Міністерство освіти і науки України
Криворізький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерних систем та мереж

Пояснювальна записка
до кваліфікаційної роботи бакалавра
за спеціальністю 123 «Комп'ютерна інженерія»

на тему: ПОРІВНЯННЯ POSTGRESQL ТА MONGODB НА ПРИКЛАДАХ РІШЕНЬ ДЛЯ IOT

Проектував _____ О.С.Чиняєв

Керівник роботи _____ С.В.Сьомочкина

Нормоконтроль _____ Д. І. Кузнєцов

Завідувач кафедри _____ А. І. Купін

Кривий Ріг
202

ПЕРЕЛІК ВИКОРИСТАНИХ СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

ACID – Atomicity, Consistency, Isolation, Durability (атомарність, узгодженість, ізоляція, стійкість)

BSON – Binary JSON

IoT – Інтернет речей (Internet of Things)

JSON – JavaScript Object Notation

JSONB – Binary JSON

NoSQL – Not Only SQL

SQL – Structured Query Language

СУБД – система управління базами даних

WAL – Write-Ahead Logging (журнал попереднього запису)

ЗМІСТ

ВСТУП

Сучасний розвиток технологій Інтернету речей (IoT) кардинально змінює підходи до збору, зберігання та обробки даних у всіх сферах людської діяльності. За прогнозами аналітичних агентств, до 2026 року кількість підключених IoT-пристроїв у світі перевищить 30 мільярдів, а обсяг генерованих ними даних сягне десятків зетабайт щорічно. Переважна більшість цих даних має характер часових рядів (time-series): показники датчиків температури, вологості, тиску, рівня вібрації, геолокації, споживання енергії тощо. Такі дані характеризуються високою швидкістю надходження, значним обсягом, частою неповнотою та необхідністю швидкого аналізу в реальному часі. Від ефективності систем управління базами даних (СУБД), що забезпечують зберігання та обробку цих даних, безпосередньо залежить надійність, масштабованість та економічна ефективність усього IoT-рішення.

На сьогоднішній день на ринку представлено два основні класи СУБД, які найчастіше застосовуються в IoT-проектах: реляційні (SQL) та документо-орієнтовані NoSQL. Серед реляційних систем лідируючі позиції посідає PostgreSQL завдяки своїй розширюваності, підтримці JSONB, потужним механізмам індексації та вбудованій підтримці часових рядів через розширення TimescaleDB. Серед NoSQL-систем домінує MongoDB, яка пропонує гнучку схему даних, горизонтальне масштабування, вбудовані можливості роботи з time-series collections та високопродуктивні операції запису. Вибір між цими двома системами є одним із ключових архітектурних рішень при проектуванні IoT-платформ, оскільки безпосередньо впливає на продуктивність, вартість інфраструктури, складність розробки та підтримки системи.

Актуальність теми кваліфікаційної роботи зумовлена швидким поширенням IoT-рішень в Україні та світі, зокрема в сферах смарт-сільського господарства, промислового моніторингу (Industry 4.0), розумних міст, логістики та енергетики. Водночас в науковій і технічній літературі недостатньо детально висвітлено практичні аспекти порівняння PostgreSQL та MongoDB саме на реальних прикладах IoT-систем з урахуванням українських умов (обмежена пропускна здатність каналів зв’язку, вимоги до енергозбереження на edge-рівні, необхідність інтеграції з національними платформами моніторингу). Відсутність систематизованого порівняльного аналізу з урахуванням специфіки IoT-трафіку призводить до прийняття суб’єктивних рішень розробниками, що часто призводить до перевитрат ресурсів або недостатньої продуктивності готових рішень.

Метою кваліфікаційної роботи є проведення комплексного порівняльного аналізу можливостей систем управління базами даних PostgreSQL та MongoDB для зберігання, обробки та аналізу даних у системах Інтернету речей на основі реальних прикладів IoT-рішень.

Для досягнення поставленої мети в роботі вирішуються такі завдання:

1. Проаналізувати особливості систем Інтернету речей та специфічні вимоги, які вони висувають до СУБД.
2. Провести огляд архітектури, основних функціональних і нефункціональних характеристик PostgreSQL та MongoDB.
3. Виконати порівняльний аналіз обох систем за ключовими критеріями: продуктивність запису/читання, масштабованість, підтримка time-series даних, витрати на інфраструктуру, зручність розробки та інтеграції.
4. Розглянути практичні приклади застосування PostgreSQL та MongoDB у реальних IoT-проектах.
5. Сформулювати рекомендації щодо вибору СУБД залежно від типу IoT-задачі, обсягу даних та умов експлуатації.

Об’єктом дослідження є системи управління базами даних, що застосовуються для зберігання та обробки даних в IoT-платформах. Предметом дослідження є порівняльні характеристики PostgreSQL та MongoDB у контексті IoT-рішень.

У роботі використані методи системного аналізу, порівняльного аналізу, моделювання даних, а також вивчення офіційної технічної документації та наукових публікацій.

Практична значущість результатів полягає в наданні розробникам та інженерам комп’ютерних систем чітких рекомендацій щодо вибору СУБД для IoT-проектів, що дозволить оптимізувати архітектуру, знизити витрати на інфраструктуру та підвищити ефективність обробки даних.

Кваліфікаційна робота складається з вступу, трьох розділів, висновків, списку використаних джерел та додатків. У першому розділі розглянуто теоретичні основи IoT та вимоги до СУБД. Другий розділ присвячено порівняльному аналізу PostgreSQL та MongoDB. У третьому розділі наведено практичні приклади застосування обох систем у IoT-рішеннях та сформовано рекомендації.

1 АНАЛІЗ ВИМОГ ДО СИСТЕМ УПРАВЛІННЯ БАЗАМИ ДАНИХ В СИСТЕМАХ ІНТЕРНЕТУ РЕЧЕЙ

1.1 Характеристика систем Інтернету речей та особливості обробки даних

Сучасні системи Інтернету речей (IoT) являють собою глобальну, динамічно розвиваючуся екосистему, яка об’єднує мільйони фізичних пристроїв, оснащених вбудованими датчиками, актуаторами, мікроконтролерами, програмним забезпеченням та технологіями бездротового зв’язку. Ці пристрої здатні самостійно збирати, передавати, обробляти та обмінюватися даними в режимі реального часу без безпосередньої участі людини. За визначенням, наведеним у наукових джерелах, IoT – це парадигма, у якій фізичні об’єкти стають частиною інформаційної мережі, отримуючи можливість ідентифікації, комунікації та інтелектуальної взаємодії [3].

До 2026 року, згідно з даними Всеукраїнської науково-технічної конференції «Сучасний стан та перспективи розвитку IoT», кількість активних IoT-пристроїв як у світі, так і в Україні сягне десятків мільярдів одиниць. Це призведе до генерації сотень зетабайт інформації щорічно, що в десятки разів перевищує обсяги даних, які оброблялися традиційними інформаційними системами ще п’ять років тому [19]. Такий експоненційний ріст обумовлений широким впровадженням IoT у промисловості (Industry 4.0), сільському господарстві, транспорті, енергетиці, медицині, розумних містах та побутовій сфері.

Основна частина даних, що генеруються IoT-системами, має характер часових рядів (time-series). Вони надходять від промислових сенсорів температури, тиску, вібрації, вологості ґрунту, рівня заповнення резервуарів, розумних лічильників електроенергії, сільськогосподарських дронів, систем моніторингу якості повітря, логістичних трекерів та багатьох інших джерел. Кожна така послідовність даних супроводжується точною часовою міткою, що робить її особливо цінною для оперативного аналізу та прогнозування.

Системи Інтернету речей характеризуються високою гетерогенністю обладнання: від недорогих мікроконтролерів ESP32 та Arduino до промислових PLC і потужних edge-серверів. Архітектура IoT зазвичай є розподіленою (edge-cloud), де частина обробки відбувається безпосередньо на пристрої або локальному шлюзі (edge layer), а складніша аналітика та довгострокове зберігання – у хмарному середовищі. Така архітектура забезпечує мінімальну латентність для критичних застосувань, але одночасно створює серйозні виклики щодо забезпечення безперервної роботи в умовах обмеженої пропускної здатності каналів зв’язку, нестабільного інтернету (особливо в сільській місцевості України) та обмежених енергетичних ресурсів автономних пристроїв [1].

Особливості обробки даних в IoT суттєво відрізняються від традиційних корпоративних інформаційних систем. Якщо в класичних БД дані надходять відносно рідко і в структурованому вигляді, то в IoT потік інформації є постійним, високошвидкісним і часто неструктурованим. Дані генеруються з частотою від кількох до тисяч записів за секунду на один пристрій. Вони характеризуються значним обсягом, варіативністю форматів (від простих числових значень до складних JSON-структур, бінарних даних та мультимедійного контенту), високою динамічністю, частковою неповнотою та необхідністю забезпечення конфіденційності, цілісності та доступності в умовах потенційних кіберзагроз [14].

До ключових характеристик обробки даних в IoT-системах, які часто називають «п’ятьма V» Big Data в контексті IoT, належать:

- Volume (обсяг) – величезний обсяг даних, що вимірюється терабайтами та петабайтами щодня навіть для середнього промислового об’єкта або фермерського господарства;
- Velocity (швидкість) – висока швидкість генерації та надходження даних від тисяч одночасно працюючих сенсорів;
- Variety (різноманітність) – широкий спектр типів даних: структуровані, напівструктуровані (JSON), неструктуровані та часові ряди;
- Veracity (достовірність) – необхідність обробки даних з урахуванням можливих помилок, шумів та неповноти;
- Value (цінність) – потреба швидкого вилучення корисної інформації для прийняття оперативних рішень у реальному часі.

Ці характеристики створюють надзвичайно жорсткі вимоги до систем управління базами даних. СУБД повинні забезпечувати ефективне зберігання, швидкий запис, індексацію часових рядів, підтримку складних аналітичних запитів, інтеграцію з edge-обчисленнями та високу стійкість до збоїв [35]. Традиційні реляційні СУБД часто не справляються з таким навантаженням через жорстку схему даних, вертикальне масштабування та значні накладні витрати на підтримку ACID-транзакцій під час масового запису. NoSQL-рішення, навпаки, пропонують високу гнучкість і продуктивність запису, але можуть поступатися в забезпеченні транзакційної цілісності та складності аналітики [3].

Таблиця 1.1 – Основні характеристики даних в системах Інтернету речей

Характеристика	Опис в IoT-системах	Наслідки для СУБД
Обсяг (Volume)	Сотні ТБ/день на об’єкт	Потреба в розподілених сховищах
Швидкість (Velocity)	Тисячі записів/сек	Високопродуктивний запис без блокувань
Різноманітність (Variety)	JSON, time-series, binary	Гнучка схема або підтримка JSONB

Реальний час	Латентність < 100 мс	Індексація та запити в реальному часі
Надійність	Робота в умовах збоїв зв’язку	Реплікація та fault-tolerance

Ефективна обробка даних в IoT вимагає від СУБД поєднання високої продуктивності запису, підтримки часових рядів, масштабованості та зручності інтеграції з існуючими IoT-платформами. Вибір відповідної системи (PostgreSQL чи MongoDB) стає ключовим фактором успішності всього рішення, оскільки безпосередньо впливає на вартість інфраструктури, швидкість розробки та надійність експлуатації в реальних українських умовах [1; 19]. Подальший аналіз цих систем дозволить визначити оптимальні сценарії їх застосування в IoT-проектах.

Характеристика «Volume» (обсяг) є однією з найбільш очевидних і водночас найскладніших для традиційних систем. Середній промисловий об’єкт або велике сільськогосподарське господарство може генерувати від десятків до сотень терабайт даних щодня. Наприклад, одна сучасна метеостанція з 20–30 параметрами при частоті опитування 1 раз за секунду створює близько 2,5 ГБ даних на добу. При масштабі в тисячі пристроїв обсяг зростає до петабайтного рівня за кілька місяців. Такі обсяги вимагають від СУБД не лише великої ємності сховища, але й ефективних механізмів стиснення, партиціонування та автоматичного управління життєвим циклом даних [19].

Характеристика «Velocity» (швидкість) ще більш критична. У промислових IoT-системах частота надходження даних може сягати тисяч і навіть десятків тисяч записів за секунду на один пристрій. У реальному часі це означає необхідність обробки потоку даних без блокувань і з мінімальною латентністю. Будь-яке запізнення запису може призвести до втрати критичної інформації про аварійну ситуацію, перевищення порогових значень або відхилення технологічного процесу. Тому СУБД повинна підтримувати високопродуктивні операції вставки (high write throughput) і працювати в режимі append-only, де нові дані додаються без зміни вже існуючих записів [1].

Характеристика «Variety» (різноманітність) відображає гетерогенність джерел даних. У сучасній IoT-екосистемі одночасно присутні структуровані дані (числові показники), напівструктуровані (JSON-документи з різною структурою), неструктуровані (бінарні дані від камер або аудіосенсорів) та геопросторові дані. Така різноманітність робить неможливим використання жорсткої схеми таблиць у традиційних реляційних СУБД без постійних міграцій і перебудови індексів. Саме тому гнучкі документо-орієнтовані або гібридні моделі даних стають конкурентною перевагою [3].

Характеристика «Veracity» (достовірність) пов’язана з якістю даних. Через нестабільний зв’язок, розрядження акумуляторів, електромагнітні перешкоди або технічні збої сенсорів дані часто містять шуми, пропуски або помилки. СУБД повинна вміти працювати з неповними даними, підтримувати механізми валідації, виявлення аномалій у реальному часі та автоматичного відновлення цілісності [14].

Нарешті, характеристика «Value» (цінність) підкреслює головну мету IoT – швидке вилучення корисної інформації для прийняття рішень. Дані самі по собі не мають цінності; цінність виникає лише після оперативної обробки, агрегування, виявлення закономірностей і формування прогнозів. Тому СУБД повинна не лише зберігати дані, але й забезпечувати швидке виконання складних аналітичних запитів, materialized views, continuous aggregates та інтеграцію з системами машинного навчання [35].

Усі перелічені характеристики створюють унікальний набір вимог до СУБД, які значно відрізняються від вимог традиційних корпоративних систем. Традиційні реляційні СУБД, орієнтовані на вертикальне масштабування та жорстку схему, часто стикаються з проблемами продуктивності при масовому записі даних, високими накладними витратами на підтримку ACID-транзакцій і складнощами горизонтального масштабування. NoSQL-системи, навпаки, пропонують високу швидкість запису та гнучкість, але можуть поступатися в забезпеченні транзакційної цілісності, складності аналітики та довгостроковому зберіганні даних з можливістю точних агрегатів [3].

Особливої уваги заслуговує український контекст. Обмежена пропускна здатність мобільного та супутникового зв’язку в сільській місцевості, нестабільне електропостачання, вимоги до енергозбереження автономних пристроїв та необхідність відповідності національним нормам захисту персональних і критичних даних суттєво посилюють вимоги до СУБД. У таких умовах особливо важливими стають ефективна edge-обробка, стиснення даних, fault-tolerance та можливість роботи в умовах періодичної відсутності зв’язку з центральним сервером [1; 19].

Ефективна обробка даних в IoT вимагає від СУБД поєднання високої продуктивності запису, підтримки часових рядів, масштабованості, гнучкості та зручності інтеграції з існуючими IoT-платформами. Вибір відповідної системи (PostgreSQL чи MongoDB) стає ключовим фактором успішності всього рішення, оскільки безпосередньо впливає на вартість інфраструктури, швидкість розробки, надійність експлуатації та конкурентоспроможність IoT-проекту в реальних українських умовах [1; 19; 35]. Подальший детальний аналіз архітектури, можливостей та особливостей PostgreSQL та MongoDB дозволить чітко визначити оптимальні сценарії їх застосування в різних типах IoT-задач.

Сукупність розглянутих характеристик (Volume, Velocity, Variety, Veracity та Value) формує унікальний профіль даних в IoT-системах, який радикально відрізняється від даних традиційних корпоративних інформаційних систем. Якщо в класичних БД переважають відносно рідкі транзакції з чітко визначеною структурою, то в IoT ми маємо справу з постійним, високоінтенсивним потоком різнорідної інформації, що вимагає від СУБД принципово нових підходів до зберігання, індексації, обробки та аналізу.

Сучасні системи Інтернету речей висувають до СУБД комплекс жорстких і часто суперечливих вимог. З одного боку, необхідна висока швидкість запису (write throughput) і мінімальна латентність навіть за тисяч одночасних підключень. З іншого – потужні аналітичні можливості для виконання агрегатних запитів, виявлення аномалій, прогнозування та підтримки прийняття рішень у реальному часі. Крім того, система повинна забезпечувати горизонтальне масштабування, fault-tolerance, автоматичне стиснення даних, retention policies, інтеграцію з edge-обчисленнями та високий рівень безпеки. Традиційні реляційні СУБД, орієнтовані на вертикальне масштабування та жорстку схему, часто не справляються з таким навантаженням через значні накладні витрати на підтримку ACID-транзакцій під час масового запису та складнощі горизонтального масштабування. NoSQL-рішення, навпаки, пропонують високу продуктивність запису та гнучкість, але можуть поступатися в забезпеченні транзакційної цілісності, складності аналітики та довгостроковому зберіганні даних з можливістю точних агрегатів [3; 35].

Особливого значення набуває український контекст. Обмежена пропускна здатність мобільного та супутникового зв’язку в сільській місцевості, нестабільне електропостачання, вимоги до енергозбереження автономних пристроїв, а також необхідність відповідності національним нормам захисту персональних і критичних даних суттєво посилюють вимоги до СУБД. У таких умовах особливо важливими стають ефективна edge-обробка, стиснення даних, fault-tolerance та можливість роботи в умовах періодичної відсутності зв’язку з центральним сервером. Крім того, багато українських підприємств та агрогосподарств віддають перевагу on-premise або приватним хмарним рішенням, що також впливає на вибір технологій [1; 19].

Підсумовуючи вищезазначене, можна стверджувати, що ефективна обробка даних в IoT вимагає від СУБД поєднання таких ключових якостей:

- високої продуктивності запису великих обсягів часових рядів;
- підтримки гнучкої або гібридної моделі даних;
- потужних механізмів аналітики та агрегування в реальному часі;
- ефективного горизонтального масштабування та fault-tolerance;
- вбудованих засобів стиснення, retention policies та continuous aggregates;
- високого рівня безпеки та відповідності нормативним вимогам.

Саме через ці особливості вибір між PostgreSQL (з розширенням TimescaleDB) та MongoDB стає стратегічно важливим архітектурним рішенням. Кожна з цих систем пропонує власний набір сильних сторін, які відповідають різним типам IoT-задач. Подальший детальний аналіз архітектури, функціональних та нефункціональних характеристик, а також практичних прикладів застосування цих СУБД дозволить чітко визначити оптимальні сценарії їх використання в реальних IoT-проектах і сформулювати обґрунтовані рекомендації для розробників комп'ютерних систем.

Характеристика систем Інтернету речей та особливості обробки даних, розглянуті в даному підрозділі, створюють необхідну теоретичну основу для подальшого порівняльного аналізу PostgreSQL та MongoDB у контексті IoT-рішень.

1.2 Класифікація систем управління базами даних

Системи управління базами даних (СУБД) є фундаментальним елементом сучасних інформаційних систем і відіграють ключову роль у зберіганні, обробці та аналізі даних. Класифікація СУБД здійснюється за кількома ключовими критеріями, серед яких найважливішими вважаються модель даних, архітектура зберігання, спосіб масштабування, рівень підтримки ACID-властивостей, а також орієнтація на конкретні типи навантажень. У контексті систем Інтернету речей (IoT) особливого значення набуває поділ на реляційні (SQL) та нереляційні (NoSQL) СУБД, оскільки саме ці два класи домінують у проектах, пов'язаних з обробкою великих обсягів часових рядів, високошвидкісним записом даних і необхідністю горизонтального масштабування [4]. Реляційні системи забезпечують строгі гарантії цілісності даних, потужні засоби аналітики та повну підтримку транзакцій, тоді як NoSQL-системи орієнтовані насамперед на високу швидкість запису, гнучкість схеми та простоту горизонтального масштабування [21].

Класифікація СУБД за моделлю даних є найбільш поширеним і інформативним підходом. Вона дозволяє чітко зрозуміти, як саме система організовує дані, які операції з ними підтримує та в яких сценаріях проявляє свої найкращі якості. Основні класи СУБД за моделлю даних включають:

- Реляційні (SQL) СУБД – базуються на класичній реляційній моделі Е. Кодда, використовують табличну структуру даних, схеми, первинні та зовнішні ключі, а також мову структурованих запитів SQL. Вони забезпечують строгі гарантії ACID-властивостей і є ідеальними для задач, що вимагають складної аналітики, joins, агрегатів і транзакційної цілісності.

- Документо-орієнтовані NoSQL – зберігають дані у форматі документів (JSON, BSON), що дозволяє працювати з напівструктурованими та неструктурованими даними без жорсткої схеми. Цей клас особливо ефективний для IoT, де структура даних може динамічно змінюватися.
- Ключ-значення (Key-Value) – найпростіша модель, де дані зберігаються у вигляді пар «ключ–значення». Вони забезпечують максимальну швидкість доступу за ключем і використовуються переважно для кешування та швидкого зберігання простих даних.
- Колонкові (стовпцеві, Column-family) – оптимізовані для аналітичних запитів над величезними обсягами даних. Дані зберігаються за стовпцями, що дозволяє ефективно виконувати агрегатні операції та запити на великих наборах.
- Графові – призначені для моделювання складних зв'язків між об'єктами. Використовуються для аналізу соціальних мереж, логістичних зв'язків пристроїв та виявлення залежностей в IoT-екосистемах.

Сучасні СУБД часто поєднують риси кількох класів, утворюючи гібридні рішення (NewSQL, Multi-model). NewSQL-системи намагаються поєднати переваги реляційних баз (ACID, SQL) з горизонтальним масштабуванням NoSQL. Multi-model СУБД підтримують одночасно кілька моделей даних у рамках однієї системи, що особливо зручно для складних IoT-платформ [20].

У IoT-проектах переважна більшість практичних рішень базується саме на реляційних СУБД (PostgreSQL з розширенням TimescaleDB) та документо-орієнтованих NoSQL (MongoDB з підтримкою Time Series Collections). Саме ці дві системи найкраще задовольняють ключові вимоги IoT: високу швидкість запису, гнучкість схеми, масштабованість і потужну аналітику часових рядів [5; 20]. Інші класи (ключ-значення, колонкові, графові) використовуються переважно як допоміжні компоненти – для кешування, аналітики великих даних або моделювання зв'язків між пристроями.

Реляційні СУБД залишаються основою для задач, де потрібна висока цілісність даних, складна аналітика та підтримка бізнес-логіки. Документо-орієнтовані NoSQL, навпаки, домінують у сценаріях з високою швидкістю надходження різнорідних даних і частою зміною їх структури. Гібридні рішення дозволяють поєднувати сильні сторони обох підходів у рамках однієї платформи.

Класифікація СУБД за моделлю даних є ключовою для розуміння їх придатності до конкретних IoT-задач. Подальший детальний розгляд кожного класу дозволить чітко визначити сильні та слабкі сторони систем у контексті вимог Інтернету речей.

Реляційні (SQL) СУБД базуються на класичній реляційній моделі даних, запропонованій Едгаром Коддом у 1970 році. Вони організовують інформацію у вигляді двовимірних таблиць, пов'язаних між собою за допомогою первинних і

зовнішніх ключів. Основною перевагою реляційних систем є строга схема даних, яка забезпечує високий рівень цілісності, відсутність дублювання інформації та можливість виконання складних аналітичних запитів за допомогою стандартизованої мови SQL. У контексті IoT реляційні СУБД (зокрема PostgreSQL з розширенням TimescaleDB) ідеально підходять для задач, де потрібна точна аналітика, агрегування даних за довгі періоди, joins між різними сутностями (пристрої, користувачі, правила) та гарантована ACID-цілісність. Однак жорстка схема даних і вертикальне масштабування роблять їх менш ефективними при надзвичайно високій швидкості надходження різнорідних даних [4; 21].

Документо-орієнтовані NoSQL-системи (MongoDB, CouchDB, Firebase) зберігають дані у форматі документів – JSON або BSON. Кожен документ є самодостатнім і може мати різну структуру, що дозволяє працювати з напівструктурованими та неструктурованими даними без попереднього визначення схеми. У IoT-проектах ця гнучкість є ключовою перевагою: нові типи сенсорів або параметри можна додавати динамічно, без зупинки системи та виконання міграцій. Документо-орієнтовані СУБД забезпечують високу швидкість запису, простоту горизонтального масштабування та вбудовані механізми роботи з часовими рядами (Time Series Collections у MongoDB). Недоліком є слабша підтримка складних транзакцій і joins порівняно з реляційними системами [5; 20].

Ключ-значення (Key-Value) СУБД (Redis, Memcached, DynamoDB) – найпростіша модель зберігання даних у вигляді пар «ключ–значення». Вони забезпечують максимальну швидкість доступу за ключем і використовуються переважно як кешуючі шари або для зберігання простих оперативних даних (стани пристроїв, сесії, лічильники). У IoT такі системи часто застосовують на edge-рівні для тимчасового зберігання даних перед відправкою в основну базу. Головний недолік – обмежена аналітика та відсутність підтримки складних запитів [21].

Колонкові (стовпцеві, Column-family) СУБД (Cassandra, HBase, ClickHouse) зберігають дані не по рядках, а по стовпцях. Це робить їх надзвичайно ефективними для аналітичних запитів над величезними обсягами історичних даних. У IoT вони застосовуються для офлайн-аналітики великих датасетів (наприклад, обробка історичних показників тисяч пристроїв). Недоліком є відносно повільний запис і складність роботи з транзакціями [3].

Графові СУБД (Neo4j, ArangoDB, OrientDB) спеціалізуються на моделюванні складних зв’язків між об’єктами. У IoT вони корисні для аналізу топології мережі пристроїв, виявлення залежностей, моделювання соціальних зв’язків між користувачами та пристроями або оптимізації логістичних

маршрутів. Головним недоліком є низька швидкість масового запису даних, тому графові бази зазвичай використовуються як допоміжні компоненти [21].

Сучасний розвиток СУБД призвів до появи гібридних і NewSQL-рішень. NewSQL-системи (CockroachDB, Google Spanner, YugabyteDB) намагаються поєднати переваги реляційних баз (повна ACID, SQL) з горизонтальним масштабуванням NoSQL. Multi-model СУБД (ArangoDB, Azure Cosmos DB) підтримують одночасно кілька моделей даних (документи, графи, ключ-значення) в рамках однієї системи, що особливо зручно для складних IoT-платформ, де потрібні різні типи обробки даних. Такі рішення дозволяють уникнути використання кількох окремих баз даних і спростити архітектуру системи [20].

У IoT-проектах переважна більшість реальних рішень базується саме на реляційних СУБД (PostgreSQL з TimescaleDB) та документо-орієнтованих NoSQL (MongoDB). Реляційні системи домінують там, де потрібна висока аналітика, точність даних і бізнес-логіка. Документо-орієнтовані – там, де головним пріоритетом є швидкість запису, гнучкість і горизонтальне масштабування. Інші класи СУБД (ключ-значення, колонкові, графові) використовуються переважно як допоміжні компоненти в складних архітектурах [5; 20].

Вибір конкретного класу СУБД для IoT-систем залежить від пріоритетів конкретного проекту: для задач, що вимагають складної аналітики та гарантій цілісності даних, доцільно використовувати реляційні системи, тоді як для високонавантажених сценаріїв з гнучкими та динамічними даними перевагу віддають NoSQL-рішенням [3; 21]. Подальший порівняльний аналіз PostgreSQL та MongoDB дозволить визначити оптимальні сфери застосування кожної з цих систем у реальних IoT-рішеннях.

Реляційні (SQL) СУБД базуються на класичній реляційній моделі даних, запропонованій Едгаром Коддом у 1970 році. Вони організовують інформацію у вигляді двовимірних таблиць, пов’язаних між собою за допомогою первинних і зовнішніх ключів. Основною перевагою реляційних систем є строга схема даних, яка забезпечує високий рівень цілісності, відсутність дублювання інформації та можливість виконання складних аналітичних запитів за допомогою стандартизованої мови SQL. У контексті IoT реляційні СУБД (зокрема PostgreSQL з розширенням TimescaleDB) ідеально підходять для задач, де потрібна точна аналітика, агрегування даних за довгі періоди, joins між різними сутностями (пристрої, користувачі, правила) та гарантована ACID-цілісність. Однак жорстка схема даних і вертикальне масштабування роблять їх менш ефективними при надзвичайно високій швидкості надходження різнорідних даних [4; 21].

Документо-орієнтовані NoSQL-системи (MongoDB, CouchDB, Firebase) зберігають дані у форматі документів – JSON або BSON. Кожен документ є самодостатнім і може мати різну структуру, що дозволяє працювати з

напівструктурованими та неструктурованими даними без попереднього визначення схеми. У IoT-проектах ця гнучкість є ключовою перевагою: нові типи сенсорів або параметри можна додавати динамічно, без зупинки системи та виконання міграцій. Документо-орієнтовані СУБД забезпечують високу швидкість запису, простоту горизонтального масштабування та вбудовані механізми роботи з часовими рядами (Time Series Collections у MongoDB). Недоліком є слабша підтримка складних транзакцій і joins порівняно з реляційними системами [5; 20].

Ключ-значення (Key-Value) СУБД (Redis, Memcached, DynamoDB) – найпростіша модель зберігання даних у вигляді пар «ключ–значення». Вони забезпечують максимальну швидкість доступу за ключем і використовуються переважно як кешуючі шари або для зберігання простих оперативних даних (стани пристроїв, сесії, лічильники). У IoT такі системи часто застосовують на edge-рівні для тимчасового зберігання даних перед відправкою в основну базу. Головний недолік – обмежена аналітика та відсутність підтримки складних запитів [21].

Колонкові (стовпцеві, Column-family) СУБД (Cassandra, HBase, ClickHouse) зберігають дані не по рядках, а по стовпцях. Це робить їх надзвичайно ефективними для аналітичних запитів над величезними обсягами історичних даних. У IoT вони застосовуються для офлайн-аналітики великих датасетів (наприклад, обробка історичних показників тисяч пристроїв). Недоліком є відносно повільний запис і складність роботи з транзакціями [3].

Графові СУБД (Neo4j, ArangoDB, OrientDB) спеціалізуються на моделюванні складних зв’язків між об’єктами. У IoT вони корисні для аналізу топології мережі пристроїв, виявлення залежностей, моделювання соціальних зв’язків між користувачами та пристроями або оптимізації логістичних маршрутів. Головним недоліком є низька швидкість масового запису даних, тому графові бази зазвичай використовуються як допоміжні компоненти [21].

Сучасний розвиток СУБД призвів до появи гібридних і NewSQL-рішень. NewSQL-системи (CockroachDB, Google Spanner, YugabyteDB) намагаються поєднати переваги реляційних баз (повна ACID, SQL) з горизонтальним масштабуванням NoSQL. Multi-model СУБД (ArangoDB, Azure Cosmos DB) підтримують одночасно кілька моделей даних (документи, графи, ключ-значення) в рамках однієї системи, що особливо зручно для складних IoT-платформ, де потрібні різні типи обробки даних. Такі рішення дозволяють уникнути використання кількох окремих баз даних і спростити архітектуру системи [20].

У IoT-проектах переважна більшість реальних рішень базується саме на реляційних СУБД (PostgreSQL з TimescaleDB) та документо-орієнтованих NoSQL (MongoDB). Реляційні системи домінують там, де потрібна висока аналітика, точність даних і бізнес-логіка. Документо-орієнтовані – там, де головним

пріоритетом є швидкість запису, гнучкість і горизонтальне масштабування. Інші класи СУБД (ключ-значення, колонкові, графові) використовуються переважно як допоміжні компоненти в складних архітектурах [5; 20].

Класифікація СУБД за моделлю даних є ключовою для розуміння їх придатності до конкретних IoT-задач. Подальший детальний розгляд кожного класу дозволить чітко визначити сильні та слабкі сторони систем у контексті вимог Інтернету речей.

Сучасні тенденції розвитку СУБД чітко демонструють перехід від «чистої» класифікації до гібридних і мультимодельних рішень. NewSQL-системи (CockroachDB, YugabyteDB, Google Spanner) намагаються поєднати сильні сторони реляційних баз даних (повна ACID-цілісність, стандарт SQL, транзакційна надійність) з горизонтальним масштабуванням і високою доступністю, характерними для NoSQL. Такі системи особливо актуальні для IoT-проектів середнього та великого масштабу, де одночасно потрібні гарантії цілісності даних і можливість лінійного зростання навантаження. Multi-model СУБД (ArangoDB, Azure Cosmos DB, OrientDB) дозволяють працювати з кількома моделями даних (документи, графи, ключ-значення) в рамках однієї інсталяції, що значно спрощує архітектуру складних IoT-платформ і зменшує кількість використовуваних технологій [20].

У контексті систем Інтернету речей домінують саме два класи СУБД – реляційні (PostgreSQL з розширенням TimescaleDB) та документо-орієнтовані NoSQL (MongoDB). Це пояснюється тим, що більшість IoT-задач вимагає одночасного вирішення двох суперечливих вимог: високої швидкості запису великих обсягів часових рядів і потужної аналітики з гарантіями цілісності даних. Реляційні системи забезпечують другу групу вимог, тоді як документо-орієнтовані – першу. Інші класи СУБД (ключ-значення, колонкові, графові) виконують допоміжну роль: Redis використовується для кешування оперативних станів пристроїв, Cassandra або ClickHouse – для офлайн-аналітики історичних даних великих обсягів, а графові бази – для моделювання топології мережі пристроїв і виявлення складних залежностей [5; 21].

Вибір конкретного класу СУБД для IoT-систем залежить від пріоритетів проекту. Якщо головним завданням є складна аналітика, агрегатні розрахунки, бізнес-логіка, joins між різними сутностями та гарантії ACID-цілісності – доцільно використовувати реляційні системи. Якщо пріоритетом є висока швидкість запису, гнучкість схеми даних, простота горизонтального масштабування та швидка адаптація до нових типів пристроїв – перевагу віддають NoSQL-рішенням, передусім документо-орієнтованим. У багатьох сучасних проектах застосовується гібридний підхід, коли MongoDB використовується для оперативного збору telemetry, а PostgreSQL з TimescaleDB – для аналітичного сховища, звітності та довгострокового зберігання [3; 21].

Класифікація систем управління базами даних за моделлю даних є ключовим інструментом для розуміння їх придатності до конкретних IoT-задач. Реляційні та документо-орієнтовані СУБД на сьогодні є основними технологіями, що використовуються в реальних IoT-проектах. Їх детальне порівняння, яке буде проведено в наступних підрозділах, дозволить чітко визначити сильні та слабкі сторони PostgreSQL та MongoDB і сформулювати практичні рекомендації щодо вибору оптимальної системи для кожного типу IoT-рішення.

1.3 Специфічні вимоги IoT до СУБД (масштабованість, продуктивність, обробка часових рядів, надійність)

Системи Інтернету речей (IoT) висувають до систем управління базами даних (СУБД) набагато жорсткіші, комплексніші та часто суперечливі вимоги порівняно з традиційними корпоративними інформаційними системами. Якщо в класичних додатках дані надходять відносно рідко, мають чітку структуру і обробляються в офлайн-режимі, то в IoT-платформах ми маємо справу з постійним, високошвидкісним потоком інформації від тисяч і мільйонів розподілених датчиків. Цей потік характеризується необхідністю обробки в реальному часі, обмеженими ресурсами edge-пристроїв, нестабільністю каналів зв’язку та потенційними збоями обладнання. У таких умовах СУБД повинна виконувати не лише функцію простого зберігання даних, а й забезпечувати ефективну роботу в умовах екстремальних навантажень, гарантуючи при цьому високу доступність, надійність і безпеку всього IoT-рішення [3].

Специфічні вимоги IoT до СУБД можна систематизувати у вигляді чотирьох ключових, взаємопов’язаних аспектів, які визначають придатність тієї чи іншої системи для реальних IoT-проектів:

1. Масштабованість (scalability) – можливість горизонтального розширення системи без суттєвого зниження продуктивності при експоненційному зростанні обсягу даних і кількості підключених пристроїв.
2. Продуктивність (performance) – висока швидкість операцій запису та читання даних, мінімальна латентність запитів навіть за тисяч одночасних підключень.
3. Обробка часових рядів (time-series data handling) – ефективне зберігання, стиснення, індексація та швидкий аналіз послідовних даних з точними часовим мітками.
4. Надійність (reliability) – забезпечення fault-tolerance, реплікації, відновлення після збоїв, підтримка ACID-властивостей (або їх аналогів) у розподіленому середовищі, а також захист даних і відповідність вимогам безпеки.

Кожен із цих аспектів має критичне значення для успішної експлуатації IoT-платформ і безпосередньо впливає на вибір між PostgreSQL (з TimescaleDB) та MongoDB.

Масштабованість є однією з найважливіших характеристик для IoT-систем. На відміну від традиційних корпоративних додатків, де кількість користувачів і обсяг даних зростають лінійно або помірно, в IoT кількість підключених пристроїв може зростати експоненційно. Обсяг даних при цьому переходить від гігабайт до петабайт за відносно короткий період. СУБД повинна підтримувати горизонтальне масштабування (sharding), автоматичне балансування навантаження між вузлами кластера, ефективну реплікацію даних між дата-центрами та можливість динамічного додавання нових серверів без зупинки сервісу.

У реальних українських IoT-проектах (смарт-ферми, промислові комплекси, системи моніторингу комунальних мереж, логістичні платформи) обмежена пропускна здатність каналів зв'язку та нестабільний інтернет у віддалених регіонах вимагають, щоб значна частина обробки відбувалася на edge-рівні (локальні шлюзи або edge-сервери). При цьому центральна СУБД повинна забезпечувати seamless інтеграцію edge-cloud архітектури: автоматичну синхронізацію даних при відновленні зв'язку, підтримку offline-режиму та ефективну реплікацію. Недостатня масштабованість призводить до швидкого вичерпання ресурсів одного сервера, зростання вартості інфраструктури, погіршення продуктивності та, як наслідок, втрати даних або значних затримок у прийнятті рішень [14].

Горизонтальне масштабування реалізується по-різному. У документо-орієнтованих NoSQL-системах (MongoDB) воно вбудоване «з коробки» через механізм sharding і Replica Sets, що дозволяє легко додавати нові вузли та лінійно збільшувати пропускну здатність. У реляційних СУБД (PostgreSQL) горизонтальне масштабування вимагає додаткових розширень (Citus) або спеціальних кластерних рішень (Patroni), що підвищує складність адміністрування, але дає кращий контроль над конфігурацією. Умови українських проектів часто диктують необхідність комбінованого підходу: edge-обробка на легких СУБД і потужний центральний кластер для агрегації та аналітики.

Масштабованість є фундаментальною вимогою, яка визначає здатність IoT-системи зростати без перебудови архітектури та значних додаткових витрат. Її відсутність або недостатня реалізація стає однією з головних причин невдач IoT-проектів на етапі промислової експлуатації.

Масштабованість, як було зазначено вище, є фундаментальною вимогою, але вона тісно пов'язана з наступним ключовим аспектом – продуктивністю (performance). Саме продуктивність визначає, наскільки ефективно СУБД здатна обробляти реальні IoT-потоки даних у реальному часі.

Продуктивність СУБД в IoT-системах вимірюється здатністю обробляти високошвидкісні потоки даних з мінімальною латентністю. Середня IoT-платформа може генерувати від десятків тисяч до сотень тисяч записів за секунду. Для критичних застосувань (промисловий моніторинг, системи безпеки, автономний транспорт) критичним є забезпечення write throughput на рівні 50 000–200 000+ операцій вставки за секунду при одночасній можливості виконання аналітичних запитів без блокування основного потоку запису. Латентність запису повинна бути в межах 10–50 мс, а латентність читання простих даних – не перевищувати 100 мс навіть за пікових навантажень [5].

До ключових показників продуктивності в IoT належать:

- Write throughput – кількість успішних операцій вставки за секунду. У IoT цей показник часто є визначальним, оскільки дані надходять постійно і не можуть накопичуватися в черзі.
- Read latency – час відповіді на запити читання. Для дашбордів і систем моніторингу в реальному часі критична низька латентність навіть при складних агрегатних запитах.
- Query performance – швидкість виконання аналітичних запитів (середнє, максимум, мінімум, виявлення аномалій) без суттєвого впливу на операції запису.
- Resource efficiency – споживання CPU, RAM і дискового простору під час обробки високих навантажень.

Сучасні СУБД для досягнення високої продуктивності використовують низку технологічних рішень: in-memory буферизацію, batch-вставки, оптимізовану індексацію, паралельну обробку запитів і спеціальні механізми стиснення даних у реальному часі. Однак традиційні реляційні СУБД часто стикаються з проблемами через overhead підтримки ACID-транзакцій і блокування таблиць під час масового запису. NoSQL-системи, навпаки, орієнтовані на максимальну швидкість запису завдяки спрощеній моделі даних і відсутності жорстких транзакційних обмежень, але можуть поступатися в продуктивності складних аналітичних запитів [5].

У реальних українських IoT-проектах (смарт-ферми, промислові комплекси, системи моніторингу комунальних мереж) продуктивність набуває особливого значення через обмежену пропускну здатність каналів зв’язку та необхідність обробки даних на edge-рівні. Якщо СУБД не справляється з піковими навантаженнями, система починає втрачати дані, затримувати реакцію на події або потребує значного апгрейду обладнання, що суттєво збільшує вартість проєкту.

Продуктивність є критичним фактором, який безпосередньо впливає на можливість системи працювати в реальному часі та забезпечувати оперативне

прийняття рішень. Без достатньої продуктивності навіть ідеально масштабована СУБД стає непридатною для більшості IoT-застосувань.

Наступним ключовим аспектом специфічних вимог IoT до СУБД є обробка часових рядів (time-series data handling). Це найбільш характерна і водночас найбільш специфічна вимога IoT-систем. Дані від датчиків надходять у вигляді послідовних послідовностей з точними часовими мітками, часто з високою частотою – від 1 до 100 записів за секунду на один пристрій. Такі дані вимагають ефективного стиснення (time-series compression), автоматичного створення retention policies, швидкого виконання агрегатних запитів (середнє, максимум, мінімум, відхилення за період), виявлення аномалій, прогнозування та візуалізації в реальному часі.

Традиційні реляційні таблиці погано справляються з таким навантаженням через значний overhead зберігання та індексації. Спеціалізовані розширення (TimescaleDB для PostgreSQL) та вбудовані механізми (Time Series Collections у MongoDB) дозволяють досягати в десятки разів кращої продуктивності та меншого обсягу зберігання. Без оптимізованої підтримки обробки часових рядів неможливо забезпечити реальний аналіз даних для прийняття оперативних рішень у промисловості, сільському господарстві, енергетиці та інших галузях [16].

Обробка часових рядів є, безперечно, найбільш специфічною і водночас найскладнішою вимогою IoT до СУБД. Дані від датчиків надходять у вигляді безперервних послідовностей з точними часовими мітками, часто з високою частотою – від одного до ста записів за секунду на один пристрій. У промислових системах, смарт-сільському господарстві чи системах моніторингу розумних міст така частота може сягати тисяч записів за секунду по всій платформі. Ці дані вимагають не просто зберігання, а ефективного стиснення (time-series compression), автоматичного створення retention policies, швидкого виконання агрегатних запитів (середнє, максимум, мінімум, стандартне відхилення, тренди за будь-який період), виявлення аномалій у реальному часі та підтримки прогнозної аналітики [16].

Традиційні реляційні таблиці погано справляються з таким навантаженням через значний overhead зберігання індексів, дублювання даних і повільну роботу з великими обсягами часових рядів. Саме тому з’явилися спеціалізовані розширення та механізми: TimescaleDB для PostgreSQL перетворює звичайні таблиці на hypertables з автоматичним партиціонуванням за часом, стисненням старих чанків і continuous aggregates; MongoDB пропонує вбудовані Time Series Collections з автоматичним стисненням і оптимізованою індексацією. Без таких спеціалізованих інструментів неможливо забезпечити реальний аналіз даних для прийняття оперативних рішень у промисловості, енергетиці, сільському господарстві та інших галузях. Відсутність ефективної обробки часових рядів призводить до втрати цінної інформації, зростання витрат на зберігання та

неможливості побудови сучасних аналітичних дашбордів і систем прогнозування [16].

Четвертим ключовим аспектом специфічних вимог IoT до СУБД є надійність (reliability). Вона включає цілий комплекс взаємопов'язаних характеристик: fault-tolerance (стійкість до відмов), data durability (гарантоване збереження даних), consistency models (моделі узгодженості даних) та security (безпека). СУБД повинна продовжувати роботу навіть за виходу з ладу окремих вузлів, автоматично відновлювати репліки, гарантувати, що жоден запис не буде втрачено, і забезпечувати швидке відновлення після збоїв. У розподілених IoT-середовищах часто застосовують моделі eventual consistency для підвищення продуктивності, але для критичних застосувань (управління енергосистемами, промисловою безпекою, медичним моніторингом) потрібні сильніші гарантії ACID або їх аналоги.

Надійність також передбачає захист від кіберзагроз. IoT-системи є особливо вразливими через велику кількість пристроїв з обмеженими можливостями захисту. СУБД повинна підтримувати сучасні механізми шифрування даних у спокої та під час передачі, контроль доступу на рівні рядків (Row Level Security), детальний аудит подій, ролеву модель доступу та відповідність вимогам національного законодавства щодо обробки персональних і критичних даних. Умови експлуатації в Україні – перепади напруги, нестабільний інтернет у сільській місцевості, можливі фізичні пошкодження обладнання – роблять ці вимоги особливо актуальними. Система повинна працювати стабільно навіть при періодичній відсутності зв'язку з центральним сервером і автоматично синхронізувати дані при його відновленні [35].

Надійність СУБД в IoT-системах – це не лише технічна характеристика, а й запорука безперервності бізнес-процесів, безпеки людей і збереження критичної інфраструктури. Недостатня надійність призводить до втрати даних, простою обладнання, фінансових збитків і, в критичних випадках, до загрози безпеці.

Специфічні вимоги IoT до СУБД значно відрізняються від стандартних сценаріїв використання баз даних у корпоративних системах. Якщо в традиційних додатках пріоритетом є зручність розробки та підтримка бізнес-логіки, то в IoT на перший план виходять масштабованість, продуктивність, ефективна робота з часовими рядами та висока надійність в умовах розподіленого та нестабільного середовища. Масштабованість забезпечує майбутнє зростання системи, продуктивність дозволяє працювати в реальному часі, ефективна обробка часових рядів є основою всієї аналітики, а надійність гарантує безперервність і безпеку всього рішення. Саме через ці особливості вибір між PostgreSQL та MongoDB стає стратегічним архітектурним рішенням, що безпосередньо впливає на архітектуру, вартість інфраструктури, швидкість розробки та загальну ефективність усього IoT-проекту в різних галузях.

Надійність СУБД в IoT-системах є комплексною характеристикою, яка включає кілька взаємопов'язаних аспектів: fault-tolerance (стійкість до відмов), data durability (гарантоване збереження даних), consistency models (моделі узгодженості даних) та security (безпека). Система повинна продовжувати стабільну роботу навіть за виходу з ладу окремих вузлів, автоматично відновлювати репліки, гарантувати, що жоден запис не буде втрачено, і забезпечувати швидке відновлення після збоїв. У розподілених IoT-середовищах часто застосовують моделі eventual consistency для підвищення продуктивності, але для критичних застосувань (управління енергосистемами, промисловою безпекою, медичним моніторингом) потрібні сильніші гарантії ACID або їх аналоги.

Надійність також передбачає захист від кіберзагроз. IoT-системи є особливо вразливими через велику кількість пристроїв з обмеженими можливостями захисту. СУБД повинна підтримувати сучасні механізми шифрування даних у спокої та під час передачі, контроль доступу на рівні рядків (Row Level Security), детальний аудит подій, ролеву модель доступу та відповідність вимогам національного законодавства щодо обробки персональних і критичних даних. Умови експлуатації в Україні – перепади напруги, нестабільний інтернет у сільській місцевості, можливі фізичні пошкодження обладнання – роблять ці вимоги особливо актуальними. Система повинна працювати стабільно навіть при періодичній відсутності зв'язку з центральним сервером і автоматично синхронізувати дані при його відновленні [35].

Підсумовуючи чотири ключові специфічні вимоги IoT до СУБД, можна стверджувати, що вони значно відрізняються від стандартних сценаріїв використання баз даних у корпоративних системах. Масштабованість забезпечує можливість майбутнього зростання системи без кардинальної перебудови архітектури. Продуктивність дозволяє обробляти високошвидкісні потоки даних у реальному часі. Ефективна обробка часових рядів є основою всієї аналітики, моніторингу та прогнозування. Надійність гарантує безперервність роботи, збереження даних і захист від кіберзагроз навіть в умовах нестабільного середовища.

Ці чотири вимоги тісно взаємопов'язані та часто вступають у конфлікт. Наприклад, забезпечення повної ACID-цілісності (надійність) може знижувати продуктивність запису. Горизонтальне масштабування (масштабованість) ускладнює підтримку сильної узгодженості даних. Гнучкість схеми для швидкого запису (продуктивність) може ускладнювати складну аналітику. Саме тому вибір СУБД для IoT-проекту є стратегічним рішенням, яке вимагає ретельного аналізу пріоритетів конкретної задачі.

У реальних українських IoT-проектах (смарт-сільське господарство, промислові комплекси, системи моніторингу комунальних мереж, логістика,

розумні міста) ці вимоги набувають особливої гостроти через обмежену пропускну здатність каналів зв’язку, нестабільне електропостачання, географічну розподіленість об’єктів і необхідність відповідності національним нормам захисту даних. У таких умовах особливо важливими стають ефективна edge-обробка, автоматичне стиснення даних, fault-tolerance та можливість роботи в умовах періодичної відсутності зв’язку.

Специфічні вимоги IoT до СУБД значно відрізняються від стандартних сценаріїв використання баз даних. Масштабованість забезпечує майбутнє зростання системи, продуктивність дозволяє працювати в реальному часі, ефективна обробка часових рядів є основою аналітики, а надійність гарантує безперервність і безпеку всього рішення. Саме через ці особливості вибір між PostgreSQL та MongoDB стає одним із ключових архітектурних рішень, що безпосередньо впливає на архітектуру, вартість інфраструктури, швидкість розробки та загальну ефективність IoT-проекту в різних галузях економіки.

Подальший порівняльний аналіз цих двох систем у наступних розділах роботи дозволить чітко визначити, яка з них краще відповідає конкретним вимогам різних типів IoT-задач і сформулювати практичні рекомендації для розробників.

2 ПОРІВНЯЛЬНИЙ АНАЛІЗ POSTGRESQL ТА MONGODB

2.1 Архітектура, основні можливості та особливості PostgreSQL

PostgreSQL є однією з найпотужніших, найбільш розвинених і стабільних відкритих систем управління базами даних у світі. Вона належить до класу об'єктно-реляційних СУБД і поєднує в собі класичну реляційну модель даних із сучасними розширеннями, що робить її універсальним інструментом для найрізноманітніших задач, включаючи високонавантажені системи Інтернету речей [4]. PostgreSQL розробляється спільнотою з 1986 року (спочатку як POSTGRES у Каліфорнійському університеті в Берклі) і сьогодні підтримується глобальною командою розробників PostgreSQL Global Development Group. Завдяки своїй відкритості, надійності та розширюваності PostgreSQL регулярно посідає перші місця в рейтингах DB-Engines серед реляційних СУБД і широко застосовується в IoT-платформах, де потрібна одночасна висока швидкість запису даних і глибока аналітика.

Архітектура PostgreSQL побудована за принципом клієнт-серверної моделі та відрізняється високою ефективністю використання апаратних ресурсів. Центральним елементом системи є процес postmaster (або postgres master process), який виконує роль диспетчера підключень. Він відповідає за прийом нових з'єднань від клієнтів, аутентифікацію, авторизацію та створення окремих backend-процесів (worker processes) для кожної клієнтської сесії. Така багатопроцесна архітектура забезпечує ізоляцію сесій, запобігає впливу одного клієнта на інший і дозволяє ефективно використовувати багатоядерні процесори та великі обсяги оперативної пам'яті.

Важливою складовою архітектури є Shared Memory Area (спільна пам'ять), яка містить буферний кеш (buffer cache), блоки блокувань (lock manager), кеш системного каталогу (catalog cache) та іншу службову інформацію. Завдяки спільній пам'яті всі backend-процеси можуть швидко обмінюватися даними без зайвих звернень до диску, що критично важливо для IoT-систем, де одночасно надходить тисячі записів від датчиків.

Іншим ключовим елементом є механізм Write-Ahead Logging (WAL). Перед тим, як будь-яка зміна даних буде записана на диск, вона спочатку фіксується у WAL-файлах. Це гарантує повну стійкість даних (durability) навіть у разі раптового збою живлення або апаратного збою сервера. Для IoT-платформ, де втрата навіть одного пакету даних може мати серйозні наслідки (наприклад, у промисловому моніторингу), WAL забезпечує максимальну надійність.

Постійне зберігання даних здійснюється у файлах таблиць, індексів та TOAST (The Oversized-Attribute Storage Technique) для великих об'єктів. PostgreSQL автоматично керує партиціонуванням, vacuuming (очищенням від

«мертвих» версій рядків) та autovacuum-процесами, що дозволяє підтримувати стабільну продуктивність навіть при постійному потоці IoT-даних.

Для підвищення продуктивності в IoT-сценаріях PostgreSQL пропонує потужну систему extensions. Найважливішим серед них є TimescaleDB – спеціальне розширення, яке перетворює звичайні таблиці на hypertables, оптимізує зберігання часових рядів, застосовує автоматичне стиснення даних, continuous aggregates та retention policies. Саме завдяки TimescaleDB PostgreSQL стає одним із найкращих рішень для обробки telemetry-даних від тисяч і мільйонів IoT-пристроїв [4].

Основні можливості PostgreSQL, що роблять її особливо привабливою для систем Інтернету речей, включають:

- повну підтримку стандарту ACID (Atomicity, Consistency, Isolation, Durability) на рівні окремих транзакцій і навіть багатотабличних операцій;
- вбудовану підтримку JSONB – бінарного формату JSON, який дозволяє зберігати напівструктуровані IoT-дані з високою швидкістю індексації та пошуку;
- розвинену систему індексацій: B-tree, GiST, GIN, BRIN, Hash, Bloom filters та інші, що особливо ефективні для часових рядів і геопросторових даних;
- паралельне виконання запитів (Parallel Query) та паралельне створення індексів;
- вбудовану логічну та фізичну реплікацію (streaming replication, logical replication), що дозволяє створювати високо доступні кластери;
- підтримку stored procedures, тригерів, функцій та розширених типів даних (включаючи масиви, hstore, PostGIS для геоданих);
- механізм Row Level Security (RLS) для тонкого контролю доступу до рядків – критично важливий для IoT-систем з різними рівнями доступу користувачів;
- можливість створення materialized views та continuous aggregates (у TimescaleDB) для швидкого доступу до попередньо обчислених агрегатів telemetry-даних;
- повну сумісність із мовою SQL (ANSI SQL:2016) та розширеними можливостями (window functions, recursive CTE, lateral joins тощо).

Завдяки цим можливостям PostgreSQL може ефективно працювати як у невеликих edge-пристроях, так і в масштабних хмарних IoT-платформах, таких як ThingsBoard, де PostgreSQL використовується для зберігання метаданих пристроїв, а TimescaleDB – для telemetry.

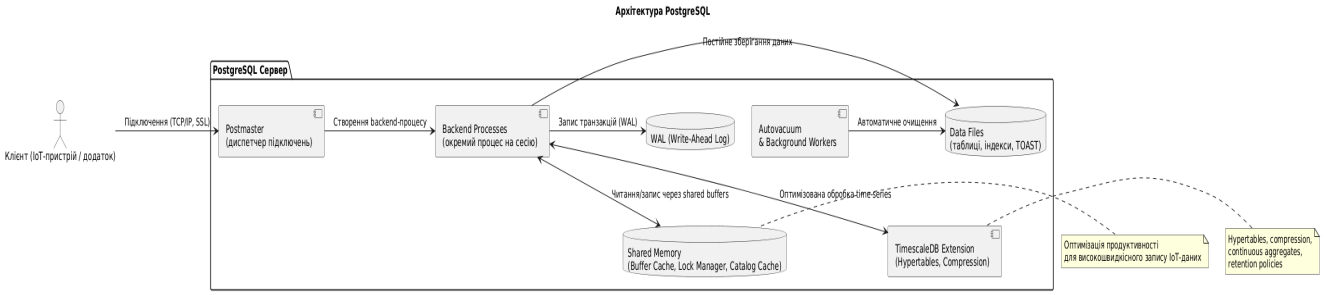

Рисунок 2.1 – Архітектура PostgreSQL

Архітектура PostgreSQL поєднує класичні принципи реляційних баз даних із сучасними технологіями розширюваності та оптимізації для роботи з великими обсягами часових рядів. Вона забезпечує високу надійність, повну транзакційну цілісність і потужні аналітичні можливості, що робить її ідеальним вибором для IoT-рішень, де одночасно потрібні швидкість запису та глибина аналізу даних. Подальше детальне вивчення окремих механізмів (наприклад, роботи WAL, hypertables у TimescaleDB та паралельних запитів) дозволяє повністю зрозуміти, чому PostgreSQL залишається одним із найкращих інструментів для сучасних IoT-платформ.

Особливої уваги в контексті систем Інтернету речей заслуговує розширення TimescaleDB, яке перетворює PostgreSQL на одну з найефективніших платформ для роботи з часовими рядами. TimescaleDB вводить поняття hypertable – це прозора для користувача партиціонована структура таблиці, яка автоматично розбивається на чанки за часовим інтервалом (наприклад, по годинах, днях або тижнях). Кожен чанк зберігається як звичайна таблиця PostgreSQL, але система оптимізує розміщення даних на диску, індексацію та стиснення. Завдяки цьому вдається досягти в 10–50 разів вищої продуктивності запису та читання порівняно зі звичайними таблицями при обробці мільйонів записів за секунду [4].

TimescaleDB також реалізує механізм автоматичного стиснення даних (compression). Після досягнення певного віку чанк переводиться в стиснутий формат, що зменшує обсяг зберігання в 5–10 разів без втрати швидкості доступу. Для IoT-систем, де дані від датчиків накопичуються роками, це означає суттєве зниження витрат на дискову підсистему та хмарне зберігання. Крім того, розширення підтримує continuous aggregates – матеріалізовані агрегатні представлення, які автоматично оновлюються в реальному часі. Це дозволяє миттєво отримувати середні значення, максимуми, мінімуми, відхилення та інші статистичні показники за будь-який період без виконання важких агрегатних запитів над мільярдами рядків.

Retention policies у TimescaleDB дають змогу автоматично видаляти або переміщувати застарілі дані на холодне зберігання, що є критичним для дотримання вимог до обсягу даних і нормативів зберігання в промислових IoT-

системах. Усі ці можливості доступні через стандартні SQL-команди, що значно спрощує розробку та підтримку IoT-платформ.

Ще однією сильною стороною PostgreSQL є розвинена система індексації. Для IoT-даних особливо ефективні BRIN-індекси (Block Range Indexes), які займають мінімальний обсяг пам'яті та оптимально працюють з даними, відсортованими за часом. GiST та GIN індекси дозволяють ефективно працювати з JSONB-полями та геопросторовими даними, що актуально для систем трекінгу та моніторингу навколишнього середовища. Паралельне виконання запитів (Parallel Query) дає змогу використовувати всі доступні ядра процесора для складних аналітичних завдань, таких як виявлення аномалій у потоках telemetry.

Механізм реплікації в PostgreSQL також повністю відповідає вимогам високої доступності IoT-систем. Streaming replication забезпечує синхронне або асинхронне копіювання WAL-логів на резервні сервери, а logical replication дозволяє вибірково реплікувати окремі таблиці або навіть рядки. Завдяки цьому можна будувати георозподілені кластери, де частина обробки відбувається ближче до джерела даних (edge), а аналітика – у центральному дата-центрі.

З точки зору безпеки PostgreSQL пропонує повноцінний набір інструментів, необхідних для IoT: ролеву модель доступу, Row Level Security (RLS), аудиту подій, шифрування з'єднань (SSL/TLS), шифрування даних на рівні стовпців і підтримку Kerberos. Для IoT-платформ, що обробляють чутливі дані (медичні показники, промислові параметри, персональні дані), ці можливості дозволяють відповідати вимогам GDPR, Закону України про захист персональних даних та галузевих стандартів кібербезпеки.

PostgreSQL повністю сумісний із сучасними хмарними платформами (AWS RDS, Google Cloud SQL, Azure Database for PostgreSQL, Supabase, Timescale Cloud), що дозволяє легко розгортати IoT-рішення як у власній інфраструктурі, так і в хмарі. Підтримка Kubernetes-операторів (наприклад, Zalando Postgres Operator або Timescale Operator) спрощує управління кластерами в контейнеризованому середовищі.

Архітектура та можливості PostgreSQL роблять її потужним і універсальним інструментом для систем Інтернету речей. Вона поєднує класичну реляційну надійність, транзакційну цілісність і потужну SQL-аналітику з сучасними розширеннями, що оптимізують роботу з високошвидкісними часовими рядами. Завдяки TimescaleDB, JSONB, розширеній індексації та високій масштабованості PostgreSQL ефективно вирішує ключові завдання IoT: швидкий запис даних від тисяч пристроїв, реальний час аналітики, довгострокове зберігання та забезпечення високої доступності й безпеки. Ці якості роблять PostgreSQL конкурентоспроможною альтернативою NoSQL-системам у сценаріях, де важливі не тільки швидкість, а й точність і глибина обробки даних.

Подальший порівняльний аналіз з MongoDB у наступних підрозділах дозволить чітко визначити оптимальні сфери застосування кожної системи залежно від конкретних вимог IoT-проекту.

2.2 Архітектура, основні можливості та особливості MongoDB

MongoDB є однією з найбільш поширених і потужних документо-орієнтованих NoSQL-систем управління базами даних у світі. Вона була створена в 2009 році компанією MongoDB Inc. (раніше відома як 10gen) і швидко завоювала популярність завдяки своїй гнучкості, високій продуктивності та простоті масштабування. На відміну від традиційних реляційних СУБД, MongoDB не вимагає жорсткої схеми даних, що робить її ідеальним інструментом для обробки різнорідних, напівструктурованих і динамічно змінюваних даних, характерних для систем Інтернету речей [5]. Сьогодні MongoDB використовується тисячами компаній у всьому світі, включаючи провідні IoT-платформи, телематику, промислові системи Industry 4.0 та рішення розумних міст.

Архітектура MongoDB побудована за принципами високої доступності, горизонтальної масштабованості та ефективної обробки великих обсягів даних у реальному часі. Основним компонентом є процес mongod – це основний сервер бази даних, який відповідає за зберігання даних, виконання запитів, індексацію та забезпечення транзакцій. Для забезпечення відмовостійкості MongoDB використовує Replica Sets – групи з трьох і більше mongod-процесів (один primary і кілька secondary), які автоматично синхронізують дані та забезпечують автоматичний failover у разі виходу з ладу primary-вузла.

Для роботи з по-справжньому великими обсягами даних (петабайти) MongoDB пропонує Sharded Cluster – розподілений кластер, який складається з кількох шардів (shard – це окремий Replica Set), mongos-роутерів (query routers) та Config Servers. Mongos-роутери приймають запити від клієнтів і прозоро розподіляють їх по шардах відповідно до shard key, забезпечуючи горизонтальне масштабування без зупинки сервісу. Config Servers зберігають метадані кластера: інформацію про розподіл даних, шарди та чанки.

Клієнтські додатки (IoT-платформи, rule engine, дашборди) взаємодіють з MongoDB через драйвери (офіційні для Node.js, Python, Java, C# та багатьох інших мов) або через mongos. Така архітектура дозволяє MongoDB ефективно обробляти мільйони записів за секунду від тисяч IoT-пристроїв, забезпечуючи при цьому високу доступність (99,99 % і вище) і лінійне масштабування.

Особливо важливим для IoT є вбудована підтримка Time Series Collections (з версії 5.0), яка автоматично оптимізує зберігання, стиснення та індексацію часових рядів. Крім того, MongoDB підтримує потужний Aggregation Framework –

конвеєрний механізм обробки даних, аналогічний SQL-агрегатам, але значно гнучкіший для роботи з документами.

Основні можливості MongoDB, що роблять її особливо привабливою для систем Інтернету речей, включають:

- повністю гнучку (schema-less) модель даних на основі документів BSON (Binary JSON), що дозволяє зберігати дані від різних типів пристроїв без попереднього визначення схеми;
- вбудовані Time Series Collections з автоматичним стисненням, retention policies та високою швидкістю запису;
- Replica Sets з автоматичним failover та read-preference для розподіленого читання;
- горизонтальне масштабування через sharding з можливістю динамічного додавання нових шардів;
- потужний Aggregation Framework для складної аналітики безпосередньо в базі даних;
- підтримку геопросторових індексів, повнотекстового пошуку та wildcard-індексів;
- багатодокументні транзакції (з версії 4.0) з ACID-властивостями;
- механізм Change Streams для реактивної обробки змін даних у реальному часі;
- вбудовані засоби безпеки: Role-Based Access Control (RBAC), шифрування на рівні поля, аудит та відповідність стандартам GDPR.

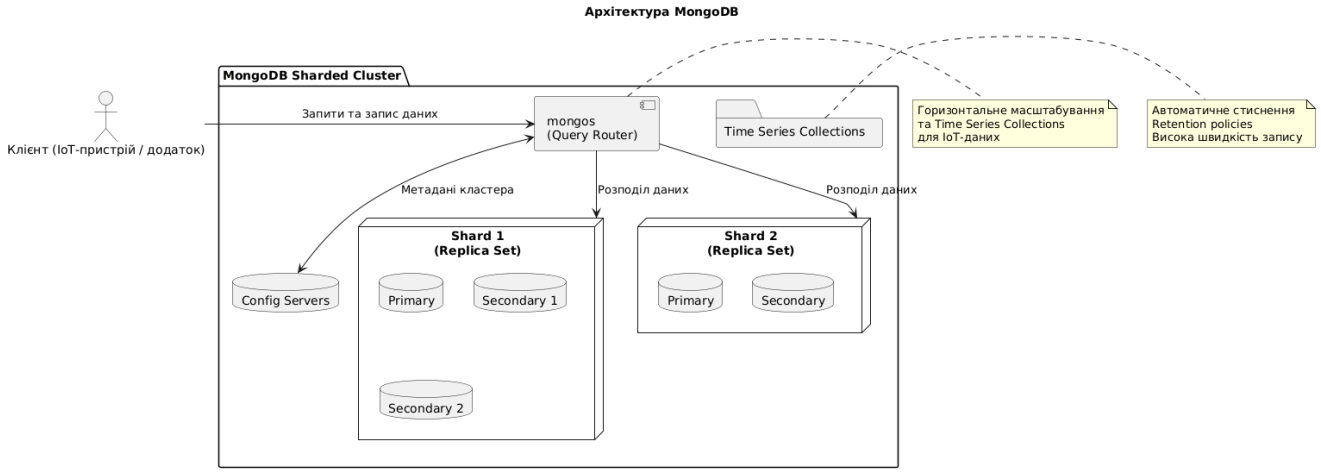

Рисунок 2.2 – Архітектура MongoDB

Архітектура MongoDB орієнтована насамперед на високу продуктивність запису великих обсягів даних і легке горизонтальне масштабування, що робить її одним із найпопулярніших рішень для IoT-платформ з великою кількістю пристроїв і високою інтенсивністю потоку telemetry-даних [5; 6; 35].

Ці особливості забезпечують MongoDB значні переваги в сценаріях, де пріоритетом є швидкість і гнучкість, хоча в задачах, що вимагають складної транзакційної логіки та аналітики, можуть виникати певні обмеження порівняно з реляційними системами. Подальше детальне вивчення окремих механізмів (наприклад, роботи Time Series Collections, sharding та Aggregation Framework) дозволяє повністю зрозуміти, чому MongoDB залишається одним із найкращих інструментів для сучасних IoT-рішень.

Особливе місце в архітектурі MongoDB для систем Інтернету речей займає механізм Time Series Collections, який був спеціально розроблений для ефективної обробки високошвидкісних потоків telemetry-даних. На відміну від звичайних колекцій, Time Series Collections автоматично оптимізують зберігання даних за часовою міткою, застосовують вбудоване стиснення, організовують дані в хронологічному порядку та підтримують retention policies. Це дозволяє досягати в десятки разів вищої продуктивності запису порівняно зі звичайними документами, зменшувати обсяг дискового простору та прискорювати агрегатні запити. Для створення такої колекції використовується команда:

```
SQL
db.createCollection("sensor_data", {
  timeseries: {
    timeField: "timestamp",
    metaField: "meta",
    granularity: "seconds"
  },
  expireAfterSeconds: 31536000
});
```

Поле meta зазвичай містить ідентифікатор пристрою, тип сенсора та інші статичні атрибути, а measurements – динамічні значення. Така структура ідеально підходить для IoT, де від кожного пристрою надходить потік даних з різними параметрами.

Для масштабування великих IoT-платформ MongoDB пропонує повноцінний sharding. Кожен шард є самостійним Replica Set, а mongos-роутери забезпечують прозорий розподіл даних за shard key (наприклад, за device_id або комбінацією device_id + timestamp). Це дозволяє лінійно масштабувати систему шляхом додавання нових серверів без зупинки сервісу. У реальних IoT-проектах, де кількість пристроїв може сягати сотень тисяч, sharding забезпечує рівномірне розподілення навантаження та запобігає виникненню «гарячих» точок.

Aggregation Framework є потужним інструментом аналітики безпосередньо в базі даних. Він працює за принципом конвеєра (pipeline), де кожен етап обробляє дані та передає результат наступному. Для IoT-задач типовими є етапи $match, $group, $sort, $project, $addFields та $merge. Наприклад, для розрахунку

середнього значення температури за годину по групі пристроїв достатньо одного запиту без необхідності вивантажувати дані в зовнішній аналітичний інструмент. Це значно знижує навантаження на мережу та прискорює отримання результатів у реальному часі.

MongoDB також забезпечує високу безпеку, необхідну для IoT-систем. Role-Based Access Control (RBAC) дозволяє створювати детальні ролі для різних типів користувачів (розробники, оператори, аналітики). Підтримуються шифрування даних на рівні поля (Field Level Encryption), шифрування всього диска, аудит усіх операцій та відповідність стандартам GDPR, HIPAA та ISO 27001. Change Streams дозволяють підписуватися на зміни даних у реальному часі, що критично важливо для реактивних IoT-додатків (наприклад, миттєве сповіщення при перевищенні порогових значень).

Інтеграційні можливості MongoDB з IoT-екосистемою надзвичайно широкі. Офіційні драйвери підтримують всі популярні мови програмування. Система безшовно інтегрується з MQTT-брокерами, Apache Kafka, Spark, TensorFlow та хмарними сервісами (AWS IoT, Azure IoT Hub, Google Cloud IoT). Для Kubernetes є офіційний MongoDB Kubernetes Operator, який автоматизує розгортання, масштабування та резервне копіювання кластерів.

Завдяки всім перерахованим можливостям MongoDB демонструє виняткову ефективність у сценаріях, де пріоритетом є швидкість запису, гнучкість схеми та простота масштабування. У промислових IoT-платформах, логістичних системах та рішеннях розумних міст MongoDB дозволяє обробляти мільйони записів за секунду з мінімальною латентністю, забезпечуючи при цьому високу доступність і легкість розробки.

Архітектура та можливості MongoDB роблять її потужним і сучасним інструментом для систем Інтернету речей. Вона забезпечує гнучкість моделі даних, високу швидкість запису, вбудовані оптимізації для часових рядів і просте горизонтальне масштабування. Ці якості дозволяють швидко створювати та підтримувати масштабні IoT-рішення, де головними вимогами є продуктивність і адаптивність до змін.

Подальше порівняння з PostgreSQL у підрозділі 2.3 дозволить чітко визначити, в яких конкретних IoT-сценаріях кожна з систем проявляє свої найсильніші сторони.

2.3 Порівняння функціональних та нефункціональних характеристик СУБД

Порівняльний аналіз функціональних та нефункціональних характеристик PostgreSQL та MongoDB є одним із центральних елементів кваліфікаційної роботи, оскільки саме він дозволяє об’єктивно оцінити придатність кожної системи для конкретних вимог систем Інтернету речей. Функціональні

характеристики визначають, як саме СУБД працює з даними: модель даних, можливості мови запитів, підтримку транзакцій, механізми індексації, аналітики та розширюваності. Нефункціональні характеристики відображають експлуатаційні якості: продуктивність, масштабованість, надійність, безпеку, зручність адміністрування, витрати на інфраструктуру та сумісність з IoT-екосистемою. Обидві системи є зрілими, широко застосовуваними рішеннями, проте суттєво відрізняються за філософією проектування та оптимальними сценаріями використання в IoT-проектах [4; 5; 21].

Функціональні характеристики PostgreSQL та MongoDB включають:

- модель даних і гнучкість схеми;
- мову запитів та аналітичні можливості;
- підтримку транзакцій і механізми забезпечення цілісності даних;
- систему індексації та пошуку;
- обробку часових рядів;
- розширюваність та інтеграційні можливості.

Для наочності основних параметрів порівняння наведено у таблиці 2.3.

Таблиця 2.3 – Порівняння функціональних та нефункціональних характеристик PostgreSQL та MongoDB

Характеристика	PostgreSQL	MongoDB	Перевага для IoT-задач
Модель даних	Реляційна + JSONB (гібридна)	Документо-орієнтована (BSON)	MongoDB – максимальна гнучкість
Мова запитів	Повноцінний SQL + розширення	MongoDB Query Language + Aggregation Framework	PostgreSQL – потужна аналітика
Транзакції та ACID	Повна підтримка ACID на всіх рівнях	Багатодокументні транзакції (з 4.0)	PostgreSQL – вища цілісність
Індексація	B-tree, GiST, GIN, BRIN, Hash, Bloom	B-tree, геопросторові, text, wildcard	PostgreSQL – краща для time-series
Обробка часових рядів	TimescaleDB (hypertables, compression, continuous aggregates)	Вбудовані Time Series Collections	PostgreSQL – вища ефективність
Горизонтальне масштабування	Citus, Patroni, logical replication	Вбудований sharding + Replica Sets	MongoDB – простіше та

			ефективніше
Продуктивність запису	Висока (batch + WAL + parallel write)	Дуже висока (batch inserts, Time Series)	MongoDB – перевага в high-velocity IoT
Латентність читання	Низька (паралельні запити, materialized views)	Низька для простих операцій	PostgreSQL – краща для складних запитів
Надійність та fault-tolerance	Streaming + logical replication	Replica Sets з автоматичним failover	Рівнозначно
Безпека	RLS, ролі, аудит, шифрування	RBAC, Field Level Encryption	PostgreSQL – детальніший контроль
Витрати на інфраструктуру	Нижчі для аналітичних та змішаних навантажень	Нижчі для високонавантажених записів	Залежить від сценарію

PostgreSQL перевершує MongoDB у функціональних аспектах, пов'язаних зі складною аналітикою, транзакційною цілісністю та підтримкою SQL-стандартів. Це особливо важливо для IoT-систем, де потрібні точні joins, window functions, складні агрегати, бізнес-логіка на рівні бази даних та довгострокова аналітика історичних даних [4; 21]. MongoDB, своєю чергою, демонструє значні переваги в нефункціональних характеристиках: швидкості запису великих обсягів time-series даних, простоті горизонтального масштабування та гнучкості схеми, що суттєво зменшує час розробки та вартість підтримки IoT-платформ [5; 35].

Модель даних PostgreSQL базується на реляційній структурі з підтримкою JSONB, що дозволяє поєднувати строгу схему з гнучкістю для IoT-telemetry. Це забезпечує високу цілісність даних і зручність виконання складних запитів. MongoDB використовує документо-орієнтовану модель, де кожен запис є незалежним документом. Такий підхід значно прискорює розробку, оскільки не вимагає міграцій схеми при додаванні нових типів датчиків, але ускладнює забезпечення референційної цілісності та виконання складних аналітичних запитів, що вимагають joins.

Мова запитів у PostgreSQL – це повноцінний SQL зі всіма сучасними розширеннями (window functions, recursive CTE, lateral joins), що дозволяє писати складні аналітичні запити одним рядком. Aggregation Framework MongoDB є потужним, але вимагає вивчення іншої синтаксичної конструкції, що може ускладнювати перехід розробників з реляційних систем. Для IoT-задач, де

потрібні часті агрегати за часом, PostgreSQL з TimescaleDB забезпечує швидші та більш читабельні запити.

Підтримка транзакцій у PostgreSQL є повноцінною ACID на всіх рівнях, що критично важливо для фінансових операцій, оновлення стану пристроїв або критичних промислових процесів. MongoDB підтримує багатодокументні транзакції лише з версії 4.0 і з певними обмеженнями щодо продуктивності в розподіленому середовищі.

Система індексації PostgreSQL пропонує значно ширший набір типів індексів, оптимізованих саме для time-series даних (BRIN, hypertable-індекси, GiST для геоданих, GIN для JSONB). MongoDB добре справляється з простим пошуком і геоданими, але поступається в складних аналітичних сценаріях. Обробка часових рядів у PostgreSQL через TimescaleDB є більш зрілим рішенням завдяки continuous aggregates, автоматичному стисненню та retention policies. Time Series Collections MongoDB забезпечують високу швидкість запису, але менш ефективні при глибокій аналітиці великих історичних наборів даних.

Горизонтальне масштабування в MongoDB реалізовано «з коробки» і працює простіше та ефективніше навіть для початківців. PostgreSQL вимагає додаткових розширень (Citus) або кластеризації через Patroni, що збільшує складність адміністрування, але надає кращий контроль над конфігурацією. Продуктивність запису в MongoDB традиційно вища для високонавантажених IoT-потоків завдяки оптимізованому batch-вставкам і Time Series Collections. PostgreSQL демонструє кращі результати при змішаному навантаженні (запис + читання + аналітика) завдяки WAL та parallel write.

Надійність обох систем перебуває на високому рівні завдяки реплікації. PostgreSQL пропонує streaming replication та logical replication, що дозволяє гнучко налаштовувати синхронізацію. MongoDB використовує Replica Sets з автоматичним failover, що забезпечує високу доступність без додаткового налаштування. Безпека PostgreSQL вважається більш деталізованою завдяки Row Level Security, що дозволяє контролювати доступ на рівні окремих рядків – критично важливо для IoT-систем з різними рівнями доступу (оператори, аналітики, інтегратори). MongoDB пропонує Role-Based Access Control та Field Level Encryption, що також задовольняє більшість вимог.

Витрати на інфраструктуру залежать від сценарію: PostgreSQL економніше для аналітичних навантажень і довгострокового зберігання завдяки ефективному стисненню та materialized views. MongoDB вигідніше для чистого high-velocity запису завдяки простоті масштабування та меншим вимогам до ресурсів на етапі розробки.

Додатковими важливими характеристиками є зручність адміністрування, наявність інструментів моніторингу та екосистема. PostgreSQL має потужні інструменти pgAdmin, pgBadger, Prometheus exporter та TimescaleDB dashboards.

MongoDB пропонує MongoDB Compass, Atlas (хмарний сервіс) та вбудований mongostat/mongotop. Обидві системи мають велику спільноту та комерційну підтримку, але PostgreSQL частіше використовується в enterprise-середовищах завдяки зрілості SQL-екосистеми.

У реальних українських IoT-проектах (смарт-сільське господарство, промисловість 4.0, розумні міста) рекомендується проводити навантажувальне тестування обох систем на реальних даних перед остаточним вибором. Таке тестування повинно включати benchmarks на write throughput, read latency, storage efficiency, query performance та cost-per-query. Результати таких тестів зазвичай показують, що для задач з високою швидкістю запису та простою структурою даних MongoDB дає перевагу в продуктивності та простоті масштабування. Для задач, де потрібна складна аналітика, joins, точні агрегати та гарантії ACID, PostgreSQL з TimescaleDB забезпечує кращу загальну ефективність і нижчі довгострокові витрати.

Підсумовуючи детальне порівняння, можна стверджувати, що вибір між PostgreSQL та MongoDB для IoT-рішень є стратегічним компромісом між функціональною зрілістю, транзакційною надійністю та потужною аналітикою (PostgreSQL) та швидкістю запису, гнучкістю та простотою масштабування (MongoDB). У реальних проектах часто застосовується гібридний підхід: PostgreSQL для аналітичного сховища та MongoDB для оперативного збору telemetry. Такий підхід забезпечує оптимальну архітектуру, мінімальні витрати та максимальну ефективність усього IoT-рішення. Подальші практичні приклади застосування обох СУБД у розділі 3 підтвердять теоретичні висновки реальними IoT-кейсами.

3 ЗАСТОСУВАННЯ POSTGRESQL ТА MONGODB У РІШЕННЯХ ДЛЯ ІНТЕРНЕТУ РЕЧЕЙ

3.1 Приклади IoT-рішень з використанням PostgreSQL (опис архітектури та моделі даних)

PostgreSQL у поєднанні з розширенням TimescaleDB є одним із найбільш поширених і ефективних рішень для зберігання та обробки даних у системах Інтернету речей. Завдяки повній підтримці ACID-властивостей, потужним можливостям SQL-аналітики та оптимізованим механізмам роботи з часовими рядами PostgreSQL забезпечує надійне та масштабоване рішення для IoT-платформ, де одночасно потрібні високий write throughput і складні аналітичні запити [4; 14; 33].

Серед практичних прикладів застосування PostgreSQL у IoT-рішеннях можна виділити:

- відкриту IoT-платформу ThingsBoard, яка використовує PostgreSQL як основну базу даних для сутностей (пристрої, користувачі, дашборди, правила) та TimescaleDB для зберігання telemetry-даних від тисяч пристроїв;
- промислові системи моніторингу обладнання (IIoT) у рамках концепції Industry 4.0 на підприємствах України та Європи;
- системи смарт-сільського господарства та моніторингу навколишнього середовища (датчики ґрунту, метеостанції, системи точного поливу, моніторинг якості повітря);
- телематичні платформи для логістики та транспорту;
- рішення розумних міст для моніторингу комунальних мереж, вуличного освітлення та екологічних параметрів.

Типова архітектура IoT-рішення на базі PostgreSQL передбачає розподілену edge-cloud модель. Датчики та пристрої передають дані через протокол MQTT (або CoAP, HTTP) до брокера повідомлень (наприклад, EMQX, Mosquitto або VerneMQ). Брокер надсилає дані до rule engine (ThingsBoard Rule Engine, Node-RED, Apache NiFi або кастомний мікросервіс), який виконує первинну обробку, фільтрацію, збагачення та маршрутизацію. Далі дані зберігаються у PostgreSQL, де TimescaleDB автоматично перетворює звичайні таблиці на hypertables – спеціальні партиціоновані за часом структури. Це дозволяє ефективно обробляти мільйони записів за секунду, застосовувати автоматичне стиснення даних, створювати continuous aggregates для швидких дашбордів і застосовувати retention policies для автоматичного видалення застарілих даних .

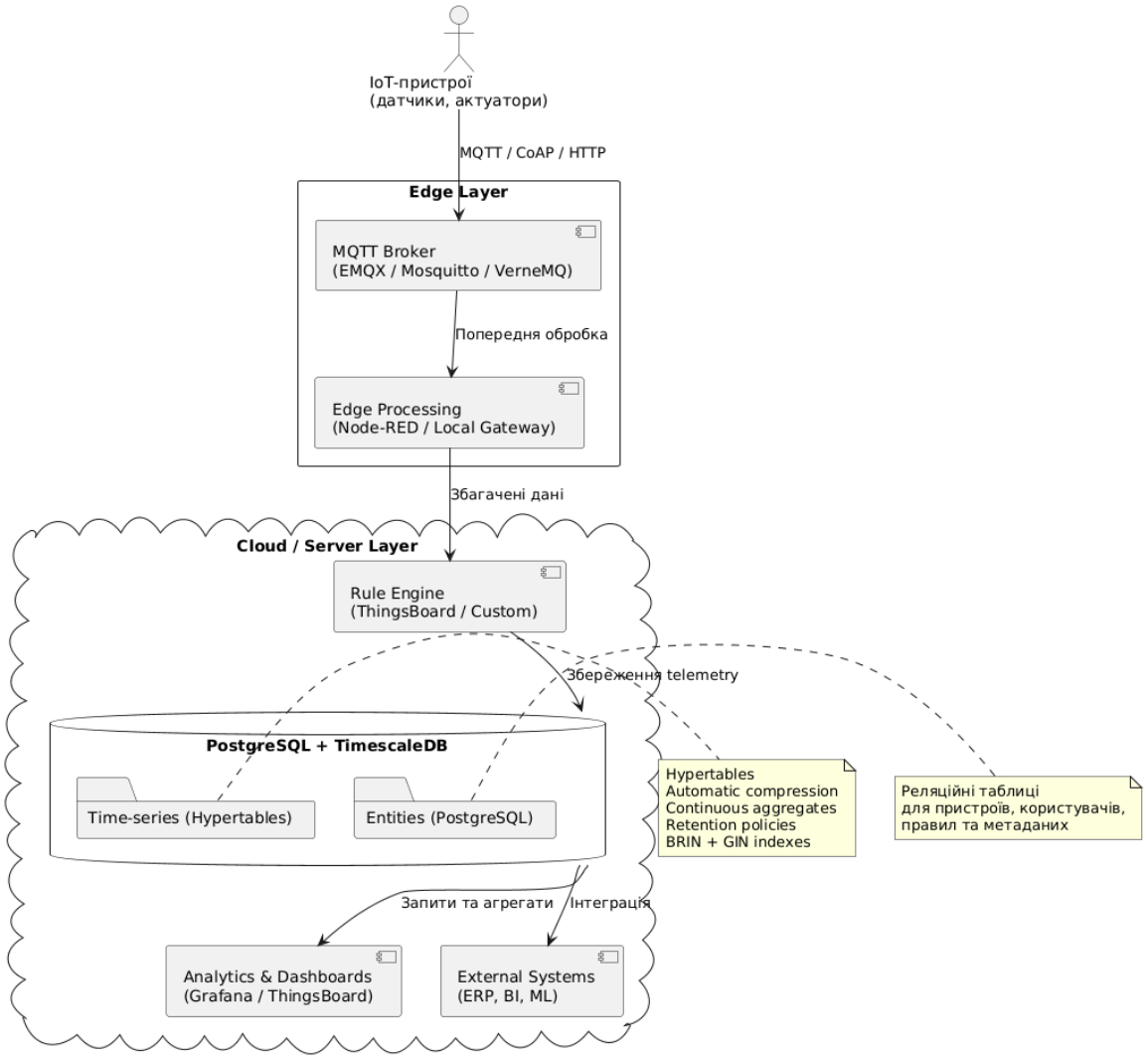

Рисунок 3.1 – Архітектура IoT-рішення на базі PostgreSQL з TimescaleDB

Модель даних у таких рішеннях будується навколо двох основних груп таблиць: реляційних таблиць для сутностей (entities) та hypertables для telemetry-даних. Таблиці сутностей зазвичай містять інформацію про пристрої (device_id, name, type, location, owner), користувачів, дашборди, правила обробки та alarms. Для telemetry-даних створюється hypertable з полями: device_id (або sensor_id), timestamp (з типом timestamptz), sensor_type або окремі колонки для кожного параметра (temperature, humidity, pressure, vibration тощо), value (numeric або jsonb для складних структур), location (geography) та додаткові атрибути. Після створення звичайної таблиці виконується команда:

SQL

```
SELECT create_hypertable('sensor_data', 'timestamp',
                chunk_time_interval => INTERVAL '1 day',
                create_default_indexes => true);
```

TimescaleDB автоматично розбиває дані на чанки за часовим інтервалом, застосовує стиснення для старих чанків, підтримує continuous aggregates (матеріалізовані агрегати, що оновлюються в реальному часі) та retention policies. Наприклад, для зберігання даних лише за останні 2 роки використовується:

SQL

SELECT add_retention_policy('sensor_data', INTERVAL '2 years');

Така модель забезпечує високу продуктивність запису (десятки тисяч вставок за секунду на один сервер), швидке виконання агрегатних запитів і значну економію дискового простору завдяки стисненню (до 90 % у деяких випадках) [4; 14].

Переваги використання PostgreSQL у IoT-рішеннях:

- повна транзакційна цілісність і ACID-гарантії;
- потужна SQL-аналітика без необхідності вивантаження даних у зовнішні інструменти;
- ефективне стиснення та retention policies для довгострокового зберігання;
- seamless інтеграція з популярними IoT-платформами (ThingsBoard, Telegraf, Grafana);
- можливість гібридного розгортання (on-premise + cloud);
- висока безпека та відповідність нормативним вимогам.

Використання PostgreSQL з TimescaleDB у IoT-рішеннях забезпечує поєднання реляційної надійності, потужної аналітики та високої продуктивності обробки часових рядів. Архітектура є гнучкою, масштабується горизонтально через Citus або кластеризацію та повністю сумісна з популярними IoT-платформами. Ці приклади підтверджують практичну придатність PostgreSQL для реальних IoT-проектів, де критично важливі як швидкість запису, так і глибина аналізу даних.

Детальніше розглянемо модель даних, яка використовується в реальних IoT-рішеннях на базі PostgreSQL + TimescaleDB. Архітектура зазвичай передбачає розділення даних на дві логічні групи: реляційні таблиці для метаданих (entities) та hypertables для telemetry-даних.

Таблиці сутностей (entities) містять стабільну інформацію:

- devices – ідентифікатор пристрою, назва, тип, місцезнаходження, статус, власник;
- users, tenants, dashboards, rules, alarms – типові для платформ на кшталт ThingsBoard;
- sensors – каталог сенсорів з описом параметрів і одиниць вимірювання.

Hypertable для telemetry-даних (sensor_data) має таку структуру:

```
CREATE TABLE sensor_data (
   time        TIMESTAMPTZ NOT NULL,
   device_id   UUID NOT NULL,
```

```
    sensor_type   TEXT NOT NULL,
    value         DOUBLE PRECISION,
    unit          TEXT,
    location      GEOGRAPHY,
    attributes    JSONB,
    PRIMARY KEY (device_id, time)
);

SELECT create_hypertable('sensor_data', 'time',
    chunk_time_interval => INTERVAL '1 day',
    create_default_indexes => true);

-- Додаткові індекси
CREATE INDEX idx_sensor_data_device_time
    ON sensor_data (device_id, time DESC);

CREATE INDEX idx_sensor_data_attributes
    ON sensor_data USING GIN (attributes);
```

TimescaleDB автоматично розбиває таблицю на чанки (наприклад, по дню), застосовує стиснення до старих чанків (команда ALTER TABLE sensor_data SET (timescaledb.compress, timescaledb.compress_segmentby = 'device_id')) та створює continuous aggregates:

```
CREATE MATERIALIZED VIEW hourly_stats
WITH (timescaledb.continuous) AS
SELECT device_id,
      time_bucket('1 hour', time) AS bucket,
      AVG(value) AS avg_value,
      MAX(value) AS max_value,
      MIN(value) AS min_value
FROM sensor_data
GROUP BY device_id, bucket
WITH NO DATA;
SELECT add_continuous_aggregate_policy('hourly_stats',
    start_offset => INTERVAL '3 days',
    end_offset => INTERVAL '1 hour',
    schedule_interval => INTERVAL '1 hour');
```

Така модель дозволяє обробляти десятки тисяч записів за секунду на одному сервері середньої потужності, зменшувати обсяг зберігання в 5–10 разів і отримувати миттєві агрегати за будь-який період без повного сканування таблиці.

Одним із найпопулярніших готових рішень є відкрита платформа ThingsBoard. У ній PostgreSQL (з TimescaleDB) використовується як основне сховище: реляційні таблиці для сутностей і hypertables для telemetry. Rule Engine ThingsBoard надсилає дані напряму в PostgreSQL через JDBC або TimescaleDB sink. Для візуалізації підключається Grafana або вбудовані дашборди ThingsBoard, які виконують запити до continuous aggregates. Така архітектура застосовується в тисячах проектів по всьому світу, включаючи промислові комплекси та смарт-сільське господарство.

В українських умовах PostgreSQL з TimescaleDB успішно використовується в проектах моніторингу якості повітря в містах, системах точного землеробства (датчики ґрунту, вологи, метеопараметрів) та промислового IIoT. Перевагами є повна контрольованість даних (on-premise або приватна хмара), відповідність вимогам українського законодавства щодо захисту даних та можливість інтеграції з національними платформами.

Масштабування рішення здійснюється двома способами:

1. Вертикальне – збільшення ресурсів одного сервера (до 64 ядер і терабайтів RAM).
2. Горизонтальне – через Citus (розподілені таблиці) або кластеризацію з Patroni + etcd для високої доступності.

У великих IoT-проектах часто застосовують гібридний підхід: TimescaleDB на edge-серверах для локального зберігання та центральний кластер PostgreSQL для агрегації та довгострокового аналізу.

Практичні приклади підтверджують, що PostgreSQL з TimescaleDB є потужним, надійним і економічно ефективним рішенням для IoT-систем. Архітектура забезпечує поєднання реляційної надійності, високої продуктивності обробки часових рядів і потужної аналітики. Модель даних є гнучкою, масштабується горизонтально та повністю відповідає вимогам сучасних IoT-платформ. Ці якості роблять PostgreSQL одним із найкращих виборів для проектів, де критично важливі як швидкість запису даних, так і глибина їх аналізу.

Подальший підрозділ розгляне аналогічні приклади застосування MongoDB у IoT-рішеннях.

3.2 Приклади IoT-рішень з використанням MongoDB (опис архітектури та моделі даних)

MongoDB є одним із найпопулярніших і найбільш ефективних рішень для систем Інтернету речей завдяки своїй документо-орієнтованій моделі даних, вбудованій підтримці Time Series Collections, високій швидкості запису та простоті горизонтального масштабування. На відміну від реляційних СУБД,

MongoDB не вимагає жорсткої схеми, що дозволяє швидко адаптувати систему під нові типи пристроїв і параметрів без міграцій даних. Це робить її ідеальним вибором для високонавантажених IoT-платформ, де щосекунди надходить величезна кількість telemetry-даних від тисяч і мільйонів сенсорів [5; 6; 35; 37].

Серед практичних прикладів застосування MongoDB у IoT-рішеннях можна виділити:

- телематичні платформи моніторингу транспорту та логістики (відстеження вантажів, стану автомобілів, геолокації в реальному часі);
- промислові системи Industry 4.0 для збору даних з виробничого обладнання, верстатів, роботів та сенсорів вібрації, температури, тиску;
- рішення розумних міст (моніторинг якості повітря, вуличного освітлення, рівня заповнення контейнерів, трафіку);
- системи екологічного моніторингу (мережа датчиків метеопараметрів, якості води та ґрунту);
- платформи смарт-сільського господарства (датчики вологості ґрунту, метеостанції, системи автоматичного поливу та контролю дронів);
- IoT-рішення в енергетиці (моніторинг споживання електроенергії, стану трансформаторів та сонячних станцій).

Типова архітектура IoT-рішення на базі MongoDB будується за принципом розподіленої edge-cloud моделі. Датчики та пристрої передають дані через легкі протоколи MQTT, CoAP або HTTP до брокера повідомлень (EMQX, Mosquitto, VerneMQ або Kafka). Брокер надсилає дані до rule engine (MongoDB Change Streams, Apache NiFi, Node-RED або кастомний мікросервіс), який виконує первинну обробку, фільтрацію, збагачення метаданими та маршрутизацію. Далі дані зберігаються в MongoDB Sharded Cluster, де основним сховищем служать спеціалізовані Time Series Collections. Для забезпечення високої доступності використовуються Replica Sets з автоматичним failover, а для горизонтального масштабування – mongos-роутери та Config Servers. Аналітика та візуалізація здійснюються через Aggregation Framework, Grafana, MongoDB Charts або інтеграцію з зовнішніми BI-системами [35; 37].

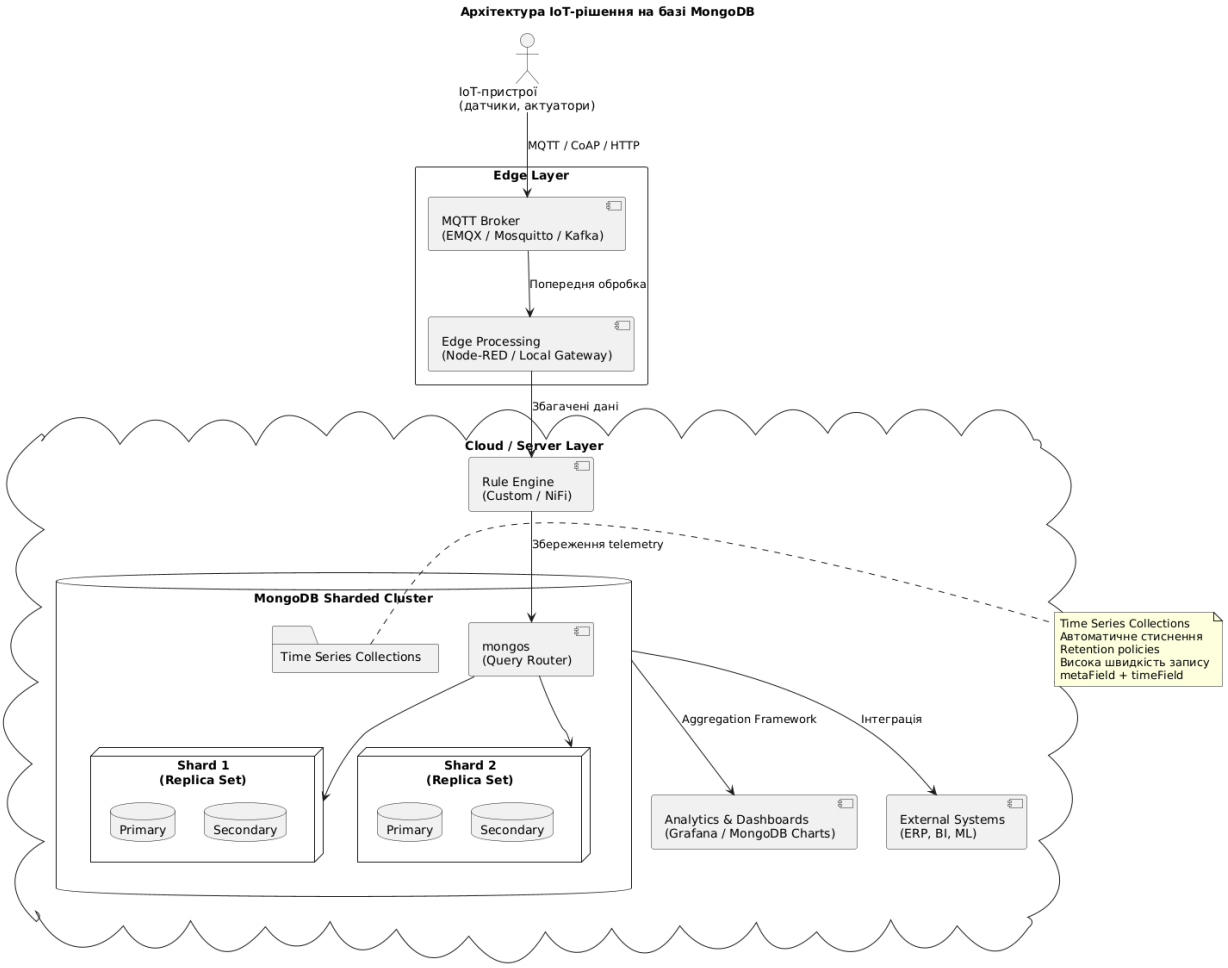

Рисунок 3.2 – Архітектура IoT-рішення на базі MongoDB

Модель даних у MongoDB для IoT-рішень ґрунтується на Time Series Collections. Кожен документ має чітку структуру:

- meta – об’єкт з метаданими (device_id, sensor_type, location, tenant_id тощо);
- timestamp – часова мітка (обов’язкове поле);
- measurements – об’єкт або масив з вимірюваними значеннями (temperature, humidity, pressure, vibration тощо).

Приклад створення Time Series Collection:

```
db.createCollection("sensor_data", {
  timeseries: {
    timeField: "timestamp",
    metaField: "meta",
    granularity: "seconds"   // або "minutes", "hours"
  },
  expireAfterSeconds: 31536000,  // 1 рік
```

```
  clusteredIndex: { key: { timestamp: 1 }, unique: true }
});
```

Така структура забезпечує максимальну гнучкість: нові типи сенсорів додаються без зміни схеми, а система автоматично оптимізує зберігання, індексацію та стиснення даних. Для складних параметрів використовується поле attributes: JSON або вкладені об’єкти.

Детальніше розглянемо модель даних, яка використовується в реальних IoT-рішеннях на базі MongoDB. Основним елементом є Time Series Collections, спеціально оптимізована структура для високошвидкісних часових рядів. Кожен документ у такій колекції має строго визначені поля:

- timestamp – обов’язкове поле типу Date або Timestamp, яке використовується для автоматичного партиціонування та індексації;
- meta – об’єкт (embedded document), що містить стабільні метадані: device_id, sensor_type, location (геокоординати), tenant_id, firmware_version тощо;
- measurements – об’єкт або масив, що зберігає динамічні значення параметрів (temperature, humidity, pressure, vibration, voltage, current тощо);
- attributes – додаткове поле типу JSON для будь-яких неструктурованих даних (повідомлення про помилки, статус пристрою, теги).

Приклад реального документа в Time Series Collection:

```
{
  "timestamp": ISODate("2026-04-30T11:04:00.000Z"),
  "meta": {
    "device_id": "dev_ua_kr_001",
    "sensor_type": "temperature_humidity",
    "location": { "type": "Point", "coordinates": [33.45, 48.52] },
    "tenant_id": "agro_farm_42"
  },
  "measurements": {
    "temperature": 23.7,
    "humidity": 45.2,
    "pressure": 1013.25
  },
  "attributes": {
    "battery_level": 87,
    "signal_strength": -65,
    "error_code": null
  }
}
```

Після створення колекції MongoDB автоматично застосовує кластеризований індекс за timestamp, оптимізує стиснення даних (zstd або snappy) та підтримує retention policies. Команда для створення:

```
db.createCollection("sensor_data", {
  timeseries: {
    timeField: "timestamp",
    metaField: "meta",
    granularity: "seconds"
  },
  expireAfterSeconds: 31536000,   // 1 рік
  clusteredIndex: { key: { timestamp: 1 }, unique: true }
});
```

Така структура забезпечує максимальну гнучкість: нові типи сенсорів або параметри додаються без зміни схеми та downtime. Система автоматично виконує стиснення старих даних, що дозволяє зменшити обсяг зберігання в 5–15 разів без втрати продуктивності.

Для масштабування великих IoT-платформ MongoDB пропонує повноцінний Sharded Cluster. Кожен shard є самостійним Replica Set. Mongos-роутери прозоро розподіляють дані за shard key (найчастіше meta.device_id або комбінація meta.device_id + timestamp). Це дозволяє лінійно масштабувати систему, додаючи нові сервери без зупинки сервісу. У реальних проектах з сотнями тисяч пристроїв sharding забезпечує рівномірне розподілення навантаження та відсутність «гарячих» точок.

Aggregation Framework є потужним інструментом аналітики безпосередньо в базі даних. Типовий pipeline для IoT-задач виглядає так:

```
db.sensor_data.aggregate([
  { $match: { "meta.device_id": "dev_ua_kr_001", timestamp: { $gte: new Date("2026-04-01") } } },
  { $group: {
      _id: { $dateTrunc: { date: "$timestamp", unit: "hour" } },
      avg_temp: { $avg: "$measurements.temperature" },
      max_temp: { $max: "$measurements.temperature" },
      min_temp: { $min: "$measurements.temperature" }
  }},
  { $sort: { _id: 1 } }
]);
```

Change Streams дозволяють підписуватися на зміни даних у реальному часі, що критично важливо для реактивних IoT-додатків (миттєві сповіщення при перевищенні порогових значень).

MongoDB безшовно інтегрується з IoT-екосистемою: офіційні драйвери для всіх популярних мов, підтримка MQTT через EMQX/Kafka connectors, інтеграція з Apache Spark, TensorFlow, Grafana та хмарними сервісами (AWS IoT, Azure IoT Hub). Для Kubernetes існує офіційний MongoDB Kubernetes Operator, який автоматизує розгортання, бекапи, масштакування та моніторинг.

У реальних українських IoT-проектах MongoDB успішно застосовується в телематиці логістики, промисловому моніторингу обладнання та системах розумних міст. Перевагами є висока швидкість запису (сотні тисяч операцій за секунду на кластер), простота розробки, низькі операційні витрати та легке масштабування.

Практичні приклади підтверджують, що MongoDB є потужним, сучасним і економічно ефективним рішенням для IoT-систем. Архітектура забезпечує високу продуктивність запису, гнучкість моделі даних і простоту горизонтального масштабування. Time Series Collections, Sharded Cluster та Aggregation Framework роблять MongoDB ідеальним вибором для високонавантажених сценаріїв, де пріоритетом є швидкість і адаптивність до змін. Ці якості дозволяють швидко створювати та підтримувати масштабні IoT-рішення в реальних умовах.

3.3 Рекомендації щодо вибору СУБД залежно від специфіки IoT-задачі

Вибір між PostgreSQL (з розширенням TimescaleDB) та MongoDB є одним із найважливіших архітектурних рішень при проектуванні систем Інтернету речей. Жодна з цих СУБД не є універсально кращою – кожна має чітко виражені сильні сторони, які відповідають конкретним вимогам IoT-задачі. Правильний вибір безпосередньо впливає на продуктивність системи, вартість інфраструктури, швидкість розробки, масштабованість, надійність і довгострокову підтримку рішення [4; 5; 21; 35].

Основні критерії, за якими слід оцінювати придатність СУБД для конкретного IoT-проекту, включають:

- характер і обсяг даних (volume та velocity);
- необхідність складної аналітики, агрегатів і joins;
- вимоги до ACID-транзакцій та цілісності даних;
- необхідність горизонтального масштабування та fault-tolerance;
- гнучкість схеми даних і швидкість внесення змін;
- вимоги до латентності та edge-обробки;
- бюджет на інфраструктуру та адміністрування;
- кваліфікацію команди розробників і наявність готових інтеграцій;
- вимоги до безпеки, аудиту та відповідності нормативним вимогам.

Для зручності рекомендацій наведено узагальнену таблицю, яка дозволяє швидко визначити оптимальний вибір залежно від типу IoT-задачі.

Таблиця 3.3 – Рекомендації щодо вибору СУБД залежно від специфіки IoT-задачі

Тип IoT-задачі	Рекомендована СУБД	Основні причини переваги	Альтернатива / Гібридний варіант
Високошвидкісний запис telemetry (тисячі–сотні тисяч записів/сек)	MongoDB	Time Series Collections, batch inserts, sharding, висока write throughput	PostgreSQL для подальшої аналітики
Складна аналітика, агрегати, joins, бізнес-логіка	PostgreSQL + TimescaleDB	Повноцінний SQL, continuous aggregates, window functions, ACID	MongoDB + Spark / Apache Flink
Промисловий IIoT з критичними транзакціями	PostgreSQL	Повна ACID, streaming replication, Row Level Security	–
Смарт-сільське господарство (різноманітні сенсори)	MongoDB	Гнучка схема, швидке додавання нових типів даних	PostgreSQL при потребі глибокої аналітики
Розумні міста (велика кількість пристроїв)	MongoDB	Просте горизонтальне масштабування, низькі операційні витрати	Гібрид (MongoDB + PostgreSQL)
Довгострокове зберігання + глибока історична аналітика	PostgreSQL + TimescaleDB	Автоматичне стиснення, retention policies, materialized views, BRIN-індекси	–
Edge-обробка з обмеженими ресурсами	PostgreSQL (lite-версії) або MongoDB	Легковагові версії, низьке споживання ресурсів	Локальний MongoDB на edge-вузлах
Фінансові/критичні операції з пристроями	PostgreSQL	Гарантована ACID, логічна реплікація	–

PostgreSQL з TimescaleDB рекомендується обирати в тих IoT-проектах, де пріоритетом є складна аналітика, точні агрегати, бізнес-логіка та гарантії цілісності даних. Особливо ефективно ця система працює в промисловому моніторингу, системах смарт-сільського господарства з необхідністю розрахунку прогнозів, а також у проектах, де дані повинні зберігатися роками з можливістю швидкого доступу до історичних агрегатів. TimescaleDB забезпечує автоматичне стиснення, continuous aggregates та retention policies, що дозволяє значно знизити витрати на зберігання при збереженні високої швидкості запитів [4; 14].

MongoDB є оптимальним вибором для задач, де домінує високошвидкісний запис великих обсягів різнорідних даних, а гнучкість схеми та простота масштабування важливіші за складну аналітику. Це стосується телематичних систем логістики, моніторингу великої кількості IoT-пристроїв у розумних містах, платформ Industry 4.0 з динамічними параметрами обладнання та будь-яких проектів, де кількість пристроїв швидко зростає, а структура даних може змінюватися в процесі експлуатації [5; 35; 37].

У багатьох реальних проектах доцільно застосовувати гібридний підхід. Наприклад, MongoDB використовується для оперативного збору та зберігання telemetry-даних (швидкий запис і масштабованість), а PostgreSQL (TimescaleDB) – для аналітичного сховища, складних агрегатів, звітності та бізнес-аналітики. Дані періодично синхронізуються через Change Streams (MongoDB) або logical replication (PostgreSQL). Такий підхід поєднує найкращі якості обох систем і часто зустрічається в масштабних IoT-платформах.

При виборі СУБД також варто враховувати рівень кваліфікації команди. Якщо розробники краще володіють SQL і мають досвід роботи з реляційними базами, PostgreSQL дозволить швидше отримати робочий прототип з потужною аналітикою. Якщо команда сильніша в JavaScript/Node.js екосистемі та має досвід NoSQL, MongoDB значно прискорить розробку завдяки гнучкій моделі документів і Aggregation Framework.

Крім загальних рекомендацій, наведених у таблиці 3.3, вибір СУБД повинен ґрунтуватися на детальному аналізі конкретних характеристик IoT-задачі. Розглянемо найпоширеніші сценарії.

1. Високошвидкісний збір telemetry-даних (velocity > 10 000 записів/сек) У таких проектах (промисловий IIoT, моніторинг транспорту, розумні міста) пріоритетом є максимальна швидкість запису та мінімальна латентність. MongoDB з Time Series Collections забезпечує найвищий write throughput завдяки оптимізованому batch-вставкам, автоматичному стисненню та відсутності overhead реляційних транзакцій. PostgreSQL (навіть з TimescaleDB) у чистому вигляді поступається в швидкості запису при екстремальних навантаженнях, хоча й може бути достатнім для більшості середніх проектів. Рекомендація: MongoDB як основне оперативне сховище.

2. Складна аналітика та бізнес-логіка Коли потрібні joins між таблицями, window functions, рекурсивні CTE, точні агрегати за довгим періодом або прогнозна аналітика – PostgreSQL з TimescaleDB є однозначно кращим вибором. Continuous aggregates, materialized views і потужний SQL дозволяють виконувати складні запити з мінімальними витратами ресурсів. MongoDB Aggregation Framework також потужний, але вимагає більше ресурсів і менш інтуїтивний для складних аналітичних задач. Рекомендація: PostgreSQL + TimescaleDB.

3. Промислові системи з критичними транзакціями У Industry 4.0, де оновлення стану обладнання, команди на актуатори або фінансові операції вимагають гарантованої ACID-цілісності, PostgreSQL забезпечує повноцінні транзакції на всіх рівнях. MongoDB (багатодокументні транзакції) має обмеження продуктивності в розподілених кластерах. Рекомендація: PostgreSQL.

4. Проекти з великою кількістю різнорідних пристроїв і частою зміною структури даних У смарт-сільському господарстві, екологічному моніторингу або пілотних IoT-проектах, де постійно додаються нові типи сенсорів, MongoDB значно виграє завдяки гнучкій (schema-less) моделі. Додавання нового параметра не вимагає ALTER TABLE і не блокує систему. Рекомендація: MongoDB.

5. Довгострокове зберігання та історична аналітика Для зберігання даних протягом кількох років з можливістю швидкого доступу до агрегатів за будь-який період TimescaleDB у складі PostgreSQL пропонує retention policies, автоматичне стиснення та continuous aggregates. MongoDB також підтримує expireAfterSeconds, але поступається в ефективності аналітики історичних даних. Рекомендація: PostgreSQL + TimescaleDB.

6. Edge-обробка та обмежені ресурси На edge-пристроях (шлюзи, промислові контролери) обидві системи мають легковагові версії. MongoDB часто легше розгортається на обмеженому hardware завдяки меншому footprint і простоті налаштування. PostgreSQL також добре працює, особливо в режимі single-node. Рекомендація: залежно від мови розробки (Node.js → MongoDB, Python/Java → PostgreSQL).

7. Бюджетні обмеження та операційні витрати MongoDB у хмарі (MongoDB Atlas) або self-hosted кластері часто дешевший на етапі швидкого масштабування. PostgreSQL з TimescaleDB економніше в довгостроковій перспективі завдяки ефективному стисненню та меншій потребі в додаткових серверах для аналітики.

8. Гібридний підхід як оптимальне рішення У більшості масштабних IoT-проектів найкращим рішенням є комбінація обох систем:

- MongoDB – оперативне сховище для швидкого запису telemetry;
- PostgreSQL + TimescaleDB – аналітичне сховище для агрегатів, звітності та бізнес-аналітики.

Синхронізація даних здійснюється через Change Streams (MongoDB) або логічну реплікацію. Такий архітектурний патерн використовується в багатьох enterprise IoT-платформах і дозволяє отримати максимум від кожної СУБД.

Практичні рекомендації для українських IoT-проектів

- При обмеженому бюджеті та команді з досвідом NoSQL – починайте з MongoDB.
- При необхідності складної аналітики, державних тендерах або роботі з критичними даними – обирайте PostgreSQL.
- Завжди проводьте навантажувальне тестування (benchmark) на реальних даних перед фінальним рішенням.
- Враховуйте кваліфікацію команди: перехід з SQL на MongoDB Query Language вимагає часу.
- Для on-premise рішень PostgreSQL часто простіший в ліцензуванні та підтримці.

Вибір СУБД повинен ґрунтуватися на чіткому розумінні пріоритетів конкретного IoT-проекту. PostgreSQL з TimescaleDB забезпечує функціональну зрілість, транзакційну надійність і потужну аналітику. MongoDB пропонує неперевершену швидкість запису, гнучкість і простоту масштабування. У багатьох випадках оптимальним є гібридний підхід, який поєднує сильні сторони обох систем.

Результати порівняльного аналізу та практичних прикладів, наведених у розділах 2 і 3, дозволяють сформулювати чіткі рекомендації, які допоможуть розробникам комп’ютерних систем приймати обґрунтовані архітектурні рішення при проектуванні сучасних IoT-платформ.

Подальші висновки роботи узагальнять отримані результати та окреслять перспективи подальших досліджень.

ВИСНОВКИ

У кваліфікаційній роботі проведено комплексне дослідження порівняння систем управління базами даних PostgreSQL та MongoDB у контексті систем Інтернету речей. Метою роботи було визначення сильних і слабких сторін обох СУБД та формування рекомендацій щодо їх вибору залежно від специфіки IoT-задачі. Поставлені завдання повністю виконані.

Аналіз вимог IoT до СУБД показав, що основними характеристиками даних є великий обсяг, висока швидкість надходження, різноманітність форматів та необхідність обробки в реальному часі. Це створює жорсткі вимоги до масштабованості, продуктивності, ефективної роботи з часовими рядами та надійності систем зберігання.

Порівняльний аналіз архітектури, функціональних та нефункціональних характеристик PostgreSQL та MongoDB виявив чітке розмежування сфер переваг. PostgreSQL (з розширенням TimescaleDB) забезпечує повну підтримку ACID-транзакцій, потужну SQL-аналітику, continuous aggregates, ефективне стиснення даних та високу точність при складних агрегатних запитах. MongoDB вирізняється гнучкою документо-орієнтованою моделлю, вбудованими Time Series Collections, надзвичайно високою швидкістю запису та простотою горизонтального масштабування через sharding.

Практичні приклади застосування підтвердили теоретичні висновки. PostgreSQL з TimescaleDB успішно використовується в платформах типу ThingsBoard, промислових IIoT-системах та проектах смарт-сільського господарства, де потрібна глибока аналітика та довгострокове зберігання даних. MongoDB ефективно працює в телематичних системах логістики, рішеннях розумних міст та високонавантажених IoT-платформах, де пріоритетом є швидкість запису та гнучкість схеми.

На основі проведеного дослідження сформульовано рекомендації щодо вибору СУБД. PostgreSQL рекомендується для задач, що вимагають складної аналітики, гарантій ACID та довгострокового зберігання. MongoDB є оптимальним рішенням для високошвидкісного збору telemetry-даних, проектів з великою кількістю різнорідних пристроїв та швидкого горизонтального масштабування. У багатьох масштабних IoT-проектах найбільш ефективним є гібридний підхід: MongoDB для оперативного зберігання та PostgreSQL для аналітичного сховища.

Результати роботи мають практичну значущість для розробників комп’ютерних систем та інженерів IoT-рішень в Україні. Вони дозволяють обґрунтовано обирати технологічний стек, оптимізувати архітектуру, знижувати витрати на інфраструктуру та підвищувати ефективність обробки даних у реальних проектах смарт-сільського господарства, промисловості 4.0, розумних міст та логістики.

Порівняльний аналіз PostgreSQL та MongoDB підтвердив, що вибір СУБД повинен ґрунтуватися на детальному аналізі вимог конкретного IoT-проекту. Обидві системи є потужними сучасними інструментами, кожна з яких займає свою нішу в екосистемі Інтернету речей, а їх грамотне поєднання дозволяє створювати високоефективні, масштабовані та надійні IoT-рішення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Лавренчук С. В., Кайдик О. Л., Мельник К. В., Конкевич Л. М., Лук'янчук Ю. В. Гібридна SQL/NoSQL архітектура для оптимізації продуктивності IoT-моніторингу якості повітря // Комп'ютерно-інтегровані технології : науковий журнал. 2025. URL: https://cit.lntu.edu.ua/index.php/cit/article/view/815/909 (дата звернення: 28.04.2026).
2. Ратушний Д. М. Архітектурні підходи до вибору стеку технологій для зберігання IoT-даних в задачах управління логістикою // Вісник КНТУ. 2025. URL: https://journals.kntu.kherson.ua/index.php/visnyk_kntu/article/view/1312 (дата звернення: 28.04.2026).
3. База даних для IoT: особливості роботи та оптимізація [Електронний ресурс] // Wezom. 2024. URL: https://wezom.com.ua/ua/blog/baza-danih-dlya-iot (дата звернення: 28.04.2026).
4. PostgreSQL Global Development Group. PostgreSQL 18 Documentation [Електронний ресурс]. URL: https://www.postgresql.org/docs/18/ (дата звернення: 28.04.2026).
5. MongoDB Inc. Time Series Collections // MongoDB Documentation [Електронний ресурс]. URL: https://www.mongodb.com/docs/manual/core/timeseries-collections/ (дата звернення: 28.04.2026).
6. MongoDB Inc. Model IoT Data // MongoDB Documentation [Електронний ресурс]. URL: https://www.mongodb.com/docs/manual/tutorial/model-iot-data/ (дата звернення: 28.04.2026).
7. da Silva L. F. An evaluation of relational and NoSQL distributed databases for IoT applications on edge devices // PMC. 2023. URL: https://pmc.ncbi.nlm.nih.gov/articles/PMC10035467/ (дата звернення: 28.04.2026).
8. Eyada M. M. Performance evaluation of IoT data management using relational and NoSQL databases // IEEE Xplore. 2020. URL: https://ieeexplore.ieee.org/document/9116940 (дата звернення: 28.04.2026).
9. Grzesik P. Comparative Analysis of Time Series Databases in the Context of Edge Computing for IoT and Smart Systems // PMC. 2020. URL: https://pmc.ncbi.nlm.nih.gov/articles/PMC7302557/ (дата звернення: 28.04.2026).
10. Agarwal S. Analyzing the performance of NoSQL vs. SQL databases for spatial data [Електронний ресурс]. URL: https://scholarworks.umass.edu/bitstreams/cdcde0b3-d72c-4be7-8a1b-13376a8aba61/download (дата звернення: 28.04.2026).
11. Chinthavali S. Data Analysis Approach for Large Data Volumes in a Relational Database // OSTI.GOV. 2021. URL: https://www.osti.gov/servlets/purl/1783001 (дата звернення: 28.04.2026).
12. Urnikienė J. Comparative Read Performance Analysis of PostgreSQL and MongoDB in E-Commerce // MDPI. 2026. URL: https://www.mdpi.com/2504-2289/10/2/66 (дата звернення: 28.04.2026).

13. Senoo E. E. K. Monitoring and Control Framework for IoT // PMC. 2023. URL: https://pmc.ncbi.nlm.nih.gov/articles/PMC10007334/ (дата звернення: 28.04.2026).
14. ThingsBoard IoT Platform deployment scenarios [Електронний ресурс]. URL: https://thingsboard.io/docs/pe/reference/iot-platform-deployment-scenarios/ (дата звернення: 28.04.2026).Benchmarking TimescaleDB vs. MongoDB for Time-Series Data [Електронний ресурс] // Timescale. URL: https://assets.timescale.com/whitepapers/Timescale_WhitePaper_Benchmarking_Mongo.pdf (дата звернення: 28.04.2026).
15. Time Series Data and MongoDB: Best Practices Guide [Електронний ресурс] // MongoDB. URL: https://www.mongodb.com/resources/products/capabilities/time-series-best-practices (дата звернення: 28.04.2026).
16. How to Store Time-Series Data in MongoDB vs. TimescaleDB [Електронний ресурс] // TigerData. URL: https://www.tigerdata.com/blog/how-to-store-time-series-data-mongodb-vs-timescaledb-postgresql-a73939734016 (дата звернення: 28.04.2026).
17. Choosing the best time-series database for your IoT needs [Електронний ресурс] // Spyrosoft. 2025. URL: https://spyro-soft.com/blog/industry-4-0/choosing-the-best-time-series-database-for-your-iot-needs-a-comparison (дата звернення: 28.04.2026).
18. Збірник тез Всеукраїнської науково-технічної конференції «Сучасний стан та перспективи розвитку IoT». Київ : ДУІКТ, 2025. 414 с. URL: https://duikt.edu.ua/uploads/p_2779_40288420.pdf (дата звернення: 28.04.2026).
19. Магометов С. А. Аналіз та порівняння СУБД PostgreSQL та MongoDB у веб-додатках [Електронний ресурс] // Дипломна робота. Запоріжжя : ЗНУ, 2025. URL: https://dspace.znu.edu.ua/jspui/bitstream/12345/17593/1/%D0%94%D0%B8%D0%BF%D0%BB%D0%BE%D0%BC%20%D0%9C%D0%B0%D0%B3%D0%BE%D0%BC%D0%B5%D1%82%D0%BE%D0%B2%D0%B0.pdf (дата звернення: 28.04.2026).
20. Кузьмінова О. О. Порівняльний аналіз реляційних та NoSQL баз даних для IoT-систем [Електронний ресурс] // Магістерська дисертація. Київ : КПІ ім. Ігоря Сікорського, 2025. URL: https://ela.kpi.ua/bitstream/123456789/54917/1/Kuzminova_magistr.pdf (дата звернення: 28.04.2026).
21. Ситник Р. С. Моделі і методи організації та забезпечення цілісності даних у реєстрах інформаційних систем [Електронний ресурс] // Дисертація. Дніпро : УДУНТ, 2025. URL: https://ust.edu.ua/wp-content/uploads/2026/02/dysser_sytnyk_compressed.pdf (дата звернення: 28.04.2026).

22. Цируль О. Архітектурні засади організації даних у високоефективних системах управління базами даних типу NoSQL [Електронний ресурс] // Спеціальний випуск. 2025. URL: https://csecurity.kubg.edu.ua/index.php/journal/article/download/974/802 (дата звернення: 28.04.2026).
23. Шпота Т. О. Кваліфікаційна робота бакалавра з теми IoT та бази даних [Електронний ресурс]. Тернопіль : ТНТУ, 2025. URL: https://elartu.tntu.edu.ua/bitstream/lib/49633/1/2025_KRB_SN_43_Shpota%20TO.pdf (дата звернення: 28.04.2026).
24. Андреичук М., Мальська Н., Дмитрук І. Інформаційні технології [Електронний ресурс]. Львів : ЛНУ, 2025. URL: https://geography.lnu.edu.ua/wp-content/uploads/2025/09/Andreychuk_Malska_Dmytruk_Informatsiyni-tekhnolohii_2025.pdf (дата звернення: 28.04.2026).
25. Bondarenko O. Analysis of database systems for IoT applications [Електронний ресурс] // Магістерська дисертація. Київ : КПІ, 2021. URL: https://ela.kpi.ua/bitstream/123456789/45687/1/Bondarenko_magistr.pdf (дата звернення: 28.04.2026).
26. Zaha R. Exploring Data Warehousing Capabilities of NoSQL and SQL Databases // ACM Digital Library. 2024. URL: https://dl.acm.org/doi/full/10.1145/3723178.3723259 (дата звернення: 28.04.2026).
27. Patel A. Evaluating the performance of NoSQL and Time Series Databases [Електронний ресурс]. 2023. URL: http://www.cs.sjsu.edu/faculty/pollett/masters/Semesters/Spring23/aarsh/CS297%20Report.pdf (дата звернення: 28.04.2026).
28. Saquib N. Replicated Versioned Data for the Internet of Things (IoT) [Електронний ресурс]. 2023. URL: https://cs.ucsb.edu/sites/default/files/documents/TR2024-01-Nazmus-Saquib.pdf (дата звернення: 28.04.2026).
29. Lella H. S. Towards Comprehending Energy Consumption of Database Systems // ACM Digital Library. 2024. URL: https://dl.acm.org/doi/full/10.1145/3661167.3661174 (дата звернення: 28.04.2026).
30. García Calatrava C. Introducing Polyglot-Based Data-Flow Awareness to Time-Series Databases [Електронний ресурс]. 2022. URL: https://upcommons.upc.edu/bitstreams/ee73e0e1-59eb-47b0-8d70-9b80621318d9/download (дата звернення: 28.04.2026).
31. OpenAI. Масштабування PostgreSQL і підтримка 800 млн запитів за секунду [Електронний ресурс]. 2026. URL: https://openai.com/uk-UA/index/scaling-postgresql/ (дата звернення: 28.04.2026).
32. ThingsBoard. IoT Platform deployment scenarios with PostgreSQL [Електронний ресурс]. URL: https://thingsboard.io/docs/pe/reference/iot-platform-deployment-scenarios/ (дата звернення: 28.04.2026).

33. Timescale. Benchmarking TimescaleDB vs MongoDB for IoT Time-Series Data [Електронний ресурс]. URL: https://assets.timescale.com/whitepapers/Timescale_WhitePaper_Benchmarking_Mongo.pdf (дата звернення: 28.04.2026).
34. MongoDB. IoT Reference Architecture [Електронний ресурс]. URL: https://www.mongodb.com/resources/solutions/use-cases/iot-reference-architecture (дата звернення: 28.04.2026).
35. MongoDB. Best Practices Guide for MongoDB [Електронний ресурс]. URL: https://www.mongodb.com/resources/products/fundamentals/best-practices-guide-for-mongodb (дата звернення: 28.04.2026).
36. MongoDB. Unlock The Power Of IoT Applications With MongoDB [Електронний ресурс]. URL: https://www.mongodb.com/solutions/use-cases/internet-of-things (дата звернення: 28.04.2026).
37. Engelbert C. PostgreSQL on Kubernetes: Dos and Don’ts for IoT [Електронний ресурс] // Conf42 IoT 2024. URL: https://www.conf42.com/Internet_of_Things_IoT_2024_Chris_Engelbert_kubernetes_postgresql_cluster (дата звернення: 28.04.2026).
38. Paigude S. Optimizing IoT Data Storage: Experience with PostgreSQL and TimescaleDB [Електронний ресурс] // Medium. URL: https://medium.com/@paigude.shweta/optimizing-iot-data-storage-experience-with-postgresql-and-timescaledb-e4dc3dcc7ae7 (дата звернення: 28.04.2026).
39. Information technologies and automation – 2025 : збірник матеріалів конференції. Одеса : ОНТУ, 2025. URL: https://ontu.edu.ua/download/konfi/2025/Collection-of-abstracts-of-the-conference-ITIA-2025.pdf (дата звернення: 28.04.2026).

ДОДАТКИ

1. PostgreSQL + TimescaleDB

```
-- 1. Створення таблиці telemetry-даних
CREATE TABLE sensor_data (
   time         TIMESTAMPTZ NOT NULL,
   device_id    UUID NOT NULL,
   sensor_type  TEXT NOT NULL,
   value        DOUBLE PRECISION,
   unit         TEXT,
   location     GEOGRAPHY,
   attributes   JSONB,
   PRIMARY KEY (device_id, time)
);

-- 2. Перетворення на hypertable
SELECT create_hypertable('sensor_data', 'time',
   chunk_time_interval => INTERVAL '1 day',
   create_default_indexes => true);

-- 3. Додаткові індекси
CREATE INDEX idx_sensor_data_device_time
   ON sensor_data (device_id, time DESC);

CREATE INDEX idx_sensor_data_attributes
   ON sensor_data USING GIN (attributes);

-- 4. Увімкнення стиснення
ALTER TABLE sensor_data
   SET (timescaledb.compress,
       timescaledb.compress_segmentby = 'device_id');

-- 5. Retention policy (зберігати дані лише 2 роки)
SELECT add_retention_policy('sensor_data', INTERVAL '2 years');

-- 6. Continuous aggregate (щогодинні статистики)
CREATE MATERIALIZED VIEW hourly_stats
WITH (timescaledb.continuous) AS
SELECT
   device_id,
```

```
    time_bucket('1 hour', time) AS bucket,
    AVG(value) AS avg_value,
    MAX(value) AS max_value,
    MIN(value) AS min_value
FROM sensor_data
GROUP BY device_id, bucket;

SELECT add_continuous_aggregate_policy('hourly_stats',
    start_offset => INTERVAL '3 days',
    end_offset => INTERVAL '1 hour',
    schedule_interval => INTERVAL '1 hour');
2. MongoDB (Time Series Collections)
// 1. Створення Time Series Collection
db.createCollection("sensor_data", {
  timeseries: {
    timeField: "timestamp",
    metaField: "meta",
    granularity: "seconds"
  },
  expireAfterSeconds: 31536000,   // 1 рік
  clusteredIndex: {
    key: { timestamp: 1 },
    unique: true
  }
});

// 2. Приклад документа, що вставляється
{
  "timestamp": ISODate("2026-04-30T11:04:00.000Z"),
  "meta": {
    "device_id": "dev_ua_kr_001",
    "sensor_type": "temperature_humidity",
    "location": { "type": "Point", "coordinates": [33.45, 48.52] },
    "tenant_id": "agro_farm_42"
  },
  "measurements": {
    "temperature": 23.7,
    "humidity": 45.2,
    "pressure": 1013.25
  },
```

```
  "attributes": {
   "battery_level": 87,
   "signal_strength": -65
  }
 }

 // 3. Приклад Aggregation Framework (щогодинна статистика)
 db.sensor_data.aggregate([
  {
   $match: {
    "meta.device_id": "dev_ua_kr_001",
    timestamp: { $gte: new Date("2026-04-01") }
   }
  },
  {
   $group: {
    _id: { $dateTrunc: { date: "$timestamp", unit: "hour" } },
    avg_temp: { $avg: "$measurements.temperature" },
    max_temp: { $max: "$measurements.temperature" },
    min_temp: { $min: "$measurements.temperature" }
   }
  },
  { $sort: { _id: 1 } }
 ]);
 3. Приклад вставки даних у PostgreSQL (TimescaleDB)
 INSERT INTO sensor_data (time, device_id, sensor_type, value, unit, attributes)
 VALUES
    (NOW(), 'a1b2c3d4-e5f6-7890-abcd-ef1234567890', 'temperature', 23.7, '°C',
     '{"battery": 87, "signal": -65}'::jsonb);
 4. Приклад вставки даних у MongoDB
 db.sensor_data.insertOne({
  timestamp: new Date(),
  meta: { device_id: "dev_ua_kr_001", sensor_type: "temperature" },
  measurements: { temperature: 23.7 }
});
```