ПРИКЛАДНІ ПРОБЛЕМИ КОМП'ЮТЕРНИХ НАУК,
БЕЗПЕКИ ТА МАТЕМАТИКИ, 2025, № 4

УДК 004.9

ОПЕРАЦІЇ ОНОВЛЕННЯ ІЄРАРХІЧНИХ СТРУКТУР У МОДЕЛІ ВКЛАДЕНИХ МНОЖИН

Лайтарук І.Ф. (ORCID: 0009-0002-9832-7759)
Куротич А.О. (ORCID: 0009-0006-8186-4063)
Булатецька Л.В. (ORCID: 0000-0002-7202-826X),
Волинський національний університет імені Лесі Українки, Луцьк, Україна

UPDATE OPERATIONS FOR HIERARCHICAL STRUCTURES IN THE NESTED SETS MODEL

Laitaruk, I.F., Kurotych, A.O., Bulatetska L.V.
Lesya Ukrainka Volyn National University, Lutsk, Ukraine

Abstract.
The article examines an approach to storing hierarchical data in relational databases using the Nested Sets model. The fundamental principles of this model, its advantages and disadvantages compared to other methods, particularly the Adjacency List, are analyzed. Special attention is given to subtree movement and node deletion algorithms within the hierarchical structure, including the update of lft and rgt values. Examples of SQL queries for subtree movement and node deletion are provided. The Nested Sets model is advisable for use in relational databases when efficient querying of hierarchical data is required. However, inserting, moving, and deleting nodes require updating multiple records due to changes in lft and rgt values. Even if restructuring is performed infrequently, it can be very slow for large trees. Understanding the restructuring process helps determine whether Nested Sets should be used in a specific case or if alternative approaches to storing hierarchical data in relational databases should be considered.
Keywords: Hierarchical data, relational database, Nested Sets, subtree relocation, SQL.

Вступ. Існує кілька поширених методів подання ієрархічних структур у реляційних базах даних, зокрема Adjacency List, Nested Sets, Closure Table і Materialized Path [1–5]. Для забезпечення ефективної роботи з ієрархічними даними важливо зберігати їх у форматі, який мінімізує час вибірки та витрати ресурсів, необхідні для цього процесу.

У роботі [3] проведено аналіз ефективності операцій управління ієрархічними структурами в реляційних базах даних на основі різних моделей подання, таких як Adjacency List, Nested Sets, Closure Table та Materialized Path. Зокрема, розглянуто виконання таких запитів:

- виведення листків дерева,
- отримання прямих дочірніх вузлів,
- отримання батьківського вузла,
- знаходження шляху між двома вузлами,
- отримання піддерева,
- додавання нового листка дерева,
- видалення вузла-листка,
- видалення піддерева з заданою вершиною.

У роботі [3] проведено аналіз продуктивності виконання запитів для різних методів зберігання ієрархічних структур у реляційних базах даних, зокрема виміряно час вибірки даних для кожного з підходів. Швидкість виконання основних операцій, таких як додавання, видалення та переміщення вузлів, значною мірою залежить від обраної моделі.

Модель Nested Sets доцільно використовувати в реляційних базах даних, коли потрібно ефективно виконувати запити на вибірку ієрархічних даних. Запити на вибірку всього дерева, або піддерева (наприклад, отримання всіх підкатегорій у дереві категорій), виведення всіх предків вузла, або підрахунок кількості нащадків вузла виконуються одним SQL-запитом без рекурсії, що може бути вигідно в системах із великою кількістю зчитувань [3]. У випадках, коли дерево містить велику кількість записів, використання рекурсивних запитів (як у Adjacency List) може бути неефективним. Nested Sets дозволяє отримувати всі необхідні дані за допомогою простого JOIN або WHERE умови на межах (left-right). Проте вставка, переміщення та видалення вузлів потребують оновлення багатьох записів, оскільки змінюються значення lft і rgt. Навіть якщо перебудова виконується рідко, для великих дерев вона може бути дуже повільною. Розуміння процесу перебудови допомагає визначити, чи варто використовувати Nested Sets у конкретному випадку або ж варто розглянути альтернативні підходи до зберігання ієрархічних даних у реляційних базах.

Метою роботи є розглянути методи зміни структури моделі Nested Sets для представлення ієрархічних даних.

Методологія дослідження.

В цій роботі для моделювання ієрархічних структур було обрано реляційну СУБД Oracle Database 18c XE (Express Edition). Сервер баз даних був розгорнутий на платформі віртуалізації Oracle VirtualBox 7.0.18 з гостьовою системою 2xCPU core, 4GB RAM, MS Windows 10 Pro. Базова система: Intel Core i3, 16GB RAM, SSD NVMe 512GB, MS Windows 11 Pro. Робоча станція була представлена ноутбуком Lenovo ThinkPad T450 з процесором Intel Core i5, 8GB RAM, SSD 180GB SATA-3, MS Windows 10 Pro. В якості комунікаційного середовища використано Mesh WiFi-5 на базі трьох тридіапазонних юнітів Linksys з швидкістю до 3Gbps.

Результати дослідження та їхнє обговорення

Модель вкладених множин (Nested Sets) це спосіб представлення дерев, який полягає в їх відображенні у вигляді вкладених множин (рис. 1) [1,2,3,6]. Кожен вузол дерева має 2 значення: Left (значення ліворуч від вузла) та Right (значення праворуч від вузла). Процедура визначення цих значень полягає у обході дерева ліворуч праворуч і нарощуванні лічильника на 1 під час проходження вузла. На рис. 1 пунктирними стрілками показано процес обходу дерева. Перевагами цієї моделі є проста операція отримання нащадків без рівня вкладеності та проста операція отримання батьківських вузлів. [3, 6].

Наведемо формули обчислення лівої (*lft*) та правої (*rgt*) межі вузлів дерева.

Для кореневого вузла:

lft =1,
rgt=2×N,
де *N* – загальна кількість вузлів у дереві.

Для кожного дочірнього вузла V, що має батьківського вузла P:

$$lft(\mathrm{V}) = lft(\mathrm{P}) + 1 + 2 * n1,$$

де $n1$ – кількість вузлів у піддеревах, що знаходяться зліва від вузла V і теж мають батьківського вузла P.

$$rgt(\mathrm{V}) = lft(\mathrm{V}) + 1 + 2 * n2,$$

де $n2$ – кількість всіх дочірніх вузлів вузла V.

```
CREATE TABLE CATEGORY_Nested_Sets
(id INTEGER PRIMARY KEY,
title VARCHAR2(5),
lft INTEGER NOT NULL,
rgt INTEGER NOT NULL
);
```

ID	TITLE	LFT	RGT
1	A	1	24
2	B	2	5
3	C	6	11
4	D	12	23
5	E	3	4
6	F	7	8
7	G	9	10
8	H	13	20
9	I	21	22
10	J	14	15
11	K	16	17
12	L	18	19

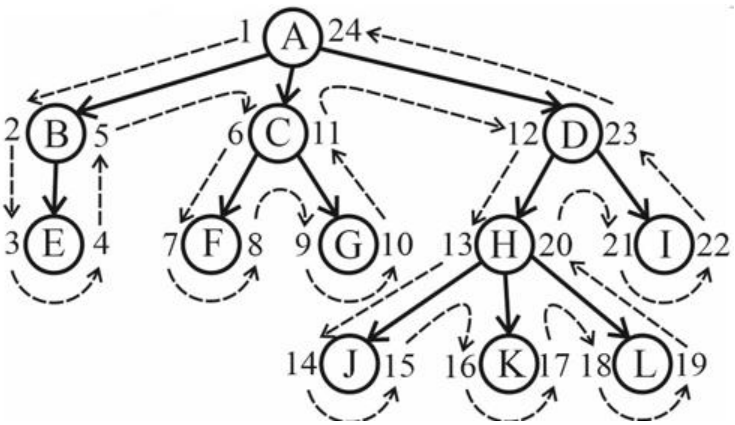

Рис. 1. Дерево змодельоване у вигляді вкладених множин з елементами від A до L, які зберігаються в таблиці CATEGORY_Nested_Sets [3].

Наведемо алгоритм переміщення піддерева до іншого батьківського вузла.

1. Отримуємо `lft`, `rgt` піддерева, що переміщається.
2. Обчислюємо ширину (`width`).
3. Отримуємо `lft`, `rgt` нового батьківського вузла.
4. Отримуємо `lft` вузла, після якого треба вставити піддерево.
5. Коригуємо `lft`, `rgt` для інших вузлів.
6. Переміщуємо піддерево.

На рис. 2. подано схему переміщення піддерева до іншого вузла, якщо цей вузол є листком дерева. Алгоритм перебудови лівої та правої межі вузлів дерев, які розміщені між батьківськими вузлами від якого переміщуємо піддерево і куди переміщуємо дерево залежить від того чи вліво переміщується дерево чи вправо.

Для того, щоб виконати перебудову дерева необхідно десь зберігати дані про піддререво, яке переміщується. Тому оголосимо тип ids_table, визначення PL/SQL collection (таблиці чисел), яка буде використовуватися для збереження id вузлів, що належать переміщуваному піддереву.

```
create or replace type ids_table as table of number;
/
```

Основна процедура `move_subtree_nested_sets` приймає два параметри:

- `id_from` – кореневий вузол піддерева, яке потрібно перемістити;
- `id_to` – вузол, до якого переносимо піддерево.

```
create or replace procedure move_subtree_nested_sets(
    id_from in integer,
```

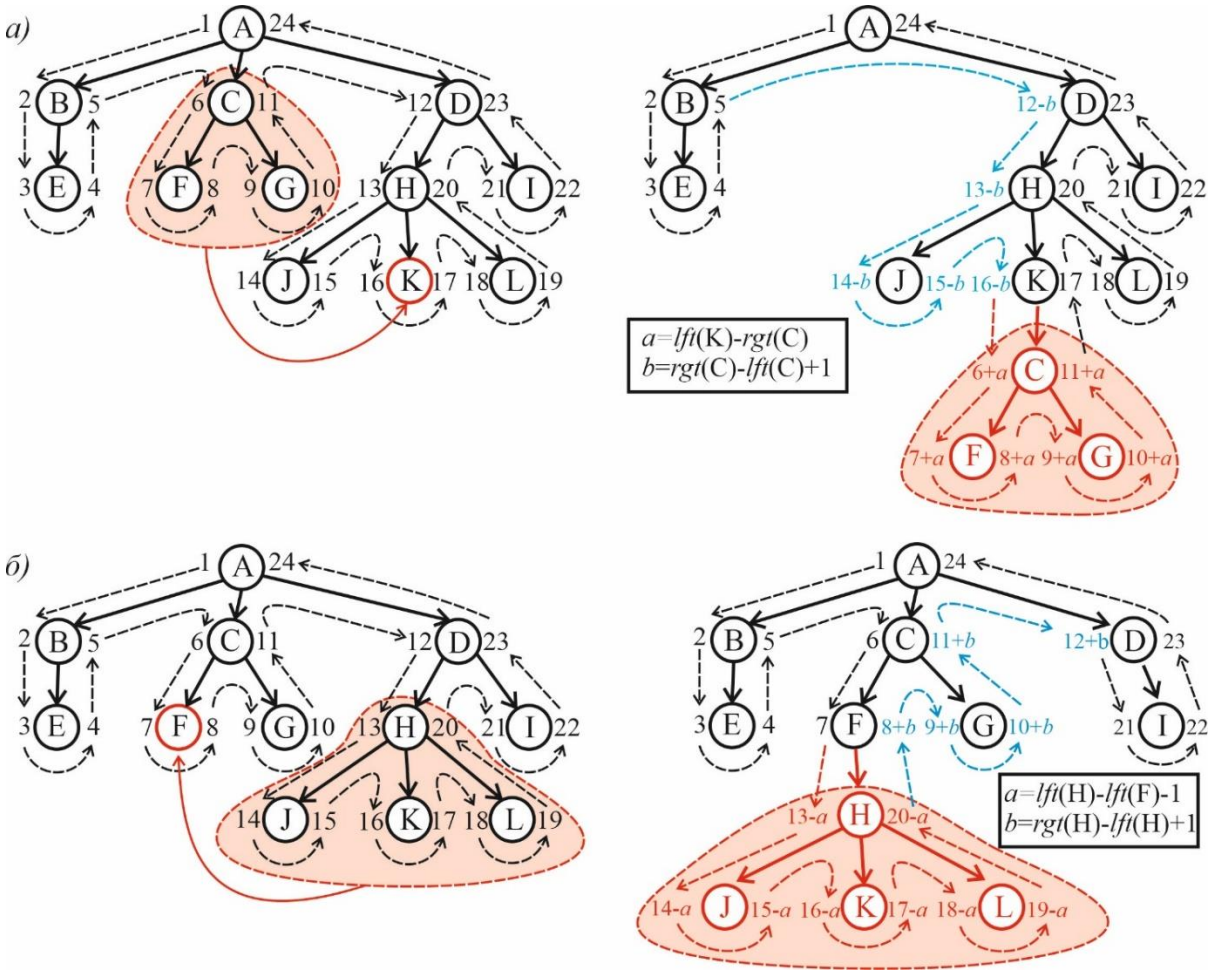

Рис. 2. Переміщення піддерева до іншого вузла, якщо цей вузол є листком дерева: *а*) переміщення піддерева вправо, *б*) переміщення піддерева вліво.

```
    id_to in integer
) is
    delta integer; -- Зсув, на який потрібно змінити значення lft та rgt
    subtree_node_amount integer; -- Кількість вузлів у піддереві

    -- Межі вихідного піддерева
    lft_from integer;

    rgt_from integer;

    --Межі нового вузла, куди переносимо піддерево
    lft_to integer;
    rgt_to integer;

    -- Колекція для збереження всіх вузлів, що переміщуються
    ids ids_table := ids_table();

begin
```

```
    -- Отримуємо межі вихідного (id_from) і цільового (id_to) вузлів.
    select lft, rgt into lft_from, rgt_from
    from category_nested_sets where id=id_from;

    select lft, rgt into lft_to, rgt_to
    from category_nested_sets where id=id_to;

    -- Обчислюємо параметри переміщення
    delta := lft_from - lft_to - 1; -- Зміщення вузлів піддерева.

 -- Кількість вузлів у піддереві
    subtree_node_amount := (rgt_from - lft_from + 1) / 2;
```

/*

Збереження всіх ID вузлів піддерева. Проходимо по всіх значеннях lft і rgt в межах переміщуваного піддерева. Шукаємо id вузла для кожного значення. Додаємо знайдені id до колекції ids.

*/

```
     for i in lft_from..rgt_from loop
        declare
            v_id number;
        begin
            select id into v_id
            from category_nested_sets
            where lft = i or rgt = i;

            ids.extend;
            ids(ids.count) := v_id;
        end;
    end loop;
```

/*

Переміщення піддерева в ліву сторону. Якщо delta > 0, значить, вузол переміщується вліво (зменшуємо значення lft і rgt).Оновлюємо lft і rgt для вузлів правіше нового місця.

*/

```
    if delta > 0
    then

    update category_nested_sets
    set lft = lft + subtree_node_amount * 2
    where lft > lft_to and lft < lft_from;

    update category_nested_sets
    set rgt = rgt + subtree_node_amount * 2
    where rgt > lft_to and rgt < lft_from;
```

/*

Наступні оновлення створюють місце для вставки піддерева. Переміщуємо піддерево, зменшуючи значення lft і rgt на delta

*/

```
    update category_nested_sets
     set rgt = rgt - delta, lft = lft - delta
     where id member of ids;

/*
        Переміщення піддерева вправо. Якщо delta <= 0, вузол переміщується вправо. Оновлюємо lft і rgt для вузлів, що знаходяться після піддерева.
*/

else

     delta := lft_to - rgt_from;
     update category_nested_sets
     set lft = lft - subtree_node_amount * 2
     where lft > rgt_from and lft <= lft_to;

     update category_nested_sets
     set rgt = rgt - subtree_node_amount * 2
     where rgt > rgt_from and rgt < lft_to;

    -- Наступні оновлення звільняють місце для вставки. Переміщуємо піддерево вправо.

    update category_nested_sets
     set rgt = rgt + delta, lft = lft + delta
     where id member of ids;

end if;
end;
/
```

На рис. 3. Подано результат роботи процедури переміщення піддерева, яке зображене на рис 2.

```
call move_subtree_nested_sets(3, 11); -- (рис. 3, а)
call move_subtree_nested_sets(8, 6);  -- (рис. 3, б)
```

Код не враховує випадок, коли вузол потрібно вставити між конкретними дочірніми вузлами нового батьківського елемента. На рис. 4 подано схему переміщення піддерева до іншого вузла, якщо цей вузол не є лиском дерева.

У роботі [3] подано процедуру для видалення вузла з усіма його нащадками в моделі ієрархічних вкладених множин (Nested Sets). Для цього необхідно визначити межі (lft і rgt) вузла, який потрібно видалити, видалити всі вузли, які мають межі, що знаходяться в межах вузла, який видаляється, оновити межі всіх вузлів, які залишилися, щоб видалити пробіли, які утворилися після видалення вузла і його нащадків.

Нехай нам потрібно видалити вузол, не видаляючи його піддерево. У такому разі всі дочірні елементи цього вузла мають отримати нового батька. Новим батьком стане батьківський елемент видаленого вузла. Це дозволить зберегти ієрархічну структуру без втрати даних. (рис. 5)

а)

ID	TITLE	LFT	RGT
1	A	1	24
2	B	2	5
3	C	11	16
4	D	6	23
5	E	3	4
6	F	12	13
7	G	14	15
8	H	7	20
9	I	21	22
10	J	8	9
11	K	10	17
12	L	18	19

б)

ID	TITLE	LFT	RGT
1	A	1	24
2	B	2	5
3	C	6	19
4	D	20	23
5	E	3	4
6	F	7	16
7	G	17	18
8	H	8	15
9	I	21	22
10	J	9	10
11	K	11	12
12	L	13	14

Рис. 3. Результат роботи процедури переміщення вузла в дереві поданого на рис. 1: *а*) переміщення піддерева вправо, *б*) переміщення піддерева вліво.

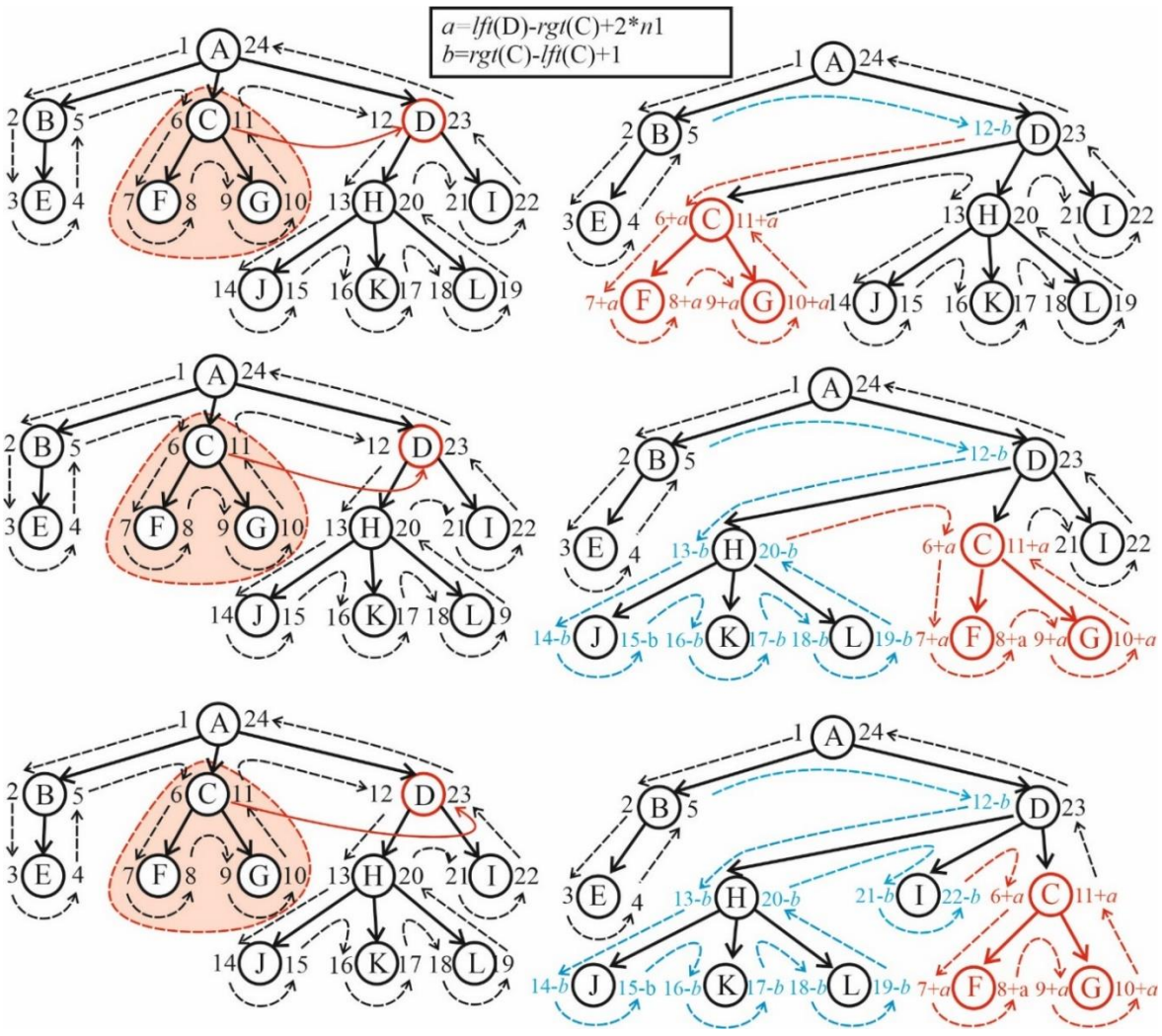

Рис. 4. Переміщення піддерева вправо до іншого вузла, якщо цей вузол не є листком дерева.

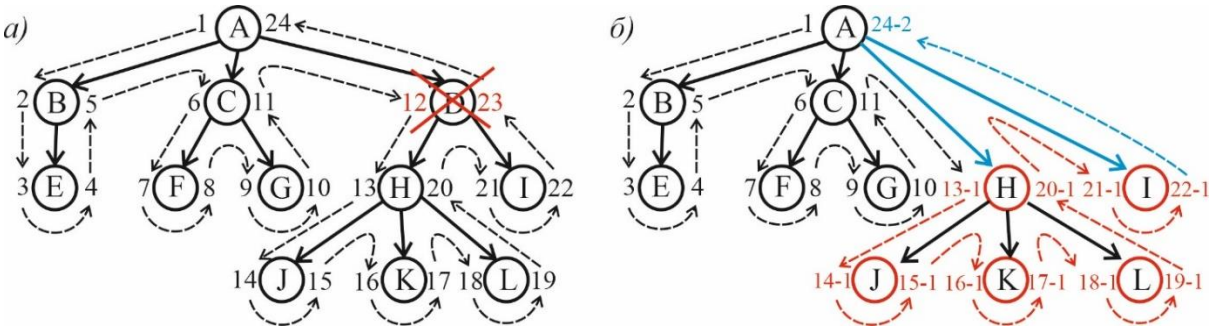

Рис. 5. Видалення вузла не видаляючи його піддерево.

Для цього нам потрібно:

- знайти межі вузла, який потрібно видалити, отримати значення lft та rgt цього вузла з таблиці;
- оновити позиції лівих меж (lft) дочірніх вузлів, зменшити lft на 1 для всіх вузлів, які знаходяться всередині піддерева вузла, що видаляється (між lft_+1 та rgt_-1);
- оновити позиції правих меж (rgt) дочірніх вузлів, зменшити rgt на 1 для всіх вузлів у піддереві, щоб зберегти правильну структуру вкладеності;
- оновити позиції лівих меж (lft) вузлів, розташованих праворуч від вузла, що видаляється, зменшити lft на 2 для всіх вузлів, які знаходяться праворуч від вузла (lft > rgt_);
- оновити позиції правих меж (rgt) вузлів, розташованих праворуч від вузла, що видаляється, зменшити rgt на 2 для всіх вузлів, які мають rgt > rgt_;
- видалити вузол з таблиці category_nested_sets, якщо його lft та rgt відповідають збереженим значенням.

Цей алгоритм дозволяє зберегти цілісність ієрархії після видалення вузла.

Наведемо процедуру, яка видаляє вузол з ієрархічної структури, що зберігається в моделі Nested Sets Model. Вона також коригує значення lft і rgt для інших вузлів у дереві, щоб зберегти коректність ієрархії. Результат роботи цієї процедури подано на рис. 6.

```
create or replace procedure delete_node_nested_sets(
    id_ in integer
) is
    lft_ integer;
    rgt_ integer;
begin

    -- Отримання меж вузла, який видаляємо (lft та rgt)
    select lft, rgt into lft_, rgt_ from category_nested_sets where
id=id_;

    -- Зміщення лівої межі для підлеглих вузлів
    update category_nested_sets
    set lft = lft - 1
    where lft between lft_+1 and rgt_-1;

    -- Зміщення правої межі для підлеглих вузлів
    update category_nested_sets
    set rgt = rgt - 1
```

```
    where rgt between lft_+1 and rgt_-1;

    -- Оновлення лівої межі для всіх вузлів, які знаходяться праворуч від видаленого вузла
    update category_nested_sets
    set lft = lft - 2
    where lft > rgt_;

    -- Оновлення правої межі для всіх вузлів, які знаходяться праворуч від видаленого вузла
    update category_nested_sets
    set rgt = rgt - 2
    where rgt > rgt_;

    -- Видалення самого вузла
    delete from category_nested_sets where lft=lft_ and rgt=rgt_;
end;
/
```

ID	TITLE	LFT	RGT
1	A	1	22
2	B	2	5
3	C	6	11
5	E	3	4
6	F	7	8
7	G	9	10
8	H	12	19
9	I	20	21
10	J	13	14
11	K	15	16
12	L	17	18

Рис. 6. Результат роботи процедури видалення вузла D
дерева, не видаляючи його піддерево.

Висновки. У цій роботі розглянуто модель Nested Sets як один із методів зберігання ієрархічних даних у реляційних базах даних. Цей підхід значно покращує продуктивність запитів на вибірку ієрархічних структур, дозволяючи отримувати цілі піддерева або визначати предків вузла без використання рекурсії. Однак модель Nested Sets має свої обмеження. Основною проблемою є висока вартість операцій оновлення структури, зокрема переміщення, вставки та видалення вузлів, оскільки вони вимагають зміни значень lft та rgt у багатьох записах. Це може негативно впливати на продуктивність у випадках, коли структура дерева часто змінюється.

Розглянуті алгоритми переміщення піддерев та видалення вузла демонструють, що хоч оновлення структури є складним, його можна реалізувати за допомогою відповідних SQL-запитів. Вибір Nested Sets як методу зберігання ієрархічних даних залежить від характеру навантаження. Цей підхід підходить для систем, де переважають операції читання, але може бути менш ефективним у середовищах з частими оновленнями даних.

Таким чином, застосування Nested Sets є доцільним у випадках, коли важливо мінімізувати витрати на вибірку даних, але слід ретельно оцінювати обсяг необхідних оновлень для підтримки структури дерева. У деяких випадках альтернативні методи, такі як Adjacency List, Closure Table або Materialized Path, можуть бути більш придатними.

Бібліографія

1. Joe Celko's Trees and Hierarchies in SQL for Smarties. (2004). Elsevier. https://doi.org/10.1016/b978-1-55860-920-4.x5000-4
2. Celko J. Trees and Hierarchies in SQL for Smarties. – Morgan-Kaufmann, 2012. – 296 p.
3. Булатецька Л. В., Булатецький В. В. Методи моделювання ієрархічних структур в реляційних базах даних. *Прикладні проблеми комп'ютерних наук, безпеки та математики.* 2024. № 1. С. 47–65. URL: https://apcssm.vnu.edu.ua/index.php/Journalone/article/view/126
4. Зберігання ієрархічих структур в реляційних базах даних. / В. О. Маркітан, М. А. Возняк, Л. В. Булатецька, В. В. Булатецький. *Кібербезпека: освіта, наука, техніка.* 2022. Т. 4, №16. С. 85–97. DOI: https://doi.org/10.28925/2663-4023.2022.16.8597
5. Маркітан В. О., Возняк М. А., Булатецька Л. В. Деревовидні структури в SQL. *Математика. Інформаційні технології. Освіта.*: матеріали XI міжнар. науково-практ.конф. Луцьк, 3–5 червн. 2022 р. Луцьк, 2022. С. 109–111.
6. Дерево каталогів NESTED SETS (вкладені множини) і управління ним | open2web. *open2web | open2web*. URL: https://open2web.com.ua/blog/derevo-katalogov-nested-sets-vlozhennye-mnozhestva-i-upravlenie-im.html (дата звернення: 10.08.2024).

References

1. Joe Celko's Trees and Hierarchies in SQL for Smarties. (2004). Elsevier. https://doi.org/10.1016/b978-1-55860-920-4.x5000-4
2. Celko J. Trees and Hierarchies in SQL for Smarties. – Morgan-Kaufmann, 2012. – 296 p.
3. Bulatetska L.V., Bulatetskyi V. V. Methods Of Modeling Hierarchical Structures In Relational Databases. Applied Problems of Computer Science, Security and Mathematics. 2024. № 1. P. 47–65. URL: https://apcssm.vnu.edu.ua/index.php/Journalone/article/view/126
4. Preservation Of Hierarchy Structures In Relative Databases / V. O. Markitan, M. A. Vozniak, L. V. Bulatetska, V. V. Bulatetskyi. Cybersecurity: Education, Science, Technique. 2022. Vol. 4, №16. P. 85–97. DOI: https://doi.org/10.28925/2663-4023.2022.16.8597
5. Markitan V. O., Vozniak M. A., Bulatetska L. V. Derevovydni struktury v SQL. Matematyka. Informatsiini tekhnolohii. Osvita. : materialy XI mizhnar. naukovo-prakt. konf. Lutsk, 3–5 chervn. 2022 r. Lutsk, 2022. S. 109–111.
6. Derevo katalohiv NESTED SETS (vkladeni mnozhyny) i upravlinnia nym | open2web. open2web | open2web. URL: https://open2web.com.ua/blog/derevo-katalogov-nested-sets -vlozhennye-mnozhestva-i-upravlenie-im.html