotavení ze spadnutí systému je klíčová akce mezi stavy `PC` a `C`. Po restartu systému je třeba provést `ROLLBACK` od kontrolního bodu *(save point)*. V každém kontrolním bodu se provádí:

- zápis bufferů na disk (fyzická databáze),
- zápis kontrolních záznamů do žurnálu,
- seznam všech právě prováděných transakcí.

Žurnál lze implementovat dvěma základními způsoby s odloženou realizací změn a s bezprostřední realizací změn.

6.1.1 Žurnál s odloženou realizací změn

Tato implementace se řídí strategií, kdy v průběhu transakce se mění pouze buffery a na konci transakce se změny pošlou na disk. Změny se do žurnálu zaznamenávají ve tvaru:

```
<id_transakce,jméno_atributu,nová_hodnota>
```

a operace `WRITE` jsou odloženy až na `PC`. Na začátku transakce se do žurnálu zapíše:

```
<id_transakce,START>
```

a ve stavu `PC`:

```
<id_transakce,COMMIT>
```

Je-li čas, kdy nastala chyba (t_F) menší, než čas, kdy se transakce dostala do stavu `PC` (t_{PC}) je provedeno `REDO(T)`, tedy projde se žurnál od konce, až dokud není nalezen záznam `<T,START>` a od tohoto místa se provádí všechny akce, zaznamenané v rámci transakce `T`. V opačném případě se ingoruje příslušný kus žurnálu a provede se restart.

Vzniká však otázka, co se děje, pokud systém spande v průběhu akce `REDO`.

6.1.2 Žurnál s bezprostřední realizací změn

Změny se do žurnálu zaznamenávají ve tvaru

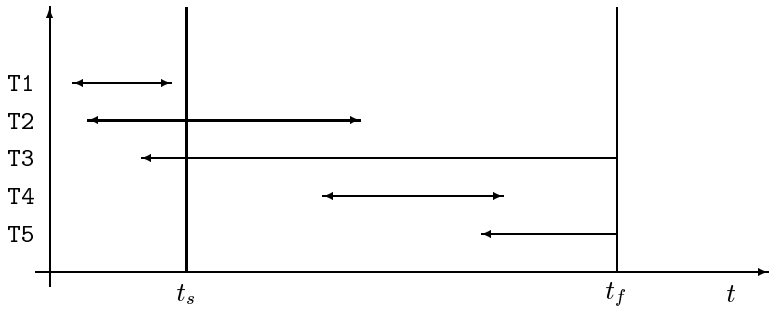
```
<id_transakce,jméno_atributu,stará_hodnota, nová_hodnota>
```

vždy před příslušným `OUTPUT(X)` (v kontrolním bodě) a bezprostředně před `WRITE(X,x)`.

[24] Brrrrrrrrrr. To je ale hnusné slovo. Abychom si trošku zlidštili tento pojem, tak anglicky se nazývá *log file*.

Po zhroucení systému se provede rekonstrukce pomocí `REDO(T)` pro každou potvrzenou transakci `T` a pomocí `UNDO` pro každou nedokončenou transakci `T`.

Příklad 6.2
Mějme transakce `T1`,...,`T5`, jejichž dobu trvání ukazuje obr. 8. Čas t_s udává kontrolní bod a t_f je okamžik zhroucení



Obrázek 8: Časová osa provádění transakcí

systému.

Transakce `T1` byla ukončena před kontrolním bodem a bylo tedy již provedeno příslušné `OUTPUT`. Ta nás při rekonstrukci již nebolí. Transakce `T2` a `T4` byly před zhroucením systému potvrzeny, rekonstruují se tedy pomocí operace `REDO`. Zbývající transakce v okamžiku t_f ještě nebyly dokončeny, rekonstrukce se tedy provede operací `UNDO`.

Při restartu se tedy postupuje podle následujícího algoritmu:

Algoritmus 6.1
`UN` a `RE` jsou seznamy transakcí.

1. Do seznamu `UN` dej všechny v čase t_s nedokončené transakce; seznam `RE` inicializuj jako prázdný.
2. prohledej žurnál počínaje od posledního kontrolního bodu do konce. Každou novou zahájenou transakci přidej do seznamu `UN`. Každou transakci `T`, pro kterou najdeš `<T,COMMIT>`, přesuň ze seznamu `UN` do seznamu `RE`.
3. Průchodem žurnálem od konce do kontrolního bodu prováděj `UNDO` pro transakce ze seznamu `UN`; průchodem od kontrolního bodu do konce prováděj `REDO` pro transakce ze seznamu `RE`.

6.2 Paralelní zpracování transakcí

Zatím jsme uvažovali transakce spíše za sebou. Paralelní zpracování umožňuje několik transakcí současně se sdílením procesoru v době realizace. Máme-li například transakce `T1` a `T2`, které se skládají z operací $t_{11}, \ldots, t_{1m}$, resp. $t_{21}, \ldots, t_{2M}$, pak to může vypadat jako na obrázku 9. Graf označený číslem `3` udává výsledné provedení jednotlivých operací. Stanovené pořadí operací více transakcí v čase se nazývá *rozvrhem*. Rozvrh, který zabezpečí, že operace každé transakce jsou provedeny pohromadě (bez prolnutí operací jiné transakce), se nazývá *sériový* rozvrh.

Proč budeme požadovat, aby rozvrhy při paralelním zpracování byly sériové? Pokud by sériové nebyly, docházelo by k tzv. ztrátám aktualizace. Příklad takové ztráty je uveden na obrázku 10

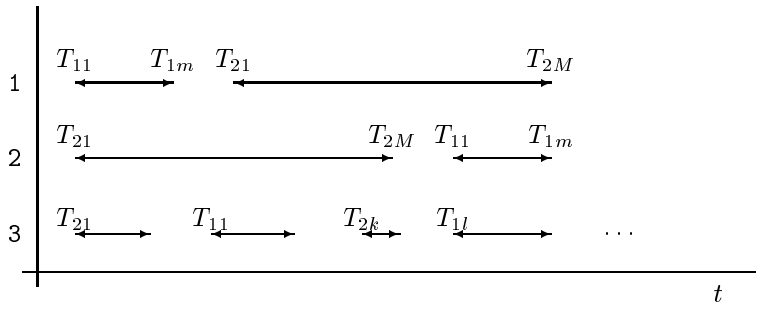
Druhou možností, kterak poškodit konzistenci databáze prostřednictvím prolínání operací více transakcí je dočasná aktualizace při chybě systému. Pokud jsou provedeny akce podle obrázku 11 a dojde k chybě (výpadku) systému (označeno symbolem †) a tedy k následné rekonstrukci databáze, ztratí se aktualizace transakce `T2`.

Rozvrh tedy budeme brát jako *korektní*, pokud je **nějak** ekvivalentní sériovému rozvrhu. Systém pak bude zaručovat *uspořádatelnost* rozvrhu. Přesněji: rozvrhy S_1 a S_2 pro $\{T_1, \ldots, T_N\}$ jsou *ekvivalentní*, je-li splněno:

(i) jestliže v T_i je `READ(A)` a tato hodnota vznikla v T_j z operace `WRITE(A)`, pak to musí platit v rozvrhu S_1 i S_2.

(ii) je-li v jednom rozvrhu poslední `WRITE(A)` v T_i, pak totéž musí platit i ve druhém rozvrhu.

Příklad 6.3
Mějme transakce `T1` a `T2`:



Obrázek 9: Paralelní zpracování transakcí

T1	T2	stav databáze
read(X)		X=80
X := X-5		zrušení pěti rezervací
	read(X)	X=80
	X := X+4	rezervuj 4 místa
write(X)		X=75
read(V)		
	write(X)	X=84, ale správně by mělo být 79
...		

Obrázek 10: Příklad ztráty aktualizace

T1	T2
read(X)	
X := X-5	
	read(X)
	X := X+4
	write(X)
read(V)	
†	

Obrázek 11: Příklad dočasné aktualizace při chybě systému

```
T1: { READ(A), akce1(A), WRITE(A), READ(B), akce2(B), WRITE(B) }
T2: { READ(A), akce3(A), WRITE(A), READ(B), akce4(B), WRITE(B) }
```

Nechť rozvrh $S_1 = \{$`T1, T2`$\}$ a $S_2 = \{$`T2, T1`$\}$. Pak rozvrhy S_1 a S_2 nejsou ekvivalentní (i když jsou oba sériové).

Poznámka V uspořádaném rozvrhu se databáze podrobí těm I/O operacím jako v ekvivalentním sériovém rozvrhu (konfliktní operace jsou provedeny ve stejném pořadí).

Příklad 6.4
Definujme rozvrhy S_3 a S_4:

S_3		S_4	
T1	T2	T1	T2
READ(A)		READ(A)	
AKCE1(A)		AKCE1(A)	
WRITE(A)			READ(A)
	READ(A)		AKCE3(A)
	AKCE3(A)		WRITE(A)
	WRITE(A)		READ(B)
READ(B)		WRITE(A)	
AKCE2(B)		READ(B)	
WRITE(B)		AKCE2(B)	
	READ(B)	WRITE(B)	
	AKCE4(B)		AKCE4(B)
	WRITE(B)		WRITE(B)

Rozvrhy S_3 a S_4 nejsou sériové (prolínají transakce), ale přitom platí, že S_3 je ekvivalentní rozvrhu S_1.

Poznámka Mohou existovat rozvrhy, které nejsou ekvivalentní a přitom produkují stejný výsledek.
Rozvrh S_1 je *uspořádatelný*, existuje-li sériový rozvrh S_2 takový, že S_1 je ekvivalentní S_2 ($S_1 \equiv S_2$).

6.3 Testování uspořádatelnosti rozvrhů

Na testování uspořádatelnosti rozvrhů existuje několik více či méně inteligentních.

6.3.1 `READ` předchází `WRITE`

Budeme předpokládat, že v transakci je hodnota atributu `A` vždy nejprve čtena a pak zapsána. K tomu se používá tzv. *precedenční graf* rozvrhu, jehož uzly jsou transakce a mezi `Ti` a `Tj` je hrana, právě když je splněna alespoň jedna z následujících podmínek:

(i) `Ti` volá `WRITE(A)` před tím, než `Tj` volá `READ(A)`

(ii) `Ti` volá `READ(A)` před tím, než `Tj` volá `WRITE(A)`

Přitom platí tvrzení, že rozvrh je uspořádatelný, pokud v jeho precedenčním grafu neexstují cykly (detekce cyklu v grafu probíhá v čase $O(n^2)$, kde n je počet uzlů).

Příklad 6.5
Z předchozích příkladů vznikají tyto precedenční vztahy pro rozvrhy S_i:

- `T1` $\xrightarrow{S_1}$ `T2`
- `T1` $\xleftarrow{S_2}$ `T2`
- `T1` $\xrightarrow{S_3}$ `T2`
- `T1` $\overset{S_4}{\rightleftharpoons}$ `T2`

Příklad 6.6
Mějme acyklický precedenční graf rozvrhu

$$\begin{array}{ccc} \texttt{Ti} & \longrightarrow & \texttt{Tk} \\ \downarrow & & \downarrow \\ \texttt{Tj} & \longrightarrow & \texttt{Tm} \end{array}$$

Pak existuje alespoň jeden uspořádaný rozvrh, ekvivalentní s původním rozvrhem. V našem případě to jsou:

- `Ti` $\longrightarrow$ `Tj` $\longrightarrow$ `Tk` $\longrightarrow$ `Tm`
- `Ti` $\longrightarrow$ `Tk` $\longrightarrow$ `Tj` $\longrightarrow$ `Tm`

6.3.2 `WRITE` kdekoli

V tomto případě neexistuje efektivní algoritmus na testování uspořádatelnosti, protože v tomto případě to je NP-úplný problém.

6.3.3 Uzamykací protokoly

Aby se dosáhlo uspořádatelnosti rozvrhu, používá se dynamické zamykání a odemykání objektů.

Zamknutí je akce na objektu, kterou vyvolá transakce, aby jej uchránila před přístupem jiných transakcí a aby k němu mohla přistupovat jen ona.

Odemknutí je akce, která uvolňuje objekt pro zpracování dalšími transakcemi.

Legální rozvrh je rozvrh, kde transakce neuzamyká již zamčený objekt.

V tomto případě však platí tvrzení, že legálnost rozvrhu ještě není postačující podmínka pro uspořádatelnost rozvrhu.

Kvůli tomuto tvrzení musíme ještě vytvořit *uzamykací protokol*, tedy množinu pravidel, které udávají, kdy bude transakce uzamykat a kdy bude odemykat objekty. Cílem takového protokolu je dosáhnout ekvivalence mezi legálností a uspořádatelností rozvrhu.

Nejpoužívanějším uzamykacím protokolem je tzv. *dvoufázový protokol*. Jeho ideou je donutit transakce, aby nejprve uzamknuly potřebné objekty (1. fáze), provedly potřebné operace nad daty a uzamčené objekty odemknuly (2. fáze).

Dvoufázový protokol tedy připouští pouze *dvoufázové transakce*, tj. transakce, která jakmile zavolá první `UNLOCK`, pak již nesmí zavolat `LOCK`.

Dále musíme vyloučit patologické případy, tedy pokud v transakci nenastane žádný z případů:

- zavolání `LOCK(A)` na transakcí již uzamčený objekt `A`,
- zavolání `UNLOCK(A)` na transakcí již odemčený objekt `A`,
- po ukončení transakce `T` je některý objekt touto transakcí uzamčen,

nazveme tuto transakci *dobře formovanou*.

Tvrzení Nechť $S = \{\texttt{T1}, \ldots, \texttt{Tn}\}$. Jsou-li všechny `Ti` dobře formované a dvoufázové, pak každý jejich legální rozvrh S je uspořádatelný.

V rozvrhu může nastat uváznutí (když `T1` uzamykáme něco, co je uzamčeno v `T2` a naopak), ze kterého se lze dostat pouze `ROLLBACK`em jedné z nich.

Existují také jiné nedvoufázové protokoly, které zaručují uspořádatelnost legálního rozvrhu dobře formovaných transakcí. Také existují jiné nedvoufázové protokoly, v nichž jsou nedobře formované transakce s legálními rozvrhy uspořádatelné.

Literatura

[1] ???: *Dotazovací jazyky*

[2] Pokorný a spol.: *Úvod do databází*, skriptum MFF UK

[3] ???: *Distribuované databáze*

[4] T. Havlát, M. Benešovský: *Úvod do databázových systémů*, skriptum PřF UJEP, Brno 1986

[5] J. Dobeš: *Překladače*, příspěvek do projektu ZKUSTO, 1995

[6] J. Pojsl: *Funkcionální programování*, příspěvek do projektu ZKUSTO, 1994

[7] Jiří Dobeš: *Organizace dat*, příspěvek do projektu ZKUSTO, 1994