SQL, NoSQL, NewSQL czyli podróż przez bogaty świat baz danych

Piotr Fulmański, Konrad Kosmatka

piotr.fulmanski@wmii.uni.lodz.pl, konrad.kosmatka@wmii.uni.lodz.pl
http://fulmanski.pl

21 listopada 2019

Table of contents

Na wykładzie przedstawimy powody dla których z SQL wyewoluował NoSQL a potem, być może, NewSQL. Powody te, jak większość rzeczy w informatyce, wynikają z naturalnych potrzeb i są podyktowane czynnikami czysto praktycznymi.
Rozszerzoną wersję prezentowanego materiału znaleźć można na stronie
`https://fulmanski.pl/tutorials/computer-science/computer-science-for-beginners/sql-nosql-newsql/`

Możemy zdefiniować bezę danych jako

Baza danych

zorganizowany zbiór danych przechowywanych zgodnie ze szczegółowymi zasadami.

W tym sensie kolekcja znaczków pocztowych, książki przechowywane na półce (w sposób systematyczny lub chaotyczny), a nawet kolekcja dziecięcych samochodzików są przykładami *baz danych*.

W praktyce zwykle mamy do czynienia z *danymi niematerialnymi* – przechowujemy liczby, teksty, obrazy i piosenki, ale nie prawdziwe przedmioty. Prawdopodobnie dlatego, że prawdziwymi obiektami trudniej jest manipulować w sposób automatyczny niż sekwencje znaków.

Zgodnie z tym praktycznym podejściem dane definiujemy jako

Dane

zestaw wartości zmiennych (cech) jakościowych lub ilościowych opisujących jakiś obiekt lub zjawisko.

Baza danych

dane, informacje, wiedza, mądrość

- *Dane* – nic więcej, jak tylko zbiór wartości (np. liczby).
- *Informacje* – dane, które zaczynają „mówić”; nadajemy im znaczenie (np. liczby oznaczają zmianę temperatury w komorze silnika).
- *Wiedza* – informacja z nadanym znaczeniem (np. zbyt wysoka wartość temperatury w komorze silnika może wiązać się z pożarem).
- *Mądrość* – umiejętne, krytyczne, korzystanie z wiedzy.

Sens tych zależności doskonale oddaje poniższe powiedzenie (Miles Kington):

Knowledge is knowing a tomato is a fruit.
Wisdom is not putting it in a fruit salad.

Baza danych

dane, informacje, wiedza, mądrość

- *Dane* – nic więcej, jak tylko zbiór wartości (np. liczby).
- *Informacje* – dane, które zaczynają „mówić”; nadajemy im znaczenie (np. liczby oznaczają zmianę temperatury w komorze silnika).
- *Wiedza* – informacja z nadanym znaczeniem (np. zbyt wysoka wartość temperatury w komorze silnika może wiązać się z pożarem).
- *Mądrość* – umiejętne, krytyczne, korzystanie z wiedzy.

Sens tych zależności doskonale oddaje poniższe powiedzenie (Miles Kington):

Knowledge is knowing a tomato is a fruit.
Wisdom is not putting it in a fruit salad.

Baza danych

dane, informacje, wiedza, mądrość

- *Dane* – nic więcej, jak tylko zbiór wartości (np. liczby).
- *Informacje* – dane, które zaczynają „mówić”; nadajemy im znaczenie (np. liczby oznaczają zmianę temperatury w komorze silnika).
- *Wiedza* – informacja z nadanym znaczeniem (np. zbyt wysoka wartość temperatury w komorze silnika może wiązać się z pożarem).
- *Mądrość* – umiejętne, krytyczne, korzystanie z wiedzy.

Sens tych zależności doskonale oddaje poniższe powiedzenie (Miles Kington):

Knowledge is knowing a tomato is a fruit.
Wisdom is not putting it in a fruit salad.

Przywiązujemy dużą wagę do automatycznego sposobu przetwarzania. Najbardziej znanym narzędziem, które pozwala nam to robić w dzisiejszych czasach, jest komputer. Dlatego niematerialne dane są dla nas tak przydatne – możemy zamienić je w dane cyfrowe i zasilić nimi systemy komputerowe, aby zrobiły dla nas rzeczy, których sami nie chcemy robić.

Powyższe dwie cechy (niematerialny charakter danych, przetwarzanie danych za pomocą komputerów) uzasadniają poniższą, obecnie używaną, definicję baz danych

Baza danych

zorganizowany zbiór danych cyfrowych gromadzonych zgodnie z zasadami przyjętymi dla danego programu komputerowego specjalizującego się w gromadzeniu, przechowywaniu i przetwarzaniu tych danych zgodnie ze szczegółowymi zasadami.

Taki program (często pakiet różnych programów) nazywa się *systemem zarządzania bazą danych* (DBMS) a w skrócie – bazą danych.

Baza danych

System zarządzania bazą danych (DBMS) to oprogramowanie, które pośredniczy pomiędzy użytkownikiem, aplikacjami i samą bazą danych w celu gromadzenia, przechowywania i przetwarzania danych. Służy jako warstwa pośrednia izolująca użytkownika od wszystkich „niepotrzebnych" szczegółów technicznych.

Nie sięgając zbyt daleko wstecz możemy stwierdzić, że pierwszym modelem używany na szeroką skalę, dominujący na rynku przez 20 lat, były powstałe w latach 70. relacyjne bazach danych. Nazywamy je bazami danych SQL ponieważ język Structured Query Language był używany przez zdecydowaną większość z nich do komunikacji z bazą.

SQL (w sensie bazy danych) wykorzystują model relacyjny Edgara F. Codda: dane są reprezentowane w krotkach, pogrupowanych w relacje.

- Relacje to odpowiednik tabeli w Excell.
- Krotki to odpowiedniki wierszy w tabeli.
- Atrybuty to klumny w tabeli.
- Możemy mieć dowolną liczbę tabel w naszej bazie danych.
- W każdej tabeli możemy mieć dowolną, ale precyzyjnie określoną liczbę kolumn.
- Każda kolumna ma ściśle określone właściwości.
- Używając kluczy (które są unikalnym identyfikatorem dla każdego wiersza w danej tabeli) możemy zdefiniować relacje między

SQL (w sensie bazy danych) wykorzystują model relacyjny Edgara F. Codda: dane są reprezentowane w krotkach, pogrupowanych w relacje.

- Relacje to odpowiednik tabeli w Excell.
- Krotki to odpowiedniki wierszy w tabeli.
- Atrybuty to klumny w tabeli.
- Możemy mieć dowolną liczbę tabel w naszej bazie danych.
- W każdej tabeli możemy mieć dowolną, ale precyzyjnie określoną liczbę kolumn.
- Każda kolumna ma ściśle określone właściwości.
- Używając kluczy (które są unikalnym identyfikatorem dla każdego wiersza w danej tabeli) możemy zdefiniować relacje między

SQL (w sensie bazy danych) wykorzystują model relacyjny Edgara F. Codda: dane są reprezentowane w krotkach, pogrupowanych w relacje.

- Relacje to odpowiednik tabeli w Excell.
- Krotki to odpowiedniki wierszy w tabeli.
- Atrybuty to klumny w tabeli.
- Możemy mieć dowolną liczbę tabel w naszej bazie danych.
- W każdej tabeli możemy mieć dowolną, ale precyzyjnie określoną liczbę kolumn.
- Każda kolumna ma ściśle określone właściwości.
- Używając kluczy (które są unikalnym identyfikatorem dla każdego wiersza w danej tabeli) możemy zdefiniować relacje między

SQL (w sensie bazy danych) wykorzystują model relacyjny Edgara F. Codda: dane są reprezentowane w krotkach, pogrupowanych w relacje.

- Relacje to odpowiednik tabeli w Excell.
- Krotki to odpowiedniki wierszy w tabeli.
- Atrybuty to klumny w tabeli.
- Możemy mieć dowolną liczbę tabel w naszej bazie danych.
- W każdej tabeli możemy mieć dowolną, ale precyzyjnie określoną liczbę kolumn.
- Każda kolumna ma ściśle określone właściwości.
- Używając kluczy (które są unikalnym identyfikatorem dla każdego wiersza w danej tabeli) możemy zdefiniować relacje między

SQL (w sensie bazy danych) wykorzystują model relacyjny Edgara F. Codda: dane są reprezentowane w krotkach, pogrupowanych w relacje.

- Relacje to odpowiednik tabeli w Excell.
- Krotki to odpowiedniki wierszy w tabeli.
- Atrybuty to klumny w tabeli.
- Możemy mieć dowolną liczbę tabel w naszej bazie danych.
- W każdej tabeli możemy mieć dowolną, ale precyzyjnie określoną liczbę kolumn.
- Każda kolumna ma ściśle określone właściwości.
- Używając kluczy (które są unikalnym identyfikatorem dla każdego wiersza w danej tabeli) możemy zdefiniować relacje między

SQL (w sensie bazy danych) wykorzystują model relacyjny Edgara F. Codda: dane są reprezentowane w krotkach, pogrupowanych w relacje.

- Relacje to odpowiednik tabeli w Excell.
- Krotki to odpowiedniki wierszy w tabeli.
- Atrybuty to klumny w tabeli.
- Możemy mieć dowolną liczbę tabel w naszej bazie danych.
- W każdej tabeli możemy mieć dowolną, ale precyzyjnie określoną liczbę kolumn.
- Każda kolumna ma ściśle określone właściwości.
- Używając kluczy (które są unikalnym identyfikatorem dla każdego wiersza w danej tabeli) możemy zdefiniować relacje między

SQL (w sensie bazy danych) wykorzystują model relacyjny Edgara F. Codda: dane są reprezentowane w krotkach, pogrupowanych w relacje.

- Relacje to odpowiednik tabeli w Excell.
- Krotki to odpowiedniki wierszy w tabeli.
- Atrybuty to klumny w tabeli.
- Możemy mieć dowolną liczbę tabel w naszej bazie danych.
- W każdej tabeli możemy mieć dowolną, ale precyzyjnie określoną liczbę kolumn.
- Każda kolumna ma ściśle określone właściwości.
- Używając kluczy (które są unikalnym identyfikatorem dla każdego wiersza w danej tabeli) możemy zdefiniować relacje między

SQL (w sensie bazy danych) wykorzystują model relacyjny Edgara F. Codda: dane są reprezentowane w krotkach, pogrupowanych w relacje.

- Relacje to odpowiednik tabeli w Excell.
- Krotki to odpowiedniki wierszy w tabeli.
- Atrybuty to klumny w tabeli.
- Możemy mieć dowolną liczbę tabel w naszej bazie danych.
- W każdej tabeli możemy mieć dowolną, ale precyzyjnie określoną liczbę kolumn.
- Każda kolumna ma ściśle określone właściwości.
- Używając kluczy (które są unikalnym identyfikatorem dla każdego wiersza w danej tabeli) możemy zdefiniować relacje między

Table "Customers"

ID	Name
1	Al
2	Betty
3	Carolina
4	Diana
5	Emma
6	Fiona

Table "Orders"

ID	CustomerID	Total
1	1	100.00
2	1	33.00
3	4	5.00
4	2	250.50
5	3	64.00
6	6	172.00

Zauważmy, że kolumna „CustomerID” w tabeli „Orders” odnosi się do kolumny „ID” w tabeli „Customers”.

Table "Customers"

ID	Name
1	Al
2	Betty
3	Carolina
4	Diana
5	Emma
6	Fiona

Table "Orders"

ID	CustomerID	Total
1	1	100.00
2	1	33.00
3	4	5.00
4	2	250.50
5	3	64.00
6	6	172.00

```
SELECT name
FROM Customers;
```

```
SELECT name
FROM Customers;

Al
Betty
Carolina
Diana
Emma
Fiona
```

Table "Customers"

ID	Name
1	Al
2	Betty
3	Carolina
4	Diana
5	Emma
6	Fiona

Table "Orders"

ID	CustomerID	Total
1	1	100.00
2	1	33.00
3	4	5.00
4	2	250.50
5	3	64.00
6	6	172.00

```
SELECT CustomerID
FROM Orders
WHERE Total > 100;
```

```
SELECT CustomerID
FROM Orders
WHERE Total > 100;
```

```
4
6
```

```
Table "Customers"   Table "Orders"
ID Name             ID CustomerID   Total
 1 Al                1           1  100.00
 2 Betty             2           1   33.00
 3 Carolina          3           4    5.00
 4 Diana             4           2  250.50
 5 Emma              5           3   64.00
 6 Fiona             6           6  172.00
```

```
SELECT Name
FROM Customers, Orders
WHERE Total > 100
AND Customers.ID = Orders.CustomerID
```

or

```
SELECT Name
FROM Customers
INNER JOIN Orders ON Customers.ID = Orders.CustomerID
WHERE Total > 100;
```

```
SELECT Name
FROM Customers, Orders
WHERE Total > 100
AND Customers.ID = Orders.CustomerID

or

SELECT Name
FROM Customers
INNER JOIN Orders ON Customers.ID = Orders.CustomerID
WHERE Total > 100;
```

```
Diana
Fiona
```

Table "Customers"

ID	Name
1	Al
2	Betty
3	Carolina
4	Diana
5	Emma
6	Fiona

Table "Orders"

ID	CustomerID	Total
1	1	100.00
2	1	33.00
3	4	5.00
4	2	250.50
5	3	64.00
6	6	172.00

```
INSERT INTO Customers(ID, Name)
VALUES (7, 'Helen');
```

```
INSERT INTO Customers(ID, Name)
VALUES (7, 'Helen');
```

Table "Customers"

ID	Name
1	Al
2	Betty
3	Carolina
4	Diana
5	Emma
6	Fiona
7	Helen

Table "Orders"

ID	CustomerID	Total
1	1	100.00
2	1	33.00
3	4	5.00
4	2	250.50
5	3	64.00
6	6	172.00

Table "Customers"

ID	Name
1	Al
2	Betty
3	Carolina
4	Diana
5	Emma
6	Fiona
7	Helen

Table "Orders"

ID	CustomerID	Total
1	1	100.00
2	1	33.00
3	4	5.00
4	2	250.50
5	3	64.00
6	6	172.00

```
UPDATE Customers
SET name = 'Grace'
WHERE ID = 7;
```

```
UPDATE Customers
SET name = 'Grace'
WHERE ID = 7;
```

Table "Customers"

ID	Name
1	Al
2	Betty
3	Carolina
4	Diana
5	Emma
6	Fiona
7	Grace

Table "Orders"

ID	CustomerID	Total
1	1	100.00
2	1	33.00
3	4	5.00
4	2	250.50
5	3	64.00
6	6	172.00

Table "Customers"

ID	Name
1	Al
2	Betty
3	Carolina
4	Diana
5	Emma
6	Fiona
7	Grace

Table "Orders"

ID	CustomerID	Total
1	1	100.00
2	1	33.00
3	4	5.00
4	2	250.50
5	3	64.00
6	6	172.00

```
DELETE FROM Customers
WHERE name = 'Grace';
```

```
DELETE FROM Customers
WHERE name = 'Grace';
```

Table "Customers"

ID	Name
1	Al
2	Betty
3	Carolina
4	Diana
5	Emma
6	Fiona

Table "Orders"

ID	CustomerID	Total
1	1	100.00
2	1	33.00
3	4	5.00
4	2	250.50
5	3	64.00
6	6	172.00

Nieodłączną częścią modelu relacyjnego jest zbiór reguł, tzw. postaci normalnych. Zbiór ten nie ma charakteru restrykcyjnego: nie trzeba ich stosować, aby pracować z relacyjną bazą danych. Należy je jednak stosować, aby to robić efektywnie.

Sposób organizacji danych narzucony przez postacie normalne skutkuje tym, jak myślimy o rzeczywistych obiektach.

Relacyjna baza danych przypomina garaż, który zmusza do rozłożenia samochodu na części i przechowywania elementów w małych szufladach za każdym razem, gdy do niego wjeżdżamy.

Sposób organizacji danych narzucony przez postacie normalne skutkuje tym, jak myślimy o rzeczywistych obiektach.
Relacyjna baza danych przypomina garaż, który zmusza do rozłożenia samochodu na części i przechowywania elementów w małych szufladach za każdym razem, gdy do niego wjeżdżamy.

Niedopasowanie między modelem relacyjnym (sposobem przechowywania danych) a rzeczywistym obiektem występującym w obiektowych językach programowania (sposób, w jaki wykorzystujemy dane) miało poważne konsekwencje. Ilekroć obiekt był przechowywany w relacyjnej bazie danych lub z niej pobierany, wymaganych było wiele operacji SQL w celu zamiany z reprezentacji zorientowanej obiektowo na reprezentację relacyjną (lub na odwrót) i prowadziło do różnego rodzaju problemów. Doprowadziło to do pierwszej próby zastąpienia relacyjnych baz danych czymś innym. Dlatego na przełomie lat 80. i 90. XX wieku opracowano obiektowe DBMS (OODBMS).

Owszem, OODBMS oferowały korzyści programistom, ale zapomniano przy tym o tych, którzy chcieli wykorzystać przechowywane w nich informacje do celów biznesowych. Implementacja modelu danych, który był zrozumiały i mógł być używany tylko przez programistę, oraz ignorowanie użytecznego interfejs SQL, okazało się ślepą uliczką.

W ten sposób bazy danych SQL obroniły swoją dominującą pozycję na rynku. Innymi słowy, relacyjne bazy danych, pomimo swoich wad, były bardzo dobrze ugruntowane w branży IT, wydawały się być doskonale dostosowane do wszystkich potrzeb i nic nie zapowiadało nadejścia nowej ery. Nieoczekiwanie w 2000 roku popularne stały się nierelacyjne bazy danych, zwane NoSQL, ponieważ używają języków zapytań innych niż dotychczas stosowany SQL.

Model relacyjny sam w sobie nie określa sposobu, w jaki baza danych obsługuje jednoczesne żądania zmiany danych (tzw. transakcje). Aby zapewnić spójność i integralność danych, używany jest model transakcji ACID

- Atomicity (niepodzielność). Transakcji nie można podzielić – albo wszystkie instrukcje w transakcji są wykonane, albo żadna z nich.
- Consistency (spójność). Baza danych pozostaje w spójnym stanie przed i po wykonaniu transakcji.
- Isolation (izolacja). Chociaż wiele transakcji może być wykonywanych przez jednego lub więcej użytkowników jednocześnie, żadna nie powinna widzieć zmian wprowadznych przez inną (do momentu jej zakończenia).
- Durability (trwałość). Gdy transakcja zostanie zapisana (zatwierdzona), oczekuje się, że dokonane zmiany utrzymają się a system potrafi uruchomić się i udostępnić spójne, nienaruszone i aktualne dane.

Z jednej strony ACID wraz z relacjami jest źródłem siły relacyjnych baz danych. Z drugiej strony jest to źródło poważnych i bardzo trudnych do przezwyciężenia problemów.

- Transakcje ACID działające na poziomie bazy danych ułatwiają korzystanie z bazy danych.
- SQL jest ujednoliconym językiem wykorzystywanym w wielu bazach relacyjnych.
- Ograniczenia nakładane na kolumny pomagają sprawdzać poprawność danych przed ich dodaniem do bazy danych, co zwiększa spójność danych przechowywanych w bazie danych.

- Transakcje ACID mogą blokować system na krótki czas, co w niektórych zastosowanich może być dużą przeszkodą.
- Mapowanie relacyjno-obiektowe jest możliwe, ale może być złożone i dodaje do systemu jeszcze jedną warstwę pośrednią.
- Ze względu na operację JOIN relacyjne systemy bardzo źle skalują się.
- Trudno jest przechowywać dane o dużej różnorodności.

Czy SQL się zestarzał?

Dlaczego stworzono NoSQL? Czy można powiedzieć, że SQL się zestarzał? Nic bardziej mylnego. Technologie informatyczne to jedna z niewielu dziedzin, w których elementy systemu nie podlegają zmianom tylko i wyłącznie ze względu na ich wiek. Jedynym impulsem do zmiany jest użyteczność. Nigdy wiek. W przypadku SQL **musiał pojawić się nowy, nieznany czynnik**, zmuszający nas do porzucenia znanych technologii.

Zadanie stemplowania N kartek.

Zwiększenie wydajności – zatrudnienie **mega pracownika**.
Na jak długo to wystraczy (problem ograniczeń fizycznych).

Zwiększenie wydajności – podział zadań pośród wielu „normalnych” pracowników.

Zadanie numerowania N kartek liczbami naturalnymi od 1 do N.

Zwiększenie wydajności – tylko przez zatrudnienie **mega pracownika**.

- Mega pracownika – zwiększanie wydajności pojedynczego systemu – skalowanie pionowe.
- Wielu „normalnych” pracowników – zwiększanie wydajności systemu przez dodanie kolejnego (pod)systemu – skalowanie poziome.

Skalowanie poziome wydaje się być bardziej perspektywiczne. Jest „tylko” jeden mały, ale istotny szczegół: nasze zadanie powinno być podzielne na niezależne podzadania, a nie zawsze tak jest, jak widzieliśmy w zadaniu z numerowaniem stron. Powiemy, że zadanie stemplowania jest *zadaniem skalowalnym*, podczas gdy zadanie numerowania jest *zadaniem nieskalowalnym*.

Kiedy dane stają się problemem

Skalowalność – kiedy zaczyna być problemem

Kiedy zadanie nieskalowalne staje się dla nas problemem? Zauważmy, że konieczność skalowania występuje jedynie w przypadku dużych zbiorów danych. W przypadku małego N nie powinno być problemów z wykonaniem w rozsądnym czasie zarówno zadania stemplowania, jak i zadania numerowania i to tylko przez jednego człowieka. Gdy danych jest stosunkowo mało, wystarczą systemy nieskalowalne (ewentualnie skalowalne w pionie). Dlatego dane mogą stanowić problem, gdy **jest ich dużo**. Ale co to znaczy *dużo*? Czy dzisiaj mamy do czynienia z dużymi ilościami danych?

Ilość danych dziś

Fakty

Przyjmuje się, że dziś organizacje i użytkownicy na całym świecie wytwarzają ponad 2,5 EB (2^{60}, ponad 10^{18}, 5 bilionów DVD) danych dziennie. Dla porównania, Biblioteka Kongresu USA posiada obecnie ponad 300 TB (2^{40}, ponad 10^{12}, 65000 DVD) danych. Prawie każdy aspekt naszego życia jest lub może być źródłem danych (innym pytaniem jest, czy naprawdę ich potrzebujemy) i mogą być generowane zarówno przez ludzi (H), maszyny (M), jak i środowisko (E).

- (H) media społecznościowe, takie jak Facebook i Twitter,
- (H) eksperymenty naukowe i badawcze, takie jak symulacja fizyczna,
- (H) transakcje online, takie jak sprzedaż w punktach handlowych i bankowość,
- (M) czujniki, takie jak czujniki GPS, RFID, inteligentne liczniki i telematyka,
- (M) dowolne pracujące urządzenia monitorowane pod kątem naszego bezpieczeństwa, takie jak samoloty lub samochody,
- (E) warunki pogodowe, promieniowanie kosmiczne.

Ilość danych dziś

Fakty

W każdej minucie

- 4.166.667 likes made by Facebook users,
- 347.222 tweets on Twieeter,
- 100.040 cals on Skype,
- 77.166 hours of movies from Netflix,
- 694 Uber users take ride,
- 51.000 application are downloaded from AppStore.

Ilość danych dziś

Fakty

Dzisiaj zbieramy wszystkie dane, które możemy uzyskać, niezależnie od tego, czy naprawdę ich potrzebujemy, czy nie. Termin data lake (jezioro danych) został stworzony dla tego typu „kolekcji" danych. Działają jak jezioro z mnóstwem różnych rzeczy w środku, nie zawsze łatwo dostępnych i widocznych. Robimy to (zbieramy dane) niekoniecznie dlatego, że ich potrzebujemy, ale dlatego, że możemy. *Może kiedyś będziemy tego potrzebować* – myślimy. Z czasem duże ilości danych mogą ukryć to, co wcześniej przechowywaliśmy.

Ilość danych dziś

Fakty

Nawiązując do wcześniej przeprowadzonych doświadczeń, możemy stwierdzić, że N określające ilość danych dziś jest ogromne i z dnia na dzień staje się co raz większe.

Na domiar złego, nie dość, że danych jest dużo, to jeszcze:

- bardzo szybko przybywają nowe,
- cechują się ogromną różnorodnością.

To pozwala nam zdefiniować pojęcie dużego zbioru danych (Big Data) jako

Duży zbiór danych (Big Data)

Big data – termin odnoszący się do dużych, zmiennych i różnorodnych zbiorów danych, których przetwarzanie i analiza jest trudna ze względu na wymaganą prędkość przetwarzania.

Na domiar złego, nie dość, że danych jest dużo, to jeszcze:

- bardzo szybko przybywają nowe,
- cechują się ogromną różnorodnością.

To pozwala nam zdefiniować pojęcie dużego zbioru danych (Big Data) jako

Duży zbiór danych (Big Data)

Big data – termin odnoszący się do dużych, zmiennych i różnorodnych zbiorów danych, których przetwarzanie i analiza jest trudna ze względu na wymaganą prędkość przetwarzania.

Na domiar złego, nie dość, że danych jest dużo, to jeszcze:

- bardzo szybko przybywają nowe,
- cechują się ogromną różnorodnością.

To pozwala nam zdefiniować pojęcie dużego zbioru danych (Big Data) jako

Duży zbiór danych (Big Data)

Big data – termin odnoszący się do dużych, zmiennych i różnorodnych zbiorów danych, których przetwarzanie i analiza jest trudna ze względu na wymaganą prędkość przetwarzania.

Na domiar złego, nie dość, że danych jest dużo, to jeszcze:

- bardzo szybko przybywają nowe,
- cechują się ogromną różnorodnością.

To pozwala nam zdefiniować pojęcie dużego zbioru danych (Big Data) jako

Duży zbiór danych (Big Data)

Big data – termin odnoszący się do dużych, zmiennych i różnorodnych zbiorów danych, których przetwarzanie i analiza jest trudna ze względu na wymaganą prędkość przetwarzania.

Duży zbiór danych (Big Data)

Big data – termin odnoszący się do dużych, zmiennych i różnorodnych zbiorów danych, których przetwarzanie i analiza jest trudna ze względu na wymaganą prędkość przetwarzania.

Uwaga

Zauważmy, że aby mówić o *Big Data*, musimy mieć do czynienia z jednoczesnym wystąpienie wszystkich podanych wcześniej czynników. Jeśli wystąpią tylko niektóre z nich, to nie stanowi do dla nas obecnie problemu.

Ilość danych dziś

Duży zbiór danych (Big Data)

$$\text{Dane} \underset{N \to +\infty}{\longrightarrow} \text{Big Data}$$

Praktycznie za każdym razem, gdy chcemy uzyskać jakieś informacje, musimy połączyć dane rozproszone między wieloma tabelami w coś jednego. Wracając do poprzednio podanego przykładu z samochodem, za każdym razem, gdy chcemy się przejechać, musimy połączyć wszystkie elementy zebrane w małe szuflady i odbudować nasz samochód. Takie operacje łączenia, nazywane JOIN, są powtarzane wiele razy w ciągu minuty lub nawet sekundy. Potrzeba na to czasu, pamięci i zasobów procesora. **Naturalnym wydaje się chęć pozbycie się tego. Spróbujmy zorganizować nasze dane w sposób nie wymagający łączenia.**

NoSQL

NoSQL to zestaw pojęć, które umożliwiają szybkie i efektywne przetwarzanie zbiorów danych z naciskiem na wydajność, niezawodność i zwinność.

- Często nie mamy pojęcia ile kolumn będziemy potrzebować.
- Często nie mamy pojęcia jakiego rodzaju kolumny będziemy potrzebować.
- Często dodajemy kolumnę aby obsługiwać opcję, która zdarza się tylko raz na milion przypadków.
- Często struktura bazy ulega zmianie pod wpływem identyfikacji nowych, ważnych czynników lub redefiniowania tych już znanych (potrzeba zwinnego podejściem do danych związanego z dużą elastycznością w gromadzeniu, przechowywaniu i przetwarzaniu każdego rodzaju danych w dowolnym momencie gdy tylko się pojawią).
- Elastyczność to także redukcja kosztów.

Dostępność jest zwykle reprezentowana przez liczbę 9-tek

Dostępność	Czas niedostępności w miesiącu	Czas niedostępności w ciągu roku
95%	36 godzin	18 dni
99%	7 godzin	3.5 dni
99.5%	3.5 godzin	<2 dni
99.9%	43m i 12s	8h i 45m
99.99%	4m i 19s	52m i 36s
99.999%	25s	5m i 15s

Amazon S3 oferuje dostępność na poziomie 99.999999999 (11 9-tek).

Prędkość w bazach SQL ograniczana jest przez warunki nałożonymi na dane (tabele), dodatkowe reguły, oraz konieczność podziału każdego obiektu na najmniejsze części (formy normalne). A przecież **bardzo często złożoność relacyjna nie jest nam potrzebna**. W zamian za to mile widziana jest szybkość działania.

Skalowanie to jedyny sposób, w jaki możemy spełnić powyższe wymagania. Ważne, aby było to tak proste i przy jak najmniejszych dodatkowych kosztach, jak to tylko możliwe.

W zależności od potrzeb konkretnej aplikacji niektóre z tych czynników mogą być ważniejsze niż inne – to wyjaśnia, dlaczego rodzina NoSQL jest tak duża.

NoSQL

Bogactwo wyboru (Key-value stores)

NoSQL

Bogactwo wyboru (Document stores)

NoSQL

Bogactwo wyboru (Graph stores)

Bogactwo wyboru (Time series databases)

ACID = Atomicity (niepodzielność) + Consistency (spójność) + Isolation (izolacja) + Durability (trwałość).
BASE jest odpowiednikiem ACID w świecie NoSQL.

- **BAsic availability** oznacza, że baza danych jest cały czas dostępna kosztem chilowych niespójności.
- **Soft-state** oznacza, że stan systemu może się zmieniać w czasie, nawet bez danych wejściowych. Wynika to z kolejnej cechy, tj. eventual consistency.
- **Eventual consistency** oznacza, że pomimo tego, iż baza danych okresowo może nie być spójna, to gdy wszystkie jej serwisy zakończą pracę znajdzie się w stanie spójnym.

- Systemy ACID koncentrują się na spójności i integralności danych stawiając je ponad wszelkie inne cechy. Tymczasowe blokowanie to rozsądna cena, którą musimy zapłacić, aby zapewnić, że nasz system zwróci wiarygodne i dokładne informacje. **Systemy ACID są pesymistyczne**: zakładają, że jeśli coś może pójść nie tak, to – zgodnie z prawem Murphy'ego – na pewno tak się stanie.
- Systemy BASE koncentrują się na czymś zupełnie innym: na dostępności. **Systemy BASE są optymistyczne**: zakładają, że w ostatecznym rozrachunku wszystkie składowe systemu znajdą się w spójnym stanie.

- Systemy ACID koncentrują się na spójności i integralności danych stawiając je ponad wszelkie inne cechy. Tymczasowe blokowanie to rozsądna cena, którą musimy zapłacić, aby zapewnić, że nasz system zwróci wiarygodne i dokładne informacje. **Systemy ACID są pesymistyczne**: zakładają, że jeśli coś może pójść nie tak, to – zgodnie z prawem Murphy'ego – na pewno tak się stanie.
- Systemy BASE koncentrują się na czymś zupełnie innym: na dostępności. **Systemy BASE są optymistyczne**: zakładają, że w ostatecznym rozrachunku wszystkie składowe systemu znajdą się w spójnym stanie.

Twierdzenie CAP dotyczy zachowania się *rozproszonych* systemów bazodanowych w obliczu niestabilności sieci.

CAP, Eric Brewer (2000)

Każdy rozproszony system bazodanowy może mieć co najwyżej dwie z trzech następujących pożądanych właściwości

- **Consistency (spójność).** Spójność polega na tym, aby dla wszystkich klientów zawsze dostępna był tylko jedna, aktualna, taka sama wersja danych. Nasze dane powinny być spójne – bez względu na to, ilu klientów odczytuje te same elementy z replikowanych i rozproszonych partycji, powinniśmy otrzymywać spójne wyniki. Wszystkie zapisy są atomowe i wszystkie kolejne żądania pobierają nową wartość.

CAP, Eric Brewer (2000)

- **High availability (wysoka dostępność).** Ta właściwość stwierdza, że rozproszona baza danych zawsze pozwoli klientom na niezwłoczne wykonywanie operacji na niej. Błędy komunikacji wewnętrznej między częściami bazy nie powinny uniemożliwiać operacji na nich. Baza danych zawsze zwraca wartości tak długo, jak działa wymagana liczba serwerów.
- **Partition tolerance (odporność na partycjonowanie (?)).** Jest to zdolność systemu do ciągłego reagowania na żądania klientów, nawet w przypadku awarii komunikacji między partycjami (częściami) bazy danych. System będzie nadal działał, nawet jeśli tymczasowo utracona zostanie komunikacja sieciowa między partycjami.

- Porzucenie zasad ACID na rzecz BASE, co w wielu przypadkach jest ceną, jaką możemy zapłacić w zamian z wysoką dostępność systemu.
- Bez konieczności określania schematu danych (tzw. schema-less databases).
- Ułatwione przechowywanie dużych ilość danych cechujących się dużą zmiennością.
- W wielu przypadkach modułowa architektura umożliwia wymianę komponentów.
- Możliwe skalowanie liniowe w miarę dodawania do klastra nowych węzłów przetwarzających.
- Redukcja nakładów finansowych.

- Porzucenie zasad ACID na rzecz BASE, co w wielu przypadkach jest nieakceptowalne ze względu na ryzyko utraty danych lub ich spójności.
- Brak ujednoliconego języka zapytań na wzór SQL znanego z relacyjnych baz danych.
- Bardzo ubogie mechanizmy bezpieczeństwa i kontroli.

Ilość danych dziś

$$\text{Big Data} \longrightarrow \text{! SQL} \longrightarrow \text{NoSQL}$$

ale

$$\text{NoSQL} \neq \text{Big Data}$$

$$\text{Big Data} \longrightarrow \text{! SQL} \longrightarrow \text{NoSQL}$$

ale

$$\text{NoSQL} \neq \text{Big Data}$$

Jest skalowalność – dobrze, brak ACID – źle, brak SQL – źle.
Poszukajmy czegoś co będzie miało obie te cechy.

Termin NewSQL po raz pierwszy został użyty w 2011 roku.
Podobnie jak w przypadku NoSQL nie należy rozumieć go dosłownie – określa raczej nowe cele (czy aby faktcznie są one nowe i ile faktycznie jest w tym nowości?) jakie stawiamy przed systemami bazodanowymi.

- Opracowanie (nowych) produktów i usług związanych z relacyjnymi bazami danych, ale zdolnych do działania w architekturze rozproszonej (łatwa skalowalność).
- Poprawa wydajności relacyjnych baz danych w takim stopniu, aby skalowalność pozioma nie była już konieczna.

Termin NewSQL po raz pierwszy został użyty w 2011 roku.
Podobnie jak w przypadku NoSQL nie należy rozumieć go dosłownie – określa raczej nowe cele (czy aby faktcznie są one nowe i ile faktycznie jest w tym nowości?) jakie stawiamy przed systemami bazodanowymi.

- Opracowanie (nowych) produktów i usług związanych z relacyjnymi bazami danych, ale zdolnych do działania w architekturze rozproszonej (łatwa skalowalność).
- Poprawa wydajności relacyjnych baz danych w takim stopniu, aby skalowalność pozioma nie była już konieczna.

Termin NewSQL po raz pierwszy został użyty w 2011 roku.
Podobnie jak w przypadku NoSQL nie należy rozumieć go dosłownie – określa raczej nowe cele (czy aby faktcznie są one nowe i ile faktycznie jest w tym nowości?) jakie stawiamy przed systemami bazodanowymi.

- Opracowanie (nowych) produktów i usług związanych z relacyjnymi bazami danych, ale zdolnych do działania w architekturze rozproszonej (łatwa skalowalność).
- Poprawa wydajności relacyjnych baz danych w takim stopniu, aby skalowalność pozioma nie była już konieczna.

- Łatwa skalowalność. Wydaje się trudne – była to jedna z przyczyn tworzenia baz danych NoSQL: niemożność skalowania baz danych SQL. Skoro kiedyś nie było można, to dlaczego niby teraz ma się to udać?
- Poprawa wydajności. Wydaje się niemożliwe – była to jedna z przyczyn tworzenia baz danych NoSQL: niezdolność do zwiększenia (lub zmniejszenia w razie potrzeby) mocy obliczeniowej pojedynczej jednostki z jednej strony z elastycznością wymaganą przez dzisiejszy świat biznesu a z drugiej bez ponoszenia zbędnie wysokich kosztów.

- SQL jako podstawowy mechanizm interakcji.
- Obsługa ACID dla transakcji.
- Nieblokujący mechanizm kontroli współbieżności, dzięki czemu odczyty w czasie rzeczywistym nie będą powodować konfliktów z zapisami, a tym samym powodować ich blokowania.
- Architektura zapewniająca znacznie wyższą wydajność pojedyńczego węzła niż dotychczas dostępna w tradycyjnych rozwiązaniach RDBMS (SQL).
- Skalowalna, współdzielona architektura, która może działać na dużej liczbie węzłów.

Czy aby termin NewSQL nie jest tylko chwytem marketingowym?
Lista cech wygląda jak przedwyborcze obietnice polityków.

Najlepszym testem słuszności pewnych idei i przyjętych założeń jest praktyka a ta nie jest korzystna dla NewSQL.
Zauważmy, że NoSQL bardzo szybko został przyjęty i zaadoptowany na rynku – trafił w realne potrzeby.
NewSQL ma z tym problemy. Komu potrzebny jest NewSQL? Małym firmom wystarcza SQL.

- Duże są zbyt konserwatywne w myśl dobrze znanego powiedzenia informatyków *jak coś działa, to nie ruszać*. Raczej mało prawdopodobne jest aby zdecydoway się na zastopienie istniejących rozwiązań SQL rozwiązaniami NewSQL – koszty mogą być zbyt wysokie. Pamiętajmy, że wciąż nie znamy odpowiedzi na pytanie, ile faktycznie nowego jest w NewSQL i ile faktycznie to będzie nas kosztowało.
- Utrzymywanie dwóch systemów, tj. SQL i NewSQL nie ma sensu ze względu na niekorzystny stosunek poniesionych kosztów do osiągniętych zysków.

Najlepszym testem słuszności pewnych idei i przyjętych założeń jest praktyka a ta nie jest korzystna dla NewSQL.
Zauważmy, że NoSQL bardzo szybko został przyjęty i zaadoptowany na rynku – trafił w realne potrzeby.
NewSQL ma z tym problemy. Komu potrzebny jest NewSQL? Małym firmom wystarcza SQL.

- Duże są zbyt konserwatywne w myśl dobrze znanego powiedzenia informatyków *jak coś działa, to nie ruszać*. Raczej mało prawdopodobne jest aby zdecydoway się na zastopienie istniejących rozwiązań SQL rozwiązaniami NewSQL – koszty mogą być zbyt wysokie. Pamiętajmy, że wciąż nie znamy odpowiedzi na pytanie, ile faktycznie nowego jest w NewSQL i ile faktycznie to będzie nas kosztowało.
- Utrzymywanie dwóch systemów, tj. SQL i NewSQL nie ma sensu ze względu na niekorzystny stosunek poniesionych kosztów do osiągniętych zysków.

Najlepszym testem słuszności pewnych idei i przyjętych założeń jest praktyka a ta nie jest korzystna dla NewSQL.
Zauważmy, że NoSQL bardzo szybko został przyjęty i zaadoptowany na rynku – trafił w realne potrzeby.
NewSQL ma z tym problemy. Komu potrzebny jest NewSQL? Małym firmom wystarcza SQL.

- Duże są zbyt konserwatywne w myśl dobrze znanego powiedzenia informatyków *jak coś działa, to nie ruszać*. Raczej mało prawdopodobne jest aby zdecydoway się na zastopienie istniejących rozwiązań SQL rozwiązaniami NewSQL – koszty mogą być zbyt wysokie. Pamiętajmy, że wciąż nie znamy odpowiedzi na pytanie, ile faktycznie nowego jest w NewSQL i ile faktycznie to będzie nas kosztowało.
- Utrzymywanie dwóch systemów, tj. SQL i NewSQL nie ma sensu ze względu na niekorzystny stosunek poniesionych kosztów do osiągniętych zysków.

- SQL jako podstawowy mechanizm interakcji.
- Obsługa ACID dla transakcji.
- Nieblokujący mechanizm kontroli współbieżności.
- Znacznie wyższa wydajność na węzeł niż dostępna w tradycyjnych rozwiązaniach RDBMS (SQL).
- Skalowalność.

- Jeśli w przypadku NewSQL możemy uzyskać znacznie wyższą wydajność na węzeł niż dotychczas dostępna w porównaniu z tradycyjnymi rozwiązaniami RDBMS (SQL), to możemy również zastosować analogiczne podejście do istniejących systemów SQL, zmniejszając tym samym przewagę NewSQL nad SQL w tym aspekcie.
- Powolne wdrażanie NewSQL w porównaniu z NoSQL jest znaczącym znakiem, który należy potraktowc jako ostrzeżenie. Może to świadczyć o nieadekwatności proponowanych rozwiązań w stosunku do faktycznych potrzeb.

Oracle, MySql, Microsoft SQL Server, PostgreSQL, IBM Db2, Microsoft Access, SQLite.

- **Column family stores**: Amazon SimpleDB, Accumulo, Cassandra, Druid, HBase, Hypertable, Vertica.
- **Key-value stores**: Apache Ignite, ArangoDB, BerkeleyDB, Dynamo, FoundationDB, InfinityDB, LevelDB, MemcacheDB, MUMPS, Oracle NoSQL Database, OrientDB, Project Voldemort, Redis, Riak, Berkeley DB.
- **Document stores**: Apache CouchDB, ArangoDB, BaseX, Clusterpoint, Couchbase, Cosmos DB, IBM Domino, MarkLogic, MongoDB, OrientDB, Qizx, RethinkDB.
- **Graph stores**: Apache Giraph, MarkLogic, Neo4J, OrientDB, Virtuoso.
- **Column databases**: Druid, Vertica.
- **Time series databases**: Druid, InfluxDB, OpenTSDB, Riak-TS, RedisTimeSeries.

Podsumowanie

Przykłady baz NewSQL

Clustrix, GenieDB, ScalArc, Schooner, VoltDB, RethinkDB, ScaleDB, Akiban, CodeFutures, ScaleBase, Translattice, NimbusDB, Drizzle, MySQL Cluster with NDB, and MySQL with HandlerSocket, Tokutek, JustOne DB, Amazon Relational Database Service, Microsoft SQL Azure, Xeround, Database.com, FathomDB