УДК 681.3.06

Кулябко П. П., Пешко О. І, Римарчук В. К

ЕФЕКТИВНІСТЬ ВИКОНАННЯ ЗАПИТІВ З МНОЖИННИМИ ПОРІВНЯННЯМИ У МОВІ SQL

На основі порівняльного аналізу виразових можливостей SQL-92 стандарту і класичного SQL(SEQUEL) пропонуються схеми реалізації складних запитів з множинними порівняннями засобами SQL Access. Досліджується ефективність виконання таких запитів.

Мова SQL на сьогодні є найпоширенішою мовою запитів СУБД реляційного типу і підтримується практично всіма програмними продуктами, які представлені на ринку технологій, пов'язаних з базами даних та інформаційними системами.

Виробники програмних продуктів при розробці мовних процесорів SQL орієнтуються на стандарт 1992 року, або SQL-92 [3]. Проте ця мова має істотно більшу історію, бо стала відомою ще у 1974 році [1] під назвою SEQUEL. Вона була розроблена у рамках проекту експериментальної реляційної СУБД System R.

Мова SEQUEL була задумана як мова функціонального типу з відповідними засобами та технологічними прийомами програмування. Під тиском користувачів, переважна більшість яких була зорієнтована на процедурний стиль програмування, в наступній версії мови SEQUEL/2 [2] з'являються додаткові засоби, такі як GROUP BY та множинні функції типу SET.

Пізніше, на початку 80-х років, мова SEQUEL/2 отримала нову назву SQL, а згодом у нових версіях цієї мови зникають можливості для множинного порівняння, що було зумовлено великою складністю алгоритмів порівняння множин, а відтак намаганням розробників зорієнтувати користувачів на застосування ефективніших схем запитів пошуку даних. Але, на нашу думку, деякі схеми запитів класичного SEQUEL/2 зручніші для користувача (принаймні з логічної точки зору), бо мають стрункішу логічну структуру і чіткіше семантичне тлумачення через теорію множин та теорію відображень порівняно з функціонально еквівалентними їм схемами запитів, що виражені засобами SQL-92. Проілюструємо це твердження рядом прикладів.

Для розгляду прикладів типів запитів опишемо структуру таблиць з предметної області «Розклад вищого навчального закладу».

Лектор	Код лектора	Прізвище	Організація	Телефон	Науковий ступінь
Lector	KL	Name	Огд	Phone	Degree

Предмет	Код предмета	Назва	Кількість годин	Контроль
Subject	KS	Subj	Hours	Test

Група	Код групи	Курс	Кафедра	Факультет	Кількість чоловік
Group	KG	Course	Cathedra	Faculty	Number

Розклад	Код лектора	Код предмета	Код групи	Аудиторія	День	Пара
Schedule	KL	KS	KG	Aud	Day	Pair

Розглянемо тип запиту, що пропонується в [4]:

«Знайти прізвища лекторів, що читають всі предмети».

У термінах мови SEQUEL (класичної) цей запит може бути представлений так:

```
SELECT DISTINCT Lector.Name
FROM Lector
WHERE Lector.KL IN
        (SELECT Schedule.KL
         FROM Schedule
         GROUP BY Schedule.KL
         HAVING SET(Schedule.KS) =
                        (SELECT Subject.KS
                         FROM Subject)}
```

Але в SQL-92 (і також в SQL Access) немає засобів для множинного порівняння, тому потрібна суттєва модифікація.

Варіант 1 (кількісний)

```
                                                  Г COUNT*/
SELECT Lector.Name
FROM Lector
WHERE Lector.KL IN
        (SELECT Schedule.KL
         FROM Schedule
         GROUP BY Schedule.KL
         HAVING Count(Schedule.KS) =
                        (SELECT Count(SubjectKS)
                         FROM Subject));
```

Така технологія дуже зручна, але, на жаль, значною мірою специфічна, бо фактично має місце заміна початкового запиту на інший:

«Знайти прізвища лекторів, які читають таку *кількість* предметів, що дорівнює *кількості* всіх існуючих предметів».

У наведеному контексті ці запити семантичне еквівалентні, але подібний прийом буде некоректним, якщо відносно існуючих предметів з'явиться певна умова (обмеження), і тоді кількісне порівняння не зможе замінити порівняння множин при збереженні семантичної еквівалентності.

Цей та ряд інших типів запитів наводяться в [5].

Варіант 2 («ні-ні»)

```
                                                  /*NOT-NOT*/
SELECT DISTINCT Lector.Name
FROM Lector
WHERE NOT EXISTS
        (SELECT *
         FROM Subject
         WHERE NOT EXISTS
                (SELECT *
                 FROM Schedule
                 WHERE Lector.KL = Schedule.KL
                 AND Subject.KS = Schedule.KS));
```

Цей варіант, що наводиться в [4], передбачає перетворення початкового запиту на рівні природної мови до такого (семантичне еквівалентного початковому):

«Знайти прізвища лекторів, для яких *не* існує предметів, які б вони *не* читали».

Специфіка цього варіанту полягає в необхідності перетворень початкового запиту на рівні природної мови, що, по-перше, в багатьох випадках досить складно, особливо з урахуванням обгрунтування еквівалентності перетворення, а по-друге, важко описати технологію такого перетворення в зв'язку з неформальністю та неоднозначністю природної мови.

Більш технологічним, на наш погляд, є підхід, який можна вважати (до певної міри) розширенням варіанта «ні-ні», але з формальними перетвореннями на рівні теоретико-множинних співвідношень.

Розглянемо запит:

«Знайти коди лекторів, які читають принаймні всім групам кафедри інформатики».

В термінах класичного SEQUEL цей запит може виглядати, наприклад, так:

```
SELECT DISTINCT KL
FROM Schedule
GROUP BY KL
HAVING SET(KG)                          /* A */
CONTAINS
        (SELECT KG                       Г В 1
         FROM Group
         WHERE Group.Cathedra = «informatics»);
```

Якщо множину кодів груп *SET(KG)* позначимо через *A,* а множину *(SELECT KG...)* - через *B,* то отримаємо співвідношення

$$(A \supseteq B) \circ (B \backslash A = 0)$$

або

$$(B \pi(-A) = 0).$$

Відповідно до останнього можемо отримати таку SQL-програму:

```
                                          Г CONTAINS - NOT STRICT*/
SELECT DISTINCT x.KL
FROM Schedule AS x
WHERE NOT EXISTS
        (SELECT *
         FROM Group
         WHERE Group.Cathedra = «informatics»
         AND (Group.KG NOT IN
                (SELECT Schedule. KG
                 FROM Schedule
                 WHERE Schedule.KL = x.KL)));
```

Якщо потрібне строге включення, то маємо

$$(A \supset B) \Leftrightarrow ((B \backslash A = \varnothing) \,\&\, (A \backslash B \neq \varnothing))$$

або

$$(B \cap (-A) = \varnothing) \,\&\, (A \cap (-B) \neq \varnothing).$$

```
                                        /* CONTAINS – STRICT*/
SELECT DISTINCT x.KL
FROM Schedule AS x
WHERE NOT EXISTS
        (SELECT *
         FROM Group
         WHERE Group.Cathedra = «informatics»
         AND (Group.KG NOT IN
                 (SELECT Schedule.KG
                  FROM Schedule
                  WHERE Schedule.KL = x.KL)))
AND EXISTS
        (SELECT Schedule.KG
         FROM Schedule
         WHERE (Schedule.KL = x.KL)
         AND (Schedule.KG NOT IN
                 (SELECT Group.KG
                  FROM Group
                  WHERE Group.Cathedra = «informatics»)));
```

Для випадку рівності множин маємо:

$$(A = B) \Leftrightarrow ((B \backslash A = \varnothing) \,\&\, (A \backslash B = \varnothing))$$

або

$$(B \cap (-A) = \varnothing) \,\&\, (A \cap (-B) = \varnothing).$$

```
                                        /* EXACTLY */
SELECT DISTINCT x.KL
FROM Schedule AS x
WHERE NOT EXISTS
        (SELECT *
         FROM Group
         WHERE Group.Cathedra = «informatics»
         AND (Group.KG NOT IN
                 (SELECT Schedule.KG
                  FROM Schedule
                  WHERE Schedule.KL = x.KL)))
AND NOT EXISTS
        (SELECT Schedule.KG
         FROM Schedule
         WHERE (Schedule.KL = x.KL)
         AND (Schedule.KG NOT IN
                 (SELECT Group.KG
                  FROM Group
                  WHERE Group.Cathedra = «informatics»)));
```

Швидкість роботи запитів перевірялась у системі MS Access та середовищі візуального програмування Delphi (тип таблиць dBASE IV). Комп'ютер, на якому проводилось тестування запитів, має такі характеристики:

Pentium III, 600B MHz, RAM 64 Mb,
Microsoft Windows98.

Програми, які були використані:

Microsoft Access 2000, Borland Delphi v. 5.0.

Спочатку розглянемо швидкості виконання варіантів запиту *«Знайти прізвища лекторів, що читають всі предмети»,* які можуть бути реалізовані через різні засоби мови SQL, а саме через оператори ***NOT EXISTS*** або ***NOT IN.*** Наведемо тексти відповідних запитів для Access.

1. У цьому варіанті відбувається перетворення початкового запиту на рівні природної мови на такий: *«Знайти прізвища лекторів, для яких не існує предметів, які б вони не читали».* Цей запит реалізується через два ***NOT EXISTS*** («ні-ні»). Надалі будемо умовно називати його «NE-NE».

```
SELECT DISTINCT Name
FROM Lector
WHERE NOT EXISTS
        (SELECT *
         FROM [Subject]
         WHERE NOT EXISTS
                 (SELECT *
                  FROM [Schedule]
                  WHERE KL = Lector.KL
                  AND KS = Subject.KS));
```

2. Другий варіант використовує оператор для роботи з множинами ***NOT IN.*** Коротко його роботу можна пояснити так. Третій *SELECT* повертає коди предметів *KS* з таблиці *Subject,* яких немає для даного лектора у таблиці *Schedule,* тобто для кожного лектора знаходяться коди предметів, які той не читає. Ті лектори, для яких ця множина пуста, і будуть лекторами, які читають всі предмети. Надалі будемо умовно називати його «NE-NI».

```
SELECT DISTINCT Name
FROM Lector
WHERE KL IN
        (SELECT KL
         FROM Schedule AS x
         WHERE NOT EXISTS
                 (SELECT KS
                  FROM [Subject]
                  WHERE KS NOT IN
                          (SELECT KS
                           FROM Schedule
                           WHERE KL = x.KL)));
```

В таблицях показано, скільки часу (в секундах) знадобилося для виконання запитів «NE-NE» та «NE-NI» у Delphi та Access відповідно.

Access	250	500	750	1000	1250	1500
NE-NE	0,4453	1,1016	2,2	3,5195	5,7695	7,6914
NE-NI	1,2109	5,2188	13,67	26,4805	42,457	62,5625

Delphi	250	500	750	1000	1250	1500
NE-NE	13,4	15,81	19,67	21,86	25,54	24
NE-NI	105,52	233,33	391,94	570,95	834,48	991,68

Крім двох попередніх, були досліджені часові характеристики ще й таких запитів:

«Знайти прізвища лекторів, які читають точно такі самі предмети, як і лектор з кодом "L010"». Робота цього запиту виглядає так. Другий *SELECT* виділяє коди тих лекторів, для яких множини предметів, які читають ці лектори і при цьому не читає лектор з кодом «L010», а також предмети, які читає «L010» і не читають ці лектори, пусті (тобто обидва *NOT EXISTS* повертають істину). Надалі будемо умовно називати його **«Ex(L010)».**

```
SELECT DISTINCT Name
FROM Lector
WHERE KL IN
        (SELECT KL
         FROM Schedule AS x
         WHERE NOT EXISTS
                 (SELECT KS
                  FROM [Schedule]
                  WHERE KL = «L010»
                  AND (KS NOT IN
```

```
                    (SELECT KS
                     FROM [Schedule]
                     WHERE KL = x.KL)))
AND NOT EXISTS
          (SELECT KS
           FROM [Schedule]
           WHERE (KL = x.KL)
           AND (KS NOT IN
                    (SELECT KS
                     FROM [Schedule]
                     WHERE KL=«L010»))));
```

Цей запит являє собою кількісний варіант запиту про лекторів, які читають всі предмети, і формулюється так: *«Знайти прізвища лекторів, які читають таку киїькість предметів, що дорівнює кількості есіе існуючих предметів»*. Надалі будемо умовно називати його «Count».

```
SELECT Name
FROM Lector
WHERE KL IN
          (SELECT KL
           FROM [Schedule]
           GROUP BY KL
           HAVING count(KS) =
                    (SELECT count(KS)
                     FROM Subject));
```

Наступний запит формулюється так: *«Знайти прізвища лекторів, які читають принаймні всім групам кафедри інформатики»*,- тобто ті лектори, які крім усіх груп кафедри інформатики можуть викладати у групах інших кафедр. Для цих лекторів пустою є множина груп кафедри інформатики, у яких вони не викладають. Будемо скорочено називати цей запит «Not Strict».

```
SELECT DISTINCT Name
FROM Lector
WHERE KL IN
          (SELECT x.KL
           FROM Schedule AS x
           WHERE NOT EXISTS
                    (SELECT *
                     FROM [Group]
                     WHERE Cathedra = «informatics»
                     AND (KG NOT IN
                              (SELECT KG
                               FROM [Schedule]
                               WHERE KL = x.KL))));
```

Запит вибирає прізвища лекторів, які викладають в усіх групах кафедри інформатики і, крім цього, обов'язково читають ще принаймні в одній групі іншої кафедри. Будемо називати цей запит **«Strict».**

```
SELECT DISTINCT Name
FROM Lector
WHERE KL IN
          (SELECT x.KL
           FROM Schedule AS x
           WHERE NOT EXISTS
                    (SELECT *
                     FROM [Group]
                     WHERE Cathedra = «informatics»
                     AND (KG NOT IN
                              (SELECT KG
                               FROM [Schedule]
                               WHERE KL = x.KL)))
          AND EXISTS
                    (SELECT KG
                     FROM [Schedule]
                     WHERE KL = x.KL
                     AND (KG NOT IN
                              (SELECT KG
                               FROM [Group]
                               WHERE Cathedra = «informatics»))));
```

Запит вибирає лекторів, які викладають у всіх групах кафедри інформатики, і при цьому не існує груп з інших кафедр, у яких би ці лектори викладали. Будемо називати цей запит «Exactly».

```
SELECT DISTINCT Name
FROM Lector
WHERE KL IN
          (SELECT x.KL
           FROM Schedule AS x
           WHERE NOT EXISTS
                    (SELECT *
                     FROM [Group]
                     WHERE Cathedra = «informatics»
                     AND KG NOT IN
                              (SELECT KG
                               FROM [Schedule]
                               WHERE KL = x.KL)))
AND NOT EXISTS
          (SELECT KG
           FROM [Schedule]
           WHERE KL = x.KL
           AND (KG NOT IN
                    (SELECT KG
                     FROM [Group]
                     WHERE Cathedra = «informatics»))));
```

Час виконання описаних вище запитів подано в табл. 1.

Отже, з наведених результатів видно, що всі запити набагато швидше виконуються у Access, ніж у Delphi. Що ж стосується типів запитів, то

Таблиця 1

Час виконання запитів в Access у секундах

Access	NE-NE	NE-NI	Ex(L010)	Count	Not Strict	Strict	Exactly
250 записів	0,4453	1,2109	3,9063	0,3281	0,875	2,6875	1,5391
500 записів	1,1016	5,2188	8,3516	0,5	3,0156	6,2578	5,1563
750 записів	2,2	13,67	35,04	0,44	5,77	13,9	10,33
1000 записів	3,5195	26,4805	56,3594	0,1094	9,5625	24,0586	16,9688
1250 записів	5,7695	42,457	82,5469	0,3906	14,5	37,457	25,3711
1500 записів	7,6914	62,5625	112,4297	0,5	20,3711	52,793	35,7617

Час виконання запитів у Delphi у секундах

Delphi	NE-NE	NE-NI	Ex(L010)	Count	Not Strict	Strict	Exactly
250 записів	13,4	105,52	110,95	0,17	56,52	59,48	59,98
500 записів	15,81	233,33	245,51	0,17	124,74	130,77	131,44
750 записів	19,67	391,94	407,17	0,17	206,52	215,09	216,68
1000 записів	21,86	570,95	592,65	0,22	298,79	314,62	315,05
1250 записів	25,54	834,48	867,28	0,17	442,26	456,38	465,71
1500 записів	23,9	991,68	1042,87	0,22	527,61	547,83	550,96

легко бачити, що в обох системах найменший час виконання має запит «Count», найбільший - «Ex(L010)». На основі отриманих результатів переконливо продемонстровано, що запит «NE-NE» хоча й складніше формулюється, проте працює набагато ефективніше, ніж еквівалентний йому запит «NE-NI», що можна пояснити великою часовою складністю програм, які працюють з множинами. Приблизно однаковий часовий рівень мають запити «Exactly», «Strict» та «Not Strict» (їх часові показники знаходяться приблизно посередині між «NE-NE» та «NE-NI»).

1. *Chamberlin D .D., Boyce R. F.* SEQUEL: A Structured English Query Language // Proc. ACM SIGMOD Workshop on Data Description, Access and Control. - Ann Arbor, Mich., 1974.
2. *Chamberlin D. D.* et al. SEQUEL/2: A Unified Approach to Data Definition, Manipulation and Control // IBM J. R&D- 1976.- V. 20, № 6- 1977.- V. 21, № 1.
3. *Datę C. J., Dan'en H. A* Guide to the SQL Standard- Reading, Mass.: Addison-Wesley, 1993.
4. *Дейт К. Дж.* Введение в системы баз данных.- 6-е изд.- К.: Диалектика, 1998.
5. *Глибовець М. М., Кулябко П. П.* Запити з множинними порівняннями у мові SQL // Наукові записки НаУКМА.- Т. 18.- Комп'ютерні науки- 2000-С. 19-21.

P. P. Kuljabko, O. L Peshko, V. K Rimarchuk

EFFICIENCE OF PERFORMANCE THE QUERY WITH SET COMPARISONS IN SQL

On the base of comparative analysis of SQL-92 standard expressive possibilities and classical SQL(SEQUEL) expressive possibilities are offerred schemes to realization of complex queries with set comparisons by SQL Access facilities. Efficience of performance of such queries is researched.