ДЕДУКТИВНЕ БАЗЕ ПОДАТАКА
DEDUCTIVE DATABASES

Боривоје Милошевић, *Висока техничка школа струковних студија - Ниш*
Ненад Стојановић, *Висока техничка школа струковних студија - Ниш*

Садржај – *Системи управљања релационим базама података (RDBMS) су веома успешни код административне обраде података. Међутим, у задње време потребе све сложенијих апликација постављају нове проблеме, и ограничења ових система постају све уочљивија. На пример, компанија која производи аутомобиле може имати потребу за одговором на упит: „Колико би нас коштала производња новог модела по садашњим ценама компоненти?“ или, „Да ли имамо довољно компоненти на стању, за производњу нашег новог модела?“ Овакви упити се не могу изразити у SQL-92. Они захтевају много моћнији релациони језик – Даталог. У њему је могуће постављати рекурзивне упите, што значи стављати табелу у релацију са самом собом.*

Abstract – *Relational database management systems (RDBMS) are very successful in administrative data processing. However, lately the needs of increasingly complex applications set new problems, and limitations of these systems become more apparent. For example, a company that produces cars may need to answer the question: "How much would cost the production of new models at the current prices of components?" Or, "Do we have enough components in stock, for the production of our new model?" Such queries cannott be expressed in SQL-92. They require a much more powerful relational language - Datalog. It is possible to set recursive queries, which means putting a table in a relationship with itself.*

1. УВОД:

Настанак дедуктивних база података јавио се као производ две врло егзактне и заокружене области: база података, које су тежиле да на теоријски заснованом релационом моделу уведу ефикасан, нередудантан и брз начин манипулације и одржавања великих количина података, и са друге стране, логичког програмирања чији је главни представник био Пролог.

На почетку, неопходно је увидети да постоји доста сличности између ова два модела и начина на који представљају податке. Са друге стране, они су компламентарни у следећем: релациони модел је супериорнији када је независност података у питању, паралелно процесирање и секундарни уређаји складиштења, док је пролог супериоран када је у питању експресивност и омогућава да ова комбинација постигне и могућност апликативног развоја , што SQL сам за себе није имао.

Функтори у Прологу нису били у стању да преузму улогу упитног и апликативног језика код релационих база, јер се испоставило да способност рекурзије и изражавања комплексних структура података не доприносе раду са релацијама. Тако да се рестрикована верзија Пролога, која не садржи функторе и прилагођена је раду са дедуктивном базом података зове Даталог. Али и у конетксту Даталога се појавило неколико проблема у раду са rdbms-ом који се могу описати као:

1. даталог може да изрази врло комплексне структуре, што није случај код релација у рдбмс-у,
2. како убацити објектну парадигму у дедуктивне базе,
3. како извршити ефикасне операције ажурирања кроз Даталог, јер су код Пролога те ствари биле ниског приоритета итд.

Даталог је дакле језик за рад са дедуктивним базама података, настао повезивањем релационих база података и логичког програмирања (Пролог). Цела дедуктивна база података представља се у облику програма записаног у Даталог језику. Дедуктивни модел израчунава вредност логичких израза (истина или неистина) или тражи вредности за које је логички израз истинит, чињенице и правила су део језика, тј. нема разлике између података и језика за њихову манипулацију.

Даталог у том случају служи као средство за формулацију аксиома. На основу тога поставља се основна разлика између Даталога и релационог језика, јер Даталог подржава рекурзивне аксиоме, а самим тим и рекурзивне упите. Представља се у облику компактног формата записа правила која замјењују хиљаде редова неког другог програмског језика.

ПРИМЕР: породично стабло
Претпоставке: женско, мушко, родитељ
Аксиоми: мајка, отац, ћерка, син, бака, деда, ...
мајка(X,Y):= женско(X), родитељ(X,Y).

Дедуктивне DBMS примењујемо на постојеће базе конструисане на стандардни начин. Дедуктивни системи база података подрзавају доказно-теоретски облик база, дедукују нове податке из постојећих користећи правила премиса – претпоставка, и силогизама – аксиома:
Дедукција: извођење закључака из премиса у логичким системима јасно је показано познатим примером:

"Сви људи су смртни.
Сократ је човек.
Сократ је смртник."

У таквом погледу, база података се посматра тако да садржи комбинацију *extensional* база података и *intensional* база података при чему extensional база података садрзи базне аксиоме, а intensional база података садржи интегритет ограничења и дедуктивне аксиоме.

Аксиоми за задату базу података (доказно-теоретски поглед) могу бити представљени експлицитно, спецификацион скупа неопходних додатних аксиома, да би били у могућности да применимо доказне технике:

» базних аксиома, које одоварају вредностима у домену и н-торке у базним релацијама. Ови аксиоми граде оно што понекад зовемо extensional database,

» "аксиоми о комплетности" изведени за сваку релацију, која изјављује неуспех или у супротном успех појављивања н-торке у релацији

» аксиоми са "јединственим именом", код којих свака константа има уникатно име

» аксиоми "затвореног домена", код којих не постоји константа ван домена базе података

Дакле, дедуктивне базр пружају основу за лакше решавање проблема са којима релациони системи имају проблема, нпр. "Набављач S6 је из Београда или Ниша", или рекурзивним упитима.
Рекурзивни упити представљају једно од најзначајнијих својстава дедуктивних база података.

2. ПРОБЛЕМ РЕЛАЦИОНЕ АЛГЕБРЕ

Упите над табелама, које представљају хијерархијску структуру чланова неког скупа, немогуће је исказати упитом релационе алгебре, односно применом стандарда SQL-92.

Ово се може илустровати једноставним примером. Уколико имамо табелу која представља компоненте неког склопа, и желимо да сазнамо које су подкомпоненте одређене инстанце склопа (нпр. трицикла), морамо да извршимо онолико релационих упита над табелом, колико има нивоа хијерархије за ту инстанцу.

Table - dbo.склоп | NENAD-PC\...\Projects\dbp.sql | Summary

компонента	подкомпонента	количина
трицикл	точак	3
трицикл	оквир	1
оквир	седиште	1
оквир	педала	1
точак	жица	2
точак	гума	1
гума	бандаш	1
гума	унутрашња гума	1
NULL	NULL	NULL

Сл. 1.1 *Табела и одговарајућа хијерархија склопа*

За инстанцу трицикл имамо три нивоа хијерархије (сл. 1.1). Одговарајући упит над табелом *склоп* би била унија упита на сваком нивоу хијерархије.

Први ниво би био директни упит над табелом *склоп* (сл. 1.2).

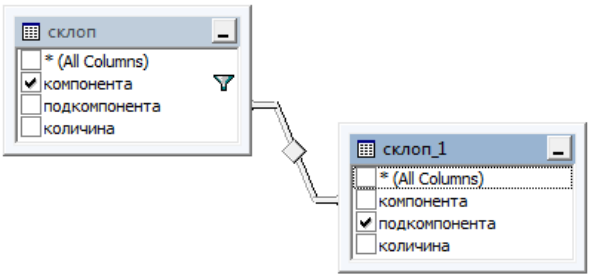
```
--први ниво компоненти
SELECT   компонента, подкомпонента
FROM     склоп
WHERE    (компонента = N'трицикл')
```

Results | Messages

	компонента	подкомпонента
1	трицикл	точак
2	трицикл	оквир

Сл. 1.2 *Упит првог нивоа хијерархије*

За добијање компоненти другог нивоа потребно је да направимо релацију табеле над самом собом (сл. 1.3).



Сл 1.3 *Релација другог нивоа хијерархије*

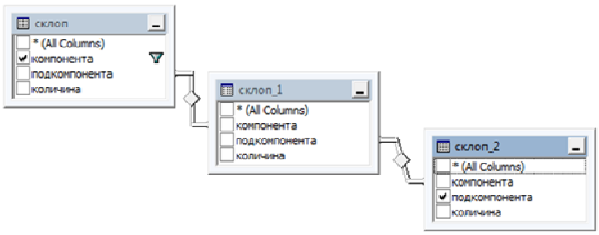
Извршавањем одговарајућег упита добићемо компоненте другог нивоа хијерархије склопа (сл. 1.4).

```
--други ниво компоненти
SELECT    склоп.компонента, склоп_1.подкомпонента
FROM      склоп INNER JOIN
                  склоп AS склоп_1 ON склоп.подкомпонента =
    склоп_1.компонента
WHERE     (склоп.компонента = N'трицикл')
```

	компонента	подкомпонента
1	трицикл	седиште
2	трицикл	педала
3	трицикл	жица
4	трицикл	гума

Сл. 1.4 *Упит другог нивоа хијерархије*

Задњи ниво хијерархије захтева релацију табеле са претходно дефинисаном релацијом другог нивоа (сл. 1.5).

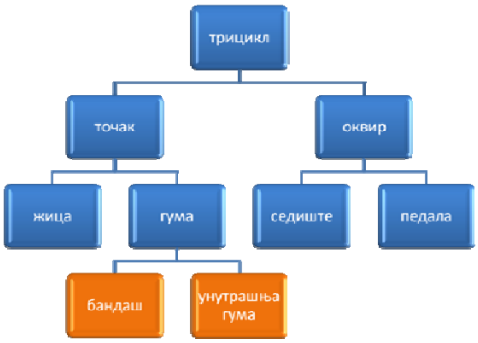



```
--трећи ниво компоненти
SELECT    склоп.компонента, склоп_2.подкомпонента
FROM      склоп INNER JOIN
                  склоп AS склоп_1 ON склоп.подкомпонента = склоп_1.компонента
    INNER JOIN
                  склоп AS склоп_2 ON склоп_1.подкомпонента = склоп_2.компонента
WHERE     (склоп.компонента = N'трицикл')
```

Сл. 1.5 *Релација и одговарајући упит трећег нивоа*

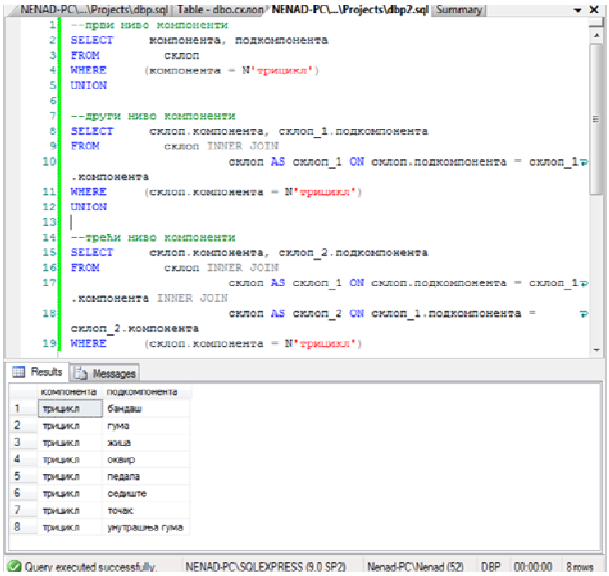
Резултат овог упита су два записа трећег нивоа хијерархије приказана на сл. 1.6, а комплетан упит са резултатима на сл. 1.7.

Међутим, за инстанцу точак, број хијерархијских нивоа би био за 1 мањи, а самим тим и број одговарајућих `SELECT` упита. Ограничење SQL-92 је да немамо могућност дефинисања упита за произвољну инстанцу склопа. Овај недостатак је превазиђен применом Даталог програма, чије су функционалности имплементиране у SQL:1999.



	компонента	подкомпонента
1	трицикл	бандаш
2	трицикл	унутрашња гума

Сл. 1.6 *Резултат упита трећег нивоа*



Сл. 1.7 *Комплетан упит који представља унију три упита различите дубине хијерархије*

3. ДАТАЛОГ ПРОГРАМ

Даталог нам пружа могућност да упит поставимо путем логичких исказа, онако како нам је природно разумљиво. Дефинишемо правила која морају бити задовољена, и то минимално потребан број правила. У

нашем конкретном случају то су два правила, чија синтакса је следећа:

Прво правило:

```
Компоненте(Компонента, Подкомпонента):-
      Склоп(Компонента, Подкомпонента).
```

Друго правило:

```
Компоненте(Компонента, Подкомпонента):-
        Склоп(Компонента, Компонента2),
Компоненте(Компонента2, Подкомпонента).
```

Релација *Компоненте* идентификује компоненте сваког дела склопа.

Прво правило се може прочитати као:
За све вредности *Компонента, Подкомпонента*,
Ако постоји запис (*Компонента, Подкомпонента*) у *Склоп*,
Онда мора постојати запис **(***Компонента, Подкомпонента*) у *Компоненте*.

Друго правило се може прочитати као:
За све вредности *Компонента, Компонента2, Подкомпонента*,
Ако постоји запис (*Компонента, Компонента2*) у *Склоп* **и** запис (*Компонента2, Подкомпонента*) у *Компоненте*,
Онда мора постојати запис (*Компонента, Подкомпонента*) у *Компоненте*.

Део са десне стране симбола :- се назива тело (**body**) правила, а део са леве стране се назива заглавље (**head**) правила. Симбол :- означава логичку импликацију:

Ако запис наведен у телу правила постоји у бази, то имплицира постојање записа наведеног у заглављу правила. Због тога, ако имамо сет записа *Склоп* и *Компоненте*, свако правило се може користити за добијање (дедукцијом) нових записа који припадају скупу *Компоненте*. Због тога, системе који подржавају Даталог правила, називамо дедуктивним системима база података (**deductive database systems**).

Свако правило је шаблон за извођење закључака. Додељивањем константи променљивим у правилу, можемо добити специфичне записе скупа *Компоненте*. На пример, додељивањем *Компонента=трицикл*, *Компонента2=оквир*, и *Подкомпонента=седиште*, можемо закључити да је (*трицикл;седиште*) у *Компоненте* (сл. 2.1). Посматрајући сваки запис у *Склоп*, на основу првог правила закључујемо да је сет записа добијен пројекцијом *Склоп* на своја прва два поља у *Компоненте*.

Друго правило нам дозвољава да комбинујемо претходно испитане записе у *Компоненте* са записима у *Склоп*, и да на основу тога добијемо нове записе у *Компоненте*. Можемо применити друго правило узимајући у обзир одговарајући производ у *Склоп* и (текућој инстанци) у *Компоненте*, и додељујући вредности променљивим из правила за сваки запис производа, један по један ред. Уочавамо како понављање коришћења променљиве *Компонента2* спречава одређене записе да стварају нове записе; у ствари, она (*Компонента2*) представља релацију између *Склоп* и *Компоненте*.

Компонента	подкомпонента
трицикл	точак
трицикл	оквир
оквир	седиште
оквир	педала
точак	жица
точак	гума
гума	бандаш
гума	унутрашња гума

Сл. 2.1 *Додељивање константи променљивим у правилу*

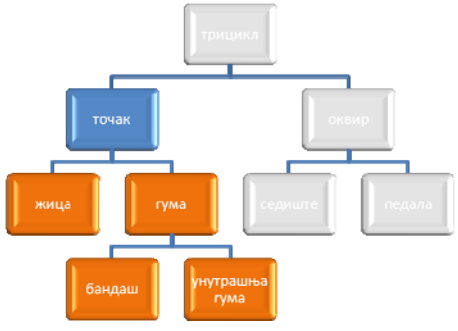
Наведена Даталог правила се могу исказати у SQL:1999. Овакав рекурзивни упит нам даје могућност да изаберемо инстанцу склопа (нпр. точак) и да добијемо одговарајуће компоненте (сл. 2.2).

```
USE DBP;
WITH компоненте(компонента,подкомпонента)
AS
(
   SELECT A1.компонента,A1.подкомпонента FROM склоп A1
   UNION ALL
   SELECT A2.компонента, C1.подкомпонента
   FROM склоп A2, компоненте C1
   WHERE A2.подкомпонента = C1.компонента
)

SELECT * FROM компоненте C2
WHERE C2.компонента = N'точак'
```

Results | Messages

	компонента	подкомпонента
1	точак	жица
2	точак	гума
3	точак	унутрашња гума
4	точак	бандаш



Сл. 2.2 *Рекурзивни упит и резултујући сет записа*

ЛИТЕРАТУРА

[1] R. Ramakrishnan and J. Gehrke, *Database Management Systems*, Boston: WCB/McGraw-Hill 1998.

[2] D.Maier, *The Theory of Relational Databases*, Rockville, Md.: Computer Science Press 1983.

[3] Microsoft, *Recursive Queries Using Common Table Expressions*, MSDN Library 2011.

[4] Silberschatz−Korth−Sudarshan, *Database System Concepts, Fourth Edition* , Copyright (c) by Foxit Software Company, 2004, Edited by Foxit PDF Editor

[5] R. Abbott and H. Garcia-Molina. *Scheduling real-time transactions: a performance evaluation*. ACM Trans

[6] E. Anwar, L. Maugis, and U. Chakravarthy. *A new perspective on rule support for object-oriented databases*. In Proc. ACM SIGMOD Conf. on the Management of Data, 1993.

[7] K. Apt, H. Blair, and A. Walker. *Towards a theory of declarative knowledge. In Foundations of Deductive Databases and Logic Programming.* J. Minker (ed.), Morgan Kaufmann, 1988.

[8] C. Beeri, S. Naqvi, R. Ramakrishnan, O. Shmueli, and S. Tsur. *Sets and negation in a logic database language (LDL1).* In ACM Symp. on Principles of Database Systems, 1987

[9] M. Derr, S. Morishita, and G. Phipps. *The glue-nail deductive database system: Design, implementation, and evaluation.* VLDB Journal, 3(2):123{160, 1994

[10] I. Mumick, H. Pirahesh, and R. Ramakrishnan. *Duplicates and aggregates in deductive databases.* In Proc. Intl. Conf. on Very Large Databases, 1990.

[11] K. Ramamohanarao. *Design overview of the Aditi deductive database system.* In Proc. IEEE Intl. Conf. on Data Engineering, 1991.

[12] K. Sagonas, T. Swift, and D. Warren. *XSB as an e_cient deductive database engine.* In Proc. ACM SIGMOD Conf. on the Management of Data, 1994.