КІБЕРБЕЗПЕКА: освіта, наука, техніка
CYBERSECURITY: EDUCATION, SCIENCE, TECHNIQUE
№ 4 (16), 2022
ISSN 2663 - 4023
DOI 10.28925/2663-4023.2022.16.8597
УДК 004.05

Маркітан Володимир Олегович
студент факультету інформаційних технологій і математики
Волинський національний університет імені Лесі Українки, Луцьк, Україна
ORCID ID: 0000-0002-3589-8784
Markitan.Volodymyr2020@vnu.edu.ua

Возняк Микола Андрійович
студент факультету інформаційних технологій і математики
Волинський національний університет імені Лесі Українки, Луцьк, Україна
ORCID ID: 0000-0002-3589-8784
Vozniak.Mykola2020@vnu.edu.ua

Булатецька Леся Віталіївна
Кандидат фіз.-мат. наук, доцент, доцент кафедри комп'ютерних наук та кібербезпеки
Волинський національний університет імені Лесі Українки, Луцьк, Україна
ORCID ID: 0000-0002-7202-826X
Bulatetska.Lesya@vnu.edu.ua

Булатецький Віталій Вікторович
Кандидат фіз.-мат. наук, доцент, доцент кафедри комп'ютерних наук та кібербезпеки
Волинський національний університет імені Лесі Українки, Луцьк, Україна
ORCID ID: 0000-0002-9883-4550
Bulatetsky.Vitaly@vnu.edu.ua

ЗБЕРІГАННЯ ІЄРАРХІЧИХ СТРУКТУР В РЕЛЯЦІЙНИХ БАЗАХ ДАНИХ

Анотація. Системи управління реляційними базами даних і сама мова SQL не мають жодних вбудованих механізмів для зберігання та управління ієрархічними структурами. Існує кілька різних способів представлення дерев в реляційних базах даних. В роботі розглянуто метод моделювання ієрархічних структур даних у вигляді списків суміжності (Adjacency List) та таблиці зв'язків (Closure Table). Для кожного методу, подано приклади написання запитів для розв'язання типових завдань, які зустрічаються під час роботи з деревовидними структурами: пошук всіх дочірніх вузлів, всіх нащадків та предків заданого вузла, переміщення вузла до іншого батьківського вузла, видалення вузлів з усіма його нащадками. Розглянуто можливість використання рекурсивних запитів при відображення всього дерева в моделі списків суміжності. У випадку, якщо глибина дерева не відома, або не відомо на якому рівні знаходиться заданий елемент, то запит не може бути побудований стандартними засобами оператора SELECT, тоді потрібно створювати рекурсивну процедуру, або писати рекурсивний запит. Для того, щоб уникнути рекурсії при виведенні всього дерева, всіх вузлів піддерева та пошук шляху від певного місця до кореня, моделювання ієрархічних структур даних виконують у вигляді таблиці зв'язків (Closure Table). При цьому ускладнюється процес додавання нового вузла, переміщення вузла до іншого батьківського вузла. В такому випадку для спрощення написання запитів пропонується створювати тригери, які будуть будувати, або перебудовувати зв'язки. Враховуючи те, що іноді виникає потреба збереження залежних, зокрема ієрархічних структур в реляційній базі даних, потрібно вміти орати модель збереження таких даних. На вибір методу, для розв'язання конкретної задачі, впливає швидкість виконання основних операцій з деревами. Дослідження різних варіантів організації деревоподібних структур SQL дозволить зрозуміти та обрати самий оптимальний спосіб побудови такої структури в реляційній базі даних для конкретної задачі. Усі, наведені в даній роботі SQL запити, створювалися та тестувалися для реляційних баз даних Oracle.

Ключові слова: база даних; ієрархічні структури даних; дерево; модель суміжних списків; модель таблиці зв'язків.

ISSN 2663 - 4023

ВСТУП

Постановка проблеми.

Реляційні бази даних орієнтовані на організацію зберігання інформації у вигляді таблиці, де кожен рядок містить дані про один окремий об'єкт, а кожен стовпець – характеристики цих об'єктів. При такій організації, рядки є незалежними один від одного. Однак дуже часто виникає потреба збереження залежних, зокрема деревоподібних структур в базі даних. Деревом називається зв'язний неорієнтований граф, що не містить циклів, тобто петель із замкнутих шляхів [1, 2]. Дерева застосовуються практично в кожній галузі програмування. Зазвичай за допомогою дерев зображають ієрархію даних: представлення структури каталогів і файлів на жорсткому диску, подання структури підрозділів і співробітників компаній, структуру сторінок веб-сайту і т.п. Зберігання ієрархічних структур у базі даних, не є таким простим, оскільки система управління реляційними базами даних і сама мова SQL не мають жодних вбудованих механізмів для зберігання та управління такими структурами [3].

Аналіз останніх досліджень і публікацій.

Деревовидні структури в реляційних базах даних можна зберігати декількома способами. На вибір методу, для розв'язання конкретної задачі, впливає швидкість виконання основних операцій. Типові завдання, які зустрічаються під час роботи з деревовидними структурами:

- знайти всі дочірні елементи;
- знайти батьківський елемент;
- знайти всіх нащадків (дітей та їхніх дітей) елемента;
- знайти всіх предків елемента (батько, його батько, і так далі)
- перемістити елемент (і його нащадків) від одного батьківського елемента до іншого;
- видалити елемент із таблиці (з усіма нащадками).

У роботах [4, 5] розглянуто метод моделювання ієрархічних структур даних у вигляді списків суміжності. Наведено приклади таких списків та їхні типи. Розглянуто можливості мови SQL по формулюванню рекурсивних запитів при зверненні до ієрархічних структур даних та наведено приклади таких запитів. Задано формальну семантику рекурсивних загальних табличних виразів. Рекурсивні запити при роботі з ієрархічними структурами, дозволяють створювати складні запити, зберігаючи набагато простіший синтаксис. Приклади написання рекурсивних запитів та їх різновиди, розглядаються в роботах [6,7]. Інший спосіб представлення дерев – подати їх як вкладені множини. Оскільки SQL є мовою, орієнтованою на множини, ця модель є кращою, ніж звичайний підхід до списку суміжності, який зустрічається більшості підручників. У роботі [8] розглядаються проблеми, які виникають в моделі списку суміжності і розглядаються способи вирішення даних проблем для моделі вкладених множин. В роботі [9] розроблено додаток, для тестування методів роботи з різними деревоподібними структурами в SQL. Для визначення оптимального методу проводився аналіз швидкості роботи, складності вставки нового вузла, переміщення вузла в структурі, підтримка цілісності даних, видалення вузла. В роботі [10] Розглянуто задачу збереження, модифікації та видобування ієрархічних даних у реляційних базах даних MS SQL Server. Проведено аналіз ефективності ієрархічних структур на підставі списку суміжних вершин і модифікованої моделі вкладених множин. У роботі [11] детально описано створення та використання ключів упорядкування вузлів дерева в одній таблиці реляційної бази даних. Ключі для кожного вузла обчислюються з ключів його

батьківського вузла таким чином, що порядок сортування розміщує кожен вузол у дереві перед усіма його нащадками і після всіх братів і сестер, які мають нижчий індекс.

Метою статті є розглянути задачу побудови, збереження, модифікації та видобування ієрархічних даних на прикладі моделей списку суміжних вершин (Adjacency List) і таблиці зв'язків (Closure Table) у реляційних базах даних Oracle.

РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

Одним із найпоширеніших способів побудови дерева в реляційній базі даних є моделювання ієрархічних структур даних у вигляді списків суміжності (Adjacency List). При такому моделюванні, кожен запис в таблиці відповідає вузлу дерева і зберігає його унікальний ідентифікатор (поле id, яке являється первинним ключем) та посилання на батьківський вузол (поле parent_id, яке описується як зовнішній ключ, що посилається на первинний ключ цієї ж таблиці). Якщо вузол немає батьківського елемента, тобто він є коренем дерева, то атрибут parent_id має особливе значення NULL. Такі ієрархічні відношення ще називають відношеннями предок-нащадок. Подання ієрархічної структури у вигляді 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 [1].

SQL запит на створення таблиці CATEGORY, яка представляє описану деревовидну структуру для бази даних Oracle, можна записати наступним чином:

```
CREATE TABLE CATEGORY (
id INTEGER PRIMARY KEY,
title VARCHAR2(5) NOT NULL,
parent_id INTEGER REFERENCES CATEGORY ON DELETE CASCADE
);
```

Використання зовнішніх ключів забезпечує збереження посилкової цілісності бази даних при зміні та видаленні записів, тому виникає проблема при видаленні вузла дерева, яке має нащадків, що пов'язані з даним вузлом зовнішніми ключами. При видаленні вузла такого дерева, для забезпечення автоматичного видалення всього піддерева потрібно використати правило ON DELETE CASCADE.

За допомогою наступного запиту можна побудувати дерево з елементами від A до L, які зберігаються в полі title (рис. 1). Для цього в створену таблицю додаємо записи:

```
INSERT ALL
INTO CATEGORY VALUES (1, 'A', Null)
INTO CATEGORY VALUES (2, 'B', 1)
INTO CATEGORY VALUES (3, 'C', 1)
INTO CATEGORY VALUES (4, 'D', 1)
INTO CATEGORY VALUES (5, 'E', 2)
INTO CATEGORY VALUES (6, 'F', 3)
INTO CATEGORY VALUES (7, 'G', 3)
INTO CATEGORY VALUES (8, 'H', 4)
INTO CATEGORY VALUES (9, 'I', 4)
INTO CATEGORY VALUES (10, 'J', 8)
INTO CATEGORY VALUES (11, 'K', 8)
INTO CATEGORY VALUES (12, 'L', 8)
SELECT * FROM dual;
```

ISSN 2663 - 4023

В результаті отримаємо ієрархічну структуру подану в таблиці CATEGORY (рис.1).

```
SQL> select * from  CATEGORY;

        ID TITLE  PARENT_ID
---------- ----- ----------
         1 A
         2 B              1
         3 C              1
         4 D              1
         5 E              2
         6 F              3
         7 G              3
         8 H              4
         9 I              4
        10 J              8
        11 K              8
        12 L              8

12 rows selected.
```

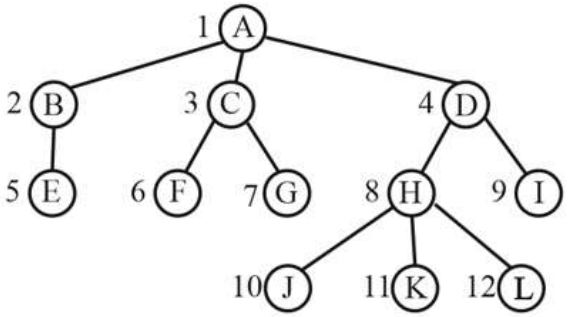


Рис.1 Дерево з елементами від A до L, які зберігаються в таблиці CATEGORY

Щоб знайти всі листки дерева, потрібно виконати SQL-запит, який шукає всі елементи дерева, які не мають дітей. Цей запит є універсальним для будь-якого дерева, незалежно від його розміру, рівнів чи западин. Звичайно, для великих дерев це виконання буде масивнішим.

```
SQL> SELECT t1.title FROM
  2  CATEGORY t1 LEFT JOIN CATEGORY t2 ON (t1.id = t2.parent_id)
  3  WHERE t2.id IS NULL;

TITLE
-----
E
L
J
K
F
G
I

7 rows selected.
```

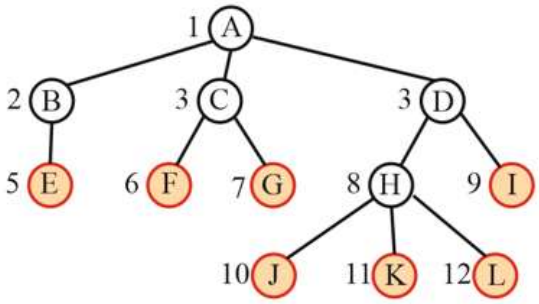


Рис. 2. Знаходження всіх листків дерева

Вибірку дочірніх елементів батьківського вузла можна здійснити маючи значення parent_id (рис. 3).

```
SQL> SELECT title FROM CATEGORY WHERE parent_id=8;

TITLE
--------
J
K
L
```

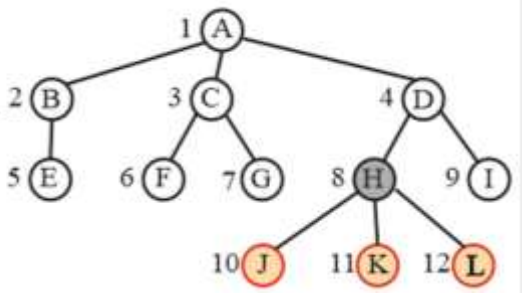


Рис. 3. Вибірка дочірніх елементів батьківського вузла H.

Виведення нащадків нащадка батьківського вузла здіснюється шляхом з'єднання таблиці самої на себе. Відображення всього дерева є досить складним, оскільки ми повинні об'єднати таблицю саму з собою, стільки разів, скільки рівнів має наше дерево (рис. 4).

```
SQL> SELECT t1.title AS level_1, t2.title AS level_2, t3.title AS level_3, t4.title AS level_4
  2  FROM CATEGORY t1
  3  LEFT JOIN CATEGORY t2 ON (t2.parent_id = t1.id)
  4  LEFT JOIN CATEGORY t3 ON (t3.parent_id = t2.id)
  5  LEFT JOIN CATEGORY t4 ON (t4.parent_id = t3.id) WHERE t1.title = 'A';

LEVEL_1  LEVEL_2  LEVEL_3  LEVEL_4
-------- -------- -------- --------
A        D        H        J
A        D        H        K
A        D        H        L
A        C        F
A        C        G
A        B        E
A        D        I

7 rows selected.
```

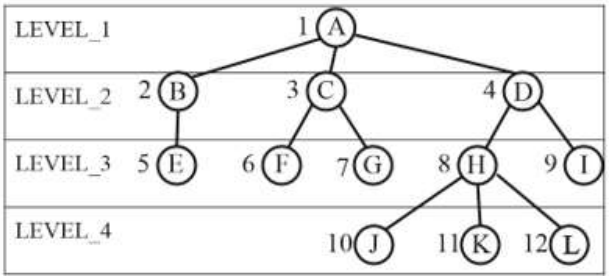


Рис. 4. Відображення всього дерева, якщо відома його глибина.

Щоб знайти шлях від певного місця дерева до кореня, необхідно запустити запит, побудований так само, як і запит, який відображає все дерево. В якості умови ми вказуємо елемент від якого і до якого ми шукаємо, тобто діапазон пошуку (рис.5).

```
SQL> SELECT t1.title AS level_1, t2.title AS level_2,
  2         t3.title AS level_3, t4.title AS level_4
  3  FROM CATEGORY t1
  4  LEFT JOIN CATEGORY t2 ON (t2.parent_id = t1.id)
  5  LEFT JOIN CATEGORY t3 ON (t3.parent_id = t2.id)
  6  LEFT JOIN CATEGORY t4 ON (t4.parent_id = t3.id)
  7  WHERE t1.title = 'A' AND t4.title = 'K';

LEVEL_1  LEVEL_2  LEVEL_3  LEVEL_4
-------- -------- -------- --------
A        D        H        K
```

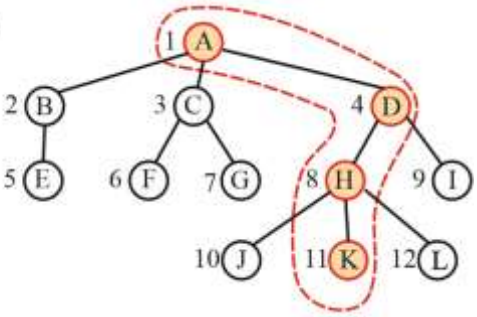


Рис. 5. Пошук шляху від певного місця до кореня.

Запити на відображення всього дерева і відображення шляху від певного місця дерева до кореня не є універсальними, так, як кожен раз, коли змінюється кількість рівнів дерева, цей запит потрібно буде налаштовувати. Якщо ж глибина дерева не відома, або не відомо на якому рівні знаходиться заданий елемент, то запит не може бути побудований стандартними засобами оператора SELECT, потрібно створювати рекурсивну процедуру, або писати рекурсивний запит [1](рис. 6).

Додавання нового вузла, відбувається з використанням простої команди SQL на додавання (INSERT). Для цього потрібно додати новий запис до таблиці CATEGORY, де як parent_id ми передаємо ідентифікатор батьківського елемента.

Наприклад, щоб додати елементи «M» і «N», ми виконуємо відповідні запити INSERT (рис. 7). Для переміщення вузла до іншого батьківського вузла слід виконати команду UPDATE (рис. 8). Додатковою перевагою є можливість переміщення всієї гілки, переміщаючи тільки один елемент. Це пов'язано з тим, що кожен елемент знає тільки свого батька, і тому при переміщенні будь-якого елемента дерево не втрачає узгодженості.

ISSN 2663 - 4023

а)

```
SQL> SELECT level, id, parent_id, title
  2  FROM CATEGORY
  3  START WITH parent_id is null
  4  CONNECT BY PRIOR id = parent_id;

     LEVEL         ID  PARENT_ID TITLE
---------- ---------- ---------- -----
         1          1            A
         2          2          1 B
         3          5          2 E
         2          3          1 C
         3          6          3 F
         3          7          3 G
         2          4          1 D
         3          8          4 H
         4         10          8 J
         4         11          8 K
         4         12          8 L
         3          9          4 I

12 rows selected.
```

б)

```
SQL> SELECT lpad(' ',3*level)||title as Tree
  2  FROM CATEGORY
  3  START WITH parent_id is null
  4  CONNECT BY PRIOR id = parent_id
  5  ORDER SIBLINGS BY title;

TREE
--------------------------------------------
   A
      B
         E
      C
         F
         G
      D
         H
            J
            K
            L
         I

12 rows selected.
```

в)

```
SQL> SELECT SYS_CONNECT_BY_PATH(title, '/') as Path
  2  FROM CATEGORY
  3  WHERE id=11
  4  START WITH parent_id is null
  5  CONNECT BY PRIOR id = parent_id;

PATH
--------------------------------------------------------
/A/D/H/K
```

Рис. 6. Рекурсивні запити на відображення всього дерева (а, б) та шляху від певного місця до кореня (в).

```
SQL> INSERT INTO CATEGORY (id, title, parent_id) VALUES (13, 'M', 6);

1 row created.

SQL> INSERT INTO CATEGORY (id, title, parent_id) VALUES (14, 'N', 11);

1 row created.

SQL> SELECT * FROM CATEGORY;

        ID TITLE     PARENT_ID
---------- -------- ----------
         1 A
         2 B                 1
         3 C                 1
         4 D                 1
         5 E                 2
         6 F                 3
         7 G                 3
         8 H                 4
         9 I                 4
        10 J                 8
        11 K                 8
        12 L                 8
        13 M                 6
        14 N                11

14 rows selected.
```

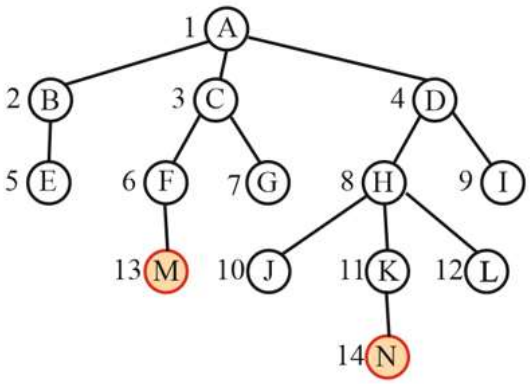


Рис. 7. Дерево з новими елементами

ISSN 2663 - 4023

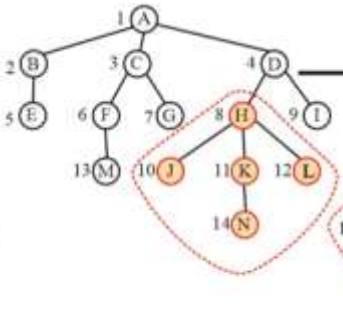
```
SQL> UPDATE CATEGORY SET parent_id=8 WHERE id=8;

1 row updated.

SQL> SELECT * FROM CATEGORY;

        ID TITLE  PARENT_ID
---------- ----- ----------
         1 A
         2 B              1
         3 C              1
         4 D              1
         5 E              2
         6 F              3
         7 G              3
         8 H              5
         9 I              4
        10 J              8
        11 K              8
        12 L              8
        13 M              6
        14 N             11

14 rows selected.
```



```
TREE
--------------------
A
  B
    E
      H
        J
        K
          N
        L
  C
    F
      M
    G
  D
    I

14 rows selected.
```

Рис. 8. Переміщенні елемента «H» разом з усіма нащадками до батьківського вузла «E».

При видаленні будь-якого вузла, враховуючи правило ON DELETE CASCADE буде видалено все піддерево (рис. 9).

```
SQL> DELETE FROM CATEGORY WHERE id=8;

1 row deleted.

SQL>  SELECT * FROM CATEGORY;

        ID TITLE     PARENT_ID
---------- -------- ----------
         1 A
         2 B                 1
         3 C                 1
         4 D                 1
         5 E                 2
         6 F                 3
         7 G                 3
         9 I                 4
        13 M                 6

9 rows selected.
```

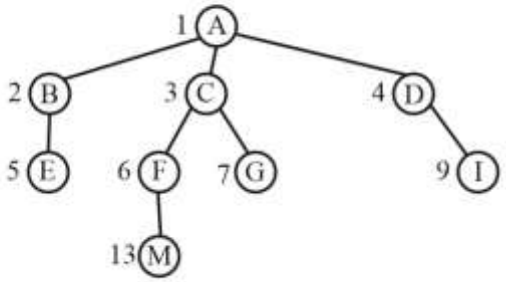


Рис. 8. Видалення вузла «H» дерева з усіма його нащадками

Маючи для кожного вузла лише ідентифікатор батьківського вузла, ми змушені використовувати рекурсію для обходу всього дерева. Для того, щоб уникнути рекурсії при виведенні всього дерева, моделювання ієрархічних структур даних виконують у вигляді таблиці зв'язків (Closure Table). При такому моделюванні список вузлів можна зберігати в одній таблиці, а зв'язки в іншій, (рис. 9).

Перша таблиці зберігає тільки назви вузлів в полі title та ідентифікатор вузла:

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

ISSN 2663 - 4023

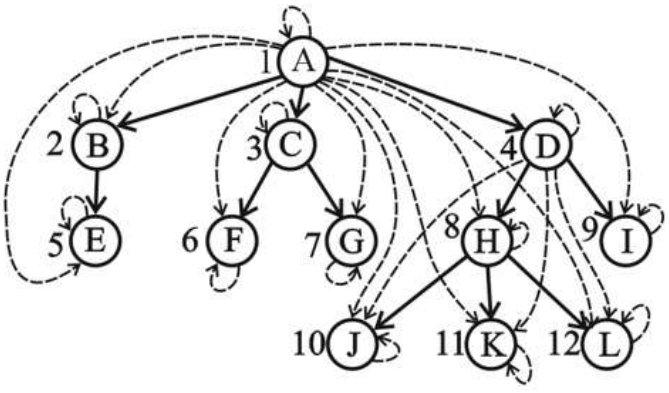
Друга таблиця зберігатиме зв’язки між кожним вузлом дерева і вузлами, що знаходяться на шляху від нього до вершини, включаючи саму вершину дерева. Зв’язок задається трьома числами: ідентифікатором вузла, від якого починається шлях, ідентифікатором кінцевого вузла на шляху та відстанню – цілим числом, що дорівнює кількості дуг між ними. Тобто, відстань між нащадком і батьком дорівнює 1, між нащадком і батьком батька дорівнює 2 і т.д. аж до самого кореня.

```
CREATE TABLE CATEGORY_link (
  Id_from INTEGER NOT NULL REFERENCES CATEGORY ON DELETE CASCADE,
  Id_to INTEGER NOT NULL REFERENCES CATEGORY ON DELETE CASCADE,
  distance INTEGER NOT NULL,
  PRIMARY KEY (Id_from, Id_to)
);
```

```
SQL>  SELECT * FROM CATEGORY;

        ID TITLE
---------- -----
         1 A
         2 B
         3 C
         4 D
         5 E
         6 F
         7 G
         8 H
         9 I
        10 J
        11 K
        12 L

12 rows selected.
```




```
SQL>  SELECT * FROM CATEGORY_link;

   ID_FROM      ID_TO   DISTANCE
---------- ---------- ----------
         1          1          0
         1          2          1
         1          3          1
         1          4          1
         1          5          2
         1          6          2
         1          7          2
         1          8          2
         1          9          2
         1         10          3
         1         11          3
         1         12          3
         2          2          0
         2          5          1
         3          3          0
         3          6          1
         3          7          1
         4          4          0
         4          8          1
         4          9          1
         4         10          2
         4         11          2
         4         12          2
         5          5          0
         6          6          0
         7          7          0
         8          8          0
         8         10          1
         8         11          1
         8         12          1
         9          9          0
        10         10          0
        11         11          0
        12         12          0

34 rows selected.
```

Рис. 9. Подання дерева у вигляді таблиці зв’язків

При такій організації даних, виведення всіх вузлів піддерева та пошук шляху від певного місця до кореня не потребує рекурсії (рис.10, 11), але ускладнюється процес додавання нового вузла, переміщення вузла до іншого батьківського вузла. Для додавання даних потрібно буде добавляти дані в дві таблиці (рис. 12). Замість запиту на заповнення другої таблиці (рис.12), можна створити тригер, який буде створювати зв’язки між доданим вузлом і батьківським вузлом та з усіма вузлами, що знаходяться вище батьківського. При побудові такого тригера, в тіло тригера потрібно передавати параметр, вузол батьківського елемента. Важливо пам’ятати, що тригер спрацьовує при настанні DML події на об’єкті бази даних, на якому визначено тригер. Цю подію не

ISSN 2663 - 4023

можна викликати явно, єдиний спосіб – виконати запит DML. Передача параметрів не є частиною визначення тригера. Одним з найпростіших рішень, це передати параметри у змінній сесії, але змінна сесії повинна бути встановлена до настання події DML. Для цього потрібно створити пакет в якій описати тільки одну змінну, яку ми будемо передавати в тригер. Тоді процес додавання вузла в дерево буде відбуватися наступним чином: спочатку за допомогою пакета, задаємо значення батьківського вузла до якого ми будемо додавати новий елемент; додаємо вузол в таблицю CATEGORY, на яку спрацьовує тригер, що заповнює таблицю CATEGORY_link (рис. 13). При написанні тригера, потрібно враховувати те, що тригер може призвести до виконання операторів SQL, які запускають ще один або декілька тригерів.

```
SQL> SELECT title FROM CATEGORY t1 LEFT JOIN
  2  CATEGORY_link t2 ON t1.id=t2.id_to WHERE t2.id_from=4;

TITLE
-----
D
H
I
J
K
L

6 rows selected.
```

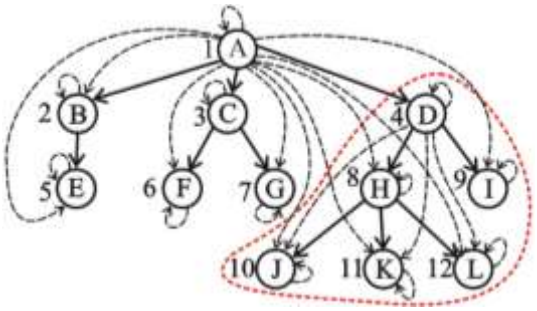


Рис. 10. Виведення всіх вузлів піддерева з вершиною «D»

```
SQL> SELECT title FROM CATEGORY t1
  2  INNER JOIN CATEGORY_link t2 ON t1.id=t2.id_from
  3  WHERE t2.id_to=11;

TITLE
-----
A
D
H
K
```

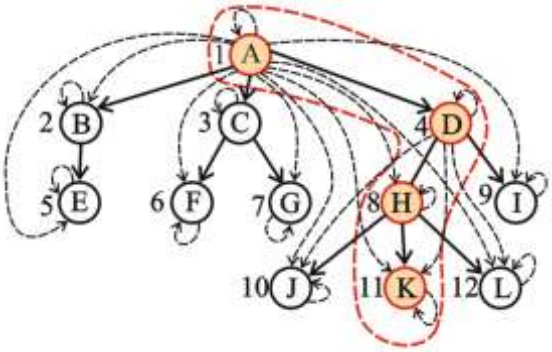


Рис. 11. Пошук шляху від 11 елемента до кореня.

```
SQL> INSERT INTO CATEGORY VALUES ('13','M');

1 row created.

SQL>
SQL> INSERT INTO CATEGORY_link (Id_from, Id_to, distance)
  2  SELECT Id_from, 13, distance+1 FROM CATEGORY_link
  3  WHERE Id_to=5
  4  UNION
  5  SELECT 13,13, 0 FROM CATEGORY_link;

4 rows created.
```

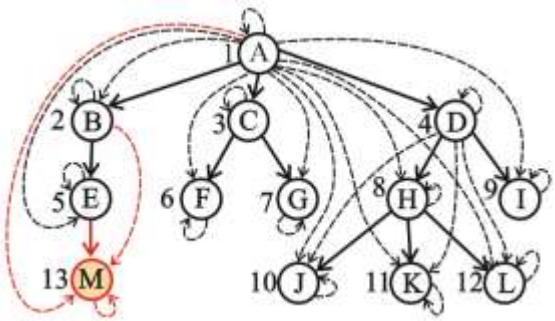


Рис. 12. Запит на давання вузла «M» в дерево з вершиною «E»

Операція видалення вузла виконується за допомогою операції DELETE, при цьому всі його нащадки стають нащадками видаленого батьківського вузла. Наприклад видалення вузла з ідентифікатором рівним 3 виконується за допомогою запиту:

ISSN 2663 - 4023

```
DELETE FROM CATEGORY WHERE id=3;
```

```
SQL> CREATE OR REPLACE PACKAGE Id_from_TREE AS
  2   Id_from_t INTEGER;
  3  END Id_from_TREE;
  4  /

Package created.

SQL>
SQL> CREATE OR REPLACE TRIGGER INSERT_TREE
  2   AFTER INSERT ON CATEGORY
  3   FOR EACH ROW
  4  DECLARE
  5
  6  BEGIN
  7
  8  INSERT INTO CATEGORY_link (Id_from, Id_to, distance)
  9  SELECT Id_from, :NEW.id, distance+1 FROM CATEGORY_link
 10  WHERE Id_to=Id_from_TREE.Id_from_t
 11  UNION
 12  SELECT :NEW.id, :NEW.id, 0 FROM CATEGORY_link;
 13
 14  END INSERT_TREE;
 15  /

Trigger created.

SQL> exec Id_from_TREE.Id_from_t:=5;

PL/SQL procedure successfully completed.

SQL> INSERT INTO CATEGORY VALUES (13, 'M');

1 row created.
```

Рис. 13. Додавання вузла в дерево з вершиною «D» , використовуючи тригер

При цьому, його нащадки, вузли з ідентифікаторами 6 і 7, стануть нащадками вузла 1, але ще потрібно змінити відстань між новим батьківським вузлом і нащадками. Тому пропонується перед тим, як видаляти вузол виконати оновлення відстаней (поле dictance) для нащадків. Для цього потрібно створити відповідний тригер (для цього потрібно враховувати те, що при створенні таблиць на обмеження зовнішнього ключа, додано правило ON DELETE CASCADE), або написати запит на оновлення.

Наприклад, для вузла 3:

```
UPDATE CATEGORY_link SET distance=distance-1
WHERE id_to IN (SELECT id_to FROM CATEGORY_link WHERE id_from=3)AND
distance>0;
```

Для того, щоб видалити все піддерево вузла з ідентифікатором рівним 3 з усіма його нащадками потрібно виконати наступний запит:

```
delete from category WHERE id IN (SELECT id_to FROM CATEGORY_link WHERE
id_from=3) ;
```

Враховуючи використання правила ON DELETE CASCADE в таблиці CATEGORY_link будуть видалені всі зв’язки вузлів піддерева.

Недоліком такої організації є велика кількість записів у таблиці CATEGORY_link через необхідність зберігати зв’язки кожного елемента дерева з усіма його предками.

Чим глибше в дереві знаходиться вузол, тим більше буде потрібно записів для опису зв'язків (рис. 9).

3. ВИСНОВКИ ТА ПЕРСПЕКТИВИ ПОДАЛЬШИХ ДОСЛІДЖЕНЬ

В даній роботі розглянута побудова ієрархічних структур в реляційних базах даних, що використовує модель суміжних списків і таблиці зв'язків. Подано приклади написання запитів для розв'язання типових завдань, які зустрічаються під час роботи з деревовидними структурами: пошук всіх дочірніх вузлів, всіх нащадків та предків заданого вузла, переміщення вузла до іншого батьківського вузла, видалення вузлів з усіма його нащадками. Дослідження різних варіантів організації деревоподібних структур SQL дозволить зрозуміти та обрати самий оптимальний спосіб побудови такої структури в реляційній базі даних для конкретної задачі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1 Wirth, N. (1986). *Algorithms and data structures.* Prentice-Hall.
2 Narasimha Karumanchi. (2017). *Data Structures And Algorithms Made Easy.* CareerMonk.
3 *Joe Celko's Trees and Hierarchies in SQL for Smarties*. (2004). Elsevier. https://doi.org/10.1016/b978-1-55860-920-4.x5000-4
4 Буй, Д., & Поляков, С. (2010). Рекурсивні запити в SQL-подібних мовах: приклади, змістова і формальна семантика. *Проблеми програмування*, (2-3).
5 Буй, Д., & Поляков, С. (2010). Композиційна семантика рекурсивних виразів та їхніх узагальнень в SQL-подібних мовах. *Наукові записки НаУКМА. Комп'ютерні науки.*, *112*, 21–26.
6 Резниченко, В. (2010). Рекурсивный SQL. *Інженерія програмного забезпечення*, (4), 63–81.
7 Sarmiento, E. (2008, June 16). *Recursive Queries using Common Table Expressions (CTE) in SQL Server*. SQL Server Tips, Techniques and Articles. https://www.mssqltips.com/sqlservertip/1520/recursive-queries-using-common-table-expressions-cte-in-sql-server/
8 *Trees in SQL. Joe Celko*. (n.d.). Firebird, InterBase, HQbird: replication, monitoring, technical support. http://www.ibase.ru/files/articles/programming/dbmstrees/sqltrees.html.
9 Шрамченко, Б. Л., & Ахматов, В. В. (2021). Обробка ієрархічних даних у реляційній базі даних. In *нформаційні технології в науці, виробництві та підприємництві* (pp. 152–155). Освіта України.
10 Голуб, В. (2010). Ієрархічна модель вкладених множин у реляційних базах даних. *ВІСНИК ЛЬВІВ. УН-ТУ. Серія прикл. матем. інформ.*, (16), 106–113.
11 Hazel, D. (2008). Using rational numbers to key nested sets. Article DocSetID-311997. https://doi.org/10.48550/arXiv.0806.3115

КІБЕРБЕЗПЕКА: освіта, наука, техніка
CYBERSECURITY: EDUCATION, SCIENCE, TECHNIQUE
№ 4 (16), 2022
ISSN 2663 - 4023

Volodymyr O. Markitan
Student of the Faculty of Informations Technologies and Mathematics
Lesya Ukrainka Volyn National University, Lutsk, Ukraine
ORCID ID: 0000-0002-3589-8784
Markitan.Volodymyr2020@vnu.edu.ua

Mykola A. Vozniak
Student of the Faculty of Informations Technologies and Mathematics
Lesya Ukrainka Volyn National University, Lutsk, Ukraine
ORCID ID: 0000-0002-3589-8784
Vozniak.Mykola2020@vnu.edu.ua

Lesia V. Bulatetska
PhD, Associate Professor, Associate Professor at the Department of Computer Science and Cybersecurity
Lesya Ukrainka Volyn National University, Lutsk, Ukraine
ORCID ID 0000-0002-7202-826X
Bulatetska.Lesya@vnu.edu.ua

Vitalii V. Bulatetskyi
PhD, Associate Professor, Associate Professor at the Department of Computer Science and Cybersecurity
Lesya Ukrainka Volyn National University, Lutsk, Ukraine
ORCID ID 0000-0002-9883-4550
Bulatetsky.Vitaly@vnu.edu.ua

PRESERVATION OF HIERARCHY STRUCTURES IN RELATIVE DATABASES

Annotation. Relational database management systems and the SQL language itself do not have any built-in mechanisms for storing and managing hierarchical structures. There are several different ways to represent trees in relational databases. This paper considers the method of modeling hierarchical data structures in the form of Adjacency Lists and Closure Tables. For each method, there are examples of writing queries to solve typical problems encountered when working with tree structures: finding all descendant leaves, all descendants and ancestors of a given leaf, moving a leaf to another ancestor leaf, and deleting leaves with all its descendants. The possibility of using recursive queries when displaying the entire tree in the Adjacency List model is considered. If the depth of the tree is not known, or it is not known at what level the specified element is, the query can not be built by standard means of the SELECT statement, then you need to create a recursive procedure, or write a recursive query. In order to avoid recursion when outputting the whole tree, all nodes of the subtree, and finding the path from a certain place to the root, the modeling of hierarchical data structures is performed in the form of a connection table (Closure Table). This complicates the process of adding a new leaf and moving the leaf to another ancestor leaf. In this case, to simplify the writing of queries, it is suggested to create triggers that will build or rebuild the links. Given the fact that sometimes there is a need to preserve dependent, in particular hierarchical structures in a relational database, you need to be able to plow the model of preservation of such data. The choice of method for solving a specific problem is influenced by the speed of basic operations with trees. Exploring different options for organizing SQL tree structures will allow you to understand and choose the best way to build such a structure in a relational database for a specific task. All SQL queries in this paper were created and tested for Oracle relational databases.

Key words: database, hierarchical data structure, tree, adjacency list model, closure table model.

REFERENCES (TRANSLATED AND TRANSLITERATED)

1 Wirth, N. (1986). *Algorithms and data structures.* Prentice-Hall.
2 Narasimha Karumanchi. (2017). *Data Structures And Algorithms Made Easy.* CareerMonk.
3 *Joe Celko's Trees and Hierarchies in SQL for Smarties*. (2004). Elsevier. https://doi.org/10.1016/b978-1-55860-920-4.x5000-4

4 Bui D., & Poliakov S. (2010). Rekursyvni zapyty v SQL-podibnykh movakh: pryklady, zmistova i formalna semantyka. *Problemy prohramuvannia*. 2010. (2-3).
5 Bui D., & Poliakov S. (2010). Kompozytsiina semantyka rekursyvnykh vyraziv ta yikhnikh uzahalnen v SQL-podibnykh movakh. *Naukovi zapysky NaUKMA. Kompiuterni nauky. 112*. 21–26.
6 Reznychenko V. (2010) Rekursyvnyi SQL. *Inzheneriia prohramnoho zabezpechennia,* (4), 63–81.
7 Sarmiento, E. (2008, June 16). *Recursive Queries using Common Table Expressions (CTE) in SQL Server*. SQL Server Tips, Techniques and Articles. https://www.mssqltips.com/sqlservertip/1520/recursive-queries-using-common-table-expressions-cte-in-sql-server/
8 *Trees in SQL. Joe Celko*. (n.d.). Firebird, InterBase, HQbird: replication, monitoring, technical support. http://www.ibase.ru/files/articles/programming/dbmstrees/sqltrees.html.
9 Shramchenko B. L., & Akhmatov V. V. (2021). Obrobka iierarkhichnykh danykh u reliatsiinii bazi danykh. Informatsiini tekhnolohii v nautsi, vyrobnytstvi ta pidpryiemnytstvi (pp. 152–155). Osvita Ukrainy.
10 Holub V. (2010). Iierarkhichna model vkladenykh mnozhyn u reliatsiinykh bazakh danykh. VISNYK LVIV. UN-TU. *Seriia prykl. matem. inform.* (16), 106–113.
11 Hazel, D. (2008). Using rational numbers to key nested sets. Article DocSetID-311997. https://doi.org/10.48550/arXiv.0806.3115