

## Bazy Danych 2 – Lab 8

### Czym jest NoSQL?

Bazy danych NoSQL przechowują dane w inny sposób niż tradycyjne relacyjne tabele. Istnieje wiele rodzajów baz NoSQL, opartych na różnych modelach danych. Do głównych typów należą:

- bazy dokumentowe,
- klucz-wartość,
- kolumnowe,
- grafowe.

Zapewniają one elastyczne schematy i łatwo się skalują przy dużych ilościach danych i obciążeniu użytkownikami.

W tym artykule dowiesz się, czym są bazy danych NoSQL, dlaczego (i kiedy!) warto z nich korzystać oraz jak zacząć pracę z tym typem baz

### Czym jest baza danych NoSQL?

Termin „NoSQL” zwykle odnosi się do każdej bazy danych nie będącej relacyjną. Niektórzy tłumaczą to jako „non-SQL” (nie SQL), inni jako „not only SQL” (nie tylko SQL). Tak czy inaczej, chodzi o systemy przechowujące dane w sposób bardziej naturalny i elastyczny. NoSQL to podejście do zarządzania bazą danych, a nie tylko język zapytań jak SQL.

Bazy danych SQL są relacyjne, a bazy danych NoSQL są nierelacyjne. System zarządzania relacyjną bazą danych (RDBMS) jest podstawą strukturalnego języka zapytań (SQL), który umożliwia użytkownikom dostęp do danych i manipulowanie nimi w tabelach o wysokim stopniu strukturyzacji. To podstawowy model dla systemów baz danych, takich jak MS SQL Server, IBM DB2, Oracle i MySQL. W przypadku baz danych NoSQL składnia dostępu może się jednak różnić pomiędzy poszczególnymi bazami danych. Aby zrozumieć bazy danych NoSQL, należy znać różnicę pomiędzy RDBMS a nierelacyjnymi typami baz danych.

Dane w systemie RDBMS są przechowywane w obiektach bazy danych zwanych tabelami. Tabela to zbiór powiązanych wpisów danych, składająca się z kolumn i wierszy. Te bazy danych wymagają wcześniejszego zdefiniowania schematu – wszystkie kolumny i powiązane z nimi typy danych muszą być wcześniej znane, aby aplikacje mogły zapisywać dane w bazie danych. Przechowują także dane łącząc różne tabele przy użyciu kluczy, tworząc relacje między wieloma tabelami. W najprostszym przypadku klucz jest używany do pobrania określonego wiersza, umożliwiając jego sprawdzenie lub zmianę.

Z drugiej strony, w bazach danych NoSQL, dane mogą być przechowywane bez wcześniejszego zdefiniowania schematu – oznacza to możliwość szybkiego działania i iteracji dzięki definiowaniu modelu danych w trakcie pracy. Może to nadawać się dla określonych wymagań biznesowych, niezależnie od tego, czy baza danych jest oparta na grafach, kolumnach, dokumentach czy parach klucz-wartość.

Do niedawna relacyjne bazy danych były najczęściej używanym modelem. Nadal są one niezwykle wszechobecne w wielu firmach; jednak różnorodność, szybkość i ilość aktualnie używanych danych czasami wymaga zdecydowanie innego typu bazy danych do uzupełnienia relacyjnej bazy danych. Wywołało to przyjęcie w niektórych miejscach baz danych NoSQL, zwanych również „nierelacyjnymi bazami danych”. Ze względu na ich zdolność szybkiego skalowania w poziomie, nierelacyjne bazy danych mogą obsługiwać duże ilości ruchu, co czyni je wybitnie elastycznymi w wykorzystaniu.

## Rodzaje baz danych NoSQL

1. **Dokumentowe** – przechowują dane w formie dokumentów podobnych do JSON.

Przykłady: MongoDB, Couchbase.

```
{  
  "_id": "12345",  
  "name": "foo bar",  
  "email": "foo@bar.com",  
  "address": {  
    "street": "123 foo street",  
    "city": "some city",  
    "state": "some state",  
    "zip": "123456"  
  },  
  "hobbies": ["music", "guitar", "reading"]  
}
```

2. **Klucz-wartość** – prosta struktura składająca się z unikalnego klucza i jednej wartości.

Przykłady: Redis, Amazon DynamoDB.

```
Key: user:12345  
Value: {"name": "foo bar", "email": "foo@bar.com", "designation": "software developer"}
```

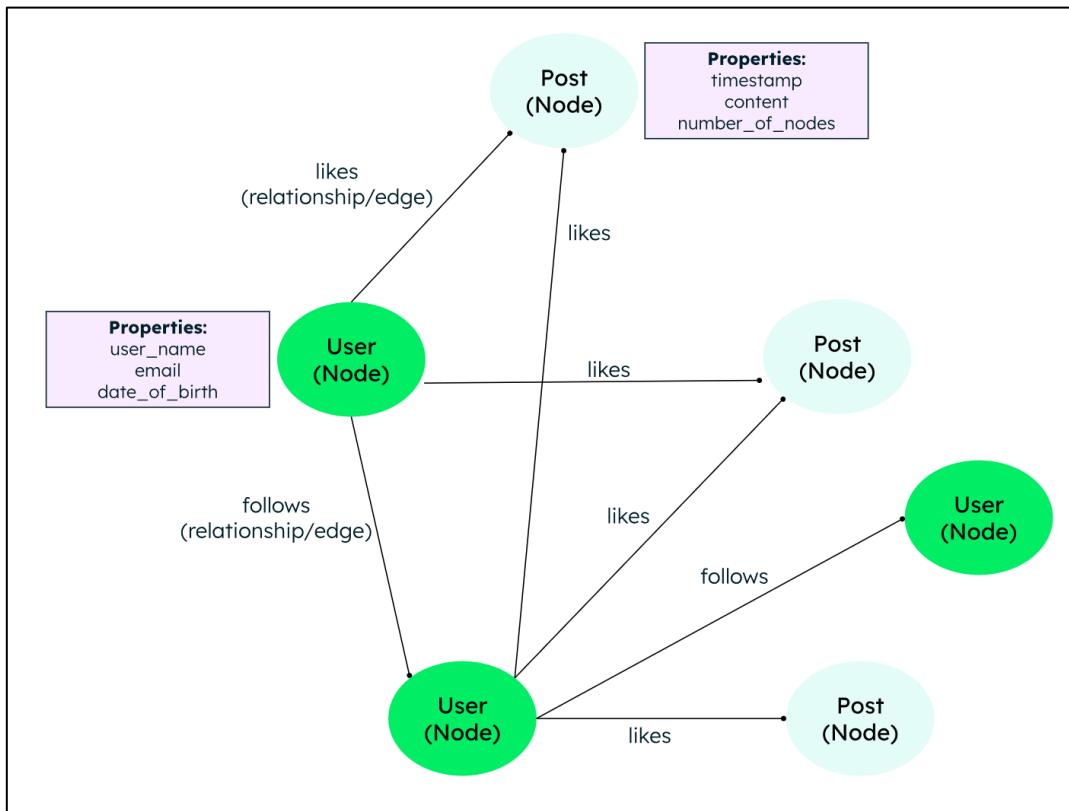
<https://www.mongodb.com/resources/basics/databases/key-value-database>

3. **Szerokokolumnowe** – dane przechowywane są w tabelach z dynamiczną liczbą kolumn.  
Przykłady: Apache Cassandra, HBase.

name	id	email	dob	city
Foo bar	12345	foo@bar.com		Some city
Carn Yale	34521	bar@foo.com	12-05-1972	

<https://www.scylladb.com/glossary/wide-column-database/>

4. **Grafowe** – przechowują dane jako węzły i krawędzie, idealne dla danych o dużym stopniu powiązań. Przykłady: Neo4j, Amazon Neptune.



<https://www.datageeks.pl/geeks-blog/10-grafowe-bazy-danych>

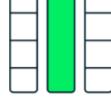
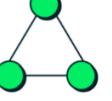
<https://www.oracle.com/pl/autonomous-database/what-is-graph-database/>

<https://neo4j.com/docs/getting-started/graph-database/>

5. **Wielomodelowe** – obsługują więcej niż jeden model danych (np. dokumenty i grafy).  
Przykłady: CosmosDB, ArangoDB.

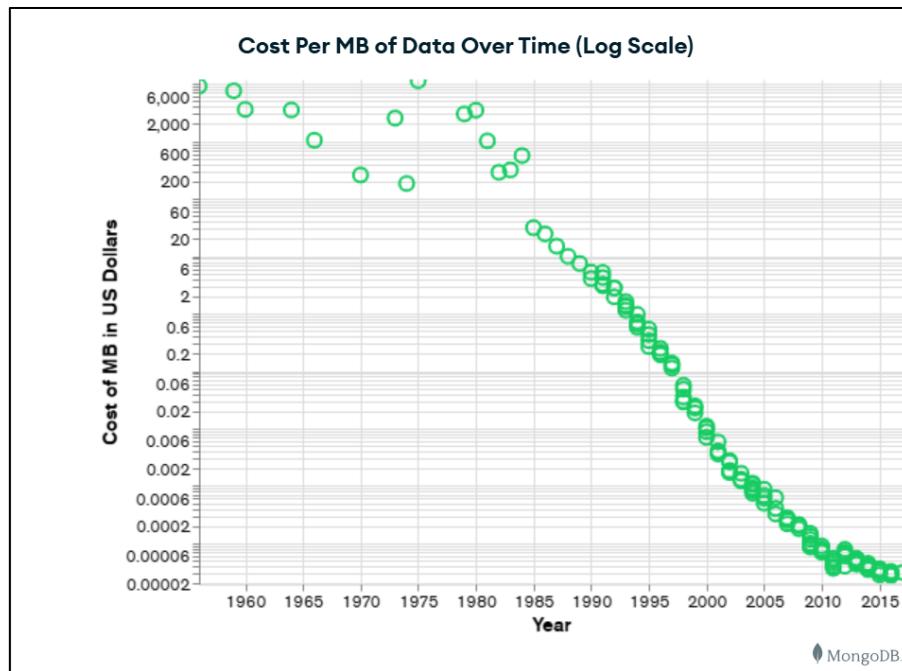
## Porównanie typów baz danych — NoSQL

Każda z baz danych NoSQL oferuje inne funkcje. Na przykład bazy danych grafowych mogą być bardziej odpowiednie do analizowania złożonych relacji i wzorców między jednostkami, podczas gdy bazy danych dokumentów zapewniają bardziej elastyczny, naturalny sposób przechowywania i pobierania dużych wolumenów danych podobnych typów jak dokumenty. Wybór bazy danych zależy od przypadku użycia, który chcesz opracować.

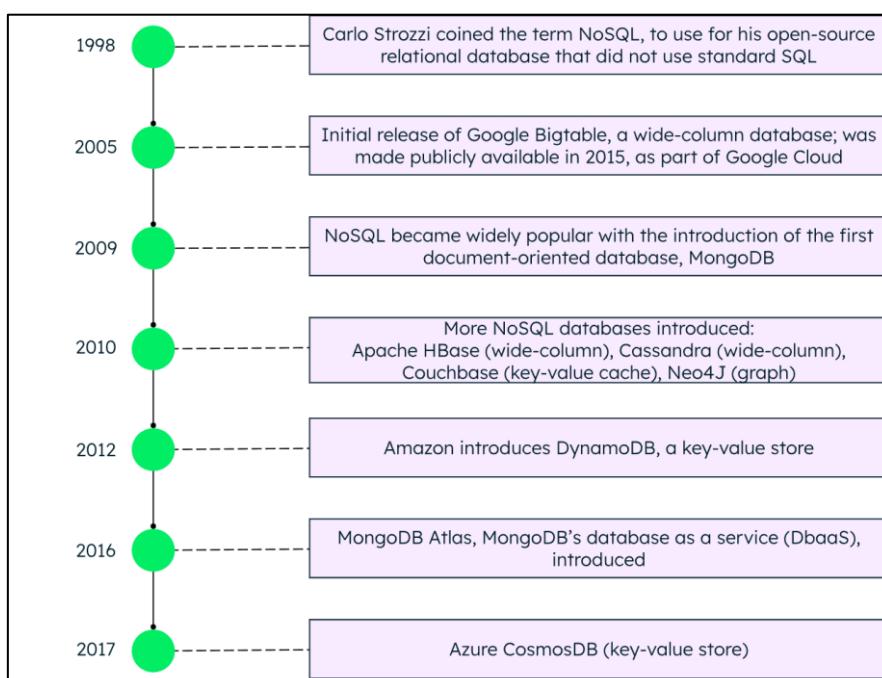
Basic support for data visualization	Basic support for data visualization	Basic support for data visualization	Basic support for data visualization
Basic authentication, limited access control	Role-based access control, encryption at-rest, encryption in transit, and in use	Role-based access control (RBAC), encryption at rest	Role-based access control (RBAC), encryption at rest, in transit
Limited support for online data archival	Automatic online data archival	Limited support for online data archival	Limited support for online data archival
Basic key-based data retrieval	Full-text search, vector search	Limited search capabilities	Limited search capabilities
Basic CRUD operations, limited data manipulation capabilities	Advanced query and data manipulation capabilities	Data manipulation and CRUD capabilities	Supports advanced graph traversal and manipulation operations
Stores any type of data	Stores JSON/BSON data	Columns and rows matrix	Nodes, edges, and relationships
Limited support for complex analytics	Suited for time-series, IoT analytics, real-time analytics	Suited for time-series data, IoT analytics	Well-suited for graph analytics
Horizontal scalability and eventual consistency	Horizontal scalability and eventual consistency	Horizontal scalability and eventual consistency	Horizontal scalability and eventual consistency
Simple indexes	Indexes on fields	Secondary indexes	Indexes on nodes/edges
Limited by key	Rich querying capability	Limited by columns	Specialized graph query
Schemaless	Flexible schema	Flexible schema	Flexible schema
Simple key-value pairs	JSON/BSON documents	Column and row families	Nodes, edges, and relationships
			
Key-value database	Document database	Wide-column database	Graph database

## Krótka historia NoSQL

Bazy danych NoSQL pojawiły się pod koniec lat 2000., kiedy koszty przechowywania danych gwałtownie spadły. Skończyły się czasy, w których konieczne było tworzenie złożonych i trudnych w zarządzaniu modeli danych tylko po to, aby uniknąć duplikacji danych. Bazy NoSQL zostały zoptymalizowane pod kątem produktywności programistów.



Wraz z gwałtownym spadkiem kosztów przechowywania, znacznie wzrosła ilość danych, które aplikacje musiały przechowywać i przetwarzać. Dane te przyjmowały wszelkie możliwe formy — **ustrukturyzowaną, częściowo ustrukturyzowaną i nieustrukturyzowaną** — i z góry określenie schematu stawało się niemal niemożliwe. Bazy danych NoSQL umożliwiają programistom przechowywanie ogromnych ilości danych nieustrukturyzowanych, zapewniając im dużą elastyczność.



Na początku lat 2000. firma Google opublikowała pracę naukową dotyczącą BigTable — kolumnowej bazy danych — która badała szeroki zakres możliwości systemu rozprozonego przechowywania danych. Rok 2009 przyniósł gwałtowny wzrost zainteresowania bazami NoSQL, a na rynku pojawiły się dwa kluczowe systemy baz dokumentowych: **MongoDB** i **CouchDB**.

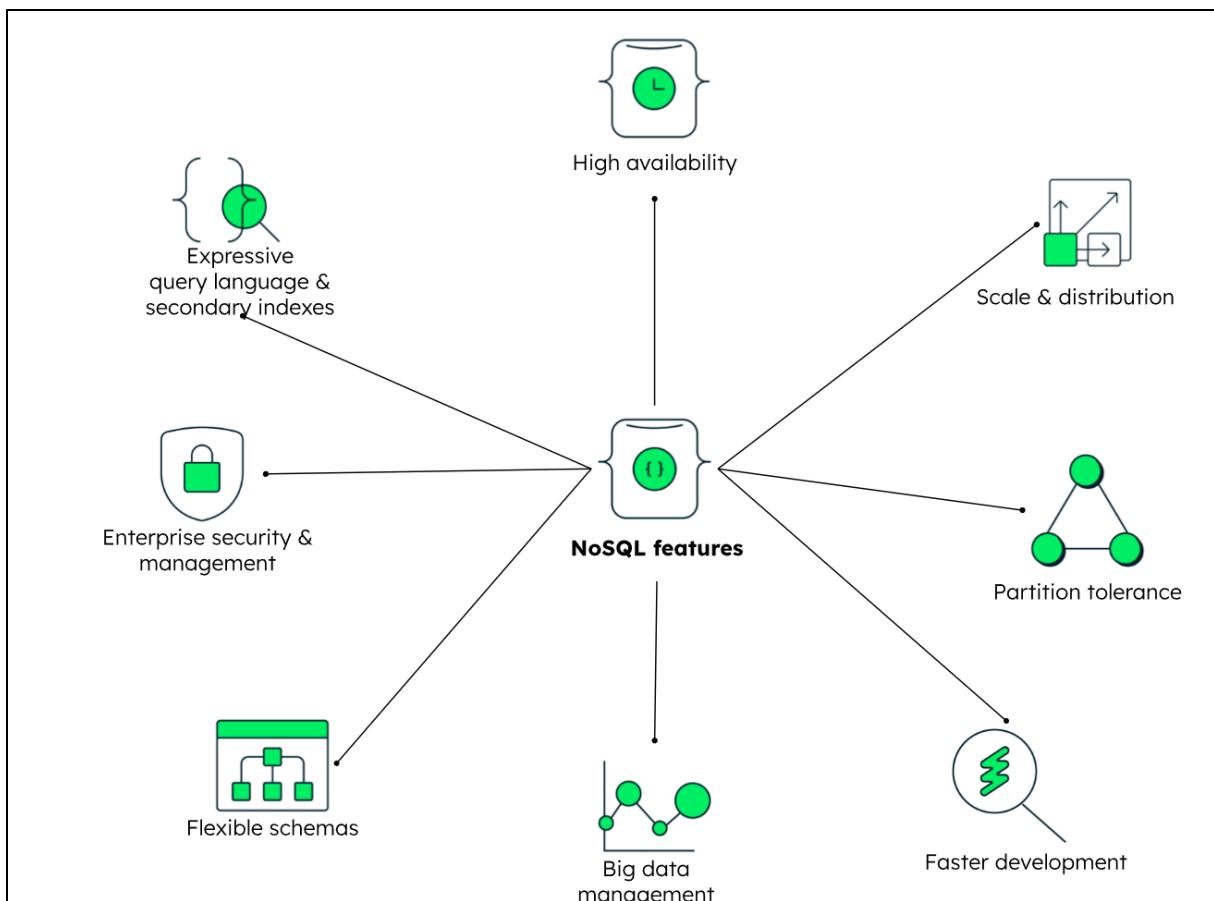
W latach 2010. zaczęły pojawiać się różne typy baz danych NoSQL, a ich akceptacja stała się powszechna, ponieważ firmy coraz bardziej opierały swoją działalność na danych.

Dodatkowo w tym czasie zyskiwał popularność **Manifest Agile**, a inżynierowie oprogramowania zaczęli na nowo przemyśleć sposób tworzenia aplikacji. Musieli szybko dostosowywać się do zmieniających się wymagań, często iterować i wprowadzać zmiany we wszystkich warstwach oprogramowania — aż po bazę danych. Bazy danych NoSQL dawały im tę elastyczność.

Popularność zaczęło również zyskiwać **przetwarzanie w chmurze**, a programiści zaczęli korzystać z chmur publicznych do hostowania swoich aplikacji i danych. Chcieli mieć możliwość rozpraszania danych pomiędzy wieloma serwerami i regionami, aby zwiększyć odporność swoich aplikacji, skalować je poziomo (zamiast tylko pionowo) oraz inteligentnie lokalizować dane geograficznie. Niektóre bazy danych NoSQL, takie jak **MongoDB Atlas**, zapewniają właśnie takie możliwości.

W wyniku **eksplozji cyfryzacji**, firmy obecnie gromadzą tyle nieustrukturyzowanych danych, ile tylko mogą. Aby móc analizować te dane i uzyskiwać z nich **użyteczne informacje w czasie rzeczywistym**, potrzebne są nowoczesne rozwiązania, które wykraczają poza zwykłe przechowywanie. Firmy potrzebują platform, które potrafią łatwo się **skalować, przetwarzać i wizualizować dane**; umożliwiać tworzenie **kokpitów menedżerskich (dashboardów), raportów i wykresów**; oraz współpracować z **narzędziami sztucznej inteligencji i analityki biznesowej**, aby przyspieszyć swoją efektywność biznesową. Ze względu na swoją elastyczność i rozproszony charakter, bazy danych NoSQL (np. MongoDB) doskonale sprawdzają się w tych zadaniach.

## Cechy baz danych NoSQL



- Skalowalność - <https://www.mongodb.com/resources/basics/scaling>
- Elastyczne schematy - <https://www.mongodb.com/docs/manual/data-modeling/#flexible-schema>
- Tolerancja na partycje - <https://www.mongodb.com/docs/manual/core/sharding-data-partitioning/>
- Wysoka dostępność - <https://www.mongodb.com/resources/basics/high-availability>
- Rozproszona architektura - <https://www.mongodb.com/resources/basics/distributed-database>
- Obsługa dużych zbiorów danych
- Zgodność z BASE (ang. Basic Availability, Soft state, Eventual consistency)

Bazy danych NoSQL są zgodne z modelem BASE, czyli:

**Basic Availability, Soft state, Eventual consistency** (podstawowa dostępność, stan przejściowy, spójność ostateczna).

- **Basic availability (podstawowa dostępność)** odnosi się do zdolności systemu do tolerowania częściowych awarii, np. utraty jednego z węzłów, bez całkowitego zatrzymania działania.

- **Soft state (stan przejściowy)** oznacza, że system dopuszcza chwilowe niespójności danych — dane mogą się zmieniać lub nie być w pełni aktualne w danym momencie.
- **Eventual consistency (spójność ostateczna)** zakłada, że system automatycznie osiągnie spójność po pewnym czasie, nawet jeśli początkowo występują różnice między kopiami danych.

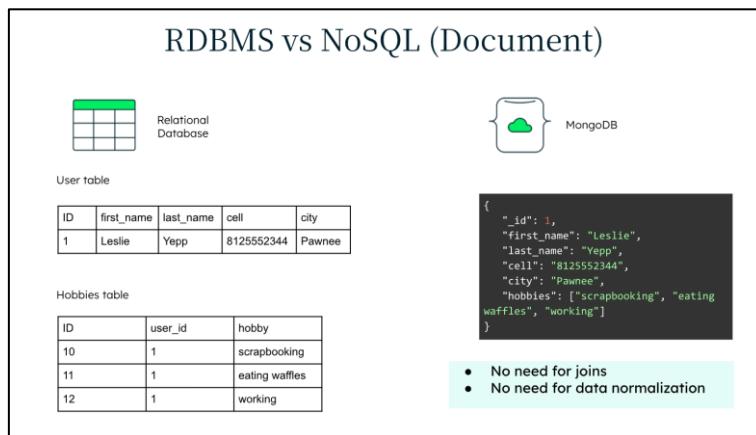
Zgodność z BASE zapewnia **wysoką dostępność, szybsze przetwarzanie danych, skalowalność i elastyczność** systemów NoSQL.

Jednakże **MongoDB** może zostać skonfigurowane także w taki sposób, aby zapewniało **zgodność z modelem ACID** (Atomicity, Consistency, Isolation, Durability) w przypadku transakcji obejmujących wiele dokumentów. Dzięki temu łączy zalety elastyczności NoSQL z rygorystycznymi wymaganiami dotyczącymi integralności danych znanymi z baz relacyjnych.

## Przykład: relacyjna vs NoSQL (dokumentowa)

**RDBMS:** dwie tabele – *Users* i *Hobbies*. Potrzebne związki (JOIN), by uzyskać pełne dane użytkownika.

**NoSQL (np. MongoDB):** wszystko zawarte w jednym dokumencie – brak potrzeby złączeń, szyszki zapytania.



## Różnice RDBMS vs NoSQL

Cechy	RDBMS	NoSQL
Model danych	Tabele (wiersze i kolumny)	Dokumenty, grafy, kolumny, klucz-wartość
Schemat	Sztywny	Elastyczny
Język zapytań	SQL	MQL (Mongo), Cypher (Neo4j), itp.
Skalowalność	Pionowa (ograniczona pozioma)	Pionowa i pozioma
Relacje	Klucze obce, JOIN-y	Zagnieżdżone, jawne lub niejawne relacje
Transakcje	ACID	BASE, czasem ACID
Spójność	Wysoka	Zwykle ostateczna (eventual consistency)
Odporność na awarie	Kopie zapasowe, replikacja	Wbudowana replikacja, wysokie SLA
Partycjonowanie	Partycjonowanie tabel	Sharding i replikacja
Mapowanie do obiektów	ORM (np. Hibernate)	ODM (np. Mongoose)

## Różnice między relacyjnymi bazami danych (RDBMS) a bazami NoSQL

Istnieje wiele różnic pomiędzy systemami zarządzania relacyjnymi bazami danych (RDBMS) a nierelacyjnymi bazami danych (NoSQL). Jedną z kluczowych różnic jest sposób modelowania danych. Poniżej przedstawiono zestawienie głównych cech i różnic:

### Modelowanie danych

- **NoSQL:** Modele danych różnią się w zależności od typu bazy – np. klucz-wartość, dokumentowe, grafowe, kolumnowe. Umożliwia to efektywne przechowywanie danych półstrukturalnych i niestrukturalnych.
- **RDBMS:** Wykorzystuje strukturę tabelaryczną – dane są reprezentowane jako zestaw wierszy i kolumn, co czyni je idealnymi do danych uporządkowanych (ustructuryzowanych).

### Schemat danych (schema)

- **NoSQL:** Schemat jest elastyczny – każdy dokument, para klucz-wartość lub wiersz może mieć różne pola. Zmiana struktury danych jest łatwa i nie wymaga dużych modyfikacji.
- **RDBMS:** Schemat jest sztywny – każda kolumna musi być zdefiniowana z góry, a każdy wiersz musi ją odzwierciedlać. Zmiana schematu po zapisaniu danych jest trudna.

### Język zapytań

- **NoSQL:** Język zależy od konkretnego typu bazy. Przykładowo, MongoDB używa MQL (MongoDB Query Language), a Neo4j – Cypher.
- **RDBMS:** Stosuje standardowy język SQL (Structured Query Language).

### Skalowalność

- **NoSQL:** Zaprojektowane z myślą o skalowaniu pionowym i poziomym (np. przez sharding i replikację).
- **RDBMS:** Głównie skalowalność pionowa (lepszy sprzęt), choć dostępne są ograniczone możliwości skalowania poziomego.

### Relacje między danymi

- **NoSQL:** Relacje mogą być zagnieżdżone, jawne lub ukryte, często reprezentowane bez konieczności łączenia (JOIN).
- **RDBMS:** Relacje tworzone są przez klucze obce (foreign keys) i odczytywane przez łączenia (JOIN).

### Transakcje

- **NoSQL:** Obsługuje transakcje zgodne z BASE lub ACID – w zależności od implementacji.
- **RDBMS:** Transakcje są zgodne z ACID (Atomowość, Spójność, Izolacja, Trwałość).

## **Wydajność**

- **NoSQL:** Odpowiednie dla środowisk czasu rzeczywistego, analiz Big Data i systemów rozproszonych.
- **RDBMS:** Skuteczne w zadaniach z dużą ilością odczytów i klasycznymi transakcjami.

## **Spójność danych**

- **NoSQL:** W większości przypadków oferuje **spójność ostateczną** (eventual consistency).
- **RDBMS:** Zapewnia **wysoką spójność danych** (strong consistency).

## **Przetwarzanie rozproszone**

- **NoSQL:** Stworzony z myślą o przetwarzaniu rozproszonym – obsługuje przechowywanie danych w różnych lokalizacjach przez sharding, replikację i klastrowanie.
- **RDBMS:** Obsługuje przetwarzanie rozproszone poprzez replikację i klastrowanie, ale z ograniczoną elastycznością – nie był pierwotnie projektowany z myślą o architekturze rozproszonej.

## **Odporność na błędy**

- **NoSQL:** Ma wbudowaną odporność na awarie i wysoką dostępność dzięki replikacji danych.
- **RDBMS:** Korzysta z mechanizmów tworzenia kopii zapasowych, replikacji i odzyskiwania danych – często wymagają one dodatkowej konfiguracji i mechanizmów Disaster Recovery.

## **Partyjonowanie danych**

- **NoSQL:** Partyjonowanie realizowane przez sharding i replikację.
- **RDBMS:** Obsługuje partyjonowanie tabel i optymalizację przez tzw. „partition pruning”.

## **Mapowanie danych do obiektów (Object Mapping)**

- **NoSQL:** Przechowuje dane w różnych formatach – np. jako dokumenty JSON, kolumny lub pary klucz-wartość. Dostępne są frameworki ODM (Object-Document Mapping), które umożliwiają obiektowe przetwarzanie danych.
- **RDBMS:** Wymaga ORM (Object-Relational Mapping) do odwzorowania struktur tabelarycznych na obiekty w kodzie aplikacji.

Szczegółowy opis różnic: <https://www.mongodb.com/resources/basics/databases/nosql-explained/nosql-vs-sql#differences-between-sql-and-nosql>

<https://www.mongodb.com/developer/products/mongodb/map-terms-concepts-sql-mongodb/>

## **Zastosowania NoSQL**

- Analiza w czasie rzeczywistym
- Zarządzanie treścią
- Internet Rzeczy (IoT)
- Systemy rekommendacji
- Wykrywanie oszustw
- Zarządzanie katalogami produktów
- Sektor finansowy i opieka zdrowotna
- Aplikacje mobilne i webowe o dużej skali

## **Kiedy używać NoSQL?**

- Dynamiczne środowiska Agile
- Dane o różnych strukturach
- Bardzo duże ilości danych
- Skalowalna architektura (np. mikroserwisy)
- Strumieniowe przetwarzanie danych w czasie rzeczywistym

<https://www.mongodb.com/resources/basics/databases/nosql-explained/when-to-use-nosql>

## **Materiały dodatkowe**

- <https://journals.indexcopernicus.com/api/file/viewById/338696>
- [https://wikizmsi.zut.edu.pl/uploads/f/f9/BD\\_W5.pdf](https://wikizmsi.zut.edu.pl/uploads/f/f9/BD_W5.pdf)
- <https://www.w3schools.com/mongodb/index.php>
- <https://www.geeksforgeeks.org/introduction-to-nosql/>
- <https://www.geeksforgeeks.org/non-relational-databases-and-their-types/>
- <https://www.youtube.com/watch?v=xh4gy1lbL2k>
- <https://github.com/alamgirqazi/IntroToNoSQL?tab=readme-ov-file>
- <https://rk.edu.pl/pl/mongodb/>
- <https://www.poswojsku.info/internet-it/MongoDB-nierelacyjna-baza-danych-NoSQL-poradnik-developera-bazy-danych-MongoDB.html>
- <https://www.javacodegeeks.com/2015/09/mongodb-mapreduce-tutorial.html>
- <https://www.datacamp.com/tutorial/nosql-tutorial>
- <https://www.altexsoft.com/blog/nosql-databases/>
- <https://aws.amazon.com/nosql/>
- <https://github.com/turiphro/NoSQLlab>
- <https://github.com/mongodb/mongo>
- <https://github.com/redis/redis>

## Przykłady

### Przykładowa kolekcja: students

<https://www.mycompiler.io/pl/new/mongodb>

Dodaj dane testowe:

```
db.students.insertMany([
  { "name": "Anna", "age": 22, "major": "Biology", "grade": 4.5 },
  { "name": "Tomasz", "age": 24, "major": "Physics", "grade": 3.7 },
  { "name": "Maria", "age": 21, "major": "Computer Science", "grade": 5.0 },
  { "name": "Kamil", "age": 23, "major": "Mathematics", "grade": 2.9 }
]);
```

---

#### 1. Wyświetlenie wszystkich dokumentów

db.students.find()

Zwraca wszystkie dokumenty z kolekcji students.

#### 2. Wyszukiwanie warunkowe (filtrowanie danych)

db.students.find({ major: "Physics" })

Zwraca studentów, których kierunek to **Physics**.

#### 3. Wyszukiwanie z operatorem porównania

db.students.find({ age: { \$gt: 22 } })

Zwraca studentów starszych niż 22 lata.

\$gt = greater than (większy niż)

#### 4. Wyświetlenie tylko wybranych pól

db.students.find({}, { name: 1, grade: 1, \_id: 0 })

Zwraca tylko pola name i grade, **bez \_id**.

#### 5. Sortowanie wyników

db.students.find().sort({ grade: -1 })

Sortuje studentów malejąco według oceny (grade).

-1 = sortuj malejąco, 1 = rosnąco.

## **6. Aktualizacja danych**

```
db.students.updateOne(  
  { name: "Kamil" },  
  { $set: { grade: 3.2 } }  
)
```

Zmienia ocenę Kamilu z 2.9 na 3.2.

\$set ustawia nową wartość pola.

## **7. Usuwanie dokumentu**

```
db.students.deleteOne({ name: "Tomasz" })
```

Usuwa jednego studenta o imieniu Tomasz.

## **8. Dodanie nowego dokumentu**

```
db.students.insertOne({  
  name: "Olga",  
  age: 20,  
  major: "Chemistry",  
  grade: 4.1  
})
```

Dodaje nowego studenta do kolekcji.

## **9. Zliczanie dokumentów**

```
db.students.countDocuments()
```

Zwraca liczbę dokumentów w kolekcji.

## **10. Użycie operatora logicznego**

```
db.students.find({  
    $or: [  
        { major: "Mathematics" },  
        { grade: { $gt: 4.0 } }  
    ]  
})
```

Zwraca studentów, którzy **studują matematykę** lub mają **ocenę powyżej 4.0**.

## **11. Sprawdzenie istnienia pola**

```
db.students.find({ grade: { $exists: true } })
```

Zwraca tylko te dokumenty, które mają pole grade.

Polecenie	Opis
use studentsDB	Wybór lub utworzenie bazy danych
db.createCollection("students")	Tworzenie kolekcji
db.students.insertOne({ name: "Anna", age: 22 })	Dodanie jednego dokumentu
db.students.insertMany([...])	Dodanie wielu dokumentów
db.students.find()	Wyszukiwanie wszystkich dokumentów
db.students.find({ age: { \$gt: 22 } })	Wyszukiwanie z warunkiem
db.students.find({ major: "Physics" }, { name: 1 })	Wyszukiwanie z projekcją (wybrane pola)
db.students.updateOne({ name: "Anna" }, { \$set: { grade: 4.8 } })	Aktualizacja jednego dokumentu
db.students.updateMany({ grade: { \$lt: 3.0 } }, { \$set: { status: "Failed" } })	Aktualizacja wielu dokumentów
db.students.deleteOne({ name: "Kamil" })	Usunięcie jednego dokumentu
db.students.deleteMany({ grade: { \$lt: 3.0 } })	Usunięcie wielu dokumentów
db.students.aggregate([...])	Agregacja dokumentów (np. średnia, grupowanie)
db.students.find().sort({ age: -1 }).limit(3)	Sortowanie i ograniczenie wyników

## Ciekawostka – Przykłady Key Value oraz bazy grafowe

### Model klucz-wartość (Key-Value) – Redis

W modelu klucz-wartość każdy rekord składa się z unikalnego **klucza** oraz przypisanej do niego **wartości**. Redis to najpopularniejszy system oparty na tym modelu.

#### Przykład 1: Prosty zapis i odczyt wartości

```
SET user:1001 "Anna Kowalska"
```

```
GET user:1001
```

Polecenie SET ustawia wartość "Anna Kowalska" pod kluczem user:1001.

Polecenie GET odczytuje tę wartość.

#### Przykład 2: Przechowywanie danych w formacie JSON (jako string)

```
SET user:1002 "{\"name\":\"Tomasz\", \"email\":\"tom@edu.com\", \"age\":24}"
```

```
GET user:1002
```

Redis nie przechowuje struktury JSON natywnie, traktuje to jako tekst. Aby wykonywać operacje na JSON, używa się rozszerzenia RedisJSON.

#### Przykład 3: Licznik odwiedzin (liczba całkowita)

```
SET counter:visits 0
```

```
INCR counter:visits
```

```
INCR counter:visits
```

```
GET counter:visits
```

Polecenie INCR zwiększa wartość liczbową o 1. Po dwóch wywołaniach wynik to 2.

#### Przykład 4: Ustawienie klucza z czasem wygaśnięcia

```
SET session:abc123 "active" EX 30
```

```
TTL session:abc123
```

Opcja EX 30 oznacza, że klucz wygaśnie po 30 sekundach.

Polecenie TTL pokazuje ile sekund pozostało do usunięcia klucza.

**Przykład 5: Przechowywanie listy (np. historia aktywności)**

RPUSH user:1003:actions "login" "browse" "logout"

LRANGE user:1003:actions 0 -1

RPUSH dodaje elementy do listy (na końcu).

LRANGE zwraca elementy z listy w podanym zakresie. 0 -1 oznacza wszystkie.

**Przykład 6: Sprawdzenie istnienia klucza**

EXISTS user:1001

Zwraca 1, jeśli klucz istnieje lub 0, jeśli nie istnieje.

**Przykład 7: Nadpisanie istniejącej wartości**

SET user:1001 "Anna Nowak"

GET user:1001

SET automatycznie nadpisuje wartość pod danym kluczem.

**Przykład 8: Usunięcie klucza**

DEL user:1001

GET user:1001

DEL usuwa klucz z bazy. GET po usunięciu zwróci pusty wynik lub nil.

**Przykład 9: Grupowanie danych przez przestrzenie nazw**

SET product:101:name "Laptop"

SET product:101:price 3299

GET product:101:name

Tworzenie nazw z : (np. product:101:name) pozwala logicznie grupować dane w klucze powiązane tematycznie.

### **Podsumowanie poleceń Redis**

Polecenie	Opis
SET	Ustawienie wartości pod kluczem
GET	Odczyt wartości spod danego klucza
DEL	Usunięcie klucza
INCR	Zwiększenie wartości liczbowej
EXISTS	Sprawdzenie, czy klucz istnieje
RPUSH	Dodanie wartości na koniec listy
LRANGE	Pobranie wartości z listy w danym zakresie
TTL	Sprawdzenie czasu życia klucza (sekundy)

## Bazy grafowe

W bazach grafowych dane przechowywane są jako:

- **węzły (nodes)** – reprezentujące obiekty (np. osoby, produkty, miasta),
- **relacje (relationships)** – opisujące powiązania między węzłami (np. zna, pracuje, mieszka w).

### Przykład 1: Tworzenie pojedynczego węzła

```
CREATE (:Person {name: "Anna", age: 22})
```

Tworzy węzeł typu Person z właściwościami name i age.

### Przykład 2: Tworzenie dwóch węzłów i relacji między nimi

```
CREATE (a:Person {name: "Anna"})
```

```
CREATE (b:Person {name: "Tomasz"})
```

```
CREATE (a)-[:FRIEND]->(b)
```

Tworzy dwa węzły typu Person oraz relację FRIEND wychodzącą od Anny do Tomasza.

### Przykład 3: Tworzenie relacji z właściwościami

```
CREATE (a:Person {name: "Maria"})
```

```
CREATE (b:Person {name: "Piotr"})
```

```
CREATE (a)-[:WORKS_WITH {since: 2021}]->(b)
```

Tworzy relację WORKS\_WITH z dodatkowymi danymi (since: 2021), opisującymi, od kiedy osoby współpracują.

**Przykład 4: Zapytanie – znajdź wszystkich znajomych Anny**

```
MATCH (a:Person {name: "Anna"})-[:FRIEND]->(friend)  
RETURN friend.name
```

Wyszukuje wszystkie węzły połączone z Anną relacją FRIEND i zwraca imię znajomych.

**Przykład 5: Zapytanie odwrotne – kto zna Piotra**

```
MATCH (person)-[:FRIEND]->(p:Person {name: "Piotr"})  
RETURN person.name
```

Zwraca osoby, które mają relację FRIEND skierowaną do Piotra.

**Przykład 6: Zapytanie z relacjami w obu kierunkach**

```
MATCH (a:Person)-[:FRIEND]->(b:Person)  
RETURN a.name, b.name
```

Zwraca wszystkie pary osób, które są znajomymi niezależnie od kierunku relacji.

**Przykład 7: Relacja między różnymi typami węzłów**

```
CREATE (s:Student {name: "Kamil"})  
CREATE (c:Course {title: "Databases"})  
CREATE (s)-[:ENROLLED_IN]->(c)
```

Tworzy relację ENROLLED\_IN między studentem a kursem.

### **Przykład 8: Filtrowanie po właściwościach w relacji**

```
MATCH (s:Student)-[r:ENROLLED_IN]->(c:Course)
```

```
WHERE r.semester = "Spring 2024"
```

```
RETURN s.name, c.title
```

Zwraca studentów zapisanych na kursy w określonym semestrze, pod warunkiem że relacja zawiera pole semester.

### **Przykład 9: Usunięcie relacji**

```
MATCH (a:Person)-[r:FRIEND]->(b:Person)
```

```
WHERE a.name = "Anna" AND b.name = "Tomasz"
```

```
DELETE r
```

Usuwa relację FRIEND między Anną a Tomaszem.

### **Przykład 10: Usunięcie węzła i wszystkich relacji**

```
MATCH (p:Person {name: "Anna"})
```

```
DETACH DELETE p
```

Usuwa węzeł Anna oraz wszystkie relacje, w których uczestniczy.

## **Narzędzia do ćwiczeń online**

<b>Narzędzie</b>	<b>Adres</b>
Neo4j Sandbox	<a href="https://sandbox.neo4j.com">https://sandbox.neo4j.com</a>
Neo4j Aura Free	<a href="https://neo4j.com/cloud/aura/">https://neo4j.com/cloud/aura/</a>
Graph Academy	<a href="https://graphacademy.neo4j.com">https://graphacademy.neo4j.com</a>
neCompiler (Neo4j)	<a href="https://onecompiler.com/neo4j">https://onecompiler.com/neo4j</a>

Polecenie	Opis
CREATE (:Student {name: 'Anna', age: 22, major: 'Biology'})	Tworzenie wierzchołka
MATCH (a:Student {name: 'Anna'}), (b:Student {name: 'Tomasz'}) CREATE (a)-[:FRIEND]->(b)	Tworzenie relacji między wierzchołkami
MATCH (s:Student) RETURN s	Wyszukiwanie wszystkich wierzchołków typu Student
MATCH (a)-[:FRIEND]->(b) RETURN a, b	Wyszukiwanie relacji typu FRIEND
MATCH (s:Student {name: 'Anna'}) SET s.grade = 4.8	Aktualizacja danych w wierzchołku
MATCH (s:Student {name: 'Tomasz'}) DETACH DELETE s	Usunięcie wierzchołka wraz z relacjami
MATCH (s:Student) RETURN s.major, AVG(s.age) AS avgAge	Agregacja – średni wiek według kierunku

## Zadania praktyczne

### Zadanie 1: Operacje CRUD online

Linki:

- [MyCompiler MongoDB](#)
- [OneCompiler MongoDB](#)
- [MongoPlayground](#)

**Wstaw dane:**

```
{ "name": "John", "age": 30, "city": "New York" }  
{ "name": "Maria", "age": 25, "city": "Berlin" }  
{ "name": "Ali", "age": 35, "city": "London" }
```

**Wykonaj operacje:**

1. Dodaj dokumenty do kolekcji users.
2. Wyszukaj wszystkich użytkowników z miasta Berlin.
3. Zaktualizuj wiek użytkownika John do 31.
4. Usuń użytkownika Ali.

### Zadanie 2: Zapytania na danych ćwiczeniowych

Link: [NoSQL Zoo](#)

Instrukcje:

1. Przejdź do sekcji "MongoDB".
2. Wykonaj dostępne zapytania.

### Zadanie 3: Laboratoria interaktywne

Link: [Labex MongoDB](#)

Zadanie:

1. Zaloguj się lub załóż konto.
2. Ukończ laby związane z MongoDB

#### **Zadanie 4: Wprowadzenie do NoSQL Injection**

**Linki:**

- [TryHackMe – NoSQL Injection](#)
- [PortSwigger – NoSQL Injection](#)

#### **Zadanie 5: Pierwsze połączenie z chmurą MongoDB**

**Link:** [MongoDB Atlas](#)

<https://www.mongodb.com/resources/products/platform/mongodb-atlas-tutorial>

**Kroki:**

1. Zarejestruj się na stronie MongoDB Atlas.
2. Stwórz darmowy klaster (Free Tier).
3. Utwórz bazę danych o nazwie School, a w niej kolekcję Students.
4. Dodaj jeden przykładowy dokument:

```
{  
  "name": "Anna Kowalska",  
  "age": 21,  
  "major": "Computer Science"  
}
```