УДК 004.655

Буй Д. Б, Поляков С. А.

КОМПОЗИЦІЙНА СЕМАНТИКА РЕКУРСИВНИХ ВИРАЗІВ ТА ЇХНІХ УЗАГАЛЬНЕНЬ В SQL-ПОДІБНИХ МОВАХ

У статті розглянуто метод моделювання ієрархічних структур даних у вигляді списків суміжності. Наведено приклади таких списків та їхні типи. Розглянуто методи побудови навігаційних запитів до ієрархічних структур даних, в тому числі за допомогою загальних табличних виразів CTE у рекурсивній формі. Наведено приклади таких запитів. Задано формальну семантику рекурсивних загальних табличних виразів.

Ключові слова: композиційна семантика, рекурсивні запити, SQL, загальні табличні вирази, CTE.

Загальні зауваження

Мова SQL надає потужні та елегантні засоби для маніпулювання табличними структурами даних, які по суті є мультимножинами рядків [1]. Операції SQL переважно набагато ефективніші при виконанні запитів над таблицями, ніж виконання тих самих запитів засобами традиційних мов програмування. При цьому SQL-запити простіші у програмуванні. Це пояснюється високим рівнем операцій SQL, орієнтованих на роботу з мультимножинами на відміну від традиційних мов програмування, які потребують поелементної обробки мультимножин. Наприклад, в SQL існують аналоги стандартних теоретико-множинних операцій, таких як перетин, об'єднання та різниця множин.

Але платою за такий високий рівень операцій є більш обмежені функціональні можливості мови SQL порівняно із традиційними мовами програмування. Існують класи запитів до табличних структур даних, які не можуть бути реалізовані засобами лише SQL. В той же час такі запити реалізуються традиційними мовами (програмування). Одним із прикладів є запити до ієрархічних структур даних [2-4]. На сьогодні розроблено декілька методів відображення ієрархічних структур у таблиці. Головне питання полягає в наявності операцій високого рівня для маніпулювання ієрархічними даними, зокрема для побудови так званих рекурсивних запитів. У перших версіях SQL ці можливості були відсутні. Тому маніпулювання такими даними програмувалося засобами процедурних розширень SQL і мало відрізнялося від програмування на класичних мовах. Тільки в стандарті SQL:99 було введено розширення, яке допускає рекурсивні запити [3]. В подальшому це розширення мови було втілено в таких провідних СУБД як DB2 та MS SQL Server. В той же час компанія Oracle не підтримала цей стандарт і має своє розширення для написання рекурсивних запитів [5].

Приклад ієрархічної структури даних і рекурсивного запиту для її опрацювання

Ієрархічні структури можуть бути представлені в таблицях кількома методами. Найбільш поширеним є використання так званих списків суміжності, про які піде мова далі. Розглянемо таку класичну ієрархічну структуру, як бюрократична організація, що будується за принципом керівник-підлеглий. Кожен підлеглий має лише одного керівника, а кожен керівник має багато підлеглих. Визначимо таблицю співробітників Employees.

Перші два атрибути цієї таблиці EmployeeId і ManagerId використовуються для завдання ієрархічних зв'язків на рядках. Атрибут EmployeeId містить унікальний ідентифікатор працівника і є первинним ключем таблиці. Атрибут ManagerId містить ідентифікатор керівника цього співробітника і є зовнішнім ключем, який посилається на атрибут EmployeeId в цій таблиці. Якщо у співробітника немає керівника, тобто він є керівником вищого рівня, то атрибут ManagerId має особливе значення NULL. Інші атрибути таблиці містять персональну інформацію про співробітника: його ім'я (FirstName) та прізвище (LastName), посаду (Title), дату народження (BirthDate), дату прийому на роботу (HireDate), місто (City) та область (Region), де він працює, адресу (Address). Фактично атрибути EmployeeId і ManagerId утворюють зв'язок типу один до багатьох на рядках однієї таблиці.

Такі ієрархічні відношення ще називають відношеннями предок-нащадок. Наведений вище приклад списку суміжності демонструє найпростіший, або базовий тип ієрархії, коли кожен на-

Таблиця 1. **Таблиця співробітників Employees**

EmployeeId	ManagerId	LastName	FirstName	Title	BirthDate	HireDate	City	Region	Address

Таблиця 2. **Таблиця родинних зв'язків FamilyTree**

PersonId	FatherId	MotherId	LastName	FirstName	BirthDate	City	Region	Address

щадок має лише одного предка. Зазвичай у кожного нащадка може бути декілька предків.

У теорії графів списку суміжності з кількістю нащадків не більше ніж *n* відповідає орієнтований ациклічний граф з валентністю вершин не більше *n*. (Зазвичай валентністю вершини називається кількість ребер, інцидентних вершині [6]). Для наведеного вище прикладу валентність вершин не перевищує одиницю.

Наступним прикладом є генеалогічне дерево, яке містить родинні зв'язки між людьми. У кожної людини є батько та матір. Таким чином, кожний нащадок має не більше двох предків. Для завдання ієрархічних зв'язків такого типу використовуються так звані дуальні списки суміжності. В теорії графів їм відповідають ациклічні орієнтовані графи з валентністю вершин не більш ніж два. Якщо валентність вершини дорівнює двом, то існує інформація про батька та матір людини; якщо валентність дорівнює одиниці, то інформація про одного з батьків відсутня; якщо ж, нарешті, валентність вершини дорівнює нулю, то відсутня інформація про обох батьків. З огляду на скінченність в будь-якому випадку в такому графі повинна бути хоча б одна вершина з нульовою валентністю.

Таблиця, яка задає генеалогічне дерево, наведена нижче. Перші три атрибути використовуються для встановлення дуальних зв'язків між рядками таблиці.

Атрибут PersonId містить, як і раніше, унікальний ідентифікатор людини, тобто є первинним ключем. Атрибути FatherId та MotherId посилаються на батька та матір людини відповідно. Кожен із них є зовнішнім ключем та посилається на один і той самий атрибут PersonId. Інші атрибути таблиці містять, як і раніше, прізвище (LastName) та ім'я (FirstName) людини, дату народження (BirthDate), місто (City), область (Region) та адресу проживання (Address) відповідно.

Далі розглянемо так звані навігаційні запити, тобто запити, в умові фільтрації яких використовується зв'язок предок-нащадок. Наприклад, потрібно знайти усі вузли, які є нащадками заданого вузла. Найпростішими навігаційними запитами є такі, коли треба знайти всіх синів вказаного вузла. Наприклад, треба знайти всіх працівників, керівником яких є співробітник з ідентифікаційним кодом «010101».

```
SELECT Employees.EmployeeId, Employees.
    LastName,
    Employees.FirstName, Employees.Title,
        Manager.LastName,
    Manager.FirstName
FROM Employees INNER JOIN Employees
    Manager
    ON Employees.ManagerId = Manager.
        EmployeeID
WHERE Manager.EmployeeID = «010101»
```

Звернімо увагу, що таблиця Employees поєднується сама з собою. Для того, щоб уникнути колізії імен, другий екземпляр таблиці перейменовується за допомогою псевдоніма Manager.

У наведеному прикладі треба перейти на один рівень нижче від заданого вузла. Аналогічним чином будуються запити, коли треба перейти на будь-яку наперед задану кількість рівнів.

Але коли кількість рівнів, які треба обійти, невідома заздалегідь, то запит не можна побудувати стандартними засобами оператора SELECT. Наприклад, нехай потрібно знайти всіх нащадків якоїсь людини. Природно, людина вважається дитиною певної особи, якщо її FatherId чи MotherId дорівнює PersonId цієї особи. Для цього можна використати курсори SQL та стандартні ітеративні методи обходу генеалогічного дерева, наприклад, обхід у глибину. Але коли даних багато, використання курсорів не є ефективним.

Більш ефективним є метод, коли створюється допоміжна тимчасова таблиця. На першому кроці туди записується лише один рядок з тією людиною, нащадки якої шукаються. Далі виконується ітеративна процедура поповнення таблиці. На і-му кроці ітерації в таблицю додаються всі нащадки, які знаходяться на і-му рівні глибини відносно свого спільного предка. Це робиться одним оператором SELECT. Процес закінчується, коли оператор SELECT повертає порожню таблицю.

Кількість операторів SELECT дорівнює кількості рівнів в ієрархічній структурі. Так, якщо є мільйон вузлів, розташованих на ста рівнях, то буде виконано лише сто операторів. Тобто множинно-орієнтований підхід діє набагато ефективніше попереднього процедурно-орієнтованого. Але він все ще вимагає створення процедур і виконання кількох операторів. Якщо процедура

виконується не на сервері, а на комп'ютері клієнта, то щоразу потрібно звертатися до сервера, а це знижує продуктивність.

У стандарті SQL:99 був введений новий оператор – так званий загальний табличний вираз CTE (Common Table Expression), який у своїй рекурсивній формі дозволяє побудувати потрібний результат без звернення до процедурного коду.

Вирази CTE у своїй нерекурсивній формі подібні звичайним представленням (view) зі сферою дії, обмеженою одним запитом. На них можна дивитися як на підзапити, відокремлені від основного запита та мають своє ім'я. Такий підзапит можна використовувати в декількох місцях основного запита. Він починається з ключового слова WITH, за яким йде ім'я запиту та перелік параметрів. Після CTE починається головний запит, який має посилання на CTE (виклик CTE). Далі наведено загальну структуру запиту з CTE.

```
/*CTE*/
WITH CTEName (parameters)
AS (Simple Subquery)
/* головний запит*/
SELECT...
FROM CTEName
```

Рекурсивна форма CTE має складніший вигляд. Вона складається з двох операторів SELECT, які поєднані операцією UNION ALL (підкреслимо, що мова йде саме про операцію UNION ALL; нижче буде показано, що замінити операцію UNION ALL на UNION не можна, адже головна відмінність між цими операціями полягає в тому, що операція UNION ALL будує таблицю з дублікатами рядків, а операція UNION – без дублікатів).

Родинне дерево будується для людини, ідентифікатор якої PersonId дорівнює 10. Після фрази WITH вказується назва таблиці, яка будується за допомогою CTE, та список її атрибутів. Атрибут Level є обчислюваним і містить рівень, на якому розташовані нащадки відповідного рівня людини, для якої будується запит. Так, сама людина має рівень 1, її діти мають рівень 2 і т.д.

Перший оператор SELECT задає ініціальну таблицю. Для цього прикладу вона буде складатися тільки з одного рядка, який відповідає вказаній людині. Другий оператор SELECT формує результуючу таблицю рекурсивним (згідно з іншою термінологією, рекурентним або індуктивним) чином.

```
WITH Tree(LastName, FirstName, PersonID,
    Level)
AS (/*початковий стан таблиці*/
    SELECT LastName, FirstName, PersonID, 1
    FROM FamilyTree
    WHERE PersonID = 10
    /*рекурсивний виклик*/
    UNION ALL
    SELECT Node.LastName, Node.FirstName,
        Node.PersonID, X.Level + 1
    FROM FamilyTree Node
    JOIN Tree X
    ON Node.MotherID = X.PersonID OR Node.
        FatherID = X.PersonID
    )
/*головний запит*/
SELECT PersonID, LastName, FirstName, Level
FROM Tree
```

Семантика рекурсивних запитів

Перейдемо до опису семантики оператора CTE. Таблиці будемо позначати символами t, можливо з індексами, функції на таблицях (тобто функції, область визначення та область значення яких є множина таблиць) будемо позначати символами f, g, теж можливо з індексами.

Нехай задані унарна функція $g : T \to T$ та бінарна функція $f : T \times T \to T$. CTE-вираз уточнюється за допомогою бінарної композиції C, яка функціям g, f ставить у відповідність нову функцію $C(g, f)$; значення останньої функції на аргументі t знаходиться наступним чином.

Будуємо послідовність таблиць $t_1, t_2, t_3, \ldots, t_i, \ldots$, де

$t_1 = g(t)$,
$t_2 = f(t_1, t)$,
$t_3 = f(t_2, t)$,
…
$t_i = f(t_{i-1}, t)$,
…

Якщо на деякому кроці $n + 1$, де $n = 1, 2, \ldots$, отримуємо порожню таблицю, причому усі попередні таблиці послідовності є непорожніми, то процес обчислень закінчується з результатом $t_1 \cup_{ALL} t_2 \cup_{ALL} \ldots \cup_{ALL} t_n$.

В іншому випадку (тобто всі таблиці t_i, $i = 1, 2, \ldots$ непорожні) значення $C(g, f)(t)$ покладається невизначеним.

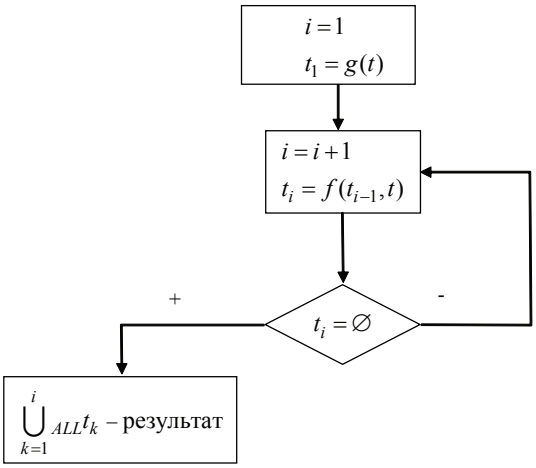
Вище операція $\cup_{ALL}$ – операція об'єднання мультимножин із врахуванням дублікатів [1, підрозд. 3.4, с. 153] (згідно з іншою термінологією, ця операція називається операцією додавання мультимножин [7-9]).

Нижченаведена діаграма ілюструє процедуру знаходження значень функції, яка будується композицією C.

Вказана процедура знаходження значень функції $C(g, f)$ модифікується очевидним чином на випадок часткових функцій.

Підкреслимо ще раз, що операція об'єднання є об'єднанням мультимножин із врахуванням дублікатів. Характеристична властивість цього

об'єднання полягає у тому, що коли обидві таблиці-аргументи не є порожніми, то результат відрізняється від аргументів (навіть якщо один аргумент «вкладається» в іншій). Точніше кажучи, мова йде про наступне бінарне відношення над мультимножинами $\alpha \lhd \beta \overset{def}{\Leftrightarrow} O(\alpha) \subseteq O(\beta) \wedge \forall x(x \in O(\alpha) \Rightarrow \alpha(x) \le \beta(x))$. Тут $O(\alpha)$, $O(\beta)$ – основи мультимножин α, β відповідно [1, підрозд. 3.4, с. 151]. Це бінарне відношення є частковим порядком, воно перетворює сім'ю мультимножин у решітку (подробиці дивись у [8-10]).



Мал. 1. Блок-схема процедури знаходження значень вигляду $C\,(g,f)\,(t)$

Вкажемо другий спосіб обчислення значення $C\,(g,f)\,(t)$, який в деяких випадках є більш зручним. А, саме: будуються дві послідовності таблиць $t_1, t_2, \ldots$, та $t_1, t_2, \ldots$,:

$$t_1' = g(t), t_1 = g(t),$$

$$t_2' = f(t_1', t), t_2 = t_1 \cup_{ALL} t_2',$$

…

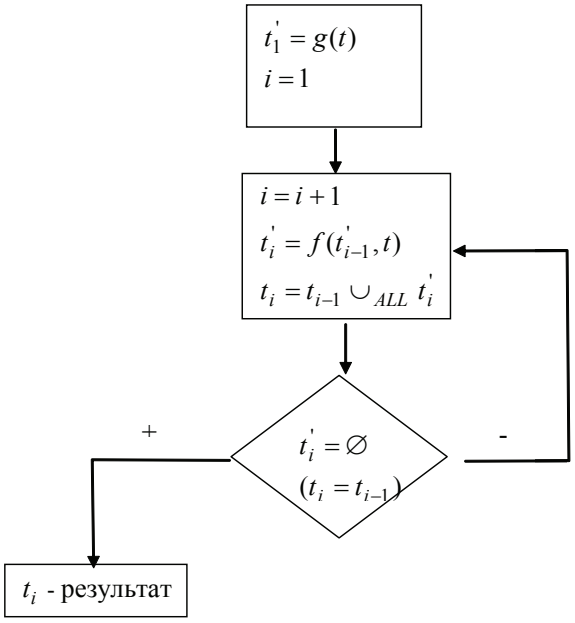
$$t_i' = f(t_{i-1}', t), t_i = t_{i-1} \cup_{ALL} t_i',$$

…

Якщо на певному кроці $n + 1$, де $n = 1, 2, \ldots$, виконується рівність $t_n = t_{n+1}$, то процес обчислень закінчується, і таблиця t_n покладається рівним результату. В іншому випадку (тобто для всіх $n = 1, 2, \ldots$, виконується нерівність $t_n \neq t_{n+1}$) значення $C\,(g,f)\,(t)$ невизначено.

Змістовно кажучи, значення нової функції визначено тоді і тільки тоді, коли послідовність t_i, $i = 1, 2, \ldots$ стабілізується.

Нижченаведена діаграма ілюструє описану процедуру знаходження значень функції, яка будується композицією C.



Мал. 2. Блок-схема процедури знаходження значень вигляду $C\,(g,f)\,(t)$

У наведеному вище прикладі побудови генеалогічного дерева функція g задається першим оператором SELECT.

```
SELECT LastName, FirstName, PersonID, 1
FROM FamilyTree
WHERE PersonID = 10
```

Функція ж f задається другим оператором SELECT.

```
SELECT Node.LastName, Node.FirstName, Node.
    PersonID, X.Level + 1
FROM FamilyTree Node
JOIN Tree X
ON Node.MotherID = X.PersonID OR Node.
    FatherID = X.PersonID
```

Узагальнимо оператор CTE на випадок багатьох таблиць. А саме: нехай задані n-арна функція $g : T_1 \times T_2 \times \ldots \times T_n \to T$ та $n + 1$-арна функція $f : T_1 \times T_2 \times \ldots \times T_n \times T_{n+1} \to T$. Тоді узагальнений CTE-вираз уточнюється за допомогою узагальненої бінарної композиції C, яка функціям g, f ставить у відповідність нову n-арну функцію $C\,(g,f) : T_1 \times T_2 \times \ldots \times T_n \to T$; значення якої на аргументах $t_1, t_2, \ldots, t_n$ знаходиться наступним чином. Будуємо послідовність таблиць $t'_1, t'_2, t'_3, \ldots, t'_i, \ldots$, де

$$t_1' = g(t_1, t_2, t_3, \ldots, t_n),$$

$$t_2' = f(t_1', t_1, t_2, t_3, \ldots, t_n),$$

$$t_3' = f(t_2', t_1, t_2, t_3, \ldots, t_n),$$

$$\dots$$

$$t_i^{'} = f(t_{i-1}^{'}, t_1, t_2, t_3, ..., t_n),$$

$$\dots$$

Якщо на певному кроці $n + 1$, де $n = 1, 2, ...$, отримується порожня таблиця, причому усі попередні таблиці послідовності не є порожніми, то процес обчислень закінчується з результатом $t_1^{'} \cup_{ALL} t_2^{'} \cup_{ALL} \dots \cup_{ALL} t_n^{'}$.

В іншому випадку (тобто всі таблиці t_i, $i = 1, 2, \dots$ непорожні) значення $C\ (g, f)\ (t_1, t_2, t_3, \dots, t_n)$ покладається невизначеним.

Відмітимо, що функція f може відразу повертати порожню таблицю. Тоді будемо мати CTE- вирази у не рекурсивній формі.

Рекурсивні вирази на сукупності запитів

Перейдемо до розгляду рекурсивних виразів на сукупності запитів. А саме: нехай є m n-арних функцій g

$$g_1 : T_1 \times T_2 \times ... \times T_n \to T$$

$$g_2 : T_1 \times T_2 \times ... \times T_n \to T$$

$$\dots$$

$$g_m : T_1 \times T_2 \times ... \times T_n \to T$$

та m m+n-арних функцій f

$$f_1 : T_1 \times T_2 \times ... \times T_m \times T_{m+1} \times ... \times T_{m+n} \to T$$

$$f_2 : T_1 \times T_2 \times ... \times T_m \times T_{m+1} \times ... \times T_{m+n} \to T$$

$$\dots$$

$$f_m : T_1 \times T_2 \times ... \times T_m \times T_{m+1} \times ... \times T_{m+n} \to T$$

CTE-вираз на множині запитів задається композицією C, яка функціям $g_1, g_2, \dots, g_m$, та $f_1, f_2, \dots, f_m$ ставить у відповідність нову функцію $C\ (g_1, f_1, g_2, f_2, \dots, g_m, f_m)$: $T_1 \times T_2 \times ... \times T_n \to T$. Її значення на аргументах $t_1, t_2, \dots, t_n$ знаходиться наступним чином. Позначимо літерою s з індексами результуючі таблиці. На першому кроці розраховуються начальні значення

$$s_1^1 = g_1(t_1, t_2, \dots, t_n)$$

$$s_2^1 = g_2(t_1, t_2, \dots, t_n)$$

$$\dots$$

$$s_m^1 = g_m(t_1, t_2, \dots, t_n)$$

На i-му кроці значення таблиці s_k^i розраховується за формулою $f_k(s_1^{i-1}, s_2^{i-1}, \dots, s_m^{i-1}, t_1, \dots t_n)$, якщо усі попередні значення $s_k^1, s_k^2, \dots, s_k^{i-1}$ не є порожніми, або приймається рівним останньому не порожньому значенню. В останньому випадку будемо говорити, що значення таблиці s_k стабілізувалося. Обчислення закінчується, коли значення всіх таблиць стабілізуються. Нехай це сталося на l-кроці. Після цього розраховуються значення результуючих таблиць

$$s_1 = s_1^1 \cup_{ALL} s_1^2 \cup_{ALL} \dots \cup_{ALL} s_1^l$$

$$s_2 = s_2^1 \cup_{ALL} s_2^2 \cup_{ALL} \dots \cup_{ALL} s_2^l$$

$$\dots$$

$$s_m = s_m^1 \cup_{ALL} s_m^2 \cup_{ALL} \dots \cup_{ALL} s_m^l$$

В якості значення всієї функції $C\ (g_1, f_1, g_2, f_2, \dots, g_m, f_m)$ домовимося брати першу таблицю s_1.

Висновки

CTE-оператор у рекурсивній формі є більш зручним та ефективним засобом опрацювання ієрархічних структур даних в мові SQL, ніж використання процедурного розширення мови. CTE-оператор працює на рівні мультимножинних аналогів теоретико-множинних операцій, що дозволяє писати запити в компактній та наочній формі. Водночас CTE-оператор непридатний, якщо результат запиту залежить від порядку обходу дерева або іншої ієрархічної структури. Зрозуміло, що в цьому випадку треба використовувати процедурно-орієнтовані методи написання запитів.

Подальшу роботу буде присвячено формальній композиційній семантиці рекурсивних запитів мови SQL – композиції C.

1. Редько В. Н. Реляційні бази даних: табличні алгебри та SQL-подібні мови / В. Н. Редько, Ю. Й. Брона, Д. Б. Буй, С. А. Поляков. – К. : Академперіодика, 2001. – 198 с.
2. Viescas J. L. SQL Queries for Mere Mortals: a hands-on guide to data manipulation in SQL / J. L. Viescas, M. J. Hernandez. – [2nd edition]. – Massachusetts : Addison-Wesley, 2007. – 631 p.
3. Neilesen P. SQL Server 2005 Bible / P. Neilesen. – Indiana : Wiley Publishing Inc., 2007. – 1293 p.
4. Celko J. Joe Celko's Trees and hierarchies in SQL for smarties / J. Celko. – San Francisco : Morgan Kaufmann Publishers, 2004. – 238 p.
5. Beaulieu A. Mastering Oracle SQL / A. Beaulieu, S. Mishra. – [2nd edition]. – Sebastopol : O'Relly Media Inc., 2004. – 492 p.
6. Уилсон Р. Введение в теорию графов / Р. Уилсон. – М. : Мир, 1977. – 207 с.
7. Петровський А. Б. Основные понятия теории мультимножеств / А. Б. Петровський. – М. : Едиториал УРСС, 2002. – 80 с.
8. Буй Д. Б. Решітка мультимножин / Д. Б. Буй, Ю. О. Богатирьова // Современные направления теоретических и прикладных исследований: международная конференция SWORD, 16–27 марта 2009 г., Одесса: Черноморье. – 2009. – Т. 2. – С. 49–52.

9. Богатырёва Ю. А. Мультимножества: библиография, решетка мультимножеств / Ю. А. Богатырёва // Theoretical and Applied Aspects of Program Systems Development: international conference, December 8–10, 2009. – Kyiv. – 2009. – С. 13–20.

10. Буй Д. Б., Богатырева Ю. А. К вопросу о решетке мультимножеств / Д. Б. Буй, Ю. А. Богатырева // Десятый международный семинар «Дискретная математика и ее приложения», Москва, МГУ, 1–6 февраля 2010 г.

D. Buy, S. Polyakov

COMPOSITIONAL SEMANTICS OF THE RECURSIVE EXPRESSIONS AND THEIR GENERALIZATIONS IN LANGUAGES LIKE TO SQL

The paper describes a method for showing hierarchies in relation databases uses an adjacency list model. The paper introduces the adjacency lists sorts and their samples. The navigations queries are described as well as common table expression in their recursive format. Samples of such queries are shown. The paper defines formal semantic of the recursive common table expression.

Keywords: compositional semantics, recursive queries, SQL, common table expression, CTE.

УДК 004.8

Глибовець А. М., Шабінський А. С.

ОДИН ПІДХІД ДО ПОБУДОВИ ІНТЕЛЕКТУАЛЬНОЇ ПОШУКОВОЇ СИСТЕМИ

У роботі подано загальне бачення архітектури інтелектуальної пошукової системи наукових матеріалів. Звуження предметної області проведено для покращення відсіву пошукового «сміття» і структурованості бази знань.

Ключові слова: інтелектуальна пошукова система (ІПС), інформаційний пошук (ІП), семантичний пошук, пошук за індексом,онтологія, пошуковий агент,

Розвиток World Wide Web (WWW) спричинив суттєве збільшення об'єму інформації в Інтернеті. Постає питання: як розрізнити інформацію від знання та прискорити час її обробки? Зрозуміло, що тут може нам допомогти контекст знань.

Метою цієї статті є розробка архітектурних принципів функціонування та впровадження інтелектуальної пошукової системи, яка б дозволяла здійснювати оптимальний пошук документів наукового та науково-публіцистичного типу з використанням семантичної мережі.

Інтелектуальна пошукова система (ІПС) має ряд суттєвих переваг порівняно із традиційними пошуковими системами, зокрема базованими на пошуку за ключовими словами. Більшість переваг полягає у використанні покращеної моделі інформаційного пошуку (ІП), інтерактивної взаємодії з користувачем, участі експертів, автоматизованих засобів формування та підтримки бази знань, спробі «зрозуміти» інформаційну потребу користувача. Важливою функцією ІПС стає автоматичне динамічне накопичення знань та мета знань у процесі роботи системи. В цьому контексті досвід – це сукупність опрацьованих і впорядкованих знань, результат роботи експертів та опрацювання відгуків користувачів. Зрозуміло, що в цьому випадку здійснюваний ІПС-пошук має бути семантичним. Тобто система має оперувати не тільки вербальним описом сутностей предметних областей, а й семантичними поняттями, які вкладені в інформаційні одиниці та зв'язки між ними.