ПРИКЛАДНІ ПРОБЛЕМИ КОМП'ЮТЕРНИХ НАУК,
БЕЗПЕКИ ТА МАТЕМАТИКИ, 2026, № 6

УДК 004.9

МЕТОДИ ЗБЕРЕЖЕННЯ ТА ОБРОБКИ ДЕРЕВОВИДНИХ СТРУКТУР У РІЗНИХ ТИПАХ БАЗ ДАНИХ: РЕЛЯЦІЙНИХ, ДОКУМЕНТООРІЄНТОВАНИХ ТА ГРАФОВИХ

Зубок Д.В.
Булатецька Л.В. (ORCID: 0000-0002-7202-826X),
Волинський національний університет імені Лесі Українки, Луцьк, Україна

METHODS FOR STORING AND PROCESSING TREE-LIKE STRUCTURES IN VARIOUS DATABASE TYPES: RELATIONAL, DOCUMENT-ORIENTED, AND GRAPH DATABASES

Zubok D.V., Bulatetska L.V.
Lesya Ukrainka Volyn National University, Lutsk, Ukraine

Abstract. This paper presents the results of an experimental study on the efficiency of storing and processing tree-like structures in various database models. The main objective is to compare the performance of executing typical queries on hierarchical data across Relational (MySQL), Graph (Neo4j), and Document-Oriented (MongoDB) database management systems (DBMS). The relevance of the study is driven by the widespread use of hierarchical structures in modern information systems, where the choice of the data model significantly impacts the speed and efficiency of query execution. Within the experiment, a three-level tree structure was modeled. We measured the execution time for three main types of queries. Retrieving the immediate children of a given node. Building the full path to the root element. Level-by-level traversal of the tree (corresponding to the Breadth-First Search algorithm). The results showed that with a small data volume, the Relational DBMS (MySQL) demonstrated the highest performance. The Graph DBMS (Neo4j) yielded a slightly worse result, while the Document-Oriented DBMS (MongoDB) proved to be the least efficient. We conclude that the relational model, despite its traditional nature, remains competitive in tasks involving the processing of hierarchical structures, particularly with small amounts of data and a well-chosen storage schema.

Keywords: Data Base, Adjacency List, hierarchical structures, Relational DBMS, Document-Oriented DBMS, Graph DBMS.

Вступ. Деревовидні структури є одним із найпоширеніших способів представлення даних у сучасних інформаційних системах. Ієрархічні моделі лежать в основі значної кількості практичних задач – від організації каталогів товарів, структури підприємств, навчальних матеріалів, мережевих топологій до графів зв'язків у соціальних мережах чи складних об'єктних взаємодій. Ефективність роботи систем зберігання даних у таких сценаріях значною мірою залежить від того, наскільки обраний спосіб моделювання та реалізації деревовидних структур відповідає природі задачі, обсягу даних та вимогам до продуктивності.

Різні типи баз даних пропонують свої підходи до моделі представлення ієрархій. У реляційних СУБД деревовидні структури зазвичай реалізують на основі додаткових схем моделювання – таких як «список суміжності» (Adjacency List), «вкладені множини» (Nested Sets) чи «таблиця шляхів» (Closure Table). Документоорієнтовані системи використовують вкладені JSON/XML-структури, а графові бази даних зберігають зв'язки природним способом за допомогою вузлів і ребер. Кожен із підходів має свої переваги й обмеження, які впливають на складність реалізації, швидкість читання і оновлення даних, можливості масштабування та зручність формування запитів.

У роботі розглянуто основні методи збереження та обробки деревовидних структур у реляційних, документоорієнтованих і графових базах даних. Порівняння здійснено з точки зору принципів моделювання, доступних засобів запитів, продуктивності операцій та прикладних особливостей використання.

Водночас варто відзначити, що дослідження виконано на прикладах із невеликими обсягами даних, що є його важливим обмеженням. Отримані результати демонструють поведінку різних типів сховищ у контексті невеликих або середніх наборів інформації, проте подальші дослідження доцільно розширити на великі і високонавантажені системи.

Попри появу сучасних нереляційних моделей даних, питання вибору оптимального інструменту для роботи з деревоподібними структурами залишається актуальним. Проведене порівняння має як теоретичне, так і практичне значення для розробників інформаційних систем, архітекторів баз даних та дослідників, що працюють із ієрархічними наборами даних..

Класичні моделі збереження ієрархій у реляційних БД. Ієрархічні структури в реляційних базах даних можуть бути реалізовані за допомогою кількох поширених методів, включаючи Adjacency List, Nested Sets, Closure Table та Materialized Path [1–7]. З метою забезпечення ефективної роботи з такими даними, ключовим завданням є вибір формату зберігання, який мінімізує час вибірки та витрати системних ресурсів. В табл.1. подано порівняльний аналіз методів подання ієрархічних структур в реляційних базах даних.

Таблиця 1

Порівняння методів подання ієрархічних структур у реляційних базах даних

Підхід	**Принцип збереження**	**Переваги**	**Недоліки**	**Примітка**
Adjacency List (SQL)	Кожен рядок містить parent_id, тобто посилання на батьківський вузол.	Дуже просто реалізувати, легко додавати або видаляти вузли.	Щоб знайти всі нащадки або шлях до кореня, потрібна рекурсія або кілька запитів.	Підходить для невеликих ієрархій та динамічних структур.
Nested Sets (SQL)	Кожен вузол має ліву і праву межу (left і right), які визначають положення в дереві.	Піддерева можна вибрати одним запитом без рекурсії.	Вставка, видалення або переміщення вузлів потребує перерахунку меж для багатьох рядків.	Оптимально для статичних дерев, де рідко змінюється структура.
Materialize Path (SQL)	У вузлі зберігається повний шлях від кореня до нього як рядок (/root/level1/level2).	Швидкий пошук піддерев через LIKE, легко отримати шлях до кореня.	Зміни структури дерева потребують оновлення рядків, що містять шлях.	Добре підходить для середніх ієрархій, де потрібен частий доступ до піддерев.
Closure Table (SQL)	Створюється окрема таблиця з усіма парами «батько-нащадок», включно з довжиною шляху.	Дуже гнучкий, дозволяє швидко отримати всі піддерева або шляхи будь-якої глибини.	Потребує додаткової таблиці та більше операцій при вставці/видаленні вузлів.	Підходить для великих і складних ієрархій, де часто виконуються запити на всі рівні.

Підходи збереження ієрархічних структур у документоорієнтованих БД. У документоорієнтованих базах даних, таких як MongoDB, збереження ієрархічних структур реалізується двома основними способами: вкладені документи та посилання на батьківські або дочірні елементи [8].

У випадку вкладених документів дочірні елементи або піддерева зберігаються всередині документа батьківського вузла. Наприклад, категорія товарів може містити масив підкатегорій, а кожна підкатегорія – масив продуктів. Такий підхід забезпечує швидкий доступ до всієї ієрархії за один запит, оскільки вся потрібна інформація знаходиться в одному документі. Він добре підходить для невеликих і середніх ієрархій,

де структура рідко змінюється, оскільки зміна глибоких рівнів потребує оновлення всього документа.

Другий підхід посилання на батьківські або дочірні елементи. Тут кожен документ зберігається окремо, а зв'язки між вузлами визначаються через унікальні ідентифікатори, наприклад, parent_id або масив children_ids. Цей метод більш гнучкий для великих і динамічних ієрархій, оскільки дозволяє змінювати структуру без дублювання даних і зручно реалізовувати пошук або агрегації за різними рівнями. Проте для отримання повного піддерева потрібне виконання кількох запитів або використання агрегованих операцій.

Ієрархічні структури у графових БД. Графові бази даних, такі як Neo4j [9] або OrientDB [10], мають природну підтримку деревових і графових структур, оскільки їхня модель даних побудована на вузлах (nodes) та зв'язках (relationships), що відображає реальні ієрархії та мережеві структури. У таких СУБД вузли можуть представляти будь-які об'єкти, а зв'язки між ними – їхні ієрархічні або асоціативні відношення, причому кожен зв'язок зберігає напрямок і властивості.

Ця природна модель дозволяє виконувати операції, які в реляційних або документоорієнтованих БД є складними та ресурсоємними, напряму і ефективно. Наприклад, у Neo4j можна одним запитом знайти всі нащадки вузла, обчислити шлях до кореня, знайти спільних предків або виконати пошук за довільною глибиною ієрархії. Графові СУБД оптимізовані для таких запитів завдяки індексуванню зв'язків та використанню алгоритмів обходу графів.

Таким чином, графові БД ідеально підходять для роботи з великими або складними ієрархіями, соціальними мережами, рекомендаційними системами та іншими завданнями, де важлива швидкість проходження зв'язків і гнучкість структури даних.

Метою роботи є порівняльний аналіз та дослідження ефективності методів збереження ієрархічних структур у різних моделях баз даних, а також розроблення експериментального прототипу для оцінки їхньої продуктивності та визначення переваг для практичного використання.

Методологія дослідження. Для проведення аналізу продуктивності виконання запитів до деревоподібних структур у різних моделях баз даних було змодельовано деревоподібну структуру каталогу товарів (рис.1).

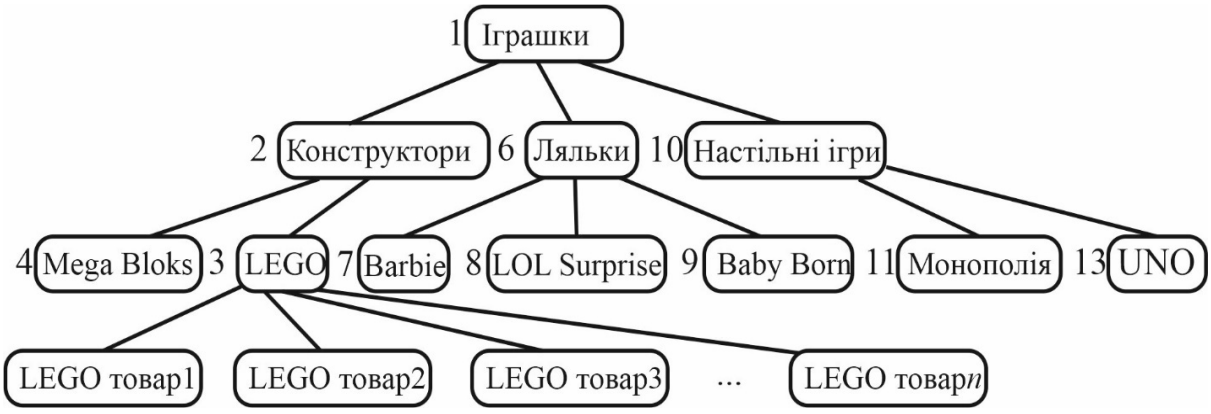


Рис. 1. Ієрархічне дерево каталогу товарів

Для порівняння методів збереження деревоподібних структур у різних моделях баз даних було використано такі СУБД:

- реляційна MySQL;
- документоорієнтована MongoDB;

- графова Neo4j.

Для порівняльного аналізу продуктивності була розроблена та реалізована ідентична ієрархічна структура даних (каталог товарів), що містить 52 категорії та 720 товарів.

В реляційній базі даних структура була змодельована за допомогою двох таблиць. Таблиця Категорії (categories) використовує модель Списку суміжних вершин (Adjacency List), де кожна категорія посилається на свого безпосереднього батька (parent_id), що дозволило побудувати трирівневе дерево (рис. 2, а). Таблиця Товари (products) пов'язує кожен із 720 товарів з його кінцевою категорією (рис. 2, б). Ключовою особливістю цієї таблиці є використання денормалізації шляху через включення колонок level_1, level_2, level_3. Ця гібридна структура значно прискорює операції читання для відображення повного шляху до товару, уникаючи складних рекурсивних запитів чи багаторазових операцій JOIN, хоча й вимагає додаткових зусиль при оновленні ієрархії.

а)

id	name	parent_id
1	Іграшки	*NULL*
2	Конструктори	1
3	LEGO	2
4	Mega Bloks	2
5	Playmobil	2
6	Ляльки	1
7	Barbie	6
8	LOL Surprise	6
9	Baby Born	6
10	Настільні ігри	1
11	Монополія	10
12	Дженга	10
13	UNO	10

б)

id	product_name	level_1	level_2	level_3	price	category_id
1	LEGO товар 1	Іграшки	Конструктори	LEGO	2673.55	3
2	LEGO товар 2	Іграшки	Конструктори	LEGO	2061.87	3
3	LEGO товар 3	Іграшки	Конструктори	LEGO	4559.77	3
4	LEGO товар 4	Іграшки	Конструктори	LEGO	1979.19	3
5	LEGO товар 5	Іграшки	Конструктори	LEGO	2080.01	3
6	LEGO товар 6	Іграшки	Конструктори	LEGO	4290.75	3
7	LEGO товар 7	Іграшки	Конструктори	LEGO	2864.99	3
8	LEGO товар 8	Іграшки	Конструктори	LEGO	4311.31	3
9	LEGO товар 9	Іграшки	Конструктори	LEGO	4928.37	3
10	LEGO товар 10	Іграшки	Конструктори	LEGO	2072.40	3
11	LEGO товар 11	Іграшки	Конструктори	LEGO	3904.92	3
12	LEGO товар 12	Іграшки	Конструктори	LEGO	1263.12	3
13	LEGO товар 13	Іграшки	Конструктори	LEGO	1810.40	3

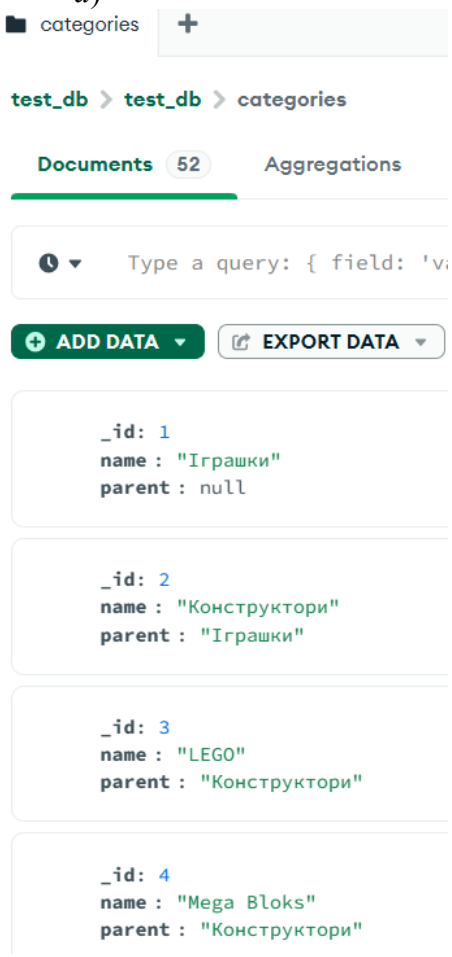
Рис. 2. Структура ієрархії каталогу товарів у реляційній базі даних: таблиця категорій (categories) (модель Adjacency List) (а)та таблиця товарів (products) (*б*).

Для порівняльного аналізу та ідентичності даних, ця ж ієрархічна структура була реалізована в MongoDB за гібридним підходом із використанням двох колекцій, заповнених тими самими 52 категоріями та 720 товарами. Колекція categories використовує адаптовану модель Списку суміжних вершин (Adjacency List) (рис. 3, а), де поле parent містить назву батьківської категорії, а не її ID. Колекція products використовує підхід вбудовування для оптимізації читання (рис. 3, б). Кожен документ товару містить повний ієрархічний шлях (поля level_1, level_2, level_3) та пряме посилання на ID категорії (category_id). Це дозволяє уникнути операцій lookup (JOIN) при відображенні детальної інформації про товар.

У Neo4j структура була реалізована як граф, де кожна категорія та товар представлені як вузли (Nodes), а їхні ієрархічні зв'язки («належить до», «є батьківським») – як ребра (Relationships). Це є найбільш природним відображенням ієрархії. База Neo4j також була заповнена точно таким же набором даних для забезпечення коректності порівняння продуктивності запитів.

Таким чином, три ідентичні набори даних (реляційна mySQL, MongoDB та Neo4j) були підготовлені для проведення оцінки ефективності виконання типових запитів до деревоподібних структур у різних архітектурах баз даних.

а)



б)

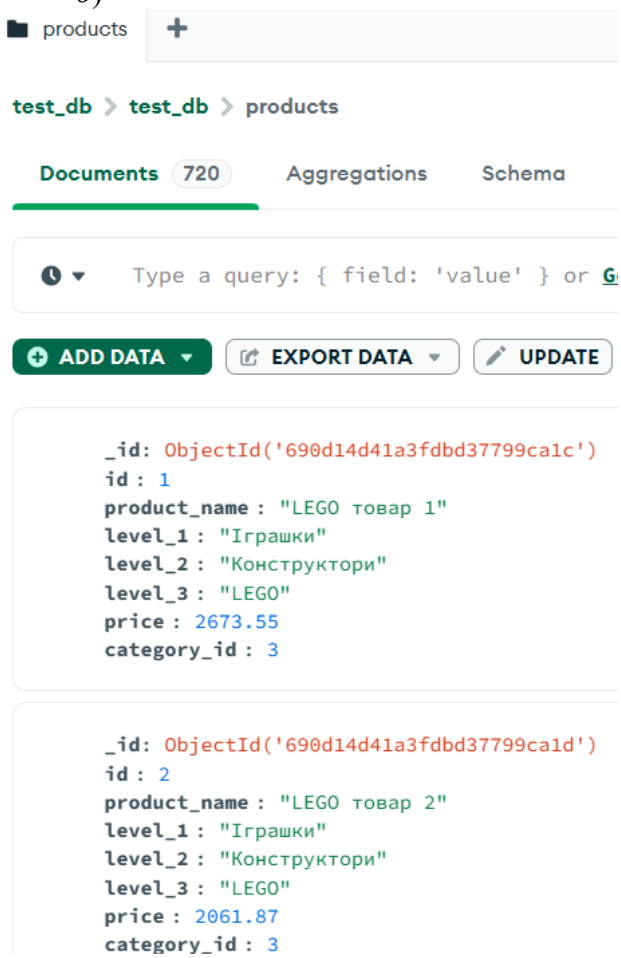


Рис. 3. Фізичне відображення даних ієрархічної структури каталогу товарів у документоорієнтованій базі даних.

Для емпіричної перевірки теоретичних переваг та недоліків різних підходів до моделювання деревовидних структур, на мові програмування Python було розроблено експериментальний прототип. Основною метою цього прототипу є демонстрація ефективності та об'єктивне порівняння швидкості виконання типових операцій (CRUD) для ієрархічних даних у трьох принципово різних типах систем керування базами даних (СУБД).

Тестування проводила на ноутбуці ASUS TUF Gaming F15 з процесором Intel Core i5-10300H (2.5GHz) і операційною системою MS Windows 11 Home (24H2).

Результати дослідження та їхнє обговорення. У цій роботі ми обмежуємо аналіз предметної області, зосереджуючись виключно на операціях вибірки даних із деревоподібних структур. Розроблений експериментальний прототип та подальше порівняння швидкості СУБД будуть зосереджені на трьох важливих для ієрархічних даних запитах:

- визначення та виведення безпосередніх нащадків (дочірніх вузлів) для заданого батьківського елемента;

- виведення повного ланцюга предків від заданого вузла до кореневого вузла дерева;
- виведення вузлів дерева послідовно, рівень за рівнем (спочатку всі елементи на глибині 1, потім на глибині 2, і так далі), що дозволяє порівняти ефективність реалізації алгоритму пошуку в ширину у різних СУБД.

Таке обмеження дозволить нам максимально детально та об'єктивно порівняти продуктивність різних моделей зберігання (реляційної, документоорієнтованої та графової) саме в контексті читання та навігації по ієрархії.

На рис. 4, рис. 5 та рис. 6 подано фрагменти коду Python, що демонструють реалізацію запиту вибірки безпосередніх дочірніх вузлів (підкатегорій) ієрархії категорій для MySQL, MongoDB та Neo4j відповідно.

```python
def get_children_mysql(level):
    start = time.time()
    mysql_cursor.execute("""
        SELECT DISTINCT level_2 FROM products WHERE level_1=%s
        UNION
        SELECT DISTINCT level_3 FROM products WHERE level_2=%s
    """, (level, level))
    records = [row[0] for row in mysql_cursor.fetchall()]
    end = time.time()
    print_result(f"MySQL - Дочірні вузли '{level}'", records, end-start)
```

Рис. 4. Фрагмент коду Python, що демонструє запит вибірки дочірніх вузлів у MySQL.

```python
def get_children_mongo(level):
    start = time.time()
    level2_children = mongo_products.distinct("level_2", {"level_1": level})
    level3_children = mongo_products.distinct("level_3", {"level_2": level})
    records = level2_children + level3_children
    end = time.time()
    print_result(f"MongoDB - Дочірні вузли '{level}'", records, end-start)
```

Рис. 5. Фрагмент коду Python, що демонструє запит вибірки дочірніх вузлів у MongoDB.

```python
def get_children_neo4j(level):
    with neo4j_driver.session() as session:
        start = time.time()
        result = session.run(
            """
            MATCH (c:Category {name:$name})-[:HAS_SUBCATEGORY*]->(child)
            RETURN DISTINCT child.name AS child_name
            """,
            name=level
        )
        records = [r["child_name"] for r in result]
        end = time.time()
        print_result(f"Neo4j - Дочірні вузли '{level}'", records, end-start)
```

Рис. 6. Фрагмент коду Python, що демонструє запит вибірки дочірніх вузлів у Neo4j.

Результати вимірювань показали, що в отриманні дочірніх вузлів ієрархії категорій найшвидше спрацював MySQL (рис. 7).

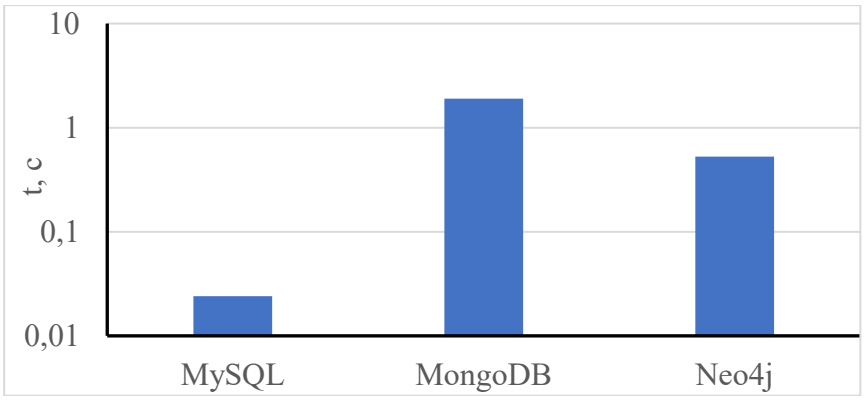


Рис. 7. Час виконання операції виведення дочірніх для категорії першого рівня (3 підкатегорії) у різних системах управління базами даних."

На рис.8, рис. 9, та рис. 10 подано фрагменти коду Python, що демонструють реалізацію запиту вибірки повного шляху від кінцевого вузла (товару або найнижчого рівня категорії) до кореня ієрархії каталогу у MySQL, MongoDB та Neo4j відповідно.

```python
def path_to_root_mysql(level):
    start = time.time()
    mysql_cursor.execute("""
        WITH RECURSIVE path AS (
            SELECT id, product_name, level_1, level_2, level_3, level_1 AS root
            FROM products
            WHERE level_3=%s
            UNION ALL
            SELECT p.id, p.product_name, p.level_1, p.level_2, p.level_3, path.root
            FROM products p
            INNER JOIN path ON p.level_2 = path.level_3
        )
        SELECT DISTINCT root, level_1, level_2, level_3 FROM path
    """, (level,))
    records = mysql_cursor.fetchall()
    end = time.time()
    print_result(f"MySQL - Шлях до кореня '{level}'", records, end-start)
```

Рис. 8. Фрагмент коду Python, що демонструє реалізацію запиту вибірки повного шляху від кінцевого вузла (товару) до кореня ієрархії каталогу у MySQL.

```python
def path_to_root_mongo(level):
    start = time.time()
    # MongoDB: збираємо шлях через рівні
    docs = list(mongo_products.find({"level_3": level}, {"_id":0,"level_1":1,"level_2":1,"level_3":1}))
    records = [[d['level_1'], d['level_2'], d['level_3']] for d in docs]
    end = time.time()
    print_result(f"MongoDB - Шлях до кореня '{level}'", records, end-start)
```

Рис. 9. Фрагмент коду Python, що демонструє реалізацію запиту вибірки повного шляху від кінцевого вузла (товару) до кореня ієрархії каталогу у MongoDB.

```python
def path_to_root_neo4j(level):
    with neo4j_driver.session() as session:
        start = time.time()
        result = session.run(
            """
            MATCH path = (root:Category)-[:HAS_SUBCATEGORY*]->(c:Category {name:$name})
            RETURN [n IN nodes(path) | n.name] AS path_names
            """,
            name=level
        )
        records = [r["path_names"] for r in result]
        end = time.time()
        print_result(f"Neo4j - Шлях до кореня '{level}'", records, end-start)
```

Рис. 10. Фрагмент коду Python, що демонструє реалізацію запиту вибірки повного шляху від кінцевого вузла (товару) до кореня ієрархії каталогу у Neo4j.

Результати експериментального дослідження швидкості виконання операції виведення шляху до кореня засвідчили, що реляційна система керування базами даних MySQL забезпечує найвищу обчислювальну продуктивність серед протестованих рішень. Цей факт є показовим, оскільки графова СУБД Neo4j, архітектурно орієнтована на оптимізацію роботи з відношеннями та обходом графових структур, продемонструвала дещо нижчу, але все ж високу швидкодію. (рис. 11)

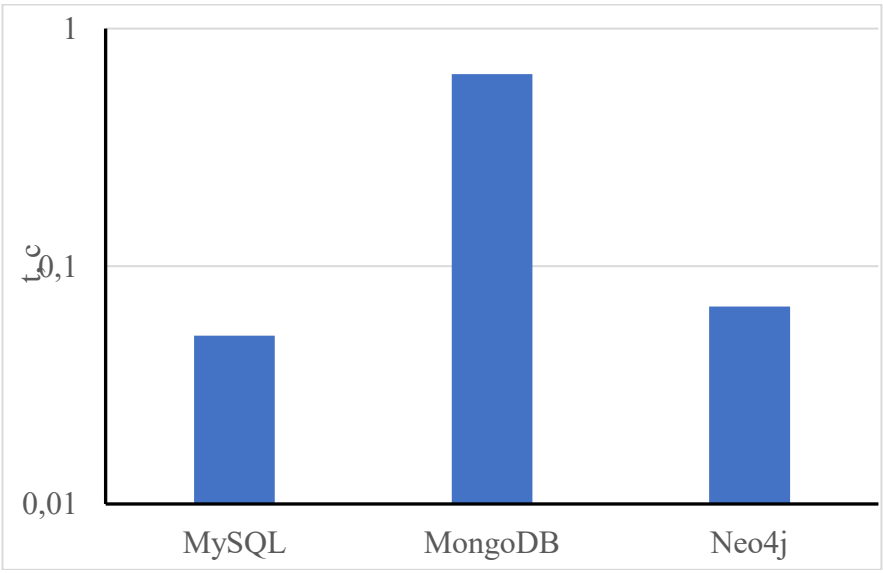


Рис. 11. Час виконання операції виведення шляху від листка дерева до кореня дерева

На рис. 12., рис. 13, та рис. 14 подано фрагменти коду Python, що демонструють реалізацію запиту для швидкого знаходження всіх товарів, які містять задане значення level «Настільні ігри» на будь-якій глибині ієрархії у MySQL, MongoDB та Neo4j відповідно.

```python
def search_by_level_mysql(level):
    start = time.time()
    mysql_cursor.execute("""
        SELECT DISTINCT level_1, level_2, level_3 FROM products
        WHERE level_1=%s OR level_2=%s OR level_3=%s
    """, (level, level, level))
    records = mysql_cursor.fetchall()
    end = time.time()
    print_result(f"MySQL - Пошук за рівнем '{level}'", records, end-start)
```

Рис. 12. Фрагмент коду Python, що демонструє запит виведення вузлів дерева на заданому рівні у MySQL.

```python
def search_by_level_mongo(level):
    start = time.time()
    docs = list(mongo_products.find({"$or":[
        {"level_1":level},
        {"level_2":level},
        {"level_3":level}
    ]}, {"_id":0,"level_1":1,"level_2":1,"level_3":1}))
    records = [[d['level_1'], d['level_2'], d['level_3']] for d in docs]
    end = time.time()
    print_result(f"MongoDB - Пошук за рівнем '{level}'", records, end-start)
```

Рис. 13. Фрагмент коду Python, що демонструє запит виведення вузлів дерева на заданому рівні у MongoDB.

```python
def search_by_level_neo4j(level):
    with neo4j_driver.session() as session:
        start = time.time()
        result = session.run(
            """
            MATCH (c:Category)
            WHERE c.name=$name
            OPTIONAL MATCH (c)<-[:HAS_SUBCATEGORY*0..]-(ancestor)
            RETURN DISTINCT c.name AS category_name, ancestor.name AS ancestor_name
            """,
            name=level
        )
        records = [(r['category_name'], r['ancestor_name']) for r in result]
        end = time.time()
        print_result(f"Neo4j - Пошук за рівнем '{level}'", records, end-start)
```

Рис. 14. Фрагмент коду Python, що демонструє запит виведення вузлів дерева на заданому рівні у Neo4j.

Під час тестування виявилось, що для виведення вузлів дерева на заданому рівні при невеликій кількості даних найкраще справляється з поставленими завданнями база даних MySQL (рис. 13).

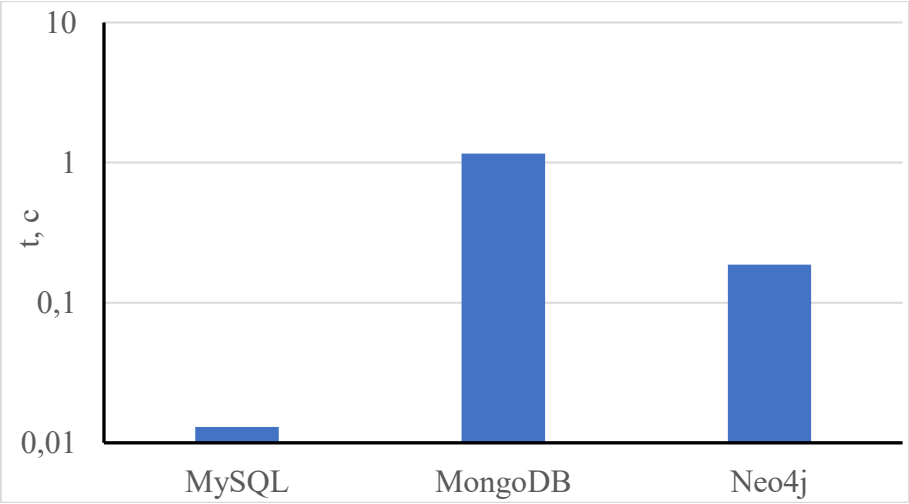


Рис.15. Час виконання операції виведення вузлів дерева на другому рівні

Висновки. Це експериментальне дослідження проводилося на малих наборах даних (каталог товарів із 52 категоріями та 720 товарами) і мало на меті порівняти ефективність виконання запитів до деревоподібної структури, реалізованої в різних моделях баз даних: реляційній (MySQL), графовій (Neo4j) та документоорієнтованій (MongoDB).

Аналіз виконання трьох типів запитів – отримання безпосередніх нащадків, побудови повного ланцюга предків і послідовного обходу дерева рівень за рівнем – показав, що за невеликого обсягу даних найвищу швидкодію демонструє реляційна СУБД MySQL.

Графова СУБД Neo4j забезпечила порівняно стабільну, але дещо нижчу продуктивність, що може бути зумовлено додатковими витратами на обробку графових зв'язків. Найменшу ефективність продемонструвала документоорієнтована СУБД MongoDB, що підтверджує її меншу придатність для операцій, пов'язаних із навігацією в ієрархічних структурах.

Отримані результати свідчать, що для невеликих ієрархічних структур реляційна модель даних може забезпечувати не гіршу, а іноді й кращу обчислювальну ефективність порівняно зі спеціалізованими графовими рішеннями. У подальших дослідженнях доцільно розширити обсяг даних і поглибити аналіз, щоб оцінити масштабованість та поведінку різних моделей за зростання складності ієрархії.

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: 24.11.2025).

7. Лайтарук, І., Куротич, А., Булатецька, Л. 2025. Операції оновлення ієрархічних структур у моделі вкладених множин . *Прикладні проблеми комп'ютерних наук, безпеки та математики*. 2025, 40–49. URL: https://apcssm.vnu.edu.ua/index.php/Journalone/article/view/132
8. Model Tree Structures - Database Manual - MongoDB Docs. *MongoDB: The World's Leading Modern Database | MongoDB*. URL: https://www.mongodb.com/docs/manual/tutorial/model-tree-structures/ (дата звернення: 24.11.2025).
9. Neo4j Documentation. *Graph Data Modeling*. URL: https://neo4j.com/docs/getting-started/ (дата звернення: 24.11.2025).
10. OrientDB Documentation. *Graph Database Introduction*. URL: https://orientdb.dev/ (дата звернення: 24.11.2025).

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
7. Laitaruk, I., Kurotych, A., Bulatetska, Л. 2025. Update operations for hierarchical structures in the nested sets model. *Applied Problems of Computer Science, Security and Mathematics*. 2025, 40–49. URL: https://apcssm.vnu.edu.ua/index.php/Journalone/article/view/132
8. Model Tree Structures - Database Manual - MongoDB Docs. *MongoDB: The World's Leading Modern Database | MongoDB*. URL: https://www.mongodb.com/docs/manual/tutorial/model-tree-structures/ (date of access: 24.11.2025).
9. Neo4j Documentation. *Graph Data Modeling*. URL: https://neo4j.com/docs/getting-started/ (date of access: 24.11.2025).
10. OrientDB Documentation. *Graph Database Introduction*. URL: https://orientdb.dev/ (date of access: 24.11.2025).