Лекція 7. Cosmos DB

Принцип суворої узгодженості (strong consistency) у **Cosmos DB** забезпечує найбільш строгий рівень узгодженості серед усіх доступних режимів узгодженості в цій базі даних. Він гарантує, що всі операції читання після запису отримають останню записану версію даних у системі, незалежно від того, з якого географічного розташування виконується запит.

Основні характеристики суворої узгодженості:

1.Глобальна послідовність: Якщо запис був здійснений в одному регіоні, інші регіони не зможуть отримати попередню версію даних. Це означає, що після завершення операції запису всі запити на читання, що виконуються будь-де у світі, повертають саме цю останню версію.

2.Лінійна узгодженість (linearizability): Читання даних гарантує повернення найсвіжіших даних. Всі операції запису та читання виконується у глобальному порядку, що створює ілюзію того, що всі операції відбуваються в одній, глобальній черзі.

3.Забезпечення порядку: Усі запити на запис виконуються в строгій послідовності. Тобто кожен наступний запит бачить зміни, внесені попереднім, і це працює на всіх рівнях системи.

4.Низька доступність і висока латентність: Оскільки сувора узгодженість вимагає підтвердження змін від усіх реплік у системі, перш ніж вони стануть видимими для читання, це збільшує затримки (латентність). Також можливі обмеження доступності у випадку, якщо деякі репліки тимчасово недоступні, оскільки всі репліки повинні підтвердити операцію.

5.Фіксація даних: Як тільки операція запису завершена, жоден клієнт не зможе отримати попередню версію цих даних. Це забезпечує консистентне відображення даних для всіх користувачів та регіонів.

Сценарії використання суворої узгодженості:

•**Фінансові транзакції**: Де важливо мати найсвіжіші дані для прийняття рішень і уникнення конфліктів при оновленнях.

•**Робота з системами управління інвентарем**: Коли потрібна гарантія того, що інформація про кількість товарів завжди актуальна.

•**Системи керування документацією**: Де критично важливо отримувати останні зміни в документі.

Переваги та недоліки:

•**Переваги**: Найвищий рівень узгодженості даних, мінімізація можливих конфліктів між даними, повна впевненість в актуальності даних.

•**Недоліки**: Збільшення латентності через необхідність узгодження всіх реплік, зниження доступності у разі мережевих проблем або недоступності певних реплік.

STRONG BOUNDED STALENESS SESSION CONSISTENT PREFIX EVENTUAL

Strong consistency provides the most predictable and intuitive programming model. When you configure your account with strong consistency level, Azure Cosmos DB provides linearizability guarantee.

Understand Strong consistency

This means that reads are guaranteed to see the most recent write.

Принцип обмеженої із запізненням узгодженості (Bounded Staleness Consistency) в **Cosmos DB** є одним із п'яти рівнів узгодженості, які пропонує ця база даних. Він дозволяє контролювати баланс між узгодженістю даних та продуктивністю, забезпечуючи більш гнучке управління тим, як швидко дані стають доступними після запису.

Основна ідея

Обмежена із запізненням узгодженість забезпечує, що всі читачі отримують дані, які можуть відставати від останнього запису лише на:

1.Проміжок часу (T): Наприклад, читачі отримують дані, які відстають не більше ніж на 10 секунд від найостанніших.

Це означає, що, на відміну від сильної узгодженості (де читачі завжди отримують найсвіжіші дані після запису), обмежена із запізненням узгодженість дозволяє тимчасове відставання, але при цьому гарантує, що це відставання обмежене певними параметрами (K або T).

Як це працює:

•**При записі**: Записані дані не обов'язково відразу стають видимими для всіх клієнтів. Існує затримка у вигляді дозволеної кількості старих версій даних або дозволеного інтервалу часу.

•**При читанні**: Клієнти можуть отримувати дані, які трохи застаріли, але вони ніколи не будуть відставати більше ніж на визначену кількість версій або час.

Переваги обмеженої із запізненням узгодженості:

1.Баланс між продуктивністю і узгодженістю: Оскільки система не завжди повинна негайно синхронізуватися між всіма репліками, це дозволяє покращити продуктивність, не вимагаючи сильної узгодженості.

2.Контрольована узгодженість: Користувач може точно визначити максимальне відставання, яке система дозволить, що надає гнучкість в залежності від вимог застосунку.

3.Глобальна реплікація: Цей рівень узгодженості особливо підходить для глобально розподілених баз даних, оскільки він дозволяє певне відставання між регіонами, що зменшує затримки при записах і покращує продуктивність для читачів.

Застосування:

Обмежена із запізненням узгодженість підходить для сценаріїв, коли система повинна забезпечити прийнятну узгодженість, але невелике відставання в даних допустиме для підвищення продуктивності. Наприклад, у глобально розподілених системах, де потрібна швидка доступність, але допускається тимчасове відставання.

STRONG | BOUNDED STALENESS | SESSION | CONSISTENT PREFIX | EVENTUAL

Bounded staleness consistency is most frequently chosen by globally distributed applications expecting low write latencies but total global order guarantees.

Maximum Lag (Time)

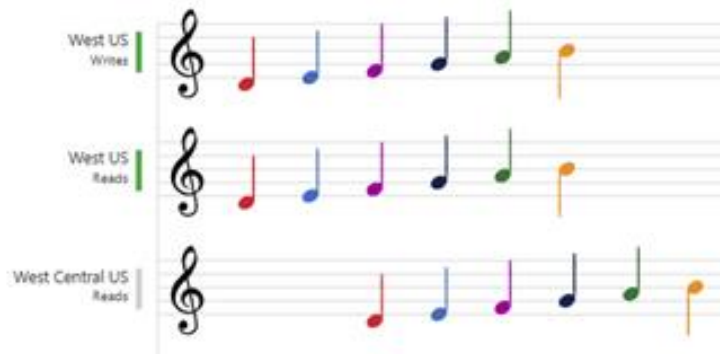
Days	Hours	Minutes	Seconds
0	0	0	5

Maximum Lag (Operations) *

100

Understand Bounded Staleness consistency

Unlike strong consistency which is scoped to a single region, you can choose bounded staleness consistency with any number of read regions (along with a write region). Bounded staleness is great for applications featuring group collaboration an publish-subscribe/queueing etc.



Принцип узгодженості сеансів (Session Consistency) у **Cosmos DB** — це один із п'яти рівнів узгодженості, які надає ця база даних для роботи з розподіленими даними. Він забезпечує баланс між строгістю узгодженості та продуктивністю, зокрема для сеансів користувачів або окремих клієнтських додатків. Ось як це працює:

Основні аспекти узгодженості сеансів:

1.Гарантована узгодженість для одного сеансу:

1. Усі операції, що виконуються в межах одного клієнтського сеансу (тобто за одну сесію підключення до Cosmos DB), гарантовано будуть узгодженими. Це означає, що клієнт завжди бачить результати своїх власних записів або змін у даних.
2. Наприклад, якщо користувач виконує запис, а потім одразу ж читає дані, він побачить щойно внесені зміни.

2.Моніторинг версій даних:

1. Протягом одного сеансу клієнт завжди бачить послідовність своїх дій, навіть якщо інші клієнти вносять зміни в дані паралельно. Це важливо для ситуацій, коли клієнт оновлює або читає дані кілька разів поспіль.
2. В інших словах, користувач працює з найактуальнішою версією даних з власних операцій, навіть якщо глобально ці дані ще не повністю поширилися серед інших клієнтів.

3.Відсутність гарантій для інших сеансів:

1. Узгодженість на рівні сеансів не забезпечує того, що всі інші клієнти побачать зміни, виконані у поточному сеансі, миттєво. Тобто інші користувачі можуть працювати зі старішими версіями даних, доки система не оновить їх на глобальному рівні.
2. Це добре підходить для додатків, де кожен клієнт має свою локальну сесію, але зміни, внесені іншими користувачами, не є критичними для негайного оновлення у клієнта.

4.Підвищена продуктивність:

1. Цей рівень узгодженості забезпечує хорошу продуктивність та низьку затримку, оскільки дані не потрібно негайно синхронізувати на всіх вузлах системи.
2. Сеансовий рівень узгодженості є ефективнішим у порівнянні з більш строгими моделями, такими як **Strong Consistency**, оскільки він не вимагає повної глобальної синхронізації для кожної операції запису.

Переваги принципу узгодженості сеансів у Cosmos DB:

•**Баланс між узгодженістю та продуктивністю**: Принцип узгодженості сеансів забезпечує хороший баланс між необхідністю бачити свої зміни (локальна узгодженість) та загальною продуктивністю системи.

•**Низькі затримки**: Оскільки немає необхідності синхронізувати дані на глобальному рівні після кожної операції, це значно знижує затримки.

•**Простота для сеансів користувачів**: Якщо клієнтська сесія потребує відслідковування своїх змін, цей рівень узгодженості ідеально підходить для таких сценаріїв, як обробка кошика покупок чи відстеження стану користувача.

Приклади використання:

•Інтернет-магазини: клієнт може бачити свої зміни в кошику або історії покупок, але не потрібно негайно бачити зміни, які зробили інші користувачі.

•Персоналізовані додатки: дані, які стосуються лише одного користувача, не потребують глобальної узгодженості, але важливо, щоб цей користувач завжди бачив актуальну інформацію у своїй сесії.

Опис принципу узгодженості префіксів (Prefix Consistency)
При виборі цього рівня консистентності гарантується, що клієнт отримуватиме оновлення в правильному порядку (з правильними префіксами операцій), але можуть бути пропущені деякі проміжні оновлення. Інакше кажучи, клієнт, який робить запит до даних, не побачить змішаних або перемішаних версій змін.
Основні характеристики:
1.Порядок оновлень зберігається: Оновлення даних застосовуються послідовно, що означає, що клієнт завжди бачить оновлення в правильному хронологічному порядку, як вони були виконані.
2.Можливе пропущення новіших оновлень: Клієнт може не бачити всі останні зміни (затримки можуть бути, наприклад, через реплікацію), але ті, які він бачить, будуть в правильному порядку.
3.Збалансований підхід: Цей рівень консистентності дає компроміс між стронкою консистентністю (де гарантується, що всі клієнти бачать однакові останні зміни) та ефективністю та низькою затримкою читання.
Приклад:
Уявімо, що три операції A, B і C записують зміни в базу даних у такій послідовності:
•Операція A — перша
•Операція B — друга
•Операція C — третя
При узгодженості префіксів клієнт може побачити послідовність:
•A
•A → B
•A → B → C
Однак клієнт ніколи не побачить B без A або C без B, що означає, що консистентність у порядку операцій гарантована.
Переваги узгодженості префіксів:
•**Забезпечення порядку**: Всі операції завжди застосовуються в правильному порядку.
•**Висока продуктивність**: У порівнянні зі стронкою консистентністю, цей підхід забезпечує меншу затримку при читанні.
•**Широка застосовність**: Підходить для сценаріїв, де важливо бачити операції у правильному порядку, але не є критичним бачити всі найсвіжіші зміни.

Принцип підсумкової узгодженості (eventual consistency) в **Cosmos DB** відноситься до одного з варіантів забезпечення узгодженості даних у розподілених системах. Цей підхід гарантує, що в результаті виконання певної кількості операцій над даними всі копії цих даних у різних репліках або вузлах системи зрештою стануть однаковими, тобто досягнуть узгодженого стану. Однак, на відміну від більш строгих рівнів узгодженості, тут не гарантується миттєва узгодженість.

Основні принципи підсумкової узгодженості:

1.Затримка в узгодженні: Після того, як дані оновлено в одній репліці (вузлі), оновлення поширюється на інші репліки з деякою затримкою. Інші репліки можуть тимчасово бачити старі значення даних до того, як оновлення досягне їх.

2.Поступове досягнення узгодженості: З плином часу і після виконання достатньої кількості операцій всі копії даних зрештою синхронізуються, і всі вузли матимуть однакові версії даних.

3.Баланс між продуктивністю та узгодженістю: Підсумкова узгодженість надає більш високу продуктивність і менші затримки в порівнянні з моделями сильної узгодженості (наприклад, сувора узгодженість або узгодженість за читанням і записом), оскільки немає необхідності в синхронному оновленні всіх вузлів.

4.Невизначеність у проміжний момент: У короткий проміжок часу після оновлення різні клієнти можуть бачити різні версії даних, в залежності від того, до якого вузла або репліки вони звернулися.

Переваги підсумкової узгодженості в Cosmos DB:

•**Висока доступність**: Завдяки асинхронному оновленню копій даних можна забезпечити високу доступність системи навіть під час збоїв або затримок у мережі.

•**Масштабованість**: Ця модель легко масштабується в умовах великих розподілених систем, таких як глобальні додатки або географічно розподілені бази даних.

•**Низькі затримки**: Підсумкова узгодженість дозволяє мінімізувати затримки при записах і читанні, що робить її підходящою для систем з високими вимогами до швидкодії.

Недоліки:

•**Можливі тимчасові неконсистентності**: Поки система досягає узгодженості, клієнти можуть працювати зі старими або несинхронізованими даними.

•**Непередбачуваність термінів узгодженості**: Немає точних гарантій щодо того, скільки часу займе досягнення узгодженості.

Приклад у Cosmos DB:

Уявімо, що ви оновлюєте запис у Cosmos DB в одному географічному регіоні. Якщо у вас увімкнено підсумкову узгодженість, клієнти в інших регіонах можуть не бачити це оновлення негайно, а отримувати старі версії даних. Проте після певного часу всі клієнти в будь-якому регіоні почнуть бачити оновлену версію даних, коли синхронізація завершиться.

Create a free tier account for API for NoSQL

```
# Create a free tier account for API for NoSQL.
$grp="DBdemo"
$location="westeurope"


az group create --name $grp --location $location
New-AzCosmosDBAccount -ResourceGroupName $grp -Name "soloveiaccount" -ApiKind "sql" -
EnableFreeTier $true -DefaultConsistencyLevel "Session" -Location $location
```

Оцінка завантаженості

Кожна операція бази даних споживає системні ресурси. Споживання залежить від складності операції. ОЗ (одиниця запиту) в секунду – це одиниця продуктивності, яка обчислює системні ресурси (наприклад, ЦП, операції введення-виведення в секунду та пам'ять), необхідні для виконання операцій бази даних, що підтримуються Azure Cosmos DB.

1 ОЗ = читання 1 КБ

Якщо потрібно виконати операцію читання 100 разі в секунду, тоді маємо 100 ОЗ.

Якщо для облікового запису включено рівень «Безкоштовний», то надається 1000 ОЗ в секунду і 25Гб сховища.

Типи пропускної здатності Azure

Підтримується 2 типи: стандартний (ручний), такий що автомасштабується, важливо спочатку зрозуміти, як працює підготовлена пропускна здатність в Azure Cosmos DB.

У наступній таблиці показано загальне порівняння стандартного (ручного) та автомасштабованого типів пропускної спроможності.

Опис	стандартний (ручний)	автомасштабується
Підходить для навантажень	з постійним трафіком (рис 1)	З непербачувальним трафиком (рис 2)
Принцип роботи	Наперед задається кількість ОЗ/с, яке не змінюється до тих пір поки, ви це не зміните самостійно. Наприклад, якщо задати 400 ОЗ/с, тоді пропускна спроможність буде завжди на цьому рівні.	Ви вказуєте найбільшу чи максимальну кількість ОЗ/с, яку система не повинна перевищувати. Система автоматично масштабує пропускну здатність Т таким чином, щоб вона не перевищувала 0.1* Tmax.
Модель обчислення вартості	Наприклад, визначено 400 ОЗ/с 1-ша година: запитів немає 2-га година: 400 ОЗ/с Кожен час буде оплачуватись однаково	Підготовлена максимальна пропускна здатність для автомасштабування: 4000 ОЗ/с (з масштабуванням від 400 до 4000 Година 1: система масштабувала пропускну здатність до максимального значення 3500 ОЗ/с Година 2: система масштабувала пропускну здатність до максимального значення в 400 ОЗ/с (завжди 10 % від Tmax) через невикористання Рахунок виставляється за 3 500 ОЗ/с за годину 1 та 400 ОЗ/с за годину 2 згідно з тарифами на автомасштабування підготовленої пропускної спроможності.
Якщо значення ОЗ/с перевищено	Всі запити будуть обмежені в швидкості	Всі запити будуть обмежені в швидкості

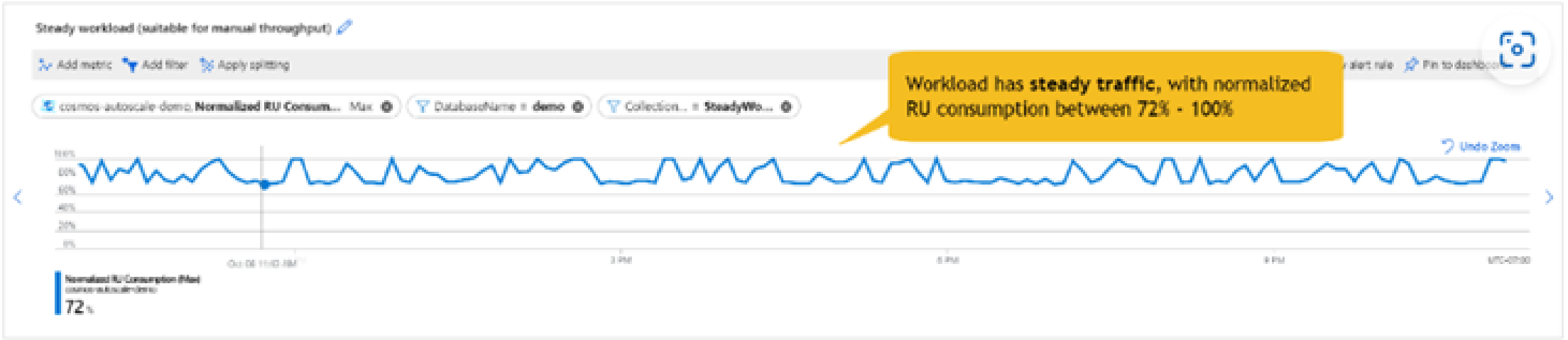


Рис 1. Стабільне навантаження. Якщо визначити пропускну здатність 30 000 ОЗ/с з нормалізованим використанням 72 % до 100 %, тоді виде використовуватись від 21 600 ОЗ/с до 30 000 ОЗ/с.

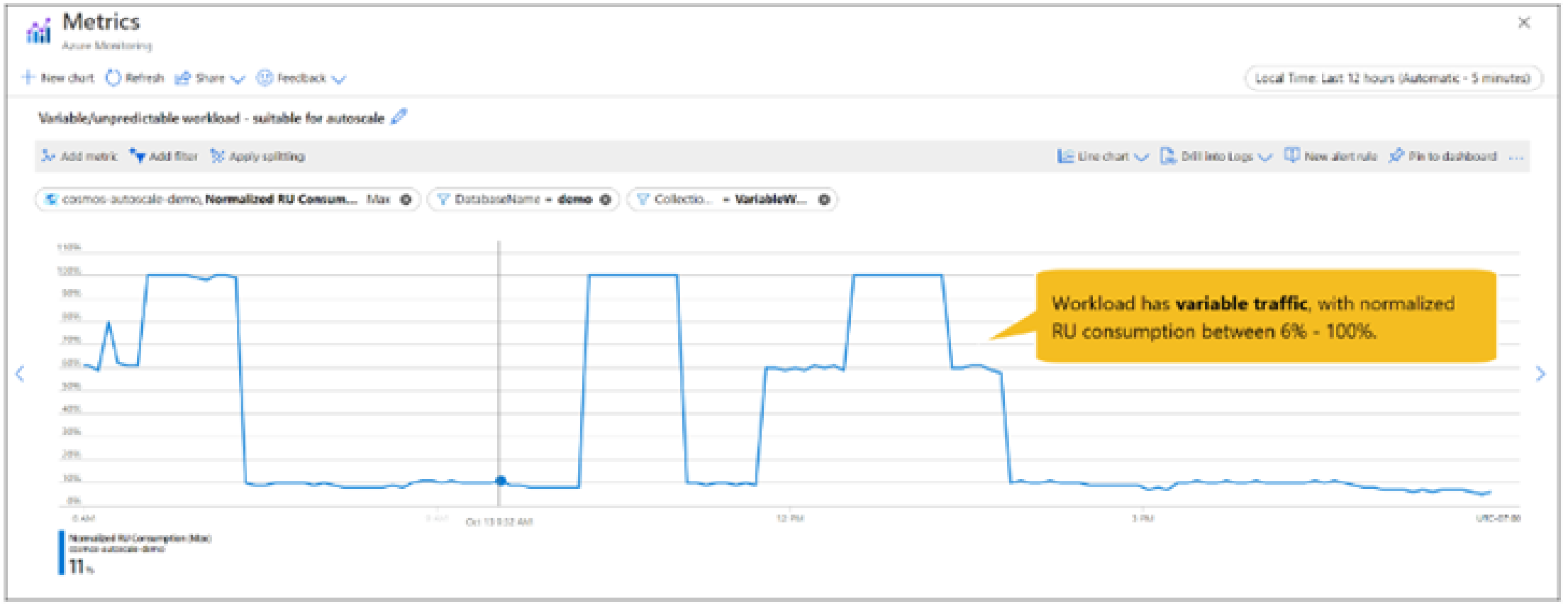
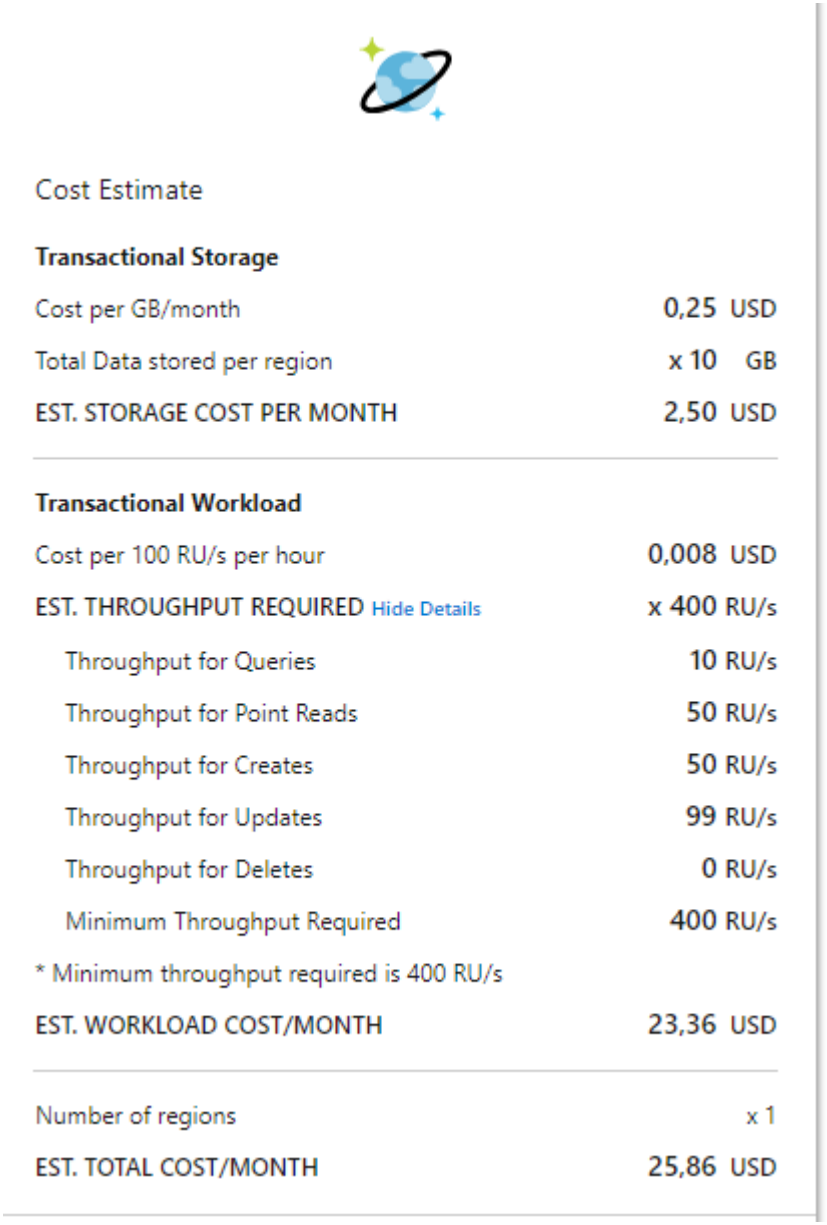


Рис 2. Робоче навантаження зі змінним трафік із нормалізованим споживанням ОЗ у діапазоні від 6% до 100%.

https://cosmos.azure.com/capacitycalculator/

«Пропускна здатність для запитів» означає кількість запитів за секунду, яку база даних може обробити на основі наданих налаштувань потужності тобто
кількість операцій читання або запису, які може підтримувати ваше робоче навантаження в межах установленого ліміту одиниці ресурсу (RU).
В межах 400 Request units буде виконано
10 запитів
50 – операцій читання окремого елемента на основі його унікального ідентифікатора (ідентифікатора та ключа розділу). Це дуже ефективна операція, яка зазвичай споживає менше одиниць запиту (RU) порівняно з запитами чи іншими операціями.
50 – операцій на «зміну» даних
99 – операцій на видалення



Cost Estimate

Transactional Storage	
Cost per GB/month	0,25 USD
Total Data stored per region	x 10 GB
EST. STORAGE COST PER MONTH	2,50 USD
Transactional Workload	
Cost per 100 RU/s per hour	0,008 USD
EST. THROUGHPUT REQUIRED Hide Details	x 400 RU/s
Throughput for Queries	10 RU/s
Throughput for Point Reads	50 RU/s
Throughput for Creates	50 RU/s
Throughput for Updates	99 RU/s
Throughput for Deletes	0 RU/s
Minimum Throughput Required	400 RU/s
* Minimum throughput required is 400 RU/s	
EST. WORKLOAD COST/MONTH	23,36 USD
Number of regions	x 1
EST. TOTAL COST/MONTH	25,86 USD

Фактори, які впливають, на рівень споживання ресурсів

Розмір елемента. У міру збільшення розміру елемента число ОЗ, споживане для читання чи запису елемента, також збільшується.

Індексація елемент. За замовчуванням кожен елемент автоматично індексується. Якщо ви вирішили не індексувати деякі елементи у контейнері, споживається менше ОЗ.

Кількість властивостей елемента. Припускаючи, що індексування виконується за умовчанням для всіх властивостей то кількість ОЗ, збільшується зі збільшенням кількості властивостей елемента.

Узгодженість даних. Суворі рівні узгодженості та узгодженості з обмеженим старінням споживають приблизно вдвічі більше ОЗ при виконанні операцій читання порівняно з іншими відстроченими рівнями узгодженості.

Тип операцій читання: точкове читання коштує менше ОЗ, ніж запити.

Шаблони запитів. Чинники, що впливають на вартість операцій запитів:

Кількість результатів запиту.

Кількість предикатів.

Характер предикатів.

Кількість функцій, що визначаються користувачем.

Розмір вихідних даних.

Розмір результуючого набору.

Проекції.

Режими керування ресурсами

Режими керування ресурсами, які називаються ***capacity modes***. Вони визначають, як база даних виділяє та використовує ресурси для обробки запитів і зберігання даних. Ці режими дають можливість вибору між гнучкістю та передбачуваністю витрат залежно від потреб бізнесу.

Основні режими:

- **Provisioned Throughput Mode (заздалегідь виділений пропускний режим)**
- **Serverless Mode (безсерверний режим)**

1. Provisioned Throughput Mode
У цьому режимі ви заздалегідь виділяєте необхідний обсяг пропускної здатності (throughput), вимірюваний у одиницях Request Units per second (RU/s). Це означає, що ви визначаєте, скільки RU/s необхідно для виконання операцій з даними (читання, запису, оновлення, запитів).

Переваги:
Передбачувані витрати: ви точно знаєте, скільки заплатите за виділену пропускну здатність, навіть якщо вона не використовується на повну.
Підтримка великих робочих навантажень: цей режим підходить для застосунків з постійним, стабільним обсягом запитів та транзакцій, де потрібно забезпечити високу продуктивність.
Автоматичне масштабування: можна використовувати Autoscale, щоб динамічно масштабувати RU/s в межах встановлених лімітів, що дозволяє адаптуватися до змін робочого навантаження.
Недоліки:
Якщо пропускна здатність перевищується, запити можуть бути обмежені, що впливає на продуктивність додатка.
Непотрібні витрати, якщо ви виділили більше ресурсу, ніж фактично потрібно.

2. ***Serverless Mode***
У безсерверному режимі вам не потрібно заздалегідь виділяти пропускну здатність. Ви платите тільки за фактично використані ресурси (RU), коли виконуються операції.

Переваги:

Гнучка оплата: ідеально підходить для нерегулярних або невеликих робочих навантажень, коли запити надходять рідко або є пікові моменти.
Відсутність необхідності управління ресурсами: ви не керуєте виділенням RU/s, оскільки вони автоматично масштабуються відповідно до фактичного навантаження.

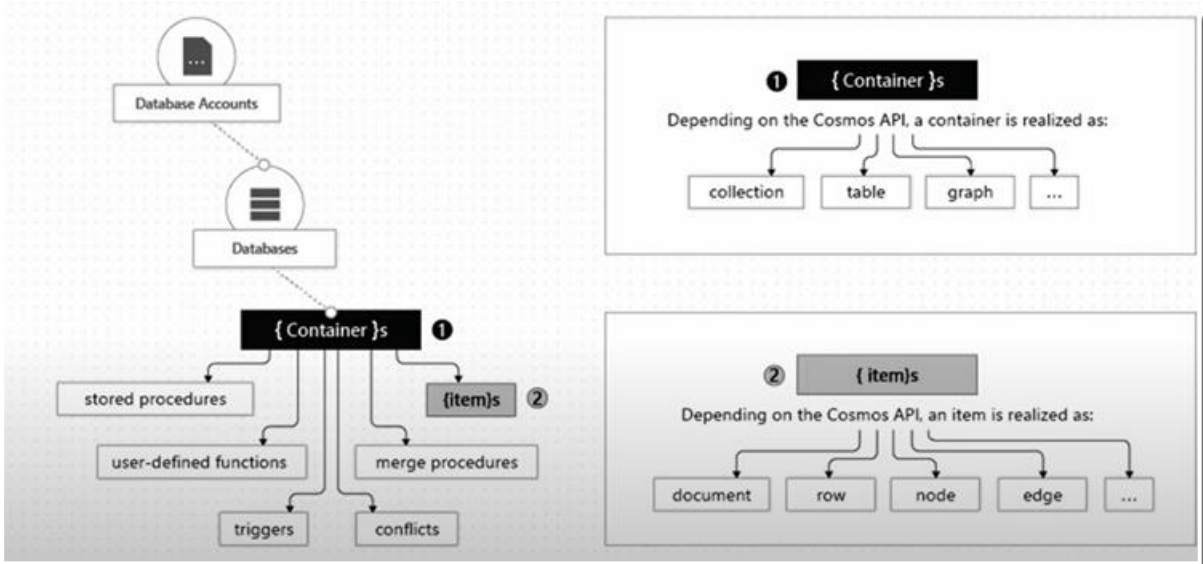
Недоліки:

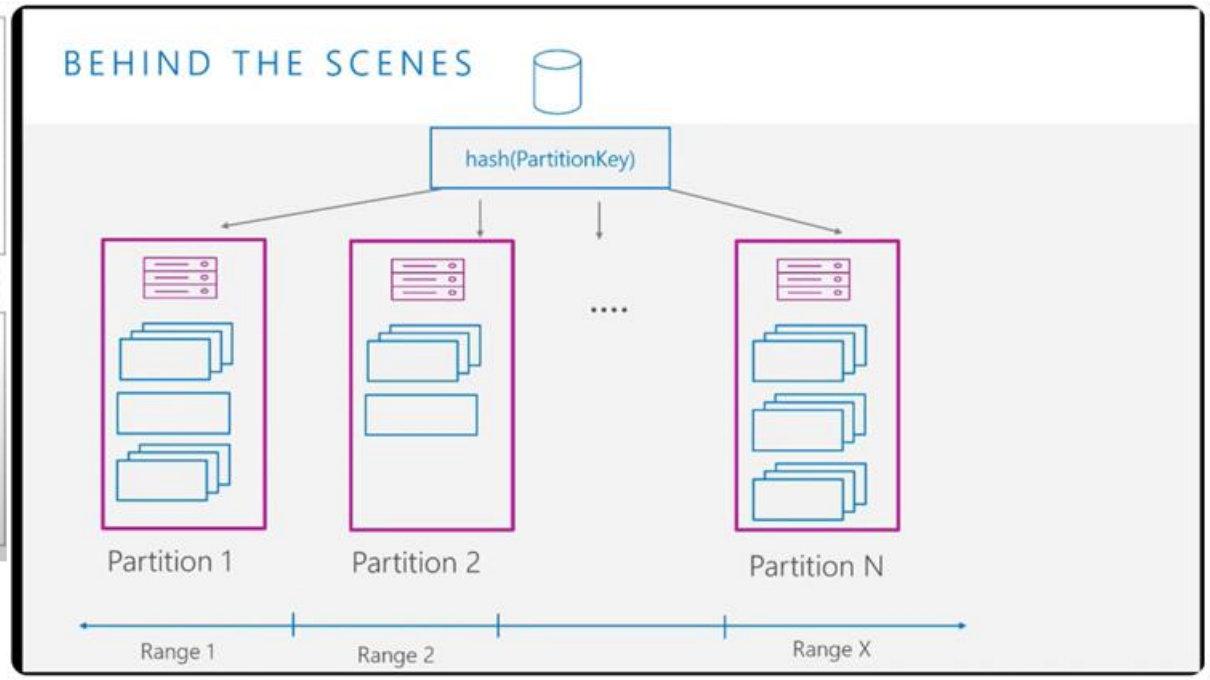
Не підходить для постійного високого навантаження: безсерверний режим менш економічно вигідний для додатків із постійно великим обсягом запитів.
Обмеження продуктивності: є обмеження на максимальний розмір робочого навантаження, яке можна обробити за певний час (наприклад, до 5000 запитів на секунду).

Вибір між режимами:
Provisioned Throughput Mode підходить для додатків, які потребують передбачуваної пропускної здатності та можуть мати стабільні або високі обсяги трафіку.
Serverless Mode краще підходить для застосунків з рідкими або нерегулярними запитами, тестових середовищ або малих проектів, де немає потреби у постійному виділенні ресурсів.

Горизонтальне та вертикальне масштабування

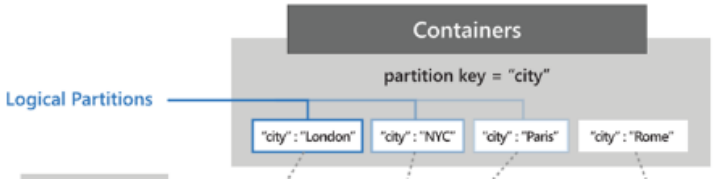
Реляційні бази даних зазвичай збільшуються шляхом збільшення розміру віртуальної машини або обчислювального середовища, де вони розміщені. Розглянемо структуру Cosmos DB для подальшого пояснення як виконується масштабування.



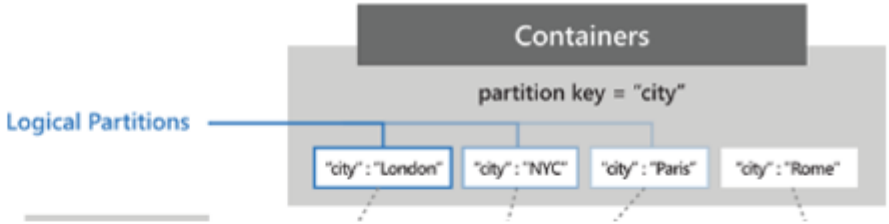


Дані одного контейнера розподіляються по різних «логічних секція» для забезпечення масштабування даних. Логічні секції формуються з урахуванням значення ключа секції, який є у кожного елемента у контейнері. Усі елементи у логічній секції мають однакове значення ключа секції.

Наприклад, для контейнеру, який включає «сутності» як показано в таблиці 1 та ключ секції: “city” – буде створено 4 логічні секції.



«сутності» items	Ключ секції – “city”

{
"id": "1001",
"city":"London",
"country":"UK"
}
{
"id": "1002",
"city": "NYC",
"country": "US",
}
{
"id": "1003",
"city":"Paric",
"country":"France"
}
 	
{
"id": "1004",
"city": "Rome",
"country": "Italy",
}
	

Крім ключа секції, що визначає логічну секцію, кожен елемент в контейнері також має ідентифікатор, який є унікальним в межах логічної секції. Поєднання ключа секції та ідентифікатора елемента створює індекс, що однозначно визначає елемент. Щоб збільшити масштаб бази даних NoSQL, потрібно додати додаткові сервери або вузли. Ці вузли також називаються **фізичними секціями** Cosmos DB.

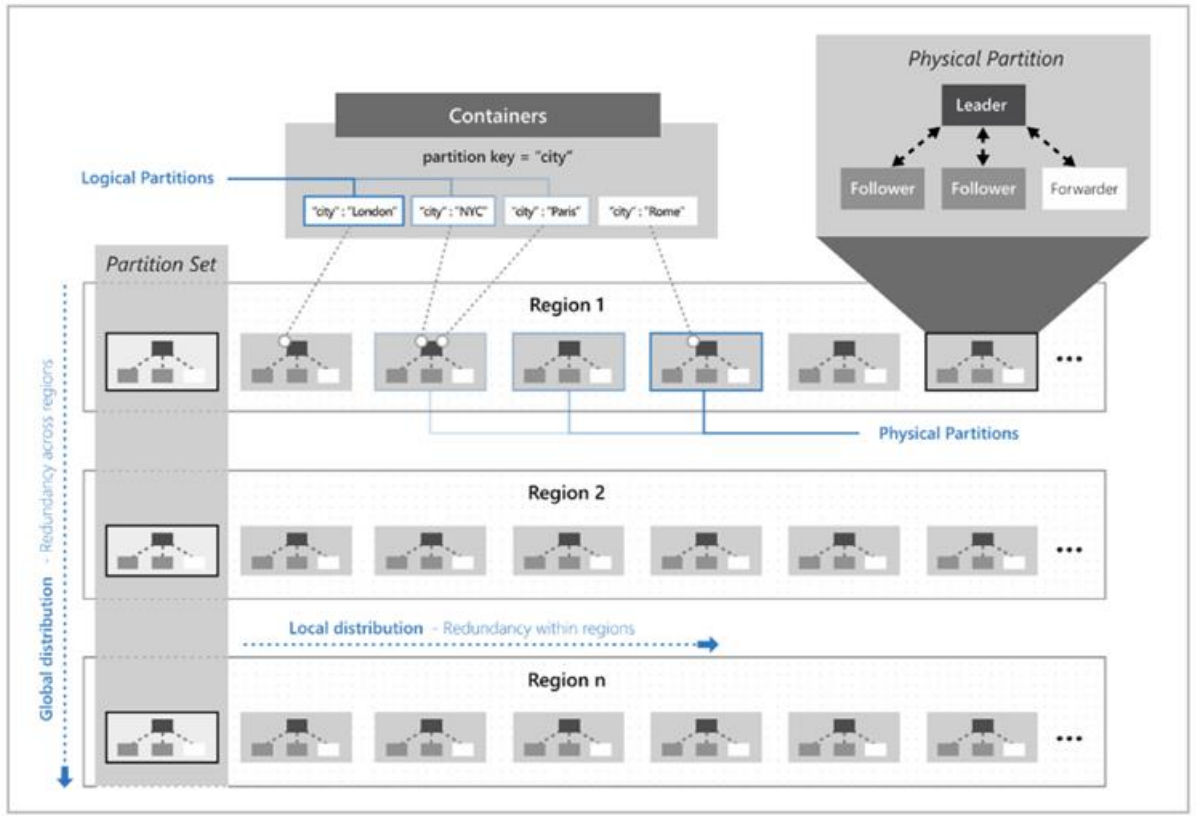
Один контейнер може мати багато логічних секцій з однією фізичною. Фізичні секції створюють відповідно правил платформи Azure, а їх кількість залежить від характеристик:

1. Підготовлена пропускна спроможність. (Кожна окрема фізична секція може забезпечити пропускну здатність до 10 000 одиниць запитів за секунду.) Обмеження 10 000 одиниць запитів за секунду для фізичних секцій передбачає, що для логічних секцій також встановлено обмеження 10 000 одиниць запитів за секунду, оскільки кожна секція зіставлена лише з однією фізичною секцією.
2. Загальне сховище даних (у кожній фізичній секції може зберігатись до 50 ГБ даних).

Загальна кількість фізичних секцій у контейнері не обмежена. У міру збільшення підготовленої пропускної спроможності або розміру даних, Azure Cosmos DB автоматично створює нові фізичні секції, розділяючи існуючі.

Розділення фізичних секцій не впливає на доступність програми. Після поділу фізичної секції всі дані з однієї логічної секції, як і раніше, зберігатимуться в тій же фізичній секції. Пропускна здатність, підготовлена для контейнера, розподіляється рівномірно між фізичними секціями.

Кожна фізична секція складається із набору реплік. У кожній репліці розміщено екземпляр ядра СУБД. На наступному малюнку показано, як логічні секції порівнюються з фізичними, які розподіляються глобально. Набір розділів у образі посилається на групу фізичних секцій, які керують одними й тими самими ключами логічних секцій у кількох регіонах:



Кожна репліка, що становить фізичну секцію, успадковує квоту сховища секції. Усі репліки фізичної секції спільно підтримують пропускну здатність, виділену для цієї секції.

Вибір ключа секції

Ключ секції – це обов'язкова властивість документа, яка гарантує, що документи з однаковим значенням ключа секції направляються у певну фізичну секцію та зберігаються у ній. Фізична секція підтримує фіксований максимальний обсяг сховища та пропускну здатність (ЕЗ/с). Azure Cosmos DB автоматично розподіляє логічні секції за доступними фізичними секціями передбачуваним чином завдяки значенню ключа секції.

В Azure Cosmos DB ви збільшуєте обсяг сховища та пропускну здатність, додаючи додаткові фізичні секції для доступу до даних та їх зберігання. Максимальний розмір сховища фізичної секції становить 50 ГБ, а максимальна пропускна спроможність – 10 000 одиниць запиту на секунду.

Для всіх контейнерів ключ секції повинен мати перелічені нижче властивості.

- ✓ Властивість із значенням, яке не змінюється.
- ✓ Ключі мають містити String тільки значення, або числа в ідеалі повинні бути перетворені на String, якщо є ймовірність того, що вони знаходяться за межами подвійної точності чисел відповідно до IEEE 754 binary64.
- ✓ Висока кратність. Іншими словами, властивість повинна мати широкий спектр можливих значень.
- ✓ Споживання одиниць запиту та зберігання даних має бути рівномірно розподіллено по всіх логічних секціях. Це забезпечує рівномірне споживання одиниць запиту та розподіл обсягів сховища за фізичними секціями.
- ✓ Мають значення, які зазвичай не перевищують 2048 б або 101 б, якщо ключі секцій великого розміру не включені.

Логічні секції в Azure Cosmos DB

Ключ секції гарантує, що документи з однаковим значенням ключа належать до однієї логічної секції, направлятимуться у певну фізичну секцію та зберігатимуться у ній. Поняття логічної секції дає змогу поєднувати документи з однаковим значенням ключа секції. Декілька логічних секцій можуть зберігатися в одній фізичній секції, а контейнер може містити необмежену кількість логічних секцій. Окремі логічні секції переміщуються до нових фізичних секцій як єдине ціле з розширенням контейнера. Переміщення логічних секцій як єдине ціле гарантує, що всі документи всередині них будуть знаходитися в одній фізичній секції. Максимальний розмір логічної секції становить 20 ГБ. Використання ключа секції з великою кратністю дозволяє уникнути обмеження 20 ГБ за рахунок того, що дані розподіляються по більшій кількості логічних секцій.

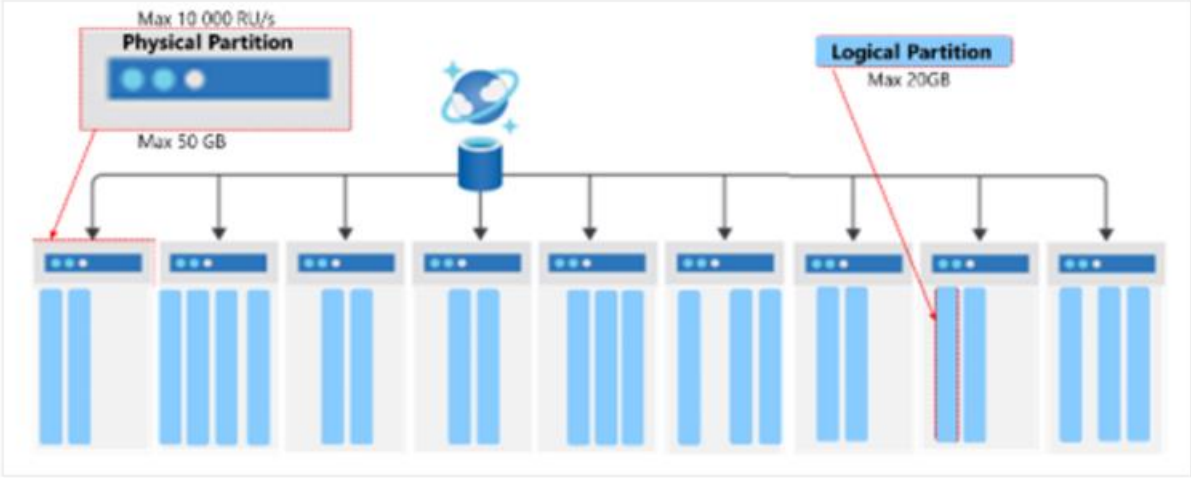
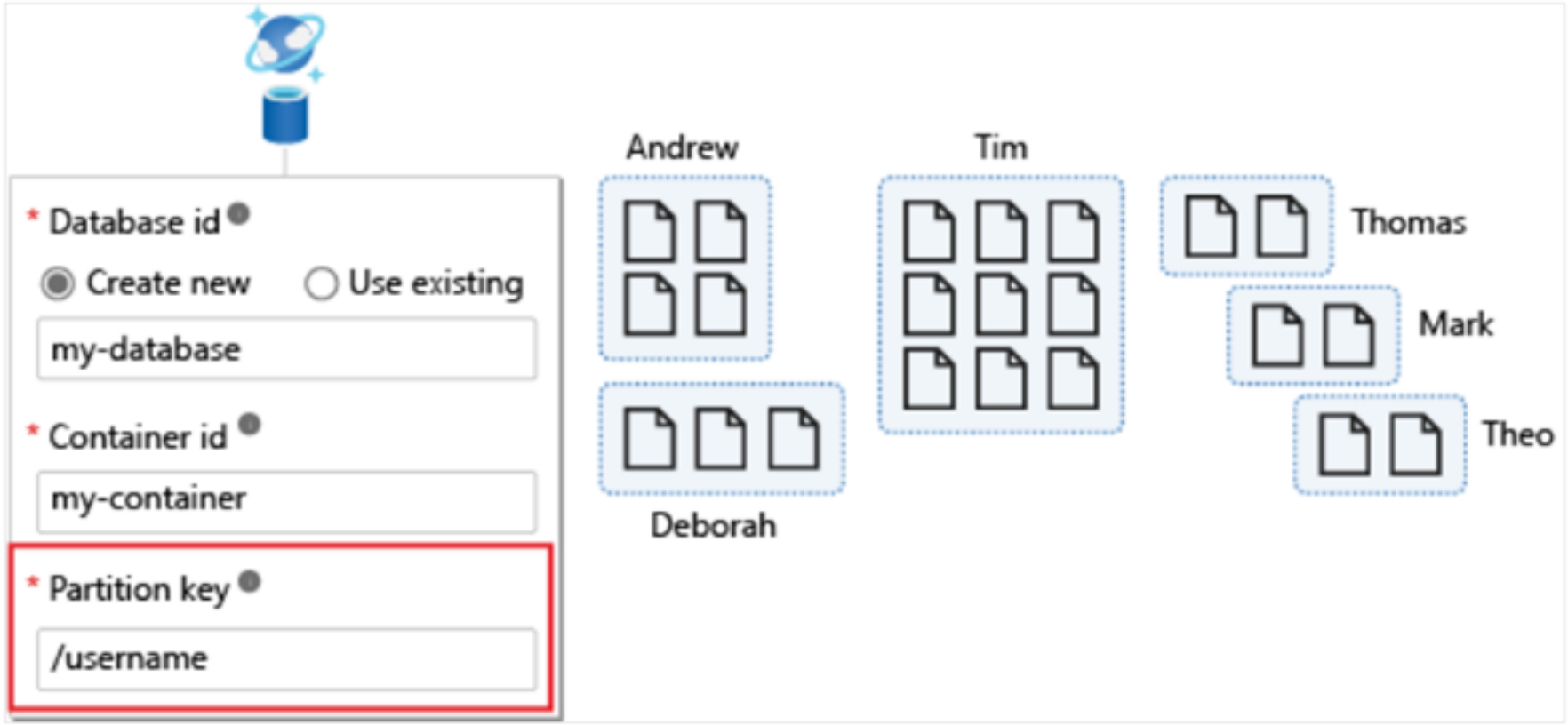


Рис 1. Логічні секції об’єднані в фізичні розділи

Ключ секції надає спосіб маршрутизації даних логічної секції. Це властивість, яка існує в кожному документі у контейнері, який маршрутизує дані.

Контейнер — це ще одна абстракція для всіх даних, що зберігаються з одним і тим самим ключем секції. Ключ секції визначається під час створення контейнера. У цьому прикладі контейнер має ключ секції /username.



Як уникнути гарячих розділів?

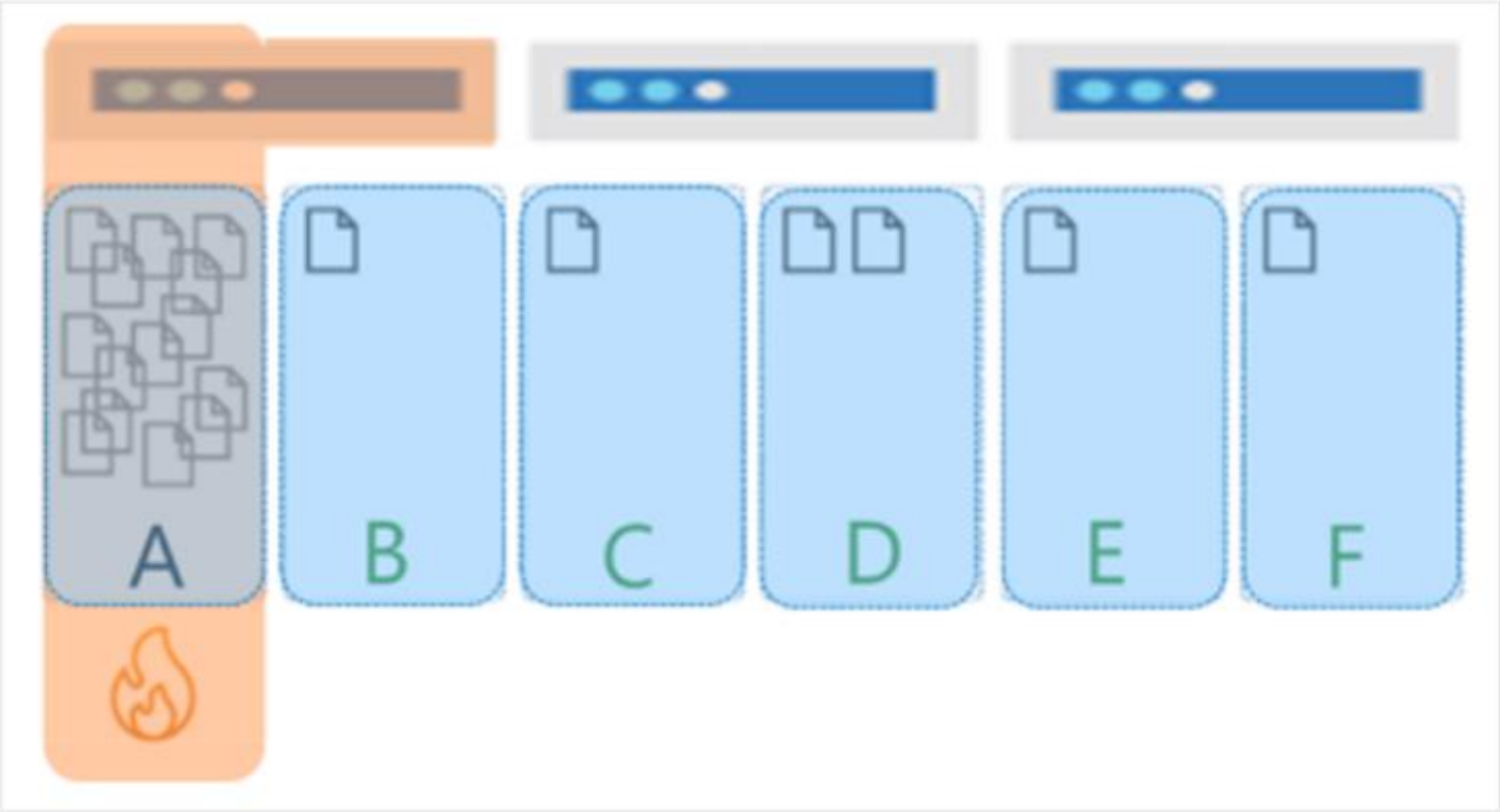
При моделюванні даних для Azure Cosmos DB дуже важливо, щоб вибраний ключ секції видавав рівномірний розподіл даних та запитів між фізичними секціями у контейнері. Це особливо важливо при розширенні контейнерів та збільшенні числа фізичних секцій.

Якщо не тестувати проект бази даних під навантаженням під час розробки, факт невдалого вибору ключа секції ***може залишатися непоміченим*** до тих пір, поки програма не опиниться в робочому середовищі і не буде записано багато даних.

Якщо дані не секційовані належним чином, це може спричинити створення гарячих розділів. Гарячі секції запобігають масштабуванню робочого навантаження програми та можуть виникати як для сховища, так і для пропускної спроможності.

Гарячі секції сховища

Гаряче секціонування у сховищі відбувається за наявності ключа секції, який призводить до високоасиметричних шаблонів зберігання. Як приклад розглянемо багатоклієнтську програму, яка використовує TenantId як ключ секції з п'ятьма клієнтами: від A до F. Клієнти B, C, D та E є дуже маленькими, а клієнт D містить трохи більше даних. Клієнт A росте і швидко досягає межі 20 ГБ для своєї секції. В цьому випадку нам потрібен інший ключ секції, який розподілятиме обсяг сховища по кількох логічних секціях.

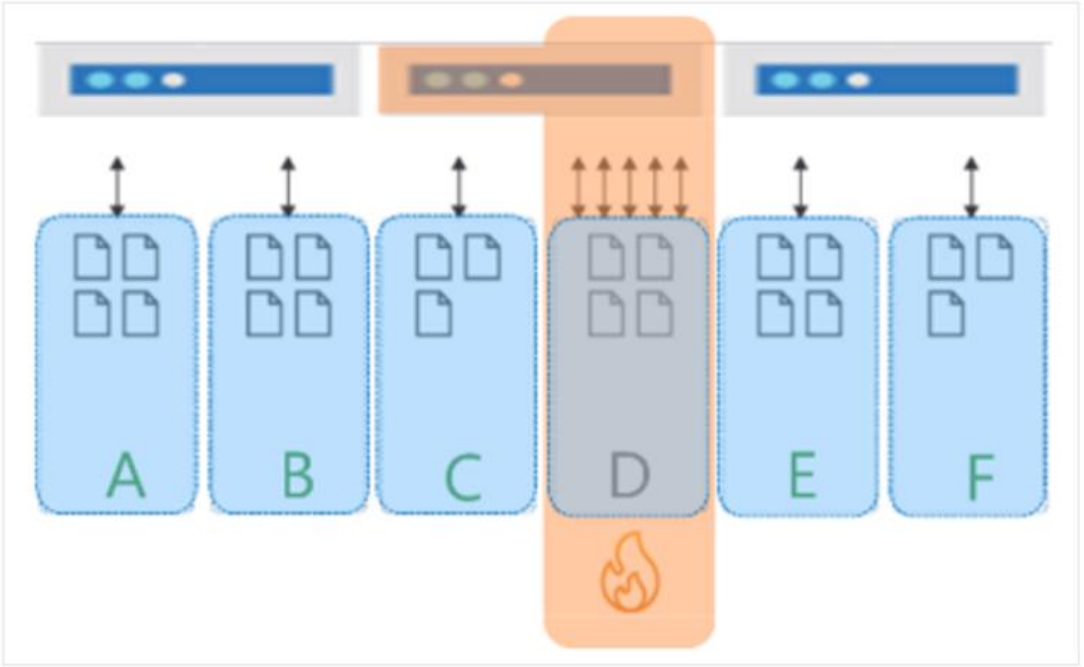


Гаряче секціонування пропускної спроможності

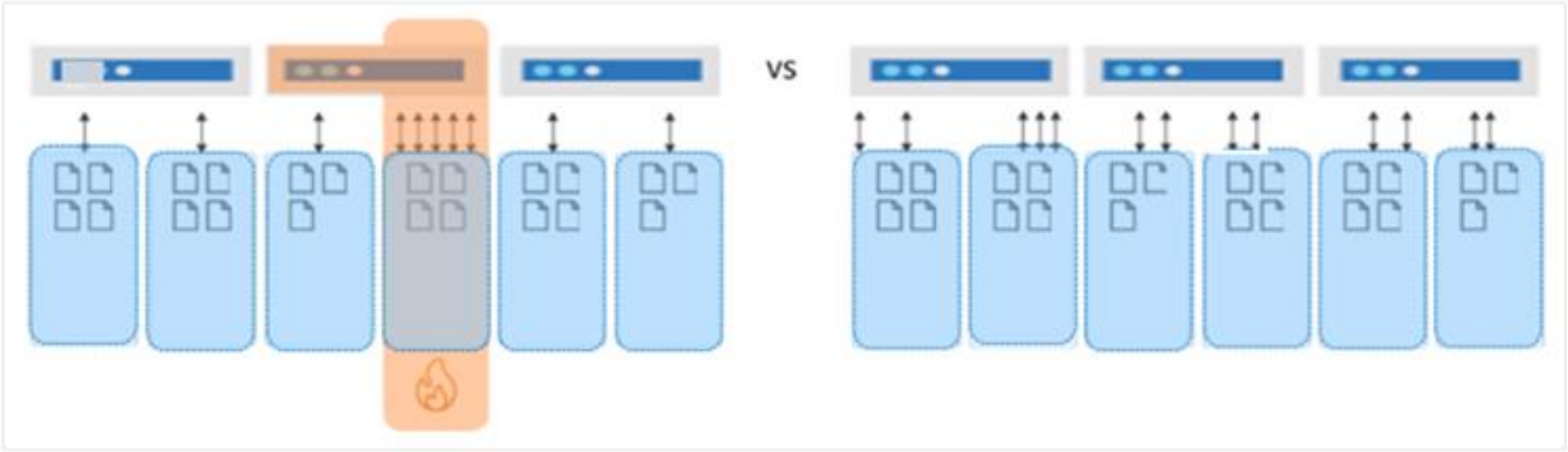
Якщо більшість або всі запити надсилаються на той самий логічний розділ, пропускна здатність може постраждати від гарячих розділів.

Важливо розуміти шаблони доступу програми, щоб забезпечити рівномірне розподілення запитів за значеннями ключа секції. Коли пропускна здатність готується для контейнера Azure Cosmos DB, вона розподіляється рівномірно по всіх фізичних секціях в контейнері.

Наприклад, якщо у вас є контейнер з 30000 ЕЗ/с, робоче навантаження буде розподіллено між трьома фізичними секціями для тих самих шести клієнтів вище. Таким чином, кожна фізична секція отримує 10000 ЕС/с. Якщо клієнт D споживає всі 10 000 ЕЗ/с, його частота запитів буде обмежена, оскільки він може користуватися пропускної спроможністю, виділеної інших секцій. Це призводить до зниження продуктивності клієнтів C і D та невикористаної ємності обчислень в інших фізичних секціях та клієнтах, що залишилися. Зрештою, цей ключ секції призводить до створення структури бази даних, в якій робоче навантаження програми не може масштабуватися.



Якщо дані та запити розподіляються рівномірно, це гарантує, що зі зростанням бази даних буде забезпечено повне використання обсягу сховища та пропускної спроможності. Результатом буде максимально можлива продуктивність та підвищена ефективність. Загалом структура бази даних масштабуватиметься.

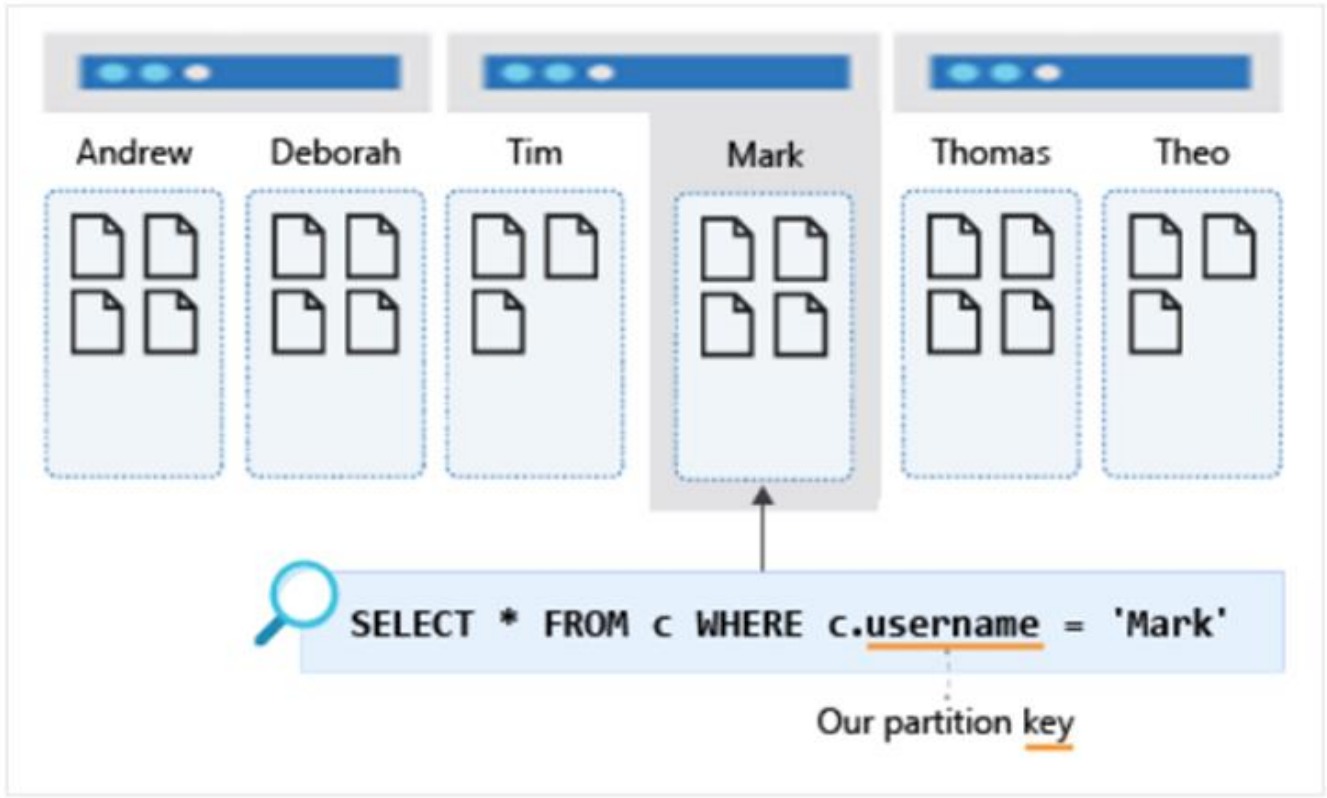


Читання порівняно із записом

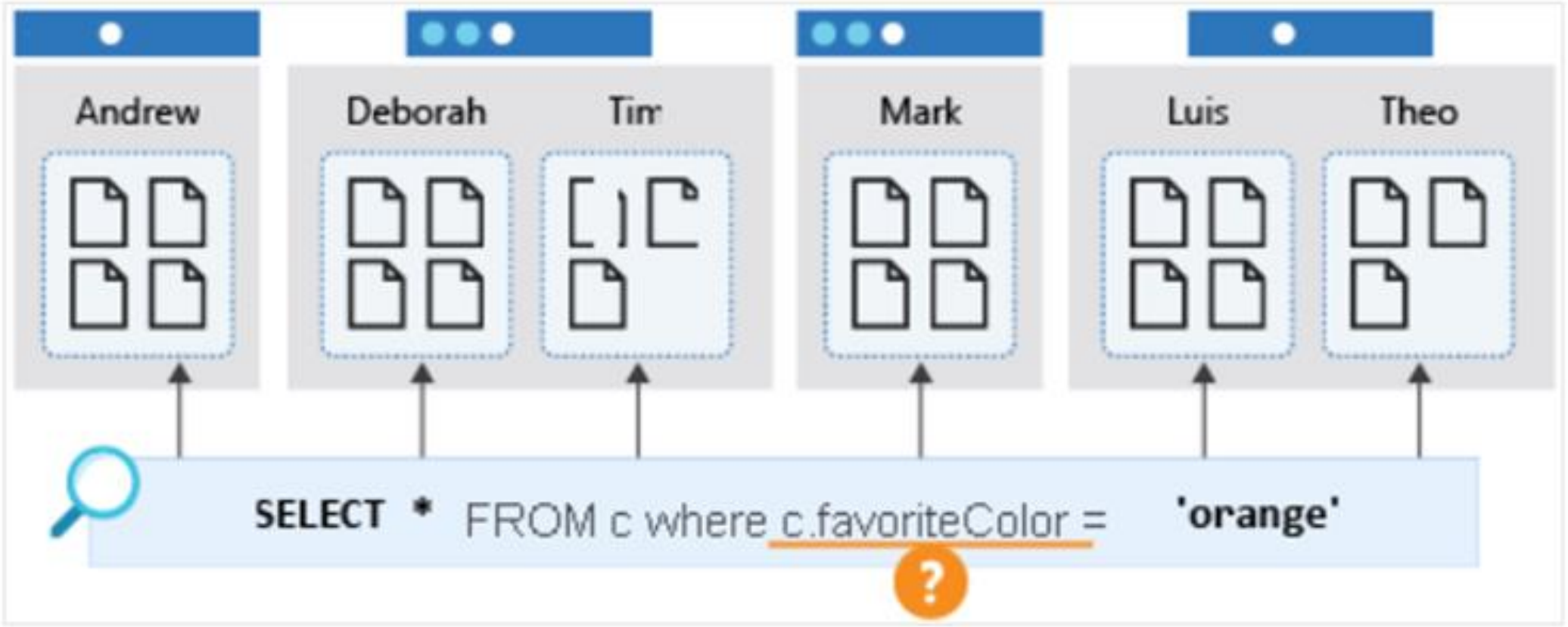
При виборі ключа секції необхідно враховувати, яка операція над даними виконується особливо інтенсивно читання чи запис. Для запитів із великим навантаженням операцій запису слід використовувати ключ секції з великою кратністю.

Для робочих навантажень з інтенсивними операціями читання слід переконатися, що запити обробляються однією фізичною секцією або невеликою кількістю фізичних секцій.

На наступному малюнку показаний контейнер, секціонований на ім'я користувача. Цей запит працює лише з єдиним логічним розділом, тому його продуктивність завжди буде оптимальною.

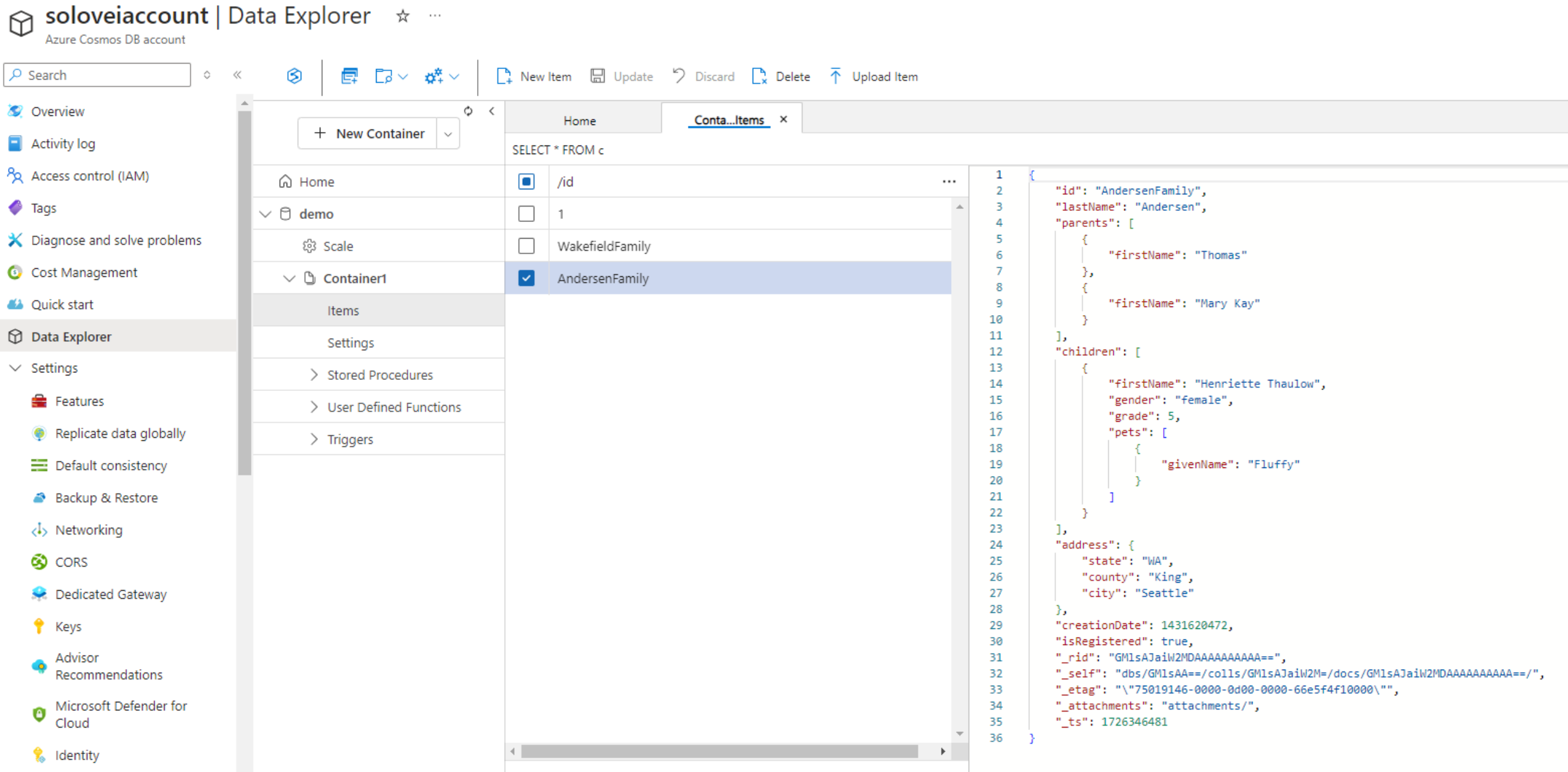


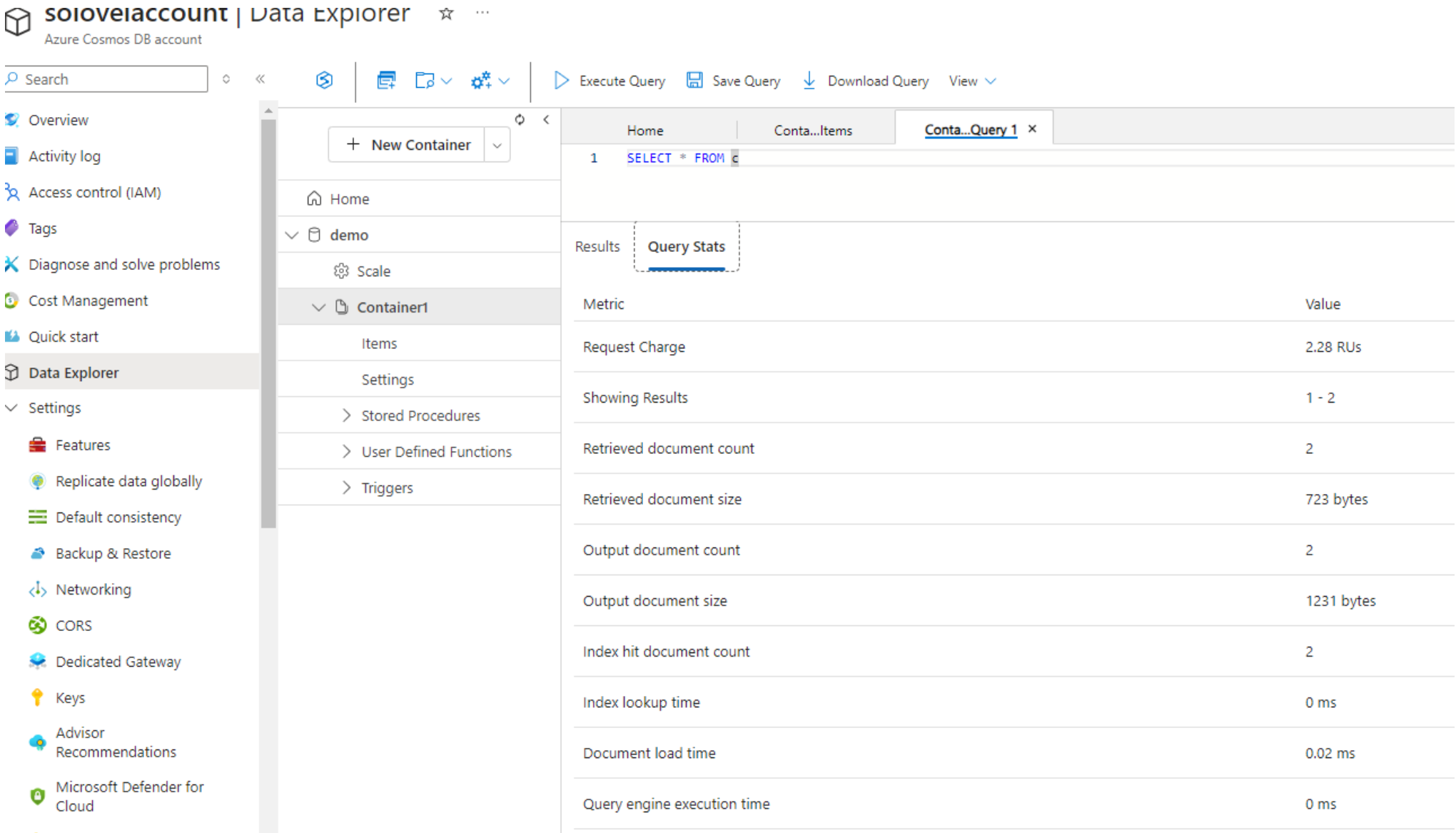
Запит, який фільтрує за іншою властивістю, наприклад favoriteColor, викликає "розкид" по всіх секціях у контейнері. Це також називається міжсекційним запитом. Такий запит працюватиме нормально, якщо контейнер невеликий і займає лише одну секцію. Однак у міру зростання контейнера та збільшення числа фізичних секцій цей запит буде виконуватися повільніше і стане дорожче, оскільки йому потрібно перевірити кожну секцію, щоб отримати результати, незалежно від того, чи містяться у фізичній секції пов'язані із запитом дані.



"limit total amount throughput" (обмеження загальної пропускної здатності) стосується кількості ресурсів, які база даних може використовувати для обробки запитів та операцій. -
Якщо запити перевищують доступну пропускну здатність (ліміт RU/s), запити можуть бути уповільнені або навіть відхилені з помилками обмеження (429 Too Many Requests). Це може призвести до зниження продуктивності системи та збільшення затримок.

Створити базу даних, додати коллекцію документів




soloveiaccount | Data Explorer
Azure Cosmos DB account
Search
Execute Query
Save Query
Download Query
View
Overview
Activity log
Access control (IAM)
Tags
Diagnose and solve problems
Cost Management
Quick start
Data Explorer
Settings
Features
Replicate data globally
Default consistency
Backup & Restore
Networking
CORS
Dedicated Gateway
Keys
Advisor Recommendations
Microsoft Defender for Cloud
New Container
Home
demo
Scale
Container1
Items
Settings
Stored Procedures
User Defined Functions
Triggers
Home
Conta...Items
Conta...Query 1
1 SELECT * FROM c
Results
Query Stats
Metric
Value
Request Charge
2.28 RUs
Showing Results
1 - 2
Retrieved document count
2
Retrieved document size
723 bytes
Output document count
2
Output document size
1231 bytes
Index hit document count
2
Index lookup time
0 ms
Document load time
0.02 ms
Query engine execution time
0 ms

```
SELECT c.givenName
FROM Families f
JOIN c IN
f.children
WHERE f.id =
'WakefieldFamily'
```

```
1  SELECT c.givenName
2  FROM Families f
3  JOIN c IN f.children
4  WHERE f.id = 'WakefieldFamily'
```

Results | **Query Stats**

Metric	Value
Request Charge	3.02 RUs
Showing Results	1 - 2
Retrieved document count	1
Retrieved document size	376 bytes
Output document count	2
Output document size	91 bytes
Index hit document count	1
Index lookup time	0.34 ms
Document load time	0.02 ms
Query engine execution time	0.03000000000000002 ms

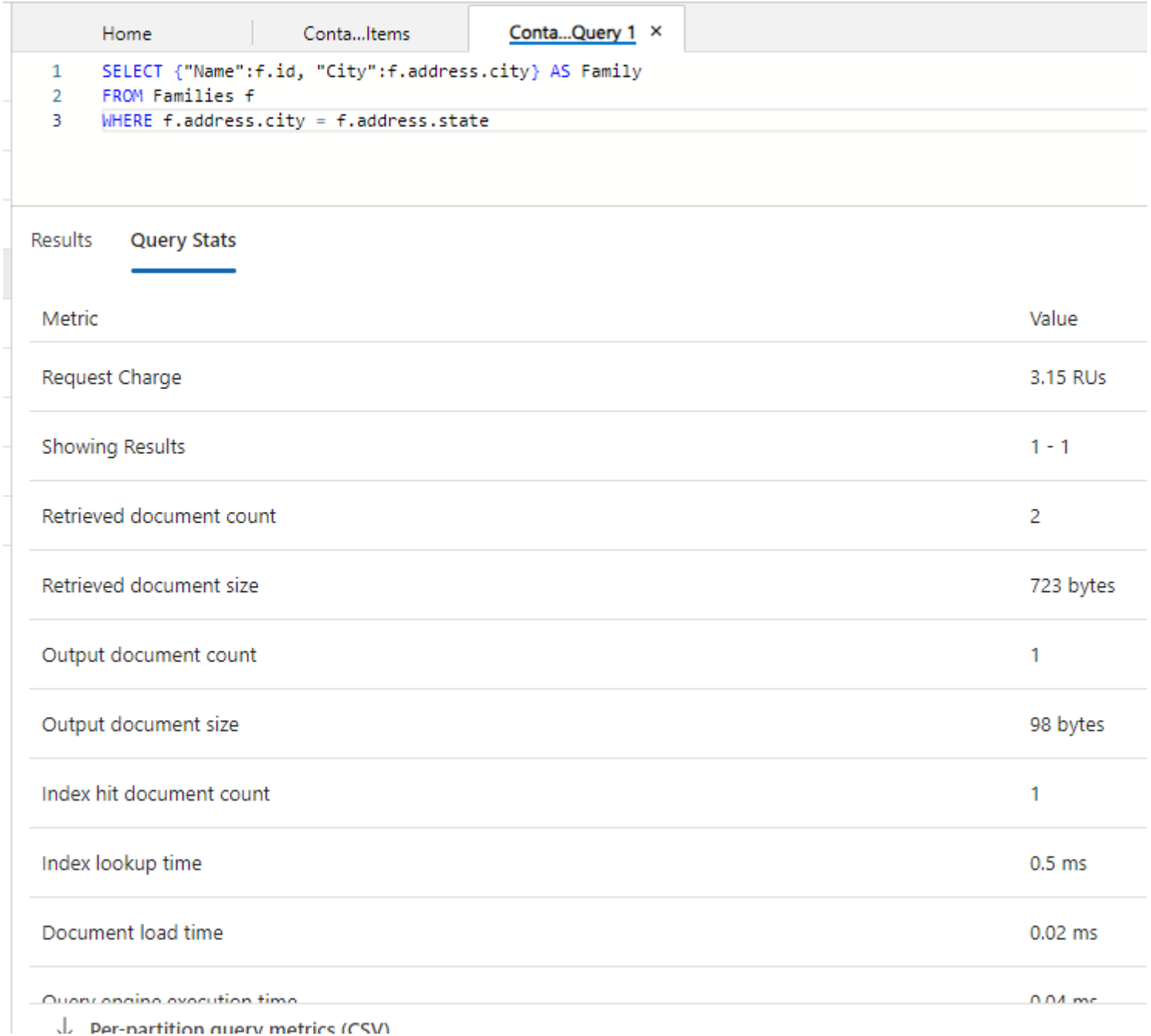
Per-partition query metrics (CSV)

Приклад №3 Запит на створення нового документу з фільтром where

SELECT {"Name":f.id, "City":f.address.city} AS Family

FROM Families f

WHERE f.address.city = f.address.state



Клієнська бібліотека API SQL Azure Cosmos DB для Python

Пакет SDK API SQL Azure Cosmos DB для Python включає функції:

1. Створення баз даних Cosmos DB і зміна їх параметрів.
2. Створення та зміна контейнерів для зберігання колекцій документів JSON.
3. Створення, читання, оновлення та видалення елементів (документів JSON) у контейнерах.
4. Запит документів у базі даних за допомогою синтаксису, подібного SQL.

Для роботи необхідно встановити пакет pip install azure-cosmos

Аутентифікація клієнта

Взаємодія з Cosmos DB починається з екземпляра класу CosmosClient. Для створення екземпляра об'єкта клієнту потрібен обліковий запис та відповідно заповнити змінні середовища в файлі конфігурації.

Провести перевірку клієнта можливо якщо передавати облікові дані в ClientSecretCredential або використовувати DefaultAzureCredential:

```
from azure.cosmos import CosmosClient
from azure.identity import ClientSecretCredential, DefaultAzureCredential

import os
url = os.environ['ACCOUNT_URI']
tenant_id = os.environ['TENANT_ID']
client_id = os.environ['CLIENT_ID']
client_secret = os.environ['CLIENT_SECRET']

# Using ClientSecretCredential
aad_credentials = ClientSecretCredential(
    tenant_id=tenant_id,
    client_id=client_id,
    client_secret=client_secret)

# Using DefaultAzureCredential (recommended)
aad_credentials = DefaultAzureCredential()

client = CosmosClient(url, aad_credentials)
```

Створення бази даних

Після перевірки CosmosClient можна працювати з будь-яким ресурсом в облікового запису. Наведений нижче фрагмент коду створює базу даних API SQL, яка використовується за умовчанням, якщо при виклику create_database не вказано API.

```
from azure.cosmos import CosmosClient, exceptions
import os

URL = os.environ['ACCOUNT_URI']
KEY = os.environ['ACCOUNT_KEY']
client = CosmosClient(URL, credential=KEY)
DATABASE_NAME = 'testDatabase'
try:
    database = client.create_database(DATABASE_NAME)
except exceptions.CosmosResourceExistsError:
    database = client.get_database_client(DATABASE_NAME)
```

Створення контейнера

Для створення контейнера з параметрами за замовчуванням необхідно скористатись методом container = database.create_container(id=CONTAINER_NAME, partition_key=PartitionKey(path="/productName")).

Якщо контейнер з таким же ім'ям вже існує в базі даних (що створює помилку 409 Conflict), замість цього отримується існуючий контейнер.

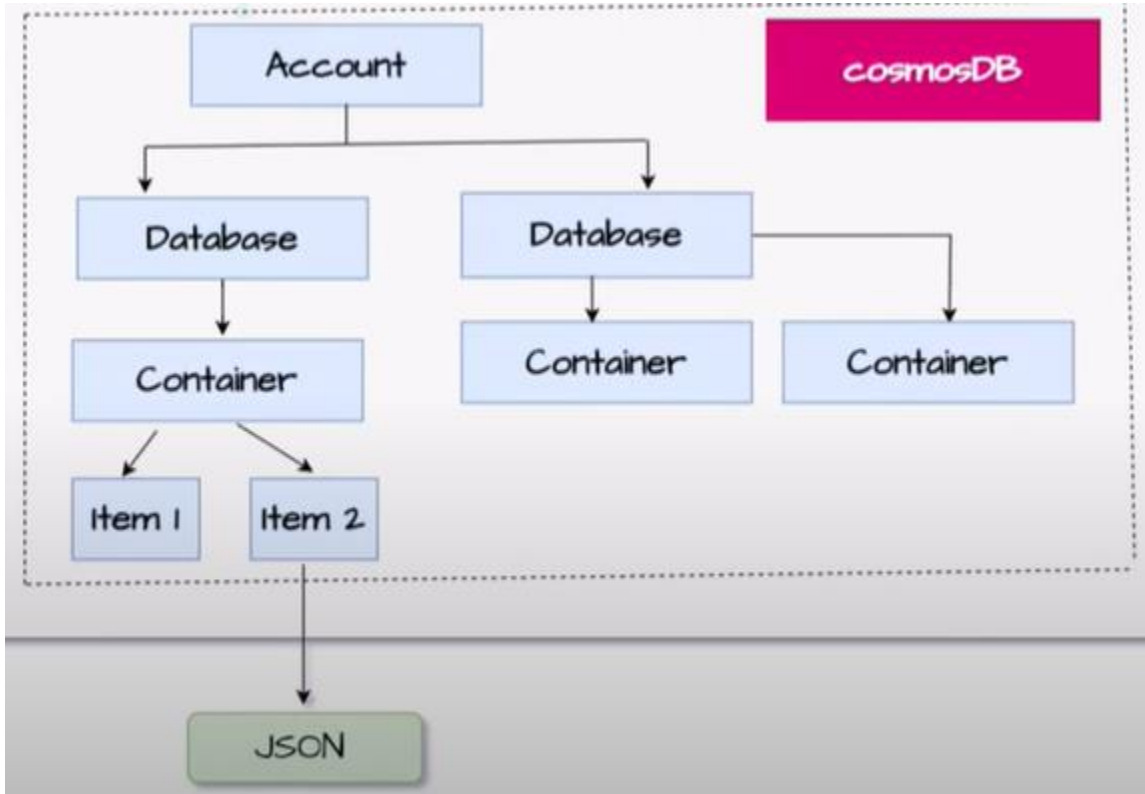
```
from azure.cosmos import CosmosClient, PartitionKey, exceptions
import os

URL = os.environ['ACCOUNT_URI']
KEY = os.environ['ACCOUNT_KEY']
client = CosmosClient(URL, credential=KEY)
DATABASE_NAME = 'testDatabase'
database = client.get_database_client(DATABASE_NAME)
CONTAINER_NAME = 'products'

try:
    container = database.create_container(id=CONTAINER_NAME, partition_key=PartitionKey(pat
except exceptions.CosmosResourceExistsError:
    container = database.get_container_client(CONTAINER_NAME)
except exceptions.CosmosHttpResponseError:
    raise
```

Cosmos DB Emulator

https://learn.microsoft.com/en-us/azure/cosmos-db/emulator

Емулятор забезпечує середовище у робочому просторі розробника.
Ключові відмінності у функціональності між емулятором і еквівалентною хмарною службою:

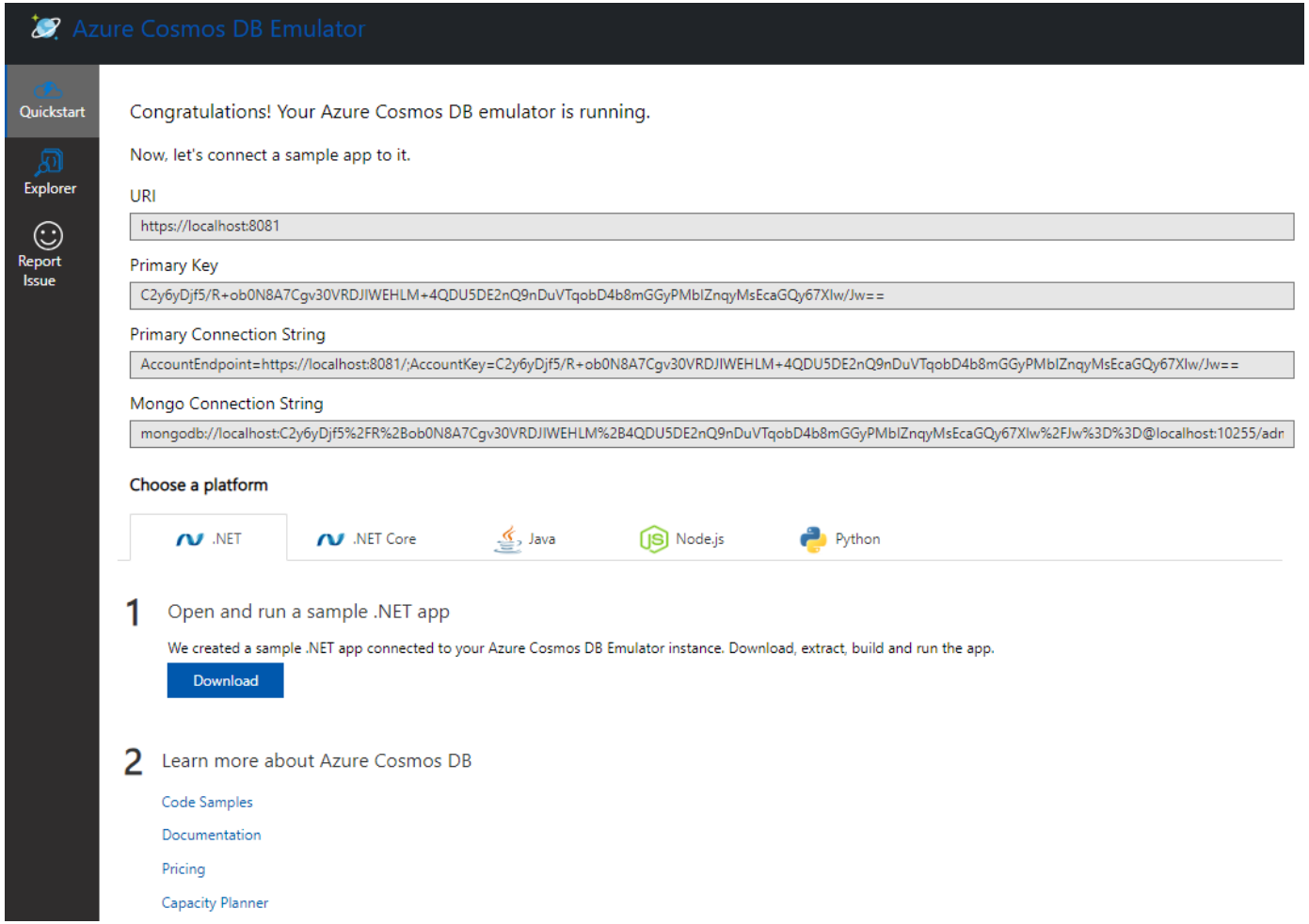
- Емулятор підтримує лише надану пропускну здатність. Емулятор не підтримує безсерверну пропускну здатність, тобто створюють модель розгортання бази даних, яка НЕ дозволяє обробляти запити без попереднього виділення фіксованих ресурсів для пропускної здатності.
- Під час запуску емулятор використовує добре відомий ключ. Ви не можете повторно згенерувати ключ для запущеного емулятора. Щоб використовувати інший ключ, потрібно запустити емулятор із указаним власним ключем.
- Емулятор не підтримує «надлишковість» даних
- Емулятор ідеально підтримує до 10 контейнерів фіксованого розміру зі швидкістю 400 RU/c або 5 контейнерів необмеженого розміру.
- Емулятор обмежує довжину унікального ідентифікатора елементів до 254 символів.
- Емулятор підтримує максимум п’ять операторів JOIN на запит.



Azure Cosmos DB emulator

https://learn.microsoft.com/en-us/azure/cosmos-db/how-to-develop-emulator?tabs=windows%2Cpython&pivots=api-nosql

cd C:\Program Files\Azure Cosmos DB Emulator
Microsoft.Azure.Cosmos.Emulator /port=8081



Клієнська бібліотека API SQL Azure Cosmos DB для Python

Пакет SDK API SQL Azure Cosmos DB для Python включає функції:

1. Створення баз даних Cosmos DB і зміна їх параметрів.
2. Створення та зміна контейнерів для зберігання колекцій документів JSON.
3. Створення, читання, оновлення та видалення елементів (документів JSON) у контейнерах.
4. Запит документів у базі даних за допомогою синтаксису, подібного SQL.

Для роботи необхідно встановити пакет pip install azure-cosmos

Аутентифікація клієнта

Взаємодія з Cosmos DB починається з екземпляра класу CosmosClient. Для створення екземпляра об’єкта клієнту потрібен обліковий запис та відповідно заповнити змінні середовища в файлі конфігурації.

Провести перевірку клієнта можливо якщо передавати облікові дані в ClientSecretCredential або використовувати DefaultAzureCredential:

CosmosClient Class

CosmosClient(url: str, credential: TokenCredential | str | Dict[str, Any], consistency_level: str | None = None, **kwargs)

URL-адреса облікового запису Cosmos DB
credential - ключем облікового запису

Методи
create_database - Створіть нову базу даних із заданим ID (ім’ям).
create_database_if_not_exists - Створіть базу даних, якщо вона ще не існує.
delete_database - Видалити базу даних із вказаним ідентифікатором (ім’ям).
get_database_client – Отримати наявну базу даних (всі контейнери)
from_connection_string - Створіть екземпляр CosmosClient на основі Connection string
get_database_account - Отримати інформацію про обліковий запис бази даних.

Створити контейнер create_container(id, partition_key, indexing_policy=None, default_ttl=None, populate_query_metrics=None, offer_throughput=None, unique_key_policy=None, conflict_resolution_policy=None, **kwargs)

Параметри:
id - Вказує унікальний ідентифікатор контейнера всередині бази даних.
partition_key: Це обов'язковий параметр, який визначає, як дані будуть розподілятися між розділами (partitions) для горизонтального масштабування.
PartitionKey(path='/id', kind='Hash’) - розподіл даних по розділам (розділам) відбувається за допомогою хешування значень ключа
indexing_policy - керує індексуванням документів внутри розділів, де використовується ключ розділу для розбиття даних. Індексація — це механізм, який дозволяє швидше перейти і відфільтрувати дані за ключами або іншими атрибутами.
unique_key_policy - задання унікальних ключів у колекціях баз даних. Це означає, що в межах одного розділу не може бути двох документів з одиничними значеннями для полів, визначених як унікальних.
offer_throughput: Визначає кількість ресурсів, виділених для контейнера. Це може бути або кількість RU (Request Units) на секунду, або авто-масштабовані RU. Якщо не вказано, буде використовуватися параметри по замовчуванню для бази даних або автоматичне управління.
default_ttl: Визначає час життя (TTL — Time to Live) документів в контейнері в секундах. Після цього часу документ буде автоматично видалений.
Якщо значення встановлено в 0, то TTL вимкнено. Якщо вказати значення >0, документи автоматично видаляються через зазначений час після їх створення.

Властивості контейнера

etag — це унікальний ідентифікатор (строка), який автоматично генерується та присвоюється кожному документу у Cosmos DB при його створенні або зміні. Кожного разу, коли документ оновлюється, його etag також оновлюється, що робить його своєрідною "відміткою" версії документа.

Основні особливості etag:
Контроль версій:
Кожна зміна документа призводить до зміни його etag. Це дозволяє відслідковувати, чи був документ змінений між запитами на читання і запис.

etag дозволяє запобігати конфліктам при одночасних оновленнях документа. Перед оновленням або видаленням документа можна перевірити його поточне значення etag, щоб переконатися, що ніхто інший не змінив його після того, як ви його прочитали.
Якщо документ змінено іншим процесом, то значення etag зміниться, і ваш запит на оновлення або видалення може бути відхилений, щоб уникнути конфлікту.

enable_cross_partition_query

За замовчуванням: Запити в Cosmos DB обмежуються одним розділом, якщо параметр enable_cross_partition_query не встановлений в True. Це покращує продуктивність і зменшує затримки для запитів, оскільки обробка даних відбувається лише в межах одного розділу.

Перетин кількох розділів: Якщо ваш запит має охоплювати більше одного розділу (наприклад, для отримання даних без фільтрації за partition key або для агрегації даних по кількох розділах), ви повинні встановити параметр enable_cross_partition_query=True. Це дозволяє запиту виконуватися по кількох розділах, але може збільшити затрати на запит (по RU/s) та час виконання.

Коли використовувати enable_cross_partition_query:
Запити без фільтрації за partition key: Якщо ви не вказуєте значення для ключа розділу і ваші дані зберігаються в різних розділах, вам необхідно встановити цей параметр для отримання даних з усіх розділів.
Агрегації: Запити, що вимагають виконання агрегацій (наприклад, COUNT, SUM, AVG) часто потребують доступу до кількох розділів.
Великі обсяги даних: Якщо ваші дані розподілені по багатьох розділах, і вам потрібно виконати запит, який охоплює більше одного розділу.
Мінуси використання:
Вищі затрати RU: Крос-розділові запити часто вимагають більше ресурсів (Request Units), оскільки вони охоплюють кілька фізичних розділів для виконання запиту.
Збільшений час виконання: Запити, що виконуються по кількох розділах, можуть займати більше часу, ніж запити по одному розділу, через те, що потрібно звертатись до кількох частин бази даних.

upsert_item(body, populate_query_metrics=None, pre_trigger_include=None, post_trigger_include=None, **kwargs)

body: - документ, якмй треба додати або оновити
partition_key (опціональний): Це ключ партиції, який використовується для визначення розташування документа в кластері даних.
disable_automatic_id_generation (опціональний, за замовчуванням False): Цей параметр вказує, чи потрібно автоматично генерувати значення для поля id, якщо воно відсутнє у переданому документі. Якщо встановити в True, функція не генеруватиме id автоматично, і документ повинен мати явно вказане поле id.