Katedra Informatiky
Fakulta Matematiky, Fyziky a Informatiky
Univerzita Komenského, Bratislava

Preklad Datalogu do SQL

(Bakalárska práca)

Peter Balko

Vedúci: Dr. Tomáš Plachetka

Bratislava, 2010

Čestne prehlasujem, že som túto bakalársku prácu vypracoval samostatne s použitím citovaných zdrojov.

.....

Poďakovanie

Chcem sa poďakovať vedúcemu mojej bakalárskej práce,
Dr. Tomášovi Plachetkovi, za cenné rady a pomoc
pri písaní tejto práce.

Abstrakt

Názov práce: Preklad Datalogu do SQL
Autor: Peter Balko
Vedúci práce: Dr. Tomáš Plachetka

Táto práca sa zaoberá prekladom Datalogovských programov do SQL dotazov. Prvé kapitoly predstavujú úvod do problematiky Datalogu, príklady programov, syntax Datalogu a vyhodnotenie programov, úskalia rekurzie a negácie. Nasleduje problematika SQL — jednoduché dotazy, syntax SQL a obmedzenia pri generovaní rekurzívnych pravidiel. Ďalej je uvedený algoritmus prekladu Datalogu do SQL, príklady prekladov a na záver popis programu, ktorý algoritmus implementuje.

Kľúčové slová: Datalog, SQL, prekladač.

Obsah

Kapitola 1

Úvod

Deduktívne databázy predstavujú oblasť, v ktorej sa prekrývajú databázy s logikou a logickým programovaním. Deduktívny databázový systém umožňuje definovať a spracovávať pravidlá, na základe ktorých je možné z faktov uložených v databáze odvodzovať ďalšie informácie. Pretože významným teoretickým základom deduktívnych databáz je matematická logika, používa sa pre ne aj označenie logické databázy. Model používaný pre deduktívne databázy je blízky relačnému dátovému modelu, pričom relácia v relačnom modeli zodpovedá predikátu v logickej databáze. Deduktívna databáza pozostáva z faktov (extenzionálnej databázy) a pravidiel (definícií intenzionálnych predikátov). Fakty sú popisované podobným spôsobom ako relácie, nemusia však obsahovať názvy atribútov. Význam atribútov je určený pozíciou v n-tici. Pravidlá sú obdobou relačných pohľadov. Určujú virtuálne relácie, ktoré získame pri aplikácii pravidiel na fakty. Pravidlá môžu byť i rekurzívne, môžu teda predstavovať rekurzívne predikáty, čo súčasné implementácie relačných databáz neumožnujú. Jazyk SQL vo svojej novej norme SQL99 (SQL3) už konštrukciu pre obmedzené rekurzívne dotazy zaviedol, v dostupných systémoch však zatiaľ implementácia rekurzie zahrnutá nie je. Nasledujúce kapitoly zavádzajú potrebné základné pojmy a princípy pre jazyky Datalog a SQL. Nasleduje popis algoritmu prekladu z jazyka Datalog do SQL. Ďalej je popísaná implementácia daného algoritmu v Delphi aplikácii.

Kapitola 2

Základné princípy a pojmy

Za deduktívnu považujeme takú databázu, v ktorej časť poskytovaných informácií nie je explicitne v databáze uložená, ale je získaná odvodením prostredníctvom deduktívnych pravidiel. Prostriedkom pre zápis pravidiel je logický jazyk Datalog (skratka z „database logic“). Datalog je variantom jazyka Prolog. Z Prologu preberá syntaktické konštrukcie a taktiež s Prologom používa základné terminologické pojmy.

Základným predpokladom logického dátového modelu je zobrazenie množiny ako predikátu.

Majme množinu r(A,B,C)={(1,2,4), (1,2,3), (1,1,1)}
(množina s tromi prvkami)
Pozerajme sa na množinu r ako na predikát r(X,Y,Z), ktorý je pravdivý (true)/(nepravdivý(false)) pokiaľ je/(nie je) <X,Y,Z> prvkom z množiny r.

Preto:
r(1,2,4) je pravdivé
r(1,3,3) je nepravdivé

2.1 Datalog

Program Datalogu sa skladá z premenných, konštánt, predikátov a obmedzenej množiny funkčných symbolov. Konjunkcia sa označuje „,“, implikácia „:-“ a negácia „not“. Použijeme definíciu, v ktorej premenné začínajú veľkým písmenom, konštantami môžu byť čísla alebo reťazce a predikátové symboly začínajú malým písmenom.

Definícia 2.1 (Term)**.** *Term je premenná alebo konštanta. (Zložené termy nie*

sú relevantné pre túto pracu.)

Definícia 2.2 (Predikát). *Ak je p predikátovým symbolom a* $t_1, t_2, \ldots, t_N$ *sú termy, potom* $p(t_1, t_2, \ldots, t_N)$ *je predikát a p je názov tohto predikátu.*

Definícia 2.3 (Porovnávací predikát). *Porovnávací predikát má nasledujúci tvar:* $X \Pi Y$.
Pričom X,Y je Term a Π *je z množiny* $\{=, <>, <, >, <=, >= \}$

Definícia 2.4 (Literál). *Literál je buď predikát, negovaný predikát („not p“) alebo porovnávací predikát.*

Definícia 2.5 (Pravidlo). *Pravidlo p kardinality N má tvar:* $p(t_1, t_2, \ldots, t_N)$:- $j_1, j_2, \ldots, j_M$. *Kde* $j_1, j_2, \ldots, j_M$ *sú literály,* $t_1, t_2, \ldots, t_N$ *sú termy.*

Definícia 2.6 (Program). *Program je zložený z pravidiel.*

Definícia 2.7 (Hlava , Telo). *Ľavá strana implikácie* $p(t_1, t_2, \ldots, t_N)$:- $j_1, j_2, \ldots, j_M$. *sa nazýva hlavou a pravá telom pravidla, kde „:-“ označuje symbol implikácie(*$\leftarrow$*).*

Definícia 2.8 (Podcieľ). *Literály* j_i *z tela pravidla* $p(t_1, t_2, \ldots, t_N)$:- $j_1, j_2, \ldots, j_M$. *označujeme ako podciele.*

Definícia 2.9 (Fakt). *Pravidlo* $p(t_1, t_2, \ldots, t_N)$:- $j_1, j_2, \ldots, j_M$., *kde* $M = 0$ *a* $t_1, t_2, \ldots, t_N$ *sú konštanty sa nazýva fakt.*

Program Datalogu je zložený z dvoch častí:

- Extenzionálna Databáza (EDB) obsahuje základné dáta uložené v databáze v podobe faktov (extenzionálne predikáty).
- Intenzionálna Databáza (IDB) obsahuje virtuálne relácie definované prostredníctvom jedného alebo viacerých pravidiel (intenzionálne predikáty).

Pravidlo v Datalogu musí byť buď z IDB, alebo EDB, avšak nie z oboch zároveň.

2.1.1 Vyhodnotenie nerekurzívneho Datalogovského programu

Na nasledujúcom príklade si ukážeme ako sa nerekurzívny program v Datalogu vyhodnotí.

Príklad:

Majme EDB definované týmito faktami:

ľúbi(osoba, pivo).
predáva(bar, pivo, cena).
navštevuje(osoba, bar).

a s týmto IDB pravidlom:

šťastný(X) :- navštevuje(X, Bar), predáva(Bar, Pivo, Cena), ľúbi(X, Pivo).

Program sa vyhodnotí nasledovne:

- Premenné v tele pravidla nahradíme všetkými možnými hodnotami.
- Pre každé takto vzniknuté pravidlo, ak sú všetky podciele pravdivé, potom pridáme hlavu medzi výsledky.

Vyjadrené pomocou relačnej algebry:
šťastný(X)=Π_{osoba} (navštevuje $\bowtie$ ľúbi $\bowtie$ predáva)

Príklad:

Majme pravidlo:
s(X,Y):- r(X,Z), r(Z,Y), not r(X,Y)
kde r={(1,2),(1,3),(2,2),(2,3)}
Prvý podcieľ bude pravdivý pre:

1. X = 1, Z = 2
2. X = 2, Z = 3

V prvom prípade, ak Y = 3, potom aj druhý a tretí podcieľ je pravdivý. Preto pridáme (X,Y)=(1,3) do relácie s.
V druhom prípade, pre žiadnu hodnotu Y nie je druhý podcieľ pravdivý. Preto s={(1,3)}

Príklad:

1. s(X) :- r(Y)
2. s(X) :- not r(X)
3. s(X) :- r(Y), X < Y

V každom príklade by bola množina výsledkov nekonečná (pre ľubovolnú konečnú reláciu r). Pre zmysluplné vyhodnotenie programu musia pravidlá spĺňať podmienky bezpečnosti.

Definícia 2.10 (Podmienky bezpečnosti)**.** *Ak sa premenná vyskytuje:*

- v hlave pravidla
- v negovanom podcieli pravidla
- v podcieli tvaru porovnávacieho predikátu

Potom sa premenná musí vyskytovať aj v pozitívnom podcieli tela tohoto pravidla.

2.1.2 Vyhodnotenie rekurzívneho programu

Najprv si zadefinujeme niekoľko pojmov k objasneniu problematiky.

Definícia 2.11 (Graf závislostí predikátov - G)**.** *G obsahuje hranu z uzla reprezentujúceho predikát q do uzla reprezentujúceho predikát p práve vtedy, keď v programe existuje pravidlo tvaru p(. . .) :- . . . , q(. . .), . . . , alebo p(. . .) :- . . . ,not q(. . .), ...*

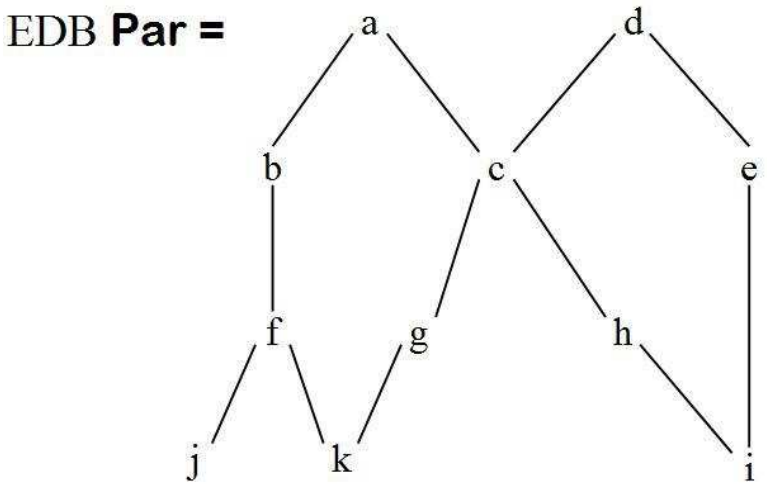
Definícia 2.12 (Rekurzívny Program)**.** *Program obsahuje rekurziu práve vtedy, keď graf závislostí predikátov obsahuje cyklus.*

Príklad:

```
sib(X,Y) :- par(X,P), par(Y,P), X <> Y
cousin(X,Y) :- sib(X,Y)
cousin(X,Y) :- par(X,XP), par(Y,YP), cousin(XP,YP)
```

Ak chceme vyhodnotiť rekurzívny program, postupujeme nasledovne:
(Metóda známa ako výpočet pevného bodu.)

1. Nech v je množina obsahujúca výsledok dotazu rekurzívneho Datalogovského programu.
   (Ide o výsledok dotazu ?- cousin(X, Y).)
2. Na začiatku je v prázdna.
3. Do v pridáme všetky hlavy z pravidiel, ktoré mali všetky podciele pravdivé, po dosadení všetkých možných hodnôt do premenných pravidiel.
4. Zmenila sa množina v ? Ak áno, opakuj 3. bod, ak nie, skonči.
   (Ak opakujeme 3. bod, za možné hodnoty premenných v podcieli cousin(X,Y) už môžeme považovať aj prvky z množiny v)



Obrázok 2.1: Poznámka: hrana z a do b znamená, že (a,b) aj (b,a) je z množiny par

Príklad: Na Obrázku 2.1 (na strane 7) je graficky znázornený EDB predikát par. Zoberieme si program z predchádzajúceho príkladu a vypočítame ho.

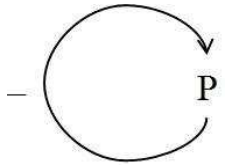
	Sib	Cousin
Inicializácia	∅	∅
Krok 1. pridáme:	(b,c),(c,e) (g,h),(j,k)	∅
Krok 2. pridáme:		(b,c),(c,e) (g,h),(j,k)
Krok 3. pridáme:		(f,g),(f,h) (g,i),(h,i),(i,k)
Krok 4. pridáme:		(k,k) (i,j)

2.1.3 Negácia

Negovaný nerekurzívny program vyhodnotíme podobne ako nerekurzívny program:

- Premenné v tele pravidla nahradíme všetkými možnými hodnotami.
- Pre každé takto vzniknuté pravidlo, ak sú všetky pozitívne podciele pravdivé a negované podciele nepravdivé, potom pridáme hlavu medzi výsledky.

Negácia v rekurzívnom programe nemusí byť na prvý pohľad intuitívna. Aj keď je negácia mimo rekurzie, môže vzniknúť nejednoznačnosť, čo dané pravidlo zna-



Obrázok 2.2: Rekurzia v negácii

mená. My však chceme, aby všetky pravidlá boli jednoznačne určiteľné. Niektoré rekurzívne programy sa pri výpočte zacyklia. Tento problém by mala riešiť stratifikácia negácie. Je to dodatočná podmienka, ktorá musí byť splnená, aby sa výpočet rekurzívneho programu nezacyklil. Nasledujúci príklad uvádza, prečo je negácia v rekurzii problémom.

Príklad:

Majme p(X) :- q(X),not p(X) , kde q = EDB = {1,2}
Pri vyhodnotení IDB predikátu p iteráciou dostaneme:
Inicializácia p = $\emptyset$
Krok 1. p = { 1,2 }
Krok 2. p = $\emptyset$
⋮

Definícia 2.13 (Stratifikovaný Graf)**.** *Graf G_S kde:*

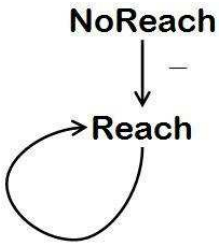
- *Vrcholy sú IDB predikáty*
- *Hrana z p do q existuje práve vtedy, keď q je z tela a p je z hlavy pravidla. Ak q je negované, označ túto hranu „-“.*

Príklad 1: Rekurzia v negácii.

p(X):- q(X), not p(X), graf k danému príkladu je na obrázku 2.2 (na strane 8).

Príklad 2: Rekurzia mimo negácie.
Ktoré cieľové vrcholy (target) nemôžu byť dosiahnuteľné (noreach) zo zdrojových vrcholov (source) ? Stratifikovaný graf k nasledujúcemu programu je na obrázku 2.3 (na strane 9)

reach(X) :- source(X)
reach(X) :- reach(Y), arc(Y,X)
noreach(X) :- target(X), not reach(X)



Obrázok 2.3: Rekurzia mimo negácie

Definícia 2.14 (Výpočet strata pre predikát a)**.** *Pre IDB predikát a vypočítame stratum ako maximálny počet záporných hrán na ľubovolnej ceste z vrcholu a v stratifikovanom grafe.*

Definícia 2.15 (Stratifikovaný program)**.** *Program Datalogu je stratifikovaný, ak každý IDB predikát má konečné stratum.*

Definícia 2.16 (Stratifikovaný model vyhodnotenia programu)**.** *Stratifikovaný program vieme vyhodnotiť tak, že budeme vyhodnocovať predikáty s najnižším stratom ako prvé.*

Príklad: Vyhodnotenie stratifikovaného modelu programu

EDB:
Source = {1}
Arc = {(1,2), (3,4), (4,3)}
Target = {2,3}

Prvé vyhodnotíme reach = {1,2} ($stratum_{reach} = 0$).
Ďalej vypočítame noreach = {3}

2.2 SQL

SQL (Structured Query Language) je najrozšírenejším databázovým jazykom, ktorý sa stal štandardom v oblasti spracovania dát. Začiatky SQL siahajú až do roku 1973, keď programátori IBM pracovali na systéme R a potrebovali zabezpečiť ad-hoc dotazy a schopnosť spracovávať produkčné transakcie. Počas vývoja tohto systému bol vyvinutý nový databázový jazyk, ktorý využíval výhody „Relačného dátového modelu“. Tento jazyk dostal názov SEQL (Structured English Query Language), neskôr sa kvôli zákonu na ochranu obchodného mena zmenil na SQL. Po prvý krát sa jazyk komerčne implementoval v roku 1979 vo firme ORACLE Corporation (vtedy Relational Software, Inc.) Nová generácia

štandardu SQL bola vo vývoji od roku 1990 a je známa ako SQL99.

Samotný jazyk SQL má nasledujúce časti:

- jazyk pre manipuláciu s dátami (Data Manipulation Language) umožňuje definovať dotazy, vkladať, rušiť a modifikovať dáta
- jazyk pre definíciu dát (Data Definition Language) podporuje vytváranie, rušenie a modifikáciu objektov databázy (tabuľky, pohľady)

Jayzk SQL patrí do triedy neprocedurálnych jazykov, ktorý popisuje, čo vyberáme z databázy, ale nie ako to vykonáme. Avšak komerčné systémy ako ORACLE, Informix alebo PostgreSQL ho implementujú s procedurálnymi rozšíreniami.

Procedurálne rozšírenia SQL

Rozširujú funkčnosť SQL o:

- lokálne premenné, CASE vetvenia ; IF, THEN, ELSE spolu s ELSEIF umožňujú výber na základe hodnoty premennej, dotazu,...
- vytváranie procedúr, funkcií, a ich volania pomocou CALL
- cykly LOOP spolu s WHILE umožňujú opakovať blok SQL príkazov, pokiaľ je splnená podmienka.

2.2.1 Data Manipulation Language

Sú to príkazy slúžiace na manipuláciu s dátami v databáze. Základné príkazy DML:

- INSERT vkladanie záznamov do tabuľky
- DELETE rušenie záznamov v tabuľke
- UPDATE modifikácia záznamov v tabuľke
- SELECT výber (vyhľadávanie) záznamov v tabuľke

Príkaz SELECT

Všeobecný tvar príkazu je SELECT... FROM... WHERE... .Podrobnejšie:

- SELECT zoznam atribútov, atribúty možno premenovať pomocou AS, ak sú relácie premenované, môžeme atribúty určiť jednoznačne
- FROM zoznam relácií (resp. kartézsky súčin, resp. join)
- WHERE selekčná podmienka (obsahuje spájacie podmienky). Za WHERE sa môžu nachádzať:
  - mená atribútov
  - porovnávacie operátory: =, <>, <, >, <=, >=
  - aritmetické operátory: +, -, *, /
  - logické spojky: NOT, AND, OR
  - porovnanie regulárnych výrazov: s LIKE p
  - špeciálne funkcie pre dátum a čas
- GROUP BY zoznam grupovacích atribútov, môžeme grupovať iba atribúty z SELECT časti
- HAVING selekčná podmienka v grupovanej relácii
- nakoniec môžeme dať ORDER BY Atribút1 DESC, Atribút2 ASC, čo usporiada výsledok podľa stĺpcov prislúchajúcich k atribútom 1. zostupne (DESC), 2. vzostupne (ASC).

Príklad: premenovania relácií s jednoznačne určenými atribútmi

produkt (meno, cena, kategória, výrobca)

SELECT P1.výrobca, P2.výrobca FROM produkt AS P1, produkt AS P2 WHERE P1.kategória = P2. kategória AND P1. výrobca <> P2. výrobca;

Príkaz SELECT: zjednotenie, prienik, rozdiel

Intuitívne ide o množinové operácie. Avšak namiesto množín sa pracuje s výsledkami SELECT dotazov.

Príklad:

(SELECT meno FROM osoba WHERE mesto = 'Trnava')

UNION
(SELECT meno FROM osoba, kontrakt WHERE kupujúci = name AND sklad = ‘Zohor’); Podobne sa používa INTERSECT (prienik) a EXCEPT (rozdiel).

Vyhodnotenie SELECT dotazu

1. Urobí sa kartézsky súčin relácií za FROM
2. Na ten kartézsky súčin sa aplikuje selekčná podmienka za WHERE (ktorá odfiltruje riadky, ktoré danú podmienku nespĺňajú)
3. Nakoniec sa urobí projekcia na atribúty, ktoré sú za SELECT (čo odfiltruje stĺpce, ktoré nie sú v projekcii)

Agregačné funkcie

SQL obsahuje viacero agregačných funkcií: SUM, MIN, MAX, AVG, COUNT (a ďalšie). S výnimkou COUNT, všetky agregačné funkcie sa aplikujú na jeden atribút.

Príklad:

produkt (meno, cena, kategória, výrobca)
kontrakt (predávajúci, kupujúci, sklad, výrobok)

SELECT sum(cena) FROM produkt WHERE výrobca = ’Vinárske Závody’;

SELECT count(*) FROM Kontrakt;

Väčšinou chceme aplikovať agregačné funkcie na konkrétne atribúty databázy (stĺpce tabuľky).

Príklad: Dotaz na objem predaja jednotlivých výrobkov.

SELECT výrobok, sum(cena) FROM produkt, kontrakt WHERE produkt.meno = výrobok GROUP BY výrobok HAVING count(kupujúci) $>$ 100

Vyhodnotenie SELECT dotazu s GROUP BY a HAVING

1. SELECT dotaz sa vyhodnotí ako bez GROUP BY a HAVING.
2. Výsledok sa usporiada podľa atribútov za GROUP BY.

3. Aplikuje sa agregačná funkcia na grupované atribúty.

4. Za HAVING nasledujú ďalšie podmienky, ktoré musia byť splnené.

Výsledok vyššie spomenutého SELECT dotazu môže obsahovať len:

1. grupované atribúty (atribúty v GROUP BY)

2. agregáty (výsledky agregačných funkcií)

2.2.2 Data definition Language

Ide o príkazy na definíciu schémy databáz. Podrobne:

- Vytvorenie tabuľky, pohľadu
- Odstránenie tabuľky, pohľadu
- Modifikácia schémy tabuľky

Príklad: *Vytvorenie tabuľky*

CREATE TABLE Osoba(meno VARCHAR(30), rodné_číslo INTEGER, vek SHORTINT, mesto VARCHAR(30), pohlavie BIT(1), dátum_narodenia DATE);

Príklad: *Odstránenie tabuľky*

DROP Osoba;

Príklad: *Modifikácia tabuľky*

ALTER TABLE osoba ADD telefón CHAR(16);
ALTER TABLE osoba DROP vek;

Pohľady (CREATE VIEW) sú dotazy zapamätané v databáze. Sú užitočné pri vyhodnocovaní zložitých dotazov. Umožňujú, aby ich prostredníctvom rôzni užívatelia videli tabuľky rôzne.

Príklad: *Vytvorenie pohľadu:*

produkt (meno, cena, kategória, výrobca)
kontrakt (predávajúci, kupujúci, sklad, výrobok)
podnik (meno, adresa, okres, cena)
osoba (meno, rodné_číslo, telefón, mesto)

CREATE VIEW nákupy_elektroniky AS SELECT výrobok, kupujúci, predávajúci, sklad FROM kontrakt, produkt WHERE kontrakt.výrobok = produkt.meno AND produkt.kategória = 'elektronika';

Príklad: *Odstránenie pohľadu*

DROP VIEW nákupy_elektroniky;

V norme SQL99 je definovaná možnosť zadať rekurzívny dotaz. Avšak v existujúcich systémoch zatiaľ nie je implementovaná. Preto prípadnú rekurziu treba riešiť pomocou procedurálneho rozšírenia ORACLE PL/SQL.

Príklad: *Rekurzívny dotaz*

ancestor(predok, potomok)
parent(rodic,potomok)

WITH RECURSIVE ancestor AS ((SELECT * FROM parent)
UNION
(SELECT * FROM parent P, ancestor A WHERE P.potomok=A.predok))

Kapitola 3

Preklad Datalogu do SQL

Vzťah medzi relačným databázovým systémom a logickým programovacím jazykom vychádza zo vzťahu medzi databázovou reláciou a predikátom s konečným definičným oborom. Predikát s n argumentami je funkcia s hodnotami {true, false}. Dá sa teda popísať databázovou reláciou s n atribútmi.
Predpokladajme logický program P_L, ktorý definuje množinu predikátov $p_1, p_2, \ldots, p_k$. Nech P_L? dotaz požaduje úplné vyhodnotenie všetkých predikátov (odpovede na iné dotazy budú podmnožinou odpovedí na tento dotaz). Úlohou prekladača je transformovať P_L na program P_R v jazyku Oracle SQL. Po vyhodnotení P_R musí databáza obsahovať relácie zodpovedajúce predikátom $p_1, p_2, \ldots, p_k$. Označme relácie rovnakými názvami ako k nim prislúchajúce predikáty, potom každá relácia p_i, i = 1, 2, . . . ,k musí obsahovať práve tie n-tice, pre ktoré je predikát p_i pravdivý. Z dôvodu úspory miesta v databáze môžu byť intenzionálne nerekurzívne predikáty vyhodnocované ako databázové pohľady.

3.1 Preklad faktov

Faktom je vždy platné pravidlo tvaru:

Príklad: názov_faktu($c_1, c_2, \ldots, c_N$).

Argumenty faktov sú konštanty. Všetky fakty s predikátovým symbolom názov_faktu budú v databáze uložené ako záznamy rovnakého mena. Argumenty faktu odpovedajú atribútom relácie. Pri preklade faktov potrebujeme názov predikátu, typ jeho argumentov a môžeme generovať SQL príkazy pre vytvorenie tabuliek:

CREATE TABLE názov_faktu (
ARG_1 typ_1,
ARG_2 typ_2,
⋮

ARG_N typ_N

);

Pretože v zdrojovom texte nemajú argumenty predikátov meno, odvodíme ho z ich poradia ($ARG_1, ARG_2, .., ARG_N$). Zistiť typ argumentu je však z predikátu nemožné. Za týmto účelom by sme potrebovali pomocný vstupný súbor. Pre testovacie účely by mohlo stačiť každý atribút nastaviť na VARCHAR(100) (pole znakov dĺžky 100). Naplnenie tabuliek zabezpečia príkazy, ktoré sú generované pre každý jeden z faktov názov_faktu:

INSERT INTO názov_faktu VALUES($c_1, c_2, \ldots, c_N$) ;
Preklad faktov nie je algoritmicky náročný a čitateľ si ho vie zo spomenutého príkladu jednoducho predstaviť. Preto je preklad faktov spomenutý iba v texte práce, avšak nie je implementovaný v Delphi aplikácii.

3.2 Preklad pravidel

Každý IDB predikát je definovaný jedným alebo viacerými pravidlami.
Pri vyhodnocovaní Dataloguu predpokladáme, že literály z pravých strán sú buď predikáty deklarované programom, alebo porovnávacie predikáty
(<, =, >, <=, >=, <>) v zaužívanej infixovej notácii. Každé pravidlo
je preložené na jeden príkaz SELECT jazyka SQL. Pokiaľ je predikát
definovaný dvoma alebo viacerými pravidlami, je jeho výsledkom zjednotenie (UNION) výsledku jednotlivých SELECT príkazov. Príkazy SELECT
sú použité pre uloženie dát do relácie reprezentujúcej predikát alebo
pre definíciu databázového pohľadu.

Postup prekladu pravidla na príkaz SELECT je nasledujúci:

1. Zoznam argumentov SELECT je tvorený menami, ktoré korešpondujú s argumentom hlavy pravidla
2. Časť za FROM je vytvorená názvami predikátov, ktoré sa vyskytujú na pravej strane pravidla. Ak sa vyskytuje predikát v tele viac ako raz, musí byť použité alias (premenovanie) pre každý jeden z výskytov.
3. Princíp konštrukcie WHERE časti vyjadruje nasledujúci pseudokód:

```
FOR p IN prvý .. posledný predikát pravej strany pravidla LOOP
  IF p nie je predikát porovnania
    THEN
      FOR a IN prvý .. posledný argument predikátu p LOOP
        IF a je premenná
          THEN
            IF a sa už skôr vyskytla ako argument predikátu q
            v tejto pravej strane
              THEN generuj do WHERE časti podmienku spojenia
                "p.názov_stĺpca = q.názov_stĺpca";
            END IF;
          ELSE generuj do WHERE časti podmienku spojenia
            "p.názov_stĺpca = a";
        END IF;
      END LOOP;
    ELSE -- p je porovnávací predikát
      IF obidva argumenty predikátu p sú premenné
        THEN generuj do WHERE časti podmienku
          "argument1 porovnávací_predikát argument2";
        ELSE --premenná je porovnávaná s konštantou
          generuj "argument porovnávací_predikát konštanta";
      END IF;
  END IF;
END LOOP;
```

Vyhodnocovanie predikátov v preloženom programe musí byť robené v poradí, ktoré rešpektuje ich závislosti. K stanoveniu poradia treba vytvoriť graf závislostí predikátov G. Predikáty sú prekladané smerom od koreňov (EDB predikáty) k listom. Pretože spracovávaný uzol predstavuje nerekurzívny predikát, preložíme ho ako databázový pohľad.

3.3 Preklad rekurzívnych pravidiel

Pokiaľ G (graf závislostí predikátov) obsahuje cyklus, potom sa v programe vyskytuje rekurzívne pravidlo. Norma SQL99 povoľuje zadávať rekurzívne dotazy.

Avšak v žiadnom existujúcom systéme nie su implementované. Preto by sme preklad rekurzívneho pravidla museli robiť pomocou procedurálneho rozšírenia SQL. Cyklický graf závislostí transformujeme tak, že každý silno súvislý komponent nahradíme jediným uzlom (novým predikátom). Vzniknutý acyklický graf G_T je ekvivalentný pôvodnému grafu G. Každý uzol G_T predstavuje jedno pravidlo programu. Programy sú vyhodnotené v poradí od listov G_T smerom ku koreňu. Ak si predstavíme spracovávaný uzol grafu G_T, jeden alebo viac rekurzívnych pravidiel v pôvodnom grafe G teraz patrí do jedného uzla. Cyklus počíta inkrementálne záznamy pre rekurzívne predikáty. Jeho výpočet končí, pokiaľ pri iterácii žiadne nové záznamy nevznikli. Program cyklu by sme generovali v procedurálnom rozšírení SQL (štandardný jazyk SQL neobsahuje príkazy na generovanie cyklov). V Delphi aplikácii som algoritmus na preklad rekurzívnych predikátov do procedurálneho rozšírenia SQL neimplementoval, pretože implementácia horeuvedeného algoritmu má veľa praktických výnimiek. A na druhej strane výsledný cyklus v procedurálnom rozšírení SQL sa vôbec intuitívne nepodobá na pôvodný program. Preto preklad rekurzívnych predikátov uvediem iba v prekladovej časti ako teoretickú možnosť.

3.4 Preklad negatívnych literálov

Pri preklade pravidiel s negáciou do SQL, môžu nastať tieto situácie:

1. Negatívny literál má tvar zloženého porovnávacieho predikátu, napr.

   $$\text{q(X) :- } \ldots, \text{ p(X), not(X} < 5), \ldots \ .$$

2. Negatívny literál je predikátom, ktorý je definovaný programom, napr.

   $$\text{q(X) :- } \ldots, \text{ r(X,Y), not p(Y) , } \ldots \ .$$

Zmeny sa premietnu do WHERE časti príkazu SELECT, do ktorého sa pravidlo prekladá. V prvom prípade má preložený príkaz tvar:

$$\text{SELECT } p0.attr_1 \text{ FROM p p0, } \ldots \text{ WHERE } \ldots \text{ AND } NOT(p0.attr_1 < 5);$$

V druhom prípade je pravidlo s tvrdením „neexistuje predikát p s hodnotou argumentu Y“ preložené na:

$$\text{SELECT r0.attr1 FROM p p0, r r0, } \ldots \text{ WHERE } \ldots \text{ NOT EXISTS}$$
$$\text{(SELECT * FROM p p1 WHERE } p1.attr_1 = r0.attr_2);$$

3.5 Preklad agregačných funkcií

Je prirodzené dovoliť použitie agregátov, ktoré sú súčasťou SQL.
Napr. pravidlo:

$$\text{p}(Y_1,\text{A}) \text{ :- } \ldots, \text{subtotal}(\text{q}(Y_1,\ldots,Y_m),[Y_1],[\text{agreg}(Y_m,\text{A})]),\ldots.$$

Kde Y_1 je premenná, podľa ktorej grupujeme (môže ich byť viac), Y_m je agregovaná premenná a agreg reprezentuje agregačnú funkciu (avg, count, max, min, sum).
Čo preložíme do tvaru:

SELECT ..., agreg($q0.attr_m$) FROM ..., q q0 ... WHERE ... GROUP BY Y_1.

Kapitola 4

Príklady prekladov

Nižšie uvedené príklady sú preložené pomocou algoritmov spomenutých v predchádzajúcej kapitole, preto vo väčšine príkladov nie je uvedený komentár, ako by sa konkrétny príklad riešil krok za krokom. Ako EDB databázu budeme používať tieto predikáty:

dodavatel(firma, mesto).
dodava(firma, vyrobok, cena, lehota).
objednavky(meno, vyrobok).
capuje(krcma, alkohol).
lubi(pijan, alkohol).
navstivil(idn, pijan, krcma, od, do).
vypil(idn, alkohol, mnozstvo).

4.1 Preklad jednoduchého pravidla.

Všetky dodávateľské firmy z Bratislavy.

dodavatel_ba(F) :- dodavatel(F,‘ Bratislava‘).

SELECT d.firma FROM dodavatel d WHERE d.mesto=´Bratislava´;

4.2 Preklad jednoduchého pravidla s dvoma podcieľmi.

Kto dodáva niečo z toho, čo si Jožo objednal.

dodava_nieco_jozovi(F) :- objednavky(jozo, V),dodava(F, V, _, _).

SELECT D.firma FROM objednavky B, dodava D WHERE B.meno=´jozo´ AND B.vyrobok=D.vyrobok ;

4.3 Preklad pravidla bez premennej v hlave.

Toto je špeciálny prípad. Datalog by takýto dotaz vyhodnotil true, ak existuje aspoň jedna firma v databáze dodávateľ.

q :- dodavatel(X).

SELECT EXISTS (SELECT d.firma FROM dodavatel d);

4.4 Preklad pravidla bez tela.

Podobne ako v predchádzajúcom príklade, aj tu by Datalog na podobný dotaz odpovedal existenciou danej relácie.

dodavatel.

SELECT EXISTS (SELECT * FROM dodavatel);

4.5 Preklad jednoduchého programu.

Každé pravidlo programu sa preloží do SELECT dotazu, ktoré sa následne spoja (UNION). Kto dodáva výrobok pc1 buď do 4 dní, alebo do 15000 korún.

dodava_pc1_do4_alebo_do15000(F) :-dodava(F, pc1, C, _), C<=15000.
dodava_pc1_do4_alebo_do15000(F) :-dodava(F, pc1, _, L), L<=4.

SELECT D.firma FROM dodava D WHERE D.vyrobok=´pc1´ AND D.cena<=15000;
UNION
SELECT D.firma FROM dodava D WHERE D.vyrobok=´pc1´ AND D.lehota<=4;

4.6 Preklad programu s negovaným podcieľom.

Kto nedodáva niečo Jožovi.

nedodava_nieco_jozovi(F) :- objednavky(jozo, V),dodavatel(F, _), not dodava(F, V, _, _).

SELECT D.firma FROM objednavky B, dodavatel D WHERE B.meno=´jozo´ AND
NOT EXISTS (SELECT * FROM dodava DD WHERE DD.firma=D.firma AND DD.vyrobok=B.vyrobok);

4.7 Preklad programu s IDB podcieľom.

Doteraz sme nemali podcieľ, ktorý nebol z extenzionálnej databázy (EDB). Preto pre preklad tohto programu nebude stačiť iba SELECT dotaz, ale bude treba intenzionálny podcieľ definovať ako pohľad nad databázou (CREATE VIEW).Kto dodáva všetko Jožovi.

nedodava_nieco_jozovi(F) :- dodavatel(F,_),objednavky(jozo, V), not dodava(F, V, _, _).
dodava_vsetko_jozovi(F) :- dodavatel(F, _), not nedodava_nieco_jozovi(F).

CREATE VIEW nedodava_nieco_jozovi AS (SELECT D.firma FROM dodavatel D, objednavky B WHERE B.meno=´jozo´ AND NOT EXISTS (SELECT * FROM dodava DD WHERE DD.firma=D.firma AND DD.vyrobok=B.vyrobok));

SELECT D.firma FROM dodavatel D WHERE NOT EXISTS (SELECT * FROM nedodava_nieco_jozovi DD WHERE D.firma=DD.firma);

4.8 Preklad programu s agregáciou.

Pri preklade programu s agregáciou pribudne GROUP BY časť do dotazu SELECT. Aký druh alkoholu sa vypil v akom množstve v každej krčme(stats—predikát).

vypilo_sa_pri_navsteve(K, A, M, I) :-navstivil(I, _, K, _, _), vypil(I, A, M).

stats(K, A, M) :- capuje(K, A),
subtotal(vypilo_sa_pri_navsteve(K, A, X, I), [K, A],[sum(X, M)]).

CREATE VIEW vypilo_sa_pri_navsteve AS (SELECT A_a.Krcma A_b.Alkohol A_b.Mnozstvo A_a.Idn FROM navstivil A_a , vypil A_b WHERE A_a.Idn = A_b.Idn;)

CREATE VIEW stats AS (SELECT A_a.Krcma A_a.Alkohol SUM(A_b.Mnozstvo) FROM capuje A_a , vypilo_sa_pri_navsteve A_b WHERE A_a.Krcma = A_b.Krcma AND A_a.Alkohol = A_b.Alkohol GROUP BY A_a.Krcma , A_a.Alkohol;

)

SELECT * FROM stats;

4.9 Preklad programu s rekurziou

Nasledujúci program je rekurzívny, pretože má cyklus v grafe závislostí predikátov. Keby bola implementovaná rekurzia zo štandardu SQL99, program by sa dal preložiť jednoducho:

ancestor(X, Y) :- parent(X,Y).
ancestor(X, A) :- parent(X, Y), ancestor(Y, A).

WITH RECURSIVE ancestor AS (
(SELECT * FROM parent)
UNION
(SELECT * FROM parent P, ancestor A WHERE P.Y=A.X))

Pre súčasné implementácie SQL si pomôžeme procedurálnym rozšíreným ORACLE PL/SQL.

```
CREATE OR REPLACE TABLE ancestor(
X VARCHAR,
Y VARCHAR,
);

CREATE OR REPLACE TABLE ancestor_EDB(
X VARCHAR,
Y VARCHAR,
);

a INTEGER=0;
b INTEGER=0;

INSERT into ancestor(X,Y) SELECT * FROM  parent;

SELECT COUNT(*) INTO a FROM ancestor(X,Y);

SELECT COUNT(*) INTO b FROM ancestor_EDB(X,Y);
```

Hore uvedené riadky SQL kódu predstavujú inicializáciu výpočtu pevného bodu.

```
WHILE a<>b LOOP

  INSERT into ancestor_EDB(X,Y)
  (SELECT * FROM ancestor)
INTERSECT
  (SELECT * FROM ancestor_EDB);

  INSERT into ancestor(X,Y)
  SELECT * FROM ancestor_EDB E,parent P WHERE P.Y=E.X;

  SELECT COUNT(*) INTO a FROM ancestor;
  SELECT COUNT(*) INTO b FROM ancestor_EDB;

END LOOP;

SELECT * FROM ancestor;
```

Kapitola 5

Implementácia

Ako programovací jazyk na implementáciu som si zvolil Delphi 6, ako mne najbližší programovací jazyk. Delphi program budem ďalej nazývať iba aplikácia, aby nedošlo k zámene s programom Datalogu.
Aplikácia ako vstup používa dva textové súbory:

- dotazy.txt - Tento súbor obsahuje Datalogovské programy, ktoré chceme preložiť. Predpokladáme, že každý program je bezpečný a nerekurzívny. Užívateľ si sem môže písať:
  - Datalogovské programy (aj viac programov oddelených prázdnymi riadkami)
  - komentár k programom, a to jednoriadkový začínajúci znakom % alebo viacriadkový obalený do /* */
  - Súbor musí byť ukončený „.“ posledného Datalogovského programu, za ktorou nesmie nasledovať prázdne miesto. („space“)
- databazy.txt - V tomto súbore musia byť napísané EDB predikáty, pre programy napísane v prvom súbore, pričom každý musí byť napísaný na práve jeden riadok. Medzi dvoma predikátmi nesmie byť voľný riadok a predikát musí mať tvar:

  $$\mathrm{p}(a_1,\ldots,a_n)$$

  kde p je názov predikátu, ktorý sa súčasne berie ako názov relačnej databázy(SQL) a a_i je názov i-teho atribútu danej databázy.

Príklad súboru dotazy.txt

```
% test ignorovania prveho commentu
vypilo_sa_pri_navsteve(K, A, M, I) :-navstivil(I, _, K, _, _), vypil(I, A, M).
/*
```

```
ignoracia druheho kommentara

cez viac riadkov
*/

stats(K, A, M) :- capuje(K, A),
subtotal(vypilo_sa_pri_navsteve(K, A, X, I), [K, A],
[sum(X, M)]).
```

Príklad súboru databazy.txt

```
capuje(Krcma, Alkohol)
lubi(Pijan, Alkohol)
navstivil(Idn, Pijan, Krcma, Od, Do)
vypil(Idn, Alkohol, Mnozstvo)
```

Pre lepšie pochopenie činnosti aplikácie si uvedieme niekoľko ďalších pojmov:
Token - je atomická (nedeliteľná) časť jazyka, môže to byť špecifický symbol ako napr.: „sum“ alebo iba jeden znak „.“
Syntax - časť gramatiky zaoberajúca sa korektnými vetnými stavbami (čo patrí do jazyka a čo nie).
Sémantika - vnútorný význam slov v jazyku.
Parser - je nástroj na kontrolu syntaxe jazyka a generovanie abstraktnej štruktúry dát z kontrolovaného textu.

5.1 Gramatika akceptujúca Datalog

Delphi aplikácia používa nasledujúcu gramatiku na:

- Rozpoznanie tokenov zo vstupného súboru („dotazy.txt“).
- Syntaktickú kotrolu.

<Prolog> -> <Statmt> | <Statmt> <Prolog> | <Comment>.
<Statmt> -> <Fact> <dot> | <Fact> <:-> <Body> <dot>.
<Fact> -> <ATOM> <LftBr> <Arglist> <RgtBr> | <ATOM>.
<Body> -> <ruleGOAL> | <ruleGOAL> <comma> <Body>.
<ruleGOAL> -> {<not>}<ATOM> <LftBr> <arglist><RgtBr>|<SUBTOTAL>|<GOAL>.
<Arglist> -> <G1>{<comma> <arglist> }.
<G1> -> <Var> | <ATOM> | <FLOAT>.
<SUBTOTAL> -> <subtotal> <LftBr> <ruleGOAL2> <comma> <Sarglist> <RgtBr>.
<ruleGOAL2> -> <ATOM> <LftBr> <arglist> <RgtBr>.

<Sarglist> -> <s1> <comma> <s2>.
<s1> -> <LftB2> <arglist> <RgtB2>.
<s2> -> <LftB2><Func><LftBr><var><comma><var><RgtBr><Rgtb2> <comma> <s2>.
<GOAL> -> <G1> <other><G1> {<comma> <GOAL>}.
<ATOM> -> <lcchar>{<anychar>} | '{<anychar>}'.
<VAR> -> <ucchar>{<anychar>} | '_'{<anychar>}.

Nasledujúce pravidlá reprezentujú tokeny Datalogu.

<FLOAT> = {'-'|'+'}[0-9]+{'.'[0-9]+}.
<other> = '<=' | '=>' | '~' | '=' | '<' | '>'.
<lcchar> = [a-z].
<ucchar> = [A-Z].
<anychar> = [a-z,A-Z,0-9,'_','+','-']*.
<comment> = '%'| <anychar> | '#10#13' | '/*' [a-z,A-Z,0-9,'#10#13']* '*/'
<dot> = '.'.
<:-> = ':-'.
<RgtBr> = ')'.
<LftBr> = '('.
<RgtB2> = ']'.
<LftB2> = '['.
<comma> = ','.
<not> = 'not' | ' +'.
<subtotal>= 'subtotal'.
<Func> = 'sum' | 'max' | 'min' | 'count' | 'avg'.

Aplikácia po spustení pracuje nasledovne:

1. Načíta sa súbor databazy.txt, tento súbor sa číta priamo do poľa reprezentujúceho EDB predikáty (relačnú databázu SQL),
2. pomocou Parser-a sa načíta súbor dotazy.txt,
3. parser na vstupe rozpozná TOKENY jazyka Datalogu,
4. ktoré sa syntakticky skontrolujú,
5. pričom sa sémantické informácie ukladajú do dátovej štruktúry.
6. Údaje získané v dátovej štruktúre sa po prečítaní celého vstupného súboru začnú spracovávať na generovanie SQL dotazov.
7. Pri generovaní SELECT dotazu, nebudú názvy atribútov pomenované: $attr_1$, ...,$attr_n$, ale budú nahradené menami atribútov získaných z databázy.txt

8. Všetky pravidlá prekladá ako pohľady do databázy, pre prípad, že by sa na ne niektoré ďalšie pravidlo odvolávalo.

9. Preložené pravidlá sa zobrazia v textovej časti aplikácie, čo je výstupom programu.

Kapitola 6

Záver

Táto práca má za cieľ oboznámiť čitateľa s algoritmom prekladu Datalogovských programov do SQL dotazov. V úvode sa čitateľ môže oboznámiť s problematikou deduktívnych databáz a tiež s problematikou jazykov Datalog a SQL. Ďalej je popísaný algoritmus prekladu Datalogu do SQL. Na precvičenie algoritmu sú uvedené príklady prekladov. Súčasťou práce je aplikácia bežiaca pod MS Windows, ktorá implementuje preklad nerekurzívnych Datalogovských programov do SQL. Jej implementácia je spomenutá v poslednej kapitole. Ak táto práca pomohla čitateľovi osvojiť si problematiku prekladu Datalogu do SQL, tak splnila svoj účeľ.

Literatúra

[1] Tomáš Plachetka, Úvod do databázových systémov Dotazovacie jazyky,

`http://www.dcs.fmph.uniba.sk/~plachetk/TEACHING/DB2006/db2006_2.pdf`

[2] Tomáš Plachetka, Úvod do databázových systémov Základné časti SQL,

`http://www.dcs.fmph.uniba.sk/~plachetk/TEACHING/DB2006/db2006_3.pdf`

[3] Tomáš Plachetka, Úvod do databázových systémov Agregácia a rekurzia,

`http://www.dcs.fmph.uniba.sk/~plachetk/TEACHING/DB2006/db2006_4.pdf`

[4] Karol Matiaško a kol., Databázové systémy,Databázové technológie a aplikácie,EDIS - vydavateľstvo ŽU,2008, ISBN 978-80-8070-821-4

[5] 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.