УДК 004.652
П.В.Косінський, С.В.Лавренчук
Луцький національний технічний університет

ОРГАНІЗАЦІЯ ДЕРЕВОПОДІБНИХ СТРУКТУР В БАЗАХ ДАНИХ

У статті розглядаються основні методи реалізації зберігання ієрархічних структур в реляційних базах даних і типові SQL - запити до них. Робиться оцінка переваг і недоліків, можливості оптимізації.

Ключові слова: *Модель даних, ієрархічна структура, база даних, списки суміжності, вкладені множини, мова SQL.*

Постановка проблеми. Більшість сучасних систем управління базами даних (СУБД) – реляційні, тобто представляють дані у вигляді двовимірної таблиці, в якій є рядки (записи) і стовпці (поля записів). У них використовуються або файл-серверні системи (dBase, Paradox, Clipper, FoxPro, Access), або SQL - сервери (Oracle, Informix, Sybase, Borland InterBase, MS SQL і так далі). Реляційні бази даних в більшості випадків задовольняють вимоги предметної області, але на практиці часто стикаються з іншою організацією даних, а саме ієрархічною. Проблема в тому, що дані, які мають ієрархічну структуру, дуже погано представляються в реляційній моделі, а ієрархічна модель має низький рівень мови запитів і маніпулювання даними.

Прикладом простого ієрархічного представлення може служити адміністративна структура вищого навчального закладу (рис.1.) : університет - факультет – спеціальність - група.

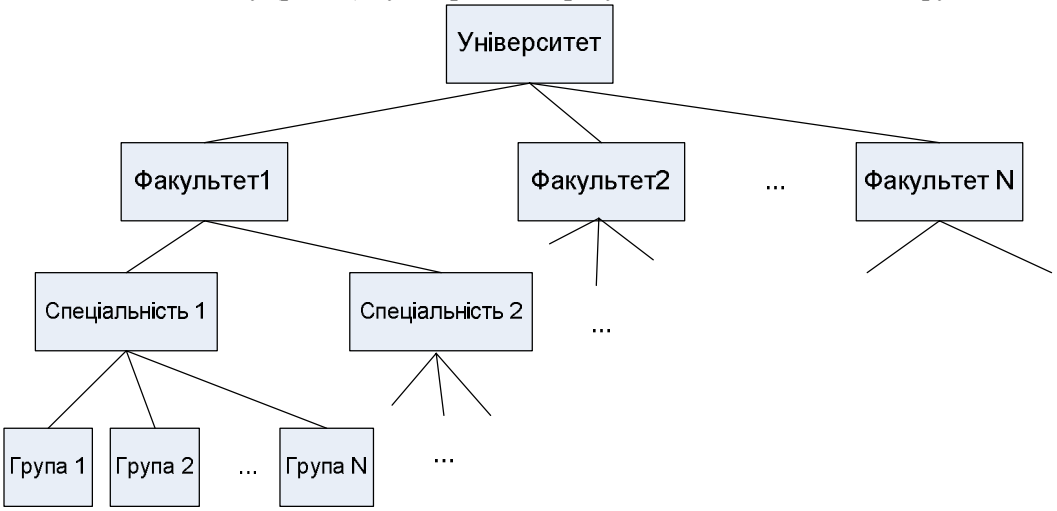

Рис.1. Ієрархічне представлення адміністративної структура вищого навчального закладу

Пошук даних у ієрархічній структурі виконується завжди по одній із гілок, починаючи з кореневого елемента, тобто повинний бути зазначений повний шлях руху по гілкам. Так, для пошуку і вибірки одного або декількох екземплярів запису типу група (рис.1.) необхідно вказати кореневий елемент університет і елементи факультет, спеціальність. Операції в ієрархічній моделі даних визначаються шляхом програмування «навігації» деревоподібною структурою даних.

Ієрархічна модель даних дозволяє економно використовувати пам'ять, найбільш реально відображає задану предметну область. Типовим представником ієрархічної СУБД (найбільш відомим і поширеним) є Information Management System (IMS) фірми IBM, яка використовує мову DL/1. Це мова програмування процедурного типу, тому результатом однієї пошукової операції є один екземпляр сегмента даних, або ж один екземпляр ієрархічного шляху до заданого екземпляра сегмента. Це вказує на низьку селективну потужність мови [3]. Крім того, не можна здійснити операцію додавання сегмента група без зазначення сегмента спеціальність, не можна видалити сегмент спеціальність, не видаляючи сегментів група.

Розглянемо основні методи реалізації зберігання ієрархічних структур в реляційних базах даних і типові SQL - запити до них.

Найпростіший спосіб представлення дерева в реляційній базі - це збереження матриці суміжності в таблиці. Кожен запис такої таблиці відповідає вузлу дерева і зберігає його унікальний ідентифікатор і посилання на батьківський вузол.

Для того, щоб зберігати інформацію про предка екземпляра об'єкту, будь-який об'єкт в таблиці повинен крім ідентифікатора мати ще й атрибут, який посилається на предка. На рисунку 1 видно, що усі факультети належать до університету. Екземпляр університет може бути в принципі взагалі відсутнім – приймаємо, що нащадки першого рівня (факультети) завжди мають одного і того ж предка (університет), тому зберігати інформацію про нього не обов'язково.

Розглянемо структуру ієрархічної таблиці (рис.2).

Рис. 2. Структура таблиці, що містить ієрархічні дані

Поле типу предок завжди посилається на значення поля ідентифікатор. Якщо вважати, що посилання відбувається на значення поля, яке вже існує, то відповідно до реляційних правил можна оголосити зв'язок конструкцією SQL "alter table . add constraint . foreign key", але атрибути предок та ідентифікатор належать одному кортежу. Тому виникають питання: Як створити екземпляр об'єкту якщо предка немає? Чи може існувати екземпляр об'єкту без предка? Щоб позбавитися на початковому етапі від першого питання, треба відмовитися від декларації зв'язку предок-ідентифікатор на рівні сервера. На друге питання можна однозначно відповісти "ні", встановивши обмеження цілісності для поля типу предок як обов'язкове значення (NOT NULL).

Для виконання вибірок, що часто використовуються, потрібна підтримка рекурсивних запитів, яка з'явилася у стандарті мови SQL-1999. Якщо СУБД не підтримує таких запитів, то вибірки доведеться будувати з використанням механізмів тимчасових таблиць і процедур (функцій), що зберігаються.

Для отримання ієрархічних даних використовується тимчасова таблиця, яка описується оператором WITH. Після цього з неї вибираються дані простим селектом. У загальному вигляді синтаксис приблизно такий:

WITH ім'я_аліасу_запиту [(список стовпців)]

AS (запит)

основний запит

Залежно від глибини дерева збільшується і текст запиту, тому представлення ієрархічних даних у вигляді списків суміжності призводить до труднощів у написанні запитів, крім того важко рухатися вгору чи вниз по ієрархії. Якщо висота дерева заздалегідь невідома, то для витягання усіх вузлів піддерева, для якого заданий вузол є вершиною, доведеться створювати рекурсивну процедуру. Деякі сервери баз даних не підтримують рекурсії, а інші мають обмеження на максимальну кількість рекурсивних викликів процедури (наприклад Firebird).

Було б непогано завантажити таблицю матриці суміжності в програму, і потім використовувати рекурсивну програму перетворення дерева, щоб побудувати модель вкладених множин (рис.3). Ця модель нам підходить краще, оскільки SQL - мова, орієнтована на множини. Корінь дерева – множина, що містить усі інші множини, і відношення предок-нащадок описуються приналежністю множини нащадків множині предка.

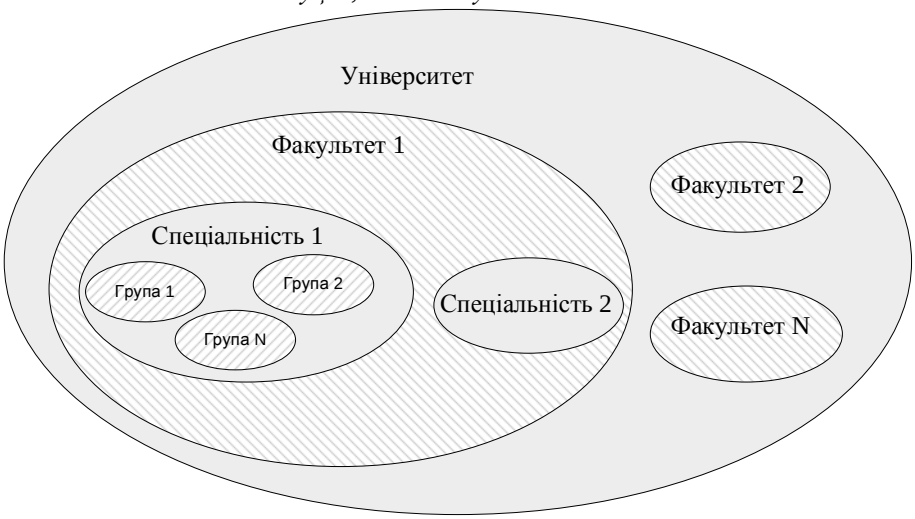

Рис.3. Представлення ієрархії у вигляді вкладених множин

Якщо таблицю, структура якої представлена на рис. 2, заповнити екземплярами об'єктів з рис. 1, то вона матиме вигляд:

Ідентифікатор	Предок	Інші атрибути
Університет	???	
Факультет 1	Університет	
Факультет 2	Університет	
Спеціальність 1	Факультет 1	
Група 1	Спеціальність 1	
…		

Як видно з таблиці, факультет 1 одночасно є і нащадком елементу університет, і предком елементу спеціальність 1.

Таку таблицю можна створити конструкцією SQL :

```
CREATE TABLE `kontingent` (
  `ID` int(11) NOT NULL,
  `parent` int(11) NOT NULL,
  `name` varchar(30) default NULL,
   PRIMARY KEY  (`ID`)
) ENGINE=InnoDB DEFAULT CHARSET=cp1251;
```

Нехай ідентифікатори екземплярів об'єктів починатимуться з номера 1. Тоді предком екземпляра університет можна вважати значення 0 (кореневий елемент). Фактично екземплярів з предком 0 може бути скільки завгодно, і саме вони представлятимуть корінь нашого дерева. Назви екземплярів нехай знаходяться в полі "name". Пронумеруємо ідентифікатори екземплярів об'єктів. Тоді таблиця матиме вигляд:

ID	parent	name
1	0	Університет
2	1	Факультет 1
3	1	Факультет 2
4	2	Спеціальність 1
5	4	Група 1

Тоді, щоб отримати з таблиці kontingent усі кореневі елементи, достатньо виконати запит:

```
SELECT * FROM kontingent
  WHERE PARENT = 0
```

Результат виконання цього запиту матиме вигляд:

ID	parent	name
1	0	Університет

Для того, щоб отримати усіх нащадків наприклад екземпляра університет, треба використовувати його ID в тому ж самому запиті як ідентифікатор предка:

```
SELECT * FROM kontingent
  WHERE PARENT =1
```

Результат виконання цього запиту матиме вигляд:

ID	parent	name
2	1	Факультет 1
3	1	Факультет 2

Таким чином отримати список записів будь-якого рівня можна одним і тим же запитом:

```
ВИБРАТИ УСІ_ПОЛЯ З ТАБЛИЦІ
ДЕ ПРЕДОК= ІДЕНТИФІКАТОР
```

Перевірити чи входить вузол в піддерево можна наступним запитом:

```
SELECT
 CASE
   WHEN EXISTS(
    SELECT 1 FROM kontingent
     WHERE ID = 4 /* вузол */
          AND parent = 2 /* корінь піддерева */)
    THEN 'вузол належить піддереву'
   ELSE 'вузол не належить піддереву'
END
```

В порівнянні зі списками суміжності, методу вкладених множин має значну надлишковість, адже на кожен рівень знадобиться стільки додаткових записів в таблиці підмножин, скільки елементів знаходиться на цьому рівні дерева, помноженому на номер рівня (вершину вважаємо першим рівнем). Кількість записів зросте в арифметичній прогресії.

Надлишковість зберігання даних можна оцінити як:

$$\sum_{i=1}^{N} Count(i) \cdot i,$$

де $Count(i)$ – кількість вузлів на i-му рівні дерева, починаючи з кореня;

N – число рівнів у дереві.

Щоб конвертувати модель матриці суміжності в ліс дерев, кожне з яких – модель вкладених множин, ідентифікована її кореневим значенням можна скористатися наступним кодом:

```
CREATE TABLE Tree(
Id CHAR (10),
parent CHAR(10),
lft INTEGER NOT NULL DEFAULT 0,
rgt INTEGER NOT NULL DEFAULT 0);

INSERT INTO Tree
SELECT P0.parent, P0.parent, 1,
2 * (SELECT COUNT(*)
FROM kontingent AS P1
WHERE P0.parent = P1.parent)
FROM kontingent AS P0;

INSERT INTO Tree
SELECT DISTINCT P0.ID, P0.parent,
2 * (SELECT COUNT(DISTINCT ID)
FROM kontingent AS P1
WHERE P1.ID < P0.ID
AND P0.parent IN (P1.ID, P1.parent)),
2 * (SELECT COUNT(DISTINCT ID)
FROM kontingent AS P1
WHERE P1.ID < P0.ID
AND P0.parent IN (P1.parent, P1.ID))
FROM kontingent AS P0;
```

```
DELETE FROM Tree
WHERE parent= 0 OR ID = 0;
```

Таким чином, ієрархічне дерево факультету 1 - набір рядків, які мають його як предка, генеалогічне дерево факультету 2 - набір рядків, які мають його як предка, і так далі:

Id	parent	lft	rgt
1	1	1	4
1	1	1	4
2	2	1	2
2	1	2	2
3	1	4	4
4	4	1	2
4	2	2	2
5	4	2	2

Щоб об'єднати отримані множини в одне дерево потрібно знайти спосіб прикріпити підпорядковане дерево до його предка. Використовуючи процедурну мову програмування можна було б застосувати наступний алгоритм:

Знайти розмір підпорядкованого дерева.

Знайти місце, куди підпорядковане дерево вставляється в дерево-предок.

Розсунути дерево-предок в точці вставки.

Вставити підпорядковане дерево в точку вставки.

Непроцедурна мова змушує виконувати ці кроки разом, використовуючи логіку усіх перерахованих пунктів.

Виявити пари зовнішніх і підпорядкованих дерев в таблиці Tree дуже просто. Наступний запит стає порожнім, коли усі предки встановлені в одне і теж значення:

```
CREATE VIEW AllPairs (superior, subordinate)
AS
SELECT W1.parent, W1.ID
FROM Tree AS W1
WHERE EXISTS( SELECT * FROM Tree AS W2 WHERE W2.parent = W1.ID)
AND W1.parent <> W1.ID;
```

Висновки. Найскладніша частина обробки дерев в SQL це знаходження способу конвертувати модель матриці суміжності в модель вкладених множин в межах структури чистого SQL. Використання методу вкладених множин дозволяє значно спростити запити, при цьому не обмежуючи глибини ієрархічного дерева, проте призводить до надмірної надлишковості при зберіганні даних.

1. Джо Селко. Стиль программирования Джо Селко на SQL. Пер. с англ. СПб.: Питер, 2006.
2. "Древовидные (иерархические) структуры данных в реляционных базах данных", Кузьменко Дмитрий, iBase – http://www.ibase.ru/devinfo/treedb.htm.
3. Пасічник В.В., Резніченко В.А., Організація баз даних та знань. – К.: Видавнича група BHV, 2006. – 384с.
4. Joe Celko. Trees in SQL. Some answers to some common questions about SQL trees and hierarchies http://www.intelligententerprise.com/001020/celko.jhtml?_requestid=1266295
5. Vadim Tropashko. Trees in SQL: Nested Sets and Materialized Path