ISSN 2522-9818 (print)
ISSN 2524-2296 (online) *Innovative technologies and scientific solutions for industries. 2023. No. 2 (24)*

УДК 004.04 DOI: https://doi.org/10.30837/ITSSI.2023.24.118

О. Мазурова, М. Андрущенко, М. Широкопетлєва

ДОСЛІДЖЕННЯ МЕТОДІВ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ *COSMOS DB API* НА ПЛАТФОРМІ *.NET*

Значна кількість сучасних розробників використовують платформу *.NET* для створення програм, що працюють із базами даних. *Cosmos DB* стає все більш популярним вибором як *NoSQL*-сховище для баз даних. *Cosmos DB* – гнучка й масштабована система, і правильний вибір відповідного *API* в програмній реалізації може значно вплинути на продуктивність самих програм. *Cosmos DB* надає різні *API* для роботи з усіма типами баз даних. Кожен із цих *API* може бути використаний за допомогою різних методів програмної реалізації. **Предметом** дослідження є програмні реалізації на платформі *.NET* під різні *Cosmos DB API*. Під час обрання найбільш підхожого *Cosmos DB API* на платформі *.NET* розробникам може допомогти не тільки документація, але й результати експериментальних досліджень *API*, що дасть змогу покращити якість коду й продуктивність самих систем. **Мета роботи** – підвищити ефективність програмних розробок на платформі *.NET*, що використовують *Cosmos DB API*, шляхом створення рекомендацій щодо обрання методів програмної реалізації *API* на основі результатів експериментального дослідження. **Завдання статті:** дослідити та порівняти методи програмної реалізації *Cosmos DB API* шляхом вивчення продуктивності різних типів запитів на цих програмних рішеннях; проаналізувати здобуті результати та розробити рекомендації з використання методів. **Методи:** багатокритеріальний аналіз *Cosmos DB API*, логічне моделювання даних, дослідження. **Результати:** розроблено програмні рішення на основі використання *CosmosClient, Entity Framework Core* для *Cosmos DB API for NoSQL* та на основі *MongoClient* для *Cosmos DB API for MongoDB*; проведено серію експериментів і вимірювань показників продуктивності для кожного з програмних рішень; проаналізовано здобуті результати та запропоновано рекомендації з використання розглянутих методів програмної реалізацій *Cosmos DB API* на платформі *.NET*. **Висновки.** Загалом вибір програмного підходу залежить від конкретного завдання, але дослідження показали, що *Cosmos DB API for NoSQL* із застосуванням *CosmosClient* – це найкращий вибір для незначних проєктів, а з використанням *Entity Framework Core Cosmos* підходить для проєктів з більшими обсягами інформації та складними запитами. Якщо в проєкті застосовується *MongoDB,* то відповідне рішення з використанням *MongoClient* є кращим варіантом, ніж *Cosmos DB API for NoSQL*.

Ключові слова: база даних; *Cosmos DB API; MongoDB; .NET; NoSQL*.

Вступ

Сучасний підхід до розроблення програмних систем спрямований на те, щоб спростити програму, зробити код простішим і дешевшим для підтримки в майбутньому. Саме тому наразі дуже популярні хмарні розподілені системи, що складаються з багатьох маленьких програм, які відповідають лише за одну функцію. Такі програми називаються сервісами, а архітектура таких систем називається мікросервісною. Така архітектура легка в підтримці та розробленні, за кожен сервіс найчастіше відповідає окрема команда. Це досягається розподіленням завдань, що має виконувати окремий сервіс. Саме тому більшість нових систем розробляється саме з такою архітектурою.

Якщо говоримо про наявні програмні рішення, а саме про системи з монолітною архітектурою, частина з них може виконувати свої функції і сьогодні, частина ж переписується на розподілену архітектуру. Так само і з базами даних. Класичний підхід використовувати реляційні бази даних (БД) нині не є 100-відсотковою панацеєю. Замість них застосовують нереляційні бази даних *NoSQL*, що здобули популярність, оскільки вони зберігають інформацію в простих зрозумілих формах, які легше змінювати за потреби, ніж типи моделей даних, що застосовуються в реляційних БД. Крім того, бази даних *NoSQL* часто дають змогу розробникам безпосередньо змінювати структуру інформації. Також нереляційні сховища даних надають дуже потужний функціонал масштабування та розподілення по регіонах. Саме тому актуальне дослідження *Azure Cosmos DB,* що є популярним рішенням для управління розподіленими даними в хмарі. *Azure Cosmos DB* забезпечує глобальну масштабованість, автоматичне реплікування та георозподіл даних, а також підтримує безліч *API*, зокрема *API for NoSQL, for MongoDB, for Apache Cassandra, for Table, for Apache Gremlin, for PostgreSQL*. Це робить її зручним вибором для багатьох програм, що потребують швидкого доступу до даних із різних

ISSN 2522-9818 (print)
Сучасний стан наукових досліджень та технологій в промисловості. 2023. № 2 (24) ISSN 2524-2296 (online)

точок світу та підтримки різних типів запитів. Вирішено дослідити та порівняти реалізацію різних програмних інтерфейсів із використанням популярної платформи *.NET*.

Аналіз проблеми й наявних методов

Інформаційні системи, що працюють на основі баз даних, створюються та використовуються майже в усіх галузях сучасного життя [1, 2]. Реляційні БД продовжують тримати лідерські позиції під час розроблення систем, де необхідна узгодженість даних на високому рівні, або, як прийнято казати в спільноті розробників, підтримка *ACID* властивостей транзакцій [3]. Водночас усе більше нових секторів бізнесу погоджуються на відкладену узгодженість інформації за умови високої масштабованості відповідних застосунків. Саме для створення таких систем розробники активно використовують бази даних *NoSQL* [4], що забезпечують високу горизонтальну масштабованість систем і, крім того, зберігають інформацію в простих формах та дають змогу розробникам змінювати структури такої інформації [5]. Так, БД документів не мають установленої структури даних для початку, тому новий тип документа можна зберігати так само легко, як і той, що зберігається наразі [6].

Також *NoSQL* є популярною основою для систем управління розподіленими даними в хмарі. Такі платформи, як *Azure Cosmos DB, Amazon Dinamo DB* тощо працюють здебільшого на основі *NoSQL* БД та забезпечують глобальну масштабованість, автоматичне реплікування та георозподіл даних. Це робить їх зручним вибором для створення інформаційних систем у багатьох галузях, що потребують швидкого доступу до даних із різних точок світу.

Електронна комерція – це досить поширена прикладна галузь для створення систем, до якої належить безліч різних операцій [7]. Це і складський облік, прийом поставок, списання товару, облік номенклатури, оновлення цін товарів, відстеження кількості товарів у різних складських приміщеннях, проведення акцій. Важливою є система збереження даних користувачів, історії замовлень, побажань і транзакцій з оплатами замовлень. *Azure Cosmos DB* нерідко використовується для розроблення програмних систем, пов’язаних з електронною комерцією.

Azure Cosmos DB часто застосовується саме в галузі роздрібної торгівлі для зберігання даних каталогу та пошуку подій у конвеєрах обробки замовлень. Також ця система широко застосовується у власних платформах електронної комерції *Microsoft*, наприклад, у *Windows Store* і *XBox Live*. *Azure Cosmos DB* підтримує гнучкі схеми та ієрархічні дані, тому є ефективним засобом для зберігання інформації каталогу продуктів.

Функції, надані *Azure Cosmos DB*, передбачають [8]: масштабування, високу доступність, геореплікацію, кілька місць для запису, автоматичне та прозоре керування сегментами, прозору реплікацію між операційними й аналітичними сховищами.

Саме тому *Azure Cosmos DB* часто використовується як джерело подій для керування подіями архітектур за допомогою функції каналу змін. Ця функція надає подальшим мікросервісам змогу надійно та поступово зчитувати вставки та оновлення (наприклад, події замовлення), внесені до БД *Azure Cosmos*. Канал змін можна застосовувати для забезпечення постійного сховища подій, як брокера повідомлень для подій, що змінюють стан, і керувати процесом оброблення між багатьма мікросервісами.

Azure Cosmos DB пропонує кілька *API* бази даних, зокрема *NoSQL, MongoDB, PostgreSQL Cassandra, Gremlin і Table.* Завдяки *API* можна працювати з інформацією, використовуючи моделі документів, ключ-значення, графіки та сімейство стовпців. Зазначені *API* дають змогу програмам обробляти *Azure Cosmos DB* так, ніби це різні технології баз даних, без накладних витрат на керування та підходи до масштабування. *Azure Cosmos DB* допомагає застосовувати екосистеми, інструменти та навички, що вже має розробник, для роботи з інформацією та виконання запитів за допомогою різноманітних *API*.

API Azure Cosmos DB для *NoSQL* зберігає інформацію у форматі документа. Він пропонує найкращий наскрізний досвід, оскільки розробник має повний контроль над інтерфейсом, сервісом і клієнтськими бібліотеками *SDK*. Будь-яка нова функція, що розгортається в *Azure Cosmos DB*, спочатку доступна в *API* для облікових записів *NoSQL*. Ці облікові записи забезпечують підтримку для запиту елементів за допомогою синтаксису мови структурованих запитів (*SQL*) та одного з найвідоміших форматів обміну даними (*JSON*), який використовується для збереження та передачі структурованих інформаційних об’єктів між різними застосунками.

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2023. No. 2 (24)

Фахівці рекомендують використовувати *API for NoSQL* тоді, коли необхідно перейти з інших баз даних, таких як *Oracle, DynamoDB, HBase* тощо, і якщо потрібно застосовувати модернізовані технології для створення своїх програм. *API* для *NoSQL* підтримує аналітику й забезпечує ізоляцію продуктивності між операційними та аналітичними навантаженнями.

API Azure Cosmos DB для *MongoDB* зберігає інформацію в структурі документа з допомогою формату *BSON*. Він сумісний із дротовим протоколом *MongoDB*, однак не застосовує жодного рідного коду, пов'язаного з *MongoDB*. Розробники рекомендують використовувати наявні програми *MongoDB* з *API* для *MongoDB,* змінивши рядок підключення та перемістивши будь-які дані за допомогою власних інструментів *MongoDB*, зокрема *mongodump* і *mongorestore*, або інструменту міграції бази даних *Azure*. Інструменти, а саме оболонка *MongoDB, MongoDB Compass і Robo3T*, можуть виконувати запити та працювати з інформацією так само, як і з нативною *MongoDB*. Вважається, що *API* для *MongoDB* – чудовий вибір, якщо потрібно застосовувати екосистему *MongoDB* і відповідні навички без шкоди для використання функцій *Azure Cosmos DB*.

Azure Cosmos DB для *PostgreSQL* – це керована служба для запуску *PostgreSQL* у будь-якому масштабі з суперпотужністю розподілених таблиць із відкритим кодом *Citus*. Він зберігає інформацію або на одному вузлі, або в конфігурації з кількома вузлами. *Azure Cosmos DB* для *PostgreSQL* побудовано на основі власного *PostgreSQL*, а не на форку *PostgreSQL*, і дає змогу обрати будь-яку основну версію бази даних, що підтримується спільнотою *PostgreSQL*. Вважається, що він ідеально підходить для запуску одновузлової бази даних із багатим індексуванням, геопросторовими можливостями та підтримкою *JSONB*. Якщо в майбутньому знадобиться більша продуктивність, рекомендується додавати вузли до кластера без простою системи.

API Azure Cosmos DB для *Cassandra* зберігає дані в схемі, орієнтованій на стовпці. *Apache Cassandra* пропонує високорозподілений, горизонтально масштабований підхід до зберігання великих обсягів інформації, одночасно пропонуючи гнучкий підхід до схеми, орієнтованої на стовпці. *API* для *Cassandra* в *Azure Cosmos DB* відповідає цій філософії наближення до розподілених баз даних *NoSQL*. *API* для *Cassandra* є дротовим протоколом, сумісним із рідним *Apache Cassandra*. Фахівці радять розглянути *API* для *Cassandra*, якщо хочете отримати переваги від гнучкості та повністю керованого характеру *Azure Cosmos DB*. Рекомендується використовувати клієнтські драйвери *Apache Cassandra* для підключення до *API* для *Cassandra*, що дає змогу взаємодіяти з інформацією за допомогою мови запитів *Cassandra* (*CQL*) і таких інструментів, як оболонка *CQL*, клієнтські драйвери *Cassandra*. *API* для *Cassandra* наразі підтримує лише сценарії *OLTP*. Розробники також рекомендують використовувати унікальні функції *Azure Cosmos DB*, такі як канал змін.

API Azure Cosmos DB для *Gremlin* дає змогу користувачам створювати запити на графіки та зберігає дані як ребра та вершини. Фахівці пропонують застосовувати *API* для *Gremlin* для сценаріїв:

– залучення динамічних даних;

– залучення даних зі складними зв'язками;

– залучення даних, надто складних для моделювання, за допомогою реляційних баз даних;

– якщо ви хочете використовувати наявну екосистему та навички *Gremlin*.

API для *Gremlin* поєднує потужність алгоритмів графічної бази даних із високомасштабованою керованою інфраструктурою. *API* для *Gremlin* наразі підтримує лише сценарії *OLTP* і ґрунтується на структурі графових обчислень *Apache TinkerPop*. *API* для *Gremlin* використовує ту саму мову запитів *Graph* для прийому та запиту інформації. Він застосовує стратегію розділу *Azure Cosmos DB* для виконання операцій читання / запису з механізму бази даних *Graph*. *API* для *Gremlin* також працює з *Apache Spark* і *GraphFrames* для сценаріїв складних аналітичних графіків.

API Azure Cosmos DB for Table зберігає інформацію у форматі "ключ / значення". Вважається, що якщо використовувати сховище *Azure Table*, то можна спостерігати певні обмеження щодо затримки, масштабування, пропускної здатності, глобального розподілу, керування індексами, низької продуктивності запитів. *API Azure Cosmos DB for Table* долає ці обмеження, тому рекомендується перенести свою програму, якщо хочете скористатися перевагами *Azure Cosmos DB*. *API* для таблиць підтримує лише сценарії *OLTP*. Програми, написані для сховища таблиць *Azure*, можна перенести на *API* для таблиць із незначними змінами коду та скористатися перевагами преміум-можливостей.

Сучасний стан наукових досліджень та технологій в промисловості. 2023. № 2 (24)
ISSN 2522-9818 (print)
ISSN 2524-2296 (online)

На жаль, рекомендації з документацій того чи іншого *API* не завжди покривають важливі для розробників питання, від яких залежатиме вибір програмного підходу. Тому результати досліджень щодо порівняння програмних інтерфейсів і сформовані рекомендації будуть корисні для розробників.

Метою цієї роботи є формулювання більш ефективних рекомендацій щодо виботу методів програмної реалізації *Cosmos DB API* на платформі *.NET* на базі аналізу результатів досліджень.

Для досягнення поставленої мети необхідно вирішити такі завдання:

– на основі аналізу наявних *Cosmos DB API* та багатокритеріального прийняття рішення обрати найкращі для подальшого дослідження *API*;

– проаналізувати обрану предметну сферу, а саме електронну комерцію, і спроєктувати відповідну логічну модель бази даних;

– розробити програмні рішення для кожного з досліджуваних *Cosmos DB API*;

– налаштувати середовище *Azure* для роботи з *Cosmos DB*, провести експерименти, здобуті результати проаналізувати та сформулювати рекомендації щодо використання кожного з досліджуваних підходів програмної реалізації.

Матеріали й методи

На основі проведеного аналізу наявних *Cosmos DB API* сформульовано багатокритеріальне завдання прийняття рішень із вибору найкращих для дослідження *API*. Множина альтернатив для цього завдання складається з таких варіантів: *API for NoSQL, API for MongoDB, API for PostgreSQL, API for Apache Cassandra, API for Apache Gremlin та API for Table* [9].

Для обґрунтованого вибору сформовано множину критеріїв, що дає змогу з різних поглядів оцінити варіанти *API* та обрати найкращі для дослідження їх застосування на платформі *.NET*, а саме:

– ступінь підтримки платформи *.NET* – важливо, щоб досліджуваний програмний інтерфейс мав підтримку цієї платформи;

– частота оновлення *API* – цей критерій показує, наскільки швидко нові функції *Cosmos DB* будуть доступні на тому чи іншому *API*;

– доступність функцій БД через код – цей критерій показує наявність функціоналу зі створювання бази даних та контейнерів із коду й перевірки, а також чи існує контейнер та інші налаштування;

– формат зберігання даних – за цим критерієм можна порівняти, у якому форматі зберігається інформація.

Сформовано шкалу оцінки кожного з обраних критеріїв. Під час цього було відкинуто критерій підтримки платформи *.NET*, бо кожен із розглянутих *API* підтримує цю технологію. Щоб оцінити корисність для дослідження відповідного *API*, використана лінійна адитивна згортка з ваговими коефіцієнтами. Нормовані значення за критеріями та результати обчислень за згортковою моделлю подано в табл. 1.

Під час визначення вагових коефіцієнтів найбільш впливовим є критерій доступності функцій БД через код (коефіцієнт 0.6). Це найбільш важливий параметр, оскільки визначає, яку кількість можливостей того чи іншого програмного інтерфейсу можна реалізувати саме через код. Критерій частоти оновлення *API* має коефіцієнт 0.2. Розробникам важливо використовувати нові функції обраної БД, тому переваги над іншими матиме *API*, у якому швидше доступні нові функції платформи. Критерій формату зберігання даних є достатньо важливим для розробників, тому йому також призначено коефіцієнт 0.2.

Таблиця 1. *Вхідні нормовані дані та результати розрахунку корисності API*

Лінійна адитивна згортка з ваговими коефіцієнтами				
API /критерії вибору	**Частоста оновлення API**	**Доступність функцій БД через код**	**Формат зберігання даних**	**Z***
API for NoSQL	1	1	1	0.36901789
API for MongoDB	0.666	0.333	1	0.179174099
API for PostgreSQL	0.666	0.333	0.75	0.164888385
API for Apache Cassandra	0.666	0.333	0.5	0.15060267
API for Apache Gremlin	0.666	0.333	0.25	0.136316956
API for Table	0.333	0.666	1	0.246674773
Вагові коефіцієнти	0.2	0.6	0.2	

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2023. No. 2 (24)

Як показав багатокритеріальний вибір, *API for NoSQL* має найвищий показник корисності, а саме 0.36901789, та найбільше підходить для подальшого дослідження. Крім того, за цим *API* можемо реалізувати роботу з БД двома програмними способами: *Entity Framework Core Cosmos* та *CosmosClient*. Наступна *API for Table* є доволі особливим інтерфейсом, що найбільше підходить не для вирішення загальних завдань із БД, а, наприклад, для переносу архітектури в *Cosmos DB* з *Azure Table*. Тому прийнято рішення не розглядати його під час дослідження. Для порівняння обрано *API for MongoDB* з наступною корисністю 0.179174099. *MongoDB* – це БД, що активно використовується для створення високодоступних і масштабованих інтернет-застосунків. Завдяки гнучкому підходу до структури він популярний серед команд розробників. *API for MongoDB* – це гарна альтернатива для порівняння з *API for NoSQL*.

У процесі порівняння та вибору *Cosmos DB API* для дослідження було обрано *API for NoSQL* та *API for MongoDB*. На платформі *.NET* [10] програмно реалізувати *API for NoSQL* можна двома способами: використовуючи *Entity Framework Core Cosmos* та *Cosmos Client* системний клас. Тому ці дві реалізації досліджуватилися й порівнюватилися як окремі.

Предметною сферою для вивчення була обрана електронна комерція (*e-commerce*), а саме інтернет-магазин одягу. Оскількі дослідження передбачає експерименти із запитами та вимірювання відповідних показників їх продуктивності, не було потреби створювати реалістично модель БД [11]. За основу взято спрощену концептуальну модель для сфери діяльності інтернет-магазину одягу, що містить такі сутності, як *Product* (товар), *Order* (замовлення), *Payment* (платіж) [12].

У сутності *Product* зберігається інформація про всі товари, кожен має власну назву, опис і назву виробника, а також унікальний ідентифікатор – *id* товару. У сутності *Order* є інформація про замовлення, кожне з яких має унікальний ідентифікатор *id*, дату замовлення, список товарів, доданих до замовлення, та загальну вартість замовлення. У сутності *OrderItem* зберігаються відомості про позиції товарів кожного замовлення та міститься, крім атрибутів для моделювання зв'язку, кількість товару в замовленні. У сутності *Payment* є інформація про оплату замовлення. Ця сутність має поле для зберігання вартості замовлення та дати оплати.

На основі побудованої *ER*-діаграми розроблено логічну модель БД для СУБД *MongoDB* (рис. 1).

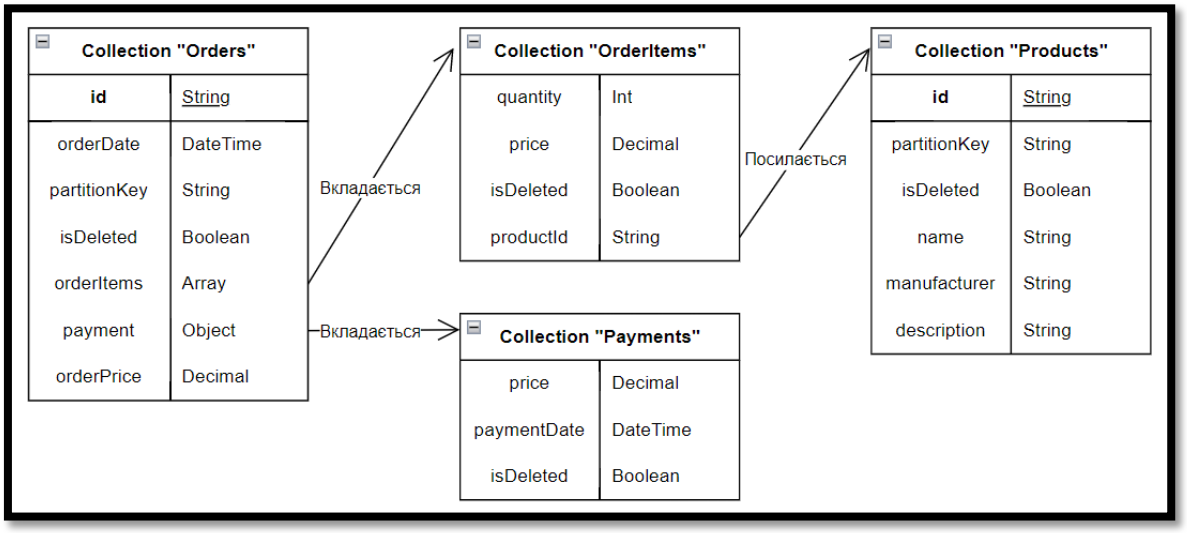

Рис. 1. Логічна модель БД для *MongoDB*

У *MongoDB* дані зберігаються в колекціях (*collections*), де є документи (*documents*) – об'єкти у форматі *BSON*, що містять ключі та значення [4]. Ключі відповідають полям документа, а значення – їх значенням. *BSON* (*Binary JSON*) – це бінарний формат серіалізації *JSON*, що забезпечує більш компактне зберігання інформації та дає змогу ефективніше використовувати ресурси [13].

Для моделювання логічної структури даних у *MongoDB* застосовано додаткові поля, необхідні для програмної реалізації роботи з колекціями *Cosmos DB*: *PartitionKey* та *isDeleted*. *PartitionKey* необхідний

Сучасний стан наукових досліджень та технологій в промисловості. 2023. № 2 (24)
ISSN 2522-9818 (print)
ISSN 2524-2296 (online)

Cosmos DB для оптимізації запитів, а властивість *isDeleted* – для реалізації м'якого вилучення [14].

Для дослідження продуктивності реалізації обрано такі важливі для будь-якої бази даних параметри, як швидкість виконання *CRUD*-операцій, що використовуються найчастіше. Отже, для дослідження було розроблено такі запити до бази даних:

– *INSERT*-запит на створення нового замовлення типу «INSERT INTO Orders ("id", "isDeleted", "orderDate", "orderPrice", "partitionKey", "orderItems", "payment") VALUES ("b0842587-42e7-4b5f-ab17-c09ca1c9629f", "false", "2023-04-27T20:16:34.4468614Z", "100", "100", [{ "price": 50, "productId": "d489b6b9-5157-408f-a249-fcfc33493c08", "quantity": 1 }, { "price": 25, "productId": "f88a4167-b622-4b7c-9770-a39d4aa2de85", "quantity": 2 }], { "paymentDate": "2023-04-27T20:16:12.0488881Z", "price": 100 });»; запити такого типу дають змогу дослідити швидкість створення нових замовлень;

– *SELECT*-запит № 1 на отримання замовлення за певним *PartitionKey* типу «SELECT * FROM Orders WHERE Orders.partitionKey = "100"» });»; запити такого типу дають змогу дослідити швидкість повернення всіх замовлень, що мають спільний ключ розділу *Cosmos DB*;

– SELECT-запит № 2 на отримання замовлення за певним *id* типу «SELECT * FROM Orders WHERE Orders.id = "b0842587-42e7-4b5f-ab17-c09ca1c9629f"» });»; запити такого типу дають змогу дослідити швидкість отримання замовлення за *id* без оптимізації *PartitionKey*;

– UPDATE-запит на оновлення полів сутності замовлення: «UPDATE Orders SET "orderDate" = "2023-04-28T11:15:49.9163777Z" WHERE Orders.id == "b0842587-42e7-4b5f-ab17-c09ca1c9629f";»; });»; запити такого типу дають змогу дослідити швидкість оновлення сутності замовлення;

– DELETE-запит на вилучення сутності замовлення з бази даних типу «DELETE FROM Orders WHERE Orders.id == "b0842587-42e7-4b5f-ab17-c09ca1c9629f";» });»; запити такого типу дають змогу дослідити швидкість вилучення запису з БД;

– SOFT DELETE-запит на безпечне вилучення замовлення з бази даних типу «UPDATE Orders SET "isDeleted" = "true" WHERE Orders.id == "b0842587-42e7-4b5f-ab17-c09ca1c9629f";» });»; запити такого типу дають змогу дослідити швидкість м'якого вилучення запису в БД та порівняти її з DELETE-запитом.

На основі аналізу підходів до оцінювання продуктивності БД і програмних систем загалом [15] обрано такі показники для порівняння перелічених операцій:

– швидкість виконання запиту в мілісекундах;

– кількість байтів, витрачених сервером на таку операцію;

– кількість витрачених ресурсів RU на виконання операції в БД.

Database throughput RU (Request Units) – це одиниця виміру продуктивності, що використовується в *Azure Cosmos DB*. Кожна дія, що виконується в *Cosmos DB*, потребує певної кількості RU [14].

Окремо було вирішено дослідити, чи можливо за умови застосування того чи іншого програмного підходу створювати базу даних або контейнер, а також перевіряти, чи база даних або контейнер уже створені. Показником для цього порівняння буде прапорець *TRUE* чи *FALSE*: можливо чи ні створити з коду базу даних або контейнер.

Також досліджено складність реалізації для кожного з підходів способом аналізу питання: наскільки багато коду потрібно написати для його роботи. Показником для цього порівняння є кількість рядків коду для кожної з реалізацій.

Для розроблення коду реалізації програмних інтерфейсів *Cosmos DB* використовувся підхід об'єктно орієнтованого програмування [15]. Отже, для кожної колекції логічної моделі БД було спроєктовано відповідні класи й описано їх поведінку (рис. 2). Колекції *Orders* та *Products* мають низку схожих властивостей, а саме унікальний ключ кожного запису (*string Id*) та булеве поле для відображення, чи вилучений запис (*bool IsDeleted*). Ці властивості було винесено в базовий абстрактний клас (*class EntityBase*), що дало змогу далі реалізувати підхід безпечного (м'якого) вилучення (*soft delete*). М'яке вилучення може покращити продуктивність, а також відновлювати вилучену інформацію.

Ключ розділу (*string PartitionKey*) необхідний для оптимізації запитів. *Cosmos DB* дає змогу групувати набір елементів або даних у колекції за подібною властивістю, визначеною ключем розділу. Ключі розділу є основним елементом для ефективного розподілу даних у різних логічних і фізичних наборах, щоб запити, які виконуються до БД, завершувалися якомога швидше. Важливо обрати ключ розділу на етапі проєктування програм, оскільки не можна змінити ключ розділу після створення контейнера. Зробивши властивість *PartitionKey* абстрактною в базовому класі, маємо змогу перевизначити її по-різному в класах-спадкоємцях.

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2023. No. 2 (24)

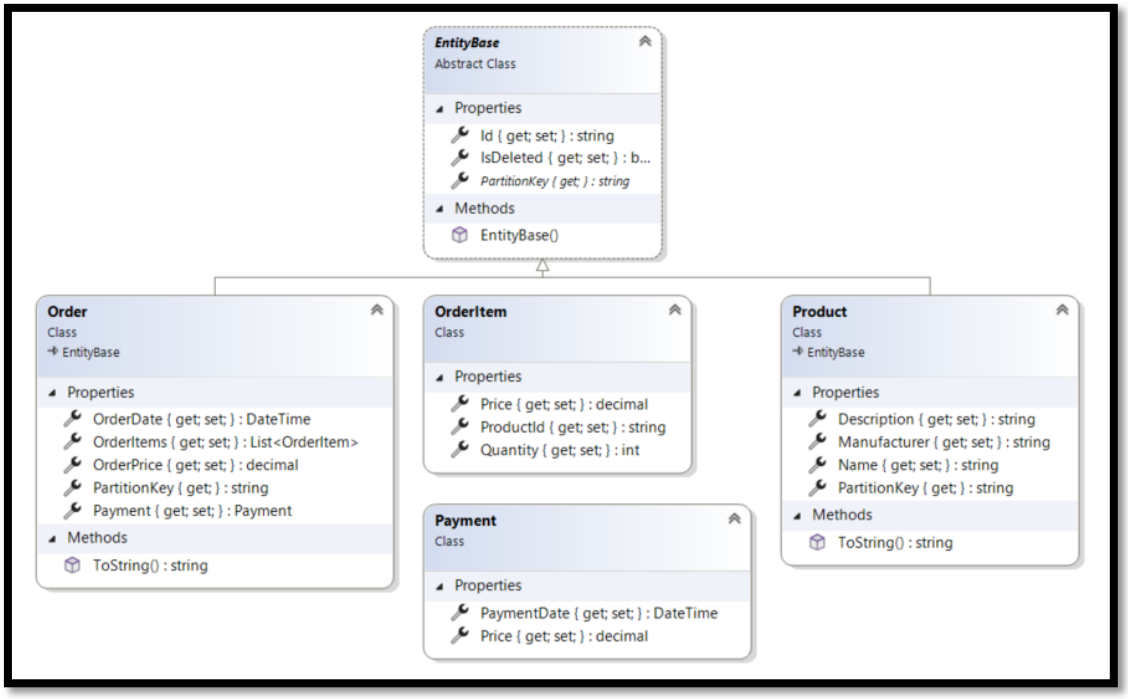

Рис. 2. Діаграма базових класів для обраної предметної сфери

Для того, щоб дослідження різних програмних реалізацій було достовірним, розроблено основні класи, які виконують параметри підключення та налаштування БД, що мають однакову структуру, а саме класи *CosmosDbSettings*, *CosmosDbExtensions* та *CosmosDbInitializer* (рис. 3).

За допомогою класу *CosmosDbSettings* реалізовано можливість конфігурувати програму, задавати такі параметри, як рядок підключення до БД (*ConnectionString*), назва БД (*DatabaseName*), список контейнерів (*ContainerNames*) та окремо назви для кожного з необхідних контейнерів, у нашому випадку для контейнерів *Orders* та *Products* (*OrdersContainerName, ProductsContainerName*).

Клас *CosmosDbExtensions* містить методи розширення для конфігурації системних класів для роботи з *CosmosDb*. Метод *ConfigureCosmosClient*() задає конфігурацію для підключення до БД, використовуючи параметри з класу *CosmosDbSettings*, та налаштовує параметри серіалізації об'єктів, після чого реєструє клас, що відповідає за роботу з БД, у контейнері інверсії залежностей.

Клас *CosmosDbInitializer* необхідний для того, щоб перевірити, чи створена БД та всі необхідні контейнери під час підключення до *Azure*, та створити їх, якщо необхідно.

Далі ці класи викликаються в основному класі *Program*, спочатку метод конфігурації, потім метод ініціалізації БД. Розроблений клас *Program.cs* конфігурує та запускає вебзастосунок. Він є однаковим для кожної з програмних реалізацій завдяки тому, що було інкапсульовано логіку підключення до БД в описаних вище методах розширення. Ці методи лише мають різну назву для кожного з проєктів:

– *ConfigureCosmosClient*() та *InitializeCosmosClient*() для *Cosmos Client*;

– *ConfigureCosmosEf*() та *InitializeCosmosEf*() для *Entity Framework Core Cosmos* [16];

– *ConfigureCosmosMongoDb*() та *InitializeCosmosMongoDb*() для *MongoDB*.

Для того, щоб кожна з програм мала однаковий інтерфейс, у процесі проведення експериментів було розроблено два контролери для роботи з кожною колекцією *OrdersController* та *ProductsController* (рис. 4). Ці контролери є реалізацією *RESTful API* – архітектурного стилю інтерфейсу застосунку (*API*), що використовує запити *HTTP* для доступу до інформації та її використання. Цю інформацію можна застосовувати для типів даних *GET*, *PUT*, *POST* і *DELETE*, що стосуються читання, оновлення, створення та вилучення операцій щодо ресурсів. Кожен із них має методи для виконання описаних запитів до БД: *Create*(), *GetByPartitionKey*(), *GetById*(), *Update*(), *Delete*(), *SoftDelete*().

Сучасний стан наукових досліджень та технологій в промисловості. 2023. № 2 (24)
ISSN 2522-9818 (print)
ISSN 2524-2296 (online)

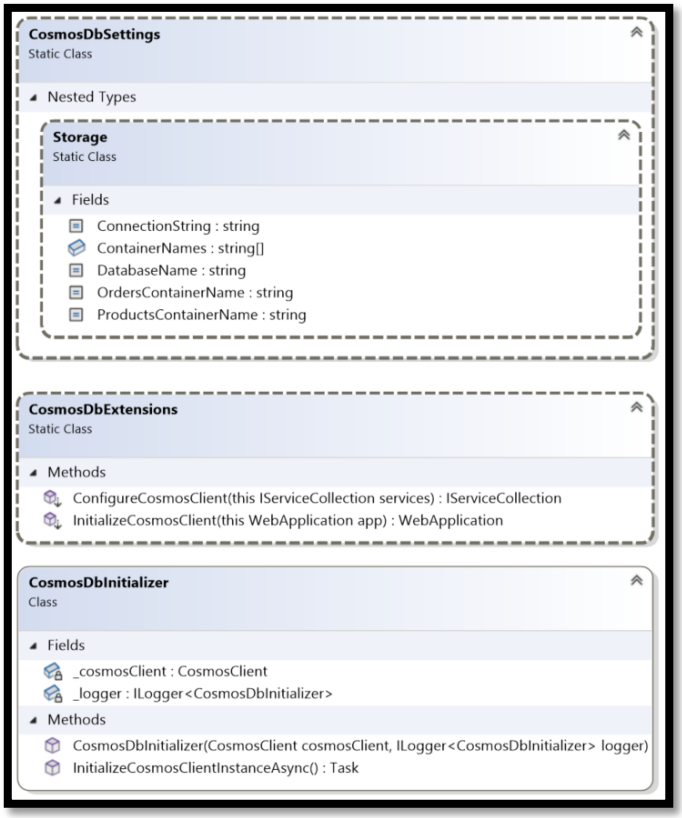

Рис. 3. Основні класи для роботи з базою даних

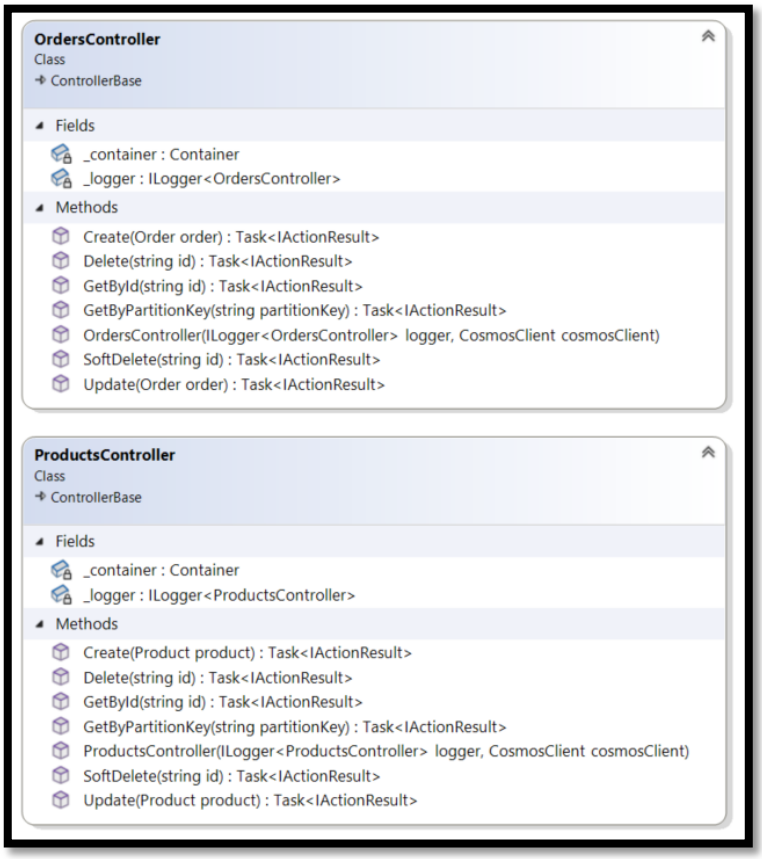

Рис. 4. Класи контролерів для роботи з колекціями

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2023. No. 2 (24)

Кожна програмна реалізація повинна мати налаштування *Swagger* – набір інструментів, що дає змогу автоматично описувати *API* на основі коду, який нам знадобиться для досліджень. Щоб було зручно вимірювати показники для запитів, вирішено використовувати *Swagger OpenAPI*, який подає опис усіх методів контролера та має зручний інтерфейс для введення даних, що будуть відправлені на сервер.

Результати досліджень та їх обговорення

Під час дослідження виконувалися серії запитів запланованих типів і розраховувалися середні значення за обраними показниками, а саме: швидкістю виконання запитів, кількістю витраченої програмою пам'яті на виконання запиту та кількістю RU з боку БД.

Результати вимірювання за показниками для "INSERT-запит" зведені в табл. 2. Бачимо, що запити на створення замовлення найбільш ефективно та швидко виконало *MongoDB API*. Було витрачено менше пам'яті та RU. Отже, такий запит коштуватиме дешевше. Це пов'язано з тим, що *MongoClient* не повертає відповідь від сервера, на відміну від *CosmosClient* та *EF Core Cosmos*.

Результати вимірювання для "SELECT-запит № 1" зведено в табл. 3, для "SELECT-запит № 2" – у табл. 4.

З аналізу виконання запитів на отримання замовлення за *PartitionKey* та *id* можемо спостерігати ефективність *Entity Framework Core Cosmos*, що краще оптимізований для таких операцій. Якщо порівнювати кількість RU, можемо побачити, що *CosmosClient* та *EF Core Cosmos* мають приблизно однакову вартість виконання запиту, а *MongoClient*, навпаки, програє за цим параметром.

Результати вимірювання для "UPDATE-запит" зведено в табл. 5, для "SOFT DELETE-запит" – у табл. 6.

Таблиця 2. *Результати виконання запитів на створення замовлення*

API / Показники	Швидкість виконання запиту (мс)	Кількість витраченої пам'яті (Byte)	Вартість виконання (RU)
Cosmos DB API for NoSQL (Cosmos Client)	48.55	27776	10.67
Cosmos DB API for NoSQL (Entity Framework Core Cosmos)	54.22	51999	10.67
Cosmos DB API for MongoDB(Mongo Client)	48	17232	9.05

Таблиця 3. *Результати виконання запитів на отримання замовлень за полем PartitionKey*

API / Показники	Швидкість виконання запиту (мс)	Кількість витраченої пам'яті (Byte)	Вартість виконання (RU)
Cosmos DB API for NoSQL (Cosmos Client)	110.33	86605	3.12
Cosmos DB API for NoSQL (Entity Framework Core Cosmos)	2.77	8453	3.22
Cosmos DB API for MongoDB (Mongo Client)	64.55	35149	7.17

Таблиця 4. *Результати виконання запитів на отримання замовлення за id*

API / Показники	Швидкість виконання запиту (мс)	Кількість витраченої пам'яті (Byte)	Вартість виконання (RU)
Cosmos DB API for NoSQL (Cosmos Client)	86.44	54949	2.83
Cosmos DB API for NoSQL (Entity Framework Core Cosmos)	55.44	29012	2.83
Cosmos DB API for MongoDB (Mongo Client)	130.77	25977	3.11

Таблиця 5. *Результати виконання запитів на оновлення замовлення*

API / Показники	Швидкість виконання запиту (мс)	Кількість витраченої пам'яті (Byte)	Вартість виконання (RU)
Cosmos DB API for NoSQL (Cosmos Client)	49.11	42915	10.67
Cosmos DB API for NoSQL (Entity Framework Core Cosmos)	106.77	21033	10.67
Cosmos DB API for MongoDB (Mongo Client)	179.11	32885	11.29

Сучасний стан наукових досліджень та технологій в промисловості. 2023. № 2 (24)
ISSN 2522-9818 (print)
ISSN 2524-2296 (online)

Таблиця 6. *Результати виконання запитів на м'яке вилучення замовлення*

API / Показники	Швидкість виконання запиту (мс)	Кількість витраченої пам'яті (Byte)	Вартість виконання (RU)
Cosmos DB API for NoSQL (Cosmos Client)	47.88	58895	10.67
Cosmos DB API for NoSQL (Entity Framework Core Cosmos)	104	28076	10.67
Cosmos DB API for MongoDB (Mongo Client)	108.77	28662	11.29

Аналізуючи результати запитів на оновлення замовлення та м'яке вилучення, спостерігаємо, що *CosmosClient* майже удвічі швидше виконує цю операцію, на відміну від *EF Core Cosmos* та *MongoClient*, але витрачає на це майже удвічі більше пам'яті. М'яке вилучення – це той самий запит на оновлення сутності зі зміною лише одного поля. З погляду ефективності в RU перші два підходи на рівних із показником у 10.67 RU проти 11.29 RU у *MongoClient*.

Результати вимірювання для "DELETE-запит" подано в табл. 7. Аналіз запитів на вилучення замовлення показав, що *EF Core Cosmos* та *Mongo Client* виконують цю операцію майже удвічі швидше, але витрачають на це значно більше пам'яті. *MongoClient* робить вилучення за 7.05 RU проти 10.67 RU у *CosmosClient* та *EF Core Cosmos*.

Отже, що кожен із розглянутих методів має свої переваги й недоліки, і вибір залежатиме від конкретних завдань розробника. Проте можна запропонувати деякі рекомендації:

– якщо говоримо про вартість запитів до БД або необхідність оптимізувати витрати на БД, а водночас є багато запитів на створення та вилучення записів, то краще обрати реалізацію *Cosmos DB API for MongoDB* через *Mongo Client*, бо цей програмний підхід використовує меншу кількість RU на виконання таких запитів;

– якщо ж частіше застосовуються операції оновлення та отримання даних, то однаково економними, але кращими, порівняно з *Mongo Client*, будуть *Cosmos DB API for NoSQL* (*Cosmos Client* та *Entity Framework Core Cosmos*);

– якщо необхідно багато та швидко читати інформацію з БД, потрібно використовувати *Cosmos DB API for NoSQL* з *Entity Framework Core Cosmos*, що може вивантажувати контекст БД в пам'ять та швидко оброблювати запити;

– якщо існують обмеження для програми по пам'яті, не рекомендується застосовувати *Entity Framework Core Cosmos*, бо він потребує багато пам'яті для роботи з контекстом БД;

– якщо необхідна оптимізація налаштування швидкості отримання певної групи об'єктів, потрібно налаштувати ключ розділу *Cosmos DB*, за допомогою якого запити на отримання певного розділу оптимізуються; найкраще цю оптимізацію можна спостерігати для *Entity Framework Core Cosmos* для *SELECT*-запиту за *PartitionKey*.

Таблиця 7. *Результати виконання запитів на вилучення замовлення*

API / Показники	Швидкість виконання запиту (мс)	Кількість витраченої пам'яті (Byte)	Вартість виконання (RU)
Cosmos DB API for NoSQL (Cosmos Client)	50.11	54949	10.67
Cosmos DB API for NoSQL (Entity Framework Core Cosmos)	106	18560	10.67
Cosmos DB API for MongoDB (Mongo Client)	99.22	31520	7.05

Щодо показника кількості коду, необхідно враховувати, що для невеликих проєктів вибір якогось із підходів не буде мати значних переваг для кожного з *API*. *Entity Framework Core Cosmos* потребує найменше коду для роботи з БД, але найбільше – для конфігурації підключення до БД. Саме тому ці переваги будуть помітні лише у великих проєктах зі складною структурованою логікою.

Висновки й перспективи подальшого розвитку

У роботі з погляду продуктивності досліджено методи програмної реалізації *Cosmos DB API* на платформі *.NET* на прикладі *Cosmos DB API for NoSQL* та *Cosmos DB API for MongoDB*. Проведено серію експериментів із вимірюваннями за показниками продуктивності виконання запитів до БД.

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) *Innovative technologies and scientific solutions for industries. 2023. No. 2 (24)*

З огляду на проведений аналіз методів програмної реалізації *Cosmos DB API* запропоновано програмні рішення на основі використання *CosmosClient, Entity Framework Core* для *Cosmos DB API for NoSQL* та на основі *MongoClient* для *Cosmos DB API for MongoDB.* Для проведення експериментів спроєктовано логічну модель БД у сфері електронної комерції для СУБД *MongoDB* та набір запитів на виконання *CRUD*-операцій, продуктивність яких і було досліджено.

Під час експериментів використано показники щодо швидкості виконання запитів (мс), кількості витраченної пам'яті на виконання запиту (*byte*), вартості запитів у RU та кількості коду, що необхідно написати.

Дослідження показало, що жоден із застосованих методів програмної реалізації не можна назвати однозначно найкращим.

Показник кількості коду, що написано під час реалізації, може братися до уваги лише в процесі розроблення великих проєктів зі складною структурованою логікою.

Під час обрання методу реалізації *Cosmos DB API* варто звертати увагу на такі фактори, як вартість запитів до БД, частота застосування тих чи інших запитів та обмеження пам'яті програми.

На основі результатів дослідження сформовано рекомендації щодо використання розглянутих методів. Ці рекомендації можуть бути застосовані для розроблення програмних систем на платформі *.NET* із використанням *Cosmos DB API.*

Список літератури

1. Filatov V., Semenets V. Methods for Synthesis of Relational Data Model in Information Systems Reengineering Problems. *International Scientific-Practical Conference Problems of Infocommunications. Science and Technology (PIC S&T). IEEE.* 2018. URL: https://www.researchgate.net/publication/331418031_Methods_for_Synthesis_of_Relational_Data_Model_in_Information_Systems_Reengineering_Problems

2. Smelyakov K., Prokopenko O., Chupryna A. Object-Based Image Comparison Algorithm Development for Data Storage Management Systems. *CEUR Workshop Proceedings*, 2022. № 3171. P. 1251–1266. URL: https://ceur-ws.org/Vol-3171/paper92.pdf

3. Mazurova, O., Naboka, A., Shirokopetleva, M. Research of ACID transaction implementation methods for distributed databases using replication technology. *Innovative technologies and scientific solutions for industries*, № 2 (16). 2021. P. 19–31. DOI: 10.30837/ITSSI.2021.16.019

4. Mazurova O., Syvolovskyi I., Syvolovska O. NOSQL database logic design methods for MONGODB and NEO4J. *Innovative technologies and scientific solutions for industries*, № 2 (20), 2022. P. 52–63. DOI: 10.30837/ITSSI.2022.20.052.

5. Sahatqija K., Ajdari J., Zenuni X., Raufi B., Ismaili F. Comparison between relational and NOSQL databases. *41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO).* 2018. P. 216–221. DOI: https://doi.org/10.23919/mipro.2018.8400041

6. Maran M., Paniavin N., Poliushkin I. Alternative Approaches to Data Storing and Processing. *V International Conference on Information Technologies in Engineering Education (Inforino).* 2020. P. 1–4. DOI: https://doi.org/10.1109/inforino48376.2020.9111708

7. Falatiuk H., Shirokopetleva M., Dudar Z. Investigation of Architecture and Technology Stack for e-Archive System. *IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T)*. 2019. P. 229–235. DOI: 10.1109/PICST47496.2019.9061407

8. Gomes C., Borba E., Tavares E., Junior M. N. de O. Performability Model for Assessing NoSQL DBMS Consistency. *IEEE International Systems Conference (SysCon)*. 2019. DOI: https://doi.org/10.1109/syscon.2019.8836757

9. Kuzochkina A., Shirokopetleva M., Dudar Z. Analyzing and Comparison of NoSQL DBMS. *International Scientific-Practical Conference on Problems of Infocommunications Science and Technology (PIC S&T). 2018.* P. 560–564. DOI: 10.1109/INFOCOMMST.2018.8632133

10. Bai Y. SQL Server Database Programming with Visual Basic.NET: Concepts, Designs and Implementations. 2020. 688 p, URL: https://www.wiley.com/en-ie/SQL+Server+Database+Programming+with+Visual+Basic+NET:+Concepts,+Designs+and+Implementations-p-9781119608608

11. Renée, M. P., Teate SQL for Data Scientists: A Beginner's Guide for Building Datasets for Analysis. 2021. 288 p, URL: https://www.wiley.com/en-us/SQL+for+Data+Scientists%3A+A+Beginner%27s+Guide+for+Building+Datasets+for+Analysis-p-9781119669364

12. Peretiatko M., Shirokopetleva M., Lesna N. Research of methods to support data migration between relational and document data storage models. *Innovative technologies and scientific solutions for industries.* № 2 (20). 2022. P. 64–74. DOI: 10.30837/ITSSI.2022.20.064

13. Palanisamy S., SuvithaVani P. A survey on RDBMS and NoSQL Databases MySQL vs MongoDB. *Conference: 2020 International Conference on Computer Communication and Informatics (ICCCI).* 2020. URL: https://www.researchgate.net/publication/341812161_A_survey_on_RDBMS_and_NoSQL_Databases_MySQL_vs_MongoDB

Сучасний стан наукових досліджень та технологій в промисловості. 2023. № 2 (24)
ISSN 2522-9818 (print)
ISSN 2524-2296 (online)

14. Ponniah P. Database Design and Development: An Essential Guide for IT Professionals. 2003. 768 p. URL: http://www.sbu.unicamp.br/bases-nfs/b131/lista_131_4.xlsx

15. Gruzdo I., Kyrychenko I., Tereshchenko G., Shanidze N. Metrics applicable for evaluating software at the design stage. *5th International Conference on Computational Linguistics and In-telligent Systems (COLINS-2021).* Kharkiv, Ukraine, April 22–23, 2021. CEUR Workshop Proceedings, 2021, Volume I. P. 916–936. URL: https://ceur-ws.org/Vol-2870/paper69.pdf

16. Perkins B., Panek W. Microsoft Azure Architect Technologies and Design Complete Study Guide: Exams AZ-303 and AZ-304. 2020. 768p. URL: https://www.wiley.com/en-ba/Microsoft+Azure+Architect+Technologies+and+Design+Complete+Study+Guide:+Exams+AZ+303+and+AZ+304-p-9781119559580

References

1. Filatov, V., Semenets, V. (2018), "Methods for Synthesis of Relational Data Model in Information Systems Reengineering Problems", *International Scientific-Practical Conference Problems of Infocommunications. Science and Technology (PIC S&T), IEEE*, available at: https://www.researchgate.net/publication/331418031_Methods_for_Synthesis_of_Relational_Data_Model_in_Information_Systems_Reengineering_Problems

2. Smelyakov, K., Prokopenko, O., Chupryna, A. (2022), "Object-Based Image Comparison Algorithm Development for Data Storage Management Systems", *CEUR Workshop Proceedings*, No. 3171, P. 1251–1266, available at: https://ceur-ws.org/Vol-3171/paper92.pdf

3. Mazurova, O., Naboka, A., Shirokopetleva, M. (2021) "Research of ACID transaction implementation methods for distributed databases using replication technology", *Innovative technologies and scientific solutions for industries*, No. 2 (16), P. 19–31. DOI: 10.30837/ITSSI.2021.16.019

4. Mazurova, O., Syvolovskyi, I., Syvolovska, O. (2022) "NOSQL database logic design methods for MONGODB and NEO4J", *Innovative technologies and scientific solutions for industries*, No. 2 (20), P. 52–63. DOI: 10.30837/ITSSI.2022.20.052

5. Sahatqija, K., Ajdari, J., Zenuni, X., Raufi, B., Ismaili, F., (2018), "Comparison between relational and NOSQL databases", *41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, P. 216–221. DOI: https://doi.org/10.23919/mipro.2018.8400041

6. Maran, M., Paniavin, N., Poliushkin, I. (2020), "Alternative Approaches to Data Storing and Processing", *V International Conference on Information Technologies in Engineering Education (Inforino)*, P. 1–4. DOI: https://doi.org/10.1109/inforino48376.2020.9111708

7. Falatiuk, H., Shirokopetleva, M., Dudar, Z. (2019), "Investigation of Architecture and Technology Stack for e-Archive System", *IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T)*, P. 229–235. DOI: 10.1109/PICST47496.2019.9061407

8. Gomes, C., Borba, E., Tavares, E., Junior, M. N. de O. (2019), "Performability Model for Assessing NoSQL DBMS Consistency", *IEEE International Systems Conference (SysCon)*, DOI: https://doi.org/10.1109/syscon.2019.8836757

9. Kuzochkina, A., Shirokopetleva, M., Dudar, Z. (2018), "Analyzing and Comparison of NoSQL DBMS", *International Scientific-Practical Conference on Problems of Infocommunications Science and Technology PIC S&T*, P. 560–564. DOI: 10.1109/INFOCOMMST.2018.8632133

10. Bai, Y., (2020), "SQL Server Database Programming with Visual Basic.NET: Concepts, Designs and Implementations". 688 p, available at: https://www.wiley.com/en-ie/SQL+Server+Database+Programming+with+Visual+Basic+NET:+Concepts,+Designs+and+Implementations-p-9781119608608

11. Renée, M. (2021), Teate SQL for Data Scientists: A Beginner's Guide for Building Datasets for Analysis. 288 p. available at: https://www.wiley.com/en-us/SQL+for+Data+Scientists%3A+A+Beginner%27s+Guide+for+Building+Datasets+for+Analysis-p-9781119669364

12. Peretiatko, M., Shirokopetleva, M., Lesna, N. (2022) "Research of methods to support data migration between relational and document data storage models", *Innovative technologies and scientific solutions for industries*, No. 2 (20), P. 64–74. DOI: 10.30837/ITSSI.2022.20.064

13. Palanisamy, S., SuvithaVani, P. (2020), "A survey on RDBMS and NoSQL Databases MySQL vs MongoDB", *International Conference on Computer Communication and Informatics (ICCCI)*, available at: https://www.researchgate.net/publication/341812161_A_survey_on_RDBMS_and_NoSQL_Databases_MySQL_vs_MongoDB

14. Ponniah, P. (2003), *Database Design and Development: An Essential Guide for IT Professionals*. 768 p, available at: http://www.sbu.unicamp.br/bases-nfs/b131/lista_131_4.xlsx

15. Gruzdo, I., Kyrychenko, I., Tereshchenko, G., Shanidze, N. (2021), "Metrics applicable for evaluating software at the design stage", *5th International Conference on Computational Linguistics and In-telligent Systems (COLINS-2021)*, Kharkiv, Ukraine, April 22–23, CEUR Workshop Proceedings, 2021, Volume I, P. 916–936, available at: https://ceur-ws.org/Vol-2870/paper69.pdf

16. Perkins, B., Panek, W. (2020), "Microsoft Azure Architect Technologies and Design Complete Study Guide: Exams AZ-303 and AZ-304". 768p. available at: https://www.wiley.com/en-ba/Microsoft+Azure+Architect+Technologies+and+Design+Complete+Study+Guide:+Exams+AZ+303+and+AZ+304-p-9781119559580

Received 5.05.2023

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) *Innovative technologies and scientific solutions for industries. 2023. No. 2 (24)*

Відомості про авторів / About the Authors

Мазурова Оксана Олексіївна – кандидат технічних наук, доцент, Харківський національний університет радіоелектроніки, доцент кафедри програмної інженерії, Харків, Україна; e-mail: oksana.mazurova@nure.ua; ORCID ID: https://orcid.org/0000-0003-3715-3476

Андрущенко Микола Олександрович – Харківський національний університет радіоелектроніки, магістр спеціальності 121 "Інженерія програмного забезпечення", Харків, Україна; e-mail: mykola.andrushchenko@nure.ua; ORCID ID: https://orcid.org/0009-0000-6866-1065

Широкопетлєва Марія Сергіївна – Харківський національний університет радіоелектроніки, старший викладач кафедри програмної інженерії, Харків, Україна; e-mail: marija.shirokopetleva@nure.ua; ORCID ID: https://orcid.org/0000-0002-7472-6045

Mazurova Oksana – PhD (Engineering Sciences), Associate Professor, Kharkiv National University of Radio Electronics, Associate Professor at the Department of Software Engineering, Kharkiv, Ukraine.

Andrushchenko Mykola – Kharkiv National University of Radio Electronics, Master of Specialty 121 "Software Engineering", Kharkiv, Ukraine.

Shirokopetleva Mariya – Kharkiv National University of Radio Electronics, Senior Lecturer at the Department of Software Engineering, Kharkiv, Ukraine.

RESEARCH OF METHODS OF SOFTWARE IMPLEMENTATION OF THE COSMOS DB API ON THE .NET PLATFORM

Large number of developers use the .NET platform to create applications that work with databases today. In turn, Cosmos DB is becoming an increasingly popular choice as a NoSQL storage for such databases. Cosmos DB is a flexible and scalable system, and the correct selection of the appropriate API during software implementation can significantly affect the performance of the programs themselves. Cosmos DB provides different APIs for working with different types of databases, such as SQL databases, running MongoDB or Cassandra. In turn, each of these APIs can be used using various methods of software implementation. **The subject** of research is software implementations on the .NET platform for various Cosmos DB APIs. When choosing the most suitable Cosmos DB API on the .NET platform, developers can be helped not only by the documentation, but also by the results of experimental studies of these APIs, which in turn will improve the quality of the code and the performance of the systems themselves. **The goal** of the work is to increase the efficiency of software development on the .NET platform. That use the Cosmos DB API, by developing a recommendation for the selection of software implementation methods for these APIs based on the results of their experimental research. **The task**: to investigate and compare the methods of software implementation of the Cosmos DB API through an experimental study of the performance of different types of queries on these software solutions; analyze the obtained results and develop recommendations for the use of methods. **Methods**: multi-criteria analysis of Cosmos DB API, logical data modeling, experimental research. **Results**: developed software solutions based on the use of CosmosClient, Entity Framework Core for Cosmos DB API for NoSQL and based on MongoClient for Cosmos DB API for MongoDB. A series of experiments and measurements of performance metrics for each of the software solutions were conducted, the obtained results were analyzed, and recommendations were offered for using the considered methods of software implementations of the Cosmos DB API on the .NET platform. **Conclusion**: In general, the choice of software approach depends on the specific task, but experiments have shown that CosmosDB API for NoSQL using CosmosClient is the best choice for small projects, and using the Entity Framework Core Cosmos is suitable for more complex projects with larger volumes of data and complex queries. If MongoDB is used in the project, then the corresponding solution using MongoClient is a better option than Cosmos DB API for NoSQL.

Keywords: DATABASE; COSMOS DB API; MONGODB; .NET; NOSQL.

Бібліографічні описи / Bibliographic descriptions

Мазурова О. О., Андрущенко М. О., Широкопетлєва М. С. Дослідження методів програмної реалізації *Cosmos DB API* на платформі *.NET*. *Сучасний стан наукових досліджень та технологій в промисловості*. 2023. № 2 (24). С. 118–130. DOI: https://doi.org/10.30837/ITSSI.2023.24.118

Mazurova, O., Andrushchenko, M., Shirokopetleva, M. (2023), "Research of methods of software implementation of the *Cosmos DB API* on the *.NET* platform", *Innovative Technologies and Scientific Solutions for Industries*, No. 2 (24), P. 118–130. DOI: https://doi.org/10.30837/ITSSI.2023.24.118