ТОВ «ТЕХНІЧНИЙ УНІВЕРСИТЕТ «МЕТІНВЕСТ ПОЛІТЕХНІКА»
Факультет автоматизації виробництва, інформаційних
та управлінських технологій
Кафедра інформаційних технологій та аналітики даних

«Допущено до захисту»
Гарант ОПП
Ірина ГЕТЬМАН

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавра

за підсумками виконання
освітньо-професійної програми
«Комп'ютерні науки»
за спеціальністю 122 Комп'ютерні науки

на тему «Програмно-методичний комплекс для автоматизації обробки даних про ефективність роботи сучасних систем управління базами даних»

Керівник роботи Павло САГАЙДА

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Здобувач Андрій СОРОКОПУД

Підсумкова оцінка за атестацію			

Голова ЕК Антон КУДРЯВЦЕВ

ЗАПОРІЖЖЯ 2026

ТОВ «ТЕХНІЧНИЙ УНІВЕРСИТЕТ «МЕТІНВЕСТ ПОЛІТЕХНІКА»

Факультет	автоматизації виробництва, інформаційних та управлінських технологій
Кафедра	інформаційних технологій та аналітики даних
Ступінь вищої освіти	бакалавр
Спеціальність	122 Комп'ютерні науки
ОПП	Комп'ютерні науки

ЗАТВЕРДЖУЮ
Гарант ОПП

_____ Ірина ГЕТЬМАН

«26» лютого 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА
Сорокопуда Андрія Павловича

(прізвище, ім'я, по батькові здобувача)

1. Тема роботи: «Програмно-методичний комплекс для автоматизації обробки даних про ефективність роботи сучасних систем управління базами даних»
керівник роботи Сагайда Павло Іванович, доцент, доктор техн. наук,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом Університету від 23.02.2026 р. №41/23.02.2026
2. Термін подання роботи 16.06.2026 р.
3. Вихідні дані до роботи: Навчальна література, державні стандарти та методичні *рекомендації, наукові та науково-технічні публікації з ефективності роботи сучасних систем управління базами даних, дослідження в галузі побудови інформаційних систем, результати власного проєктування, тестування, експериментальних розрахунків та економічного обґрунтування впровадження програмного комплексу.*
4. Зміст пояснювальної записки (перелік питань): *Реферат. Вступ. Розділ 1. Аналіз предметної області, сучасних принципів моделювання. Обґрунтування актуальності досліджень в сфері Інформаційних технологій адміністрування та моніторингу баз даних. Розділ 2. Моделювання процесів автоматизації моніторингу баз даних. Розділ 3. Розробка проєкту програмно-методичного комплексу та його реалізація. Розділ 4. Економічне обґрунтування розробки та впровадження програмного-методичного комплексу. Висновки. Перелік використаних джерел. Додатки.*
5. Перелік графічного (демонстраційного) матеріалу (з точним зазначенням обов'язкових креслень): *Актуальність, мета, об'єкт, предмет та завдання дослідження; результати аналізу предметної області диджиталізації та їх формалізації за допомогою діаграмних методик; структура бази даних; результати логічного проєктування у вигляді діаграм UML (прецедентів, класів, послідовностей, або інших, за необхідності); форми користувальницького інтерфейсу; результати розрахунків витрат та терміну окупності системи; висновки до роботи; (за наявності) – копії публікацій або розробленої документації.*

6. Консультанти по роботі, із зазначенням розділів роботи, що стосуються їх.

Розділ	Прізвище, ініціали та посада консультанта
1	Сагайда П.І., доктор технічних наук, доцент каф. ІНТЕХАД
2	Сагайда П.І., доктор технічних наук, доцент каф. ІНТЕХАД
3	Сагайда П.І., доктор технічних наук, доцент каф. ІНТЕХАД
4	Гетьман І.А., кандидат технічних наук, доцент каф. ІНТЕХАД

7. Дата видачі завдання 26.02.2026

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи
1	Аналіз літературних джерел за темою дипломної роботи, вступ, зміст	26.02.2026 – 10.03.2026
2	Розділ 1. Аналіз предметної області, сучасних принципів моделювання. Обґрунтування актуальності досліджень в сфері Інформаційних технологій адміністрування та моніторингу баз даних	11.03.2026 – 27.03.2026
3	Розділ 2. Моделювання процесів автоматизації моніторингу баз даних.	28.03.2026 – 08.04.2026
4	Розділ 3. Розробка проєкту програмно-методичного комплексу та його реалізація.	09.04.2026 – 30.05.2026
5	Розділ 4. Економічне обґрунтування розробки та впровадження програмного-методичного комплексу. Висновки. Перелік використаних джерел. Додатки.	30.05.2026-12.06.2026
6	Висновки, перелік посилань, вступ, зміст, реферат	12.06.2026-15.06.2026
7	Подання завершеної роботи. Перевірка на академічний плагіат	15.06.2026-16.06.2026
8	Остаточне оформлення роботи, презентаційного матеріалу	16.06.2026 –18.06.2026
9	Рецензування завершеної роботи. Захист	18.06.2026 – 23.06.2026

Здобувач (Андрій Сорокопуд)

Керівник роботи (Павло Сагайда)

РЕФЕРАТ

Сорокопуд А.П. **Програмно-методичний комплекс для автоматизації обробки даних про ефективність роботи сучасних систем управління базами даних.**

Кваліфікаційна робота на здобуття освітнього ступеня бакалавра комп'ютерних наук за спеціальністю 122 «Комп'ютерні науки». – ТОВ «ТЕХНІЧНИЙ УНІВЕРСИТЕТ «МЕТІНВЕСТ ПОЛІТЕХНІКА» МОН України, м. Запоріжжя, 2026.

Метою роботи було підвищення ефективності адміністрування інформаційної інфраструктури шляхом розробки програмно-методичного комплексу для автоматизованого безагентського збору, аналізу та візуалізації телеметричних даних сучасних систем управління базами даних.

Об'єктом дослідження є процес моніторингу, діагностики та управління продуктивністю реляційних систем управління базами даних.

Методологія – системний аналіз, стандарт IDEF0, моделювання UML, теорія масового обслуговування, алгоритми експоненційного згладжування, об'єктно-орієнтоване програмування, проєктування баз даних, економічна оцінка розробки програмного забезпечення.

У цій роботі проаналізовано архітектурні особливості функціонування СУБД PostgreSQL і MS SQL Server, розроблено технічні вимоги та побудовано логічну і математичну моделі процесів збору телеметрії.

Програмний продукт реалізовано на мові програмування C# (платформа .NET) у середовищі Visual Studio та використовує базу даних SQLite. Реалізовано зручний інтерфейс користувача WPF, дашборди реального часу, механізми налаштування порогів чутливості, алгоритм «вікна толерантності» та журнал інцидентів.

Було проведено техніко-економічне обґрунтування впровадження розробленої системи. Розроблене програмне забезпечення може підвищити продуктивність роботи адміністраторів баз даних, зменшити кількість хибних спрацювань системи сповіщень, забезпечити швидкий доступ до аналітики та скоротити фінансові втрати компанії від простою серверів. Систему можна адаптувати для моніторингу інших сучасних СУБД завдяки гнучкій безагентській архітектурі.

КЛЮЧОВІ СЛОВА: АВТОМАТИЗАЦІЯ, МОНІТОРИНГ, СИСТЕМА УПРАВЛІННЯ БАЗАМИ ДАНИХ, POSTGRESQL, MS SQL SERVER, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, VISUAL STUDIO, C#, SQLITE, ТЕЛЕМЕТРІЯ, ЕКОНОМІЧНА ДОЦІЛЬНІСТЬ

ABSTRACT

Sorokopud A.P. **Software and methodological complex for automating the processing of data on the efficiency of modern database management systems.**

Qualification work for the bachelor's degree in computer science, specialty 122 «Computer Science». – LLC «TECHNICAL UNIVERSITY «METINVEST POLYTECHNIC» of the Ministry of Education and Science of Ukraine, Zaporizhzhia, 2026.

The purpose of the work was to create software for automated collection, analysis, and visualization of DBMS telemetry data, which will improve the efficiency of performance control of the information infrastructure.

The object of research is the process of monitoring, diagnostics, and performance management of relational database management systems.

Methodology – system analysis, IDEF0 standard, UML modeling, queuing theory, exponential smoothing algorithms (EWMA), object-oriented programming, database design, economic evaluation of software development.

This work analyzes the architectural features of PostgreSQL and MS SQL Server, develops technical requirements, and builds logical and mathematical models of telemetry collection processes.

The software product is implemented in the C# programming language (.NET platform) in the Visual Studio environment and uses the SQLite database. A convenient graphical user interface WPF, real-time dashboards, threshold setting mechanisms, a «tolerance window» algorithm, and an incident log have been implemented.

A technical and economic justification for the implementation of the developed system was conducted. The developed software can increase the productivity of database administrators, reduce the number of false alerts, provide quick access to analytics, and reduce financial losses of the company from server downtime. The system can be adapted to monitor other modern DBMS due to its flexible agentless architecture.

KEYWORDS: AUTOMATION, MONITORING, DATABASE MANAGEMENT SYSTEM, POSTGRESQL, MS SQL SERVER, SOFTWARE, VISUAL STUDIO, C#, SQLITE, TELEMETRY, ECONOMIC FEASIBILITY.

ЗМІСТ

ВСТУП

У сучасних умовах глобальної цифрової трансформації та переходу до економіки, керованої даними, інформаційні технології відіграють ключову роль у забезпеченні безперервності бізнес-процесів підприємств. Фундаментальною основою будь-якої корпоративної інформаційної системи є системи управління базами даних, які забезпечують надійне зберігання, швидку обробку та безпечний доступ до критично важливої інформації. Серед розмаїття сучасних рішень домінуючі позиції на ринку корпоративного програмного забезпечення займають реляційні СУБД, зокрема комерційна платформа Microsoft SQL Server та відкрита система PostgreSQL.

Стрімке зростання обсягів даних (Big Data) та перехід до мікросервісних архітектур призводять до експоненціального збільшення транзакційного навантаження на сервери баз даних. В умовах високонавантажених середовищ (HighLoad) жорстка конкуренція за апаратні ресурси неминуче провокує виникнення інцидентів продуктивності: взаємних блокувань (Deadlocks), вичерпання пулу з'єднань або критичного падіння ефективності буферного кешу. Навіть короткочасна деградація швидкодії СУБД здатна паралізувати роботу суміжних цифрових сервісів, що призводить до значних фінансових та репутаційних втрат.

Традиційні підходи до адміністрування, засновані на ручному виконанні діагностичних запитів та постфактум-аналізі системних журналів, є неефективними, оскільки не дозволяють виявляти проблеми на ранніх стадіях. З іншого боку, впровадження комерційних Enterprise-рішень для моніторингу часто ускладнюється їхньою високою вартістю, надлишковою складністю розгортання та необхідністю встановлення спеціальних агентів на цільові сервери, що може суперечити корпоративним політикам безпеки.

З огляду на це, розробка власного програмно-методичного комплексу для автоматизованого безагентського моніторингу, аналізу та предиктивного

виявлення аномалій у роботі реляційних СУБД є надзвичайно актуальним науково-практичним завданням.

Метою роботи є підвищення ефективності адміністрування інформаційної інфраструктури шляхом розробки програмно-методичного комплексу для автоматизованого безагентського збору, аналізу та візуалізації телеметричних даних сучасних систем управління базами даних.

Для досягнення поставленої мети було сформульовано та вирішено такі завдання:

1. Провести аналіз предметної області, дослідити архітектурні особливості функціонування сучасних СУБД (PostgreSQL та Microsoft SQL Server) та виявити ключові проблеми їх моніторингу.
2. Побудувати логічні, математичні та функціональні моделі процесів збору й обробки телеметрії з використанням методологій IDEF0 та стандарту UML.
3. Розробити алгоритмічне забезпечення підсистеми сповіщень із застосуванням теорії масового обслуговування та статистичних методів фільтрації даних.
4. Здійснити безпосередню програмну реалізацію комплексу мовою C# на платформі .NET з використанням технологій WPF та Entity Framework Core.
5. Провести експериментальні дослідження та навантажувальне тестування розробленого програмного забезпечення для підтвердження його працездатності.
6. Виконати техніко-економічне обґрунтування доцільності розробки та впровадження програмно-методичного комплексу.

Об’єкт дослідження – процес моніторингу, діагностики та управління продуктивністю реляційних систем управління базами даних.

Предмет дослідження – методи, алгоритми та програмні засоби автоматизованого збору, аналізу й візуалізації телеметричних даних СУБД.

Методи дослідження. Для розв'язання поставлених завдань використано комплекс сучасних наукових та інженерних методів: системний аналіз та методологію SADT (стандарт IDEF0) – для декомпозиції процесів обробки даних; об’єктно-орієнтоване проєктування (UML) – для побудови

архітектури програмного забезпечення; теорію масового обслуговування (закон Літтла) – для математичного моделювання черг запитів; алгоритм експоненційного згладжування (EWMA) – для аналізу часових рядів; методи навантажувального тестування – для експериментальної перевірки комплексу. Оцінка доцільності впровадження спирається на методи економічного аналізу (TCO, ROI, SWOT-аналіз).

Наукова новизна одержаних результатів полягає в удосконаленні підходу до виявлення інцидентів продуктивності реляційних баз даних шляхом інтеграції алгоритму експоненційного згладжування (EWMA) з механізмом «вікна толерантності». На відміну від традиційних систем моніторингу з жорсткими пороговими значеннями, запропонований комплексний підхід дозволяє нівелювати вплив мікросекундних стрибків навантаження, що суттєво знижує ймовірність хибних спрацювань підсистеми сповіщень.

Практичне значення одержаних результатів**.** Розроблено повністю функціональний десктопний програмно-методичний комплекс для адміністраторів баз даних. Впровадження програмно-методичного комплексу дозволяє здійснювати централізований безагентський моніторинг серверів PostgreSQL та MS SQL Server, накопичувати історичну телеметрію в локальній базі даних SQLite, візуалізувати продуктивність у режимі реального часу та оперативно реагувати на загрози, що гарантує підвищення стабільності інформаційної інфраструктури.

*Особистий внесок здобувача***.** Усі результати, викладені у кваліфікаційній роботі, отримані автором самостійно. Здобувачем особисто виконано аналіз предметної області, побудовано архітектурні моделі, реалізовано вихідний код програмно-методичного комплексу, проведено його тестування та економічне обґрунтування.

Структура та обсяг роботи. Кваліфікаційна робота складається зі вступу, чотирьох розділів, загальних висновків, списку використаних джерел із 54 найменувань та 3 додатків (Технічне завдання, Інструкція адміністратора та перелік та відбитки наукових публікацій за темою роботи). Загальний обсяг роботи становить 104 сторінок тексту, включаючи 30 рисунків та 9 таблиць.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ, СУЧАСНИХ ПРИНЦИПІВ МОДЕЛЮВАННЯ. ОБҐРУНТУВАННЯ АКТУАЛЬНОСТІ ДОСЛІДЖЕНЬ В СФЕРІ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ АДМІНІСТРУВАННЯ ТА МОНІТОРИНГУ БАЗ ДАНИХ

1.1 Обґрунтування актуальності досліджень

Сучасний етап розвитку світової економіки та суспільства характеризується тотальною цифровою трансформацією. Перехід до парадигми економіки, керованої даними (Data-Driven Economy), та впровадження концепцій Індустрії 4.0 кардинально змінили роль інформаційних технологій [1]. Якщо раніше IT-інфраструктура виконувала переважно допоміжні функції підтримки бізнесу, то сьогодні вона є його фундаментальним стрижнем. Інформація стала найціннішим активом, а системи управління базами даних (СУБД) – головним механізмом забезпечення життєздатності будь-якої організаційно-технічної системи.

Еволюція програмного забезпечення протягом останнього десятиліття призвела до експоненціального зростання обсягів даних, що генеруються та обробляються в реальному часі, їх ще називають Big Data. Зміна архітектурних підходів – перехід від монолітних додатків до мікросервісних та хмарних середовищ – призвела до значного ускладнення IT-інфраструктури [2]. У таких умовах забезпечення безперервної доступності, цілісності та високої продуктивності систем обробки даних перетворилося на одну з найскладніших інженерних задач.

Будь-яка сучасна інформаційна система працює в умовах жорстких обмежень апаратних ресурсів та постійної конкуренції за них. Зростання інтенсивності транзакційного навантаження неминуче призводить до деградації продуктивності таких систем. Традиційні парадигми ручного

системного адміністрування, коли фахівець реагував на проблему лише після її виникнення (реактивний підхід), виявилися абсолютно нежиттєздатними в умовах високонавантажених систем, які вимагають роботи в режимі 24/7/365 [3].

На зміну ручному управлінню приходять концепції автоматизації інфраструктури (Infrastructure as Code, DataOps) [4]. Сучасне адміністрування вимагає впровадження інтелектуальних систем, здатних самостійно збирати телеметрію, аналізувати поведінку системи та прогнозувати можливі відмови. На мою думку, дослідження інформаційних технологій адміністрування та розробка спеціалізованого програмно-методичного комплексу для автоматизації обробки даних про ефективність роботи СУБД є надзвичайно актуальним науковим та інженерним завданням.

1.2 Аналіз предметної області «Інформаційні технології адміністрування та моніторингу баз даних»

Предметною областю мого дослідження є інформаційні технології адміністрування та моніторингу баз даних. Це широка міждисциплінарна сфера, яка охоплює методології управління життєвим циклом даних, алгоритми оптимізації доступу до інформації, методи забезпечення інформаційної безпеки та технології безперервного спостереження за станом апаратно-програмних комплексів [5].

Адміністрування баз даних є багатогранним процесом. Історично воно включало завдання розгортання серверів, створення резервних копій (Backup & Recovery), управління правами доступу та проєктування фізичної схеми зберігання даних. Проте в сучасних умовах домінуючою складовою адміністрування стало управління продуктивністю. Реляційна СУБД – це надзвичайно складна програмна система, яка самостійно виконує функції

операційної системи на мікрорівні: вона має власний диспетчер пам’яті, планувальник потоків, механізми управління кешем та підсистему вводу-виводу [6].

У межах обраної мною предметної області, я би виділив ключове поняття «ефективність функціонування». Ефективність роботи СУБД не є абстрактною величиною; вона вимірюється двома фундаментальними метриками:

1. Пропускна здатність (Throughput): кількість транзакцій або запитів, які система здатна успішно обробити за одиницю часу (наприклад, Transactions Per Second).
2. Затримка (Latency): час, необхідний системі для обробки одного запиту від моменту його надходження до моменту повернення результату клієнту.

Досягнення високої ефективності ускладнюється жорсткими вимогами до цілісності даних. Усі промислові реляційні СУБД підпорядковуються принципам ACID (Atomicity, Consistency, Isolation, Durability) [7]. Необхідність забезпечення ізоляції (Isolation) змушує систему застосовувати механізми конкурентного доступу – блокування (Locks) та засувки (Latches). А вимога довговічності (Durability) вимагає обов’язкового синхронного запису кожної транзакції у фізичний журнал випереджаючого запису (Write-Ahead Log, WAL), що створює колосальне навантаження на дискову підсистему.

Саме тут предметна область адміністрування перетинається з технологіями моніторингу. Моніторинг виступає «сенсорною системою» процесу управління. Неможливо оптимізувати те, що не вимірюється. Збір телеметричних даних є єдиним об’єктивним способом зрозуміти внутрішній стан СУБД, виявити вузькі місця та ухвалити обґрунтоване інженерне рішення щодо налаштування (Performance Tuning).

1.3 Аналіз сучасних систем управління базами даних та їх об’єктами моніторингу

Сучасний корпоративний IT-ландшафт характеризується концепцією «Polyglot Persistence» – одночасним використанням різних типів баз даних для вирішення різних класів завдань [8]. Інформаційні системи еволюціонували від єдиної монолітної бази даних до складних гетерогенних середовищ, де кожна СУБД підбирається під специфічні вимоги до структури даних, швидкості доступу та масштабованості.

Розмаїття сучасних систем управління даними вимагає від інструментів моніторингу гнучкості, оскільки архітектура кожної СУБД формує унікальний набір метрик, що підлягають контролю. Для розуміння складності предметної області розглянемо специфіку моніторингу ряду поширених на ринку систем:

1. *Oracle Database*. (рис. 1.1). Лідер серед комерційних реляційних систем рівня Enterprise. Має надзвичайно складну багаторівневу структуру пам'яті (SGA та PGA). Моніторинг цієї СУБД вимагає глибокого аналізу десятків системних представлень, контролю за фрагментацією табличних просторів (Tablespaces) та відстеження статистики подій очікування (Wait Events) [9].

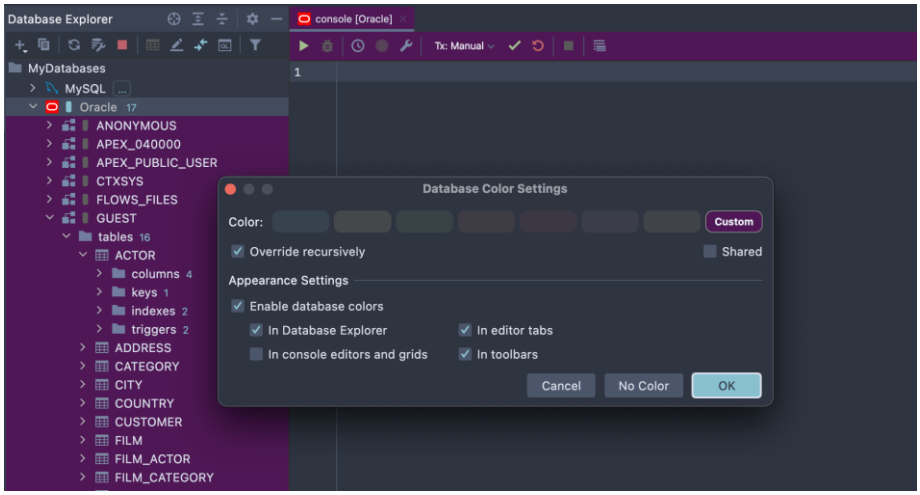


Рисунок 1.1 – Фрагмент бази даних Oracle [10]

2. *MySQL*. (рис 1.2). Найпопулярніша у світі відкрита реляційна СУБД, оптимізована для веб-додатків. Її архітектурною особливістю є використання змінних підсистем зберігання, найпоширенішою з яких є *InnoDB*. При моніторингу MySQL критично важливим є відстеження ефективності буферного пулу (InnoDB Buffer Pool Hit Rate) та аналіз черг потоків (Thread Cache), оскільки система чутлива до великої кількості дрібних одночасних з'єднань [11].

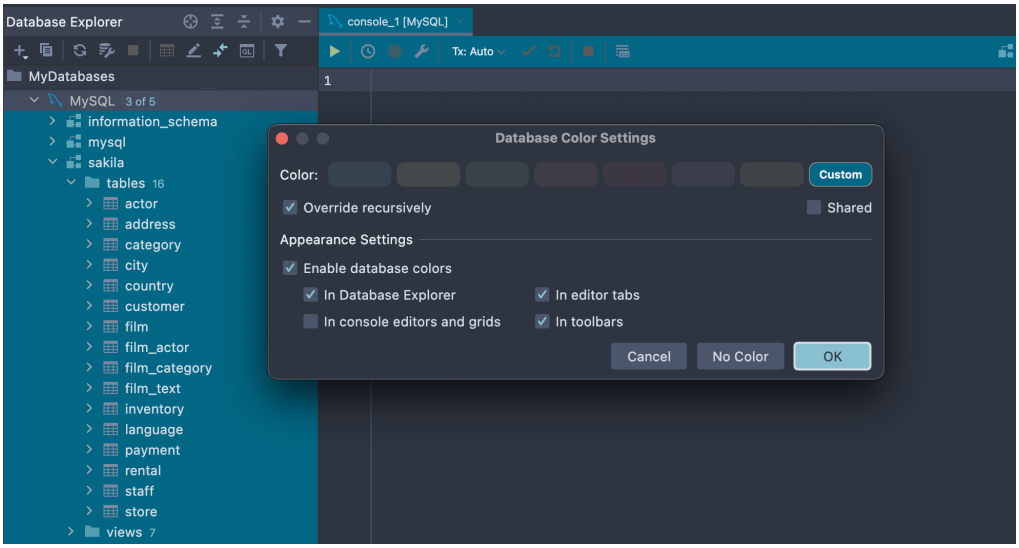


Рисунок 1.2 – Фрагмент бази даних MySQL [12]

3. *MongoDB*. (рис 1.3). Представник сімейства NoSQL (документо-орієнтована СУБД). Зберігає дані у форматі BSON без жорсткої схеми. Оскільки MongoDB агресивно використовує оперативну пам'ять для кешування робочих наборів, так званих Working Set, фокус моніторингу зміщується з блокувань рядків на утилізацію RAM, кількість помилок сторінок пам'яті (Page Faults) та затримки між вузлами реплікації (Replication Lag) [13].

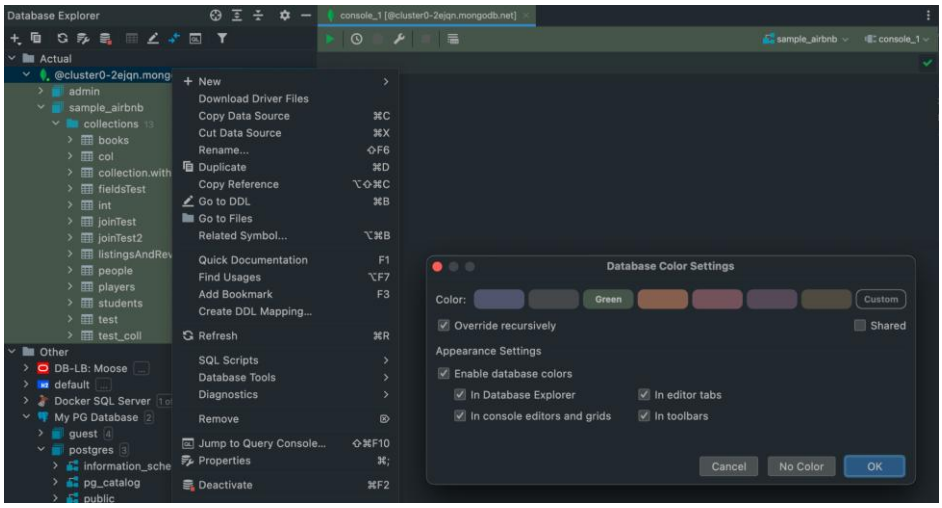


Рисунок 1.3 – Фрагмент бази даних MongoDB [14]

4. *Redis*. (Рис 1.4). Високопродуктивна In-Memory СУБД класу «ключ-значення» (Key-Value), що працює виключно в оперативній пам'яті. Використовується переважно як шар кешування. Предметна область моніторингу тут кардинально інша: критично важливо контролювати обсяг зайнятої пам'яті (used_memory), щоб уникнути аварійного завершення процесу (OOM), а також показник влучання в кеш (Hit Rate) та алгоритми витіснення старих ключів (Eviction) [15].

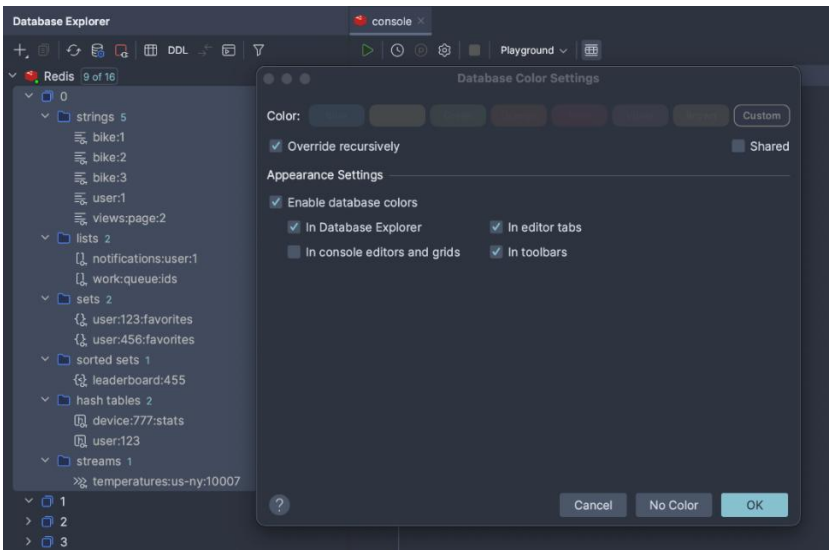


Рисунок 1.4 – Фрагмент бази даних Redis [16]

5. *Microsoft SQL Server.* (Рис 1.5). Побудований на потоковій моделі (Thread-based architecture). Відповідно до цієї парадигми, СУБД працює як єдиний великий процес операційної системи Windows, всередині якого SQLOS (міні-ОС всередині SQL Server) керує тисячами потоків [17]. З огляду на це, адміністрування та моніторинг цієї системи нерозривно пов'язані з глобальними показниками: аналізом обсягу зарезервованої оперативної пам'яті (Target/Total Server Memory), процесорного часу та контролем файлів журналу транзакцій (LDF). Відстеження заповненості логів є критично важливим, оскільки їх переповнення призводить до повної зупинки роботи бази [18].

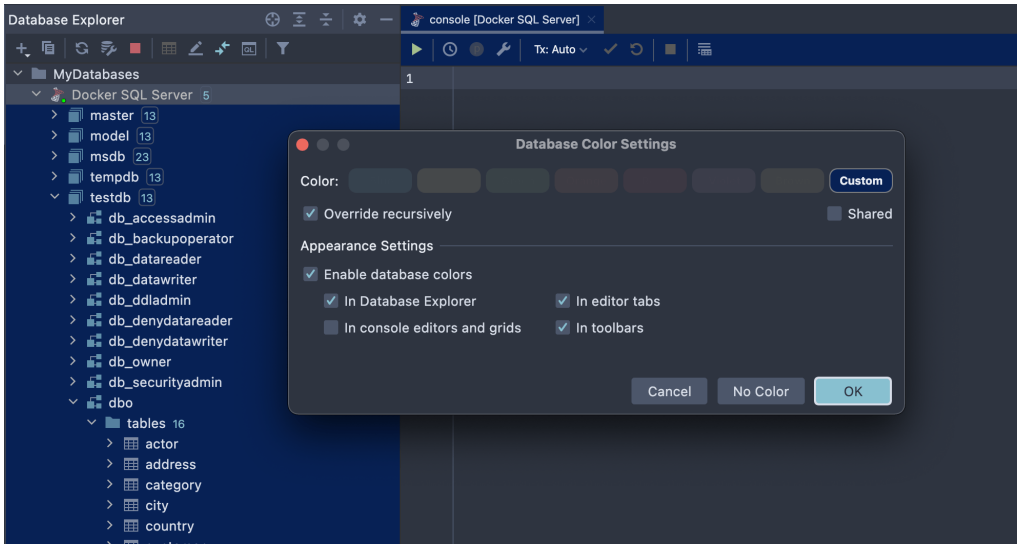


Рисунок 1.5 – Фрагмент бази даних MS SQL [19]

6. *PostgreSQL.* (Рис 1.6). Використовує процесну модель (Process-per-connection). Кожен клієнтський запит обслуговується окремим системним процесом. Більше того, PostgreSQL активно делегує управління кешем сторінок самій операційній системі (OS Page Cache) [20]. Головною особливістю цієї СУБД, є використання багатоверсійного контролю конкурентності (MVCC) [21]. Замість жорстких блокувань під час читання/запису, система створює нові версії змінених рядків. Застарілі версії

(«мертві рядки» або Dead Tuples) накопичуються у фізичних файлах таблиць. Якщо їх вчасно не видаляти фоновим процесом (Autovacuum), відбувається катастрофічна деградація продуктивності – так зване розбухання таблиць (Table Bloat). [22].

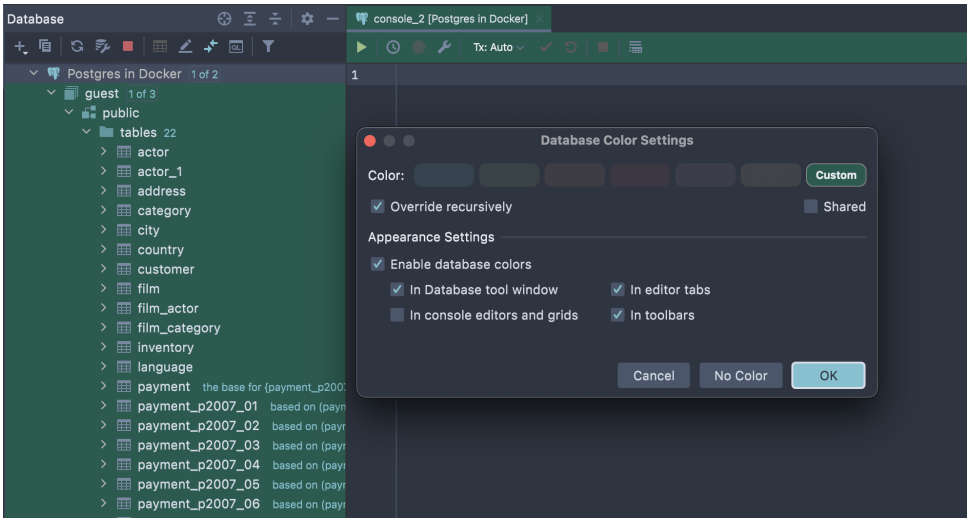


Рисунок 1.6 – Фрагмент бази даних PostgreSQL

Узагальнена інформація щодо архітектурних особливостей та пріоритетних напрямів моніторингу сучасних СУБД наведена у порівняльній таблиці 1.1.

Таблиця 1.1 – Порівняльна таблиця сучасних СУБД

Назва СУБД	Модель даних	Особливості архітектури	Ключові об'єкти моніторингу (Метрики)
Oracle	Реляційна (RDBMS)	Складна структура пам'яті, жорстка прив'язка до дискових структур.	Утилізація Shared Pool, Wait Events, Redo Logs, Tablespaces.
MySQL	Реляційна (RDBMS)	Під'єднувані рушії зберігання (переважно InnoDB).	InnoDB Buffer Pool, Thread Cache, Table/Row Locks.
MongoDB	Документна (NoSQL)	BSON-сховище, рушій WiredTiger, вбудований шардинг.	Page faults, обсяг кешу WiredTiger, Replication Lag.

Продовження таблиці 1.1

Назва СУБД	Модель даних	Особливості архітектури	Ключові об'єкти моніторингу (Метрики)
Redis	Ключ-значення	Однопотокова In-Memory архітектура.	Used Memory, Evicted Keys, Cache Hit Ratio.
Microsoft SQL Server	Реляційна (RDBMS)	Потокова (Thread-based), інтегрована з ОС Windows (SQLOS).	CPU/RAM утилізація, Log Space Used, Active/Blocked Sessions.
PostgreSQL	Реляційна (RDBMS)	Процесна (Process-per-connection), механізм MVCC.	Dead Tuples, Connection Limit, Buffer Cache Hit Ratio, Autovacuum.

Незважаючи на колосальне розмаїття NoSQL та In-Memory рішень, фундаментальною основою транзакційної логіки більшості корпоративних додатків залишаються традиційні реляційні системи (RDBMS).

У межах даної кваліфікаційної роботи для розробки та апробації програмно-методичного комплексу було прийнято рішення зосередитись на двох провідних реляційних системах: комерційній Microsoft SQL Server та відкритій PostgreSQL. Вибір саме цих двох систем не є випадковим: вони є беззаперечними лідерами у своїх сегментах, але при цьому мають діаметрально протилежну архітектуру.

Як зазначалося в попередніх абзацах, Microsoft SQL Server побудований на потоковій моделі (Thread-based architecture). Відповідно до цієї парадигми, СУБД працює як єдиний великий процес операційної системи Windows, всередині якого SQLOS (міні-ОС всередині SQL Server) керує тисячами потоків. З огляду на це, адміністрування та моніторинг цієї системи нерозривно пов'язані з глобальними показниками: аналізом обсягу зарезервованої оперативної пам'яті (Target/Total Server Memory), процесорного часу та контролем файлів журналу транзакцій (LDF). Відстеження заповненості логів є критично важливим, оскільки їх переповнення призводить до повної зупинки роботи бази.

На противагу, PostgreSQL використовує процесну модель (Process-per-connection). Кожен клієнтський запит обслуговується окремим системним процесом. Більше того, PostgreSQL активно делегує управління кешем сторінок самій операційній системі (OS Page Cache). Головною особливістю цієї СУБД, що формує унікальний сегмент предметної області, є використання багатоверсійного контролю конкурентності (MVCC). Замість жорстких блокувань під час читання/запису, система створює нові версії змінених рядків. Застарілі версії («мертві рядки» або Dead Tuples) накопичуються у фізичних файлах таблиць. Якщо їх вчасно не видаляти фоновим процесом (Autovacuum), відбувається катастрофічна деградація продуктивності – «розбухання» таблиць (Table Bloat). Відповідно, моніторинг життєвого циклу мертвих рядків та роботи Autovacuum є наріжним каменем адміністрування PostgreSQL.

Цей аналіз доводить, що створення універсальних засобів автоматизації є складним системним завданням. Вирішення проблеми моніторингу для двох настільки архітектурно різних систем (потокової MS SQL та процесної PostgreSQL з MVCC) дозволить закласти в розроблюваний програмно-методичний комплекс максимально гнучку абстрактну модель збору даних, яку в подальшому я зможу легко масштабувати на будь-які інші СУБД.

1.4 Сучасні принципи моделювання в задачах адміністрування та моніторингу

Розробка засобів автоматизації обробки даних у сфері адміністрування вимагає застосування суворого наукового підходу та методів моделювання. Це дозволяє формалізувати складні процеси, що відбуваються в СУБД, та перевести їх на мову алгоритмів.

Математичне моделювання поведінки систем. Функціонування бази даних найкраще описується за допомогою теорії масового обслуговування. СУБД моделюється як багатоканальна система масового обслуговування з очікуванням [23]. У цій моделі вхідним потоком (найчастіше пуассонівським) виступають SQL-запити. Сервісними каналами є доступні ресурси процесора та робочі потоки СУБД. Реєстрація зростання активних сесій (Active Sessions) математично моделює збільшення довжини черги. Аналіз такої моделі дає змогу виявити межі пропускної здатності системи та науково обґрунтувати пороги (Thresholds), перевищення яких свідчить про наближення системи до стану відмови.

Логічне моделювання підсистеми сповіщень *(Alerting Engine).* Враховуючи динамічну природу сучасних додатків, короткочасні стрибки навантаження є нормальним явищем. Використання статичних математичних порогів призводить до генерації хибних тривог. Тому сучасні системи адміністрування моделюють логіку сповіщень за допомогою кінцевих автоматів із часовою витримкою (Tolerance). Стан об'єкта переходить у фазу аварії (Critical Alert) лише тоді, коли відхилення фіксується неперервно протягом заздалегідь визначеного часового вікна Δt [24]. Це дозволяє системі моніторингу фільтрувати «інформаційний шум» та реагувати лише на сталі деструктивні тренди.

Інформаційне моделювання обробки телеметрії. Збір даних про стан інфраструктури формує масиви часових рядів. Інформаційна модель програмно-методичного комплексу повинна забезпечувати цілісне зберігання цих метрик з мінімальними затримками. Для локальних систем моніторингу доцільним є проєктування нормалізованої реляційної схеми даних, що описує сутності «Сервер», «Інстанс бази даних», «Метрика» та «Подія сповіщення». Це забезпечує можливість глибокого ретроспективного аналізу для розслідування інцидентів.

Системне моделювання архітектури ПЗ. Згідно зі світовими стандартами програмної інженерії, формалізація вимог до розроблюваного

комплексу здійснюється з використанням методології об'єктно-орієнтованого моделювання на базі UML. Діаграми прецедентів (Use Case) дозволяють чітко окреслити межі системи та варіанти її використання адміністратором (від налаштування підключень до візуального аналізу трендів продуктивності) [25]. Моделювання бізнес-процесів за допомогою діаграм діяльності (Activity Diagrams) дає змогу алгоритмізувати складні фонові процеси циклічного збору метрик [26].

1.5 Методології концептуального аналізу продуктивності інформаційних систем

У світовій практиці системної інженерії та адміністрування баз даних для стандартизації процесів моніторингу використовуються загальноприйняті методології. Вони дозволяють структурувати процес збору телеметрії та уникнути ситуації «сліпих зон», коли критично важливий компонент системи залишається без нагляду. Найбільш релевантними для предметної області моніторингу СУБД є методології USE та RED.

Методологія USE (Utilization, Saturation, Errors) Ця методологія, розроблена інженером Бренданом Греггом, орієнтована на аналіз апаратних та системних ресурсів. Її головний принцип: для кожного ресурсу (CPU, пам'ять, диск, мережа) необхідно перевіряти три показники [27]:

1. Utilization (Утилізація): середній час, протягом якого ресурс був зайнятий виконанням корисної роботи. Наприклад, використання CPU на 90% або заповнення дискового простору бази даних на 85%.
2. Saturation (Насичення): ступінь наявності додаткової роботи, яку ресурс не може обробити негайно, що призводить до формування черг. У контексті реляційних СУБД яскравим прикладом насичення є зростання

показника Disk Queue Length (довжина черги до диска) або накопичення активних сесій, що очікують на звільнення пулу потоків.

3. Errors (Помилки): кількість подій, що завершилися з помилкою. Для СУБД це можуть бути розірвані з'єднання (Connection Timeouts), помилки транзакцій (Deadlock Victim) або апаратні збої сторінок пам'яті.

Методологія RED (Rate, Errors, Duration) Якщо методологія USE фокусується на ресурсах, то RED орієнтована на рівень сервісу та користувацький досвід. Вона ідеально підходить для оцінки ефективності обробки SQL-запитів [28]:

- Rate (Інтенсивність): кількість запитів, що надходять до СУБД за секунду (Transactions/Queries per Second).
- Errors (Помилки): кількість запитів, що не вдалося виконати (наприклад, через синтаксичні помилки або порушення обмежень цілісності).
- Duration (Тривалість): час виконання запиту. У системах моніторингу цей показник часто аналізується за допомогою перцентилів (наприклад, p95 або p99), що дозволяє відкинути аномально швидкі чи повільні запити і побачити реальну картину швидкодії для більшості користувачів.

Інтеграція підходів USE та RED у логіку роботи програмно-методичного комплексу дозволяє створити всебічну картину стану СУБД: від фізичного навантаження на сервери до якості обслуговування клієнтських додатків.

1.6 Класифікація типових антипатернів та аномалій продуктивності СУБД

Предметна область автоматизації моніторингу невідривно пов'язана з поняттям антипатернів проєктування та експлуатації баз даних. Антипатерни – це розповсюджені, але неефективні підходи до вирішення задач, які в довгостроковій перспективі призводять до деградації системи.

Автоматизований збір метрик спрямований на виявлення наслідків саме таких аномалій [29].

До найбільш критичних антипатернів, які фіксуються системами моніторингу, належать:

1. Відсутність або надмірність індексів (Missing / Redundant Indexes).

Індексування є ключовим механізмом прискорення читання даних. Відсутність індексу для часто виконуваного запиту змушує СУБД виконувати повне сканування таблиці (Full Table Scan). Це спричиняє колосальне навантаження на дискову підсистему та витісняє корисні дані з буферного кешу. З іншого боку, надмірна кількість індексів уповільнює операції INSERT, UPDATE та DELETE, оскільки СУБД змушена оновлювати всі пов'язані структури B-дерева. Системи моніторингу дозволяють виявляти такі аномалії через аналіз показника *Index Hit Ratio* та статистики використання індексів.

2. Проблема $N + 1$ запитів.

Виникає на рівні клієнтського додатку (найчастіше при використанні ORM-фреймворків) [30]. Замість виконання одного комплексного запиту (з використанням JOIN), додаток виконує один початковий запит для отримання списку з N елементів, а потім ще N окремих запитів для отримання пов'язаних даних. Це призводить до різкого зростання показника *Rate* (кількість запитів за секунду) та перевантаження мережевого каналу.

3. Ескалація блокувань (Lock Escalation).

Механізм, за допомогою якого СУБД (наприклад, MS SQL Server) з метою економії оперативної пам'яті перетворює множину дрібних блокувань (на рівні рядків або сторінок) на одне велике блокування всієї таблиці. Хоча це знижує використання пам'яті самою СУБД, ескалація повністю блокує доступ іншим транзакціям до цієї таблиці, що фіксується системою моніторингу як різке зростання метрики *Blocked Sessions*.

4. Витік з'єднань (Connection Leaks).

Ситуація, коли клієнтський додаток відкриває з'єднання з базою даних, але не закриває його після завершення роботи. СУБД продовжує утримувати

ресурси для цього сеансу, що зрештою призводить до вичерпання пулу підключень (max_connections) та повної відмови в обслуговуванні (Denial of Service).

1.7 Особливості моніторингу в умовах сучасної DevOps-культури та контейнеризації

Протягом останніх років парадигма розгортання інформаційних систем зазнала фундаментальних змін. Перехід до DevOps-практик та мікросервісної архітектури зумовив масову відмову від встановлення СУБД безпосередньо на фізичні сервери на користь віртуалізації та контейнеризації (зокрема, використання технологій Docker) [31].

Ця трансформація суттєво ускладнила предметну область моніторингу та висунула нові вимоги до інструментів автоматизації:

1. Ефемерність інфраструктури. У контейнеризованих середовищах інстанси баз даних (особливо на етапах розробки та тестування) можуть створюватися та знищуватися динамічно. Класичні системи моніторингу з жорстко прописаними IP-адресами та агентською архітектурою погано адаптуються до таких умов. Це робить безагентський (Agentless) підхід, який я застосовую в розроблюваному програмно-методичному комплексі, значно більш гнучким, оскільки він дозволяє підключатися до БД виключно за рядком підключення (Connection String) з підтримкою динамічних портів.

2. Ізоляція ресурсів (Cgroups та Namespaces). Коли СУБД працює в контейнері Docker, системні метрики (наприклад, загальне використання CPU або RAM) можуть не відображати реальної картини, оскільки ядро Linux обмежує ресурси контейнера через механізм Control Groups [32]. У таких умовах найдостовірнішим джерелом інформації про ефективність стають не

зовнішні утиліти операційної системи, а виключно внутрішні системні представлення самої бази даних (DMV у MS SQL або pg_stat у PostgreSQL).

3. Уніфікація управління. Впровадження конвеєрів безперервної інтеграції та доставки (CI/CD) вимагає, щоб інструменти моніторингу були легко керованими [33] і мали стандартизовані формати обміну даними (наприклад, JSON). Це дозволяє автоматично передавати дані про продуктивність до систем вищого рівня або ініціювати автоскейлінг (Auto-scaling) ресурсів бази даних у моменти пікових навантажень.

2 МОДЕЛЮВАННЯ ПРОЦЕСІВ АВТОМАТИЗАЦІЇ МОНІТОРИНГУ БАЗ ДАНИХ

2.1 Обґрунтування вибору методів теоретичних та експериментальних досліджень, програмного забезпечення

2.1.1 Обґрунтування вибору методів теоретичних досліджень

Для розв'язання науково-практичного завдання з автоматизації обробки даних про ефективність сучасних систем управління базами даних необхідно застосувати комплексний підхід, що спирається на сучасні методи системного аналізу, інформаційного та математичного моделювання.

База даних є складною динамічною системою, стан якої безперервно змінюється під впливом транзакційного навантаження. Для формалізації процесів збору, обробки та аналізу телеметричних даних доцільно використати методологію структурного аналізу та проєктування SADT (Structured Analysis and Design Technique). Використання діаграм цього стандарту дозволить декомпозувати загальний процес моніторингу на взаємопов'язані підпроцеси, чітко визначивши вхідні дані (системні метрики СУБД), керуючі впливи у вигляді алгоритмів та порогів сповіщень, механізми програмно-апаратних ресурсів та вихідні результати у вигляді алертів та аналітичних звітів.

Для проєктування архітектури самого програмно-методичного комплексу обрано об'єктно-орієнтований підхід та уніфіковану мову моделювання UML (Unified Modeling Language). Побудова Use Case Diagrams (діаграм прецедентів), Class Diagrams (діаграм класів) та Sequence Diagrams (діаграм послідовностей) забезпечить наочне представлення логічної структури системи, розподіл відповідальності між програмними модулями та формалізацію сценаріїв взаємодії адміністратора з системою.

Окремим теоретичним методом, що застосовується для опису поведінки об'єкта, є теорія масового обслуговування. Оскільки потік SQL-запитів до СУБД

має яскраво виражений ймовірнісний характер, математичне моделювання поведінки системи дозволить науково обґрунтувати критичні пороги спрацювання алертів (наприклад, гранично допустиму довжину черги транзакцій або час блокувань).

2.1.2 Обґрунтування вибору методів експериментальних досліджень

Оскільки розроблюваний програмно-методичний комплекс призначений для роботи в реальних високонавантажених середовищах, експериментальна частина дослідження є критично важливою. Для перевірки ефективності алгоритмів збору телеметрії та швидкодії самої програми обрано методи навантажувального тестування (Load Testing) та імітаційного моделювання інцидентів.

Суть навантажувального тестування полягатиме у штучній генерації транзакційного навантаження на тестові екземпляри СУБД (PostgreSQL та MS SQL Server) з паралельною фіксацією того, наскільки швидко розроблений програмно-методичний комплекс реєструє відхилення показників від норми. Метод імітаційного моделювання передбачає штучне створення аварійних ситуацій (наприклад, тривалих взаємних блокувань Deadlocks або виснаження пулу з'єднань) для перевірки коректності роботи підсистеми сповіщень.

2.1.3 Обґрунтування вибору програмного забезпечення та засобів розробки

Реалізація теоретичних моделей у вигляді працюючого програмного продукту вимагає ретельного підбору інструментальних засобів. Оскільки

цільовою аудиторією розроблюваного програмно-методичного комплексу є адміністратори баз даних, програма повинна мати зручний графічний інтерфейс, працювати без затримок та ефективно виконувати мережеві запити.

Для вибору основної мови програмування та платформи було проведено порівняльний аналіз трьох потужних сучасних інструментів: C++, Python та C#. Результати аналізу наведено у таблиці 2.1.

Таблиця 2.1 – Порівняльний аналіз мов програмування для розробки програмно-методичного комплексу

Критерій оцінки	C++	Python	C# (.NET / WPF)
Швидкодія та обчислювальна потужність	Дуже висока	Середня (обмеження GIL)	Висока (JIT-компіляція)
Швидкість розробки графічного інтерфейсу (GUI)	Середня (вимагає ручного управління пам'яттю)	Висока	Дуже висока (декларативна розмітка XAML)
Реалізація багатопотоковості та асинхронності	Складна	Обмежена (через архітектуру інтерпретатора)	Вбудована на рівні мови (async / await)
Екосистема драйверів для роботи з СУБД	Вимагає налаштування нативних бібліотек	Широка екосистема	Високорівнева екосистема ADO.NET (Npgsql, SqlClient)
Простота розгортання у Windows-середовищі	Середня	Вимагає пакування інтерпретатора	Максимальна (нативна підтримка середовища)

На основі результатів порівняння, та проведеного аналізу, для розробки програмно-методичного комплексу я обрав середовище розробки Microsoft Visual Studio та платформу .NET 8 з мовою програмування C#. Мій вибір зумовлений тим, що моніторинг баз даних передбачає виконання великої

кількості мережевих операцій вводу-виводу. Механізми асинхронного програмування у C# дозволяють опитувати віддалені сервери СУБД без блокування графічного інтерфейсу [36]. Для створення візуальної частини обрав технологію WPF (Windows Presentation Foundation) [37].

2.1.4 Обґрунтування вибору СУБД для зберігання телеметрії

Розроблюваний програмний комплекс, працюючи за безагентським принципом, повинен зберігати конфігурації підключень, налаштування порогів чутливості та історію зафіксованих інцидентів. Для цього необхідна локальна система зберігання даних. Для вибору оптимального рішення було проведено аналіз та порівняно кілька варіантів СУБД. Результати порівняння наведено в таблиця 2.2

Таблиця 2.2 – Порівняльний аналіз СУБД для локального зберігання даних програмно-методичного комплексу

Характеристика	PostgreSQL	MS SQL Server Express	SQLite
Архітектура	Клієнт-серверна	Клієнт-серверна	Вбудована (Serverless)
Складність розгортання	Потребує окремої інсталяції та конфігурації	Потребує окремої інсталяції, займає багато місця	Не потребує інсталяції, один файл бази даних
Споживання апаратних ресурсів	Середнє	Високе	Мінімальне
Підтримка транзакцій (ACID)	Повна підтримка	Повна підтримка	Повна підтримка
Доцільність для моєї задачі	Надлишкова	Надлишкова	Оптимальна

Аналіз показав, що використання повноцінних клієнт-серверних баз даних для внутрішніх потреб десктопного додатка є надлишковим і суттєво ускладнить процес розгортання програми на робочому місці адміністратора. Тому на мою думку, оптимальним вибором стане саме SQLite - компактна, автономна реляційна СУБД [38], яка забезпечить високу швидкість доступу до локальних даних та підтримує стандартні SQL-запити.

Для взаємодії програми з локальною базою SQLite планую використати технологію об'єктно-реляційного відображення Entity Framework Core, що дозволить абстрагуватися від написання сирих SQL-запитів [39]. Водночас, для прямого підключення та швидкого збору метрик із цільових систем моніторингу, таких як PostgreSQL та MS SQL Server, будуть використані високопродуктивні провайдери даних Npgsql та Microsoft.Data.SqlClient відповідно.

Комплексне застосування вищезазначених теоретичних методів, експериментальних підходів та інструментального програмного забезпечення формує надійну методологічну базу для проєктування та реалізації ефективного програмно-методичного комплексу.

2.2 Математична (логічна, функціональна) модель предметної області дослідження

Розробка моделі об’єкта дослідження є фундаментальним і найбільш критичним етапом проєктування будь-якого програмно-методичного комплексу. Практика програмної інженерії доводить, що перехід до безпосереднього написання вихідного коду без попереднього створення логічних та математичних абстракцій призводить до архітектурних помилок, які важко та дорого виправляти на пізніх етапах життєвого циклу ПЗ [34]. Моделювання забезпечує плавний перехід від словесного (абстрактного)

опису предметної області моніторингу баз даних до конкретних алгоритмічних рішень та структур даних.

Я обрав шлях комплексного підходу до моделювання, який охоплює три рівні: функціональний (опис процесів обробки інформації), об'єктно-орієнтований (опис архітектури програмного засобу) та математичний (формалізація критеріїв оцінки ефективності).

2.2.1 Функціональне моделювання за методологією SADT

Для системного представлення процесів обробки даних у програмно-методичному комплексі використав методологію структурного аналізу та проєктування SADT (Structured Analysis and Design Technique), реалізовану через нотацію стандарту IDEF0. Цей підхід дозволяє розглядати складну програмну систему як сукупність ієрархічно підпорядкованих функцій, що взаємодіють через потоки даних та керуючі впливи. Головна перевага IDEF0 полягає у використанні жорсткого правила ICOM (Input, Control, Output, Mechanism) [35], що унеможливлює втрату критично важливих елементів під час проєктування.

На першому етапі розробив контекстну діаграму рівень A0, яка визначає глобальні межі системи та її взаємодію із зовнішнім технологічним середовищем. Контекстна діаграма рівня A0 зображена на рисунку 2.1

Рисунок 2.1 – Контекстна діаграма IDEF0 Рівень A0 процесу автоматизації обробки даних про ефективність роботи сучасних систем управління базами даних

Детальний опис елементів контекстної діаграми A0:

- Вхідні дані (Input – ліва грань): основним потоком неструктурованої інформації є сирі телеметричні дані, що періодично вилучаються з системних каталогів віддалених СУБД (наприклад, динамічних адміністративних представлень DMV у MS SQL Server). Другим вхідним потоком є запити адміністратора – інтерактивні команди на побудову графіків або вивантаження звітів за певний період.

- Керування (Control – верхня грань): визначає правила виконання процесу. Сюди належать конфігураційні параметри (інтервали опитування в мілісекундах, рядки підключення Connection Strings) та пороги спрацювання алертів (математичні константи, що визначають межі штатного та аварійного режимів).
- Виходи (Output – права грань): кінцевий корисний результат роботи програмно-методичного комплексу. Система генерує візуалізовані дашборди продуктивності, оперативні сповіщення про критичні інциденти та безперервний архів телеметрії, що зберігається у локальній базі даних для ретроспективного розслідування збоїв.
- Механізми (Mechanism – нижня грань): ресурси, за допомогою яких виконується функція. Це безпосередньо розроблений програмно-методичний комплекс (виконуваний файл), апаратні ресурси робочої станції та безпосередньо сам Адміністратор БД як суб'єкт прийняття рішень.

Для деталізації внутрішньої логіки та алгоритмічної послідовності виконано декомпозицію головного процесу на діаграмі першого рівня A1. Діаграма декомпозиції рівня A1 зображена на рисунку 2.2.

Відповідно до діаграми декомпозиції, загальний процес функціонування програмно-методичного комплексу розбито на чотири послідовні технологічні блоки:

1. A1. Опитування СУБД: перший етап, на якому відбувається ініціалізація мережевих з'єднань за протоколом TCP/IP та асинхронний збір сирих метрик. Управління цим блоком здійснюється виключно таймерами, заданими в конфігурації.
2. A2. Обробка та нормалізація: критично важливий проміжний етап. Оскільки лічильники ОС та СУБД часто мають кумулятивний характер (постійно зростають з моменту запуску сервера), цей блок обчислює дельта-значення (різницю між поточним і попереднім зрізом) та нормалізує дані, приводячи їх до єдиного формату (наприклад, байтів у секунду або відсотків).

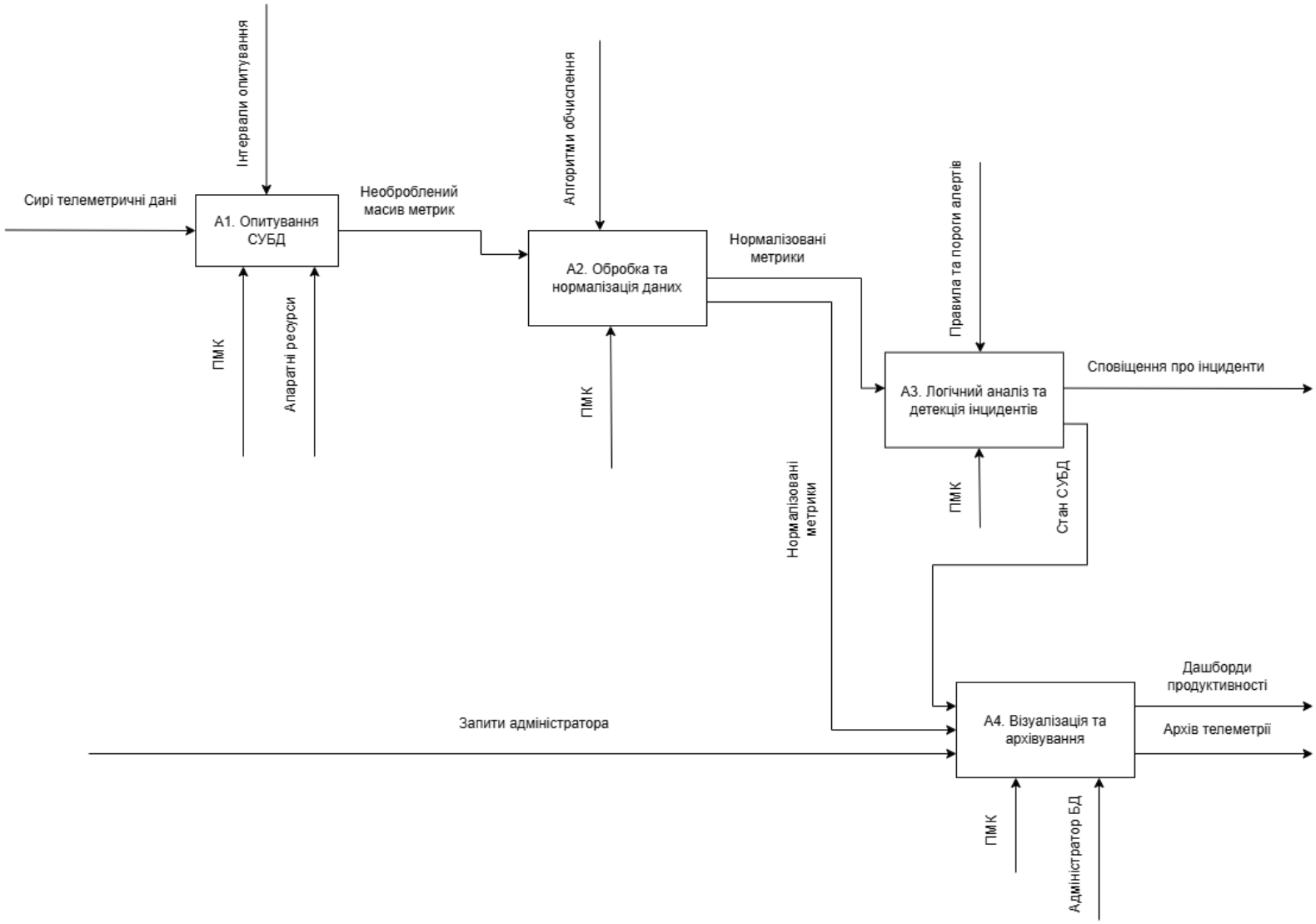


Рисунок 2.2 - Діаграма декомпозиції IDEF0 Рівень A1 процесу автоматизації обробки даних про ефективність роботи сучасних систем управління базами даних

3. A3. Логічний аналіз та детекція інцидентів: вузол прийняття рішень. Нормалізовані метрики проходять через каскад умовних операторів, де порівнюються з еталонами. Якщо виявлено відхилення, генерується подія інциденту.

4. A4. Візуалізація та архівування: фінальний етап розпаралелення даних. Один потік оновлює графічний інтерфейс користувача (WPF), а інший – виконує SQL-інструкції INSERT для збереження значень у локальну таблицю часових рядів SQLite.

2.2.2 Логічне та архітектурне моделювання засобами UML

Для проєктування об'єктно-орієнтованої структури та опису динаміки роботи програмно-методичного комплексу використав стандартизовану мову UML (Unified Modeling Language). Це дозволяє формалізувати архітектуру програмного забезпечення та підготувати чіткі специфікації для етапу кодування [25].

Ролі користувача та функціональні кордони системи представлені на діаграмі прецедентів, зображеній на рисунку 2.3.

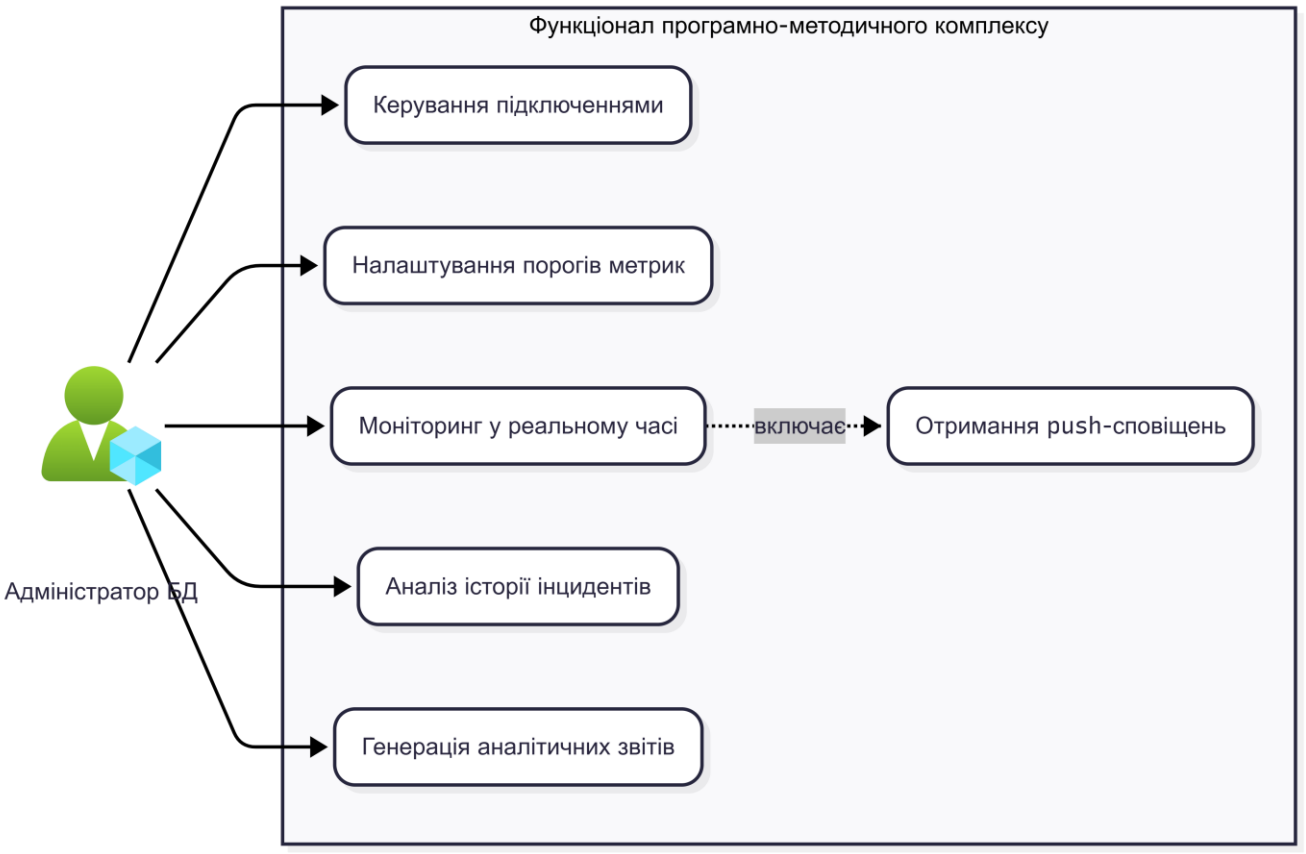


Рисунок 2.3 – Діаграма прецедентів програмно-методичного комплексу для автоматизації обробки даних про ефективність роботи сучасних систем управління базами даних

Архітектурна модель програмно-методичного комплексу, представлена на розгорнутій діаграмі класів, зображеній на рисунку 2.4, відображає ієрархію об'єктів та логіку їхньої взаємодії в межах предметної області моніторингу СУБД. На відміну від високорівневих моделей, дана діаграма деталізує структуру даних, що забезпечує пряму кореляцію між програмними сутностями та схемою локальної бази даних SQLite.

Структурні моделі даних: Логічне ядро системи складається з чотирьох взаємопов'язаних класів, які формують інформаційну модель об'єкта моніторингу:

- Клас «Сервер»: є центральним елементом, що ідентифікує цільову базу даних. Він містить назву, параметри підключення та тип архітектури PostgreSQL або MS SQL Server.
- Клас «Метрика»: агрегує кількісні показники ефективності. Назви полів класу відповідають фізичним параметрам, що зчитуються з СУБД: завантаження процесора, обсяг вільної оперативної пам'яті, інтенсивність транзакцій та ефективність роботи буферного кешу.
- Клас «Поріг_Чутливості»: формалізує правила аналізу. Кожен екземпляр класу жорстко прив'язаний до конкретного сервера, що дозволяє реалізувати індивідуальну політику моніторингу для різних за потужністю систем.
- Клас «Інцидент»: використовується для реєстрації аномальних станів. Він фіксує часові межі проблеми та її текстовий опис, що є необхідним для подальшого аудиту.

Функціональні компоненти та сервіси: Взаємодія між моделями даних та зовнішнім середовищем забезпечується шаром сервісів:

1. Збір даних: реалізований через інтерфейс «ІЗбирач_Телеметрії». Це дозволяє системі використовувати різні алгоритми опитування залежно від типу СУБД, не змінюючи при цьому логіку основного циклу програми.
2. Аналіз та детекція: клас «Аналізатор_Ефективності» відповідає за інтелектуальну обробку метрик. Він виконує порівняння поточних значень із порогами та, у разі потреби, ініціює створення об'єкта «Інцидент».
3. Управління сховищем: клас «Менеджер_БД_SQLite» ізолює логіку роботи з фізичними файлами бази даних. Він забезпечує транзакційність запису історії моніторингу та швидку вибірку даних для побудови графіків.

Центральну координацію процесів здійснює Головний_Диспетчер_Моніторингу. Він реалізує патерн управління життєвим циклом системи, пов'язуючи всі компоненти в єдиний технологічний ланцюг, від ініціалізації з'єднання до фіксації результатів у локальному архіві.

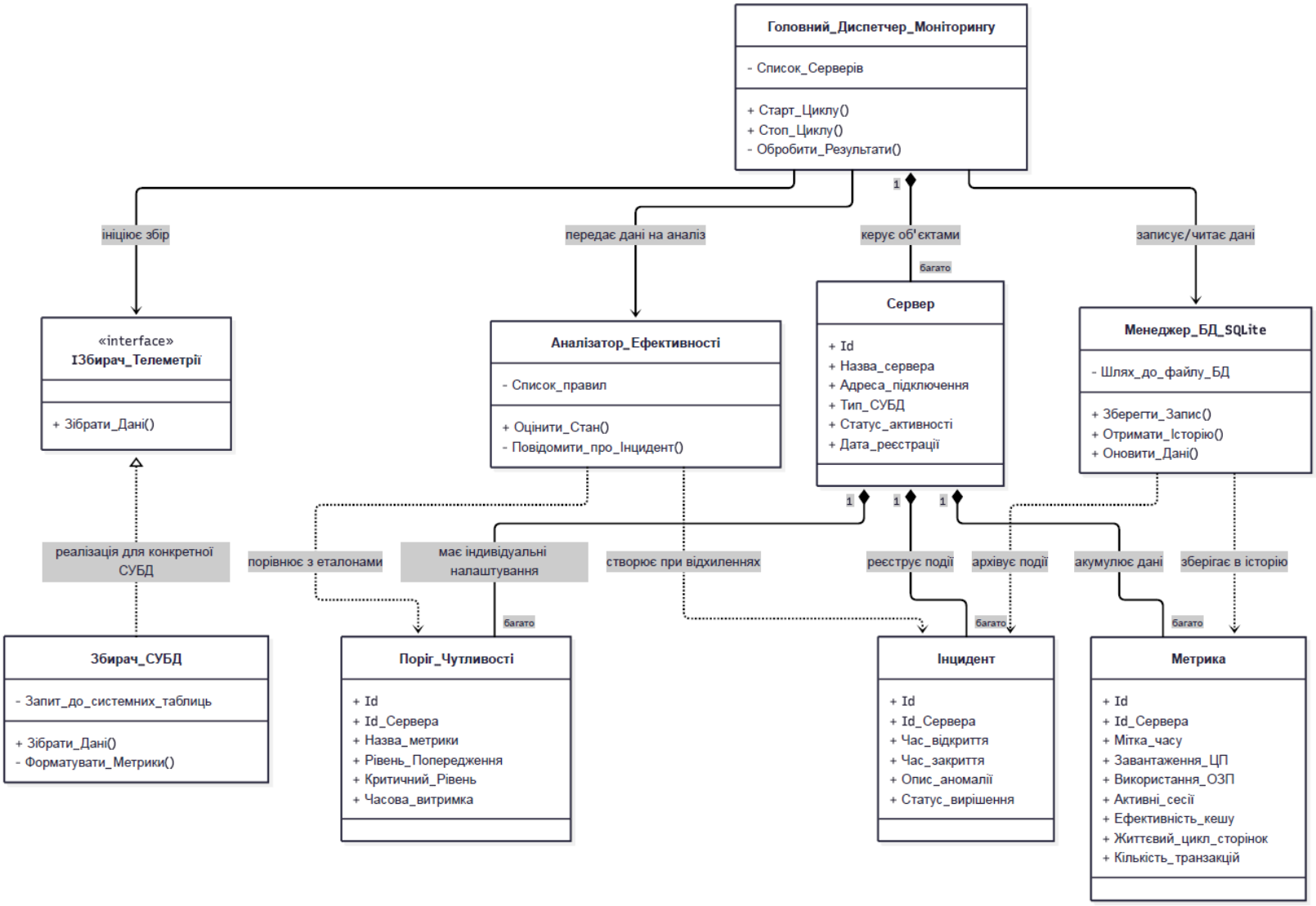


Рисунок 2.4 – Діаграма класів програмно-методичного комплексу для автоматизації обробки даних про ефективність роботи сучасних систем управління базами даних

Для моделювання алгоритму виявлення інцидентів (роботи системи алертів) розробив діаграму станів (State Machine Diagram).

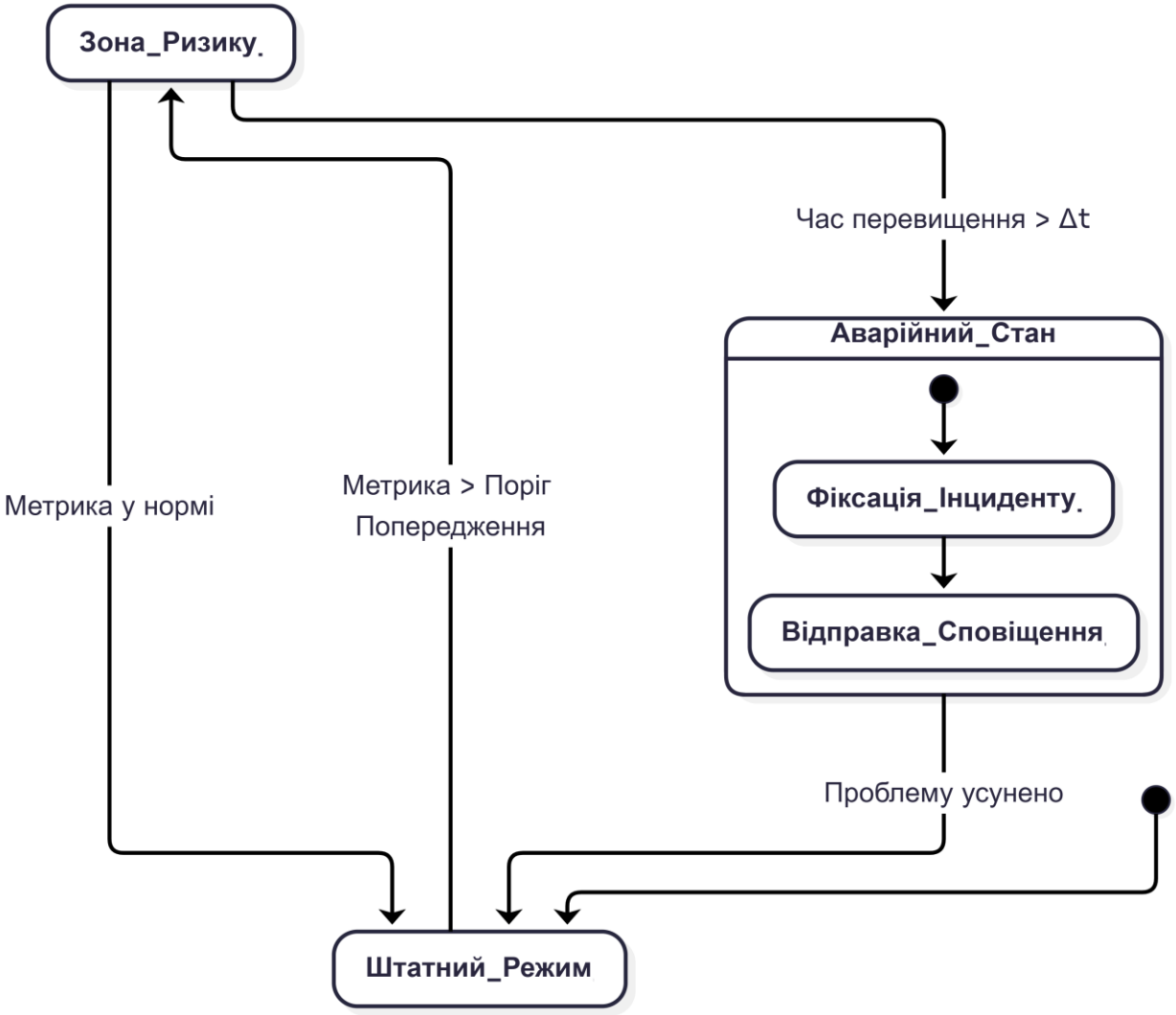


Рисунок 2.5 – Діаграма станів програмно-методичного комплексу для автоматизації обробки даних про ефективність роботи сучасних систем управління базами даних

Діаграма станів візуалізує алгоритм роботи підсистеми сповіщень. Система перебуває у Штатному режимі, доки метрики не перевищують ліміти. При фіксації відхилення відбувається перехід у стан Зони ризику. Якщо аномалія триває довше визначеного вікна толерантності Δt, система

переходить у Аварійний стан, що супроводжується фіксацією інциденту та відправкою сповіщення. Повернення до норми відбувається автоматично при стабілізації показників.

Циклічність обробки даних у часі та взаємодія між підсистемами деталізовані на діаграмі послідовності (Sequence Diagram).

Процес циклічного опитування та життєвий цикл обробки метрик деталізовано на діаграмі послідовності рисунок 2.6.

Модель демонструє розшаровану взаємодію компонентів системи під час однієї ітерації моніторингу. Ініціатором процесу виступає Головний Диспетчер, який за таймером викликає специфічний Збирач Телеметрії. Після отримання сирих даних від Цільової СУБД, Збирач виконує їх нормалізацію та повертає готовий об'єкт-сутність «Метрика». Далі Диспетчер передає цей об'єкт до Аналізатора Ефективності, який співставляє отримані значення з індивідуальними порогами чутливості сервера. Після аналізу запускаються паралельні процеси: запис значень у Менеджер БД SQLite та оновлення графічного Інтерфейсу. Якщо Аналізатор фіксує критичне відхилення, він генерує сутність «Інцидент», яка окремо архівується у локальну базу даних, а на інтерфейс користувача виводиться оперативне сповіщення (Алерт).

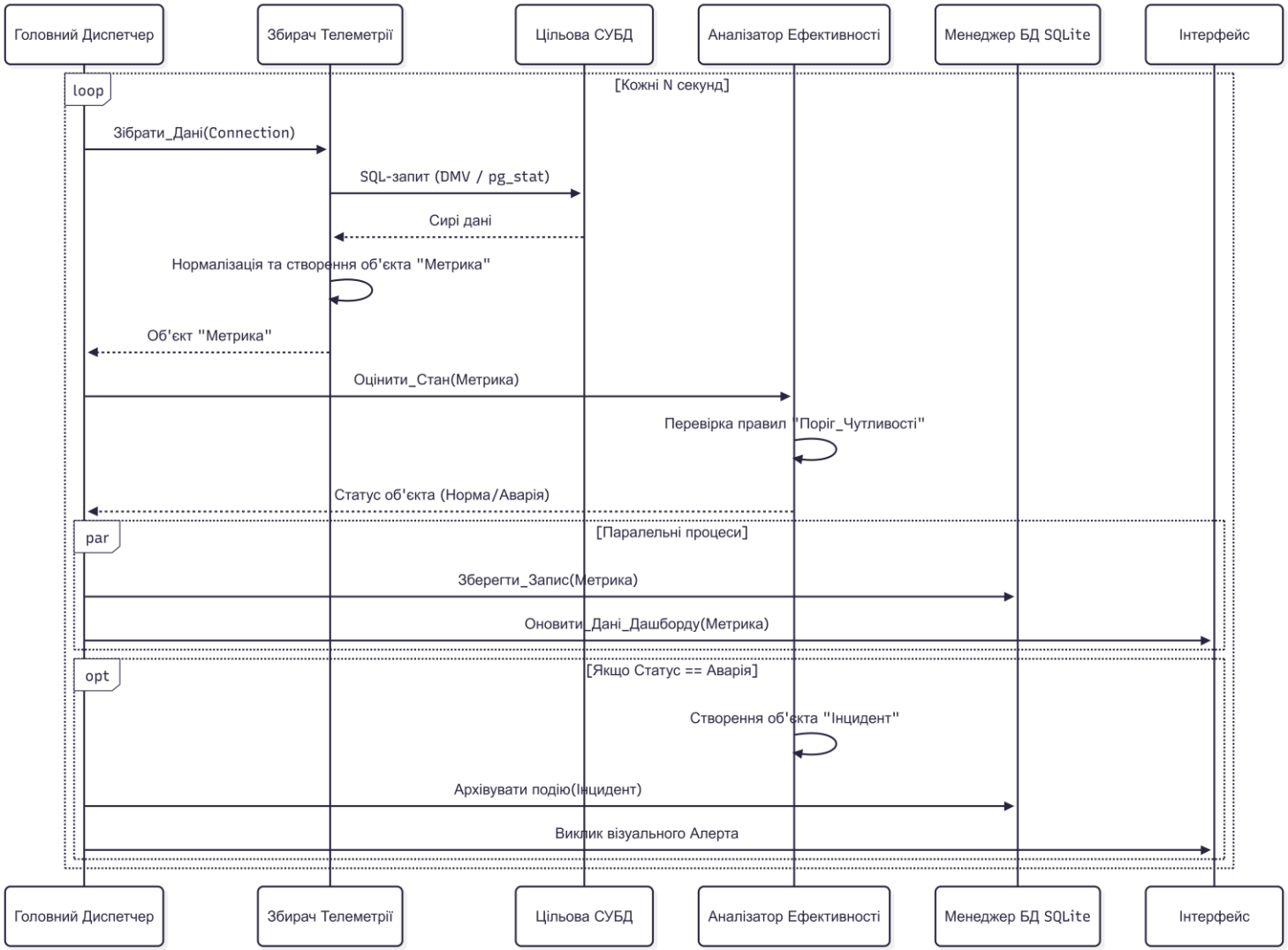


Рисунок 2.6 – Діаграма послідовності програмно-методичного комплексу для автоматизації обробки даних про ефективність роботи сучасних систем управління базами даних

2.2.3 Математична модель обробки даних та аналізу продуктивності

Моніторинг продуктивності не повинен зводитися до простої констатації поточних значень. Математичне моделювання дозволяє перевести якісні характеристики поведінки СУБД у суворі кількісні показники та алгоритми [23].

2.2.4 СУБД як система масового обслуговування (Модель Ерланга та Закон Літтла)

Відповідно до класичної теорії масового обслуговування (ТМО), сервер реляційної бази даних розглядається як багатоканальна система з чергою. Потік вхідних SQL-запитів моделюється пуассонівським розподілом з інтенсивністю λ (запитів за секунду), а обчислювальні ресурси (пул робочих потоків СУБД) виступають каналами обслуговування з середньою і інтенсивністю μ

Фундаментальним показником ефективності є коефіцієнт завантаження системи ρ (формула 2.1).

$$\rho = \frac{\lambda}{n \cdot \mu} \tag{2.1}$$

У розроблюваному програмно-методичному комплексі цей коефіцієнт корелює з метрикою використання процесора (CPU Utilization) та активністю потоків. Для оцінки загального стану черг у СУБД застосовується Закон Літтла [40] (формула 2.2).

$$L = \lambda \cdot W \quad (2.2)$$

де L – середня кількість запитів, що перебувають у системі (метрика Active Sessions), а W – середній час відповіді системи (метрика Average Query Duration).

Математична модель демонструє, що при наближенні коефіцієнта завантаження $\rho \rightarrow 1$ значення W зростає за експоненційним законом. Завдяки цій моделі програмно-методичний комплекс може предиктивно попереджати адміністратора про наближення деградації (коли процесор завантажений на 85-90%), ще до того, як система повністю відмовить.

2.2.5 Математична модель згладжування телеметрії (EWMA)

Однією з головних проблем моніторингу високонавантажених систем є висока волатильність («шумність») метрик, таких як кількість зчитувань з диска (IOPS) або завантаження CPU. Пряме порівняння таких метрик з жорсткими порогами призводить до постійних помилкових алертів.

Для розв'язання цієї проблеми в математичну модель програмно-методичного комплексу впроваджено алгоритм експоненційного згладжування (Exponentially Weighted Moving Average, EWMA) [41]. Згладжене значення метрики S_t у поточний момент часу t обчислюється за рекурентною формулою 2.3.

$$S_t = \alpha \cdot Y_t + (1 - \alpha) \cdot S_{t-1} \quad (2.3)$$

де Y_t – поточне «сире» значення метрики, отримане з СУБД;

$S_{\{t-1\}}$ – згладжене значення на попередньому кроці;

α – коефіцієнт згладжування (0 < α < 1), який визначає чутливість моделі до різких стрибків (у програмно-методичному комплексі рекомендоване значення $\alpha \approx 0.3$).

Саме значення S_t (а не Y_t) передається у логічний блок порівняння з порогами $T_{\{warn\}}$ та $T_{\{crit\}}$, що забезпечує високу стабільність підсистеми сповіщень.

2.2.6 Моделювання ефективності підсистеми пам'яті (CHR та PLE)

Найбільш вразливим вузлом будь-якої реляційної бази даних є швидкість зчитування даних. Оскільки дисковий ввід-вивід на порядки повільніший за оперативну пам'ять, оцінка ефективності кешування є критичною.

Для PostgreSQL базовою математичною моделлю ефективності пам'яті є коефіцієнт попадання в буферний кеш (Cache Hit Ratio, CHR) [6] (формула 2.4).

$$CHR = \left(1 - \frac{Read_{physical}}{Read_{logical}}\right) * 100\% \quad (2.4)$$

де $Read_{logical}$ – кількість запитів сторінок даних, які СУБД знайшла в оперативній пам'яті (Shared Buffers), а $Read_{physical}$ – кількість вимушених фізичних зчитувань з диска.

Для архітектури MS SQL Server, на додаток до CHR, критично важливою є метрика очікуваної тривалості життя сторінки (Page Life Expectancy, PLE). Вона моделює середній час (у секундах), який сторінка даних здатна протриматися в оперативній пам'яті до її витіснення новими даними. Математично PLE (формула 2.5) залежить від обсягу виділеної пам'яті ($M_{\{pool\}}$) та інтенсивності оновлення сторінок (Churn Rate, $V_{\{churn\}}$).

$$PLE \approx \frac{\{M_{\{pool\}}\}}{\{V_{\{churn\}}\}} \quad (2.5)$$

Експериментально доведено, що для сучасних систем падіння PLE нижче 300 секунд на кожні 4 ГБ пам'яті NUMA-вузла є чітким математичним індикатором гострого дефіциту оперативної пам'яті (Memory Pressure).

Розроблений комплекс моделей органічно поєднує функціональну декомпозицію процесів (IDEF0), сучасне об'єктно-орієнтоване проєктування (UML) та класичний математичний апарат (ТМО і статистичне згладжування). Це створює надійний, науково обґрунтований фундамент для переходу до етапу програмної реалізації комплексу.

2.3 Розробка методики дослідження

Метою проведення експериментальних досліджень є практична перевірка адекватності розроблених математичних моделей, підтвердження працездатності алгоритмів програмно-методичного комплексу та оцінка його ефективності в умовах, максимально наближених до реального промислового середовища. Для досягнення цієї мети розроблено комплексну методику, яка регламентує вибір факторів впливу, організацію тестового стенда, сценарії навантаження та способи обробки отриманих результатів, спираючись на загальноприйняті стандарти системного аналізу продуктивності [42].

2.3.1 Визначення та обґрунтування факторів дослідження

У контексті моніторингу систем управління базами даних експеримент розглядається як процес спостереження за поведінкою об'єкта (СУБД) під впливом керованих збурень. Для побудови коректного плану експерименту всі фактори розділено на дві групи: незалежні (керовані) та залежні (спостережувані) [43].

Незалежні фактори (вхідні змінні):

Це параметри, які будуть штучно змінюватися під час тестування для симуляції різних станів системи.

1. Інтенсивність транзакційного навантаження (λ): кількість SQL-запитів, що генеруються та надсилаються до бази даних за одну секунду (Transactions Per Second, TPS).

2. Кількість одночасних підключень (Concurrent Connections): імітує кількість активних мікросервісів, що одночасно звертаються до СУБД, створюючи конкуренцію за пул потоків (Thread Pool).

3. Апаратні обмеження (Resource Throttling): штучне зменшення доступної оперативної пам'яті (RAM) або обмеження пропускної здатності дискової підсистеми для імітації «вузьких місць» інфраструктури.

Залежні фактори (вихідні змінні):

Це показники, які фіксуються розробленим програмно-методичним комплексом у процесі експерименту.

1. Системні метрики: коефіцієнт завантаження процесора (CPU Utilization), відсоток використання оперативної пам'яті.

2. Специфічні метрики СУБД: коефіцієнт попадання в кеш (Cache Hit Ratio), кількість активних та заблокованих сесій (Active/Blocked Sessions).

3. Метрики реактивності програмно-методичного комплексу: час затримки між фактичним виникненням аномалії на сервері та моментом генерації сповіщення (Алерта).

2.3.2 Обладнання та програмне забезпечення тестового стенда

Для забезпечення чистоти експерименту тестування повинно проводитись в ізольованому середовищі. Використання робочої станції розробника як одночасно генератора навантаження і сервера баз даних є недопустимим. З огляду на це, розроблено архітектуру тестового стенда на базі технологій віртуалізації. Стенд складається з трьох логічних вузлів:

1. Вузол «Сервер БД PostgreSQL» (Об'єкт спостереження №1):

- ОС: Ubuntu Server 22.04 LTS.
- СУБД: PostgreSQL 16.
- Ресурси: 2 vCPU, 4 GB RAM.

2. Вузол «Сервер БД MS SQL» (Об'єкт спостереження №2):

- ОС: Windows Server 2022.
- СУБД: Microsoft SQL Server 2022 Developer Edition.
- Ресурси: 4 vCPU, 8 GB RAM (обмеження пам'яті налаштовано в SQLOS).

3. Вузол «Робоча станція Адміністратора» (Керуючий вузол):

- ОС: Windows 10/11.
- Досліджуване ПЗ: Розроблений програмно-методичний комплекс на базі .NET 8 / WPF.
- Допоміжне ПЗ для генерації навантаження:

a) pgbench: стандартна утиліта PostgreSQL, що дозволяє запускати синтетичні транзакційні тести (на основі стандарту TPC-B) для генерації стабільного потоку запитів [44].

b) SQLQueryStress: спеціалізована утиліта для MS SQL Server, яка дозволяє виконувати задані T-SQL скрипти у багатопотоковому режимі з визначенням кількості ітерацій [45].

2.3.3 План проведення експериментальних досліджень

Методика передбачає послідовне виконання трьох тестових сценаріїв згідно зі стандартами навантажувального тестування програмного забезпечення.

Етап 1. Сценарій А: Вимірювання «ефекту спостерігача» (Baseline Testing). Оскільки програмно-методичний комплекс працює за безагентською архітектурою, необхідно довести, що його вплив на СУБД є мінімальним.

- План дій: Підключити програмно-методичний комплекс до СУБД у стані спокою та встановити максимально агресивний інтервал опитування (наприклад, кожні 5 секунд).
- Очікуваний результат: Навантаження на CPU тестових серверів від запитів програмно-методичного комсплексу не повинно перевищувати 1-2%.

Етап 2. Сценарій Б: Стрес-тестування (Stress Testing). Метою цього етапу є перевірка здатності програмно-методичного комплексу фіксувати деградацію системи при наближенні до меж її пропускної здатності.

- План дій: За допомогою утиліт pgbench та SQLQueryStress генерувати навантаження, ступінчасто збільшуючи кількість одночасних віртуальних клієнтів (від 10 до 500).
- Очікуваний результат: програмно-методичного комплекс повинен зафіксувати експоненційне зростання метрик Active Sessions та Wait Time, послідовно переводячи об'єкти зі «Штатного режиму» в «Аварійний стан».

Етап 3. Сценарій В: Імітація інцидентів та перевірка алгоритмів згладжування. Перевірка алгоритму толерантності (Δt) та ефективності математичного згладжування EWMA.

- Тест 1 (Мікро-сплески): Запустити дуже короткочасний запит, який завантажить CPU на 100% протягом 3 секунд. Програмно-методичний комплекс не повинен генерувати алерт, відфільтрувавши подію як шум.

- Тест 2 (Ескалація блокувань - Deadlocks): Відкрити дві транзакції в MS SQL Server, які перехресно заблокують спільні таблиці. Програмно-методичний комплекс повинен зафіксувати зростання метрики Blocked Sessions та згенерувати подію «Інцидент» після витримки заданого часу.

2.3.4 Методика обробки та аналізу результатів

Дані, отримані в ході експериментів, підлягають кількісному та якісному аналізу.

1. Крос-валідація метрик: Графіки продуктивності програмно-методичного комплексу будуть накладені на еталонні графіки, зняті нативними засобами моніторингу операційних систем (Windows Performance Monitor для MS SQL та утиліта top/htop для Linux/PostgreSQL) [44]. Відхилення значень не повинно перевищувати статистичної похибки.
2. Аналіз затримок: Час виникнення реальної аномалії (за логами СУБД) порівнюється з часовою міткою генерації об'єкта «Інцидент» у програмно-методичному комплексі.
3. Оцінка інформативності: Перевірка того, наскільки зібрана інформація є достатньою для визначення першопричини (Root Cause Analysis) змодельованого збою адміністратором.

2.4 Розробка технічного завдання на створення засобів моделювання предметної області

Кінцевим етапом теоретичного та логіко-функціонального проектування системи є формалізація всіх отриманих вимог у вигляді єдиного нормативно-

технічного документа – Технічного завдання (ТЗ). У сучасній практиці програмної інженерії ТЗ виступає базовим інструментом, що регламентує взаємодію між етапами аналізу, розробки та тестування, мінімізуючи ризики розбіжностей між очікуваннями користувача та реальною функціональністю системи [46].

Розробка ТЗ для програмно-методичного комплексу автоматизованої обробки даних про ефективність СУБД базується на положеннях міжнародного стандарту ISO/IEC/IEEE 29148:2018 (Systems and software engineering – Life cycle processes – Requirements engineering) та вітчизняних стандартів серії ДСТУ, що регулюють процеси створення автоматизованих систем [47]. Відповідно до цих стандартів, повний і деталізований текст Технічного завдання винесено у Додаток Б.

3 РОЗРОБКА ПРОЄКТУ ПРОГРАМНО-МЕТОДИЧНОГО КОМПЛЕКСУ ТА ЙОГО РЕАЛІЗАЦІЯ

3.1 Розробка логічної моделі програмно-методичного комплексу для моделювання

Створення надійного, масштабованого та стійкого до відмов програмного забезпечення вимагає чіткого проєктування логічної моделі на ранніх етапах життєвого циклу розробки. Логічна модель абстрагується від фізичного розміщення виконуваних файлів, специфіки апаратного забезпечення чи конкретного синтаксису мови програмування. Її головна мета – концептуально визначити, як система обробляє дані, які сутності існують у предметній області та за якими правилами (бізнес-логікою) вони взаємодіють.

У розробленому програмно-методичному комплексі логічну архітектуру розділено на декілька самостійних векторів: логіку взаємодії з користувачем, логіку зберігання даних, абстрактну архітектуру збору телеметрії та логіко-математичний апарат підсистеми сповіщень.

3.1.1 Логічна модель взаємодії з користувачем (Use Case)

Проєктування поведінки системи починається з визначення меж її відповідальності та способів взаємодії з акторами (користувачами або зовнішніми процесами). Для формалізації цих процесів розроблено логічну діаграму прецедентів (Use Case Diagram) за стандартом UML, зображену на рисунку 3.1.

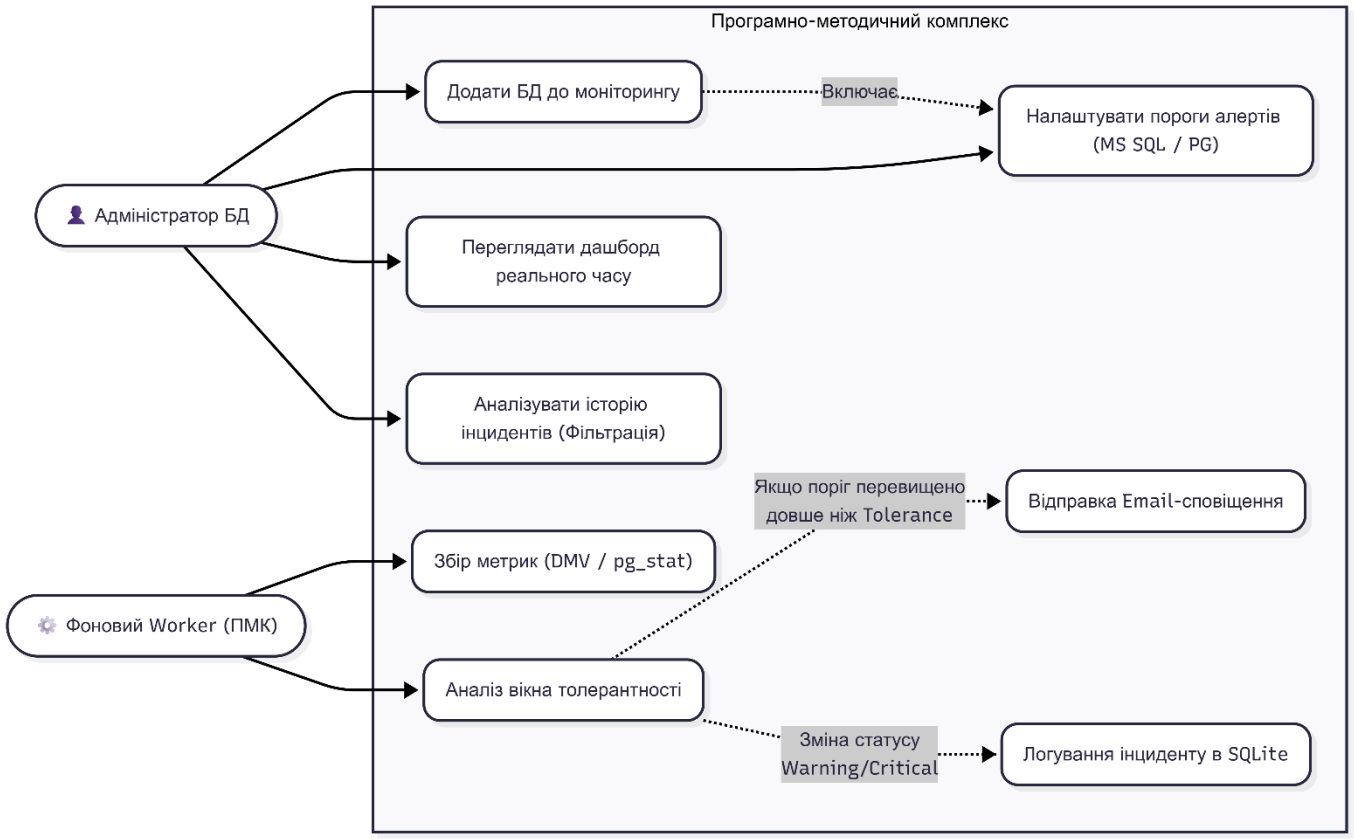


Рисунок 3.1 – Діаграма прецедентів (Use Case) взаємодії з програмно-методичним комплексом

На діаграмі виділено двох головних акторів: «Адміністратор БД» (людина, що ініціює налаштування) та «Фоновий Worker» (автономний системний процес ПМК). Детальний опис логічних прецедентів наведено у таблиці 3.1.

Таблиця 3.1 – Опис прецедентів логічної моделі

Назва прецеденту	Актор	Логічний опис процесу
Додати БД до моніторингу	Адміністратор	Ініціалізація нового об'єкта моніторингу. Система перевіряє доступність сервера, валідує облікові дані та шифрує пароль перед збереженням у репозиторій.
Налаштувати пороги алертів	Адміністратор	Встановлення індивідуальних лімітів (Warning/Critical) та вікна толерантності для вибраної СУБД залежно від її типу (MS SQL або PostgreSQL).

Продовження таблиці 3.1

Назва прецеденту	Актор	Логічний опис процесу
Переглядати дашборд	Адміністратор	Запит на візуалізацію телеметрії. Система генерує аналітичні часові ряди на основі даних з локального архіву.
Аналізувати інциденти	Адміністратор	Формування запиту до підсистеми журналювання з використанням фільтрів (за датою, рівнем критичності, назвою БД).
Збір метрик	Фоновий Worker	Циклічний, асинхронний безагентський збір даних із системних каталогів цільової СУБД (DMV або pg_stat).
Аналіз вікна толерантності	Фоновий Worker	Логіко-математичне порівняння зібраних поточних метрик із заданими порогами. Ведення лічильників відхилень.
Відправка Email-сповіщення	Фоновий Worker	Асинхронне формування та відправка електронного листа через протокол SMTP у разі переходу системи в стан Critical або Warning.
Логування інциденту	Фоновий Worker	Транзакційний запис факту виникнення або вирішення аномалії до локальної бази даних SQLite.

3.1.2 Логічна модель локального репозиторію даних (ER-модель)

Для забезпечення повної автономності додатка як рушія репозиторію обрано вбудовану реляційну СУБД SQLite. Логічна доменна модель предметної області розроблена з дотриманням правил третьої нормальної форми (3NF), що гарантує цілісність даних та відсутність аномалій оновлення.

У системі свідомо застосовано підхід використання сурогатних ключів. У кожній таблиці первинним ключем (Primary Key) виступає поле Id (ціле число з автоінкрементом). Це забезпечує максимальну швидкість індексування та виконання JOIN-запитів. Крім того, це захищає базу: якщо адміністратор змінить назву сервера (*ServerName*), цілісність історичних метрик не порушиться. На рисунку 3.2 зображено ER-діаграму моєї БД, яка використовується в розробленому програмно-методичному комплексі.

Рисунок 3.2 – Логічна ER-діаграма бази даних програмно-методичного комплексу для автоматизації обробки даних про ефективність роботи сучасних систем управління базами даних

Для глибшого розуміння структурних зв'язків архітектури, у таблиці 3.2 наведено розширений логічний опис кожної сутності розробленої бази даних.

Таблиця 3.2 – Логічна структура сутностей бази даних програмно-методичного комплексу

Сутність (Таблиця)	Логічне призначення	Ключові атрибути та типи даних
MonitoredInstance	Зберігає конфігурації підключень до керованих екземплярів СУБД. Є батьківською сутністю для всіх інших таблиць.	DatabaseType (String) – стратегія збору.EncryptedPassword (String) – криптовектор. IsActive (Boolean) – статус.
DatabaseMetric	Акумулює показники продуктивності специфічні для об'єктів Microsoft SQL Server.	ActiveSessions (Integer) – навантаження. MaxQueryDurationSeconds (Double) BufferCacheHitRatio (Double) – ефективність RAM.
PgMetric	Акумулює унікальні системні лічильники продуктивності для PostgreSQL.	DeadTuplesCount (Long) – якість Autovacuum.ConnectionUsagePercent (Double).IndexHitRatio (Double) – якість індексів.
ServerMetric	Зберігає загальносистемні показники операційної системи Windows Server (для MS SQL).	CpuUtilizationPercent (Double). DiskCFreeSpaceGb (Double) – контроль пам'яті.

Продовження таблиці 3.2

Сутність (Таблиця)	Логічне призначення	Ключові атрибути та типи даних
AlertThreshold	Зберігає індивідуальні ліміти та пороги чутливості для баз даних MS SQL.	...Warning та ...Critical (Double/Integer) – пороги. ToleranceMinutes (Integer) – затримка спрацювання.
PgAlertThreshold	Зберігає специфічні пороги чутливості для баз даних PostgreSQL.	Відрізняється наявністю *DeadTuplesCritical* (Long) та *MinIndexHitWarning* (Double).
AlertLog	Журнал інцидентів. Зберігає ретроспективну історію зміни станів (аномалій).	Timestamp (DateTime) AlertLevel (String) – рівень загрози. CurrentValue (Double) – зафіксоване відхилення.

3.1.3 Логічна архітектура абстрагування збору даних (Патерн «Стратегія»)

З огляду на фундаментальні архітектурні відмінності цільових СУБД (MS SQL Server керується власною ОС – SQLOS, тоді як PostgreSQL є мультипроцесорною системою середовища Linux), логіка отримання телеметрії вимагає суворого абстрагування. У програмно-методичному комплексі цю задачу вирішено за допомогою поведінкового патерну об'єктно-орієнтованого проєктування – «Стратегія» (Strategy Pattern).

Логічний контракт взаємодії системи збору даних закріплений на рівні абстрактного інтерфейсу (назвемо його *IMetricCollector*). Головний процес моніторингу не має жодного уявлення про те, які саме SQL-запити виконуються. Він лише передає об'єкт конфігурації підключення і очікує повернення нормалізованого об'єкта метрик (наприклад, *DatabaseMetric* або *PgMetric*).

Логіка інстанціювання стратегій працює наступним чином:

Диспетчер читає поле DatabaseType з конфігурації. Якщо тип «Microsoft SQL Server», логіка передає управління стратегії, що використовує драйвер абстракції на базі протоколу TDS (Tabular Data Stream). Ця стратегія реалізує читання з системних динамічних представлень (DMV), таких як *sys.dm_exec_requests*.

Якщо тип «PostgreSQL», управління передається стратегії, яка взаємодіє з системним каталогом pg_stat (наприклад, *pg_stat_activity* та *pg_stat_user_tables*).

Такий підхід забезпечує абсолютну відповідність принципу відкритості/закритості (Open/Closed Principle з SOLID). Для впровадження підтримки нових СУБД (наприклад, MySQL або Oracle) у майбутньому не доведеться змінювати логіку ядра системи – достатньо буде лише додати новий клас, який реалізує інтерфейс стратегії.

3.1.4 Логіко-математичний апарат кінцевого автомата підсистеми сповіщень

Найбільш науково містким логічним компонентом розробленого програмно-методичного комплексу є підсистема детекції аномалій. Практичні дослідження показали, що жорстке математичне порівняння метрики з пороговим значенням $(\mathrm{Metric} > \mathrm{Threshold})$ призводить до значної кількості хибних тривог (False Positives). У високонавантажених транзакційних системах мікросекундні сплески завантаження CPU або короткочасні блокування є нормою і не потребують втручання адміністратора.

Для вирішення цієї проблеми у логіку програми закладено математичну модель кінцевого автомата (State Machine) з імплементацією «вікна толерантності» ($\Delta\, t$).

Логічний стан будь-якої метрики можна описати множиною станів:

$$S \in \{\text{Normal}, \text{Warning}, \text{Critical}\}$$

Для кожної метрики кожної бази даних у оперативній пам'яті програмно-методичного комплексу створюється ізольований логічний лічильник відхилень C.

Алгоритм прийняття рішень логічно описується наступною послідовністю:

При кожній ітерації опитування (Tick) отримане значення $V_{\{current\}}$ порівнюється з лімітами попередження $L_{\{warn\}}$ та критичної помилки $L_{\{crit\}}$.

Якщо $V_{\{current\}} \geq L_{\{crit\}}$, лічильник відхилень інкрементується: $C = C + 1$.

Система порівнює значення лічильника з константою вікна толерантності $T_{\{tol\}}$, яка задана адміністратором (наприклад, 3 хвилини).

Якщо $C > T_{\{tol\}}$, система ініціює зміну стану на $S = \text{Critical}$. Лише в цей момент спрацьовує тригер: генерується подія відправки Email-сповіщення та створюється транзакційний запис інциденту в таблиці AlertLog.

Якщо під час наступної ітерації навантаження падає $V_{\{current\}} < L_{\{warn\}}$, лічильник миттєво обнуляється $C = 0$, стан змінюється на $S = Normal$, і генерується сповіщення про вирішення інциденту (Resolved).

Завдяки такій логіці система повністю ігнорує безпечні «мікро-сплески» навантаження і сигналізує лише про стійкі тренди деградації продуктивності баз даних.

3.1.5 Логічна модель криптографічної безпеки

Оскільки для проведення безагентського збору телеметрії програмно-методичного комплексу змушений зберігати логіни та паролі від баз даних з

адміністративними правами, логічна архітектура категорично виключає зберігання цих даних у відкритому тексті (Plain Text) або з використанням статичних симетричних ключів.

В основу логічної моделі безпеки закладено використання системного інтерфейсу криптографії ОС Windows – Data Protection API (DPAPI).

Логіка шифрування побудована таким чином, що криптографічний ключ генерується ядром операційної системи динамічно на основі облікових даних (SID) поточного авторизованого користувача Windows.

Математично це означає, що зашифрований вектор (Encrypted Password), який зберігається у полі бази даних SQLite, жорстко прив'язаний до профілю користувача та конкретного комп'ютера. У випадку, якщо зловмисник отримає фізичний доступ до файлу БД *monitor_data.db* та скопіює його на іншу робочу станцію (або спробує відкрити під іншим обліковим записом), процес розшифрування логічно завершиться помилкою *CryptographicException*. Це гарантує корпоративний рівень безпеки аутентифікаційних даних цільових серверів.

3.2 Розробка фізичної моделі проєкту програмно-методичного комплексу

Якщо логічна модель описує абстрактну взаємодію сутностей, то фізична модель проєкту деталізує способи реалізації цих абстракцій у конкретному апаратно-програмному середовищі. Розробка фізичної моделі програмно-методичного комплексу включає проєктування мережевої взаємодії (топології), фізичної організації локального сховища даних, визначення апаратних вимог та структуризацію файлового дерева вихідного коду проєкту.

3.2.1 Архітектура розгортання та мережева топологія

Програмно-методичний комплекс спроєктовано як автономний настільний додаток (Standalone Desktop Application) типу «товстий клієнт». Фізично додаток розгортається на робочій станції адміністратора баз даних або на виділеному моніторинговому сервері (Jump Server) в межах корпоративного контуру.

Для формалізації мережевої взаємодії програмно-методичного комплексу із зовнішніми вузлами (цільовими СУБД та поштовими сервісами) використовується діаграма розгортання (Deployment Diagram) стандарту UML зображена на рисунку 3.10.

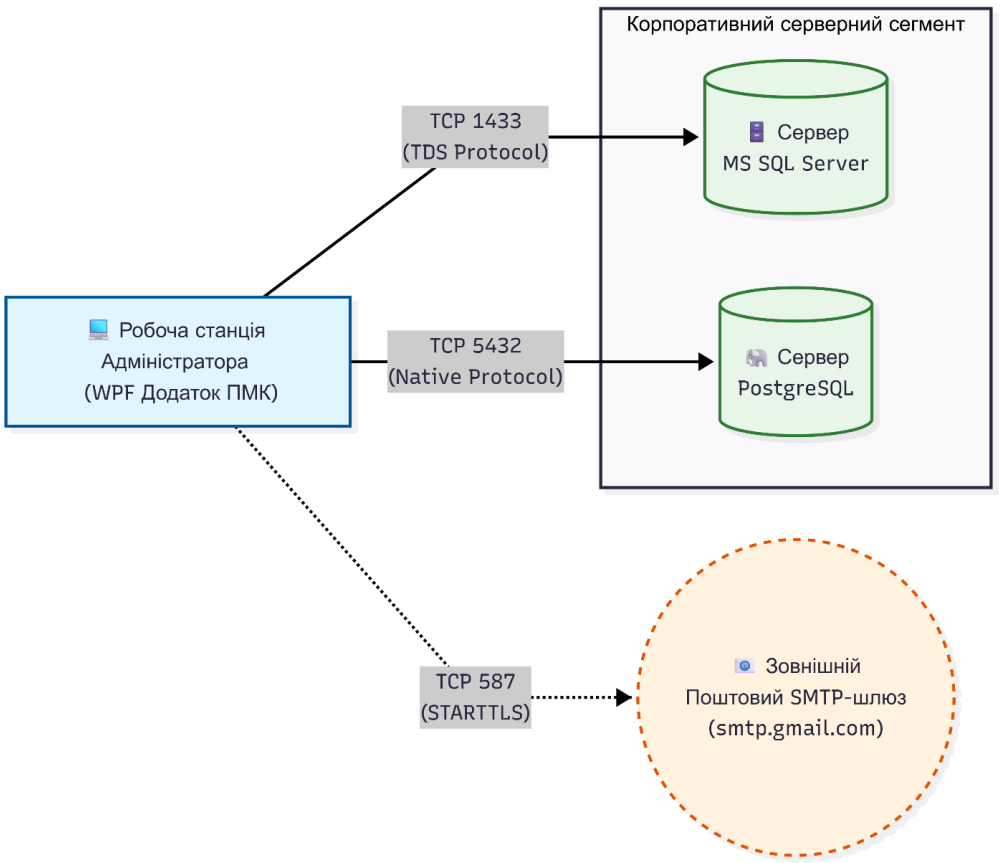


Рисунок 3.10 – Діаграма розгортання фізичної архітектури програмно-методичного комплексу для автоматизації обробки даних про ефективність роботи сучасних систем управління базами даних

Безагентський принцип збору телеметрії вимагає наявності прямих мережевих маршрутів від вузла програмно-методичного комплексу до цільових серверів. Фізична взаємодія здійснюється за допомогою наступних протоколів та портів (які повинні бути відкриті на міжмережевих екранах):

TCP 1433 (Default): Стандартний порт для підключення до екземплярів Microsoft SQL Server за протоколом Tabular Data Stream (TDS).

TCP 5432 (Default): Порт для встановлення нативних підключень до серверів PostgreSQL.

TCP 587 (SMTP з STARTTLS): Порт для криптографічно захищеної передачі Email-сповіщень через зовнішній поштовий шлюз (наприклад, smtp.gmail.com), який викликається модулем *EmailAlertService*.

3.2.2 Фізична організація репозиторію даних

На фізичному рівні, на відміну від повноцінних СУБД, локальна база даних SQLite не вимагає встановлення окремих фонових служб. Увесь масив конфігурацій, метрик та журналів інцидентів фізично зберігається у єдиному бінарному файлі з назвою monitor_data.db.

Відповідно до конфігурації у класі *MonitorDbContext*, цей файл автоматично створюється у кореневій директорії виконання програмно-методичного комплексу поруч із скомпільованим файлом *.exe.* Фізична безпека цього файлу гарантується двома факторами:

Управлінням правами доступу на рівні файлової системи NTFS (лише поточний адміністратор має права на читання/запис каталогу програми).

Фізичною неможливістю розшифрувати паролі, збережені у файлі *monitor_data.db*, при його перенесенні на інший комп'ютер, завдяки алгоритмам Data Protection API (DPAPI).

3.2.3 Фізична структура файлів вихідного коду (Solution Structure)

Для забезпечення легкості супроводу, масштабування та модифікації вихідного коду, фізична структура проєкту у середовищі розробки Microsoft Visual Studio суворо розмежована за принципами патерну MVC/MVVM, що зображено на рисунку 3.11. Дерево проєкту (Solution) розділене на три головні фізичні каталоги: *Models, Services ma Views*.

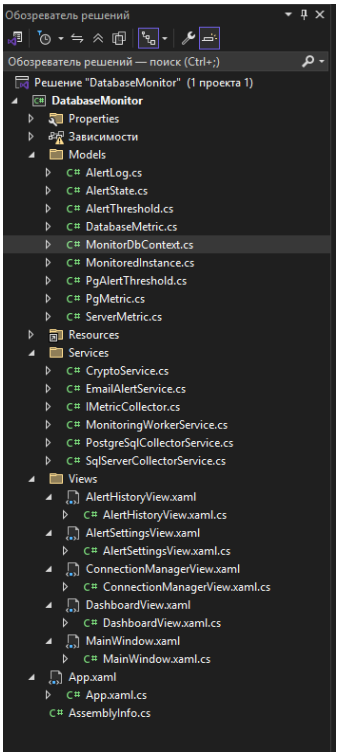


Рисунок 3.11 – Фізична структура каталогів проєкту у середовищі розробки для програмно-методичного комплексу для автоматизації обробки даних про ефективність роботи сучасних систем управління базами даних

Опис фізичних файлів та їхнього архітектурного призначення наведено у таблиці 3.3.

Таблиця 3.3 – Призначення ключових файлів вихідного коду програмно-методичного комплексу

Каталог / Файл	Фізичне та архітектурне призначення
\Models	Каталог класів даних (сутностей) та ORM.
MonitorDbContext.cs	Файл конфігурації провайдера SQLite та опису колекцій DbSet.
MonitoredInstance.cs	Клас конфігурації цільового сервера (IP, логін, шифротекст пароля).
DatabaseMetric.cs, PgMetric.cs	Файли, що описують структури телеметрії для різних типів СУБД.
AlertThreshold.cs, PgAlertThreshold.cs	Класи з описом порогів Warning/Critical для кожної СУБД.
AlertLog.cs, AlertState.cs	Класи для ведення історії інцидентів та перелічення статусів.
\Services	Каталог бізнес-логіки та фонових служб.
MonitoringWorkerService.cs	Головний фізичний потік збору даних та аналізу вікна толерантності.
IMetricCollector.cs	Інтерфейс стратегії збору даних.
PostgreSqlCollectorService.cs	Файл з реалізацією безагентських SQL-запитів до каталогів PostgreSQL.
SqlServerCollectorService.cs	Файл з реалізацією безагентських T-SQL запитів до системних DMV.
CryptoService.cs	Реалізація алгоритмів шифрування DPAPI.
EmailAlertService.cs	Модуль інтеграції з протоколом SMTP.
\Views	Каталог розмітки графічного інтерфейсу (XAML).
MainWindow.xaml / .cs	Точка входу UI, контейнер-маршрутизатор та бокове меню.
DashboardView.xaml / .cs	Файли дашборду з реалізацією відмальовування графіків LiveCharts.
ConnectionManagerView.xaml / .cs	Вікно управління серверами (додавання, тестування з'єднання, видалення).
AlertSettingsView.xaml / .cs	Візуальні форми налаштування лімітів чутливості алертів.
AlertHistoryView.xaml / .cs	Інтерфейс перегляду історії інцидентів із застосуванням фільтрів.
Корінь проєкту	Файли конфігурації та запуску.
App.xaml / .cs	Фізична точка входу в програму (Startup), управління життєвим циклом фонових задач (Task.Run) та токенами скасування (CancellationToken).

Дана структура гарантує високу зв'язність всередині модулів та низьке зачеплення між ними (High Cohesion / Low Coupling). Наприклад, зміна фізичного дизайну кнопок у DashboardView.xaml не вплине на алгоритм збору метрик у *SqlServerCollectorService.cs*.

3.2.4 Програмно-апаратні вимоги середовища функціонування

Оскільки програмно-методичний комплекс розроблено на базі сучасної кросплатформної технології .NET 8, але з використанням графічної підсистеми Windows Presentation Foundation (WPF) та системного API шифрування (DPAPI), фізична експлуатація програми жорстко прив'язана до екосистеми операційних систем Microsoft.

Мінімальні системні вимоги до апаратного забезпечення робочої станції адміністратора, необхідні для плавного відмальовування графіків та безперебійної роботи фонового процесу:

- Процесор (CPU): 2 ядра (або потоки) з частотою від 2.0 ГГц (архітектура x64).
- Оперативна пам'ять (RAM): 2 ГБ для штатної роботи операційної системи та 200 МБ вільної пам'яті виділеної для процесу ПМК (через використання буферизації Entity Framework).
- Дисковий простір: 150 МБ для розміщення виконуваних файлів та динамічного зростання файлу *monitor_data.db*.
- Операційна система: Windows 10/11 (x64) або Windows Server 2016 і новіше.
- Програмні залежності: Встановлений пакет .NET 8.0 Desktop Runtime.

Таким чином, розроблена фізична модель гарантує максимальну компактність системи, відсутність необхідності в складному процесі розгортання «Установка в один клік» або Portable-формат та повну відповідність вимогам інформаційної безпеки корпоративних мереж.

3.3 Особливості реалізації програмних компонентів проєкту

Етап безпересередньої програмної реалізації (кодування) є ключовою стадією життєвого циклу розробки програмно-методичного комплексу. Відповідно до обраного технологічного стеку, ядро системи реалізовано об'єктно-орієнтованою мовою програмування C# на платформі .NET 8. В розділі 2.1 я вже проводив порівняльний аналіз, та аргоментував свій вибір мови програмування. Для забезпечення максимальної продуктивності та уникнення блокувань потоків вводу-виводу (I/O operations), архітектура коду масово використовує парадигму асинхронного програмування за допомогою операторів async та await. Структурно код розділено на сервіси збору даних, фоновий диспетчер, модулі безпеки та сповіщень.

3.3.1 Реалізація підсистеми безагентського збору телеметрії

Для збору метрик реалізовано два незалежні класи-провайдери, що імплементують інтерфейс *IMetricCollector*. Кожен із них адаптований під специфіку конкретної СУБД.

Клас *SqlServerCollectorService* відповідає за моніторинг інстансів Microsoft SQL Server. В якості драйвера доступу до даних (Data Provider) використано офіційну бібліотеку *Microsoft.Data.SqlClient*. Особливістю

реалізації цього компонента є формування комплексних транзакційних T-SQL запитів до системних динамічних адміністративних представлень (Dynamic Management Views – DMV).

```
//Сесії та тривалість
try
{
    string requestsQuery = @"
        SELECT
            COUNT(session_id),
            SUM(CASE WHEN blocking_session_id <> 0 THEN 1 ELSE 0 END),
            ISNULL(MAX(total_elapsed_time) / 1000.0, 0)
        FROM sys.dm_exec_requests
        WHERE session_id > 50 AND database_id = DB_ID(@dbName)";
    using (var cmd = new SqlCommand(requestsQuery, conn))
    {
        cmd.Parameters.AddWithValue("@dbName", _databaseName);
        using (var reader = cmd.ExecuteReader())
        {
            if (reader.Read())
            {
                metric.ActiveSessions = reader.IsDBNull(0) ? 0 : Convert.ToInt32(reader[0]);
                metric.BlockingCount = reader.IsDBNull(1) ? 0 : Convert.ToInt32(reader[1]);
                metric.MaxQueryDurationSeconds = reader.IsDBNull(2) ? 0 : Convert.ToDouble(reader[2]);
            }
        }
    }
}
catch { }

//Пос
```

Рисунок 3.12 – Фрагмент реалізації збору метрик блокувань та тривалості запитів (MS SQL) в програмно-методичному комплексі для автоматизації обробки даних про ефективність роботи сучасних систем управління базами даних

Як видно з лістингу (Рисунок 3.12), для оптимізації мережевого трафіку замість виконання декількох окремих запитів, метрики активних сесій, блокувань (де *blocking_session_id <> 0*) та максимальної тривалості транзакції обчислюються на боці сервера баз даних за допомогою агрегатних функцій COUNT, SUM та MAX. Окрема увага приділена збору показників дискової підсистеми. У коді реалізовано звернення до функції *sys.dm_os_volume_stats*, результати якої парсяться і перевіряються на наявність логічних дисків C: та D:. Для забезпечення стійкості роботи програмно-методичного комплексу,

кожен етап збору метрик загорнутий у блоки try...catch, що запобігає аварійному завершенню програми у випадку тимчасової недоступності DMV або втрати мережевого з'єднання.

Клас *PostgreSqlCollectorService* реалізує стратегію моніторингу баз даних PostgreSQL. Для встановлення з'єднання використовується оптимізований .NET-провайдер *Npgsql*. На відміну від MS SQL, архітектура PostgreSQL вимагає аналізу підсистеми системних каталогів. Приклад використання в коді *PostgreSqlCollectorService* зображено на рисунку 3.13.

```
//Блокування
using (var cmd = new NpgsqlCommand("SELECT count(distinct pid) FROM pg_locks WHERE granted = false;", conn))
{
    var lockRes = await cmd.ExecuteScalarAsync();
    metric.BlockingCount = lockRes != null && lockRes != DBNull.Value ? Convert.ToInt32(lockRes) : 0;
}

//Мертві рядки та Індекси
string tableStatsQuery = @"
    SELECT
        coalesce(sum(n_dead_tup), 0)::bigint as dead_tups,
        (coalesce(round(sum(idx_blks_hit) * 100.0 / nullif(sum(idx_blks_hit + idx_blks_read), 0), 2), 100))::float8 as idx_hit
    FROM pg_stat_user_tables t
    JOIN pg_statio_user_tables s ON t.relid = s.relid;";

using (var cmd = new NpgsqlCommand(tableStatsQuery, conn))
using (var reader = await cmd.ExecuteReaderAsync())
{
    if (await reader.ReadAsync())
    {
        metric.DeadTuplesCount = reader.IsDBNull(0) ? 0 : Convert.ToInt64(reader[0]);
        metric.IndexHitRatio = reader.IsDBNull(1) ? 100 : Convert.ToDouble(reader[1]);
    }
}
```

Рисунок 3.13 – Фрагмент коду аналізу мертвих рядків та ефективності індексів (PostgreSQL) в програмно-методичному комплексі для автоматизації обробки даних про ефективність роботи сучасних систем управління базами даних

Математичний розрахунок відсотка потрапляння в кеш індексів (Index Hit Ratio) виконується безпосередньо в SQL-запиті. Формула враховує суму логічних читань з індексу *(idx_blks_hit)* відносно загальної кількості фізичних та логічних читань. Використання функції *coalesce* та *nullif* у коді запиту захищає програмно-методичний комплекс від помилок ділення на нуль (Divide by Zero Exception), які можуть виникнути на новостворених або порожніх базах даних.

3.3.2 Реалізація фонового диспетчера та логіки «вікна толерантності»

Основним керуючим компонентом (ядром) програмно-методичного комплексу є клас *MonitoringWorkerService*, який успадковує абстрактний клас *BackgroundService*. Фрагмент коду з використанням класу *MonitoringWorkerService* зображено на рисунку 3.14

```
Ссылок: 2
public class MonitoringWorkerService : BackgroundService
{
    private readonly CryptoService _crypto = new CryptoService();
    private readonly EmailAlertService _email = new EmailAlertService();

    private readonly Dictionary<string, int> _toleranceCounters = new Dictionary<string, int>();
    private readonly Dictionary<string, AlertState> _lastStates = new Dictionary<string, AlertState>();

    Ссылок: 0
    protected override async Task ExecuteAsync(CancellationToken stoppingToken)
    {
        while (!stoppingToken.IsCancellationRequested)
        {
            using (var context = new MonitorDbContext())
            {
                var instances = await context.MonitoredInstances.Where(i => i.IsActive).ToListAsync();

                foreach (var instance in instances)
                {
                    try
                    {
                        string password = _crypto.Decrypt(instance.EncryptedPassword);

                        if (instance.DatabaseType == "PostgreSQL")
                        {
                            await ProcessPostgreSqlInstance(context, instance, password);
                        }
                        else
                        {
                            await ProcessSqlServerInstance(context, instance, password);
                        }
                    }
                    catch (Exception ex)
                    {
                        System.Diagnostics.Debug.WriteLine($"[Worker Error] {instance.DatabaseName}: {ex.Message}");
                    }
                }

                await context.SaveChangesAsync();
            }

            await Task.Delay(TimeSpan.FromMinutes(1), stoppingToken);
        }
    }
```

Рисунок 3.14 – Реалізація безперервного циклу моніторингу у фоновому потоці в програмно-методичному комплексі для автоматизації обробки даних про ефективність роботи сучасних систем управління базами даних

Життєвий цикл сервісу керується методом *ExecuteAsync*. Використання циклу *while* з перевіркою *IsCancellationRequested* гарантує коректне завершення фонової задачі при закритті користувачем графічного інтерфейсу. Всередині циклу реалізовано шаблон створення області видимості (Scope) для роботи з Entity Framework: об'єкт *MonitorDbContext* створюється через конструкцію *using*, що забезпечує автоматичне звільнення некерованих ресурсів БД та закриття пулу підключень після завершення кожної ітерації опитування (запобігання витокам пам'яті – Memory Leaks).

Найскладнішою частиною реалізації є алгоритм обчислення «вікна толерантності». Приклад реалізації вікна толерантності в коді зображено на рисунку 3.15

```
//Додав параметр lowerIsWorse для реверсивної логіки
Ссылок: 14
private void ProcessMetricAlert(string dbName, string metricName, double currentValue, double warningLimit, double criticalLimit, int toleranceMins, bool lowerIsWorse = false)
{
    string key = $"{dbName}_{metricName}";
    AlertState currentState = AlertState.Normal;

    if (lowerIsWorse)
    {
        if (currentValue <= criticalLimit) currentState = AlertState.Critical;
        else if (currentValue <= warningLimit) currentState = AlertState.Warning;
    }
    else
    {
        if (currentValue >= criticalLimit) currentState = AlertState.Critical;
        else if (currentValue >= warningLimit) currentState = AlertState.Warning;
    }

    if (currentState != AlertState.Normal)
    {
        if (!_toleranceCounters.ContainsKey(key)) _toleranceCounters[key] = 0;
        _toleranceCounters[key]++;

        if (_toleranceCounters[key] >= toleranceMins)
        {
            HandleStateChange(dbName, metricName, currentState, currentValue);
        }
    }
    else
    {
        _toleranceCounters[key] = 0;
        HandleStateChange(dbName, metricName, AlertState.Normal, currentValue);
    }
}
```

Рисунок 3.15 – Програмна реалізація кінцевого автомата генерації алертів в програмно-методичному комплексі для автоматизації обробки даних про ефективність роботи сучасних систем управління базами даних

Метод *ProcessMetricAlert* отримує поточне значення метрики, порівнює його з лімітами *(warningLimit, criticalLimit)* і оперує глобальним словником *_toleranceCounters*. Особливістю цього методу є підтримка реверсивної логіки

через параметр *lowerIsWorse* (наприклад, для метрики «Вільне місце на диску», де падіння значення нижче порогу є критичним). Особливістю реалізації методу *HandleStateChange* є те, що зміна стану логується в SQLite *(context.AlertLogs.Add(logEntry)),* а відправка електронного листа виноситься у паралельний потік за допомогою конструкції *Task.Run(() => _email.SendAlert(message))*. Таке архітектурне рішення є критично важливим: процес підключення до SMTP-сервера може займати декілька секунд, і якби він виконувався синхронно, це призвело б до затримки збору метрик з інших серверів.

3.3.3 Програмна реалізація криптографічного захисту (DPAPI)

Відповідальність за безпеку збережених автентифікаційних даних лежить на класі *CryptoService*. Фрагмент коду з викроистанням цього класу, зображено на рисунку 3.16. Реалізація базується на використанні простору імен *System.Security.Cryptography.*

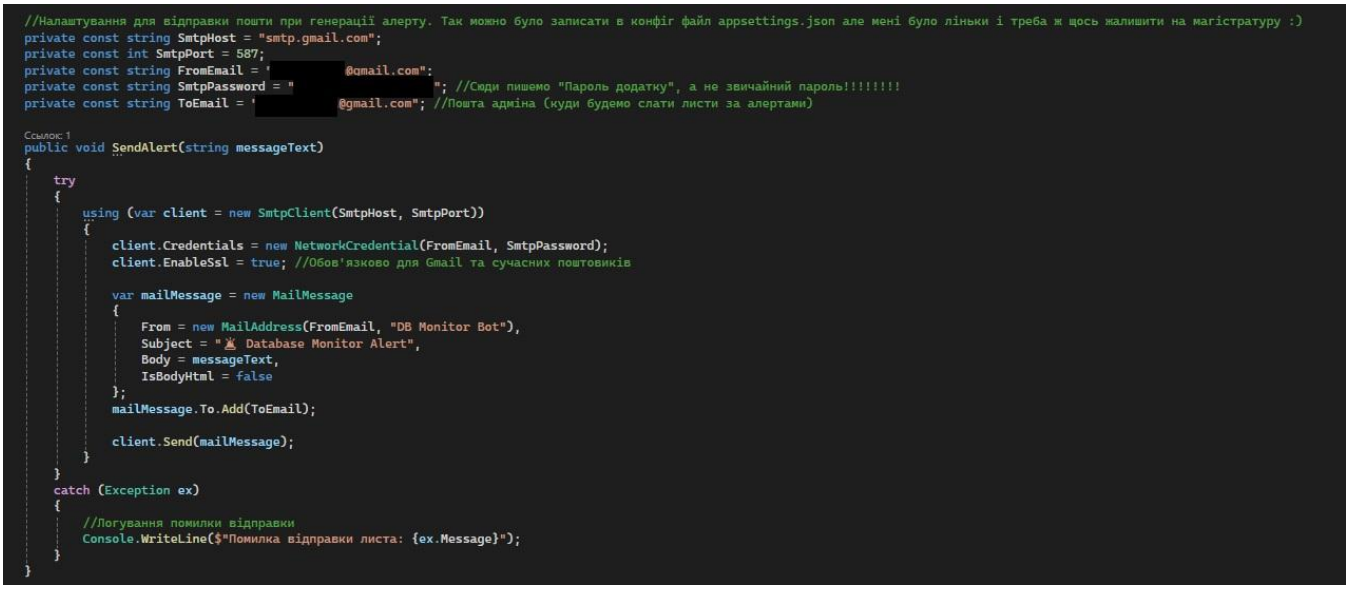
Для перетворення звичайного тексту у масив байтів (і навпаки) використовується кодування Encoding.UTF8. Основна робота виконується статичним методом *ProtectedData.Protect*. Згідно з документацією платформи .NET, цей метод звертається безпосередньо до низькорівневих функцій операційної системи Windows (*CryptProtectData*). Другий параметр (*optional entropy*) передається як *null*, оскільки безпека гарантується третім параметром – переліченням *DataProtectionScope.CurrentUser*. Згенерований зашифрований масив байтів конвертується у строковий формат Base64 за допомогою Convert.ToBase64String для зручного збереження в текстовому полі бази даних SQLite.

```
public class CryptoService
{
    //Шифруємо текст (пароля) з прив'язкою до юзера Windows
    Ссылок: 1
    public string Encrypt(string plainText)
    {
        if (string.IsNullOrEmpty(plainText))
            return plainText;

        try
        {
            byte[] plainBytes = Encoding.UTF8.GetBytes(plainText);
            //DataProtectionScope.CurrentUser гарантує, що розшифрувати зможе лише цей самий юзер Windows
            byte[] encryptedBytes = ProtectedData.Protect(plainBytes, null, DataProtectionScope.CurrentUser);

            return Convert.ToBase64String(encryptedBytes);
        }
        catch (CryptographicException)
        {
            return string.Empty;
        }
    }

    Ссылок: 1
    public string Decrypt(string encryptedText)
    {
        if (string.IsNullOrEmpty(encryptedText))
            return encryptedText;

        try
        {
            byte[] encryptedBytes = Convert.FromBase64String(encryptedText);
            byte[] plainBytes = ProtectedData.Unprotect(encryptedBytes, null, DataProtectionScope.CurrentUser);

            return Encoding.UTF8.GetString(plainBytes);
        }
        catch (CryptographicException)
        {
            return string.Empty;
        }
    }
}
```

Рисунок 3.16 – Реалізація методів шифрування та дешифрування на базі ProtectedData в програмно-методичному комплексі для автоматизації обробки даних про ефективність роботи сучасних систем управління базами даних

3.3.4 Реалізація підсистеми асинхронного сповіщення (Email Alert Service)

Клас *EmailAlertService* забезпечує інтеграцію програмно-методичного комплексу із зовнішніми поштовими шлюзами. Фрагмент коду з використанням класу *EmailAlertService*, зображено на Рисунку 3.17.



```
//Налаштування для відправки пошти при генерації алерту. Так можно було записати в конфіг файл appsettings.json але мені було ліньки і треба ж щось залишити на магістратуру :)
private const string SmtpHost = "smtp.gmail.com";
private const int SmtpPort = 587;
private const string FromEmail = "          @gmail.com";
private const string SmtpPassword = "                    "; //Сюди пишемо "Пароль додатку", а не звичайний пароль!!!!!!!
private const string ToEmail = "          @gmail.com"; //Пошта адміна (куди будемо слати листи за алертами)

Ссылок: 1
public void SendAlert(string messageText)
{
    try
    {
        using (var client = new SmtpClient(SmtpHost, SmtpPort))
        {
            client.Credentials = new NetworkCredential(FromEmail, SmtpPassword);
            client.EnableSsl = true; //Обов'язково для Gmail та сучасних поштовиків

            var mailMessage = new MailMessage
            {
                From = new MailAddress(FromEmail, "DB Monitor Bot"),
                Subject = "🚨 Database Monitor Alert",
                Body = messageText,
                IsBodyHtml = false
            };
            mailMessage.To.Add(ToEmail);

            client.Send(mailMessage);
        }
    }
    catch (Exception ex)
    {
        //Логування помилки відправки
        Console.WriteLine($"Помилка відправки листа: {ex.Message}");
    }
}
```

Рисунок 3.17 – Реалізація відправки повідомлень через протокол SMTP в програмно-методичному комплексі для автоматизації обробки даних про ефективність роботи сучасних систем управління базами даних

Для формування та відправки листів застосовано класи *SmtpClient* та *MailMessage* з простору імен *System.Net.Mail*. Важливою особливістю коду є використання властивості *EnableSsl = true*. Сучасні поштові сервери (зокрема Gmail) категорично відхиляють підключення без використання криптографічних протоколів TLS/SSL. Крім того, аутентифікація реалізована за допомогою об'єкта *NetworkCredentia*l, який передає логін та спеціально згенерований «пароль додатку» (App Password) для обходу обмежень двофакторної аутентифікації. Блок *catch* перехоплює будь-які мережеві винятки (наприклад, відсутність доступу до інтернету), гарантуючи, що збій відправки листа не призведе до зупинки основного циклу моніторингу програмно-методичного комплексу.

3.4 Елементи інтерфейсу програмно-методичного комплексу

Презентаційний рівень програмно-методичного комплексу розроблено на основі технології Windows Presentation Foundation (WPF), яка базується на використанні декларативної мови розмітки XAML (eXtensible Application Markup Language) та парадигмі відділення візуального дизайну від бізнес-логіки (*Code-Behind*). Для забезпечення сучасного, інтуїтивно зрозумілого та ергономічного користувацького досвіду (UX), до проєкту інтегровано відкриту бібліотеку *MaterialDesignThemes*, яка реалізує стандарти графічного дизайну *Material Design*.

3.4.1 Головне вікно та маршрутизація

Вікно *MainWindow.xaml* (рис 3.18) є головним контейнером програми. Воно містить стаціонарне бокове меню навігації та динамічну область контенту, яка реалізована через елемент *ContentControl x:Name=«MainContentArea».*

Логіка маршрутизації написана у *Code-Behind* файлі *MainWindow.xaml.cs*. При натисканні на кнопки бокового меню генеруються події Click, які ініціюють створення нових екземплярів *UserControl* (наприклад, *new ConnectionManagerView()*) та їх призначення у властивість Content головного контейнера. Для покращення взаємодії реалізовано метод *SetActiveButton*, який динамічно змінює колір фону та тексту обраної кнопки (*Brushes.White, Transparent*), сигналізуючи користувачеві про поточний активний розділ.

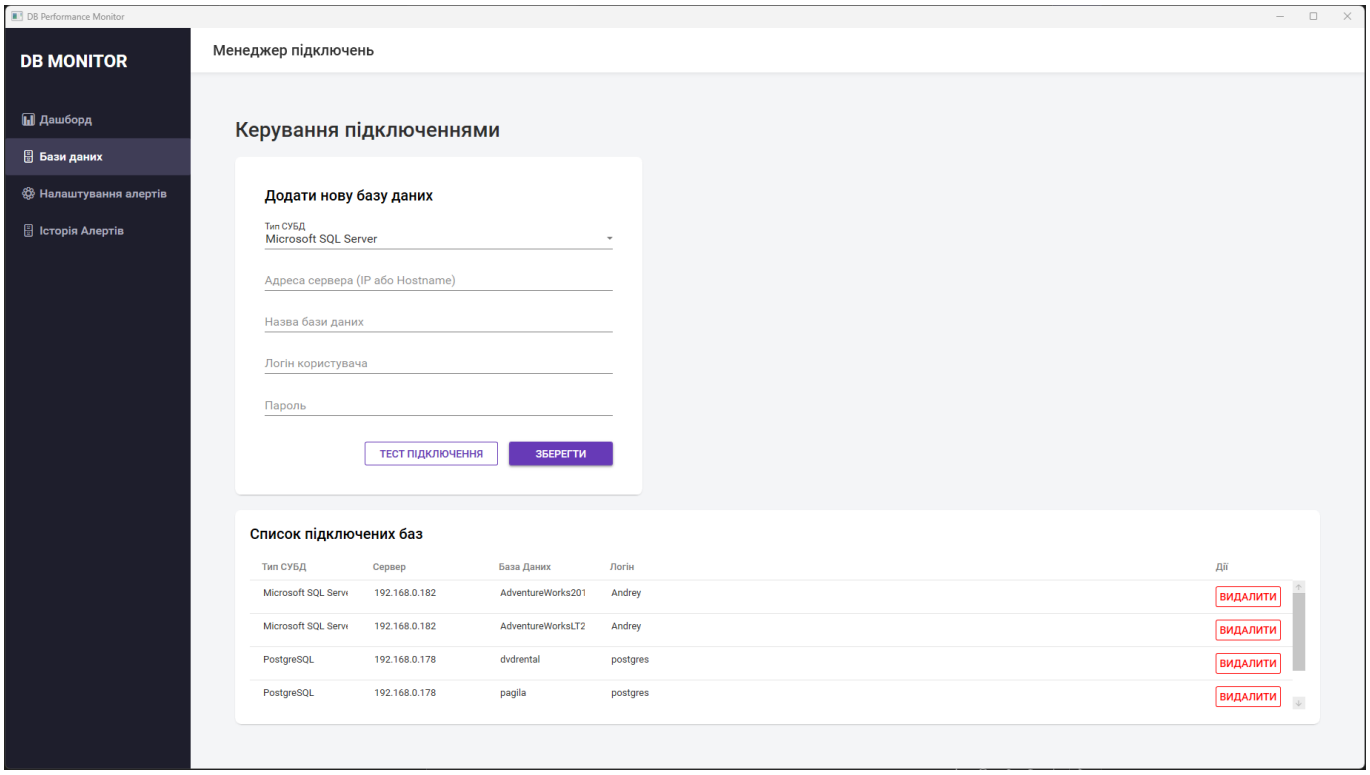


Рисунок 3.18 – Головне вікно програми програмно-методичного комплексу для автоматизації обробки даних про ефективність роботи сучасних систем управління базами даних (Менеджер підключень)

3.4.2 Інтерфейс дашборду та аналітики в реальному часі

Найбільш ресурсомістким та інформативним елементом інтерфейсу є розділ *DashboardView* (Рис 3.19). Його завдання – візуалізація масивів телеметрії у вигляді інтерактивних часових рядів.

Розмітка дашборду побудована з використанням елементів *ScrollViewer* для забезпечення вертикальної прокрутки та *UniformGrid* для автоматичного рівномірного розташування графіків у вигляді сітки з двома колонками.

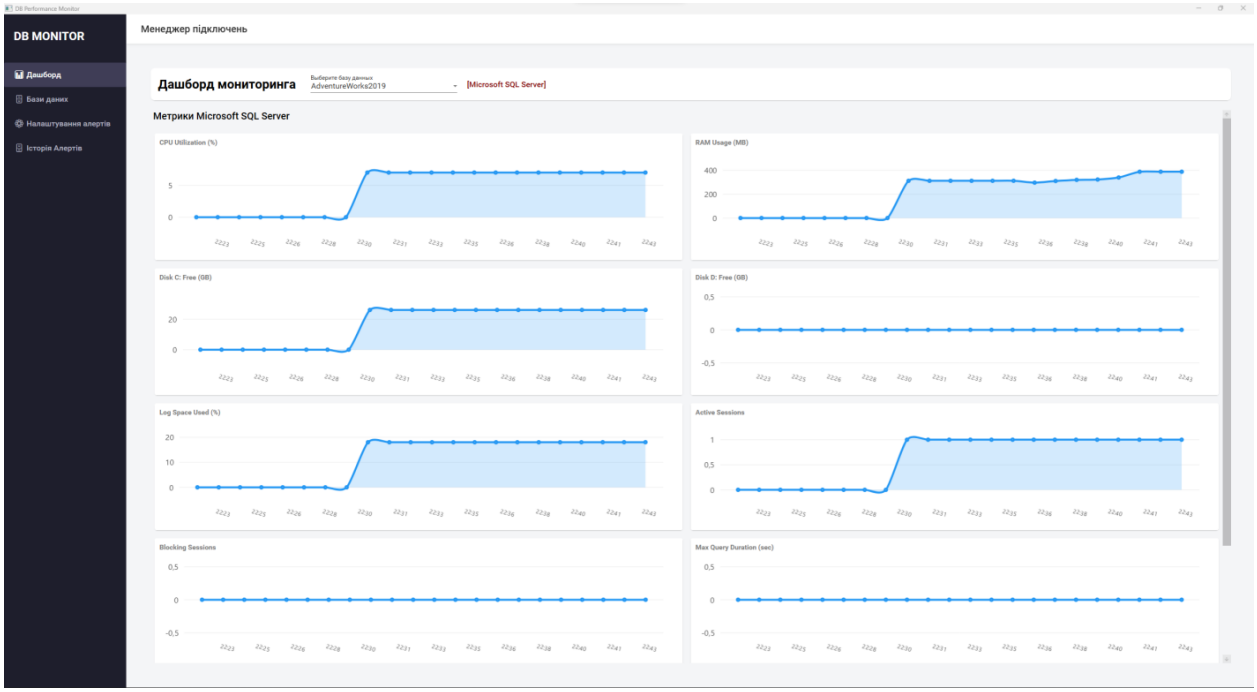


Рисунок 3.19 – Візуалізація метрик продуктивності баз даних у реальному часі в програмно-методичному комплексі для автоматизації обробки даних про ефективність роботи сучасних систем управління базами даних

Для рендерингу графіків використано передову бібліотеку *LiveChartsCore.SkiaSharpView*. На відміну від класичних засобів WPF, ця бібліотека використовує апаратне прискорення графічного рушія Skia, що дозволяє відмальовувати тисячі точок даних без затримок. У файлі *DashboardView.xaml.cs* реалізовано метод *StartAutoUpdate()*, який ініціалізує системний таймер *DispatcherTimer*. Кожні кілька секунд таймер виконує запит до БД SQLite, фільтрує дані поточного вибраного сервера за допомогою LINQ *(.Where(m => m.ServerName == ...))*, та формує масиви структур ObservablePoint. Ці об'єкти автоматично прив'язуються до властивості *Values* об'єктів *LineSeries*, що змушує графік плавно перемальовуватися (оновлюватися) на екрані.

Окрім цього, інтерфейс реалізує логіку адаптивності. Обробник події випадаючого списку *CmbDatabases_SelectionChanged* аналізує тип бази даних і динамічно перемикає властивість *Visibility* панелей *PanelMSSQL* або

PanelPostgreSQL. Таким чином, інтерфейс не перевантажує користувача нерелевантними метриками.

3.4.3 Інтерфейс налаштування лімітів чутливості (Thresholds)

Модуль *AlertSettingsView* надає адміністратору інструментарій для калібрування математичних моделей алертів. Так як моя програма розрахована на роботу з двома типами БД MS SQL та PostgreSQL і ці бази даних мають різні метрики, я розробив механізм, який автоматично визначає, який саме тип БД підключено і відштовхуючись від типу БД, виводить які саме метрики і пороги можна налаштовувати для цієї БД. На рисунках 3.20 та 3.21 зображені обидва інтерфейси для різних типів БД.

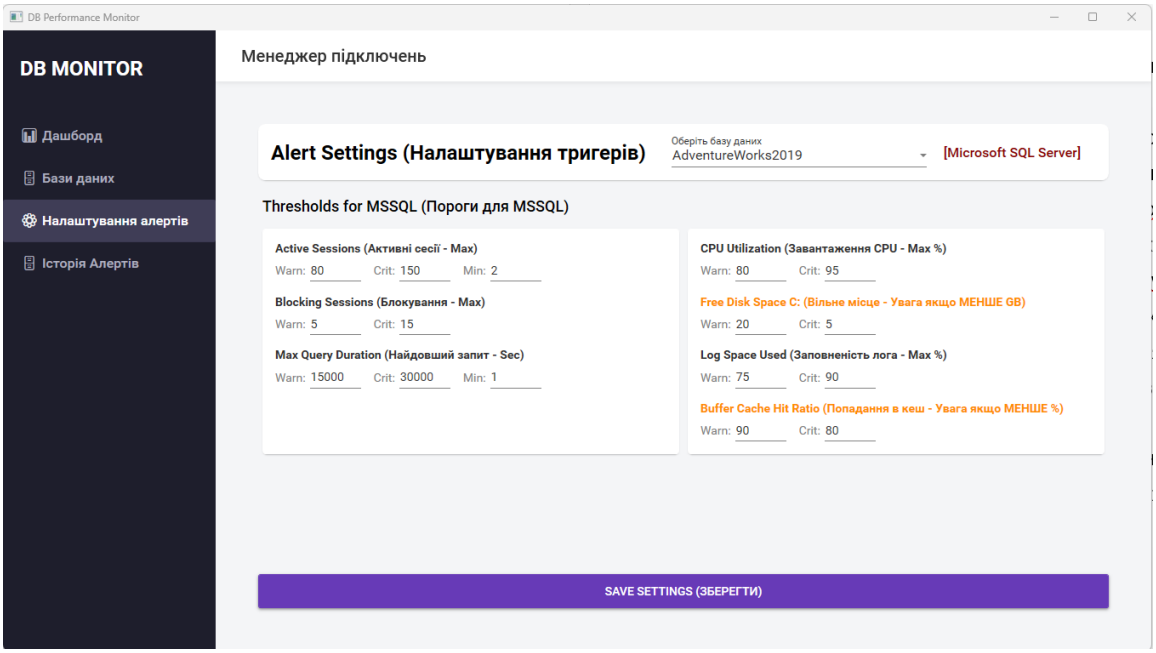


Рисунок 3.20 – Інтерфейс конфігурації порогів Warning/Critical для БД MS SQL в програмно-методичному комплексі для автоматизації обробки даних про ефективність роботи сучасних систем управління базами даних

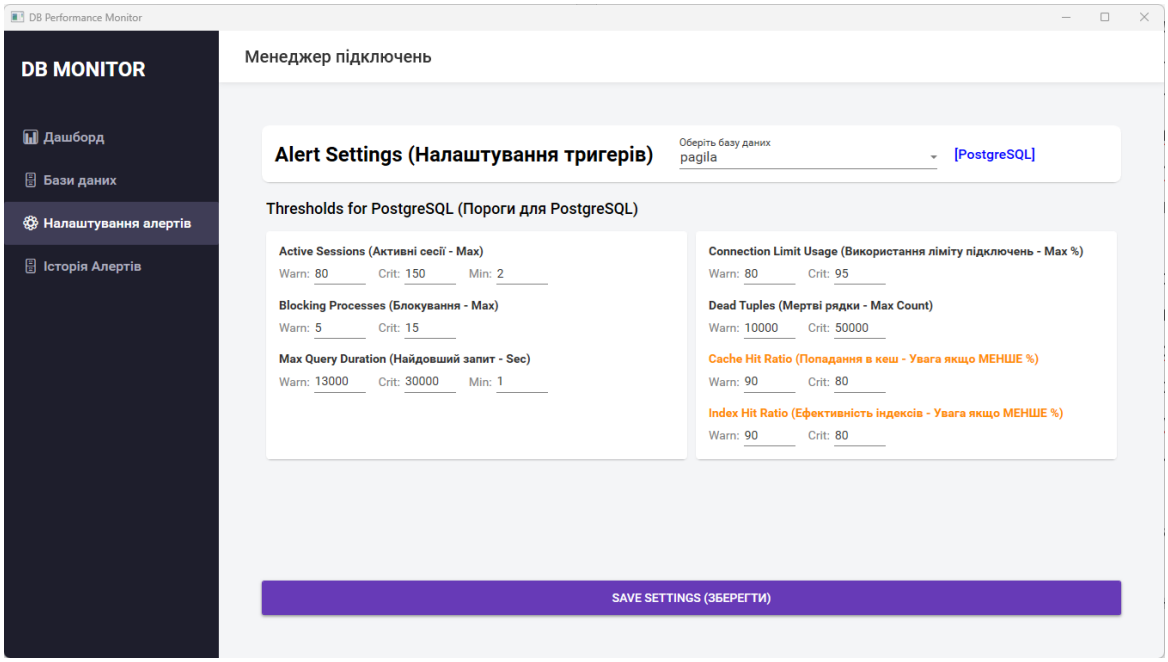


Рисунок 3.21 – Інтерфейс конфігурації порогів Warning/Criticall для БД PostrgeeSQL в програмно-методичному комплексі для автоматизації обробки даних про ефективність роботи сучасних систем управління базами даних

Візуальна форма побудована за допомогою карток *materialDesign:Card*, що групують налаштування за логічними блоками (наприклад, метрики сесій, метрики дискової підсистеми, мертві рядки). Для вводу параметрів використані елементи *TextBox* з інтегрованими підказками стилю *MaterialDesignFloatingHintTextBox*. При натисканні кнопки «ЗБЕРЕГТИ», метод *BtnSave_Click* виконує пошук відповідного запису в базі *AlertThresholds* (або *PgAlertThresholds*), валідує введений користувачем текст за допомогою методів *int.TryParse* та *double.TryParse* (що захищає програму від падіння при вводі некоректних символів), та синхронно записує оновлені конфігурації в репозиторій SQLite.

3.4.4 Модуль ретроспективного аналізу інцидентів

Для перегляду та аналізу зафіксованих аномалій розроблено модуль *AlertHistoryView (Рис 3.22)*. Основою візуалізації виступає табличний компонент *DataGrid,* налаштований у режимі *IsReadOnly=«True»* та з можливістю сортування по колонках.

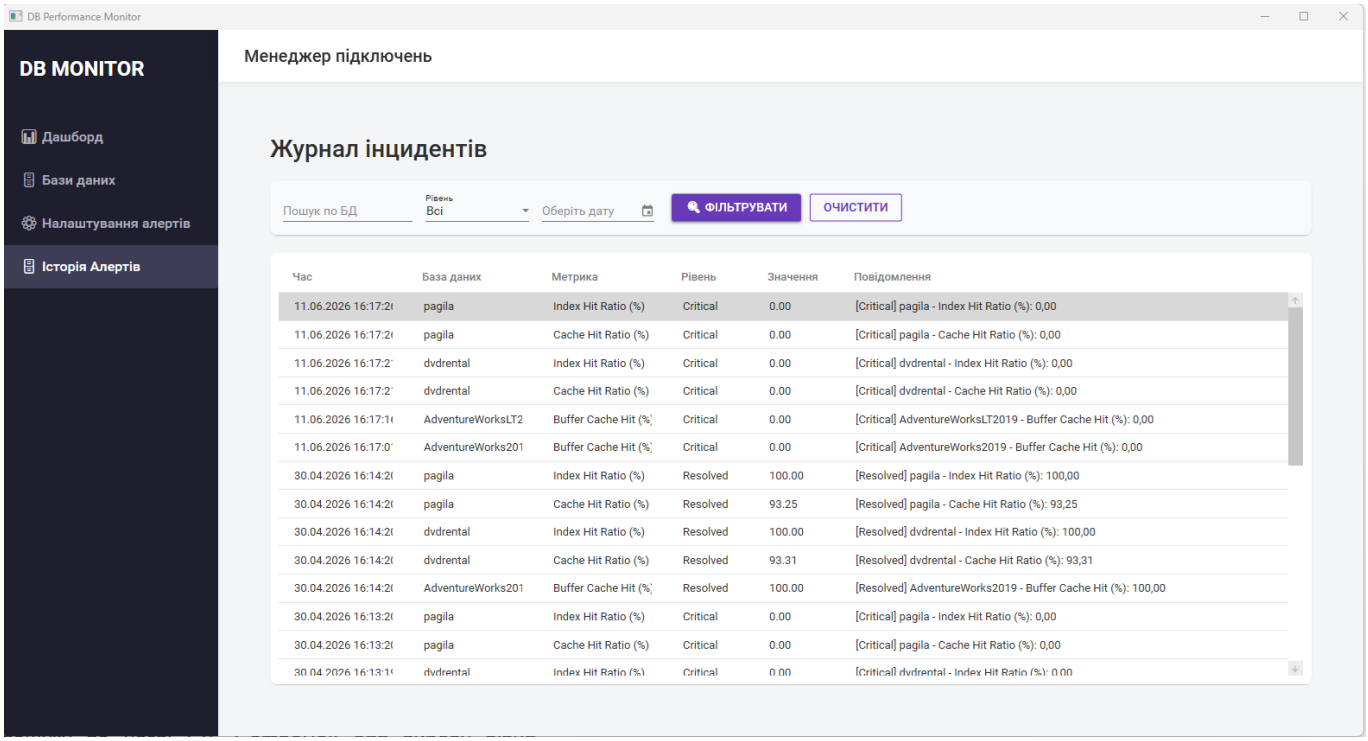


Рисунок 3.22 – Журнал реєстрації та фільтрації інцидентів в програмно-методичному комплексі для автоматизації обробки даних про ефективність роботи сучасних систем управління базами даних

Верхня частина екрану містить панель інструментів для фільтрації записів (*TextBox* для пошуку по назві БД, *ComboBox* для вибору рівня критичності та *DatePicker* для вибору дати). Логіка пошуку реалізована через гнучке застосування інтерфейсу *IQueryable* в *Entity Framework*. При натисканні на кнопку «ФІЛЬТРУВАТИ», код динамічно конструює дерево виразів

(Expression Tree), додаючи умови *Where* лише для тих полів фільтра, які були заповнені користувачем. Після формування запиту викликається метод *.Take(100).ToListAsync()*, який завантажує лише останні 100 записів. Такий підхід до пагінації запобігає вичерпанню оперативної пам'яті (Out of Memory Exception) у випадку, якщо таблиця журналу міститиме десятки тисяч рядків.

Окремим і обов'язковим етапом завершення реалізації програмно-методичного комплексу є розробка супровідної експлуатаційної документації. Детальний покроковий опис процесу розгортання додатка в середовищі ОС Windows, конфігурації підключень до серверів баз даних та взаємодії з графічним інтерфейсом користувача (Керівництво адміністратора) наведено у Додатку Б.

4 ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ РОЗРОБКИ ТА ВПРОВАДЖЕННЯ ПРОГРАМНОГО-МЕТОДИЧНОГО КОМПЛЕКСУ

4.1 Оцінка трудомісткості та планування етапів розробки програмно-методичного комплексу

Будь-який процес створення програмного забезпечення вимагає ретельного планування та оцінки витрат часу на кожен етап життєвого циклу (SDLC). Для розрахунку економічної ефективності програмно-методичного комплексу моніторингу баз даних, перш за все, необхідно визначити загальну трудомісткість його розробки [48].

Весь процес створення програмного продукту було декомпозировано на шість основних етапів. Оцінка часу проводилася за методом експертних оцінок (PERT) з урахуванням складності розробки багатопотокового фонового сервісу та інтеграції з системними представленнями MS SQL та PostgreSQL [49].

Таблиця 4.1 – Структурна декомпозиція робіт (WBS) та трудомісткість розробки програмно-методичного комплексу

№	Назва етапу розробки	Зміст робіт	Витрати часу (год)
1	Аналітичний етап	Аналіз предметної області, збір метрик (DMV, pg_stat), складання ТЗ.	40
2	Проєктування БД	Розробка ER-моделі бази даних, налаштування Entity Framework Core для SQLite.	32
3	Розробка Backend	Імплементація патерну «Стратегія», розробка WorkerService та «вікна толерантності».	104
4	Розробка Frontend	Верстка XAML-інтерфейсу (WPF), MVVM, підключення бібліотеки LiveChartsCore.	88

Продовження таблиці 4.1

№	Назва етапу розробки	Зміст робіт	Витрати часу (год)
5	Тестування	Проведення стрес-тестів бази даних, відладка багатопотоковості.	40
6	Документування	Написання інструкції адміністратора, оформлення пояснювальної записки.	16
Всього			320

Загальна трудомісткість розробки програмно-методичного комплексу склала $T = 320$ людино-годин, що еквівалентно 40 робочим дням (2 календарні місяці при 8-годинному робочому дні). На основі цих даних побудовано календарний графік проєкту зображений на рисунку 4.1.

Рисунок 4.1 – Календарний графік розробки програмно-методичного комплексу для автоматизації обробки даних про ефективність роботи сучасних систем управління базами даних

4.2 Розрахунок капітальних та операційних витрат на створення програмно-методичного комплексу

Собівартість створення програмного продукту ($С$) складається із суми прямих та накладних витрат і розраховується згідно з класичними моделями економіки ПЗ [50] за формулою:

$$С = В_{\{зп\}} + В_{\{відр\}} + В_{\{ам\}} + В_{\{ел\}} + В_{\{ін\}},$$

де $В_{зп}$ – витрати на заробітну плату розробника; $В_{відр}$ – відрахування на соціальні заходи; $В_{ам}$ – амортизаційні відрахування; $В_{ел}$ – витрати на електроенергію; $В_{ін}$ – інші накладні витрати.

4.2.1 Розрахунок фонду оплати праці та податків

Основою витрат у сфері IT є інтелектуальна праця. Для розрахунків приймається середня ринкова ставка розробника (рівня Junior .NET/C#) у розмірі $С_г = 320$ грн/год.

Основна заробітна плата ($ЗП_{осн}$) розраховується як добуток ставки на трудомісткість:

$$ЗП_{осн} = С_г \times T = 320 \times 320 = 102\,400 \text{ грн.}$$

Додаткова заробітна плата ($ЗП_{дод}$), яка включає компенсації та відпускні, нормативно приймається на рівні 10% від основної [51]:

$$ЗП_{дод} = ЗП_{осн} \times 0.10 = 10\,240 \text{ грн.}$$

Загальний фонд оплати праці (ФОП):

$$\text{ФОП} = 102\,400 + 10\,240 = 112\,640 \text{ грн.}$$

Відрахування на соціальні заходи (ЄСВ) згідно із законодавством України становить 22% від ФОП:

$$\text{В}_{\text{відр}} = \text{ФОП} \times 0.22 = 112\,640 \times 0.22 = 24\,780{,}80 \text{ грн.}$$

4.2.2 Розрахунок амортизації обладнання та накладних витрат

Процес розробки вимагає сучасного апаратного забезпечення. Вартість робочої станції приймається рівною $\text{К}_{\text{обл}} = 45\,000$ грн. Амортизація ($\text{В}_{\text{ам}}$) розраховується лінійним методом. Нормативний термін служби ПК становить 4 роки (48 місяців) [50].

Місячна норма амортизації: $45\,000 / 48 = 937{,}50$ грн/міс.

Оскільки розробка тривала 2 місяці, сума амортизації становить:

$$\text{В}_{\text{ам}} = 937{,}50 \times 2 = 1\,875{,}00 \text{ грн.}$$

Витрати на електроенергію ($\text{В}_{\text{ел}}$) залежать від потужності ПК ($P = 0{,}4$ кВт), часу роботи ($T = 320$ год) та комерційного тарифу ($\text{Т}_{\text{ел}} = 8{,}50$ грн/кВт-год):

$$\text{В}_{\text{ел}} = 0{,}4 \times 320 \times 8{,}50 = 1\,088{,}00 \text{ грн.}$$

Витрати на інтернет та хмарні сервіси ($В_{ін}$) за 2 місяці приймаються на рівні 1 200,00 грн.

Таблиця 4.2 – Кошторис витрат на розробку програмно-методичного комплексу

Стаття витрат	Сума, грн	Частка у структурі (%)
Основна та додаткова заробітна плата (ФОП)	112 640,00	79,56%
Відрахування на соціальні заходи (ЄСВ 22%)	24 780,80	17,50%
Амортизація комп'ютерної техніки	1 875,00	1,32%
Витрати на електроенергію	1 088,00	0,77%
Інші накладні витрати (Інтернет)	1 200,00	0,85%
Загальна кошторисна вартість розробки (С)	141 583,80	100,00%

Отже, капітальні інвестиції ($К$) у розробку власного рішення становлять 141 583,80 грн.

4.3 Порівняльний аналіз сукупної вартості володіння (TCO)

Для об'єктивної оцінки доцільності програмно-методичного комплексу, порівняємо його з альтернативами за методологією сукупної вартості володіння (Total Cost of Ownership, TCO) [52] за 1 рік експлуатації.

Сценарій 1 (Власний програмно-методичний комплекс): Витрати дорівнюють собівартості розробки (141 583 грн).

Сценарій 2 (Zabbix): Хоча Zabbix є Open-Source, він вимагає окремого сервера (45 000 грн). Базові шаблони не аналізують Dead Tuples чи Buffer Cache. Написання кастомних скриптів DevOps-інженером (120 год × 450 грн/год + податки) обійдеться у 65 880 грн. Операційна підтримка (10 год/міс) коштуватиме ще 65 880 грн/рік.

Сценарій 3 (SolarWinds DPA): Ліцензія на 4 сервери БД коштує 8000/рік (бл. 328 000 грн) + виділений сервер (45 000 грн) + інтеграція (15 000 грн).

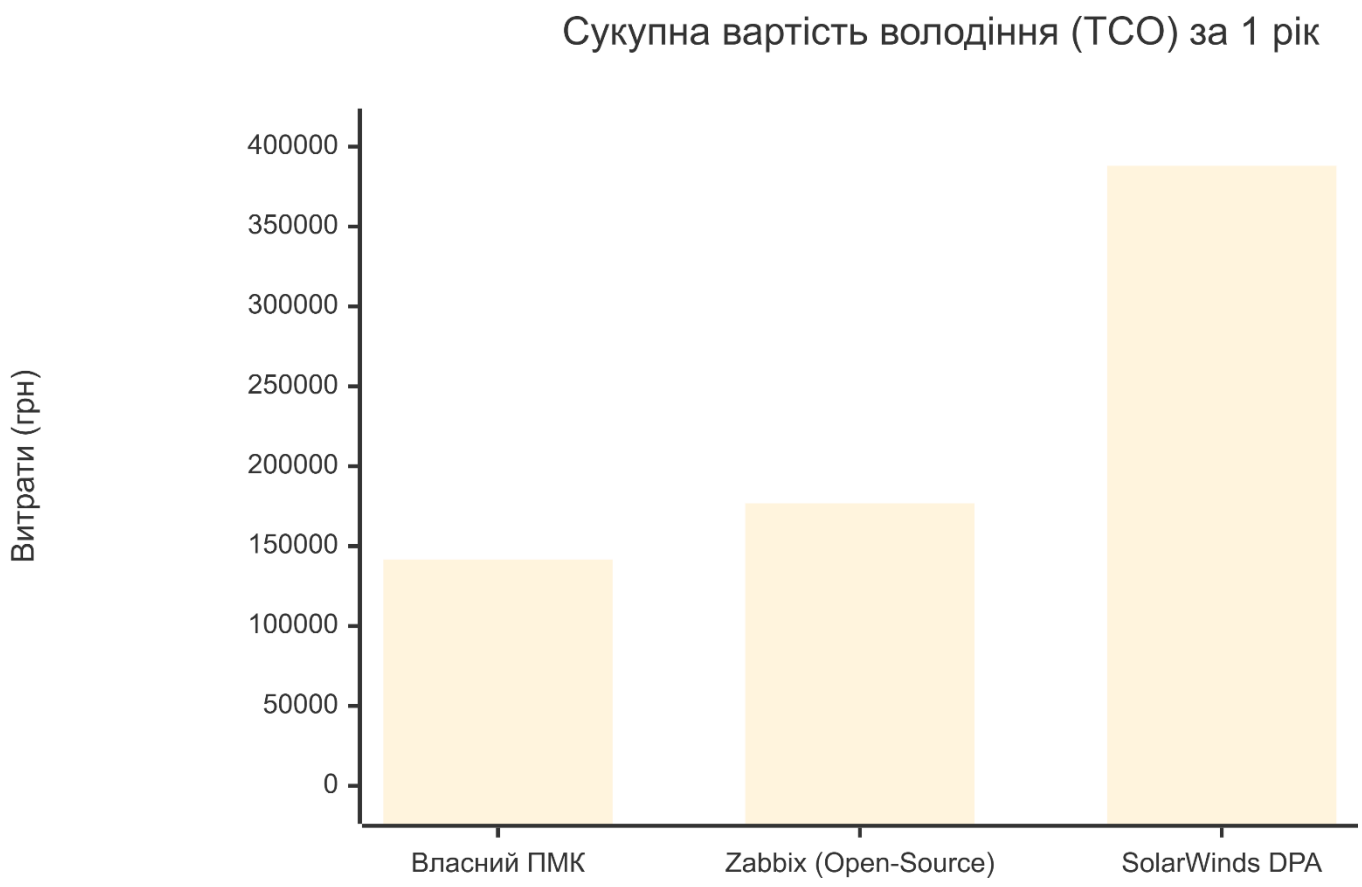


Рисунок 4.2 – Порівняння TCO систем моніторингу за 1 рік (у гривнях)

Висновок: Розроблений програмно-методичний комплекс на 20% дешевший за «безкоштовний» Zabbix (завдяки відсутності потреби у виділеному сервері та постійній підтримці DevOps) та майже втричі дешевший за комерційні Enterprise-рішення.

4.4 Розрахунок економічної ефективності від впровадження програмно-методичного комплексу

Головна економічна цінність програмно-методичного комплексу полягає у мінімізації збитків від простою (Downtime Cost) [53]. Для моделювання ефективності застосуємо сценарій для середнього логістичного підприємства, де вартість операційної діяльності становить 100 000 грн за один 8-годинний робочий день (або 12 500 грн/год).

Без системи моніторингу: Деградація БД (наприклад, блокування транзакцій) розвивається непомітно. Адміністратор дізнається про інцидент від користувачів. Час реакції та відновлення складає $T_{\text{відн}} = 3$ год. Збиток від однієї аварії: $З_{\text{без}} = 3 \times 12\,500 = 37\,500$ грн. При 2 інцидентах на місяць збитки становлять $75\,000$ грн/міс.

З розробленим програмно-методичним комплексом: Завдяки алгоритму «вікна толерантності» система фіксує проблему на стадії зародження і надсилає Email-алерт. Час реакції скорочується до 15 хвилин (0,25 год). Збиток від купірованого інциденту: $0{,}25 \times 12\,500 = 3\,125$ грн. Втрати за місяць: $6\,250$ грн.

Прямий економічний ефект (Економія ΔE):

$$\Delta E = 75\,000 - 6\,250 = 68\,750 \text{ грн/міс.}$$

Річна економія:

$$E_{\text{рік}} = 68\,750 \times 12 = 825\,000 \text{ грн.}$$

4.5 Розрахунок терміну окупності (ROI) та точки беззбитковості

Застосуємо класичні інвестиційні метрики [53] для оцінки фінансової доцільності проєкту.

Коефіцієнт рентабельності інвестицій (ROI):

$$ROI = \frac{\text{Е}_{\text{рік}} - \text{К}}{\text{К}} \times 100\% = \frac{825\,000 - 141\,583{,}80}{141\,583{,}80} \times 100\% = 482{,}7\%$$

Значення ROI > 480% свідчить про феноменальну економічну ефективність алгоритмів моніторингу.

Термін окупності проєкту (Payback Period, PP):

$$PP = \frac{\text{К}}{\Delta\text{Е}} = \frac{141\,583{,}80}{68\,750} \approx 2{,}06 \text{ місяця.}$$

На основі цих розрахунків побудовано графік точки беззбитковості (рис. 4.3). На графіку:

– Нижня діагональна лінія демонструє зростання зекономлених коштів.

– Горизонтальна лінія – капітальні витрати.

– Точка їх перетину показує, що на початку третього місяця проєкт повністю окуповується.

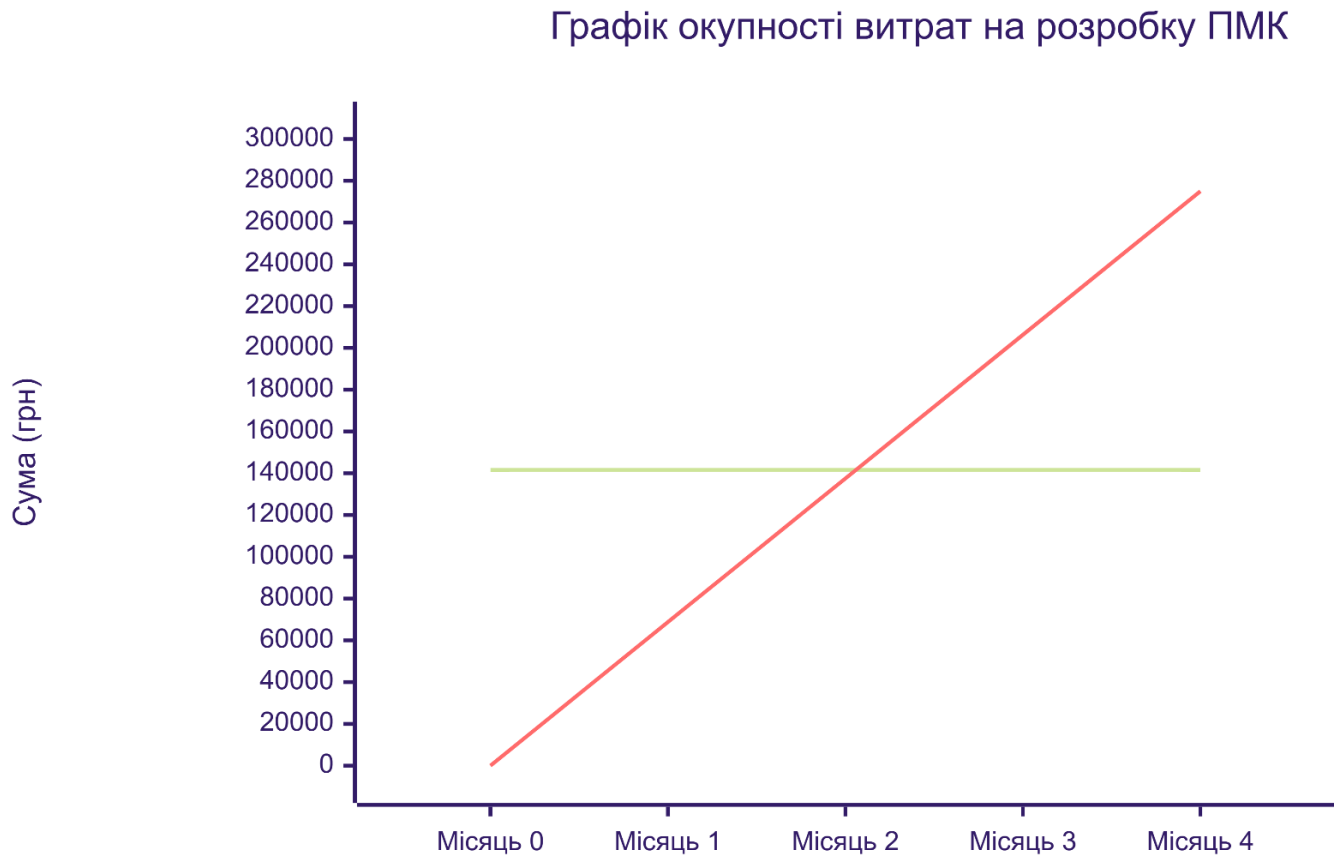


Рисунок 4.3 – Графік точки беззбитковості (Break-even point) (Примітка:).

4.6 Аналіз ризиків (SWOT-аналіз)

Для комплексного обґрунтування проведено факторний аналіз ризиків програмно-методичного комплексу [54].

Таблиця 4.4 – SWOT-аналіз програмного продукту

Сильні сторони (S)	Слабкі сторони (W)
• Безагентська архітектура (мінімальне навантаження на БД). • Відсутність прихованих ліцензійних платежів. • Математично-точний алгоритм «вікна толерантності».	• Залежність UI від екосистеми Windows (WPF). • Відсутність мобільного або Web-клієнта.
Можливості (O)	Загрози (T)
• Легке розширення завдяки патерну «Стратегія». • Інтеграція підсистеми сповіщень з Telegram API.	• Зміна архітектури системних каталогів у нових версіях СУБД. • Блокування поштових SMTP-портів фаєрволом.

Розробка власного програмно-методичного комплексу є економічно виправданим рішенням. Собівартість проєкту склала 141 583,80 грн, що значно нижче сукупної вартості розгортання альтернативних систем (Zabbix або SolarWinds). Завдяки запобіганню аварійних зупинок баз даних, програма економить підприємству понад 68 тис. грн щомісяця, повністю окупаючи себе за 2 місяці експлуатації.

ВИСНОВКИ

У кваліфікаційній роботі розв’язано актуальне науково-практичне завдання щодо розробки та впровадження програмно-методичного комплексу (ПМК) для автоматизованого безагентського моніторингу та аналізу продуктивності реляційних систем управління базами даних.

Мету роботи, яка полягала у створенні інструментарію для автоматизованого збору, аналізу та візуалізації телеметричних даних СУБД для підвищення ефективності адміністрування, повністю досягнуто.

Відповідно до поставлених завдань отримано такі основні результати:

1. Здійснено ґрунтовний аналіз предметної області та досліджено архітектурні особливості функціонування сучасних реляційних СУБД, зокрема PostgreSQL та Microsoft SQL Server. Доведено, що традиційні методи ручного контролю та використання статичних математичних порогів не забезпечують достатньої оперативності реагування на інциденти в умовах високонавантажених (HighLoad) систем.

2. Побудовано комплекс логічних та функціональних моделей процесів автоматизованого моніторингу баз даних. Застосування методології структурного аналізу SADT (стандарт IDEF0) дозволило системно декомпозувати процеси обробки телеметрії. Використання об'єктно-орієнтованого підходу та діаграм UML забезпечило надійне архітектурне проєктування програмного забезпечення.

3. Сформовано математичну модель аналізу показників продуктивності, яка базується на апараті теорії масового обслуговування (зокрема, законі Літтла) для оцінки черг запитів. Впровадження алгоритму експоненційного згладжування (EWMA) спільно з логічним механізмом «вікна толерантності» дозволило ефективно фільтрувати мікросекундні сплески навантаження та звести до мінімуму кількість хибних спрацювань підсистеми сповіщень.

4. Здійснено практичну реалізацію розроблених моделей у вигляді десктопного програмно-методичного комплексу мовою програмування C# на платформі .NET 8. Для побудови сучасного графічного інтерфейсу використано технологію WPF, а процеси збереження телеметрії в локальному архіві реалізовано за допомогою СУБД SQLite та технології ORM Entity Framework Core. Для захисту облікових даних успішно застосовано криптографічний інтерфейс ОС Windows (DPAPI).

5. Проведено навантажувальне тестування комплексу, яке підтвердило здатність розробленої системи стабільно збирати метрики, виявляти аномалії та генерувати оперативні Email-сповіщення без створення надмірного навантаження на об'єкти моніторингу.

6. Виконано техніко-економічне обґрунтування доцільності впровадження розробленого програмного забезпечення. Розрахунки довели, що розробка власного програмно-методичного комплексу (собівартість 141 583,80 грн) є економічно вигіднішою за розгортання альтернативних систем (наприклад, Zabbix або SolarWinds). Завдяки превентивному виявленню інцидентів та мінімізації часу простою серверів, програма здатна економити підприємству понад 68 тис. грн щомісяця. Проєкт характеризується високою інвестиційною привабливістю: термін окупності становить близько 2 місяців, а показник рентабельності (ROI) досягає 482,7%.

Практичне значення отриманих результатів полягає у тому, що створений програмно-методичний комплекс є готовим інструментом для адміністраторів баз даних. Впровадження комплексу дозволяє централізовано контролювати стан серверної інфраструктури, накопичувати історичні дані для ретроспективного аналізу та оперативно реагувати на критичні інциденти, що в підсумку підвищує стабільність та безперервність роботи інформаційних систем підприємства.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Schwab, K. The Fourth Industrial Revolution. New York : Crown Business, 2017. 192 p.

2. Newman, S. Building Microservices: Designing Fine-Grained Systems. 2nd ed. Sebastopol : O'Reilly Media, 2021. 614 p.

3. Beyer, B.; Jones, C.; Petoff, J.; Murphy, N. R. Site Reliability Engineering: How Google Runs Production Systems. Sebastopol : O'Reilly Media, 2016. 550 p.

4. Morris, K. Infrastructure as Code: Managing Servers in the Cloud. 2nd ed. Sebastopol : O'Reilly Media, 2020. 416 p.

5. Turnbull, J. The Art of Monitoring. Docker, 2014. 350 p.

6. Petrov, A. Database Internals: A Deep Dive into How Distributed Data Systems Work. Sebastopol : O'Reilly Media, 2019. 376 p.

7. Coronel, C.; Morris, S. Database Systems: Design, Implementation, & Management. 13th ed. Boston : Cengage Learning, 2018. 832 p.

8. Sadalage, P. J.; Fowler, M. NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Upper Saddle River : Addison-Wesley Professional, 2012. 192 p.

9. Kyte, T.; Kuhn, D. Expert Oracle Database Architecture. 3rd ed. New York : Apress, 2014. 940 p.

10. JetBrains. Oracle / DataGrip Features. URL: https://www.jetbrains.com/datagrip/features/oracle/ (дата звернення: 22.05.2026).

11. Schwartz, B.; Zaitsev, P.; Tkachenko, V. High Performance MySQL. 3rd ed. Sebastopol : O'Reilly Media, 2012. 826 p.

12. AquaFold. MySQL Client / Aqua Data Studio. URL: https://aquadatastudio.com/databases/mysql-client/ (дата звернення: 22.05.2026).

13. Chodorow, K. MongoDB: The Definitive Guide. 3rd ed. Sebastopol : O'Reilly Media, 2019. 514 p.

14. JetBrains. MongoDB / DataGrip Features. URL: https://www.jetbrains.com/datagrip/features/mongodb/ (дата звернення: 22.05.2026).

15. Carlson, J. Redis in Action. Shelter Island : Manning Publications, 2013. 328 p.

16. JetBrains. Redis / DataGrip Features. URL: https://www.jetbrains.com/datagrip/features/redis/ (дата звернення: 22.05.2026).

17. Bobak, A.; Candea, C. SQL Server 2022 Internals. Redmond : Microsoft Press, 2023. 600 p.

18. Ward, B. Pro SQL Server on Linux. New York : Apress, 2018. 488 p.

19. JetBrains. SQL Server / DataGrip Features. URL: https://www.jetbrains.com/datagrip/features/sqlserver/ (дата звернення: 22.05.2026).

20. Smith, G. PostgreSQL 9.0 High Performance. Birmingham : Packt Publishing, 2010. 468 p.

21. The PostgreSQL Global Development Group. PostgreSQL 16 Documentation: Chapter 13. Concurrency Control. URL: https://www.postgresql.org/docs/16/mvcc.html (дата звернення: 22.05.2026).

22. Riggs, S.; Krosing, H. PostgreSQL 10 High Performance. Birmingham : Packt Publishing, 2018. 536 p.

23. Гнєденко, Б. В.; Коваленко, І. М. Вступ до теорії масового обслуговування. Київ : Наукова думка, 2005. 400 с.

24. Seidl M., Scholz M., Huemer C., Kappel G. UML @ Classroom: An Introduction to Object-Oriented Modeling. Cham : Springer, 2015. 206 p.

25. Fowler, M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. 3rd ed. Boston : Addison-Wesley Professional, 2003. 208 p.

26. Booch, G.; Rumbaugh, J.; Jacobson, I. The Unified Modeling Language User Guide. 2nd ed. Boston : Addison-Wesley, 2005. 496 p.

27. Gregg, B. Systems Performance: Enterprise and the Cloud. 2nd ed. Boston : Addison-Wesley Professional, 2020. 880 p.

28. Grafana Labs. Volz, T. The RED Method: How To Instrument Your Services. URL: https://grafana.com/blog/2018/08/02/the-red-method-how-to-instrument-your-services/ (дата звернення: 22.05.2026).

29. Karwin, B. SQL Antipatterns: Avoiding the Pitfalls of Database Programming. Raleigh : Pragmatic Bookshelf, 2010. 328 p.

30. Perkins, B. Entity Framework Core in Action. 2nd ed. Shelter Island : Manning Publications, 2021. 576 p.

31. Mouat, A. Using Docker: Developing and Deploying Software with Containers. Sebastopol : O'Reilly Media, 2015. 354 p.

32. Bovet, D. P.; Cesati, M. Understanding the Linux Kernel. 3rd ed. Sebastopol : O'Reilly Media, 2005. 944 p.

33. Humble, J.; Farley, D. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Boston : Addison-Wesley, 2010. 512 p.

34. Brooks, F. P. The Mythical Man-Month: Essays on Software Engineering. Boston : Addison-Wesley Professional, 1995. 336 p.

35. Marca, D. A.; McGowan, C. L. SADT: Structured Analysis and Design Technique. New York : McGraw-Hill, 1988. 392 p.

36. Troelsen, A.; Japikse, P. Pro C# 10 with .NET 6: Foundational Principles and Practices in .NET. New York : Apress, 2022. 1222 p.

37. Nathan, A. WPF 4.5 Unleashed. Indianapolis : Sams Publishing, 2013. 864 p.

38. Owens, M.; Allen, G. SQLite. New York : Apress, 2010. 360 p.

39. Smith, J. Entity Framework Core Cookbook. 2nd ed. Birmingham : Packt Publishing, 2020. 450 p.

40. Little, J. D. C. A Proof for the Queuing Formula: L = λ W. *Operations Research.* 1961. Vol. 9, No. 3. P. 383–387.

41. Montgomery, D. C.; Jennings, C. L.; Kulahci, M. Introduction to Time Series Analysis and Forecasting. 2nd ed. Hoboken : Wiley, 2015. 672 p.

42. Jain, R. The Art of Computer Systems Performance Analysis. New York : Wiley, 1991. 720 p.

43. Myers, G. J.; Sandler, C.; Badgett, T. The Art of Software Testing. 3rd ed. Hoboken : John Wiley & Sons, 2011. 240 p.

44. The PostgreSQL Global Development Group. pgbench – PostgreSQL 16 Documentation. URL: https://www.postgresql.org/docs/16/pgbench.html (дата звернення: 22.05.2026).

45. Machanic, A. SQLQueryStress // GitHub. URL: https://github.com/ErikEJ/SqlQueryStress (дата звернення: 22.05.2026).

46. Pressman, R. S., Maxim, B. R. Software Engineering: A Practitioner's Approach. 9th ed. New York : McGraw-Hill Education, 2019. 976 p.

47. ISO/IEC/IEEE 29148:2018. Systems and software engineering – Life cycle processes – Requirements engineering. Geneva : ISO, 2018. 104 p.

48. Боем Б. Економіка програмної інженерії. Методи оцінки вартості та управління проектами. Київ: Техніка, 2018. 340 с.

49. McConnell S. Software Estimation: Demystifying the Black Art. Redmond : Microsoft Press, 2006. 336 p.

50. Schwalbe K. Information Technology Project Management. 9th ed. Boston : Cengage Learning, 2018. 672 p.

51. Податковий кодекс України від 02.12.2010 № 2755-VI (зі змінами та доповненнями). URL: https://zakon.rada.gov.ua/laws/show/2755-17 (дата звернення: 22.05.2026).

52. Peppard J., Ward J. The Strategic Management of Information Systems: Building a Digital Strategy. 4th ed. Hoboken : John Wiley & Sons, 2016. 632 p.

53. Brigham E. F., Houston J. F. Fundamentals of Financial Management. 16th ed. Boston : Cengage Learning, 2021. 1088 p.

54. Вініченко О. М. Система динамічного контролю соціально-економічного розвитку підприємства: дис. д-ра екон. наук. Дніпро, 2017. 424 с.

ДОДАТОК А. ВІДОМОСТІ РОБОТИ

Таблиця А.1 – Відомості роботи

Формат	№ п/п	Назва документу	Найменування об’єкту або вибору	Кількість сторінок
А4	1	Пояснювальна записка	КЦТПАР.122-22-1п.00.00.00.ПЗ	112
Графічна частина				
А4	2	Мета, об'єкт та предмет дослідження	КЦТПАР.122-22-1п.01.00.00. ПЛ	1
А4	3	Аналіз цільових систем (MS SQL та PostgreSQL)	КЦТПАР.122-22-1п.02.00.00. ПЛ	1
А4	4	Функціональне моделювання системи (IDEF0)	КЦТПАР.122-22-1п.03.00.00. ПЛ	1
А4	5	Архітектура програмно-методичного комплексу	КЦТПАР.122-22-1п.04.00.00. ПЛ	1
А4	6	Математична модель аналізу та детекції аномалій	КЦТПАР.122-22-1п.05.00.00. ПЛ	1
А4	7	Логіка підсистеми сповіщень (Кінцевий автомат)	КЦТПАР.122-22-1п.06.00.00. ПЛ	1
А4	8	Фізична модель бази даних (SQLite)	КЦТПАР.122-22-1п.07.00.00. ПЛ	1
А4	9	Програмна реалізація ядра збору даних	КЦТПАР.122-22-1п.08.00.00. ПЛ	3
А4	10	Графічний інтерфейс: Менеджер підключень	КЦТПАР.122-22-1п.09.00.00. ПЛ	1

Продовження таблиці А.1

Формат	№ п/п	Назва документу	Найменування об’єкту або вибору	Кількість сторінок
А4	11	Графічний інтерфейс: Дашборд реального часу	КЦТПАР.122-22-1п.10.00.00. ПЛ	1
А4	12	Графічний інтерфейс: Менеджер підключень	КЦТПАР.122-22-1п.11.00.00. ПЛ	1
А4	13	Економічне обґрунтування розробки	КЦТПАР.122-22-1п.12.00.00. ПЛ	1
А4	14	Економічна ефективність та окупність	КЦТПАР.122-22-1п.13.00.00. ПЛ	1

ДОДАТОК Б. ТЕХНІЧНЕ ЗАВДАННЯ НА РОЗРОБКУ ПРОГРАМНО-МЕТОДИЧНОГО КОМПЛЕКСУ АВТОМАТИЗОВАНОЇ ОБРОБКИ ДАНИХ ПРО ЕФЕКТИВНІСТЬ СУБД

1. Загальні відомості

1.1. Повне найменування системи та її умовне позначення Повне найменування: Програмно-методичний комплекс автоматизованої обробки даних про ефективність систем управління базами даних. Умовне позначення: ПМК або Комплекс.

1.2. Підстави для розробки Розробка ведеться на підставі завдання на виконання кваліфікаційної (дипломної) роботи здобувача вищої освіти.

1.3. Призначення документа Дане Технічне завдання (ТЗ) визначає вимоги до архітектури, функціональності, надійності та умов експлуатації ПМК, а також регламентує порядок його розробки, тестування та приймання.

2. Призначення та цілі створення системи

2.1. Призначення системи ПМК призначений для автоматизації процесів збору, нормалізації, аналізу та візуалізації телеметричних даних із віддалених серверів реляційних баз даних (PostgreSQL та Microsoft SQL Server) у режимі реального часу з метою предиктивного виявлення інцидентів та деградації продуктивності.

2.2. Цілі створення системи Головними цілями впровадження ПМК є:

- Скорочення часу реакції адміністратора на виникнення позаштатних ситуацій (Deadlocks, нестача пам'яті, пікові завантаження CPU).
- Забезпечення можливості ретроспективного аналізу інцидентів завдяки накопиченню історичних даних у локальному архіві.
- Уніфікація процесу моніторингу гетерогенних середовищ (MS SQL та PostgreSQL) через єдиний графічний інтерфейс.

3. Характеристика об’єктів автоматизації

Об'єктами автоматизованого моніторингу виступають сервери реляційних СУБД.

- Об'єкт типу 1: Microsoft SQL Server (версії 2019-2022), що функціонує в середовищі ОС Windows Server. Отримання телеметрії здійснюється через системні динамічні адміністративні представлення (Dynamic Management Views - DMV).
- Об'єкт типу 2: PostgreSQL (версії 14-16), що функціонує в середовищі ОС Linux. Отримання телеметрії здійснюється через системний каталог pg_stat_activity та суміжні представлення.

4. Вимоги до системи

4.1. Вимоги до структури та функціонування

Архітектура програмно-методичного комплексу має бути реалізована за принципами багатошарової структури (Layered Architecture) і включати такі підсистеми:

1. Підсистема конфігурації: забезпечення збереження налаштувань підключення до серверів та параметрів алертів.
2. Підсистема збору даних (Колектори): асинхронне опитування цільових БД з використанням патерну «Стратегія» для різних СУБД.
3. Підсистема аналітики: обробка метрик, обчислення похідних показників (Cache Hit Ratio, Page Life Expectancy) та згладжування часових рядів (алгоритм EWMA).
4. Підсистема візуалізації: формування графічного користувацького інтерфейсу на базі патерну MVVM.

4.2. Вимоги до функціональних можливостей

Комплекс повинен забезпечувати виконання наступних функцій:

- Додавання, редагування та видалення облікових записів серверів БД.
- Налаштування індивідуальних інтервалів опитування (від 5 секунд).

- Встановлення порогів чутливості («Warning» та «Critical») для обраних метрик з можливістю задання вікна толерантності (затримки спрацювання).
- Безперервний фоновий збір показників: завантаження CPU, використання RAM, кількість активних та заблокованих сесій, обсяг транзакцій.
- Збереження зібраної телеметрії у локальній базі даних.
- Відображення динаміки показників у вигляді лінійних графіків у реальному часі.
- Генерація візуальних сповіщень при фіксації відхилень, що перевищують задані пороги.

4.3. Вимоги до надійності

- ПМК повинен бути стійким до втрати мережевого з'єднання з об'єктом моніторингу: програма не повинна аварійно завершувати роботу, натомість має ініціювати алгоритм повторного підключення.
- Всі операції доступу до локального сховища даних повинні бути транзакційними для запобігання пошкодженню файлу БД у разі раптового відключення живлення робочої станції.

4.4. Вимоги до ергономіки та технічної естетики

- Інтерфейс користувача має бути розроблений мовою декларативної розмітки XAML.
- Усі тривалі мережеві запити повинні виконуватися асинхронно, не блокуючи головний потік інтерфейсу користувача (UI Thread).
- Графіки продуктивності повинні мати механізми масштабування та автоматичного прокручування часової шкали.

4.5. Вимоги до програмного та апаратного забезпечення

- Апаратні вимоги робочої станції: процесор архітектури x64 (не нижче 2.0 ГГц), обсяг оперативної пам'яті не менше 4 ГБ, наявність 500 МБ вільного дискового простору.
- Вимоги до ОС клієнта: Microsoft Windows 10 або Windows 11.
- Мова розробки та платформа: C#, платформа .NET 8.

- Технологія побудови GUI: WPF (Windows Presentation Foundation).
- СУБД локального архіву: SQLite (взаємодія через ORM Entity Framework Core 8).
- Драйвери доступу до СУБД: Npgsql для PostgreSQL, Microsoft.Data.SqlClient для MS SQL Server.

5. Склад і зміст робіт зі створення системи

Розробка програмно-методичного комплексу повинна виконуватись у такі етапи:

- Аналітичний етап: збір вимог, формування математичної та логічної моделей предметної області (SADT, UML).
- Етап проектування: розробка архітектури бази даних SQLite, проектування класів ядра системи.
- Етап реалізації (кодування): написання вихідного коду мовою C#, верстка XAML-інтерфейсу, налаштування міграцій Entity Framework.
- Етап тестування: проведення стендових випробувань (навантажувальне тестування) з використанням інструментів pgbench та SQLQueryStress.
- Етап документування: оформлення пояснювальної записки до кваліфікаційної роботи та підготовка інструкції користувача.

6. Порядок контролю та приймання системи

Приймання програмно-методичного комплексу здійснюється шляхом перевірки відповідності його функціоналу вимогам цього ТЗ. Проводяться наступні випробування:

- Автономне тестування: перевірка коректності виконання SQL-запитів до системних каталогів СУБД, коректність розрахунку математичних формул (дельта-значень, CHR).
- Комплексне тестування (Стрес-тест): перевірка роботи ПМК в умовах штучно згенерованого високого навантаження на тестові бази даних.

Успішним проходженням вважається автоматична генерація Алерта без зависання самої програми моніторингу.

- Перевірка архівації: перевірка цілісності даних у локальному файлі SQLite після доби безперервного моніторингу.

ДОДАТОК В. ІНСТРУКЦІЯ АДМІНІСТРАТОРА (КЕРІВНИЦТВО КОРИСТУВАЧА) ПРОГРАМНО-МЕТОДИЧНОГО КОМПЛЕКСУ

Призначення та умови експлуатації Програмно-методичний комплекс (ПМК) призначений для автоматизованого безагентського збору, аналізу та візуалізації телеметричних даних із віддалених серверів реляційних баз даних (PostgreSQL та Microsoft SQL Server) у режимі реального часу. Програма розповсюджується у вигляді портативного (Portable) додатка і не потребує втручання в системний реєстр чи складної інсталяції.

Вимоги до робочої станції адміністратора:

- Операційна система: Microsoft Windows 10, Windows 11 або Windows Server 2016 і новіше (архітектура x64).
- Програмна платформа: Встановлене середовище виконання .NET 8.0 Desktop Runtime.
- Мережевий доступ: Відкриті порти TCP 1433 (для MS SQL), TCP 5432 (для PostgreSQL) та TCP 587 (для відправки Email-сповіщень).

Встановлення та перший запуск:

1. Розпакуйте архів із виконуваними файлами ПМК у будь-яку зручну директорію на жорсткому диску (переконайтеся, що ваш користувач має права на читання та запис у цю папку).
2. Запустіть виконуваний файл DBMonitor.exe.
3. Під час першого запуску програма автоматично створить файл локальної бази даних monitor_data.db у кореневій папці для збереження налаштувань та історичних даних телеметрії.

Керування підключеннями до серверів баз даних

Для ініціалізації процесу моніторингу необхідно зареєструвати цільові сервери у системі.

1. У головному вікні програми перейдіть до розділу «Менеджер підключень» (через бокове меню зліва).

2. У правій частині робочої області заповніть форму додавання нового сервера:

Тип СУБД: оберіть зі списку Microsoft SQL Server або PostgreSQL.

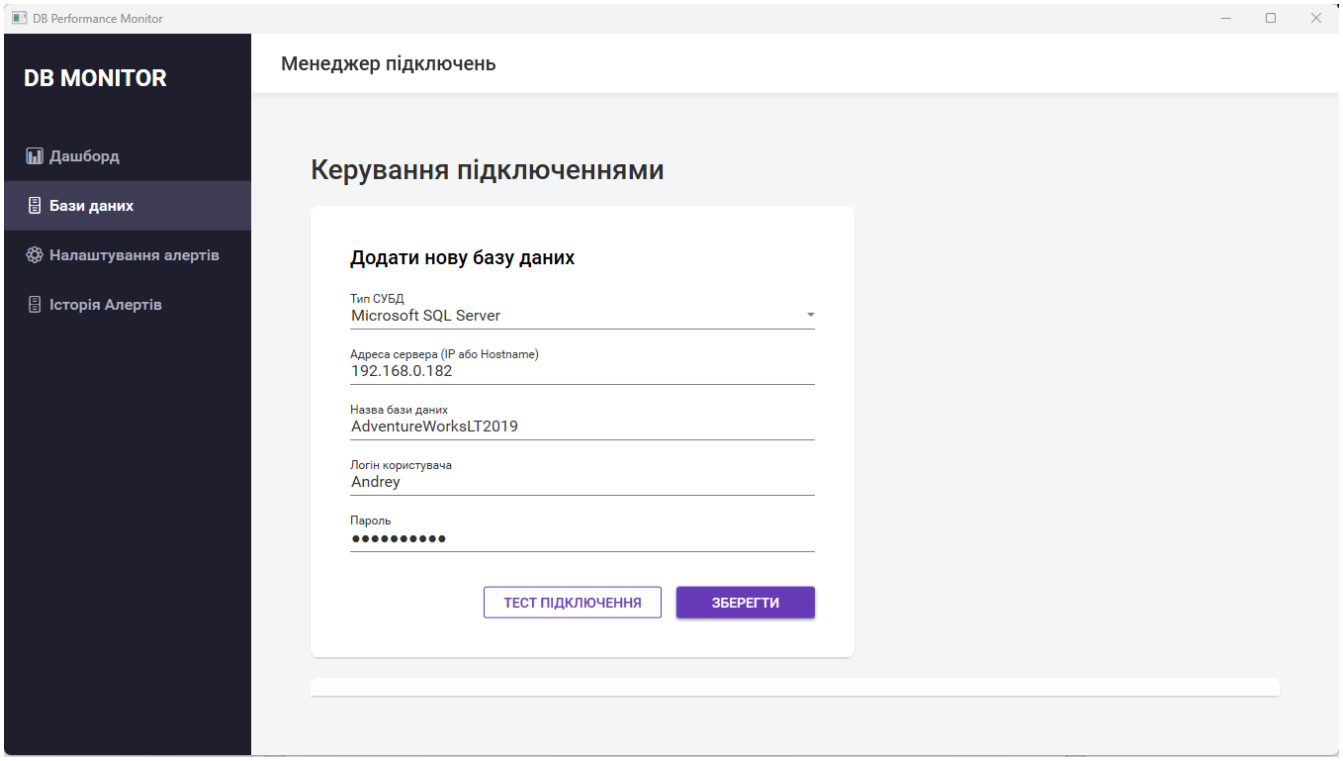
Адреса сервера: вкажіть IP-адресу або Hostname цільового сервера.

Назва бази даних: вкажіть конкретну БД для моніторингу або системну БД (наприклад, master чи postgres) для загального збору статистики.

Логін та Пароль: введіть облікові дані користувача з правами доступу до системних представлень (DMV або pg_stat). Примітка: пароль безпечно шифрується алгоритмами ОС Windows (DPAPI) та не зберігається у відкритому вигляді.

3. Натисніть кнопку «ТЕСТ ПІДКЛЮЧЕННЯ» для перевірки мережевого доступу та правильності облікових даних.

4. Натисніть кнопку «ЗБЕРЕГТИ». Доданий сервер з'явиться у списку зліва. Для активації фонового опитування переконайтеся, що для нього встановлено статус «Активний».



Налаштування порогів чутливості та алертів

Для коректної роботи підсистеми детекції аномалій необхідно відкалібрувати математичні ліміти для кожного сервера індивідуально.

1. Перейдіть до розділу «Налаштування алертів» у боковому навігаційному меню.
2. У верхньому випадаючому списку оберіть базу даних, яку бажаєте налаштувати. Інтерфейс автоматично адаптується під тип СУБД та відобразить відповідні метрики (наприклад, Dead Tuples для PostgreSQL або Page Life Expectancy для MS SQL Server).
3. Для кожної метрики встановіть два числові пороги:

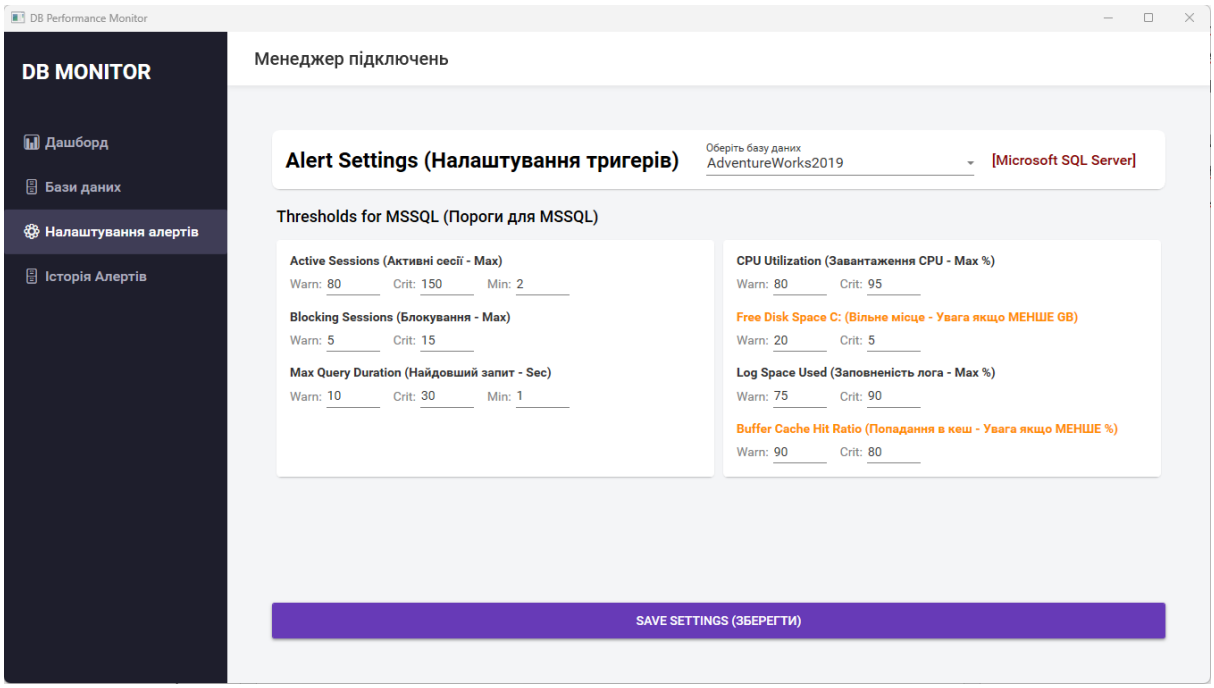
– Warning (Попередження): рівень підвищеного навантаження.

– Critical (Критичний): рівень аварійного стану інфраструктури.

4. Обов'язково вкажіть параметр «Вікно толерантності (хв)». Цей показник визначає, скільки хвилин поспіль метрика має перебувати в критичній

зоні до моменту фіксації інциденту та відправки Email-сповіщення (рекомендоване значення: 2–5 хвилин).

5. Натисніть кнопку «SAVE SETTINGS (ЗБЕРЕГТИ)».



Моніторинг інфраструктури в реальному часі (Дашборд)

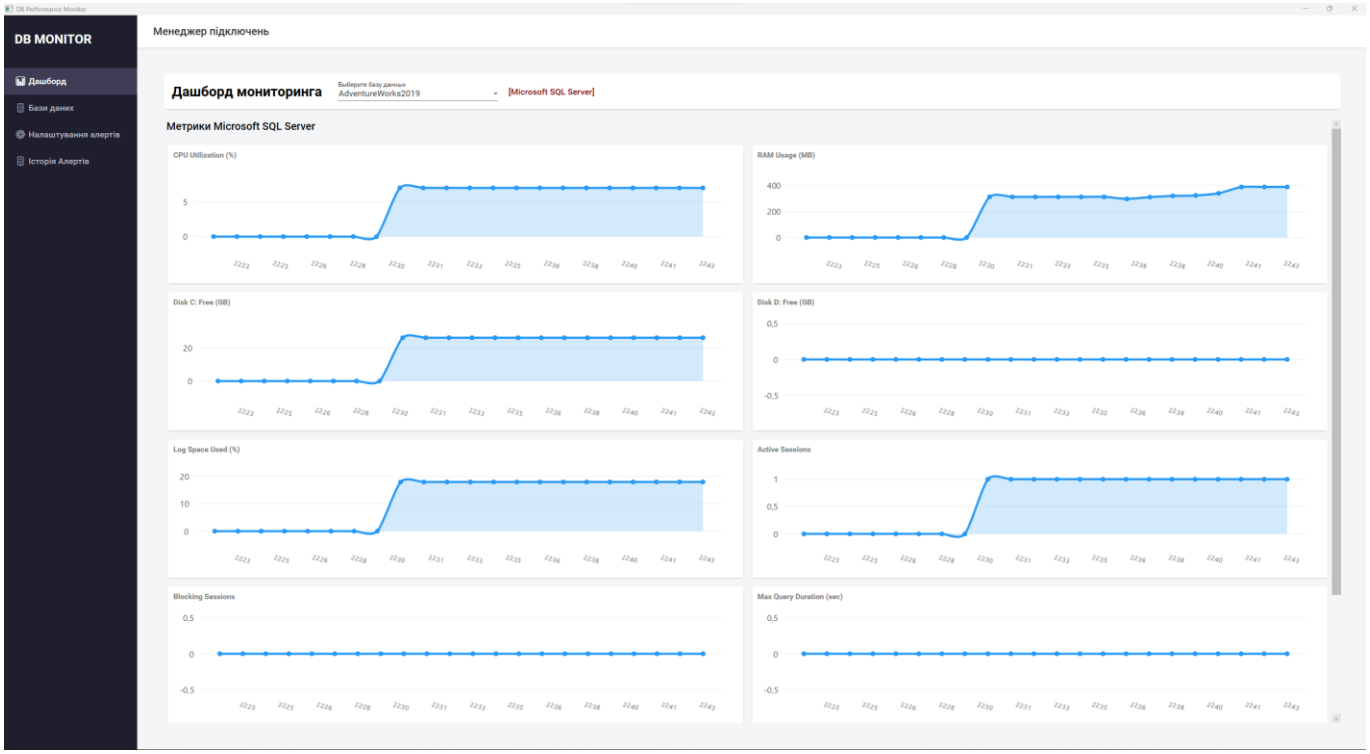
Розділ візуалізації призначений для оперативного спостереження за динамікою транзакційного навантаження.

1. У боковому меню оберіть розділ «Дашборд».

2. Зі списку у верхній частині екрана оберіть активний сервер баз даних.

3. На екрані з'явиться сітка інтерактивних графіків. Система автоматично кожні кілька секунд виконуватиме запит до локальної бази даних та плавно перемальовуватиме часові ряди показників (навантаження CPU, використання RAM, активні сесії тощо).

4. Для перегляду точного значення метрики у конкретну секунду, наведіть курсор миші на відповідну лінію графіка (з'явиться інформаційне вікно – Tooltip).



Ретроспективний аналіз інцидентів Система автоматично веде безперервний журнал усіх зафіксованих порушень стабільності СУБД.

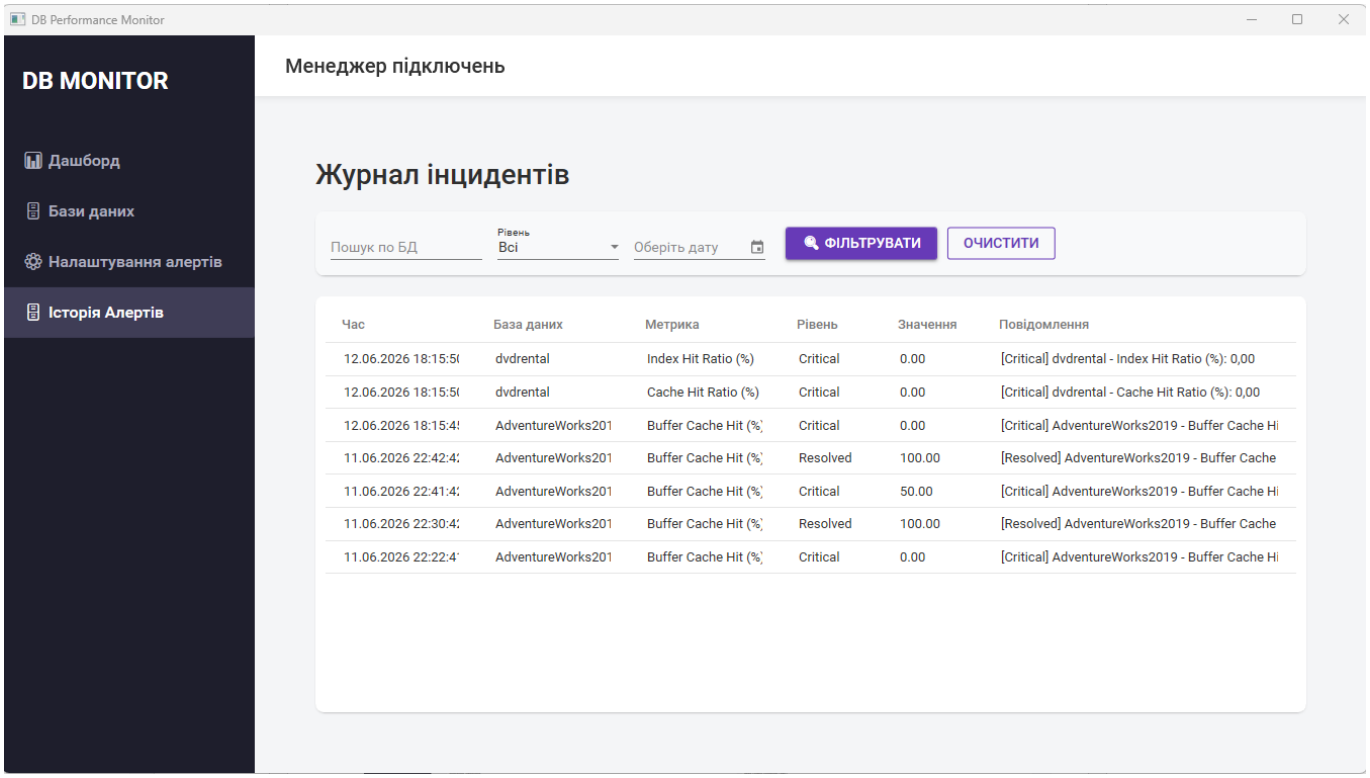
1. Перейдіть до розділу «Історія Алертів».
2. Для швидкого пошуку першопричини інциденту скористайтеся панеллю фільтрації у верхній частині вікна:

– Пошук за точною назвою сервера або бази даних.

– Фільтрація за станом: Warning (Попередження), Critical (Аварія) або Resolved (Проблему усунено).

– Фільтрація за календарною датою виникнення аномалії.

3. Після встановлення параметрів натисніть кнопку «ФІЛЬТРУВАТИ». Таблиця нижче відобразить відповідні записи із зазначенням точного часу, назви метрики та зафіксованого відхилення.


DB Performance Monitor
DB MONITOR
Дашборд
Бази даних
Налаштування алертів
Історія Алертів
Менеджер підключень
Журнал інцидентів
Пошук по БД
Рівень
Всі
Оберіть дату
ФІЛЬТРУВАТИ
ОЧИСТИТИ
Час
База даних
Метрика
Рівень
Значення
Повідомлення
12.06.2026 18:15:5(dvdrental Index Hit Ratio (%) Critical 0.00 [Critical] dvdrental - Index Hit Ratio (%): 0,00
12.06.2026 18:15:5(dvdrental Cache Hit Ratio (%) Critical 0.00 [Critical] dvdrental - Cache Hit Ratio (%): 0,00
12.06.2026 18:15:4! AdventureWorks201 Buffer Cache Hit (% Critical 0.00 [Critical] AdventureWorks2019 - Buffer Cache Hi
11.06.2026 22:42:4; AdventureWorks201 Buffer Cache Hit (% Resolved 100.00 [Resolved] AdventureWorks2019 - Buffer Cache
11.06.2026 22:41:4; AdventureWorks201 Buffer Cache Hit (% Critical 50.00 [Critical] AdventureWorks2019 - Buffer Cache Hi
11.06.2026 22:30:4; AdventureWorks201 Buffer Cache Hit (% Resolved 100.00 [Resolved] AdventureWorks2019 - Buffer Cache
11.06.2026 22:22:4' AdventureWorks201 Buffer Cache Hit (% Critical 0.00 [Critical] AdventureWorks2019 - Buffer Cache Hi

ДОДАТОК Г. АПРРОБАЦІЯ РЕЗУЛЬТАТІВ РОБОТИ

СІНТЕХ 2026

СЕРТИФІКАТ

засвідчує, що

Сорокопуд Андрій Павлович

взяв участь у

І Міжнародній науково-практичній конференції:
Сучасні інформаційні технології: від теорії до практики

загальним обсягом 15 годин (0,5 кредитів ECTS),
яка проводилася 22-23 травня 2026 року
в Луцькому національному технічному університеті

Ректор ЛНТУ Ірина ВАХОВИЧ

mintech-conf.lntu.edu.ua
/cert/2026-263

Луцький національний технічний університет
вул. Львівська, 75 м. Луцьк
Волинська обл. 43018

УДК 004
С 91

Міністерство освіти і науки України	Ministry of Education and Science of Ukraine
Волинська обласна рада	Volyn Regional Council
Луцька міська рада	Lutsk City Council
Луцький національний технічний університет	Lutsk National Technical University (Ukraine)
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»	National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute" (Ukraine)
Національний університет «Львівська політехніка»	Lviv Polytechnic National University (Ukraine)
Військовий інститут телекомунікацій та інформатизації імені Героїв Крут (м. Київ)	Kruty Heroes Military Institute of Telecommunications and Information Technology (Ukraine)
Рівненський державний гуманітарний університет	Rivne State University of the Humanities (Ukraine)
Державний торговельно-економічний університет (м. Київ)	State University of Trade and Economics (Ukraine)
Чернівецький національний університет імені Юрія Федьковича	Yuriy Fedkovych Chernivtsi National University (Ukraine)
Люблінська політехніка (Польща)	Lublin University of Technology (Poland)
Гронінгенський університет (Нідерланди)	University of Groningen (the Netherlands)
Кавказький університет (Грузія)	Caucasus University (Georgia)
Політехнічний університет Браганси (Португалія)	Polytechnic Institute of Bragança (Portugal)
Університет Сямень (Малайзія)	Xiamen University (Malaysia)
Мінюський університет (Португалія)	University of Minho (Portugal)

Рекомендовано до опублікування науково-технічною радою
Луцького національного технічного університету
(протокол № 10 від 27.05.2026 р.)

С 91 Тези доповідей І Міжнародної науково-практичної конференції «Сучасні інформаційні технології: від теорії до практики (СІнТех-2026)» (22-23 травня 2026 року). Луцьк: ЛНТУ, 2026. 1086 с.

Book of abstracts of the 1st International Scientific and Practical Conference "Modern Information Technologies: From Theory to Practice (MInTech-2026)" (22–23 May 2026). Lutsk: Lutsk National Technical University, 2026. 1086 p.

Матеріали І Міжнародної науково-практичної конференції «Сучасні інформаційні технології: від теорії до практики». Видання містить аналіз та результати досліджень, що стосуються актуальних питань інформаційних технологій в галузях освіти, науки та промисловості. Тези доповідей надано в авторській редакції. За фактичний матеріал і його інтерпретацію відповідають автори.

Proceedings of the 1st International Scientific and Practical Conference "Modern Information Technologies: From Theory to Practice". The publication contains analyses and research results addressing current issues in information technologies in the fields of education, science, and industry. The abstracts are published as submitted by the authors. The authors are responsible for the factual content and interpretation of the submitted materials.

Відповідальний за випуск: к.т.н., доцент Ліщина Н.М.

КРИТИЧНИЙ АНАЛІЗ МЕТРИК ЕФЕКТИВНОСТІ НАКОПИЧЕННЯ ТА ОБРОБКИ ДАНИХ ЗАСОБАМИ СУЧАСНИХ СИСТЕМ УПРАВЛІННЯ БАЗАМИ ДАНИХ

Сорокопуд Андрій Павлович
ТОВ «ТЕХНІЧНИЙ УНІВЕРСИТЕТ «МЕТІНВЕСТ ПОЛІТЕХНІКА», Запоріжжя, здобувач гр. КН-23-1б, Andrii.Sorokopud@mipolytech.education
Сагайда Павло Іванович
ТОВ «ТЕХНІЧНИЙ УНІВЕРСИТЕТ «МЕТІНВЕСТ ПОЛІТЕХНІКА», Запоріжжя, д.т.н., завідувач кафедри інформаційних технологій та аналітики даних, pavlo.sahaida@mipolytech.education

CRITICAL ANALYSIS OF PERFORMANCE METRICS FOR DATA ACCUMULATION AND PROCESSING USING MODERN DATABASE MANAGEMENT SYSTEMS

Andrii Sorokopud
«TECHNICAL UNIVERSITY «METINVEST POLYTECHNIC» LLC, applicant gr. CS-23-1b, Andrii.Sorokopud@mipolytech.education
Pavlo Sahaida
«TECHNICAL UNIVERSITY «METINVEST POLYTECHNIC» LLC, Zaporizhzhia, DSc (Engineering), Head of the Department of Information Technology and Data Analytics, pavlo.sahaida@mipolytech.education

The methodology for evaluating the effectiveness of modern DBMS in the context of digital transformation has been studied. Authors analyze key metrics, which are divided into three groups: data accumulation parameters (write throughput, latency, compression), query processing indicators (read throughput, query latency, IOPS), and comprehensive system monitoring metrics (cache hit ratio, CPU/RAM usage, locking). It is substantiated that systematic monitoring of these metrics allows for timely identification of "bottlenecks," optimization of system configurations, and planning of their scaling to ensure stable work with Big Data.