Deduktivní databáze a jejich implementace v relacním prostredí

KAREL JEŽEK[1]

[1]*Katedra informatiky a výpocetní techniky, FAV ZCU Plzen*
Univerzitní 8, 306 14 Plzen
jezek_ka@kiv.zcu.cz

Abstrakt. Deduktivní databázové systémy sdružují schopnosti databází zpracovávat velké objemy dat s vyjadrovacími schopnostmi logických programovacích jazyku. Príspevek podává prehled existujících deduktivních systému a popisuje zpusob implementace deduktivního databázového systému založeného na prekladu jazyka Datalog do jazyka SQL. Uvedený experimentální deduktivní databázový systém (EDD) pracuje jako front end systému Oracle a využívá jeho procedurální nadstavbu PL/SQL. Ke zvýšení vyjadrovací síly byl jazyk Datalog rozšíren o možnosti použití agregacních funkcí, prirazovacího príkazu a neurcitosti.

Klícová slova: Deduktivní databáze, Datalog, integrace deduktivní a relacní databáze

1 Úvod

Deduktivní databáze predstavují oblast, ve které se prolínají databáze s logikou, logickým programováním, umelou inteligencí a znalostními bázemi.

Deduktivní databázový systém umožnuje definovat a zpracovávat pravidla, na základe kterých je možné z faktu uložených v databázi odvozovat další informace. Protože významným teoretickým základem deduktivních databází je matematická logika, používá se pro ne i oznacení logické databáze. Jejich blízkými príbuznými vzniklými v oblasti umelé inteligence jsou expertní databázové systémy a systémy pro zpracování znalostí. Oproti deduktivním databázovým systémum však tyto systémy uprednostnují uložení dat v operacní pameti a místo extrakce informací, které jsou implicitne obsaženy v datech, získávají informace od doménove orientovaného experta.

Model používaný pro deduktivní databáze je blízký relacnímu datovému modelu. Databáze je specifikována pomocí faktu a pravidel. Fakta jsou popisována podobným zpusobem jako relace, nemusí však zahrnovat jména atributu. Význam atributu je urcen jeho pozicí v n-tici. Pravidla jsou obdobou relacních pohledu. Urcují virtuální relace, které jsou získány usuzovacím mechanismem pri aplikaci pravidel na fakta. Pravidla mohou být i rekurzivní, mohou tedy predstavovat relace, které standardní relacní pohledy definovat nemohou. Jazyk SQL ve své nové norme SQL99 (SQL3) již

konstrukci pro omezené rekurzivní dotazy zavádí, v dostupných systémech však dosud implementace rekurze zahrnuta není.

Deduktivní databáze jsou predmetem zájmu databázové komunity od osmdesátých let. V soucasné dobe se výzkum a vývoj v této oblasti soustreduje jednak na propojení s objektovými principy, bud pridáním deduktivních schopností do objektove orientovaného databázového systému nebo zahrnutím objektove orientovaného jazyka do deduktivního systému. Druhý smer bádání lze charakterizovat jako snahu o zakomponování neurcitosti do procesu dedukce a vyvinutí dokonalejších, realite sveta adekvátnejších prostredku vyhledávání informací [1], [4], [6].

Následující kapitola zavádí potrebné základní pojmy a principy. Kapitola 3. podává prehled o vybraných deduktivních databázových systémech. Výber je motivován bud historickým významem nebo pozoruhodným rešením popisovaného projektu a v žádném prípade není vycerpávající. Jeho hlavním cílem je nasmerovat zájemce o tuto oblast na další zdroje informací. Kapitola 4. popisuje zpusob prekladu základního jazyka Datalog do SQL a 5. kapitola uvádí rozšírení jazyka Datalog v experimentálním deduktivním databázovém systému (EDD) [2].

2 Základní principy a pojmy

Za deduktivní považujeme takovou databázi, ve které cást jí poskytovaných informací není explicitne v databázi uložena a je získávána odvozením prostrednictvím dedukcních pravidel. Prostredkem pro zápis pravidel je logický dotazovací a definicní jazyk Datalog (zkratka z "database logic"). Datalog je variantou jazyka Prolog. Z Prologu prejímá syntaktické konstrukce a shodne s Prologem používá i základní terminologické pojmy. Program Datalogu sestává z promenných, konstant, predikátu a v prípade EDD-Datalogu i z omezené množiny funkcních symbolu. Konjunkce je oznacena ",", implikace ":-" a negace "not". Použijeme notaci, ve které promenné zacínají velkým písmenem, konstantami mohou být císla nebo retezce, predikátové symboly zacínají malým písmenem.

2.1 Datalogovský program

Pro formální definici datalogovského programu zavedeme následující pojmy. Term je promenná nebo konstanta. Je-li p predikátovým symbolem a $t_1, t_2, \ldots, t_N$ jsou termy, pak $p(t_1, t_2, \ldots, t_N)$ je atomem. Atom, jehož všechny termy t_i jsou konstantami, nazýváme základním atomem. Literál je bud atom nebo negovaný atom.

Program je kolekcí pravidel. Pravidlo pro predikát p arity N má tvar

$$p(t_1, t_2, \ldots, t_N) :- L_1, L_2, \ldots, L_M.$$

$L_1, L_2, \ldots, L_M$ jsou literály, $t_1, t_2, \ldots, t_N$ jsou termy. Levá strana implikace je nazývána hlavou a pravá telem pravidla. Literály tela je zvykem oznacovat jako podcíle. Takové pravidlo, které má M=0 a $t_1, t_2, \ldots, t_N$ jsou konstanty (je urceno základním atomem) se nazývá faktem.

Deduktivní databáze je složena ze dvou cástí:

- z extenzionální databáze (EDB) obsahující základní data uložená v databázi v podobe faktu (extenzionálních predikátu),
- z intenzionální databáze (IDB) obsahující virtuální relace (intenzionální predikáty) definované prostrednictvím jednoho ci více pravidel.

Pro smysluplnost programu se požaduje, aby každý predikát patril bud mezi intenzionální nebo mezi extenzionální (ne však do obou skupin soucasne).

Pr. Necht je EDB tvorena relacemi:

miluje(osoba, pivo).
prodává(bar, pivo, cena).
navštevuje(osoba, bar).

a IDB je definována pravidlem:

```
štastný(X) :- navštevuje(X, Bar),
              prodává(Bar, Pivo, Cena),
              miluje(X, Pivo).
```

Ekvivalence uvedeného pravidla s výrazem relacní algebry je v tomto prípade zrejmá:

$$štastný(X) = projekce_{osoba}\,(navštevuje\ join\ miluje\ join\ prodává)$$

Pro vyhodnotitelnost virtuálních relací musí pravidla splnovat tzv. podmínky bezpecnosti. Vyskytuje-li se promenná

- v hlave pravidla nebo
- v negovaném podcíli tela pravidla nebo
- v podcíli tela tvaru aritmetického porovnání,

pak se musí také vyskytovat v pozitivním podcíli tela tohoto pravidla.

Dotazem v Datalogu je kolekce jednoho nebo více pravidel. Predikáty hlav pravidel systém vyhodnotí. Forma zprístupnení výsledku uživateli závisí na konkrétní implementaci. Bežné je použití klauzule ve tvaru prologovského dotazu, napr.

```
?- štastný(X).
```

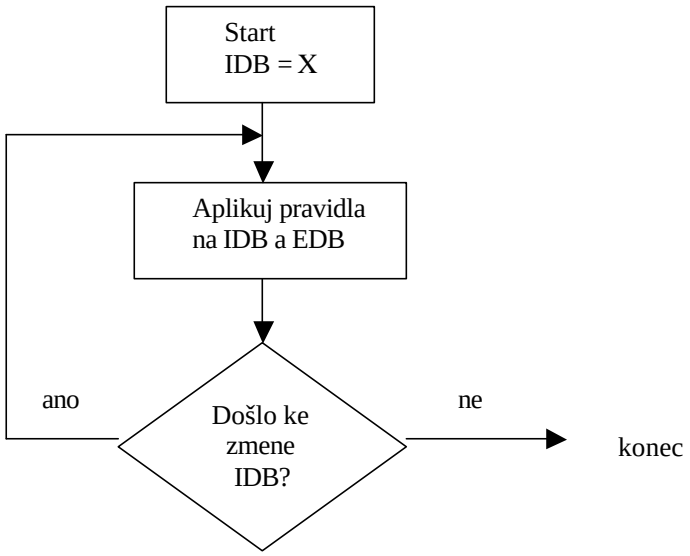
2.2 Vyhodnocování programu a rekurze

Obecný tvar dotazu v Datalogu má tvar ?-w(x), kde x je vektorem promenných a konstant a w(x) je konjunkcí literálu. Odpovedí na dotaz je množina všech instancí x taková, že w(x) je pravdivé vzhledem k intenzionální i extenzionální databázi. Datalog je schopen prostrednictvím rekurzivních pravidel definovat i rekurzivní vztahy mezi daty. Relacní algebrou ale není možné tyto vztahy vyjádrit. Pri vyhodnocování dotazu zahrnujícího rekurzivní pravidla je možné postupovat dvema zpusoby:

- Shora dolu, kdy vyhodnocovací procedura zacíná od cílového dotazu q a aplikuje pravidla tak, aby q bylo pravdivé. Podmínky pravdivosti jsou dány predikáty, kterými je q definováno a lze je chápat jako další podcíle. Proces se opakuje dokud podmínky nemají podobu faktu.
- Zdola nahoru. Vyhodnocování zacíná od faktu a aplikuje deduktivní pravidla dokud lze odvozovat nová, pravidlum vyhovující fakta.

Na metode shora dolu je založeno vyhodnocování Prologu. Datalogovské programy se vyhodnocují zdola nahoru. Duvody jsou následující: Vyhodnocování

shora dolu je závislé na poradí pravidel a faktu v programu i literálu v pravidlech, oproti tomu Datalog respektuje deklarativní smysl programu. Napr. v prípade vyhodnocování zdola nedochází k nekonecnému cyklu pri výskytu levé rekurze v pravidlech programu. Jeho další predností je množinový zpusob zpracování. Zatímco pri vyhodnocování metodou shora dolu je výsledkem jeden záznam a pri nalézání dalších dochází k opakovanému vyhodnocování podcílu, metodou zdola nahoru jsou získána všechna fakta (vcetne odvoditelných) odpovídající dotazu. Tato množina faktu se nazývá pevným bodem procesu odvození. Postup výpoctu je iterativní, znázornený je na obr.1.



Obr.1. Vyhodnocování pevného bodu rekurzivních pravidel

Výpocet zdola nahoru má však i nevýhody. Oproti výpoctu shora dolu nebere pri vyhodnocení pevného bodu v úvahu konstanty dotazu a pocítá proto i prípadne nedotazovaná fakta., která je nutné v následujícím kroku odfiltrovat. Z metod, které zohlednují tuto skutecnost a zefektivnují výpocet je nejpoužívanejší tzv. metoda magických množin. Lze ji chápat jako kombinaci obou zmínených zpusobu vyhodnocení. Pro zadaný dotaz Q je prepsána puvodní množina pravidel D na množinu pravidel D' vyhodnocující pouze fakta relevantní k dotazu Q a z dotazu Q je vytvorena dodatecná extenzionální informace (zvaná seménko).

Pr. Predpokládejme EDB zadanou relací rodic(R, D), IDB definovanou pravidly

```
predek(P,D) :- rodic (P,D).
predek(P,D) :- rodic(P,X), predek(X,D).
```

a dotaz tvaru

```
?-predek(X, eva).
```

Metodou magických množin získáme ekvivalentní (produkující stejné výsledky) "magický program", který nevyhodnocuje nedotazovaná fakta. Pozustává ze seménka, modifikovaných puvodních pravidel a v tomto konkrétním prípade jediného magického pravidla.

```
m_seed_predek(eva).                                  //seménko
m_predek(A):-m_seed_predek(A).                       //magický predikát
predek(P, D):-m_predek(D), rodic(P, D).              //modifikované pravidlo
predek(P, D):-m_predek(D), rodic(P, X), predek(X, D). //      "       "
```

2.3 Negace

Datalog dovolující použití negativních literálu v telech pravidel je oznacován jako Datalog^{P} (Datalog s negací). Jestliže pravidla obsahují v tele negované podcíle, mohou vzniknout paradoxní situace. jako príklad uvedme jeden z Russellových paradoxu - holic ve meste je osoba, která holí každého, kdo se neholí sám. Vyjádreno logickým programem:

```
osoba(holic).
osoba(starosta).
…
holí(holic, Osoba) :- osoba(Osoba), not(holí(Osoba, Osoba)).
```

Hodnota holí(holic,holic) není ani pravdivá ani nepravdivá. Aby programy s negací byly vyhodnotitelné, vyžaduje se splnení dalších podmínek. Efektivne vyhodnotitelné jsou tzv. stratifikované programy. Program je stratifikovaný, pokud neobsahuje negaci v rekurzi. Napr. rozšíríme-li program definující predka o pravidlo

```
nocykl(X, Y) :- predek(X, Y), not(predek(Y, X)).
```

je výsledný program stratifikovaný. Pri vyhodnocení zdola nahoru bude nejprve vypocten pevný bod predikátu predek a teprve poté bude pocítán predikát nocykl.
Rozšírením stratifikovaných programu je trída lokálne stratifikovaných programu. Program je lokálne stratifikovaný pro danou databázi, jestliže po nahrazení promenných v pravidlech konstantami (všemi možnými zpusoby) je výsledný program stratifikovaný.

Dalším rozšírením je modulární stratifikace. Pro její zavedení je potrebné definovat graf G závislosti predikátu, který bude užitecný i v dalších cástech textu. G obsahuje hranu z uzlu reprezentujícího predikát q do uzlu reprezentujícího predikát p práve tehdy, když v programu existuje pravidlo tvaru

p(…) :- …, q(…), … .

Program je modulárne stratifikovaný, jestliže každá silne souvislá komponenta grafu závislosti predikátu programu je lokálne stratifikovaná po odstranení pravidel s literály, které mají nepravdivou hodnotu v nižší silne souvislé komponente.

3 Prehled deduktivních databázových systému

V prubehu minulých dvaceti let byla implementována celá rada deduktivních databázových systému a publikováno mnoho s nimi souvisejících teoretických prací. Pro zainteresované ctenáre mužeme jako zdroj prohlubujících informací doporucit stránku http://www.acm.org/sigmod/dblp/db/index.html, kde je k breznu 2002

uvádeno 270000 clánku z oblasti DBLP (Database Systems and Logic Programming). V této kapitole uvedeme ty z projektu, které považujeme za nejduležitejší. Zacneme s projekty LDL a NAIL, které patrí k prvním implementacím deduktivních databází, dále podáme prehled o systémech Coral, Aditi, Florid, XSB a ConceptBase.

3.1 LDL a NAIL!

První projekty deduktivních databází se objevují v polovine osmdesátých let. Vývoj systému LDL (Logic Data Language - http://pike.cs.ucla.edu/ldl/) zacal v r. 1984, první prototyp byl vyvinut v r. 1988 a k použití nabídnut o rok pozdeji.

Jazyk LDL podporuje složené termy (mohou mít tvar výrazu z jednoduchých termu) ve faktech i pravidlech, stratifikovanou negaci a zvláštní pozornost venuje množinovým termum. Každé pravidlo s množinovým termem je pri prekladu nahrazeno množinou pravidel s normálními termy. Upravený LDL program je preložen do relacní algebry rozšírené o operaci výpoctu pevného bodu. Exekucním prostredím je algebraický stroj realizující potrebné operace. Relacní výrazy jsou interne reprezentovány "grafem exekuce", což je orientovaný graf jehož uzly korespondují bud databázovým relacím nebo operacím, vetve indikují vztahy mezi operacemi a operandy.

Pokracovatelem LDL projektu je LDL^{++} . Ve verzi 5.1 byl zverejnen v roce 1998. Jeho nejvýznamnejší zdokonalení zahrnují:

- Otevrenou architekturu a integraci s SQL systémy. Je použita architektura založená na interní databázi v operacní pameti a možnosti propojení s ruznými externími SRBD.
- Jazyková rozšírení podporující nemonotonní a nedeterministické programy. Pripouští se i omezená množina nestratifikovaných programu.
- Nový model kompilace a exekuce. Oproti LDL, který byl pro vlastní exekuci prekládán do C, je preklad provádén do struktury nazvané LAM (LDL^{++} Abstract Machine), jejíž uzly jsou C++ trídami. Výpocet je proudove orientovaný. Pro cíl je vypoctena jedna hodnota, která je predána dalším cílum pravidla. Teprve když cíle provedly výpocet pro tuto hodnotu, je pocítána další.

Systém NAIL! (Not Another Implementation of Logic!) byl iniciován v roce 1985 na Stanfordské Univerzite. Jeho pocátecním cílem bylo studium optimalizace vyhodnocování logických programu. Jazyk podporovaný tímto systémem je Datalog rozšírený o funkcní symboly, negaci a množiny. Množiny jsou zavedeny prostrednictvím operátoru findall, jehož sémantika vychází z obdobného Prologovského predikátu. Nejsou zavedeny množinové termy, findall produkuje ve skutecnosti seznamy, i když poradí prvku v seznamu není kontrolováno. Zdrojový kód je nejdríve zpracován preprocesorem, který izoluje not a findall predikáty. Jeho výstupem je program sestávající z pravidel trí typu:

- pravidla, jejichž telo je konjunkcí literálu,
- pravidla, jejichž telo je jeden negovaný literál,
- pravidla, jejichž telo je jeden findall podcíl.

Dále je program preveden do interní formy nazvané ICODE. Ta zahrnuje príkazy relacní algebry spolu s rídícími príkazy. Príkazy ICODE jsou optimalizovány a pak vykonány prostrednictvím interpretu, který je v prípade potreby preloží do SQL.

Revize systému na pocátku devadesátých let doplnila ryze deklarativní jazyk Nail! o procedurální prvky. V systému Glue-Nail jsou logická pravidla vkládána do konvencních jazykových konstrukcí jako jsou cykly, procedury a moduly.

3.2 Aditi

Aditi (viz http://www.cs.mu.oz.au/research/aditi/) je deduktivní systém vyvinutý na University of Melbourne ve spolupráci s Information Research Institute. Cílem projektu byl vývoj implementacní a optimalizacní techniky deduktivních systému, která by umožnila zefektivnit zpracování dotazu na srovnatelnou úroven s relacními databázemi.

Na rozdíl od jiných systému, které k realizaci persistence používají externí databázový stroj, byl Aditi od pocátku koncipována jako integrovaný systém, orientovaný na diskové zpracování. Jeho klient-server architektura dovoluje víceuživatelský provoz. *Front-end* procesy komunikují s uživateli, zpracování dotazu provádí *Query Server*, *Database Acces Processes* zodpovídají za základní databázové operace a *Relational Database Processes* vykonávají operace relacní algebry.

Programy v jazyce Aditi (Aditi-prologu) neobsahují fakta.. Ta jsou uložena pouze v databázových relacích obdobne jako ve standardním databázovém systému. Program v Aditi-prologu je preložen do interní reprezentace a uložen v databázi. Uživatel se pak muže dotazovat na intenzionální predikáty definované programem.

V r. 2001 byla zverejnena alfa verze Aditi 2. Hlavní inovací je zmenené schéma diskové reprezentace databáze, používající interne generované identifikátory souboru. V soucasnosti probíhá reimplementace sítové vrstvy Aditi 2 a zefektivnení klient-server zpracování.

3.3 Coral

Coral (Control Relation And Logic) (viz http://www.cs.wisc.edu/coral/) je projektem University of Wisconsin, Madison. Cílem Coral týmu bylo vytvorit system nabízející širokou škálu vyhodnocovacích a optimalizacních metod, spolu s možností kombinovat logický a imperativní styl programování.

Ve své základní verzi Coral ukládá data pouze v operacní pameti. Je však možná rekompilace, dovolující využívat externí databázové systémy (Sybase, Ingres, Exodus).

Coral podporuje deklarativní jazyk s možností propojení na C++, což dovoluje kombinovat deklarativní a imperativní styl programování. Deklarativní dotazovací jazyk pripouští zápisy ve tvaru datalogovských klauzulí rozšírených o složené termy, množiny, agregaci, negaci a relace s n-ticemi, které obsahují promenné. Program muže být modulárne usporádán. K dispozici jsou i konstrukce pro udate, insert a delete pravidel.

Systém Coralu má implementováno více vyhodnocovacích strategií a automaticky vybírá vhodnou strategii pro každý modul programu. Ovládat optimalizaci dotazu je umožneno také uživateli.

Projekt byl zahájen v r.1988. Soucasne platná verze 1.5.2 byla zverejnena v r.1997.

3.4 FLORID

FLORID (F-Logic Reasoning In Databases - http://www.informatik.uni-freiburg.de/~dbis/florid/) je projektem Univerzity ve Freiburgu. Podstatne se liší od ostatních deduktivních systému. Používá vlastní nový formalismus logického programování nazvaný *F-logic (Frame-logic)*, ve kterém je sdruženo paradigma logického a objektove orientovaného programování.

Florid vyhodnocuje programy zdola nahoru s použitím naivní metody pro výpocet tranzitivního uzáveru relací. Jazyk F-logic pracuje s objekty. Objekty jsou interne reprezentovány jejich identifikátory, které jsou pro uživatele skryty. Uživatelé pracují se jmény objektu (tzv. id-termy). Objekty mohou mít metody a jsou organizovány do tríd. Syntaxe jazyka je relativne komplikovaná, umožnuje však vyjádrit všechny typické vlastnosti objektove orientovaného prístupu pomocí jazyka logického typu. I když F-logic je základním jazykem Floridu, je umožneno používat v programech i zápisy v predikátové logice.

Pokracováním projektu Florid je LoPiX, zamerený na zpracování XML dat., který byl zverejnen v r.2001

3.5 XSB

XSB (http://xsb.sourceforge.net/) je spolecným projektem Computer Science Department, Stony Brook University, ve spolupráci s Katholieke Universiteit Leuven, Universidade Nova de Lisboa, a Uppsala Universitet.. Projekt byl zahájen v r. 1990, jeho soucasná verze 2.5 byla zverejnena v breznu 2002.

XSB používá dva zpusoby vyhodnocování predikátu. Normální zpusob odpovídá prologovskému vyhodnocování shora dolu. Prostrednictvím deklarace muže však uživatel pro zvolené predikáty použít metody zdola nahoru, která je v XSB nazývána *tabled resolution* a dovoluje vyhodnocovat cyklické závislosti, pro které prologovský program nemá konecný výpocet. Vyhodnotitelné jsou i obecne nestratifikované programy, pokud splnují podmínku tzv. modulární stratifikovanosti.

Systém je vybaven rozhraním pro C, Javu, Perl, Oracle a další produkty. Zajímavostí jazyka XSB (HiLog) je možnost konstruovat predikáty vyššího rádu, kdy predikátový symbol muže mít význam promenné. Tím je poskytnut prostredek pro definici generických predikátu. Napr.

```
uzaver (R) (X, Y) :- R(X, Y).
uzaver (R) (X, Y) :- uzaver (R) (X,Z), R(Z,Y).
```

kde uzaver (R) / 2 je predikátem druhého rádu, který pro libovolnou relaci R vyhodnocuje její tranzitivní uzáver.

Další vlastností XSB je možnost rozširování logických programu prostrednictvím knihoven. V podobe XSB modulu je k dispozici objektove orientovaný systém Flora, F-Logic a další.

3.6 ConceptBase

ConceptBase [http://www-i5.informatik.rwth-aachen.de/CBdoc/] je projektem *Aachen University of Technology*. Autori jej predstavují jako multi-uživatelský deduktivní manažer objektu, urcený pro konceptuální modelování, který v sobe sdružuje vlastnosti objektove orientovaných a deduktivních databází.

Projekt byl zahájen v r. 1988, verze 5.23 je z kvetna 2002. ConceptBase je údajne používána ve 150 institucích jak pro výzkumné tak i výukové úcely, címž se stává nejrozšírenejší deduktivní a objektove orientovanou databází.

Z klícových vlastností jmenujme:

- Všechny trídy, meta trídy, instance, atributy, pravidla, omezení a dotazy jsou jednotne reprezentovány jako objekty. Za objekt je dokonce považováno i clenství trídy a je možné je zmenit.
- Jazyk, který ConceptBase implementuje (O-Telos), je založen na Datalogu. Podporuje možnosti vyjadrování v logickém formátu, v podobe sémantických sítí a v podobe rámcu.
- Meta modelovací schopnosti dovolují reprezentovat ruzné modelovací jazyky, jako napr. E-R diagramy. Objekty popsané v jednom jazyce lze propojovat s objekty popsanými jiným modelovacím jazykem.
- ConceptBase respektuje architekturu klient-server. Klientské programy komunikují se serverem prostrednictvím protokolu Internetu. Je k dispozici rozhraní dovolující uživatelum vytváret jejich vlastní klientské programy v jazycích Java, C nebo Prolog.
- Celá báze objektu je umístena ve virtuální operacní pameti, takže pokud rozsah báze objektu presahuje 500 megabytu, neprobíhá zpracování efektivne.

4 Preklad základního Datalogu do SQL v systému EDD

Vztah mezi relacním databázovým systémem a logickým programovacím jazykem vychází ze souvislosti mezi databázovou relací a predikátem s konecným definicním oborem. Predikát s aritou n je funkcí s hodnotami {true, false}, lze jej tedy popsat databázovou relací s n atributy.

Predpokládejme logický program P_L, který definuje množinu predikátu p_1, p_2, …,p_k. Necht?dotaz požaduje úplné vyhodnocení všech predikátu (odpovedi na jiné dotazy budou podmnožinami odpovedi na tento dotaz). Úkolem prekladace je transformovat P_L na program P_R v jazyce Oracle SQL. Po provedení P_R musí databáze obsahovat relace odpovídající predikátum p_1, p_2, …,p_k. Oznacíme-li relace stejnými jmény jako jim korespondující predikáty, pak každá relace p_i, i = 1, 2, …,k musí obsahovat práve ty n-tice, pro které je predikát p_i pravdivý.

Z duvodu úspory místa v databázi, mohou být intenzionální nerekurzivní predikáty vyhodnocovány jako databázové pohledy. Zpusob prekladu je možné volit direktivou prekladace.

4.1 Preklad faktu

Faktem je vždy platné pravidlo tvaru:

jméno_faktu(c_1, c_2, …, c_N).

Argumenty faktu jsou konstanty. Všechna fakta s predikátovým symbolem *jméno_faktu* budou v databázi uložena jako záznamy stejného jména. Argumenty faktu odpovídají atributum relace. Prekladac zjistí jména predikátu, typy jejich argumentu a generuje SQL príkazy pro vytvárení tabulek:

```
CREATE TABLE  jméno_faktu(
    ARG1     typ1,
    ARG2     typ2,
    ARGN     typN
);
```

Protože ve zdrojovém textu nemají argumenty predikátu jména, prekladac je odvodí z jejich poradí. Naplnení tabulek zajistí príkazy, které jsou generované pro každý z faktu:

INSERT INTO jméno_faktu VALUES(c_1, c_2, …, c_N) ;

Prekladac akceptuje i fakta, která jsou mu sdelena v podobe databázových tabulek. Potrebuje pouze informaci o jménech tabulek, jménech atributu a jejich typech. Tyto údaje jsou prekladaci predány deklaracní formulí tvaru:

jméno_tabulky(atr1 typ1, atr2 typ2, …, atrN typN).

4.2 Preklad pravidel

Každý intenzionální predikát je definován jedním nebo nekolika pravidly. V základním Datalogu predpokládáme, že literály z pravých stran jsou bud predikáty deklarované programem nebo porovnávací predikáty (<, =, >, <=, >=, <>) v obvyklé infixové notaci.

Každé pravidlo je preloženo na jeden príkaz SELECT jazyka SQL. Pokud je predikát definován dvema ci více pravidly, je výsledkem sjednocení výsledku jednotlivých SELECT príkazu. Príkazu SELECT je použito pro vložení dat do relace reprezentující predikát nebo pro definici databázového pohledu.

Postup prekladu pravidla na príkaz SELECT je následující:

1. Seznam argumentu SELECT je tvoren jmény, která korespondují argumentum hlavy pravidla
2. Cást FROM je tvorena jmény predikátu, které se vyskytují na pravé strane pravidla. Vyskytuje-li se predikát v tele více než jednou, musí být použito alias pro každý z výskytu.
3. Princip konstrukce WHERE cásti vyjadruje cyklus:

```
FOR p IN prvý .. poslední predikát pravé strany pravidla LOOP
    IF p není vestavený predikát porovnání
        THEN
            FOR a IN prvý .. poslední argument predikátu p LOOP
                IF a je promenná
                    THEN
                        IF a se již dríve vyskytnul jako argument
                            predikátu q v této pravé strane
                            THEN generuj do WHERE cásti podmínku
```

```
                                spojení
                                "p.jméno_sloupce = q.jméno _sloupce";
                        END IF;
                    ELSE generuj do WHERE cásti podmínku spojení
                        "p.jméno_sloupce = a";
                END IF;
            END LOOP;
        ELSE -- p je porovnávací predikát
            IF oba argumenty predikátu p jsou promenné
                THEN generuj do WHERE cásti podmínku
                    "argument_1 porovnání argument_2;
                ELSE --promenná je porovnávána s konstantou
                    generuj "argument porovnání konstanta";
            END IF;
    END IF;
END LOOP;
```

Vyhodnocování predikátu v preloženém programu musí být provádeno v poradí, které respektuje jejich závislosti. Ke stanovení poradí vytvárí prekladac graf závislosti predikátu G. Jestliže zpracovávaný uzel predstavuje nerekurzivní predikát, je do výstupního souboru zapsána definice databázového pohledu nebo databázové tabulky, v závislosti na zvoleném zpusobu prekladu.

Pokud G obsahuje cyklus (v programu se vyskytují rekurzivní pravidla), je nutné vyhodnotit pevný bod predikátu cyklu. Cyklický graf závislostí je prekladacem transformován tak, že každou silne souvislou komponentu nahradí jediným uzlem. Vzniklý acyklický graf G_T je kondenzací puvodního grafu G. Predikáty jsou vyhodnocovány v poradí od listu G_T smerem ke korenu. Predstavuje-li zpracovávaný uzel grafu G_T jeden nebo více rekurzivních predikátu, je do výstupního souboru generována smycka pro výpocet pevného bodu. Smycka pocítá inkrementálne záznamy pro rekurzivní predikáty. Její výpocet koncí, jestliže behem iterace žádné nové záznamy nevznikly. Program smycky je generován v procedurálním rozšírení SQL.

5 Rozšírení Datalogu v systému EDD

Vyjadrovací schopnosti základního Datalogu jsou omezené. Pro využití deduktivního databázového systému v roli znalostního systému byl EDD-Datalog rozšíren o možnosti používat:

- negativní literály,
- agregáty,
- prirazovacími príkazy,
- neurcitost.

5.1 Preklad negativních literálu

Pri prekladu pravidel s negací do SQL, mohou nastat situace:

1. Negativní literál má tvar vestaveného porovnávacího predikátu, napr.

q(X) :- …, p(X), not X < 5, … .

2. Negativní literál je predikátem, který je definován programem, napr.
   q(X) :- …, r(X,Y), not p(Y), … .

Zmeny se promítnou do WHERE cásti príkazu SELECT, do nehož se pravidlo prekládá. V prvém prípade má preložený príkaz tvar

SELECT p0.attr_1 FROM p p0, … WHERE … AND NOT(p0.attr_1<5);

Ve druhém je pravidlo s tvrzením "neexistuje predikát p s hodnotou argumentu Y" preloženo na

SELECT r0.attr1 FROM p p0, r r0, … WHERE … NOT EXISTS
(SELECT * FROM p p1 WHERE p1.attr_1= r0.attr_2);

5.2 Preklad agregacních funkcí

Je prirozené dovolit použití agregátu, které jsou soucástí SQL. Implementace je relativne snadná, pokud jejich výskyt pripustíme jen v hlavách pravidel. Napr. pravidlo

p(Ψ, agreg(Y)) :- …, q(…, Y, …), … .

kde Ψ je vektor grupovaných promenných, Y je agregovaná promenná a agreg reprezentuje agregacní funkci (avg, count, max, min, sum), bude preloženo do tvaru

SELECT …, agreg(q0.attr_k) FROM …, q q0 … WHERE …
GROUP BY atributy reprezentované Ψ.

5.3 Prirazovací príkaz

Použití prirazovacího príkazu je z praktických duvodu užitecné i v deklarativním prostredí deduktivního systému. Jeho implementace je však složitejší než napr. implementace agregacních funkcí. Použití prirazení totiž dovoluje uživateli deklarovat predikáty, které nelze z databázových relací vyhodnocovat zdola nahoru. Program pak muže obsahovat intenzionální predikáty, které nezávisí na faktech. Samozrejme ani samotné prirazení nelze preložit na príkaz SELECT. Pro rešení problému prirazování rozdeluje prekladac pravidla a predikáty do nekolika typu pomocí multigrafu odvozeného z grafu závislosti predikátu.

Pravidlo muže být typu A, B nebo C. Známe-li typy všech pravidel, která definují urcitý predikát, mužeme priradit typ i tomuto predikátu. Typy predikátu oznacíme A, B, C, CA. Výskyt predikátu typu A (A-predikátu) v tele pravidla oznacme jako A-literál. Typy pravidel a predikátu jsou definovány takto:

1. Uvažujme pravidlo π. Jestliže
   a) π je faktem, nebo
   b) telo π neobsahuje prirazení ani B-literál a obsahuje alespon jeden A-literál nebo C-literál, nebo CA-literál, pak π typu A. Jsou-li typu A všechna pravidla definující predikát, pak tento predikát je typu A.
2. Obsahuje-li telo π pouze vestavené porovnávací predikáty nebo prirazení nebo B-literály, pak π je typu B. Jsou-li typu B všechna pravidla definující predikát, pak tento predikát je typu B.
3. Jestliže telo π obsahuje
   a) alespon jeden A-literál nebo C-literál nebo CA-literál a

b) alespon jeden B-literál nebo alespon jedno prirazení,
pak π je typu C. Jsou-li typu C všechna pravidla definující predikát, pak tento predikát je typu C.

4. Je-li predikát definován soucasne pravidly typu A i pravidly typu C, pak predikát je typu CA. B-pravidla nemohou být v definicích predikátu kombinována s jinými typy pravidel.

Preklad A-predikátu:
A-predikáty neobsahují prirazení a nejsou jím ani ovlivneny. Jejich preklad probíhá zpusobem popsaným v kap.4.

Preklad B-predikátu:
B-predikáty jsou intenzionálními predikáty, které nemohou být vyhodnocovány zdola nahoru. V grafu závislosti predikátu nejsou propojeny se žádným z faktu. Proto jsou vyhodnocovány metodou shora dolu, která je soucástí vyhodnocování tech intenzionálních predikátu, jejichž pravidla ve svém tele obsahují B-predikáty.

Pro preklad B-predikátu je nutné použít procedurálního rozšírení SQL a preložit ho do tvaru uložené funkce. Je-li definován *k* pravidly, bude funkce obsahovat konstrukci CASE s *k* vetvemi, jednu vetev pro každé pravidlo.

K vyhodnocení B-predikátu musíme volat jemu odpovídající funkci. Z tohoto duvodu pravidla obsahující ve svém tele B-predikáty musí být rovnež preložena do procedurálního rozšírení SQL. Tato okolnost je duvodem zavedení pravidel typu C.

Preklad C- a CA-predikátu:
K C-predikátum existuje v závislostním grafu z faktu cesta, mohou být tedy vyhodnocovány zdola nahoru. G-pravidla ale používají prirazení nebo závisí na B-predikátech., takže nemohou být prekládána na príkaz SELECT.

Rešením je generovat procedurální SQL príkazy, které vkládají postupne zjištované záznamy do relace reprezentující G-predikát. Takový programový kód muže obsahovat prirazování i funkce reprezentující volané B-predikáty. Protože vyhodnocování je provádeno postupne záznam po záznamu, je použito databázového kurzoru.

CA-predikáty jsou v podstate zvláštním prípadem C-predikátu. Nekterá jejich pravidla jsou typu C, zbývající jsou typu A. Jejich prekladem je proto posloupnost príkazu, z nichž nekteré jsou príkazy jazyka procedurálního rozšírení SQL a jiné jsou tvoreny príkazy SELECT.

5.4 Zpracování neurcitosti v systému EDD

Pro zdokonalení deduktivních schopností a jejich sblížení s realitou našeho sveta, ve kterém není nic jisté, je systém EDD rozšíren o možnost použití fuzzy logiky. Stupen pravdivosti predikátu muže nabývat hodnot reálných císel z intervalu <0..1>. Shodne s terminologií expertních systému nazýváme tuto hodnotu faktorem verohodnosti a oznacujeme CF. Zavedení CF do EDD-Datalogu se odráží v syntaxi pravidel a faktu, která pak mají tvar:

jméno_faktu(<Args>) CF hodnota.
hlava_pravidla((<Args>) :- telo_pravidla CF hodnota.

kde <Args> reprezentuje seznam argumentu. Pokud údaj CF není uveden, predpokládá se jeho hodnota rovna 1. Naše experimenty se soustredily na možnosti implementace Gödelovy, Lukasiewiczovy, Zadehovy a produktové logiky.

Zavedení CF vyžaduje zmeny v procesu prekladu. Relace pro uložení faktu jsou rozšíreny o atribut reprezentující CF. Obdobne i relace reprezentující intenzionální predikáty mají pridaný tento atribut. CF hodnota pravidla je nazývána apriorní verohodností, CF literálu jsou oznacovány jako vstupní verohodnosti pravidla. Výstupní CF, prirazená n-ticím, které jsou výsledkem pravidla je vypoctena jako soucin apriorní verohodnosti a hodnoty (spoctené na základe zadaného typu logiky) ze vstupních verohodností.

Zpusob prekladu datalogovských konstrukcí do SQL ilustrujme na príkladu Gödelovy logiky, jejíž fuzzy konjunkce, fuzzy disjunkce a fuzzy negace jsou definovány následovne:

Necht p, q jsou predikáty a CF(p), CF(q) jejich faktory verohodnosti, pak

CF(p ? q) = min(CF(p), CF(q))
CF(p ∨ q) = max(CF(p), CF(q))
CF(not p) = (if CF(x) = 0 then 1 else 0

Zatím predpokládejme nerekurzivní pravidla, o rekurzi pojednáme pozdeji. Pro úsporu použijme opet oznacení <Args> pro seznam argumentu, napr. zápis
a. <Args1> = b. <Args2> má význam a.arg1 = b.arg1 & a.arg2 = b.arg2 …

1. Konjunkce
   Predpokládejme pravidlo tvaru:
   l(<Args>) :- r1(<Args1>), r2(<Args2>) CF x.
   Vzhledem k sémantice Gödelovy konjunkce bude preloženo na:
   ```
   create view l(<Args>, CF) as
     select r1. <Args11>, r2. <Args21>, GodelovaKonjunkce(r1.CF, r2.CF)*x
         from r1, r2
         where r1. <Args12> = r2. <Args22>;
   ```
   Kde deklarace funkce GodelovaKonjunkce má tvar:
   ```
   function godelovaKonjunkce(X1 real, X2 real) return real is
     begin if X1 < X2 then return X1; else return X2; end if;
     end;
   ```
   <Args11> a <Args21> jsou ty z argumentu r1 a r2, které participují v hlave pravidla. <Args12> a <Args22> reprezentují ty argumenty r1 a r2, které se úcastní na spojení mezi r1 a r2.

2. Disjunkce
   Predpokládejme pravidla:
   l(<Args>) :- r1(<Args1>) CF x.
   l(<Args>) :- r2(<Args2>) CF y.
   Vzhledem k sémantice Gödelovy disjunkce budou preložena na:
   ```
   create view l_tmp(<Args>, CF) as
     select r1. <Args>, r1.CF * x  from r1
           union
     select r2. <Args>, r2.CF * y  from r2;
   create view l(<Args>, CF) as
   ```

select l_tmp. <Args>, max(l_tmp.CF) from l_tmp
group by l_tmp. <Args>;
Pohled l_tmp obsahuje mezivýsledky výpoctu CF.

3. Negace
Predpokládejme pravidlo tvaru:
l(<Args>) :- r1(<Args1>), not r2(<Args2>) CF x.
V souladu se sémantikou Gödelovy negace bude preloženo na:
create view l(<Args>, CF) as
select r1. <Args11>, r1.CF * x from r1
where not exists (select * from r2
where r2. <Args2> = r1. <Args12>;
Kde <Args11> jsou ty z argumentu <Args1>, které jsou obsaženy v <Args>
a <Args12> jsou ty argumenty z <Args1>, které jsou obsaženy v <Args2>.

V prípade, kdy program obsahuje rekurzi, musí být všechny hodnoty relace definované rekurzivním predikátem pocítány iteracne, obdobným zpusobem jako v prípade dvouhodnotové logiky. Pro rekurzivní intenzionální predikáty jsou místo pohledu konstruovány databázové tabulky. Vyhodnocování CF, založené na výše popsaných principech, je pridáno do smycky iteracního výpoctu:

- smycka nejdríve zpracovává pomocnou relaci s príponou tmp,
- behem zpracování pomocné relace se vyhodnocuje CF v souladu s fuzzy konjunkcí,
- následne je zpracována cílová relace s respektováním zásad fuzzy disjunkce,
- smycka se opakuje, pokud jsou do výsledné relace pridávány n-tice.

Vyhodnocují-li se vzájemne rekurzivní predikáty, musí být postupne vypocteny všechny príslušné tabulky (vcetne pomocných) v poradí daném cyklickým grafem závislosti predikátu.

Pro zefektivnení výpoctu v prípade, že dotaz obsahuje nekteré argumenty konstantní, byla zobecnena metoda magických množin [3]. Výsledek zobecnení lze formulovat následovne:

Necht P je program s neurcitostí a P^m jeho magická verze. Jestliže pro vyhodnocení magických pravidel programu P^m bude použita dvouhodnotová logika a pro modifikovaná pravidla magického programu bude použita fuzzy logika, pak P a P^m jsou ekvivalentní programy (ve smyslu produkce stejných výsledku).

6 Záver

V soucasné dobe dle [5] bychom jen težko hledali plne komercne využívaný deduktivní databázový systém. Duvodem je, že techniky vyvinuté pri výzkumu deduktivních DB jsou prebírány a zahrnovány do relacních (objektove relacních technologií). Nelze ani ocekávat, že v dohledné dobe dojde k výrazné zmene. Situace je obdobná jako v prípade objektove orientovaných databází. I zde nové paradigma vyjadrování (na rozdíl od logického se v tomto prípade jedná o objektové) je vstrebáváno relacní technologií.

Deduktivní databáze jsou stálým predmetem zájmu predevším výzkumných a univerzitních pracovišt. Jejich rozvoj má podobu zkoumání možností vyhodnotitelnosti dotazu pracujících s intenzionální informací. Z množství publikovaných odborných prací na toto téma plyne, že problematika dedukce v databázích není uzavrená a nabízí užitecné a zároven i zajímavé úlohy k rešení.

Podekování
Tato práce vznikla za cástecné podpory výzkumného zámeru MSM 235200005. Na vytvorení systému EDD se podílela skupina pracovníku, z nichž bych chtel podekovat zejména Vl. Toncarovi a M. Zímovi.

Literatura

1. Giannotti F., Manco G., Nanni M., Pedreschi D.: Nondeterministic, Nonmonotonic Logic Databases. *IEEE Trans on Knowledge and Data Engineering* 5 (2001) 813-823.
2. Ježek K., Toncar Vl.: Experimental Deductive Database, In *Proc of Workshop on Information Systems Modeling - MOSIS'98* T.Hruška (Ed.), MARQ1998, 83-90.
3. Ježek K. Zíma M.: Magic Sets Method with Fuzzy Logic, Accepted on *ADVIS2002 Int. Conf.*
4. Laks V.S. Lakshmanan, Nematollaah Shiri : A Parametric Approach to Deductive Databases with Uncertainty. *IEEE Trans on Knowledge and Data Engineering* 4 (2001) 554-570.
5. Piattini M., Diaz O.: *Advanced Database Technology and Design*, Artech House, Boston London, 2000.
6. Vojtáš P.: A formal model for fuzzy flexible querying over deductive databases with uncertainty. In *DATAKON 2001*, M.Bieliková (Ed.), STU Bratislava (2001), 169-188.

Annotation:

Deductive Databases and their Implementation in Relational Environment

Deductive databases integrate the capabilities of traditional databases to process a large amount of data and expressive abilities of logic programming. The paper firstly gives an overview of existing deductive systems and secondly it describes an experimental deductive database system (EDD). The EDD system is built on top of a relational database. It uses Datalog as its programming language and is based on the translation of Datalog to Oracle SQL. To increase the expressive power of EDD-Datalog, its extensions include imperative instructions, aggregates and the way of introducing uncertainty into the Datalog program. EDD-Datalog constructions and extensions are presented together with implementation principles.