https://doi.org/10.31891/2307-5732-2025-359-47
УДК 004.89: 004.3

МОЛЧАНОВ БОГДАН
Київський Національний Університет Технологій та Дизайну
https://orcid.org/0000-0002-4633-9654
e-mail: molchanovbohdan@gmail.com
СТАЦЕНКО ВОЛОДИМИР
Київський Національний Університет Технологій та Дизайну
https://orcid.org/0000-0002-3932-792X
e-mail: statsenko.v@knutd.edu.ua

ДОСЛІДЖЕННЯ ВПЛИВУ ІНДЕКСАЦІЇ ТА ПАРТИЦІОНУВАННЯ БАЗИ ДАНИХ НА ШВИДКІСТЬ ВИКОНАННЯ ОПЕРАЦІЙ В СИСТЕМІ УПРАВЛІННЯ СКЛАДОМ

У роботі розглянуто проблему підвищення ефективності виконання транзакцій у реляційних базах даних, що використовуються в системах управління складом. З урахуванням сучасних викликів масштабування інформаційних систем та обробки великих обсягів даних, дослідження має на меті визначити вплив різних архітектурних підходів до структурування таблиць, зокрема індексації та партиціонування таблиць, на швидкість виконання ключових бізнес-операцій. У якості прикладного середовищасистеми управління базами даних (СУБД) обрано PostgreSQL, що дозволило реалізувати повноцінну експериментальну модель, наближену до умов промислової експлуатації.

Методика дослідження включає побудову реляційної бази даних із восьми взаємопов'язаних таблиць, що охоплює типову логіку обліку складів, продуктів, замовлень та утилізації списання з обліку. Було реалізовано три варіанти конфігурацій основної таблиці обліку запасів на складі з комбінаціями різних типів індексації й партиціонування за ідентифікатором складу та продукту. Кожна конфігурація моделює різні сценарії використання інструментів оптимізації в умовах інтенсивного навантаження.

Для об'єктивного вимірювання продуктивності використано утиліту pgbench зі встановленими параметрами багатопотокового навантаження та обмеженим часовим інтервалом тестування. Чотири базові операції — перевірка залишків, переміщення, приймання та списання товарів — були реалізовані у вигляді SQL-скриптів, що імітують реальні бізнес-виклики задачі. За результатами тестування сформовано порівняльну таблицю продуктивності кожної конфігурації, з подальшим аналізом ефективності підходів.

Запропоноване дослідження дозволяє системно оцінити вплив архітектурних налаштувань на продуктивність реляційної бази даних у контексті високонавантажених транзакційних сценаріїв. Побудована методика може бути використана для подальших експериментів на більших обсягах даних, із альтернативними підходами до партиціонування, порівнянням із нереляційними СУБД або іншими технічними конфігураціями. Робота має прикладне значення для фахівців у галузі баз даних, оптимізації продуктивності та розробки систем класу WMS (Warehouse Management System).

Ключові слова: *PostgreSQL, реляційна база даних, архітектура баз даних, індексація, партиціонування, управління запасами.*

MOLCHANOV BOHDAN
STATSENKO VOLODYMYR
Kyiv National University of Technologies and Design

RESEARCH ON THE IMPACT OF INDEXING AND PARTITIONING OF THE DATABASE ON THE PERFORMANCE OF OPERATIONS IN THE WAREHOUSE MANAGEMENT SYSTEM

This paper addresses the problem of optimizing transaction performance in relational databases used in warehouse management systems. In response to the growing demands for scalability and efficient data processing in high-load environments, the research aims to evaluate the impact of structural configurations—particularly indexing and partitioning—on the execution speed of key business operations. PostgreSQL was selected as the experimental platform due to its advanced optimization capabilities and wide industrial adoption.

The study employs a relational data model comprising eight interrelated tables representing a typical warehouse system, including products, categories, suppliers, stock balances, and disposal records. Three architectural configurations of the main operational table (warehouse stock) were developed, including default indexing and combinations of partitioning and composite indexing on warehouse and product identifiers. Each configuration models a distinct approach to database optimization under intensive transactional load.

To assess performance objectively, the pgbench benchmarking tool was used with multithreaded workloads and fixed-duration tests. Four core operations—stock availability check, product transfer between warehouses, goods intake, and inventory disposal—were implemented as SQL procedures simulating real-world scenarios. Each test scenario yielded performance metrics in terms of transactions per second, enabling comparative evaluation of all configurations.

The proposed methodology allows for a systematic assessment of how architectural decisions influence relational database performance in transaction-heavy systems. The experiment provides a foundation for further research involving larger datasets, alternative partitioning schemes, or comparative evaluation with other DBMS platforms and hardware setups. The work holds practical significance for professionals involved in database architecture, performance optimization, and the development of scalable warehouse management solutions.

Keywords: *PostgreSQL, relational database, database architecture, indexing, partitioning, inventory management.*

Стаття надійшла до редакції / Received 14.09.2025
Прийнята до друку / Accepted 17.10.2025

Постановка проблеми

Сьогодні реляційні бази даних є одним з найбільш розповсюджених інструментів зберігання та роботи з даними. Вони дозволяють створювати складні структури даних, оптимізувати її, виконувати пошук,

створювати звіти, мають вбудовані засоби забезпечення цілісності даних, контролю доступу, резервного копіювання та інші інструменти. Поширення веб-застосунків та іншого програмного забезпечення, що працює через Інтернет, зумовило підвищення вимог до баз даних, оскільки саме вони забезпечують централізоване зберігання інформації багатьох тисяч користувачів. Це зумовлює збільшення швидкості виконання операцій базою даних, необхідність оптимізації її структури, створення індексів та постійний моніторинг її роботи.

В більшості випадків структура бази даних, зокрема, перелік таблиць та зв'язки між ними, визначаються розробником, виходячи з конкретних задач. Відправною точкою для її побудови є структура інформації, яку потрібно зберігати, та перелік операцій, що має виконувати застосунок. Складність такого проєктування полягає в тому, що ефективність тих чи інших методів оптимізації залежить від багатьох факторів, що визначаються особливостями роботи застосунку, зокрема, складністю SQL запитів, їх кількістю, характером запитів (читання або зміна даних), загальною кількістю інформації, що зберігається в таблицях. Так, використання індексів в загальному випадку зменшує час пошуку даних. Але при цьому збільшується час операцій вставки та зміни даних, окрім того, фактична величина збільшення швидкості пошуку залежить від унікальності індексу, яку можливо оцінити тільки за фактичними даними в конкретних таблицях. В результаті, не зважаючи на те, що існують добре відомі принципи оптимізації роботи бази даних, ефективність їх використання в кожному конкретному випадку потребує дослідження.

Аналіз досліджень і публікацій

Наявні дослідження показують, що правильний підбір налаштувань, а згодом індексація та за потреби – партиціонування, за правильного підходу значно підвищують продуктивність та ефективність в управлінні базами даних [1, 2].

Стандартна B-tree індексація у PostgreSQL дозволяє визначити колонки, значення яких із рядків увійдуть у окрему оптимізовану таблицю у структурі багаторівневого дерева, де значенням листка буде кортеж із значенням з рядка та фізичним посиланням на нього у таблиці [3]. Це дозволяє значно пришвидшити виконання пошукових запитів за ціною сповільнення операцій запису та підвищенням вимог до сховища через потребу в місці для зберігання індексів.

Партиціонування – це процес фізичного розділення однієї таблиці на вказану кількість менших частин, що називаються партиціями, при чому для користувача та для сховища таблиця залишиться єдиним логічним об'єктом. Для партиціонування таблиці необхідно мати ключ, який буде розділяти партиції, що буде передаватись у кожному запиті для того, щоб система управління базами даних могла автоматично визначити, у якій партиції знаходяться шукані рядки. Дана практика може значно пришвидшити продуктивність за перспективи наявності великих даних у таблиці, хоч і надає деякі обмеження, оскільки пошук по таблиці без передачі ключа партиціонування буде виконувати пошук у кожній із встановлених партицій [4].

Обидва процеси, як індексація, так і партиціонування, допомагають знаходити рядки значно швидше. Окрім цього, поєднання цих стратегій може призвести до покращення ефективності на великих обсягах даних.

Формулювання цілей статті

Метою дослідження є: визначення ефективності використання індексів таблиць та створення партицій в базі даних, що призначена для зберігання та поточної роботи з даними системи керування складом.

Виклад основного матеріалу

У роботі спроєктовано структуру бази даних для роботи інформацією, що має зберігатись в системі керування складом. Запропонована база даних складається з восьми пов'язаних між собою таблиць.

Інформація про товари зберігається у таблиці Products. Кожен запис в цій таблиці пов'язаний з виробником товару, який зберігається в таблиці ProductManufacturers. Тип зв'язку між цими таблицями «один-до-багатьох». Також кожний рядок з інформацією про товар пов'язаний із своєю категорією, таблиця ProductCategories, що має зв'язок до Products як «один-до-багатьох».

Таблиця Warehouses містить в собі інформацію про склади. Вона пов'язана як «один-до-багатьох» із таблицею ProductOrders, яка містить інформацію про замовлення товарів на склад. В свою чергу, ProductOrders пов'язана як «один-до-багатьох» із таблицею з інформацією про постачальників DeliveryOrganizations, а також з Products.

Основною операційною таблицею, над якому проводились експерименти, є WarehouseProducts. Основним її призначенням є вміщення інформації про наявність продукту на складі. Вона пов'язана як «багато-до-одного» із таблицями Warehouses та Products. Окрім цього, вона пов'язана як «один-до-багатьох» із таблицею WarehouseProductDisposals, що містить рядки про списання чи утилізацію товару зі складу.

Структурна схема бази показана на рис. 1.

З метою дослідження часу виконання бізнес-операцій була створена база даних в СУБД Postgres, яка сьогодні є однією з найбільш поширених та функціональних [5].

Запропонована база дозволяє виконувати основні бізнес-операції з даними, що є характерними для більшості систем керування складами, а саме:

1. Перевірка кількості товару на складі
2. Переміщення товару поміж складами
3. Доставка товарів до складу
4. Списання товарів зі складу

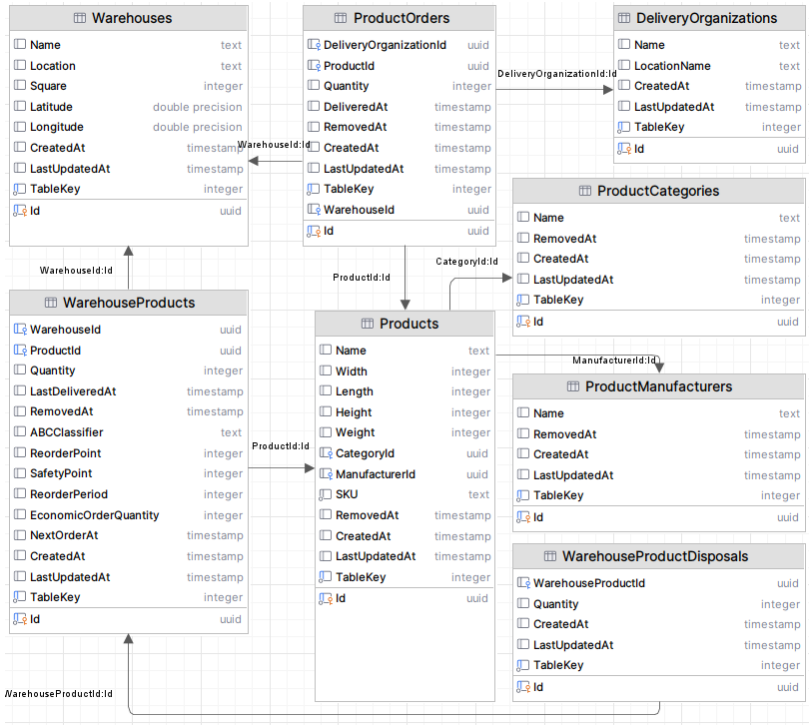

Рис 1. Структурна схема бази даних системи керування складом.

Ці операції були обрані для дослідження тому, що вони включають кілька дій та зумовлюють більше навантаження на систему керування базою даних ніж, наприклад, операції додавання товару чи пошуку за назвою. Водночас, саме ці операції забезпечують зберігання інформації про рух та наявність товарів, що є одним з основних завдань системи керування складом.

Для проведення досліджень всі таблиці були наповнені тестовими даними. Кількість тестових записів наведена в табл. 1.

Таблиця 1

Кількість тестових рядків у таблицях

№	Назва таблиці	Кількість записів
1	DeliveryOrganizations	5
2	ProductCategories	10
3	ProductManufacturers	10
4	ProductOrders	10 000
5	Products	1 000
6	WarehouseProductDisposals	10 000
7	WarehouseProducts	1 000 000
8	Warehouses	1 000

Дослідження швидкодії виконання операцій проводились за налаштувань Postgres, зазначених у табл. 2.

Таблиця 2

Налаштування бази даних, відмінні від стандартних

№	Назва налаштування	Значення	№	Назва налаштування	Значення
1	max_connections	300	10	work_mem	27235kB
2	shared_buffers	8GB	11	huge_pages	try
3	effective_cache_size	24GB	12	min_wal_size	2GB
4	maintenance_work_mem	2GB	13	max_wal_size	8GB
5	checkpoint_completion_target	0.9	14	max_worker_processes	8
6	wal_buffers	16MB	15	max_parallel_workers_per_gather	4
7	default_statistics_target	100	16	max_parallel_workers	8
8	random_page_cost	4	17	max_parallel_maintenance_workers	4
9	effective_io_concurrency	2			

Для визначення впливу індексів та партицій створено шість баз даних з однаковою структурою, але для кожної з них встановлені наступні налаштування для таблиці WarehouseProducts, де виконуються основні операції, а саме:

A. Унікальний індекс по WarehouseId, ProductId (зв'язці складу та продукту).
B. Партиціонування по WarehouseId з індексом ProductId.

C. Індексація по ключу WarehouseId, ProductId без унікальності.

Визначення швидкості виконання операцій здійснювалось за допомогою утиліти pgbench із налаштуваннями, описаними у табл. 3.

Таблиця 3

Параметри запуску утиліти pgbench

Налаштування	Опис налаштування	Значення
-c	Визначає кількість паралельних клієнтських сесій	64
-j	Визначає кількість потоків для виконання операцій всередині однієї сесії	8
-T	Визначає час виконання тесту в секундах	60

Параметри сервера, на якому виконувалось дослідження:

- Процесор AMD Ryzen 7 7800X3D, 4.20 ГГц, 8 ядер, 16 потоків.
- Оперативна пам'ять 32 Гб, 6000 МТ/с.
- Накопичувач Samsung SSD 970 EVO 1 Тб.

По кожному експерименту для кожної конфігурації зафіксовано загальну кількість транзакцій, виконаних за хвилину, а також кількість транзакцій на секунду.

Процедура знаходження кількості продукту на складі. Для здійснення даної операції користувачі знаходять продукт та склад, на якому він знаходиться. В результаті – виводиться кількість даного товару на складі. Запит наведений у лістингу 1.

Лістинг 1

Процедура знаходження кількості продукту на складі

```
DO $$
DECLARE
  v_product_id UUID; v_warehouse_id UUID; v_quantity INTEGER;
BEGIN
  SELECT "Id" INTO v_product_id FROM "Products" ORDER BY random() LIMIT 1;
  SELECT "Id" INTO v_warehouse_id FROM "Warehouses" ORDER BY random() LIMIT 1;
  SELECT COALESCE((
SELECT "Quantity" FROM "WarehouseProducts" WHERE "ProductId" = v_product_id
      AND "WarehouseId" = v_warehouse_id LIMIT 1
  ), 0) INTO v_quantity;
END $$;
```

Результати дослідження наведено у табл. 4.

Таблиця 4

Результати тестування швидкодії операції знаходження продукту на складі

Налаштування	Загальна кількість оброблених транзакцій	Транзакцій на секунду
A	2026169	33775.0
B	1916995	31958.3
C	2069143	34492.7

Найбільш ефективною при виконанні процедури знаходження кількості продукту на складі є налаштування C. Водночас, швидкість виконання для A виявилась на 2.12% менша.

Процедура переміщення товарів. Даний процес часто затребуваний, оскільки часто є потреба змінити розміщення товару у зв'язку з переміщенням чи потребі звільнити простір. Процедура передбачає транзакцію із знаходження складу v_source_wh_id, на якому наявний товар, знаходження іншого складу v_target_wh_id. Перевіряється, чи був товар колись на складі v_target_wh_id. Якщо був створений, то до кількості даного товару додається кількість товару v_move_qty, якщо не був створений – позиція товару на складі створюється і до нього додається v_move_qty товару. Після цього v_move_qty товару списується із складу v_source_wh_id. Запит наведено у лістингу 2.

Лістинг 2

Процедура переміщення товару до іншого складу

```
DO $$
DECLARE
  v_product_id UUID; v_source_wp_id UUID; v_target_wp_id UUID;
  v_source_wh_id UUID; v_target_wh_id UUID; v_current_qty INTEGER;
  v_move_qty INTEGER;
BEGIN
  SELECT wp."ProductId", wp."Id", wp."WarehouseId", wp."Quantity"
  INTO v_product_id, v_source_wp_id, v_source_wh_id, v_current_qty
  FROM "WarehouseProducts" wp WHERE wp."Quantity" > 0
  ORDER BY random() LIMIT 1;
  IF v_source_wp_id IS NULL THEN
    RETURN;
  END IF;
```

Продовження. Лістинг 2

```
  SELECT "Id" INTO v_target_wh_id FROM "Warehouses"
  WHERE "Id" != v_source_wh_id ORDER BY random() LIMIT 1;
  SELECT "Id" INTO v_target_wp_id FROM "WarehouseProducts"
  WHERE "ProductId" = v_product_id AND "WarehouseId" = v_target_wh_id
  LIMIT 1;
  IF v_target_wp_id IS NULL THEN
   SELECT gen_random_uuid() INTO v_target_wp_id;
      INSERT INTO "WarehouseProducts"
      ("Id", "WarehouseId", "ProductId", "Quantity")
       VALUES (v_target_wp_id, v_target_wh_id, v_product_id, 0);
  END IF;
  v_move_qty := LEAST(v_current_qty, GREATEST(1, FLOOR(random() * 50)::int));
  UPDATE "WarehouseProducts" SET
    "Quantity" = "Quantity" - v_move_qty, "LastUpdatedAt" = now(),
    "RemovedAt" = null, "LastDeliveredAt" = now()
  WHERE "WarehouseId" = v_source_wh_id AND "Id" = v_source_wp_id;
  UPDATE "WarehouseProducts" SET
    "Quantity" = "Quantity" + v_move_qty, "LastUpdatedAt" = now()
  WHERE "WarehouseId" = v_target_wh_id AND "Id" = v_target_wp_id;
END $$;
```

Результати дослідження наведено у табл. 5.

Таблиця 5

Результати тестування швидкодії операції переміщення товару між складами

Налаштування	Загальна кількість оброблених транзакцій	Транзакцій на секунду
A	15104	251.3
B	15085	242.1
C	15884	264.3

Найбільш ефективною при виконанні процедури переміщення є комбінація C. Водночас, швидкість виконання для налаштування A тільки на 5,17% менша.

Процедура прибуття продукту на склад. Дана процедура знаходить продукт, склад, організацію, що доставляє товари та кількість товару, що прибула. Після цього перевіряється існування рядка із позицією товару. Якщо він не існує, позиція створюється. Відбувається запис факту прибуття товару на склад і інкрементується кількість продукту згідно його позиції на складі. Запит наведено у лістингу 3.

Лістинг 3

Запит прийняття продукту на склад

```
DO $$
DECLARE
  v_product_id UUID; v_warehouse_id UUID; v_wp_id UUID;
  v_quantity INTEGER; v_do_id UUID;
BEGIN
  SELECT "Id" INTO v_product_id FROM "Products"
  ORDER BY random() LIMIT 1;
  SELECT "Id" INTO v_warehouse_id FROM "Warehouses"
  ORDER BY random() LIMIT 1;
  SELECT "Id" INTO v_do_id FROM "DeliveryOrganizations"
  ORDER BY random() LIMIT 1;
  v_quantity := FLOOR(random() * 200 + 1)::int;
  SELECT "Id" INTO v_wp_id FROM "WarehouseProducts"
  WHERE "ProductId" = v_product_id AND "WarehouseId" = v_warehouse_id LIMIT 1;
  IF v_wp_id IS NULL THEN
    v_wp_id := gen_random_uuid();
    INSERT INTO "WarehouseProducts" (
      "Id", "WarehouseId", "ProductId", "Quantity",
      "CreatedAt", "LastUpdatedAt")
    VALUES ( v_wp_id, v_warehouse_id, v_product_id, 0,
      now(), now() );
END IF;
  INSERT INTO "ProductOrders" (
    "Id", "DeliveryOrganizationId", "ProductId", "Quantity", "DeliveredAt",
    "CreatedAt", "LastUpdatedAt" ) VALUES ( gen_random_uuid(), v_do_id,
    v_product_id, v_quantity, now(), now(), now());
  UPDATE "WarehouseProducts" SET
    "Quantity" = "Quantity" + v_quantity, "LastDeliveredAt" = now(),
    "LastUpdatedAt" = now() WHERE "WarehouseId" = v_warehouse_id
    AND "Id" = v_wp_id;
END $$;
```

Результати дослідження наведено у табл. 6.

Таблиця 6

Результати тестування швидкодії операції прийняття продукту на склад

Налаштування	Загальна кількість оброблених транзакцій	Транзакцій на секунду
A	903532	15063.6
B	837492	13957.5
C	908607	15144.3

Найбільш продуктивним у даній операції виявилось рішення C, де найближчий результат – у A, що відстає на 0.54%.

Процедура списання продукту зі складу. Даний процес зазначає списання товару зі складу для відправки замовнику або для утилізації. Транзакція передбачає знаходження товару, складу, позиції товару на складі. Якщо на складі нічого немає – процедура завершається. Проте у тестовій базі даних кожна позиція має велику кількість товару, тому продовжуючись, процедура додає запис до таблиці зі списаннями, а також декрементує кількість товару на складі. Запит наведено у лістингу 4.

Лістинг 4

Запит на списання товару зі складу

```
DO $$
DECLARE
  v_product_id UUID; v_warehouse_id UUID;
  v_wp_id UUID; v_current_qty INTEGER;
  v_dispose_qty INTEGER;
BEGIN
  SELECT "Id" INTO v_product_id
  FROM "Products" ORDER BY random() LIMIT 1;
  SELECT "Id" INTO v_warehouse_id
  FROM "Warehouses" ORDER BY random() LIMIT 1;
  SELECT "Id", "Quantity" INTO v_wp_id, v_current_qty
  FROM "WarehouseProducts"
  WHERE "ProductId" = v_product_id
    AND "WarehouseId" = v_warehouse_id
  LIMIT 1;
  IF v_wp_id IS NULL OR v_current_qty <= 0 THEN
    RETURN;
  END IF;
  v_dispose_qty := LEAST(v_current_qty, GREATEST(1, FLOOR(random() * 100)::int));
  INSERT INTO "WarehouseProductDisposals" (
    "Id", "WarehouseProductId", "Quantity", "CreatedAt", "LastUpdatedAt", "WarehouseId"
  ) VALUES (
    gen_random_uuid(), v_wp_id, v_dispose_qty, now(), now(), v_warehouse_id
  );
UPDATE "WarehouseProducts"
  SET
    "Quantity" = "Quantity" - v_dispose_qty,
    "LastUpdatedAt" = now()
  WHERE "WarehouseId" = v_warehouse_id AND "Id" = v_wp_id;
END $$;
```

Результати наведено у табл. 7.

Таблиця 7

Результати тестування швидкодії операції списання товару

Налаштування	Загальна кількість оброблених транзакцій	Транзакцій на секунду
A	1286179	21444.2
B	1030831	17186.7
C	1337681	22299.0

Найвища ефективність у даній операції – у процедури C. Процедура з налаштуваннями A відстає на 3.99%.

Висновки

1. Розроблено структуру бази даних для системи управління складом, що дозволяє виконувати найбільш поширені операції з товарами.
2. Розроблена база даних реалізована з використанням СУБД PostgreSQL та виконано її наповнення тестовими даними.

3. Найбільш високу продуктивність на кількості даних із експерименту продемонструвала конфігурація бази даних із неунікальним індексом по полям складу та продукту, які є ключовими під час обробки усіх транзакцій під час проведення експерименту.

4. Індексація з цими ж полями, але з унікальністю, показала в середньому на 2.41% гірший результат, що свідчить про додаткові витрати часу на валідацію обмежень.

5. Варіант із партиціонуванням по ідентифікатору складу та індексом по ідентифікатору товару відпрацював повільніше на 14.06% від найкращого варіанту.

6. Подальші дослідження можуть включати випробування на більших обсягах даних, інших варіантах партиціонування, а також порівняння з іншими СУБД та апаратними конфігураціями.

Література

1. Avula S. PostgreSQL Table Partitioning Strategies: Handling Billions of Rows Efficiently // International Journal of Emerging Trends in Computer Science and Information Technology. – 2024. – №. 5(3). – S. 23-37. – DOI: https://doi.org/10.63282/3050-9246.IJETCSIT-V5I3P103.

2. Vellanki R. PostgreSQL Configuration: Best Practices for Performance and Security // European Journal of Computer Science and Information Technology. – 2025. – №. 13(47). – S. 172-182. – DOI: https://doi.org/10.37745/ejcsit.2013/vol13n47172182.

3. Saringat M., Mostafa S., Mustapha A., Hassan M. A Case Study on B-Tree Database Indexing Technique // Journal of Soft Computing and Data Mining. – 2020. – №. 1(1). – S. 27-35. – DOI: https://doi.org/10.30880/jscdm.2020.01.01.004.

4. Гедеон Т., Гедеон Г. ПАРТИЦІОНУВАННЯ ЯК ПІДХІД ДО ОПТИМІЗАЦІЇ ЗБЕРІГАННЯ ТА ВИБІРКИ ВЕЛИКИХ МАСИВІВ ДАНИХ У ІНФОРМАЦІЙНИХ СИСТЕМАХ // Наука і техніка сьогодні. – 2024. – №. 5(33). – С. 1119-1128. – DOI: https://doi.org/10.52058/2786-6025-2024-5(33)-1119-1128.

5. Choina M. Performance analysis of relational databases MySQL, PostgreSQL and Oracle using Doctrine libraries // Journal of Computer Sciences Institute. – 2022. – №. 24. – S. 250-257. – DOI: https://doi.org/10.35784/jcsi.3000.

References

1. Avula S. PostgreSQL Table Partitioning Strategies: Handling Billions of Rows Efficiently // International Journal of Emerging Trends in Computer Science and Information Technology. – 2024. – №. 5(3). – S. 23-37. – DOI: https://doi.org/10.63282/3050-9246.IJETCSIT-V5I3P103.

2. Vellanki R. PostgreSQL Configuration: Best Practices for Performance and Security // European Journal of Computer Science and Information Technology. – 2025. – №. 13(47). – S. 172-182. – DOI: https://doi.org/10.37745/ejcsit.2013/vol13n47172182.

3. Saringat M., Mostafa S., Mustapha A., Hassan M. A Case Study on B-Tree Database Indexing Technique // Journal of Soft Computing and Data Mining. – 2020. – №. 1(1). – S. 27-35. – DOI: https://doi.org/10.30880/jscdm.2020.01.01.004.

4. Gedeon T., Gedeon H. PARTITIONING AS AN APPROACH TO OPTIMIZING THE STORAGE AND RETRIEVAL OF LARGE DATA ARRAYS IN INFORMATION SYSTEMS // Nauka i tekhnika sohodni. – 2024. – №. 5(33). – P. 1119-1128. – DOI: https://doi.org/10.52058/2786-6025-2024-5(33)-1119-1128.

5. Choina M. Performance analysis of relational databases MySQL, PostgreSQL and Oracle using Doctrine libraries // Journal of Computer Sciences Institute. – 2022. – №. 24. – S. 250-257. – DOI: https://doi.org/10.35784/jcsi.3000.