УДК 681.3.062

РЕКУРСИВНІ ЗАПИТИ В SQL-ПОДІБНИХ МОВАХ: ПРИКЛАДИ, ЗМІСТОВА І ФОРМАЛЬНА СЕМАНТИКА

Д.Б. Буй, С.А. Поляков

Київський національний університет імені Тараса Шевченка,
МСП, 01601, вул. Володимирська, 64, Київ, Україна, факс/тел. (044)521 3345,
buy@unicyb.kiev.ua, polyakov@unicyb.kiev.ua

Розглянутий метод моделювання ієрархічних структур даних у вигляді списків суміжності. Наведені приклади таких списків та їх типи. Розглянуті методи побудови навігаційних запитів до ієрархічних структур даних, у тому числі за допомогою загальних табличних виразів у їх рекурсивній формі. Наведені приклади таких запитів. Задана формальна семантика рекурсивних загальних табличних виразів CTE.

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. Simple navigations queries are described as well as common table expression in their recursive format. Samples of the recursive queries are shown. The paper defines formal semantic of the recursive common table expression.

Загальні зауваження

Мова SQL надає потужні та елегантні засоби для маніпулювання табличними структурами даних, які по суті є мультимножинами рядків [1]. Операції SQL в більшості випадків виявляються набагато ефективнішими при виконанні запитів над таблицями, ніж виконання тих же 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 операції перетину, об'єднання та різниці множин.

Але платою за такий високий рівень операцій є більш обмежені функціональні можливості мови SQL у порівняні з традиційними мовами програмування. Існують класи запитів до табличних структур даних, які не можуть бути реалізовані засобами тільки 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 програмувалось засобами процедурних розширень SQL і мало чим відрізнялось від програмування класичними мовами програмування. Тільки в стандарті SQL:99 було введено розширення, яке допускає рекурсивні запити [3]. У подальшому це розширення мови було втілено в таких провідних СУБД як DB2 та MS SQL Server. В той же час Oracle не підтримав 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, які будуть розглянуті далі. Розглянемо таку класичну ієрархічну структуру як бюрократична організація, що будується за принципом керівник-підлеглий. Кожний підлеглий має тільки одного керівника і кожний керівник має багато підлеглих. Визначимо таблицю співробітників Employees (табл. 1).

Таблиця 1. Таблиця співробітників Employees

EmployeeId	ManagerId	LastName	FirstName	Title	BirthDate	HireDate	City	Region	Address

Перші два атрибути табл. 1 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 тій же 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 найму в організацію (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 кожний нащадок має тільки одного предка. В загальному випадку у кожного нащадка може існувати декілька предків.

У теорії графів списку суміжності з кількістю нащадків не більше ніж n відповідає орієнтований ациклічний граф з валентністю вершин не більше n [1]. Для вищенаведеного 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 двох предків (точніше кажучи, 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 і 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 у будь-якому випадку в такому графі має бути хоча б одна вершина з нульовою валентністю.

У табл. 2, задається генеалогічне дерево. Перші три атрибути використовуються для встановлення дуальних зв'язків між рядками таблиці.

Таблиця 2. Таблиця родинних зв'язків FamilyTree

PersonId	FatherId	MotherId	LastName	FirstName	BirthDate	City	Region	Address

Атрибут PersonId містить, як і раніше, унікальний ідентифікатор людини, тобто є первинним ключем. Атрибути FatherId та MotherId посилаються на батька та матір людини відповідно. Кожний з 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.

Розглянемо так звані навігаційні запити, тобто запити, в умові фільтрації яких використовується зв'язок предок/нащадок. Наприклад, потрібно знайти всі вузли, які є нащадками заданого вузла. Найпростішими навігаційними запитами є такі, коли треба знайти всіх синів зазначеного 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»
```

Звернемо 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 може бути побудований стандартними засобами оператора SELECT. Наприклад, нехай потрібно знайти всіх нащадків якої-небудь людини. Природно, людина вважається дитиною деякої 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 в глибину. Але коли даних багато, використання курсорів не є ефективним.

Більш ефективним є метод, коли створюється допоміжна тимчасова таблиця. На першому кроці туди записується тільки 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, тоді 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 кожний раз потрібно звернення до сервера, що 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 потрібну множину без необхідності звернення до процедурного коду.

[1] Стандартно валентністю вершини називається кількість ребер, інцидентних вершині [6].

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

```
/*CTE*/
WITH CTEName (parameters)
AS (Simple Subquery)

/* головний запит*/
SELECT...
FROM CTEName
```

Рекурсивна форма CTE має більш складний вигляд. Вона складається з двох операторів SELECT, які поєднані операцією UNION ALL (підкреслимо, що йдеться саме про операцію UNION ALL; нижче буде показано, що замінити операцію UNION ALL на UNION не можна[2]).

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 даного прикладу вона буде складатися тільки з одного рядка, який відповідає вказаній людині.

Другий оператор SELECT формує результуючу таблицю рекурсивним (в іншій термінології рекурентним або індуктивним) чином.

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

Семантика рекурсивних запитів

Перейдемо тепер 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

$$t_1 = g(t),$$

$$t_2 = f(t_1, t),$$

$$t_3 = f(t_2, t),$$

$$\ldots$$

$$t_i = f(t_{i-1}, t).$$

$$\ldots$$

[2] Головна відмінність між цими операціями полягає в тому, що операція UNION ALL будує таблицю з дублікатами рядків, а операція UNION – без дублікатів.

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

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

Вище операція $\cup_{ALL}$ – операція об'єднання мультимножин з врахуванням дублікатів [1, підрозд. 3.4, с.153].[3]

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

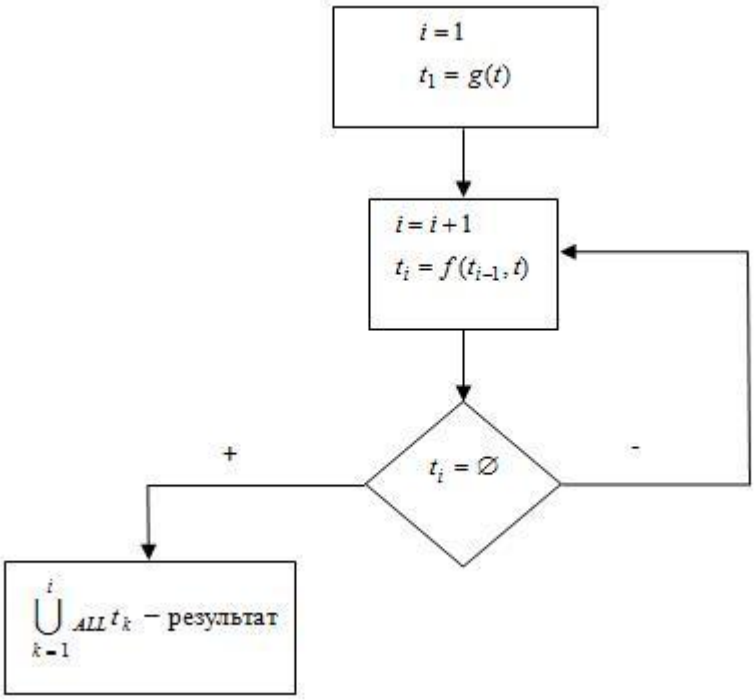

Рис. 1. Блок-схема процедури знаходження значень вигляду $C(g,f)(t)$

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 з врахуванням дублікатів. Характеристична властивість цього об'єднання полягає в тому, що коли обидві таблиці-аргументи не є порожніми, то результат відрізняється від аргументів (навіть якщо один аргумент "вкладається" в іншій).[4]

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 $t_1, t_2, \ldots,$:

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

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

$$\ldots$$

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

$$\ldots$$

[3] За іншою термінологією ця операція називається операцією додавання мультимножин [7, 9].

[4] Точніше кажучи, йдеться 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 в решітку ([7, 8]).

Якщо на деякому 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. У протилежному 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.

Нижченаведана діаграма (рис. 2) ілюструє описану процедуру знаходження значень функції, яка будується композицією C.

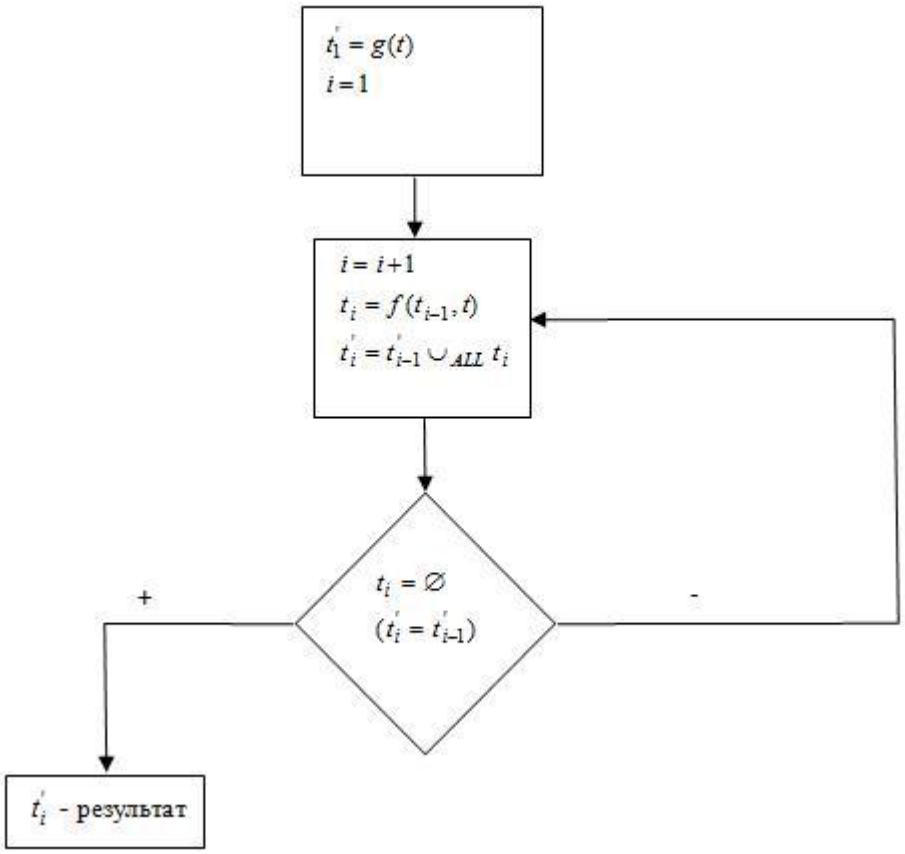

Рис. 2. Блок-схема процедури знаходження значень вигляду $C(g, f)(t)$

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

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

А рекурсивна функція f задається другим оператором SELECT.

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

Висновки

CTE-оператор в 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. Разом с тим CTE-оператор є непридатним, коли результат запиту залежить від порядку обходу дерева або іншої ієрархічної структури. Зрозуміло, що в цьому випадку треба використовувати процедурно-орієнтовані методи написання запитів.

Подальша робота буде присвячена формальній композиційній семантиці рекурсивних запитів мови SQL – композиції C.

1. *Реляційні* бази даних: табличні алгебри та SQL-подібні мови / В.Н. Редько, Ю.Й. Брона, Д.Б. Буй, С.А. Поляков. – К.: Видавничий дім "Академперіодика", 2001. – 198 с.
2. *Viescas J.L.* SQL Queries for Mere Mortals: a hands-on guide to data manipulation in SQL [2nd edition] / J.L. Viescas, M.J. Hernandez. – 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 [2nd edition] / A. Beaulieu, S. Mishra – Sebastopol: "O'Relly Media Inc.", 2004. – 492 p.
6. *Уилсон Р.* Введение в теорию графов / Р. Уилсон – Введение в теорию графов. – М.: Мир, 1977. – 207 с.
7. *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.
8. *Богатырева Ю.А.* Мультимножества: библиография, решетка мультимножеств / Ю.А. Богатырева // Theoretical and Applied Aspects of Program Systems Development: international conference, December 8–10, 2009. – Kyiv, 2009. – С. 13–20.
9. *Петровський А.Б.* Основные понятия теории мультимножеств / А.Б. Петровський. – М.: Едиториал УРСС, 2002. – 80 с.