Посібник з RavenDB

Документно-орієнтована, транзакційна NoSQL СУБД з повними ACID-гарантіями

RavenDB — документно-орієнтована, транзакційна NoSQL СУБД з **повними ACID-гарантіями** на рівні однієї бази/транзакції. Вона поєднує гнучкість JSON-документів із продуктивністю індексів і простотою розробки:

- **Архітектура**: один або кілька вузлів (кластер), вбудований HTTP(S) сервер, зручна **Studio** (веб-консоль) для адміністрування, запитів і моніторингу.
- **Деплой**: локально (Windows/Linux/macOS), **Docker/Kubernetes**, хмара. В Dev-режимі стартує без складних налаштувань; у Prod — TLS/сертифікати, резервні копії, моніторинг.
- **Модель даних**: документи JSON у **колекціях**, без жорсткої схеми. Посилання між документами — через зберігання id (**reference**), а не **foreign keys** у класичному розумінні.
- **Потужні індекси**: статичні/автоматичні, **Map** і **Map-Reduce** (інкрементальні агрегати з оновленням у фоні). Підтримка LoadDocument у індексах для “join-подібних” проєкцій.
- **Запити**: **RQL** (SQL-подібний синтаксис) поверх індексів, повнотекстовий пошук, **fuzzy, boosting, spatial, attachments/counters/time series.**
- **Транзакційна модель**: сесія (**Unit of Work**) з локальною транзакцією на SaveChanges(). За потреби — **cluster-wide transactions** із Compare-Exchange.
- **Масштабування і доступність**: реплікація (**push/pull, hub/sink**), **шардінг** (рівномірний розподіл даних), автоматичне відновлення індексів, конфлікт-резолюція.
- **Інтеграції**: клієнтські SDK для **.NET/Java/Node/Python**, **ETL** у SQL/OLAP/Raven, **Subscriptions** для event-driven обробки, **Revisions** для аудиту/відкату.
- **Коли обирати RavenDB**:

☐ складні читальні запити з проєкціями/агрегаціями, що мають кешуватись/інкрементально оновлюватись;

☐ доменні моделі з гнучкою схемою (еволюція структури без міграцій);

☐ транзакційні сценарії (ACID) у NoSQL;

☐ edge/branch-сценарії (**pull-replication hub↔️sink**), офлайн-режими, георозподіл.

- **Чого очікувати**: дизайн під індекси (думати “які запити я виконуватиму?”), вибір ключів/шард-ключів, контроль довгих патчів та індексації у продакшені.

Архітектура та основні можливості

Архітектура

Один або кілька вузлів (кластер), вбудований HTTP(S) сервер, зручна **Studio** (веб-консоль) для адміністрування, запитів і моніторингу.

Деплой

Локально (Windows/Linux/macOS), **Docker/Kubernetes**, хмара. В Dev-режимі стартує без складних налаштувань; у Prod — TLS/сертифікати, резервні копії, моніторинг.

Модель даних

Документи JSON у **колекціях**, без жорсткої схеми. Посилання між документами — через зберігання id (reference), а не foreign keys у класичному розумінні.

Потужні індекси та запити

- **Статичні/автоматичні індекси**, Map і Map-Reduce (інкрементальні агрегати з оновленням у фоні). Підтримка LoadDocument у індексах для "join-подібних" проєкцій.
- **RQL** (SQL-подібний синтаксис) поверх індексів, повнотекстовий пошук, fuzzy, boosting, spatial, attachments/counters/time series.

Транзакційна модель та масштабування

- Сесія (**Unit of Work**) з локальною транзакцією на SaveChanges(). За потреби — **cluster-wide transactions** із Compare-Exchange.
- Реплікація (push/pull, hub/sink), **шардінг** (рівномірний розподіл даних), автоматичне відновлення індексів, конфлікт-резолюція.

Коли обирати **RavenDB** та інтеграції

1 Складні читальні запити

Складні читальні запити з проєкціями/агрегаціями, що мають кешуватись/інкрементально оновлюватись.

2 Гнучка схема

Доменні моделі з гнучкою схемою (еволюція структури без міграцій).

3 Транзакційні сценарії

Транзакційні сценарії (ACID) у NoSQL.

4 **Edge** та офлайн-режими

Edge/branch-сценарії (pull-replication hub↔sink), офлайн-режими, георозподіл.

Інтеграції та очікування

Клієнтські SDK для .NET/Java/Node/Python, **ETL** у SQL/OLAP/Raven, **Subscriptions** для event-driven обробки, **Revisions** для аудиту/відкату.

Чого очікувати: дизайн під індекси (думати "які запити я виконуватиму?"), вибір ключів/шард-ключів, контроль довгих патчів та індексації у продакшені.

Основні концепції RavenDB

Document (документ)

Самодостатній JSON-об'єкт з бізнес-даними. Зберігається під унікальним **id**: artists/1-A, orders/2025-000123. Документ може містити вкладені об'єкти/масиви, **не потребує схеми**. Еволюція — шляхом додавання/видалення полів у нових версіях застосунку. Рекомендація: зберігати агрегати як 1 документ (DDD aggregate), щоб мінімізувати крос-документні транзакції.

Collection (колекція)

Логічна група документів (визначається сервером за полем системної метадані @collection). Приклади: Artists, Events, Orders. Індекси, правила ревізій, політики безпеки часто задаються **на рівні колекції**.

ID (ключ)

Рядок, який можна задати вручну або доручити RavenDB. За замовчуванням .NET-клієнт використовує **HiLo**-генератор (пакетами видає діапазони ідентифікаторів для мінімізації мережевих викликів). Патерн іменування допомагає шардінгу й читабельності:

- Префікс колекції: artists/, events/
- Людиночитні слаґи: artists/daft-punk

Індекси та RQL

Index (індекс)

Автоматичні створюються RavenDB під запити; **статичні** — описуються розробником для контролю мап/ред'юс і проєкцій. Індекс — це **персистентне уявлення** документів, що підтримується у фоні, тому читання швидке, а запис тригерить асинхронну індексацію.

Map

Прості проєкції: вибрати поля/трансформації, "розплескати" масиви, підготувати дані для пошуку. Може використовувати **LoadDocument** (у статичних індексах) для звернення до пов'язаних документів — аналог "join" на етапі індексації.

Map-Reduce

Агрегації з **групуванням** (sum, count, min/max, custom) та **інкрементальним оновленням**. Підходить для метрик "по містах/датах/категоріях", топ-N, дашбордів, де повторні підрахунки з нуля були б дорогими.

RQL (Raven Query Language)

SQL-подібний синтаксис поверх індексів: from, where, order by, group by, load, select, update/delete (patch). Підтримує **повнотекстовий пошук**, fuzzy, boosting, spatial, include (зменшує додаткові звернення в сесії). Запити виконуються **проти індексів** (авто/статичних). Якщо відповідного індексу нема — автоіндекс створиться на льоту.

Session, Attachments, Counters та Revisions

Session (сесія)

Unit of Work на стороні клієнта: трекінг завантажених об'єктів, **identity map**, відкладення змін до SaveChanges(). Вбудоване **кешування** в межах сесії, Includes зменшують кількість запитів. Паралельні оновлення контролюються через **ETag** (optimistic concurrency).

Attachments (вкладення)

Бінарні файли (фото, PDF) прив'язуються до документа **без зберігання в самому JSON**. Індексами напряму не індексуються, але метадані/імена можна використовувати в логіці.

Counters / Time Series

Counters — числові лічильники (перегляди, лайки), підтримують інкремент/декремент із мінімальною конкуренцією. **Time Series** — часові ряди (телеметрія, заміри), зберігаються поряд із документом, мають ефективне зберігання та запити по інтервалах/агрегатах.

Revisions (версіонування)

Прозоме зберігання історії змін документів (**audit trail**, відкат). Налаштовується політикою на рівні бази/колекції (вік, кількість, включати/виключати вкладення).

Compare-Exchange (cluster-wide)

Кластерна **узгоджена** ключ-значення колекція. Використовується для **унікальностей** (email/slug), **лідер-елекшн**, глобальних прапорів. Базується на Raft і бере участь у **кластерних транзакціях**.

Встановлення та запуск **RavenDB**

Варіант **1: Docker** (рекомендовано для розробки)

Швидкий старт (Dev-режим)

```
docker run -d --name ravendb -p 8080:8080 -p 38888:38888 \
  -e RAVEN_Setup_Mode=Dev \
  -e RAVEN_License_Eula_Accepted=true \
  -v ravendb-data:/opt/RavenDB/Server/RavenData \
  ravendb/ravendb:latest
```

8080 — HTTP/Studio; 38888 — TCP для кластера/реплікацій. Дані в volume ravendb-data. Відкрий Studio: **http://localhost:8080** (Dev-сертифікат, спрощена безпека).

Docker Compose (зручно для проєкту)

```
version: "3.8"
services:
  ravendb:
    image: ravendb/ravendb:latest
    container_name: ravendb
    ports:
      - "8080:8080"
      - "38888:38888"
    environment:
      RAVEN_Setup_Mode: "Dev"
      RAVEN_License_Eula_Accepted: "true"
    volumes:
      - ravendb-data:/opt/RavenDB/Server/RavenData
volumes:
  ravendb-data:
```

Запуск: docker compose up -d

Варіант 2: Windows/Linux (zip/installer) Windows

Варіант 2: Windows/Linux (zip/installer) Windows

Завантаж інсталятор або zip.

Встанови та запусти Raven.Server (служба або консоль).

Studio: http://localhost:8080.

Служба Windows вручну (zip-версія):

```
.\Raven.Server.exe --Install Start-Service RavenDB
```

Видалити: .\Raven.Server.exe --Uninstall

Linux

- Завантаж та розпакуй архів.
- Додай права виконання: chmod +x Raven.Server.
- Запусти: ./Raven.Server.
- Studio: http://localhost:8080.

Приклад systemd unit:

```
[Unit] Description=RavenDB Server After=network.target

[Service] WorkingDirectory=/opt/ravendb/Server ExecStart=/opt/ravendb/Server/Raven.Server Restart=always User=ravendb Group=ravendb LimitNOFILE=65535

[Install] WantedBy=multi-user.target
```

```
sudo systemctl daemon-reload sudo systemctl enable ravendb sudo systemctl start ravendb sudo systemctl status ravendb
```

Перше відкриття Studio

http://localhost:8080 → майстер налаштування (для Dev усе мінімальне вже готово).

> У Prod: обов’язково TLS/сертифікати, користувачі/ролі, бекапи.

Типові прапорці та Prod-налаштування

Типові прапорці/змінні середовища

- `RAVEN_Setup_Mode=Dev|Secured` — режим налаштування (Dev для локалки).
- `RAVEN_License_Eula_Accepted=true` — прийняти EULA (обов'язково).
- `RAVEN_Security_Certificate_Path=/certs/server.pfx` — (Prod, за потреби)
- `RAVEN_Security_Certificate_Password=***`
- `RAVEN_Security_UnsecuredAccessAllowed=None` — у продакшені не дозволяти.

Проброс томів з сертифікатами: `-v ./certs:/certs:ro`

Оновлення образу

```
docker pull ravendb/ravendb:latest
docker stop ravendb && docker rm ravendb
# повторити docker run з тим самим volume (дані збережуться)
```

Трейсинг/логи

```
docker logs -f ravendb
```

Prod-чеклист (коротко)

- TLS-сертифікат (PFX), закриті порти іззовні (відкривати лише потрібні)
- Регулярні резервні копії (Snapshots/Backups у S3/Azure/FTP/локально)
- Моніторинг індексів і довгих запитів
- Політика Revisions
- Тест відновлення з бекапу

CRUD-операції та практичні приклади

Через **Studio (GUI)**

- **Create**: **Databases → Documents → New Document** → встав JSON → **Save**.
  - Якщо не задав `id` — Raven присвоїть автоматично.
- **Read**: пошук за `id` або через **Query** (RQL) у вкладці **Query**.
- **Update**: відкрий документ, змінюй JSON → **Save**.
- **Delete**: кнопка **Delete** на документі.

Корисне у **Studio**

- **Query params**: справа є панель параметрів (для `where field = $p1`).
- **Includes**: у Query → `include` поля для зменшення дод. звернень у сесії.
- **Patch by query**: масові оновлення (уважно у продакшені!).
- **Streaming**: для дуже великих результатів (кнопка у вкладці Query).

Через **RQL (в Studio → Query)**

Створити (patch + put)

```
put artists/1-A {
  "display_name": "Daft Punk",
  "artist_type": "duo",
  "@collection": "Artists"
}
```

Прочитати

```
from Artists where artist_type = "duo"
order by display_name
select { id: id(), name: display_name }
```

Оновити (Patch)

```
from Artists as a where a.display_name = "Daft Punk"
update {
  this.tags = (this.tags || []);
  this.tags.push("electronic");
}
```

Видалити

```
from Artists where display_name = "Unknown Artist"
delete
```

Вкладення (Attachments) та Лічильники (Counters)

Attachments:

```
using var session = store.OpenSession();
var artist = session.Load("artists/2-A");
using var file = File.OpenRead("press-photo.jpg");
session.Advanced.Attachments.Store(artist, "press-photo.jpg", file, "image/jpeg");
session.SaveChanges();
```

Counters:

```
session.CountersFor("events/1-A").Increment("views", 1);
session.SaveChanges();
```

Через .NET клієнт (C#)

Підключення

```
using Raven.Client.Documents;

var store = new DocumentStore
{
    Urls = new[] { "http://localhost:8080" },
    Database = "MusicFest"
}.Initialize();
```

Базовий CRUD

```
using (var session = store.OpenSession())
{
    // CREATE
    var artist = new Artist { DisplayName = "Daft Punk", ArtistType = "duo" };
    session.Store(artist);          // id з'явиться після SaveChanges
    session.SaveChanges();

    // READ (Query)
    var dp = session.Query<Artist>()
                    .Where(x => x.ArtistType == "duo")
                    .OrderBy(x => x.DisplayName)
                    .ToList();

    // UPDATE (Load + Save)
    var a = session.Load<Artist>("artists/1-A");
    a.Tags ??= new();
    a.Tags.Add("electronic");
    session.SaveChanges();

    // DELETE
    session.Delete("artists/1-A");
    session.SaveChanges();
}
```

Async-варіант

```
using (var session = store.OpenAsyncSession())
{
    var artist = new Artist { DisplayName = "Justice", ArtistType = "duo" };
    await session.StoreAsync(artist);
    await session.SaveChangesAsync();

    var list = await session.Query<Artist>()
                        .Where(x => x.ArtistType == "duo")
                        .ToListAsync();
}
```

Вкладення (Attachments)

```
using var session = store.OpenSession();
var artist = session.Load<Artist>("artists/2-A");
using var file = File.OpenRead("press-photo.jpg");
session.Advanced.Attachments.Store(artist, "press-photo.jpg", file, "image/jpeg");
session.SaveChanges();
```

Лічильники (Counters)

```
session.CountersFor("events/1-A").Increment("views", 1);
session.SaveChanges();
```

Time Series (базовий приклад)

```
var ts = session.TimeSeriesFor("events/1-A", "temperature");
ts.Append(DateTime.UtcNow, 21.7);
session.SaveChanges();
```

Пара корисних фішок клієнта

- **Includes** (менше звернень до сервера):

```
var ev = session
    .Include<Event>(e => e.Venue)   // підвантажить venue у кеш сесії
    .Load<Event>("events/2025-opening");
var venue = session.Load<Venue>(ev.Venue); // без дод. мережевого запиту
```

- **Optimistic Concurrency**:

```
session.Advanced.UseOptimisticConcurrency = true;
var a = session.Load<Artist>("artists/1-A");
a.DisplayName = "Updated";
session.SaveChanges(); // кине ConcurrencyException, якщо ETag змінився
```

- **Очікування індексації (обережно!)**:

```
var fresh = session.Query<Artist>()
    .Customize(x => x.WaitForNonStaleResults(TimeSpan.FromSeconds(5)))
    .Where(x => x.ArtistType == "duo")
    .ToList();
```

Використовуй точково в тестах/адмін-операціях, а не в гарячому проді.

- **Bulk Insert** (масове завантаження):

```
using (var bulk = store.BulkInsert())
{
    for (int i = 0; i < 100_000; i++)
        await bulk.StoreAsync(new Artist { DisplayName = $"Artist #{i}" });
}
```

- **Streaming** (читання великих результатів):

```
using var stream = await session.Advanced.StreamAsync(
    session.Query<Artist>().Where(x => x.Country == "FR"));
while (await stream.MoveNextAsync())
{  var artist = stream.Current.Document;
    // обробка }
```

Практичний приклад: “Музичний фестиваль”

Модель колекцій

- **Artists**: display_name, artist_type (solo|band|duo), country, tags[].
- **Venues**: name, display_name, city, address{city, street}, capacity.
- **Events**: name, date, venue (reference id), artists[] (ref ids), status, price_tier.
- **Tickets**: eventId, amount, price, buyerId?.
- **Providers** (агенти/партнери): name, type, contacts[].

Створення документів (RQL)

```
put artists/daft-punk {
  "display_name": "Daft Punk",
  "artist_type": "duo",
  "country": "FR",
  "@collection": "Artists"
}

put venues/main-stage {
  "name": "Main Stage",
  "display_name": "Main Stage",
  "city": "Lviv",
  "address": { "city": "Lviv", "street": "Shevchenka 10" },
  "capacity": 15000,
  "@collection": "Venues"
}

put events/2025-opening {
  "name": "Opening Night",
  "date": "2025-07-12T19:00:00.0000000Z",
  "venue": "venues/main-stage",
  "artists": ["artists/daft-punk"],
  "status": "Scheduled",
  "price_tier": "A",
  "@collection": "Events"
}

put tickets/1-A {
  "eventId": "events/2025-opening",
  "amount": 2,
  "price": 80.0,
  "@collection": "Tickets"
}
```

Запити (RQL)

Список подій з назвою майданчика

```
from Events as e
load e.venue as v
select {
  id: id(e),
  event: e.name,
  date: e.date,
  venue: v.display_name ?? v.name,
  city: v.city
}
order by date
```

Пошук артистів за країною та тегами

```
from Artists
where country = "FR" and any(tags[], t => t = "electronic")
select { id: id(), name: display_name }
```

Агрегація продажів по містах (через завантаження Venue)

```
from Tickets as t
load t.eventId as ev, ev.venue as v
group by v.city
select { city: key(), sold: sum(t.amount), revenue: sum(t.amount * t.price) }
order by revenue desc
```

Map-Reduce індекс доходів по містах (C#)

```
public class Revenue_ByCity : AbstractIndexCreationTask<Ticket, Revenue_ByCity.Result>
{
    public class Result { public string City { get; set; } public int Sold { get; set; } public decimal
Revenue { get; set; } }

    public Revenue_ByCity()
    {
        Map = tickets => from t in tickets
                         let ev = LoadDocument<Event>(t.EventId)
                         let v = LoadDocument<Venue>(ev.Venue)
                         select new { City = v.City, Sold = t.Amount, Revenue = t.Amount * t.Price };

        Reduce = results => from r in results
                            group r by r.City into g
                            select new { City = g.Key, Sold = g.Sum(x => x.Sold), Revenue = g.Sum(x =>
x.Revenue) };
    }
}
```

Підписки (обробка покупок)

Наприклад, вести лог/перевіряти квоти:

```
var id = await store.Subscriptions.CreateAsync<Ticket>(options: new SubscriptionCreationOptions
{
    Name = "tickets-processor",
    Filter = t => t.Amount > 0
});

var worker = store.Subscriptions.GetSubscriptionWorker<Ticket>(id);
_ = worker.Run(async batch =>
{
    foreach (var item in batch.Items)
    {
        var t = item.Result;
        // бізнес-логіка: резерв, перевірка capacity, нотифікації тощо
    }
});
```

Глосарій **RavenDB** та висновки

RQL

Мова запитів Raven.

Session

Контекст транзакції для читання/запису.

Index

Структура для швидких запитів і проєкцій (Map / Map-Reduce).

Patch

Серверний скрипт-оновлення документів.

Attachments

Бінарні файли, прив'язані до документів.

Counters / Time Series

Числові лічильники / часові ряди, що зберігаються поряд з документом.

Revisions

Версіонування документів.

Subscriptions

Потік змін документів під умовою (event-driven processing).

External / Pull Replication

Види реплікації між базами/кластерами.

Sharding

Розподіл даних по шардах (горизонтальне масштабування).

Compare-Exchange

Кластерні ключі для унікальності/ синхронізації.

ETL

Обмін даними між Raven↔️Raven/SQL/OLAP.