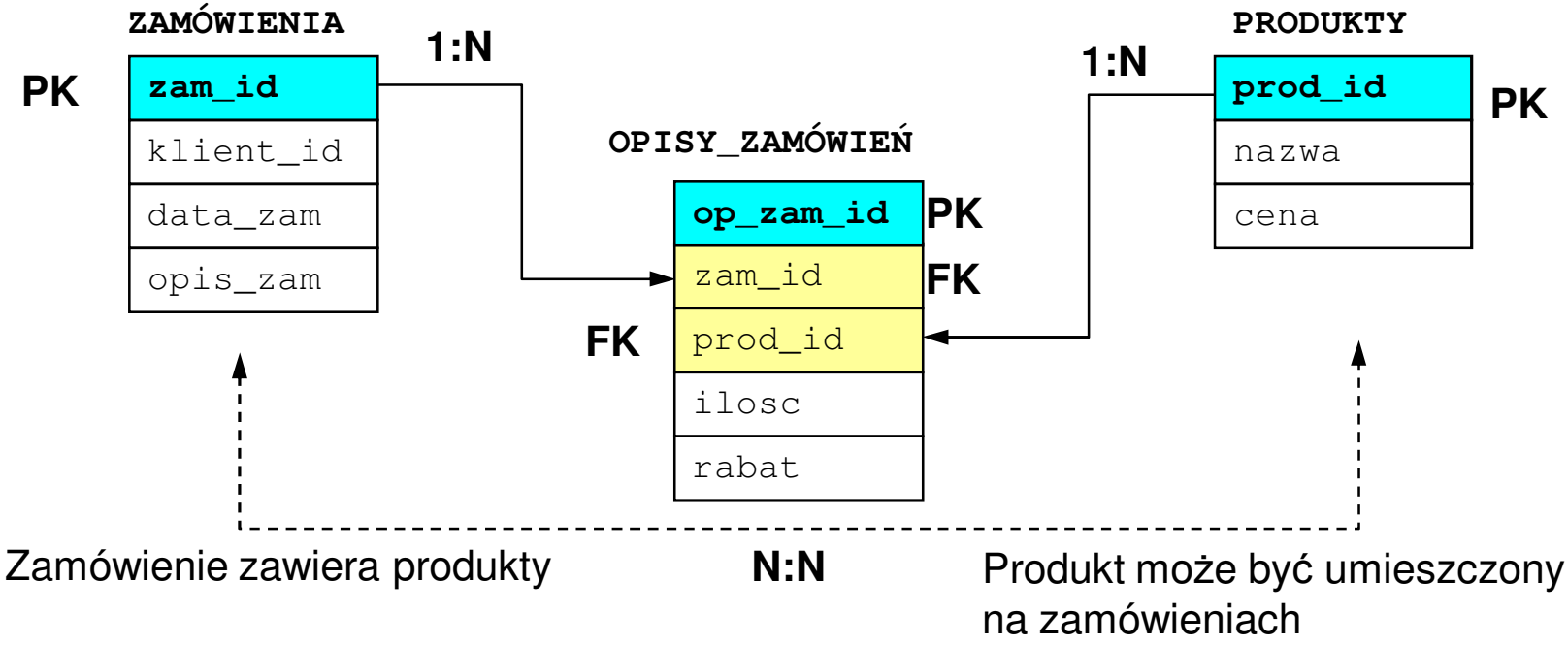
Bazy danych NoSQL

Artur Gramacki
a.gramacki@issi.uz.zgora.pl
Uniwersytet Zielonogórski
Instytut Sterowania i Systemów Informatycznych
2020

Bazy relacyjne

- Dominującym modelem nadal wydaje się model relacyjny
  - tabele, relacje (związki) między tabelami 1:1, 1:N:, M:N, klucze główne i obce, ograniczenia bazodanowe



Bazy relacyjne

- Bazy relacyjne powstały w latach 70-tych, przez wiele lat nie miały praktycznie żadnej poważnej konkurencji
- W latach 90-tych chwilowo pojawiły się bazy obiektowe, ale świat szybko o nich zapomniał
  - w sumie nie wniosły do baz danych nic na tyle interesującego, aby można je było potraktować, jako następcę baz relacyjnych
- Wiek XXI to niesamowity rozwój baz NoSQL
  - to dobrze i zarazem źle
  - trudno poruszać się w gąszczu rozwiązań
  - niepewność, co do systematycznego i długofalowego rozwoju konkretnych rozwiązań

Bazy relacyjne

- Zalety baz relacyjnych
  - niezwykle mocno ustandaryzowany model relacyjny
    - każda baza relacyjna działa „tak samo”
  - ustabilizowana technologia, gwarantuje bezpieczne przechowywanie danych na bardzo wiele lat, niezależnie od zmieniających się trendów i mód w metodach i technikach wytwarzania aplikacji opartych o bazy danych
  - niezwykle mocno ustandaryzowany język SQL
    - bazy typu NoSQL jak na razie nie oferują nic równie elastycznego, prostego i wygodnego w użyciu
  - współbieżność – jednoczesna praca wielu użytkowników, którzy są nawzajem dla siebie niewidoczni (perfekcyjna niemal obsługa transakcji)

Bazy relacyjne

- Wady baz relacyjnych
  - dane przechowywane są w tabelach (wiersze, kolumny), co skutkuje tym, że raz ustalony model trudno jest zmienić, bez konieczności często bardzo dużych zmian w aplikacjach
  - brak struktur zagnieżdżonych (ale łatwo to realizować projektując odpowiednie relacje między tabelami)
  - dane przechowywane są (a przynajmniej tak powinny być przechowywane) w postaci znormalizowanej. Jest to w sumie i wada, i zaleta
  - każdy obiekt w relacji (inaczej: wiersz w tabeli) z definicji musi być opisany identycznymi atrybutami (inaczej: kolumny w tabeli). Często jest to zbyt mało elastyczne podejście
  - podział bazy relacyjnej na klastry jest możliwy ale są to bardzo nienaturalne działania
  - model danych relacyjnych ma się nijak do fizycznej organizacji danych systemie plikowym

Krótki wtręt o normalizacji relacji

PRODUKTY

prod_id	opis_prod
10	toner
20	pamięć
30	pendrive
40	HD
50	drukarka

ZAMÓWIENIA

zam_id	klient_id	data
1	25	06.10.2001
2	50	13.07.1999
3	75	14.08.1995
4	100	20.12.2003
5	125	11.11.2000

KLIENCI

klient_id	nazwisko
25	Gramacki
50	Nowak
75	Pawlak
100	Kowalski
125	Barski

OPISY_ZAMÓWIEŃ

zam_id	ilosc	prod_id
1	1	2
1	2	10
1	3	1
2	1	10
3	1	1
3	2	2
4	1	100
5	1	1
5	2	1

Krótki wtręt o normalizacji relacji

prac_id	imie	nazwisko	dzial_id	nazwa_dzialu
1	Artur	Nowakowski	1	serwis
2	Jan	Kowalski	1	serwis
3	Roman	Nowak	2	księgowość
4	Stefan	Antkowiak	3	obsługa klienta
5	Ewa	Konieczna	3	obsługa klienta
6	Anna	Wojtasik	3	obsługa klienta
7	Marek	Pawlak	1	serwis

- Przykładowe anomalie
  - anomalia wstawiania: nie można wstawić nazwy działu, który nie zatrudnia (na razie) żadnych pracowników
  - anomalia wstawiania: wstawiając nową nazwę działu trzeba zawsze dbać o spójność z polem dzial_id
  - anomalia usuwania: po wykasowaniu pracownika o numerze 3 tracimy informację, że kiedykolwiek istniał dział księgowości
  - anomalia modyfikowania: zmiana nazwy działu z „serwis” na „serwis i sprzedaż” wymaga zmiany w trzech miejscach (tu: rekordach)
  - redundancja danych: wielokrotnie wpisane są te same nazwy działów

Krótki wtręt o normalizacji relacji

Zamówienie nr: **001**	
Klient: **Gramacki**	Identyfikator klienta: **25**
Data złożenia zamówienia: **06.10.2018**	
Opis zamówienia:	

Pozycja	Numer kat.	Ilość sztuk
toner	**10**	**2**
pendrive	**30**	**10**
HD	**40**	**1**

wartości nie są elementarne

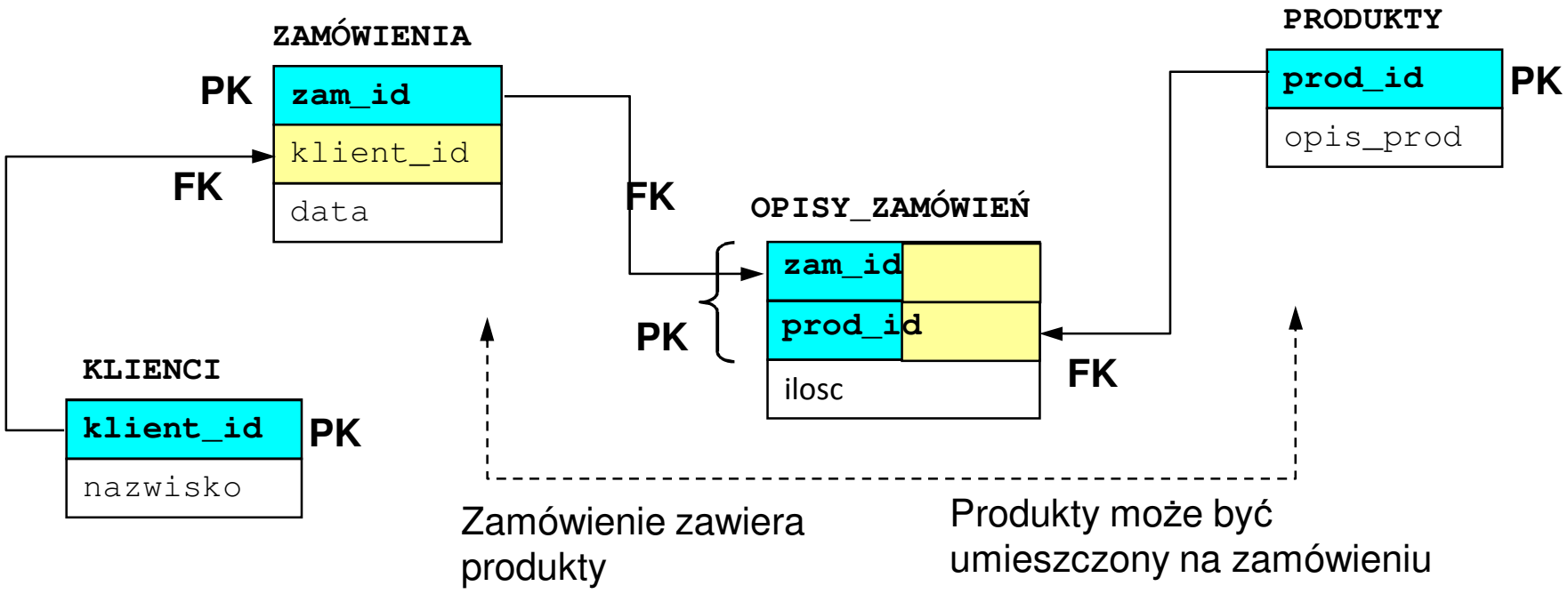
zam_id	klient_id	nazwisko	data	opis_zam
001	25	Gramacki	06.10.2018	2 tonery, 10 pendrive-ów, 1 dysk twardy
002	50	Nowak	13.07.2017	10 pendrive-ów
003	75	Pawlak	14.08.2008	1 drukarka, 2 dyski twarde
004	100	Kowalski	20.12.2017	100 pendrive-ów
005	125	Barski	11.11.2018	1 drukarka, 1 pamięć

dużo pustych komórek

zam_id	klient_id	nazwisko	data	poz_1	ilosc_1	poz_2	ilosc_2	poz_3	ilosc_3
001	25	Gramacki	06.10.2018	toner	2	pendrive	10	HD	1
002	50	Nowak	13.07.2017	pendrive	10				
003	75	Pawlak	14.08.2008	drukarka	1	HD	2		
004	100	Kowalski	20.12.2017	pendrive	100				
005	125	Barski	11.11.2018	drukarka	1	pamięć	1		

istnieją powtarzające się grupy

Krótki wtręt o normalizacji relacji

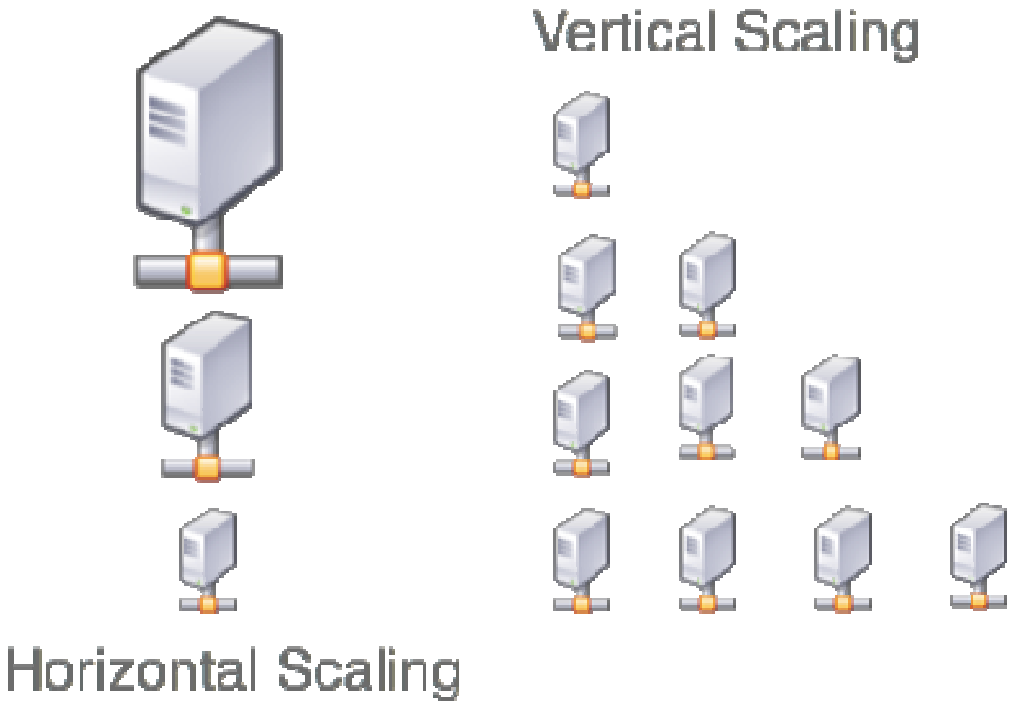


Big data

- W wielu współczesnych zastosowaniach, które przetwarzają wielkie ilości danych (BigData) oraz działają w architekturach rozproszonych, model relacyjny okazuje się niezbyt wygodny
- Podstawowe problemy
  - wielkie bazy danych wymagają równie wielkich serwerów, a te naprawdę wielkie są też bardzo drogie i kosztowne w utrzymaniu
  - trudności w efektywnej obsłudze wielkich ilości transakcji (często współbieżnych)
  - trudności w szybkim reagowaniu na zmiany w strukturze danych, które często są bardzo szybko zmieniające się w czasie
  - trudności w obsłudze systemów rozproszonych
  - trudności w skalowalności (przykład: sklep internetowy w gorących okresach przedświątecznych notuje wielokrotnie większe obciążenie, niż w pozostałych okresach)
  - wydajność może okazać się niewystarczająca, gdy np. lawinowo przyrasta ilość gromadzonych danych
  - wielkie (miliony+) ilości użytkowników
  - zapewnienie dostępności do danych, nawet w przypadku awarii

Big data

- Dwa rodzaje skalowania systemów
  - skalowanie pionowe (zwiększanie mocy pojedynczych serwerów)
  - skalowanie poziome (łączenie coraz to większej liczby słabszych maszyn w klastry)



Big data

- W przypadku wielkich systemów baz danych (SZBD) należy więc zwracać uwagę na 4 kluczowe elementy
  - skalowalność
  - koszt
  - elastyczność
  - dostępność
- Użycie baz NoSQL może bardzo pomóc w racjonalizacji powyższych elementów
- Nie należy bezrefleksyjnie odchodzić od baz relacyjnych, zastępując je całkowicie bazami nierelacyjnymi
  - projektanci i programiści baz danych MUSZĄ znać zarówno bazy relacyjne, jak i NoSQL aby umieć ocenić, który model jest bardziej odpowiedni do danego zadania

Bazy relacyjne / nierelacyjne

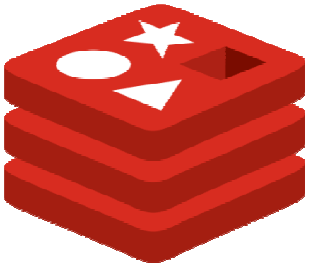
- Każda relacyjna baza danych oparta jest dokładnie na tych samych założeniach (teoria relacji)
- Implementacje różnych producentów, co do pryncypiów, są identyczne (model relacyjny)
  - systemy różnych producentów różnią się bardzo wieloma szczegółami technicznymi, jednak to są ciągle dobrze znane bazy relacyjne
  - Oracle, MySQL, SQL Server, DB2, Sybase, PostgreSQL …
- Wszystkie inne rodzaje baz danych będziemy ogólnie nazywać bazami nierelacyjnymi
  - czyli nie opartymi na tabelach i powiązaniach między nimi
  - będziemy je nazywać bazami NoSQL

Bazy relacyjne / nierelacyjne

- Małe ostrzeżenie:
  - nie próbujmy na siłę tworzyć systemów opartych o bazy NoSQL, tam gdzie nie jest to naprawdę dobrze uzasadnione
  - nie próbujmy na siłę tworzyć wyłącznie systemów opartych o bazy SQL, tam gdzie zasadne może być użycie baz NoSQL
- Wydaje się, że obecnie bazy NoSQL nie wyprą całkowicie baz relacyjnych a raczej będą je uzupełniać i zastępować tam, gdzie będzie to przynosiło realne korzyści

Bazy NoSQL

- Najpopularniejsze bazy NoSQL



redis

Bazy NoSQL

- Bazy wymienione na stronie Wikipedii

Model danych	Przykładowe bazy danych
Klucz - wartość	Aerospike, Apache Ignite, ArangoDB, BerkeleyDB, Couchbase, Dynamo, FairCom c-treeACE, FoundationDB, InfinityDB, LevelDB ,MemcacheDB, MUMPS, Oracle NoSQL Database, OrientDB, Project Voldemort, Redis, Riak, Berkeley DB, SDBM/Flat File dbm, ZooKeeper
Dokument	Apache CouchDB, ArangoDB, BaseX, Clusterpoint, Couchbase, Cosmos DB, IBM Domino, MarkLogic, MongoDB, OrientDB, Qizx, RethinkDB
Rodzina kolumn	Amazon SimpleDB, Accumulo, Cassandra, Druid, HBase, Hypertable, Vertica.
Graf	AllegroGraph, ArangoDB, InfiniteGraph, Apache Giraph, MarkLogic, Neo4J, OrientDB, Virtuoso
Multi-model	Apache Ignite, ArangoDB, Couchbase, FoundationDB, InfinityDB, MarkLogic, OrientDB

Bazy NoSQL

- Tym terminem będziemy po prostu oznaczać wszystkie bazy, które nie są bazami relacyjnymi
  - przeważnie są to bazy typu open-source
  - są to rozwiązania powstałe w ostatnich kilku latach XXI wieku
  - nie posiadają klasycznego języka SQL (ale np. QCL w systemie Cassandra)
- Prawdopodobnie nie ma co roztrząsać pochodzenia i znaczenia tej nazwy. Termin ten przyjął się, nikt do tej pory nie podał sensownej alternatywy, temat nazwy więc zamykamy

Bazy NoSQL

- Pozwalają na przechowywanie danych bez schematu
  - może to być i wada i zaleta
  - niebezpieczeństwo „zabałaganienia” danych, a co za tym idzie również aplikacji
- Są dużo łatwiejsze w skalowaniu
  - niektórzy uważają, że też łatwiejsze w programowaniu, ale to zapewne jest bardzo subiektywna ocena
- Brak wsparcia dla znanej z relacyjnych baz danych bardzo rygorystycznej spójności danych
  - w bazach NoSQL spójność (consistency) jest powszechnie poświęcana na rzecz zapewnienia wysokiej dostępności do danych i szybkości działania systemu bazodanowego
  - utrzymanie spójności w systemach rozproszonych jest niezwykle trudne do osiągnięcia
  - czasami nie ma możliwości zrezygnowania ze spójności (np. konta bankowe, dane finansowe, dane ubezpieczeniowe)
  - w wielu systemach rozproszonych utrzymanie spójności danych nie jest potrzebne a na pewno nie jest priorytetem

Zapytania w bazach danych

- W bazach relacyjnych język SQL to absolutna dominacja
  - język bardzo dobrze ustandaryzowany
  - umożliwia wyciąganie danych w praktycznie każdym żądanym przez użytkownika układzie
    - … a jeżeli czegoś nie można wykonać wprost w SQL-u, to zawsze można wspomóc się programowym dostępem do danych (PHP, Python i wiele innych)
- Bazy NoSQL nie posiadają sformalizowanego języka zapytań
  - dostęp do danych w zasadzie głownie programowy
  - zaczynają pojawiać się pewne rozwiązania „SQL-podobne”, ale póki co, nie zbliża się to nawet to funkcjonalności SQL-a
- Bazy NoSQL nie spełniają założeń określanych akronimem ACID
  - akronim czterech właściwości baz relacyjnych
- Bazy NoSQL wspierają natomiast właściwości BASE
  - często przedstawiane jako opozycja dla ACID lub kompromis dla ACID

ACID

- Zbiór właściwości gwarantujących poprawne przetwarzanie transakcji w bazach danych
- Niepodzielność (ang. **A**tomicity)
  - niepodzielność transakcji, oznacza, że każda transakcja albo zostanie wykonana w całości, albo w całości zostanie anulowana
- Spójność (ang. **C**onsistency)
  - spójność transakcji oznacza, że po wykonaniu transakcji system będzie spójny, czyli nie zostaną naruszone zasady integralności
- Izolacja (ang. **I**solation)
  - izolacja transakcji oznacza, że jeśli dwie transakcje wykonują się współbieżnie, to zwykle (w zależności od poziomu izolacji) nie widzą wprowadzanych przez siebie zmian. Poziom izolacji w bazach danych jest zazwyczaj konfigurowalny i określa, jakich anomalii możemy się spodziewać przy wykonywaniu transakcji
- Trwałość (ang. **D**urability)
  - system potrafi uruchomić się i udostępnić spójne, nienaruszone i aktualne dane zapisane w ramach zatwierdzonych transakcji (też po np. niespodziewanych awariach zasilania)

https://pl.wikipedia.org/wiki/ACID

ACID

- Poziomy izolacji transakcji
  - read uncommitted – jedna transakcja może odczytywać wiersze, na których działają inne transakcje (najniższy poziom izolacji)
  - read committed – transakcja może odczytywać tylko wiersze zapisane
  - repeatable read – transakcja nie może czytać ani zapisywać na wierszach odczytywanych lub zapisywanych w innej transakcji
  - serializable (szeregowalne) – wyniki współbieżnie realizowanych zapytań muszą być identyczne z wynikami tych samych zapytań realizowanych szeregowo (pełna izolacja)

https://pl.wikipedia.org/wiki/ACID

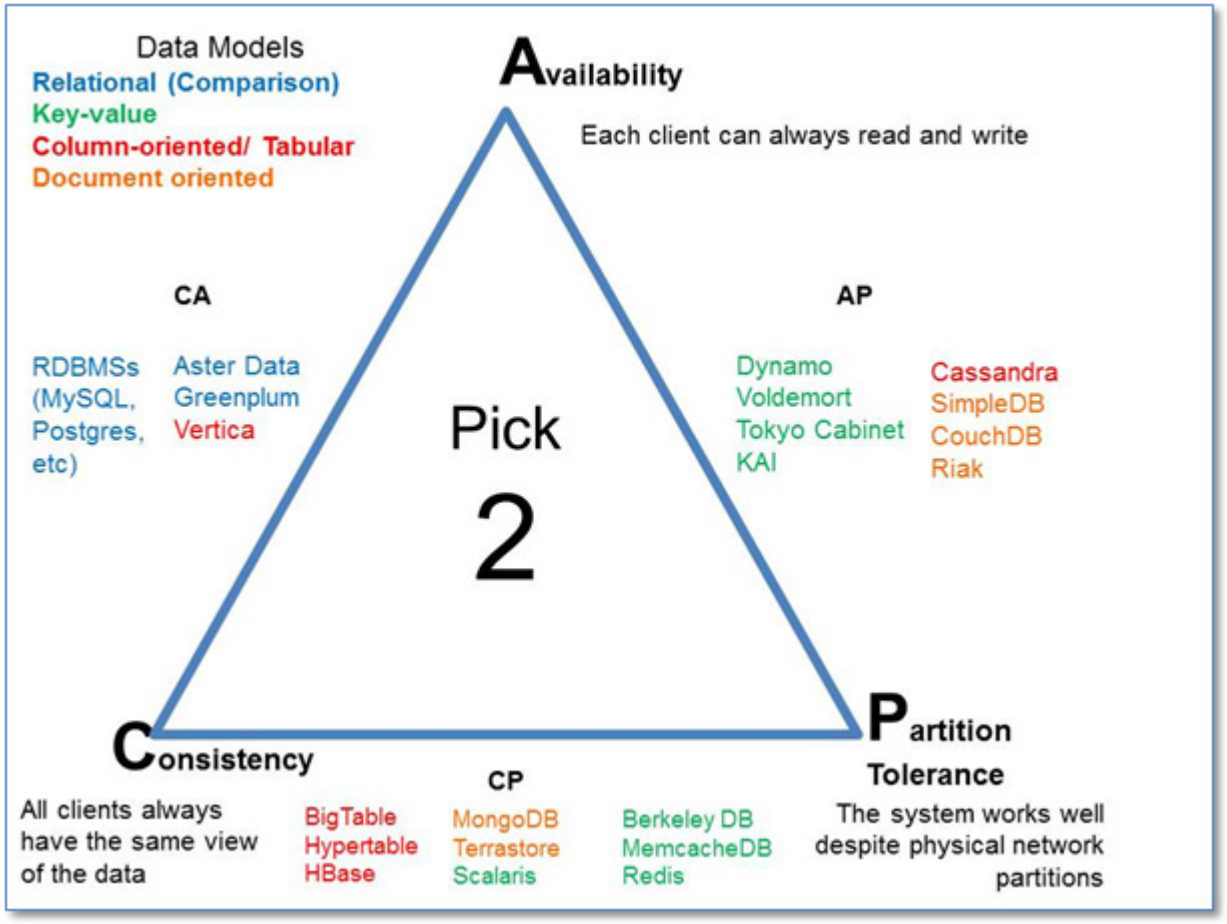
BASE

- UWAGA: w dostępnych źródłach nie ma jakiejś jednej uznanej definicji. Wielu autorów dość swobodnie definiuje BASE
- Zasadnicza dostępność (ang. **B**asically **A**vailable)
  - system nie gwarantuje zawsze pełnej dostępności
- Miękki stan (ang. **S**oft state)
  - wskazuje, że stan systemu może zmieniać się w czasie, nawet bez wprowadzania zmian w danych
- Ostateczna spójność (ang. **E**ventually consistent)
  - wskazuje, że system po pewnym czasie stanie się spójny (np. że dwa węzły po pewnym czasie zostaną zsynchronizowane), nawet w sytuacji, gdy żadne dane nie są do systemu wprowadzane/zmieniane/kasowane.Dane mogą stać się niespóje np. w przypadku czasowej niedostępności jednego z węzłów

Twierdzenie CAP

- W dużym systemie rozproszonym możliwe jest spełnienie tylko dwóch z trzech ważnych charakterystyk
  - spójności (ang. **C**onsistency) – wszyscy klienci mają zawsze ten sam obraz danych (dane są zawsze spójne)
  - dostępności (ang. **A**vailability) – system jest zawsze dostępny. Każdy klient może zawsze odczytywać i zapisywać dane, choć nie ma gwarancji, że odczytuje dane w najbardziej aktualnej postaci
  - ochrony przed partycjonowaniem (ang. **P**artition tolerance) – system działa mimo, że jest problem komunikacyjny między serwerami (węzłami klastra)
- Teoria CAP potwierdza intuicyjny fakt, że nie ma idealnej bazy danych do wszystkich zastosowań i każda z nich jest pewnym kompromisem, kładącym nacisk na pewne własności, kosztem innych
- Trójkąt CAP

Trójkąt CAP



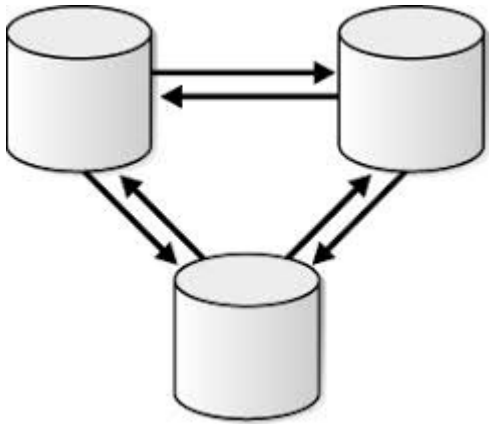
Rysunek ze strony: http://java-questions.com/cap-theorem.html

Trójkąt CAP

- Tradycyjne relacyjne bazy danych (MySQL, PostgreSQL itp.) lokują się na boku CA trójkąta
  - kładą nacisk na spójność i dostępność danych ale skalują się przede wszystkim pionowo
- Pozostałe boki obstawione są przede wszystkim przez bazy NoSQL
  - Baza Cassandra jest przykładem bazy AP, która nastawiona jest na skalowanie poziome z równoczesną odpornością na awarie i wysoką dostępnością, przy czym nie mamy gwarancji, że odczytujemy najbardziej aktualne dane
  - MongoDB, którą można zaklasyfikować jako CP, czyli pozwala na rozproszenie z zachowaniem integralności danych, lecz w przypadku awarii węzła, należy liczyć się z brakiem dostępu do części danych

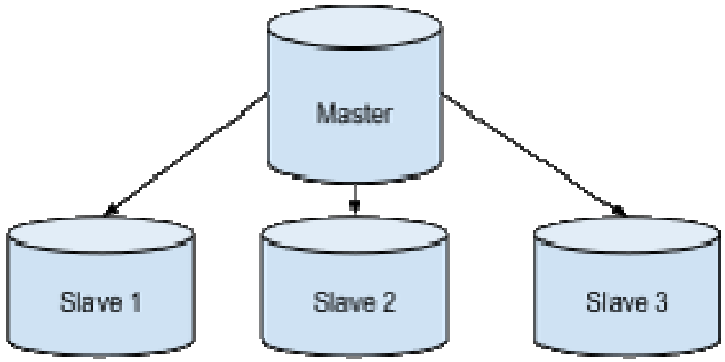
Replikacja peer-to-peer

- Replikacja peer-to-peer umożliwia zapis do dowolnego węzła
- Każdy węzeł koordynuje synchronizację danych z innymi węzłami
- Brak serwera głównego
- Zwiększa się dostępność ale zgoda na niespójność
  - ale można pilnować spójności kosztem mniejszej dostępności



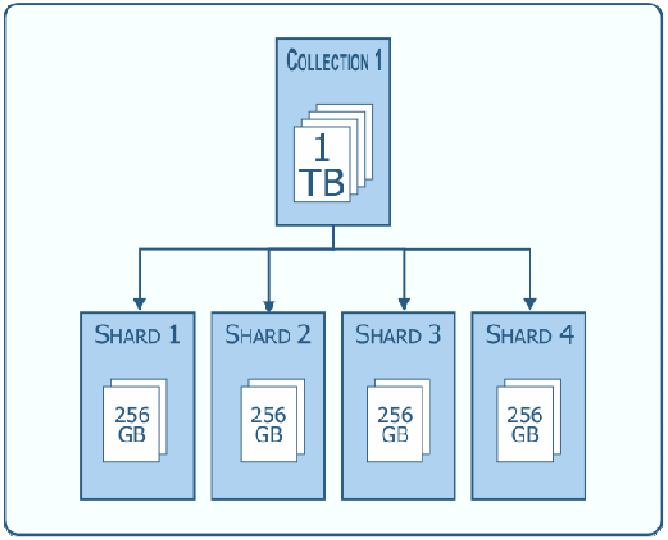
Replikacja master-slave

- Zapis tyko przez węzeł typu *master*
- Węzeł typu *master* jest odpowiedzialny za synchronizację danych pomiędzy węzłami typu *slave*
- Odczyt z dowolnego węzła
- Zapewnia dużą niezawodność odczytu
  - dobrze się nadaje do systemów, gdzie głównie odczytuje się dane
- Problemy ze spójnością



Sharding (współdzielenie)

- Różne części danych na różnych serwerach
  - np. 10 serwerów, każdy obsługuje 10% zapytań (np. podział na kraje, dni tygodnia, kategorie produktów, itp.)
  - albo różnych obszarach tego samego serwera
- Zwiększa szybkość zapisu i odczytu
- Nie poprawia niezawodności i bezpieczeństwa danych,



Map-Reduce

- Dwa rozumienia pojęcia Map-reduce
  - platforma (framework, biblioteka) do przetwarzania równoległego dużych zbiorów danych w klastrach komputerów stworzona przez firmę Google (popularność zdobył dzięki platformom takim jak Hadoop czy Spark)
  - metodologia (wzorzec) przetwarzania dużych zbiorów danych w środowisku klastrów. Wykorzystywany jest wszędzie tam, gdzie dane liczy się w terabajtach
- Wydaje się, że Map-Reduce to znana od wielu lat koncepcja przetwarzania równoległego o nazwie Master-Slave „odświeżona” pod nową nazwą i wzbogacona o rozwiązania hardware-owe

Map-Reduce

- Po co?
  - gdy danych nie ma zbyt dużo zwykła baza relacyjna i zwykły SQL bez problemu poradzi sobie z grupowanie danych w stylu

```
SELECT
    branza, SUM(wartosc_zamowienia)
FROM
    zamowienia
GROUP BY
    branza;
```

  - dopóki ilość danych i czas w jakim zapytanie się wykonuje mieszczą się w granicach wymagań biznesu, to wszystko jest ok!
  - problem zaczyna się wtedy, gdy wydajność maleje, bo jeden serwer SQL nie radzi sobie z przetwarzaniem coraz szybciej i ciągle napływających danych
  - oczywiste rozwiązanie: zrównoleglić i rozproszyć obliczenia. Do tego sprowadza się metodologia map-reduce

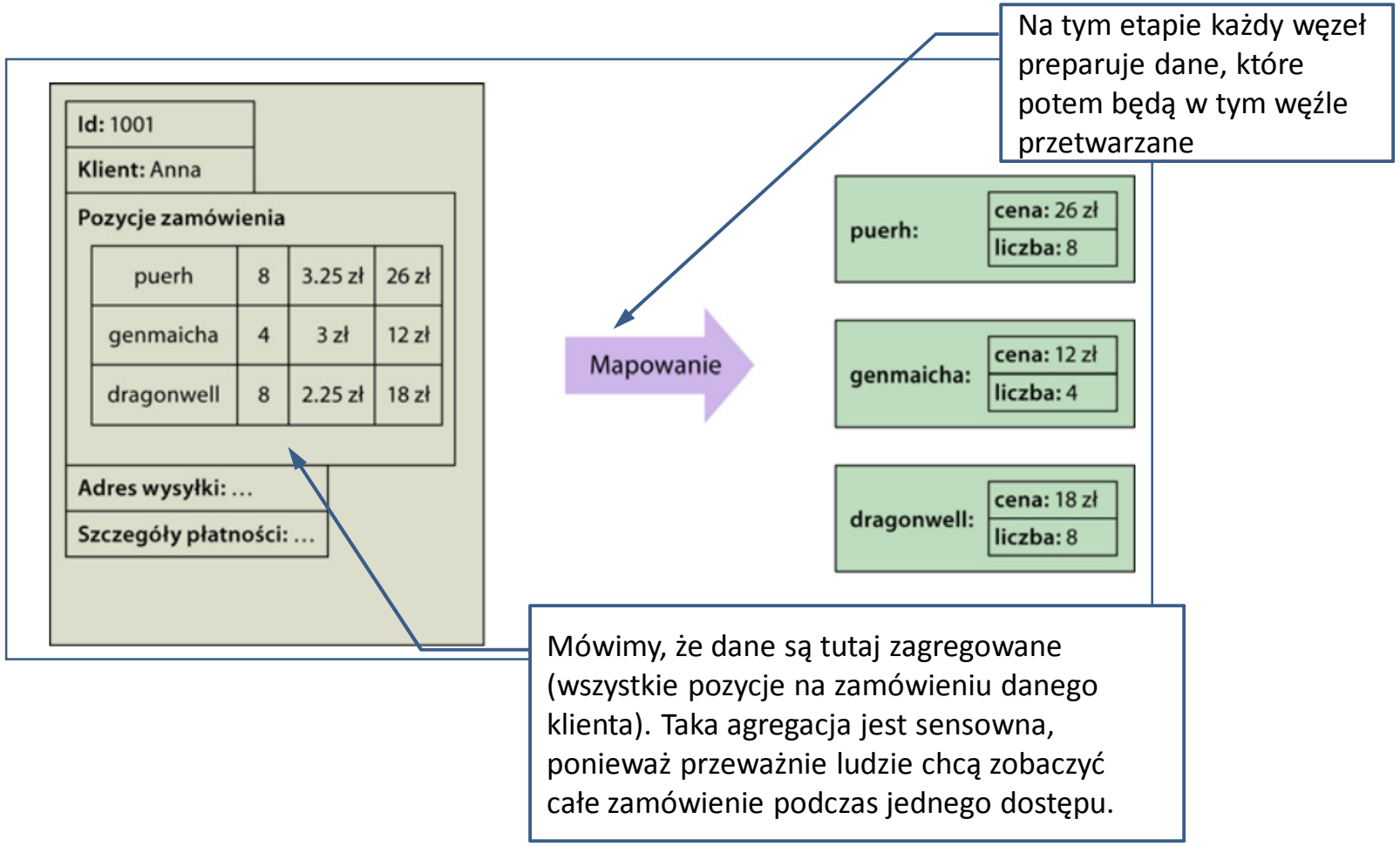
Map-Reduce

- Operacje realizowane są podczas dwóch kroków:
  - Krok map - węzeł nadzorczy (master node) pobiera dane z wejścia i dzieli je na mniejsze podproblemy, po czym przesyła je do węzłów roboczych (worker nodes). Każdy z węzłów roboczych może albo dokonać kolejnego podziału na podproblemy, albo przetworzyć problem i zwrócić odpowiedź do głównego programu
  - Krok reduce - główny program pobiera odpowiedzi na wszystkie podproblemy i łączy je w jeden finalny wynik
- Główną zaletą MapReduce jest umożliwienie łatwego rozproszenia operacji. Zakładając, że każda z operacji map jest niezależna od pozostałych, może być ona realizowana na osobnym serwerze

źródło: https://pl.wikipedia.org/wiki/MapReduce

Map-Reduce– koncepcja

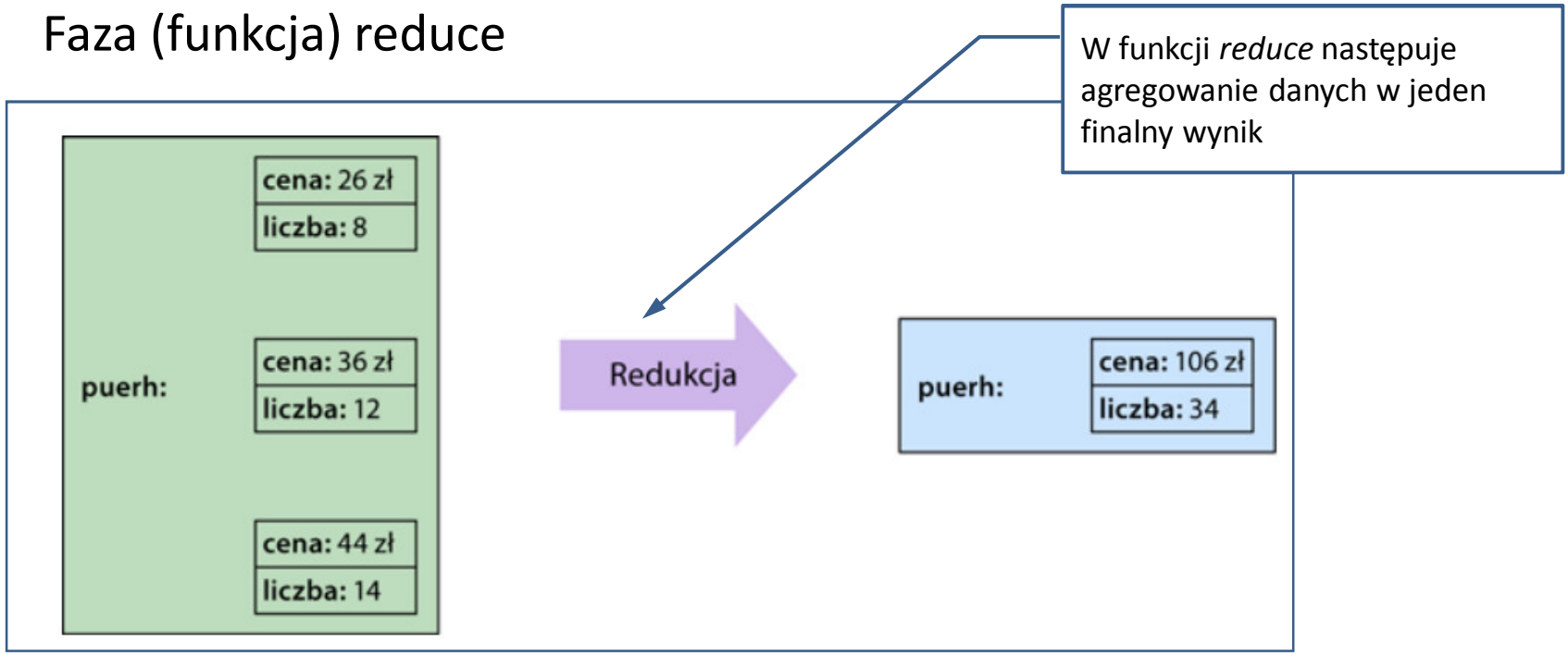
- Na "oklepanym do bólu" przykładzie "klienci-zamówienia"



Rysunek z książki: Sadalage P. J. oraz Fowler M. - NoSQL. Kompendium wiedzy

Map-Reduce– koncepcja

- Faza (funkcja) reduce

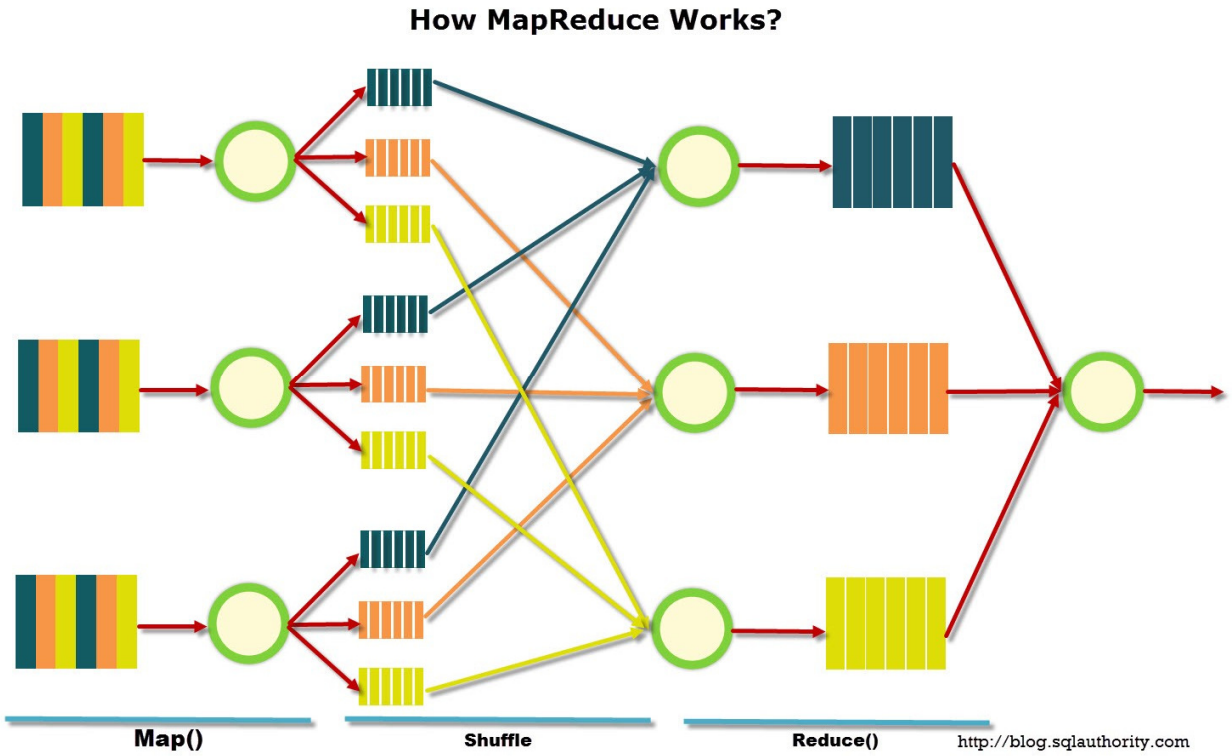


- Biblioteka map-reduce dba o to, aby zadania mapujące uruchamiane były na odpowiednich serwerach w celu przetworzenia wszystkich dokumentów, i o to, aby dane zostały przekazane do funkcji redukującej

Rysunek z książki: Sadalage P. J. oraz Fowler M. - NoSQL. Kompendium wiedzy

Map-Reduce

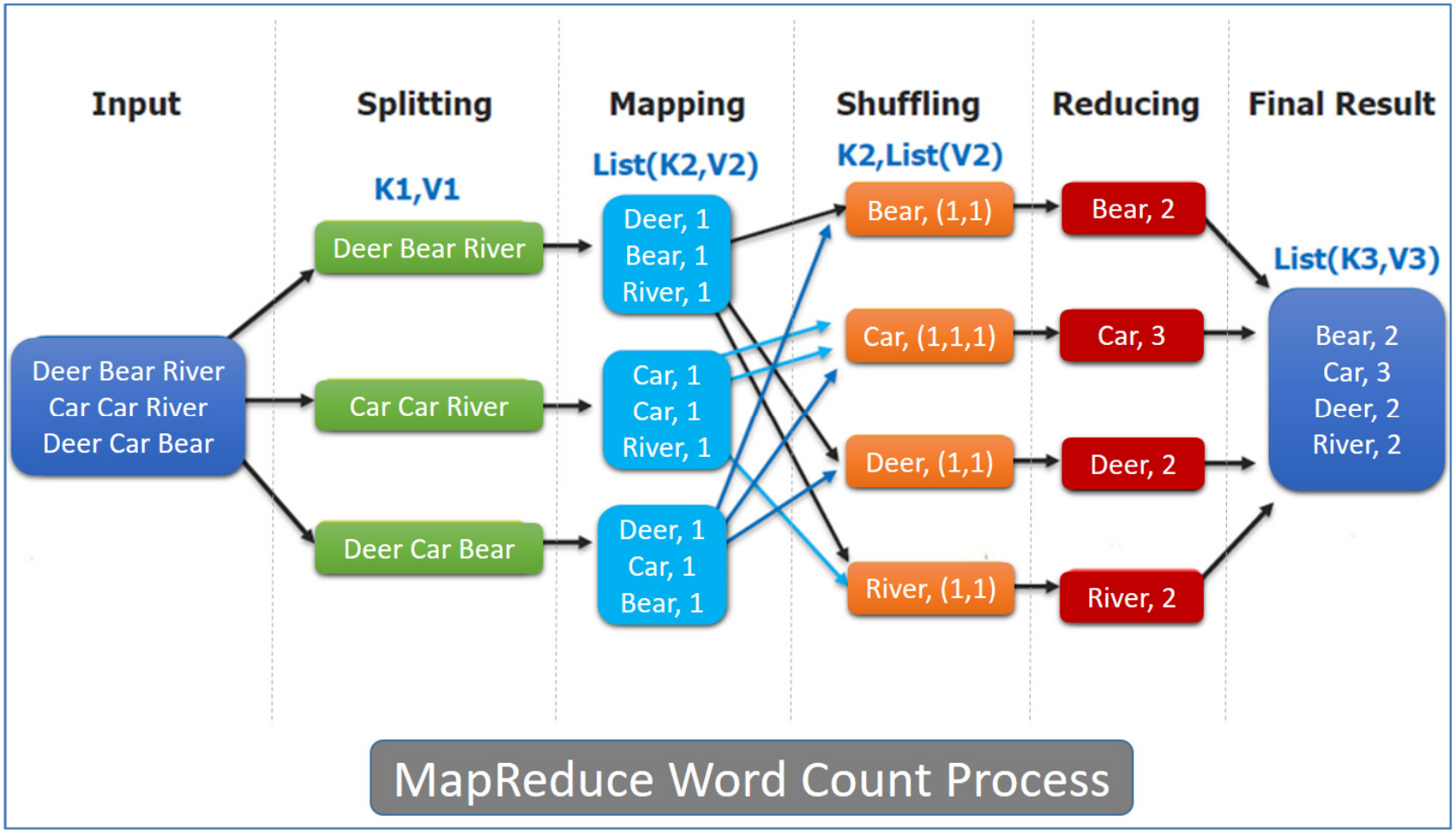
- Implementacja koncepcji MR składa się zwykle z więcej niż 2 tytułowych procesów (map oraz reduce)
  - wiele zależy od konkretnych implementacji. Główna idea jest jednak niezmienna



shuffle – przenosić, przestawiać, przesuwać, przekładać, tasować

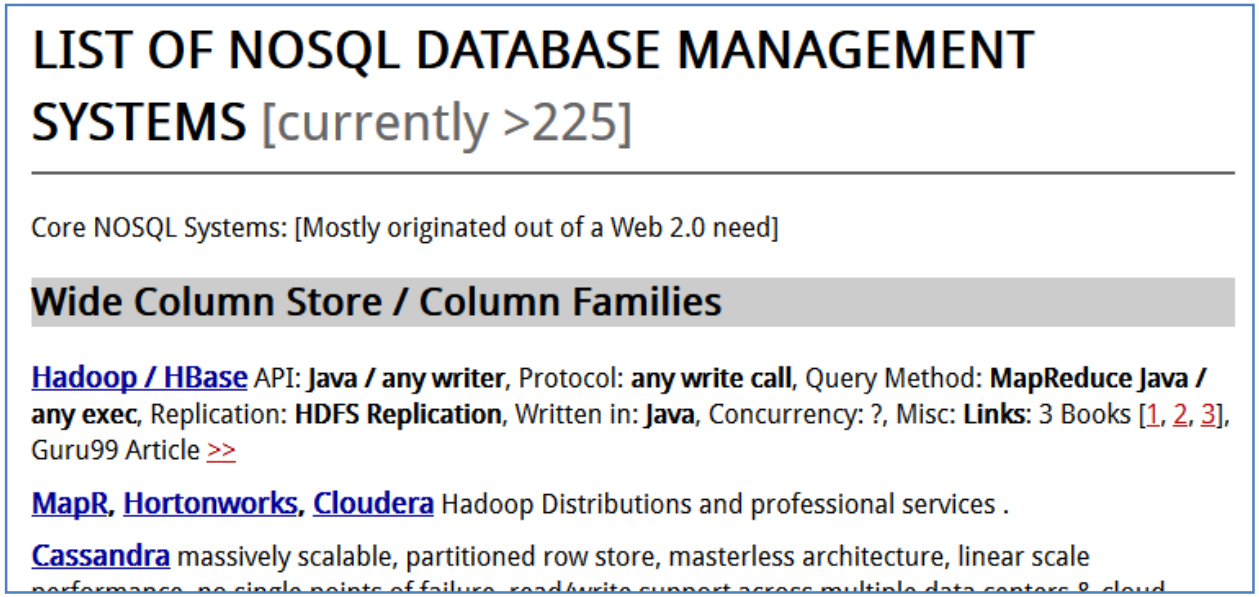
Etap shuffle: dane są tutaj przydzielane do odpowiednich węzłów, które je przetworzą

Map-Reduce


Input
Splitting
Mapping
Shuffling
Reducing
Final Result
K1,V1
List(K2,V2)
K2,List(V2)
List(K3,V3)
Deer Bear River
Car Car River
Deer Car Bear
Deer Bear River
Car Car River
Deer Car Bear
Deer, 1
Bear, 1
River, 1
Car, 1
Car, 1
River, 1
Deer, 1
Car, 1
Bear, 1
Bear, (1,1)
Car, (1,1,1)
Deer, (1,1)
River, (1,1)
Bear, 2
Car, 3
Deer, 2
River, 2
Bear, 2
Car, 3
Deer, 2
River, 2
MapReduce Word Count Process

Klasyfikacja systemów NoSQL

- http://nosql-database.org/



LIST OF NOSQL DATABASE MANAGEMENT SYSTEMS [currently >225]

Core NOSQL Systems: [Mostly originated out of a Web 2.0 need]

Wide Column Store / Column Families

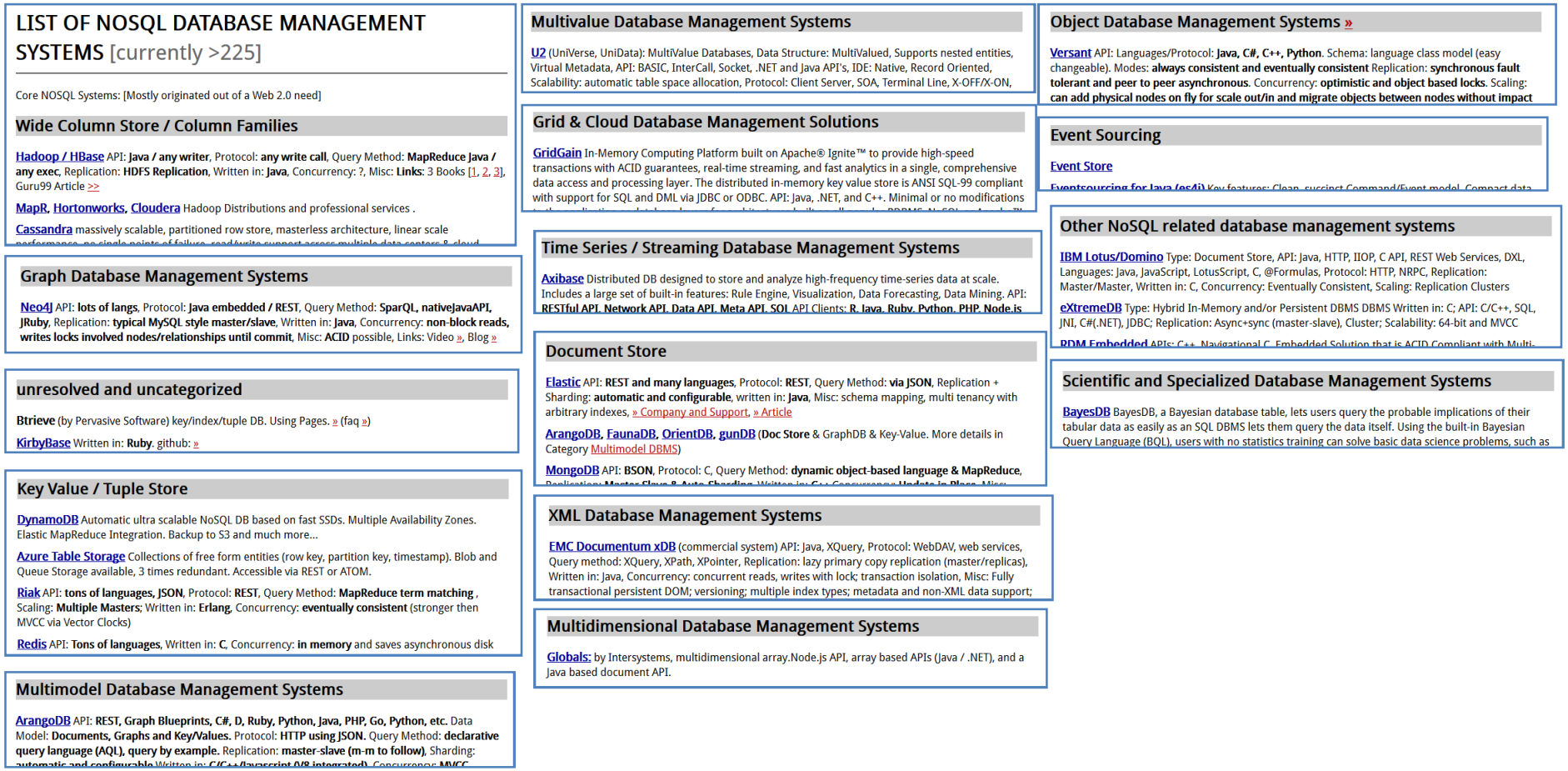
Hadoop / HBase API: **Java / any writer**, Protocol: **any write call**, Query Method: **MapReduce Java / any exec**, Replication: **HDFS Replication**, Written in: **Java**, Concurrency: ?, Misc: **Links**: 3 Books [1, 2, 3], Guru99 Article >>

MapR, Hortonworks, Cloudera Hadoop Distributions and professional services .

Cassandra massively scalable, partitioned row store, masterless architecture, linear scale

Klasyfikacja systemów NoSQL

- http://nosql-database.org/



LIST OF NOSQL DATABASE MANAGEMENT SYSTEMS [currently >225]

Core NOSQL Systems: [Mostly originated out of a Web 2.0 need]

Wide Column Store / Column Families

Hadoop / HBase API: **Java / any writer**, Protocol: **any write call**, Query Method: **MapReduce Java / any exec**, Replication: **HDFS Replication**, Written in: **Java**, Concurrency: ?, Misc: **Links**: 3 Books [1, 2, 3], Guru99 Article >>

MapR, Hortonworks, Cloudera Hadoop Distributions and professional services .

Cassandra massively scalable, partitioned row store, masterless architecture, linear scale

Graph Database Management Systems

Neo4J API: **lots of langs**, Protocol: **Java embedded / REST**, Query Method: **SparQL, nativeJavaAPI, JRuby**, Replication: **typical MySQL style master/slave**, Written in: **Java**, Concurrency: **non-block reads, writes locks involved nodes/relationships until commit**, Misc: **ACID** possible, Links: Video », Blog »

unresolved and uncategorized

Btrieve (by Pervasive Software) key/index/tuple DB. Using Pages. » (faq »)

KirbyBase Written in: **Ruby**. github: »

Key Value / Tuple Store

DynamoDB Automatic ultra scalable NoSQL DB based on fast SSDs. Multiple Availability Zones. Elastic MapReduce Integration. Backup to S3 and much more...

Azure Table Storage Collections of free form entities (row key, partition key, timestamp). Blob and Queue Storage available, 3 times redundant. Accessible via REST or ATOM.

Riak API: **tons of languages, JSON**, Protocol: **REST**, Query Method: **MapReduce term matching** , Scaling: **Multiple Masters**; Written in: **Erlang**, Concurrency: **eventually consistent** (stronger then MVCC via Vector Clocks)

Redis API: **Tons of languages**, Written in: **C**, Concurrency: **in memory** and saves asynchronous disk

Multimodel Database Management Systems

ArangoDB API: **REST, Graph Blueprints, C#, D, Ruby, Python, Java, PHP, Go, Python, etc.** Data Model: **Documents, Graphs and Key/Values.** Protocol: **HTTP using JSON.** Query Method: **declarative query language (AQL), query by example.** Replication: **master-slave (m-m to follow)**, Sharding:

Multivalue Database Management Systems

U2 (UniVerse, UniData): MultiValue Databases, Data Structure: MultiValued, Supports nested entities, Virtual Metadata, API: BASIC, InterCall, Socket, .NET and Java API's, IDE: Native, Record Oriented, Scalability: automatic table space allocation, Protocol: Client Server, SOA, Terminal Line, X-OFF/X-ON,

Grid & Cloud Database Management Solutions

GridGain In-Memory Computing Platform built on Apache® Ignite™ to provide high-speed transactions with ACID guarantees, real-time streaming, and fast analytics in a single, comprehensive data access and processing layer. The distributed in-memory key value store is ANSI SQL-99 compliant with support for SQL and DML via JDBC or ODBC. API: Java, .NET, and C++. Minimal or no modifications

Time Series / Streaming Database Management Systems

Axibase Distributed DB designed to store and analyze high-frequency time-series data at scale. Includes a large set of built-in features: Rule Engine, Visualization, Data Forecasting, Data Mining. API: RESTful API, Network API, Data API, Meta API, SQL API Clients: R, Java, Ruby, Python, PHP, Node.js

Document Store

Elastic API: **REST and many languages**, Protocol: **REST**, Query Method: **via JSON**, Replication + Sharding: **automatic and configurable**, written in: **Java**, Misc: schema mapping, multi tenancy with arbitrary indexes, » Company and Support, » Article

ArangoDB, FaunaDB, OrientDB, gunDB (**Doc Store** & GraphDB & Key-Value. More details in Category Multimodel DBMS)

MongoDB API: **BSON**, Protocol: C, Query Method: **dynamic object-based language & MapReduce**,

XML Database Management Systems

EMC Documentum xDB (commercial system) API: Java, XQuery, Protocol: WebDAV, web services, Query method: XQuery, XPath, XPointer, Replication: lazy primary copy replication (master/replicas), Written in: Java, Concurrency: concurrent reads, writes with lock; transaction isolation, Misc: Fully transactional persistent DOM; versioning; multiple index types; metadata and non-XML data support;

Multidimensional Database Management Systems

Globals: by Intersystems, multidimensional array.Node.js API, array based APIs (Java / .NET), and a Java based document API.

Object Database Management Systems »

Versant API: Languages/Protocol: **Java, C#, C++, Python**. Schema: language class model (easy changeable). Modes: **always consistent and eventually consistent** Replication: **synchronous fault tolerant and peer to peer asynchronous**. Concurrency: **optimistic and object based locks**. Scaling: **can add physical nodes on fly for scale out/in and migrate objects between nodes without impact**

Event Sourcing

Event Store

Other NoSQL related database management systems

IBM Lotus/Domino Type: Document Store, API: Java, HTTP, IIOP, C API, REST Web Services, DXL, Languages: Java, JavaScript, LotusScript, C, @Formulas, Protocol: HTTP, NRPC, Replication: Master/Master, Written in: C, Concurrency: Eventually Consistent, Scaling: Replication Clusters

eXtremeDB Type: Hybrid In-Memory and/or Persistent DBMS DBMS Written in: C; API: C/C++, SQL, JNI, C#(.NET), JDBC; Replication: Async+sync (master-slave), Cluster; Scalability: 64-bit and MVCC

Scientific and Specialized Database Management Systems

BayesDB BayesDB, a Bayesian database table, lets users query the probable implications of their tabular data as easily as an SQL DBMS lets them query the data itself. Using the built-in Bayesian Query Language (BQL), users with no statistics training can solve basic data science problems, such as

Ranking systemów baz danych

- https://db-engines.com/en/ranking
- W serwisie wymienionych jest (wrzesień 2018) aż 345 różnych systemów baz danych !!!

345 systems in ranking, September 2018

Rank Sep 2018	Rank Aug 2018	Rank Sep 2017	DBMS	Database Model	Score Sep 2018	Score Aug 2018	Score Sep 2017
1.	1.	1.	Oracle	Relational DBMS	1309.12	-2.91	-49.97
2.	2.	2.	MySQL	Relational DBMS	1180.48	-26.33	-132.13
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1051.28	-21.37	-161.26
4.	4.	4.	PostgreSQL	Relational DBMS	406.43	-11.07	+34.07
5.	5.	5.	MongoDB	Document store	358.79	+7.81	+26.06
6.	6.	6.	DB2	Relational DBMS	181.06	-0.78	-17.28
7.	↑ 8.	↑ 10.	Elasticsearch	Search engine	142.61	+4.49	+22.61
8.	↓ 7.	↑ 9.	Redis	Key-value store	140.94	+2.37	+20.54
9.	9.	↓ 7.	Microsoft Access	Relational DBMS	133.39	+4.30	+4.58
10.	10.	↓ 8.	Cassandra	Wide column store	119.55	-0.02	-6.65

Ranking systemów baz danych

- https://db-engines.com/en/ranking
- Październik 2019 – 355 różnych systemów baz danych !!!

355 systems in ranking, October 2019

Rank Oct 2019	Rank Sep 2019	Rank Oct 2018	DBMS	Database Model	Score Oct 2019	Score Sep 2019	Score Oct 2018
1.	1.	1.	Oracle	Relational, Multi-model	1355.88	+9.22	+36.61
2.	2.	2.	MySQL	Relational, Multi-model	1283.06	+3.99	+104.94
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	1094.72	+9.66	+36.39
4.	4.	4.	PostgreSQL	Relational, Multi-model	483.91	+1.66	+64.52
5.	5.	5.	MongoDB	Document, Multi-model	412.09	+2.03	+48.90
6.	6.	6.	IBM Db2	Relational, Multi-model	170.77	-0.79	-8.91
7.	7.	↑ 8.	Elasticsearch	Search engine, Multi-model	150.17	+0.90	+7.85
8.	8.	↓ 7.	Redis	Key-value, Multi-model	142.91	+1.01	-2.38
9.	9.	9.	Microsoft Access	Relational	131.18	-1.53	-5.62
10.	10.	10.	Cassandra	Wide column	123.22	-0.18	-0.17

Ranking systemów baz danych

- https://db-engines.com/en/ranking
- Kwiecień 2020 – 355 różnych systemów baz danych !!!

355 systems in ranking, April 2020

Rank			DBMS	Database Model	Score		
Apr 2020	Mar 2020	Apr 2019			Apr 2020	Mar 2020	Apr 2019
1.	1.	1.	Oracle	Relational, Multi-model	1345.42	+4.78	+65.48
2.	2.	2.	MySQL	Relational, Multi-model	1268.35	+8.62	+53.21
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	1083.43	-14.43	+23.47
4.	4.	4.	PostgreSQL	Relational, Multi-model	509.86	-4.06	+31.14
5.	5.	5.	MongoDB	Document, Multi-model	438.43	+0.82	+36.45
6.	6.	6.	IBM Db2	Relational, Multi-model	165.63	+3.07	-10.42
7.	7.	↑ 8.	Elasticsearch	Search engine, Multi-model	148.91	-0.26	+2.91
8.	8.	↓ 7.	Redis	Key-value, Multi-model	144.81	-2.77	-1.57
9.	↑ 10.	↑ 10.	SQLite	Relational	122.19	+0.24	-2.02
10.	↓ 9.	↓ 9.	Microsoft Access	Relational	121.92	-3.22	-22.73

Ranking systemów baz danych

- https://db-engines.com/en/ranking
- Październik 2020 – 359 różnych systemów baz danych !!!

Rank			DBMS	Database Model	Score		
Oct 2020	Sep 2020	Oct 2019			Oct 2020	Sep 2020	Oct 2019
1.	1.	1.	Oracle	Relational, Multi-model	1368.77	-0.59	+12.89
2.	2.	2.	MySQL	Relational, Multi-model	1256.38	-7.87	-26.69
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	1043.12	-19.64	-51.60
4.	4.	4.	PostgreSQL	Relational, Multi-model	542.40	+0.12	+58.49
5.	5.	5.	MongoDB	Document, Multi-model	448.02	+1.54	+35.93
6.	6.	6.	IBM Db2	Relational, Multi-model	161.90	+0.66	-8.87
7.	↑ 8.	7.	Elasticsearch	Search engine, Multi-model	153.84	+3.35	+3.67
8.	↓ 7.	8.	Redis	Key-value, Multi-model	153.28	+1.43	+10.37
9.	9.	↑ 11.	SQLite	Relational	125.43	-1.25	+2.80
10.	10.	10.	Cassandra	Wide column	119.10	-0.08	-4.12

Ranking systemów baz danych

- Kryteria, na bazie których powstaje ranking

Method of calculating the scores of the DB-Engines Ranking

The DB-Engines Ranking is a list of database management systems ranked by their current popularity. We measure the popularity of a system by using the following parameters:

- **Number of mentions of the system on websites**, measured as number of results in search engines queries. At the moment, we use Google, Bing and Yandex for this measurement. In order to count only relevant results, we are searching for <system name> together with the term database, e.g. "Oracle" and "database".
- **General interest in the system.** For this measurement, we use the frequency of searches in Google Trends.
- **Frequency of technical discussions about the system.** We use the number of related questions and the number of interested users on the well-known IT-related Q&A sites Stack Overflow and DBA Stack Exchange.
- **Number of job offers, in which the system is mentioned.** We use the number of offers on the leading job search engines Indeed and Simply Hired.
- **Number of profiles in professional networks, in which the system is mentioned.** We use the internationally most popular professional networks LinkedIn and Upwork.
- **Relevance in social networks.** We count the number of Twitter tweets, in which the system is mentioned.

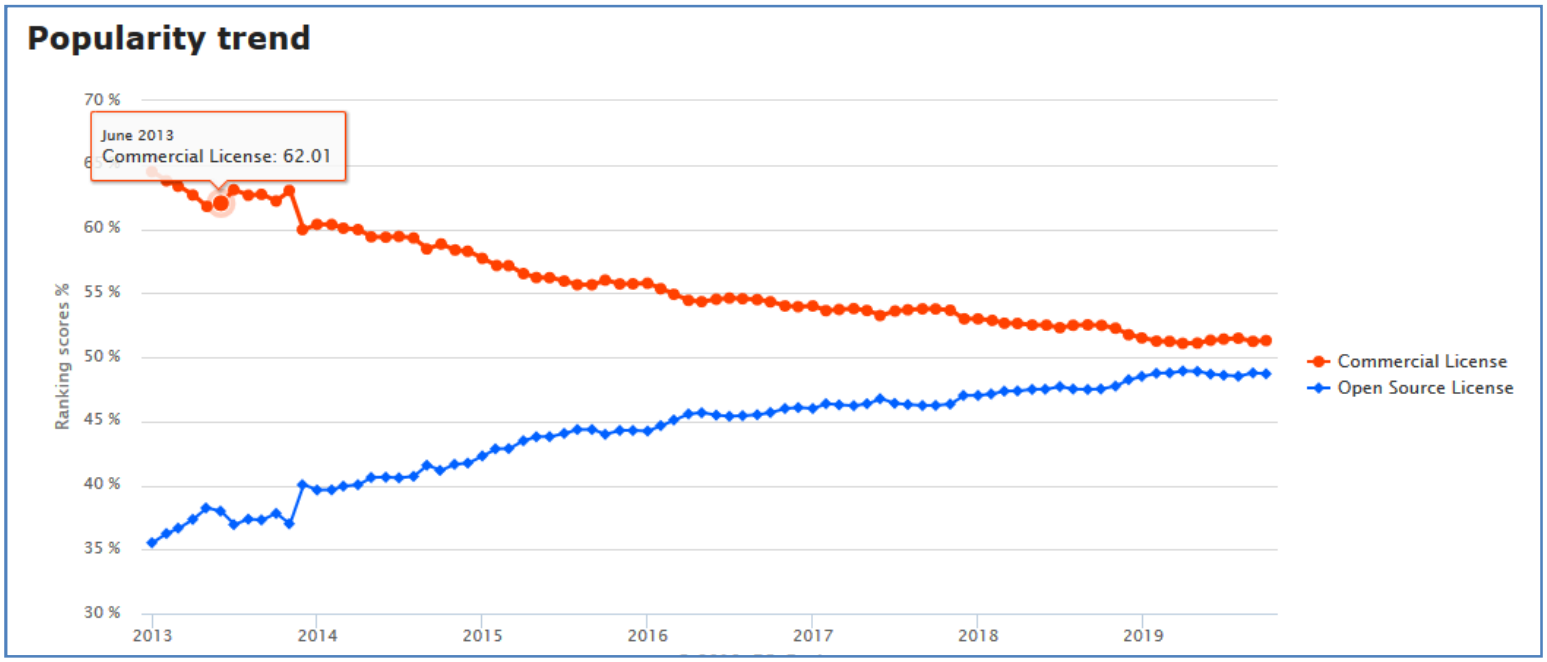
Ranking systemów baz danych

- Kryteria, na bazie których powstaje ranking

> The DB-Engines Ranking does not measure the number of installations of the systems, or their use within IT systems. It can be expected, that an increase of the popularity of a system as measured by the DB-Engines Ranking (e.g. in discussions or job offers) precedes a corresponding broad use of the system by a certain time factor. Because of this, the DB-Engines Ranking can act as an early indicator.

Ranking systemów baz danych

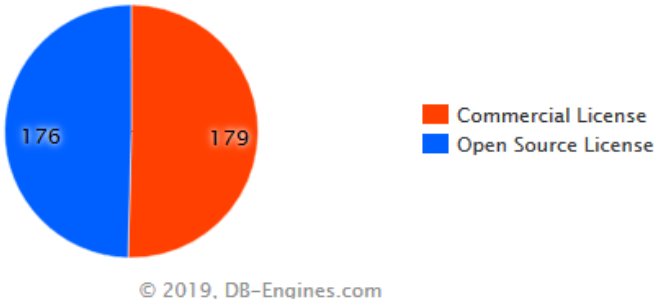
- Komercyjne / niekomercyjne



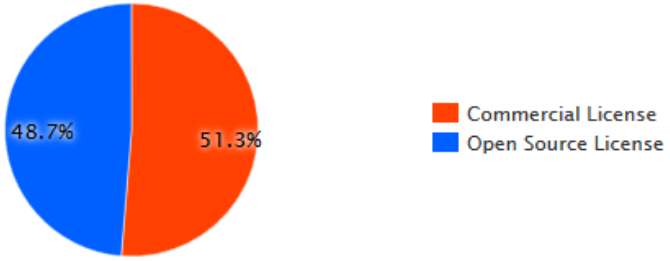
Ranking systemów baz danych

- Komercyjne / niekomercyjne

Number of systems, October 2019



Popularity scores, October 2019



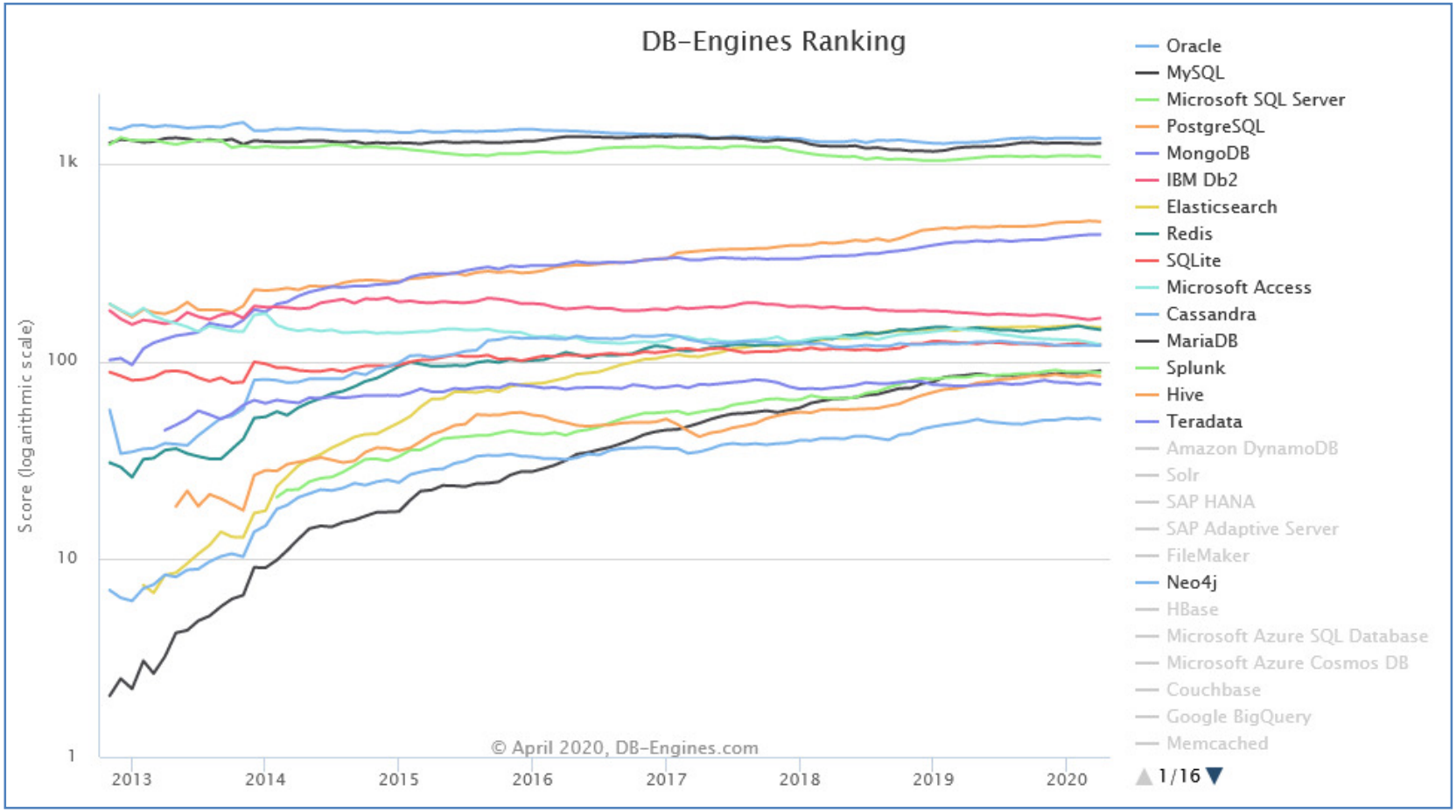
The top 5 commercial systems, October 2019

Rank	System	Score	Overall Rank
1.	Oracle	1356	1.
2.	Microsoft SQL Server	1095	3.
3.	IBM Db2	171	6.
4.	Microsoft Access	131	9.
5.	Splunk	87	12.

The top 5 open source systems, October 2019

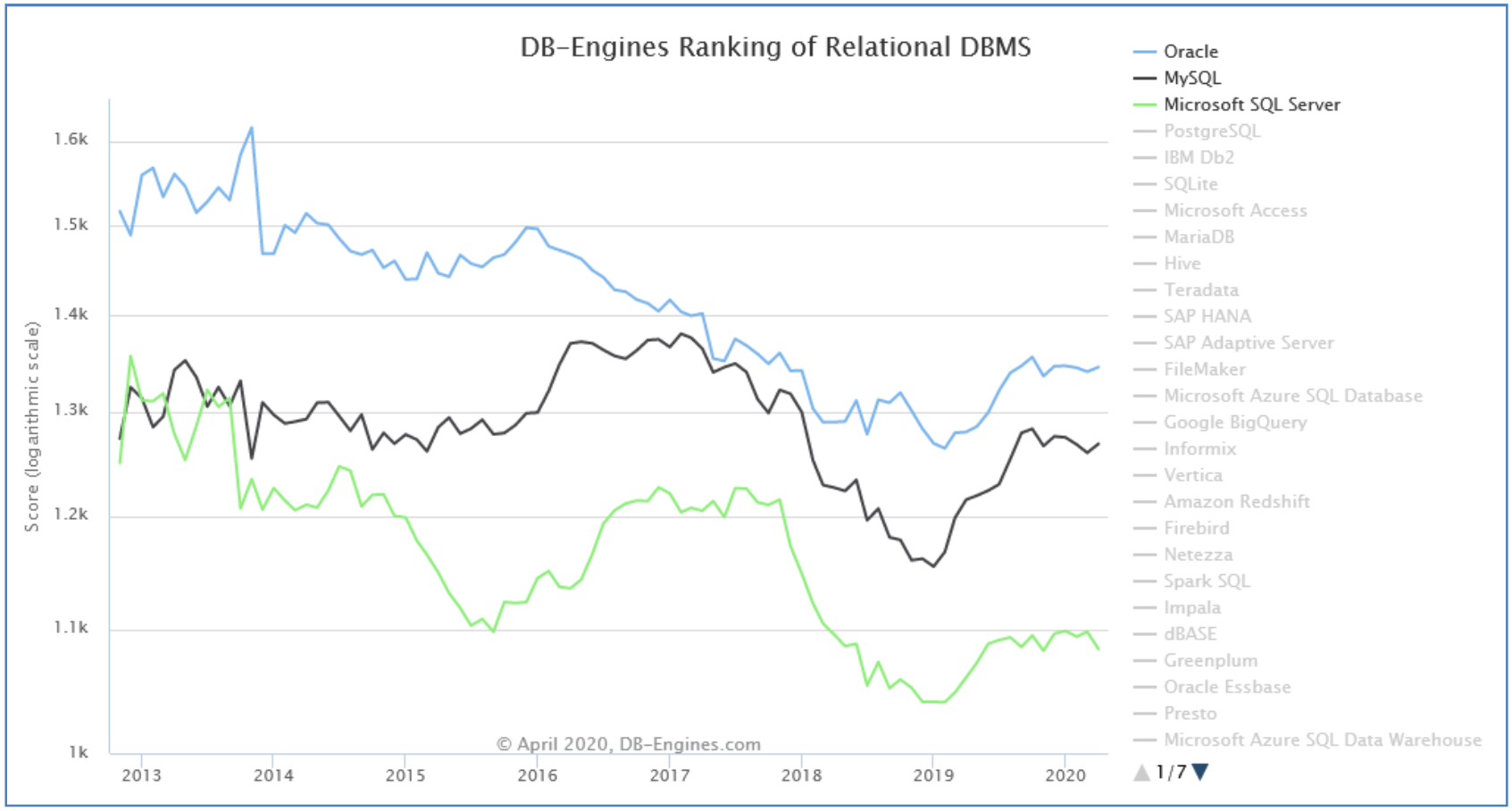
Rank	System	Score	Overall Rank
1.	MySQL	1283	2.
2.	PostgreSQL	484	4.
3.	MongoDB	412	5.
4.	Elasticsearch	150	7.
5.	Redis	143	8.

Ranking systemów baz danych



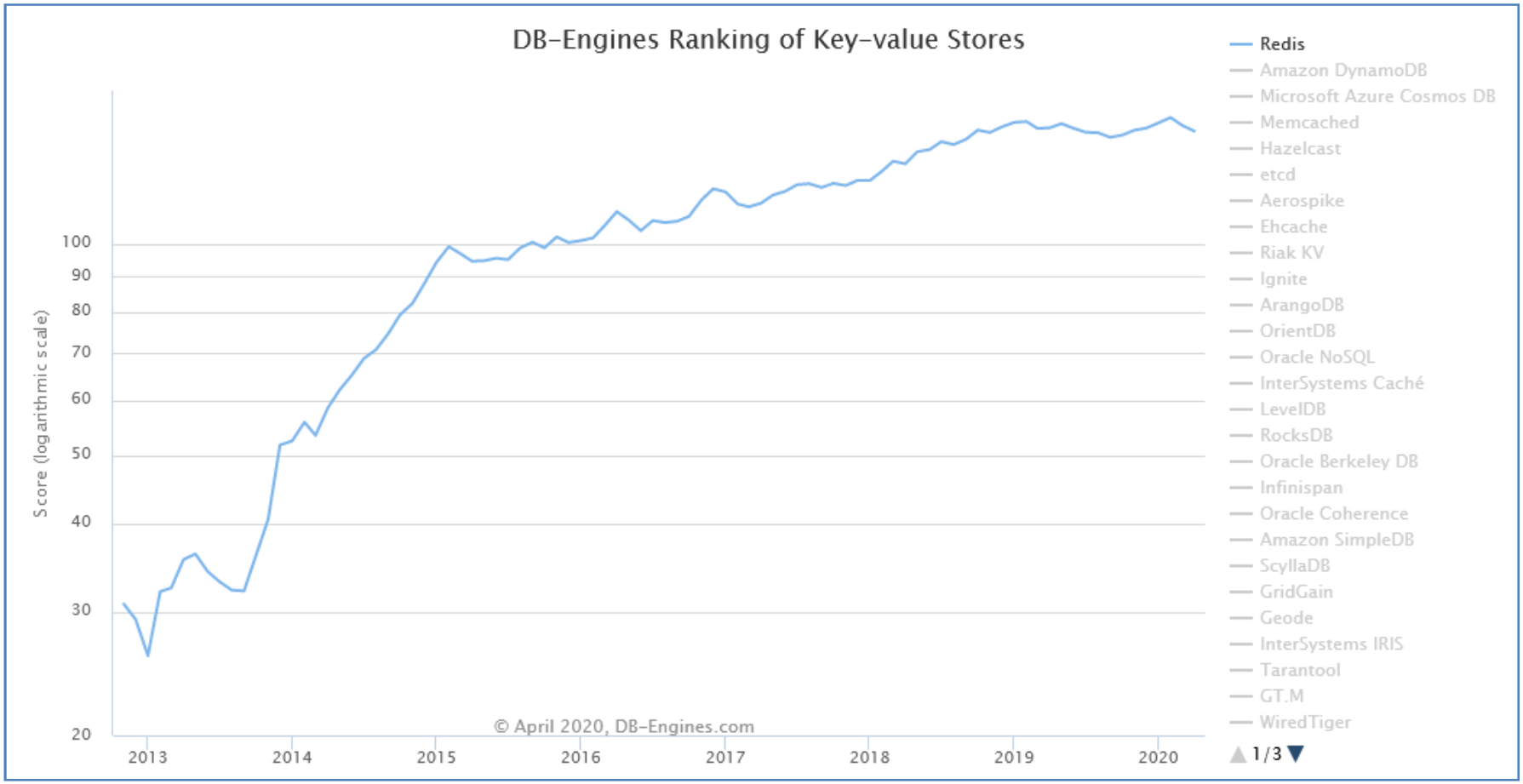
Ranking systemów baz danych

- W odpowiedniej skali widać lekki trend spadkowy baz relacyjnych…



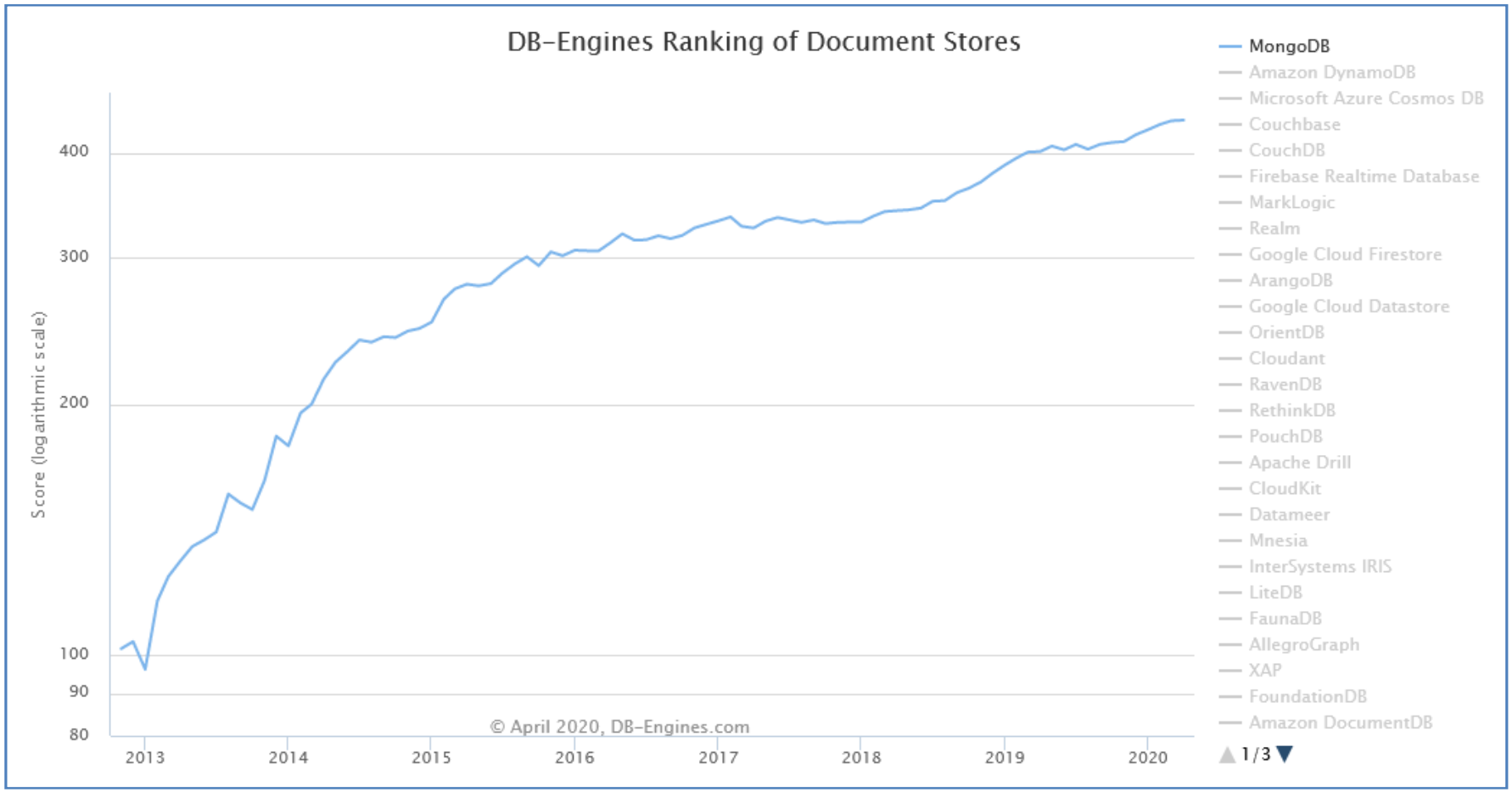
Ranking systemów baz danych

- … oraz spore wzrosty baz nierelacyjnych



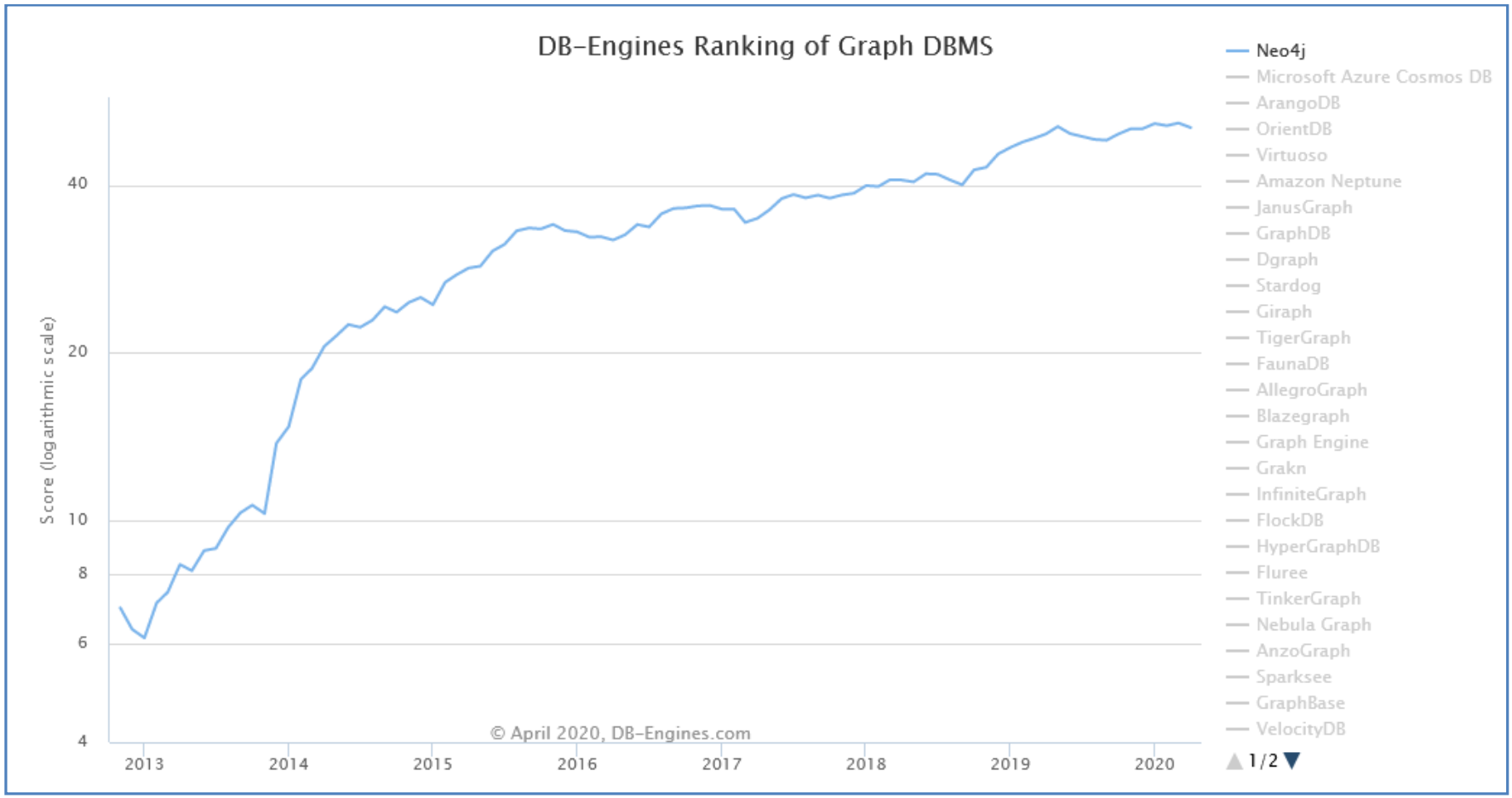
Ranking systemów baz danych

- … oraz spore wzrosty baz nierelacyjnych



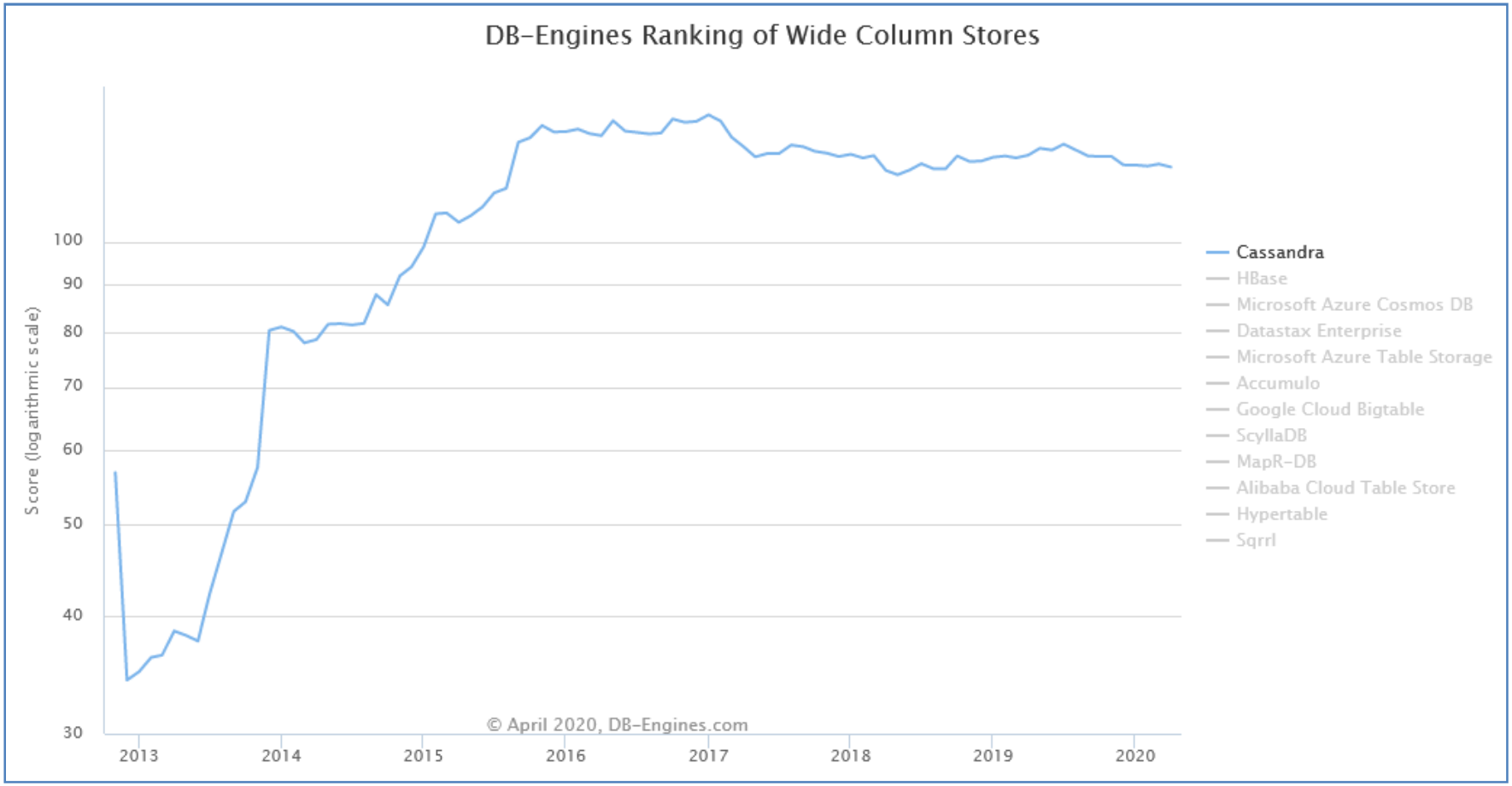
Ranking systemów baz danych

- … oraz spore wzrosty baz nierelacyjnych



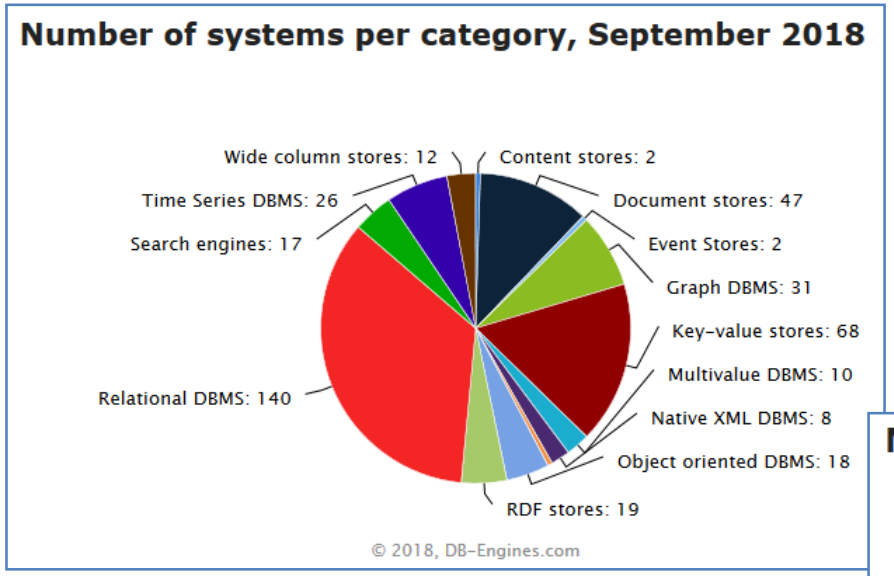
Ranking systemów baz danych

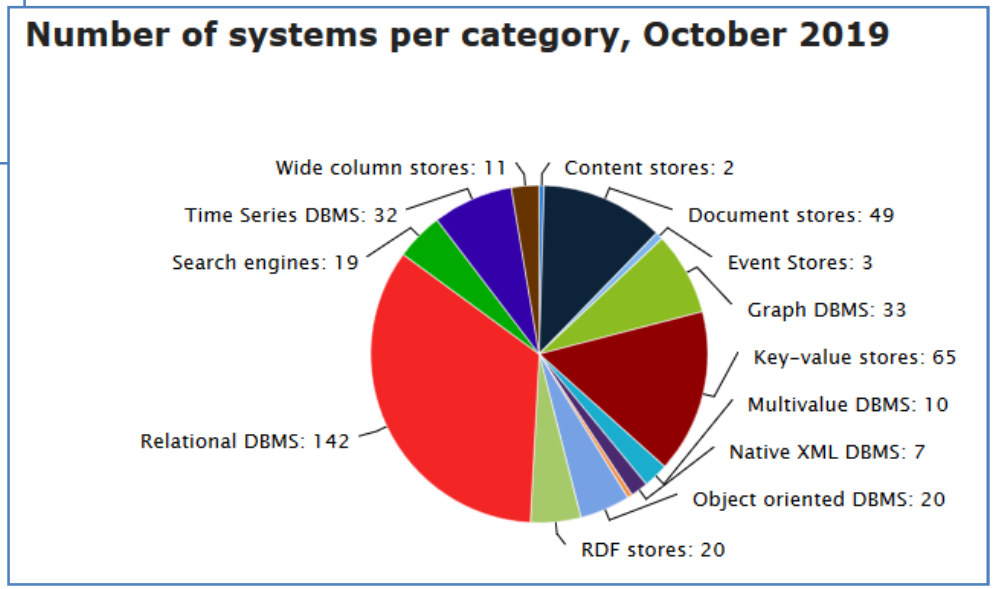
- Baza typy rodzina kolumn Cassandra chyba traci na popularności



Ranking systemów baz danych

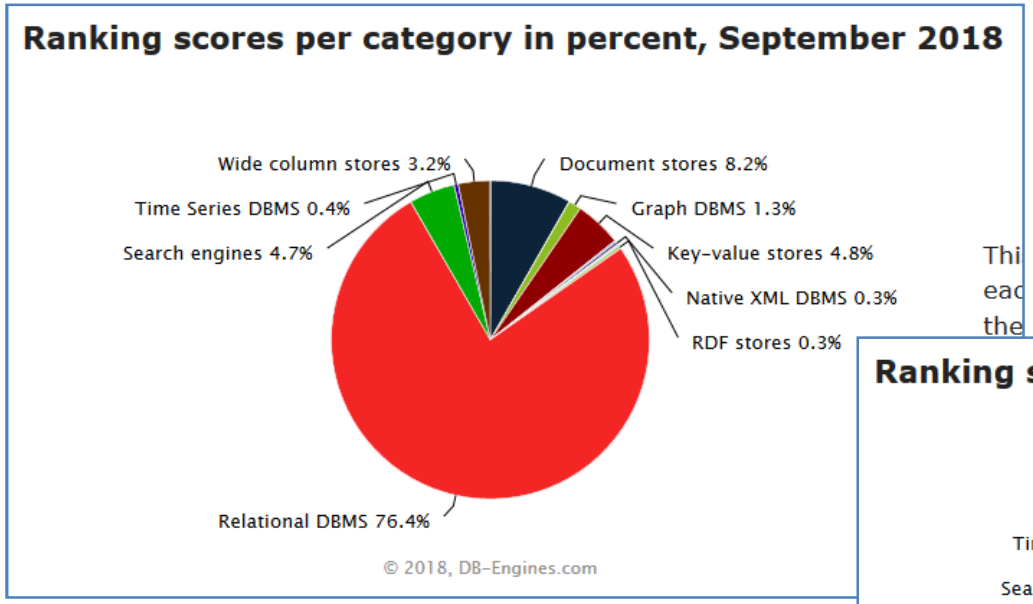
- https://db-engines.com/en/ranking_categories

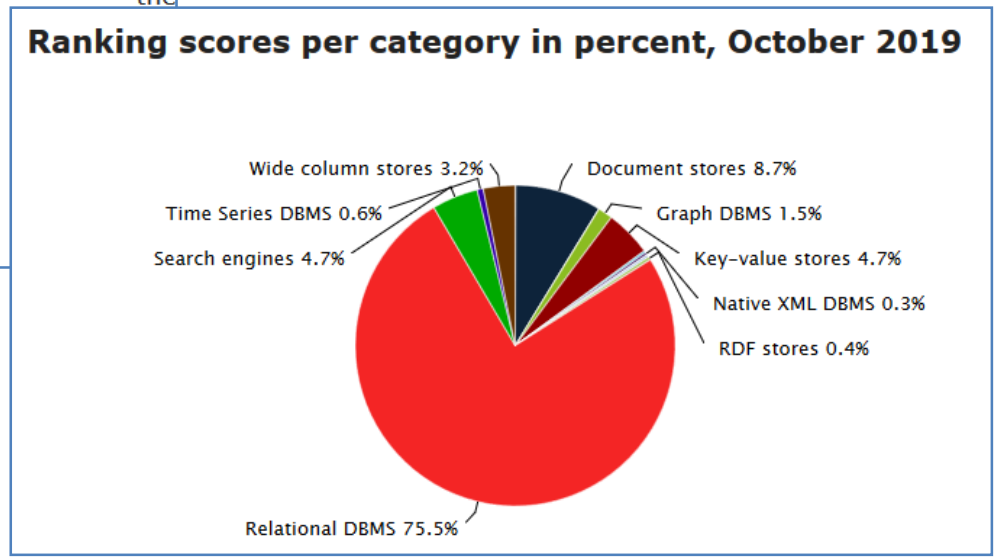




Ranking systemów baz danych

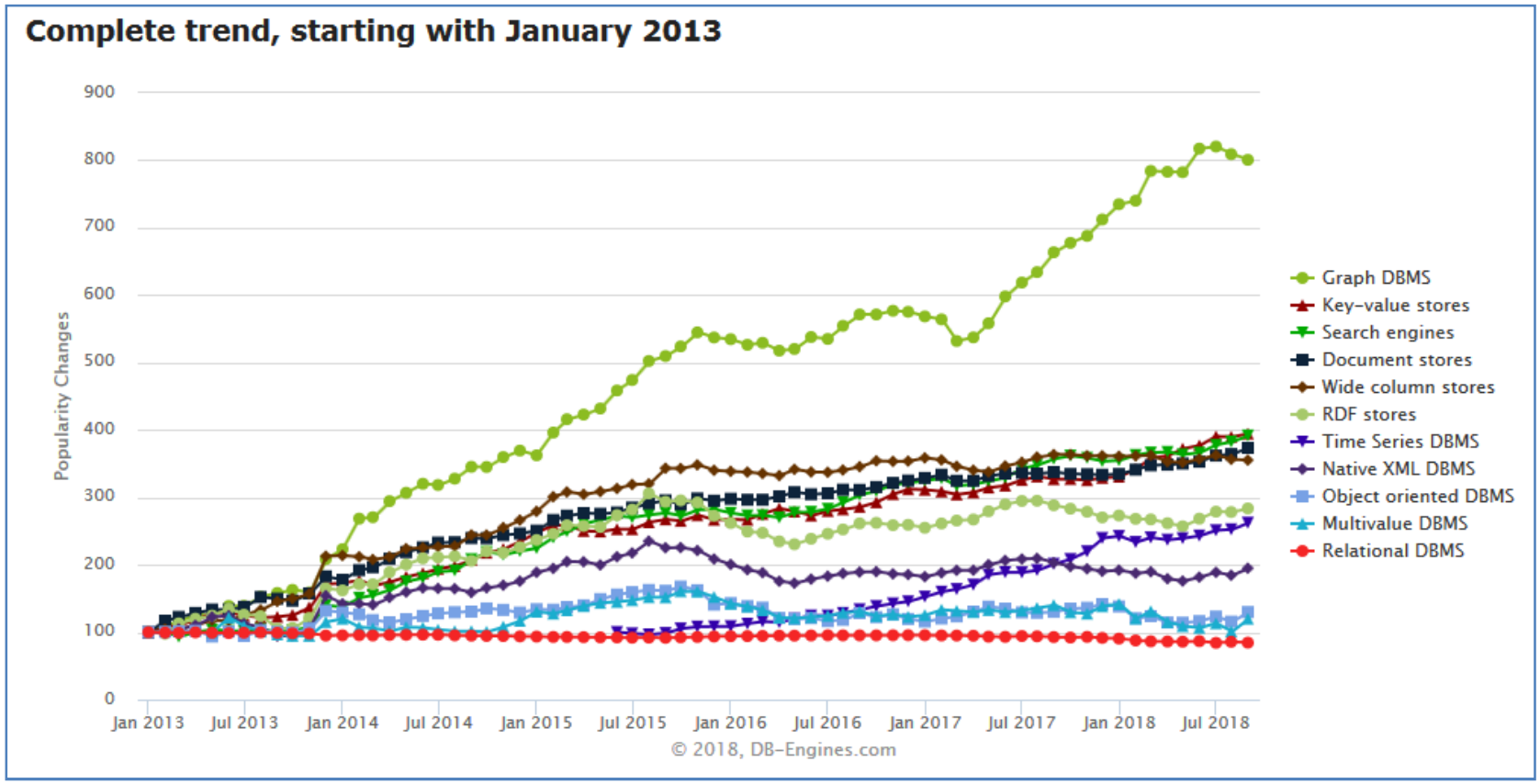
- https://db-engines.com/en/ranking_categories





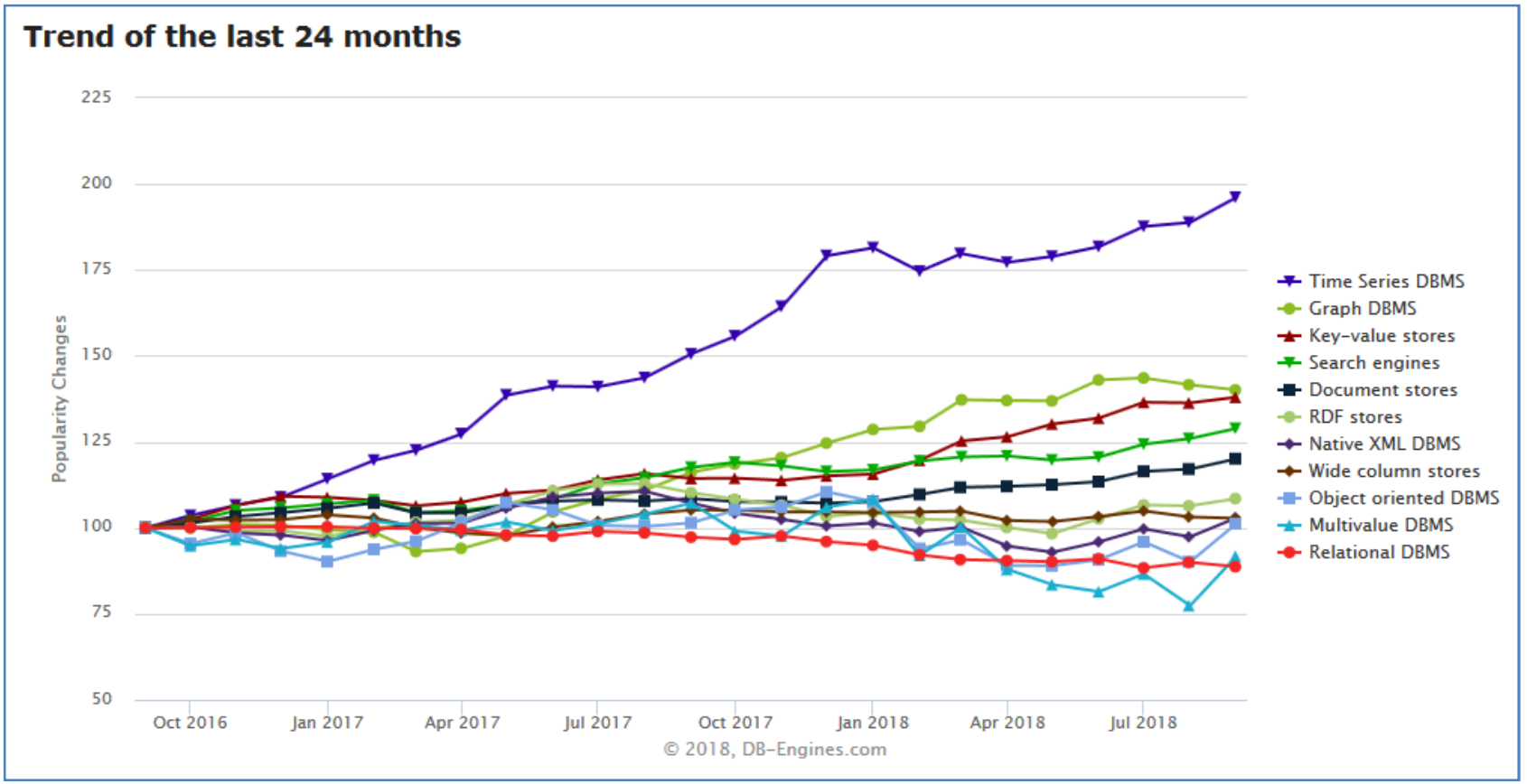
Ranking systemów baz danych

- https://db-engines.com/en/ranking_categories



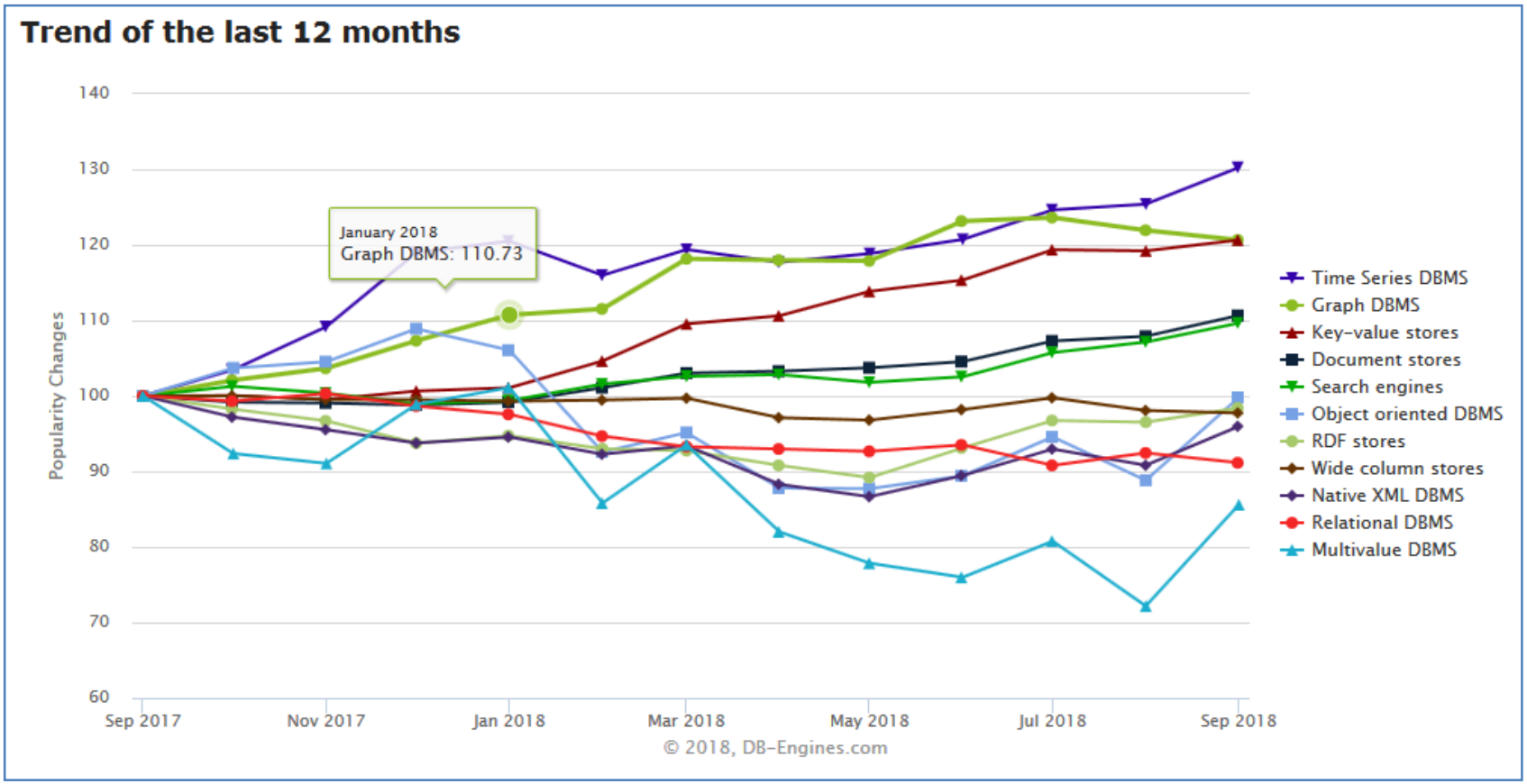
Ranking systemów baz danych

- https://db-engines.com/en/ranking_categories



Ranking systemów baz danych

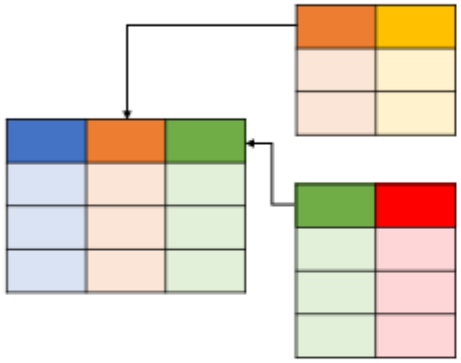
- https://db-engines.com/en/ranking_categories



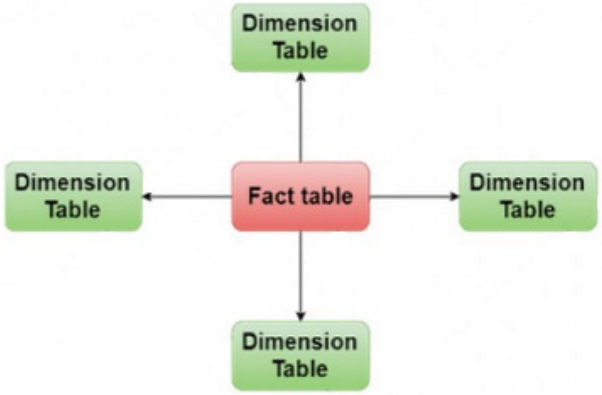
6 podstawowych rodzajów baz danych

SQL DATABASES

Relational, transactional (OLTP)



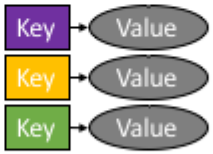
Relational, analytical (OLAP)



NoSQL DATABASES

Column

Graph



Key-Value

Document

Klasyfikacja systemów NoSQL

- Key-Value (Klucz-Wartość)
  - Aerospike, Apache Ignite, ArangoDB, Berkeley DB, Couchbase, Dynamo, FairCom c-treeACE, FoundationDB, InfinityDB, MemcacheDB, MUMPS, Oracle NoSQL Database, OrientDB, Redis, Riak, SciDB, SDBM/Flat File dbm, ZooKeeper
- Document Stores (Bazy dokumentowe)
  - Apache CouchDB, ArangoDB, BaseX, Clusterpoint, Couchbase, Cosmos DB, IBM Domino, MarkLogic, MongoDB, OrientDB, Qizx, RethinkDB
- Graph DB (Bazy grafowe)
  - AllegroGraph, ArangoDB, InfiniteGraph, Apache Giraph, MarkLogic, Neo4J, OrientDB, Virtuoso
- Column Stores (Rodzina kolumn)
  - Accumulo, Cassandra, Druid, HBase, Vertica.

Klasyfikacja systemów NoSQL

- Kolejne kategorie (mniej systematyczny podział niż na poprzednim slajdzie)
  - Object Databases
  - Grid & Cloud Database Solutions
  - XML Databases
  - Time Series / Streaming Databases
  - Event Sourcing
  - Multivalue Databases
  - Multidimensional Databases
  - Multimodel Databases
  - niedające się zaklasyfikować do żadnej z powyższych klas

Agregacyjne modele danych

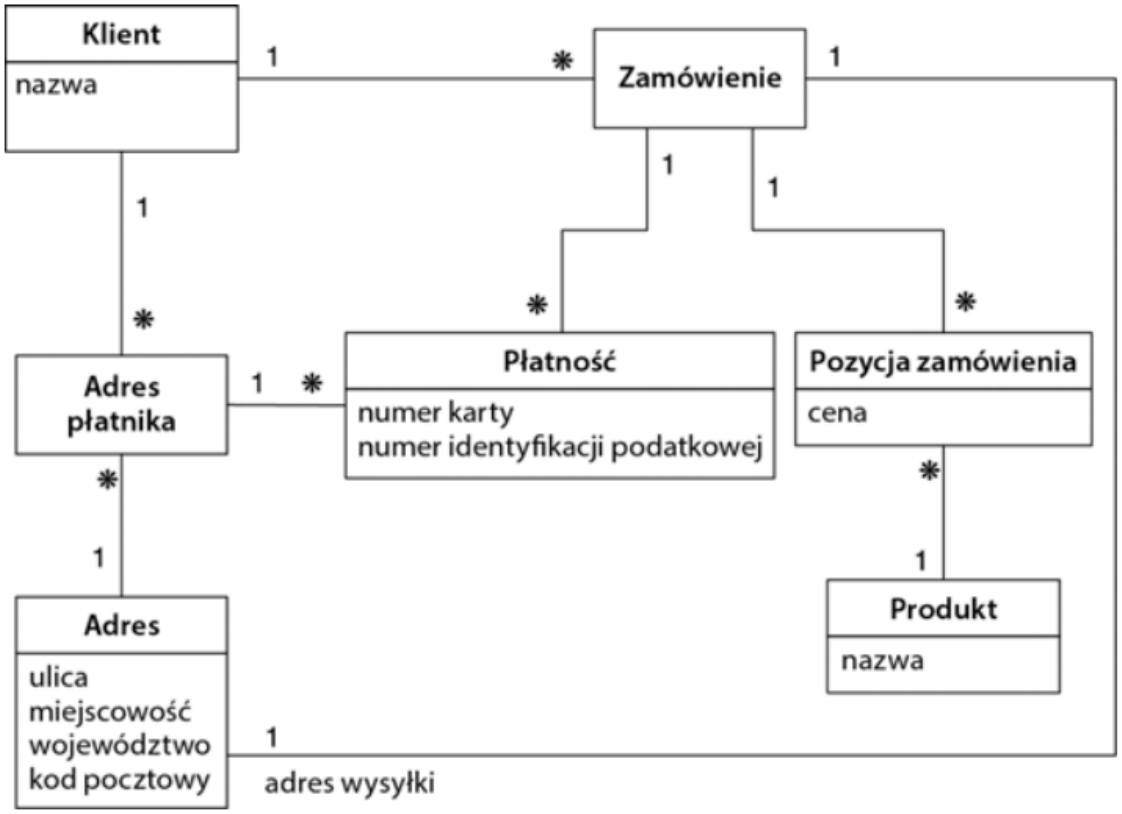
- Modele: key-value, dokumentowe, rodzina kolumn są koncepcyjnie trochę do siebie podobne. Mażemy nadać im wspólną nazwę modeli zorientowanych na agregacje
- Zupełne odejście od modelu relacyjnego
  - w modelu relacyjnym wiersz w tabeli jest podstawową porcją danych
  - w modelu relacyjnym wewnątrz wierszy nie można zagnieżdżać innych wierszy
  - działanie modelu relacyjnego de facto opiera się na tworzeniu (INSERT), modyfikowaniu (UPDATE), kasowaniu (DELETE) oraz zwracaniu (SELECT) wierszy
    - oczywiście w bazach NoSQL też te operacje wykonujemy ale nie z poziomu SQL-a tylko z poziomu różnych indywidualnych rozwiązań poszczególnych implementacji baz

Agregacyjne modele danych

- Orientacja na agregacje
  - operujemy na danych o bardziej skomplikowanych strukturach niż zestaw wierszy
  - agregacja jest pewną kolekcją obiektów, które traktujemy jako jednostka
  - operowanie na agregacjach ułatwia bazom danych pracę w klastrach
- Przykłady relacji i agregacji (patrz kolejny slajd)
  - na podstawie książki: Sadalage P. J. oraz Fowler M. "NoSQL. Kompendium wiedzy"

Agregacyjne modele danych

- Relacyjny model danych (w notacji UML)



Rysunek z książki: Sadalage P. J. oraz Fowler M. - NoSQL. Kompendium wiedzy

Agregacyjne modele danych

- Dane w przykładowej relacyjnej bazie
  - wszystko jest elegancko znormalizowane

Klient

Id	Nazwa
1	Marcin

Zamowienie

Id	KlientId	AdresWysylkiId
99	1	77

Produkt

Id	Nazwa
27	NoSQL. Kompedium

AdresPlatnosci

Id	KlientId	AdresId
55	1	77

PozycjaZamowienia

Id	ZamowienieId	ProduktId	Cena
100	99	27	32.45

Adres

Id	Miejscowosc
77	Warszawa

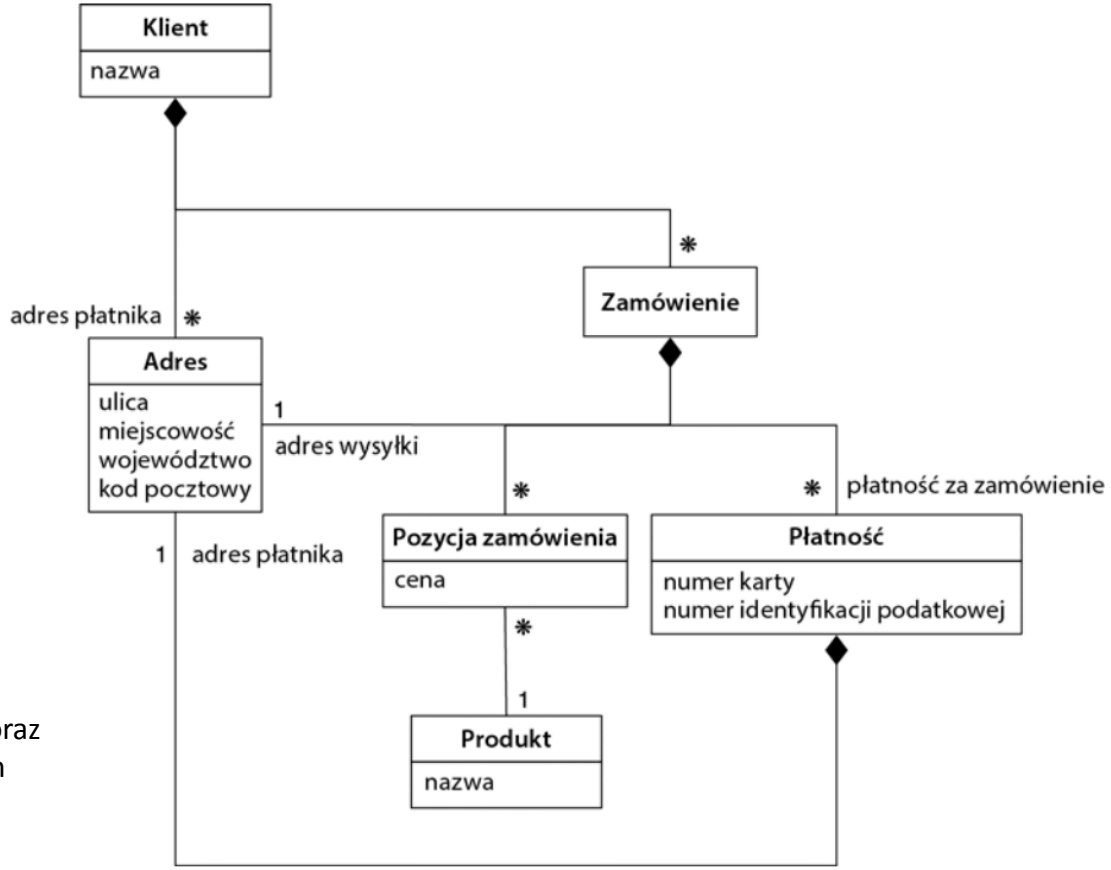
ZamowieniePlatnosc

Id	ZamowienieId	NumerKarty	AdresPlatnosciId	nipId
33	99	1000-1000	55	abelif879rft

Rysunek z książki: Sadalage P. J. oraz Fowler M. - NoSQL. Kompendium wiedzy

Agregacyjne modele danych

- Ten sam model ale w orientacji na agregowanie
  - mamy dwie podstawowe agregacje: klient oraz zamówienia (czarne romby pokazują strukturę agregacji)



Rysunek z książki: Sadalage P. J. oraz Fowler M. - NoSQL. Kompendium wiedzy

Agregacyjne modele danych

- Przykładowe dane dla modelu z poprzedniego slajdu (w formacie JSON)

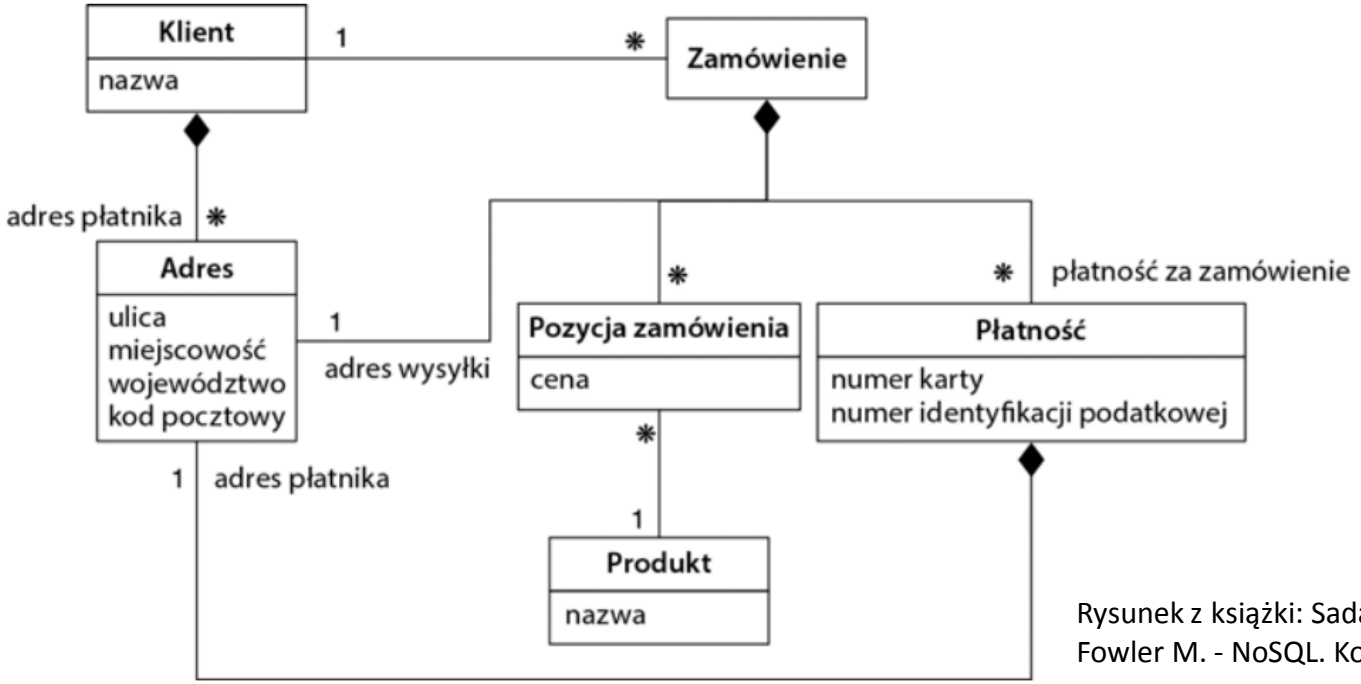
```
// w klientach
{
  "id":1,
  "nazwa":"Marcin",
  "adresPlatnika":
  [
    {"miasto":"Warszawa"}
  ]
}
```

```
// w zamówieniach
{
   "id":99,
   "klientId":1,
   "pozycjeZamowienia":[
      {
         "produktId":27,
         "cena":32.45,
         "produktNazwa":"NoSQL. Kompedium"
      }
   ],
   "adresWysylki":[
      {
         "miasto":"Warszawa"
      }
   ],
   "zamowieniePlatnosc":[
      {
         "numerKarty":"1000-1000-1000-1000",
         "NIP":"abelif879rft",
         "adresPlatnika":{
            "city":"Warszawa"
         }
      }
   ]
}
```

Użyto formatera kodów JSON:
https://jsonformatter.curiousconcept.com/

Agregacyjne modele danych

- Przykład jak poprzednio, ale inny sposób wyznaczenia granic między poszczególnymi agregacjami
  - uwaga: w tym podejściu już na etapie projektowania struktury bazy trzeba myśleć, w jakim układzie dane będą pobierane! W podejściu relacyjnym niemal nigdy to nie jest potrzebne (mamy przecież SQL!)



Rysunek z książki: Sadalage P. J. oraz Fowler M. - NoSQL. Kompendium wiedzy

Agregacyjne modele danych

- Przykładowe dane dla modelu z poprzedniego slajdu (w formacie JSON)

```
// w klientach
{
   "klient":{
      "id":1,
      "nazwa":"Marcin",
      "adresPlatnika":[
         {
            "miasto":"Warszawa"
         }
      ],
      "zamowienia":[
         {
            "id":99,
            "klientId":1,
            "pozycjeZamowienia":[
               {
                  "produktId":27,
                  "cena":32.45,
                  "produktNazwa":"NoSQL. Kompedium"
               }
            ],
            "adresWysylki":[
               {
                  "miasto":"Warszawa"
               }
            ],
            "zamowieniePlatnosc":[
               {
                  "numerKarty":"1000-1000-1000-1000",
                  "NIP":"abelif879rft",
                  "adresPlatnika":{
                     "city":"Warszawa"
                  }
               }
            ]
         }
      ]
   }
}
```

Bazy Key-Value

- Najprostsze z baz NoSQL
- Szybkie i skalowalne
- Używane, kiedy łatwość zapisu i pobierania danych jest ważniejsza od skomplikowanej struktury danych
- Przechowywane pary klucz: wartość
  - dostęp do danych tylko za pośrednictwem klucza
  - za pomocą klucza wstawiamy, pobieramy i usuwamy wartości
  - klucz jest jedynym sposobem dotarcia do wartości
- Baza nic nie wie o strukturze przechowywanych danych
  - można przechowywać co się chce, jedynym faktycznym ograniczeniem jest rozmiar danych
  - nie wymagają definiowania typów przechowywanych wartości
  - wynikająca z tego podstawowa wada: odpowiedniki relacyjnej operacji z klauzulą WHERE są bardzo nienaturalne

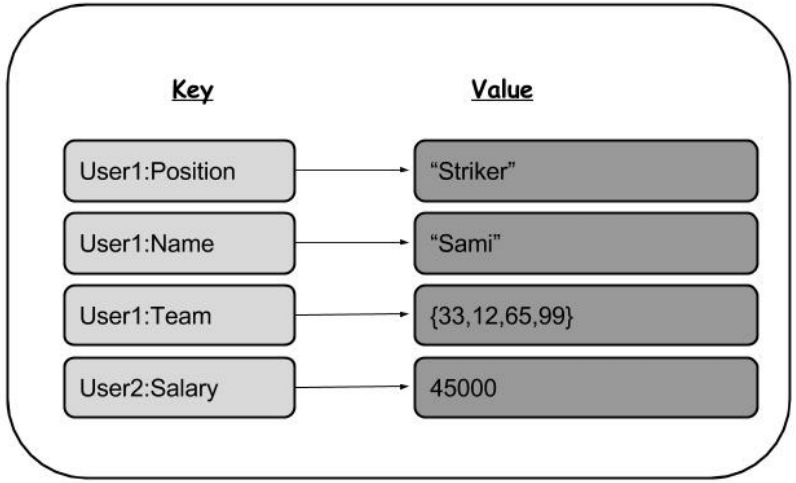
Bazy Key-Value

- Na pierwszy rzut oka, przypominają one bazy relacyjne, ponieważ również zawierają tabele, a klucze można interpretować, jako kolumny
  - w przeciwieństwie jednak do RDBMS, nie ma oczywiście narzuconego formatu. Każdy wiesz może zawierać różny zestaw kluczy
- korzysta z tablicy asocjacyjnej jako podstawowego modelu danych. W tym modelu dane są reprezentowane jako zbiór par klucz–wartość
  - tablica asocjacyjna (tablica skojarzeniowa, mapa, słownik, ang. associative array, map, dictionary) – nazwa dla powszechnie stosowanego w informatyce abstrakcyjnego typu danych, który przechowuje pary (unikatowy klucz, wartość) i umożliwia dostęp do wartości poprzez podanie klucza

Bazy Key-Value

- Przykłady

Key	Value
K1	AAA,BBB,CCC
K2	AAA,BBB
K3	AAA,DDD
K4	AAA,2,01/01/2015
K5	3,ZZZ,5623



Key	Value
AAAAA	11010011110101001101011 11...
AABAB	10011000010110011010111 10...
DFA766	00000000001010101101010 10...
FABCC4	11101101101010101001011 01...

Bazy dokumentowe

- Dokument jest głównym pojęciem baz dokumentów
  - np. XML, JSON, YAML, a także formy binarne tj. BSON
  - bazy dokumentowe „rozumieją” struktury przechowywanych agregacji
  - ograniczają to, co można w nich przechowywać, poprzez definicję dopuszczalnych struktur i typów

Key	Document
1001	{ "CustomerID": 99, "OrderItems": [{ "ProductID": 2010, "Quantity": 2, "Cost": 520 }, { "ProductID": 4365, "Quantity": 1, "Cost": 18 }], "OrderDate": "04/01/2017" }
1002	{ "CustomerID": 220, "OrderItems": [{ "ProductID": 1285, "Quantity": 1, "Cost": 120 }], "OrderDate": "05/08/2017" }

Bazy dokumentowe

- W praktyce różnice między bazami klucz-wartość i dokumentów częściowo się zacierają
  - w bazach danych klucz-wartość spodziewamy się pobierania danych przede wszystkim za pomocą klucza
  - w bazach dokumentów wykonujemy najczęściej pewnego rodzaju zapytanie na bazie wewnętrznej struktury dokumentu
- Dokumenty są adresowane w bazie danych za pomocą unikalnego klucza, który reprezentuje ten dokument
  - w tym wyraża się podobieństwo baz typu dokumentowego oraz baz typu key-value
  - jedną z innych cech charakterystycznych bazy danych zorientowanej na dokumenty jest to, że oprócz wyszukiwania klucza przeprowadzanego przez magazyn klucz–wartość, baza danych oferuje również interfejs API lub język zapytań, który pobiera dokumenty na podstawie ich zawartości

Bazy typu rodzina kolumn

- Zwane także *wide column databases*
- Są to bazy kolumnowe. Dane przechowywane są w porządku kolumnowym, nie wierszowym (jak to jest w klasycznych bazach relacyjnych)
  - wszystkie dane w jednej kolumnie są tego samego typu
  - kolumn może być bardzo wiele (setki, tysiące, dziesiątki tysięcy)
  - kolumna może liczyć sobie tysiące, setki tysięcy, miliony pozycji
  - złączenia są praktycznie niemożliwe, dlatego bazy kolumnowe w pewnym sensie odpowiadają zdenormalizowanym tabelom baz relacyjnych z wielką ilością kolumn i wierszy. Takie są np. tabele wymiarów i faktów w hurtowniach danych

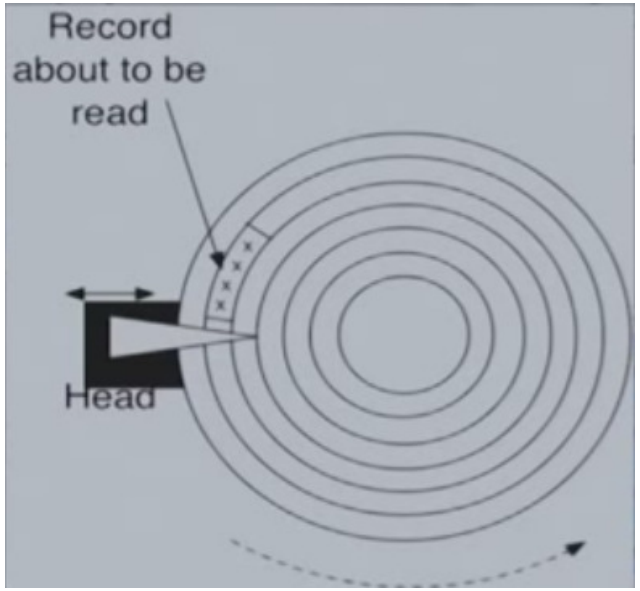
Bazy typu rodzina kolumn

- Z punktu widzenia użytkownika baza typu rodzina kolumn jest niemal tym samym, co klasyczna tabela relacyjna. Gdzie więc jest różnica?
  - w bazach relacyjnych dane są indeksowane w wierszach, w bazach RK są indeksowane w kolumnach. To diametralnie zmienia czasu dostępu do danych w zapytaniach analitycznych, które często wykorzystują tylko jedną (lub niewiele więcej) kolumn

```
1  Piotr   Zielinski
2  Artur   Gramacki
3  Jan     Kowalski
```

```
1, 2, 3, Piotr, Artur, Jan, Zielinski, Gramacki, Kowalski
```

  - bardzo zgrabne video pokazujące istotę RK: https://www.youtube.com/watch?v=mRvkikVuojU



Bazy typu rodzina kolumn

id	customer_id	date_ordered	date_shipped	sales_rep_id	total	payment_t...	order_fi...
97	201	1992-08-28 00:00:00	1992-09-17 00:00:00	12	84000.00	CREDIT	Y
98	202	1992-08-31 00:00:00	1992-09-10 00:00:00	14	595.00	CASH	Y
99	203	1992-08-31 00:00:00	1992-09-18 00:00:00	14	7707.00	CREDIT	Y
100	204	1992-08-31 00:00:00	1992-09-10 00:00:00	11	601100.00	CREDIT	Y
101	205	1992-08-31 00:00:00	1992-09-15 00:00:00	14	8056.60	CREDIT	Y
102	206	1992-09-01 00:00:00	1992-09-08 00:00:00	15	8335.00	CREDIT	Y
103	208	1992-09-02 00:00:00	1992-09-22 00:00:00	15	377.00	CASH	Y
104	208	1992-09-03 00:00:00	1992-09-23 00:00:00	15	32430.00	CREDIT	Y
105	209	1992-09-04 00:00:00	1992-09-18 00:00:00	11	2722.24	CREDIT	Y
106	210	1992-09-07 00:00:00	1992-09-15 00:00:00	12	15634.00	CREDIT	Y
107	211	1992-09-07 00:00:00	1992-09-21 00:00:00	15	142171.00	CREDIT	Y
108	212	1992-09-07 00:00:00	1992-09-10 00:00:00	13	149570.00	CREDIT	Y
109	213	1992-09-08 00:00:00	1992-09-28 00:00:00	11	1020935.00	CREDIT	Y
110	214	1992-09-09 00:00:00	1992-09-21 00:00:00	11	1539.13	CASH	Y
111	204	1992-09-09 00:00:00	1992-09-21 00:00:00	11	2770.00	CASH	Y
112	210	1992-08-31 00:00:00	1992-09-10 00:00:00	12	550.00	CREDIT	Y

- w takim zapytaniu jak niżej odpytujemy się tylko o 2 kolumny. W klasycznej bazie relacyjnej dane będą odczytywane całymi wierszami, w bazie RD odczytane zostaną z dysku tylko wymagane kolumny

```
SELECT AVG(total) FROM ord WHERE payment_type LIKE "CASH";
```

- z kolei zapytanie nieanalityczne (jak niżej) pewnie szybciej wykona się w klasycznej bazie relacyjnej

```
SELECT * FROM ord WHERE id=100;
```

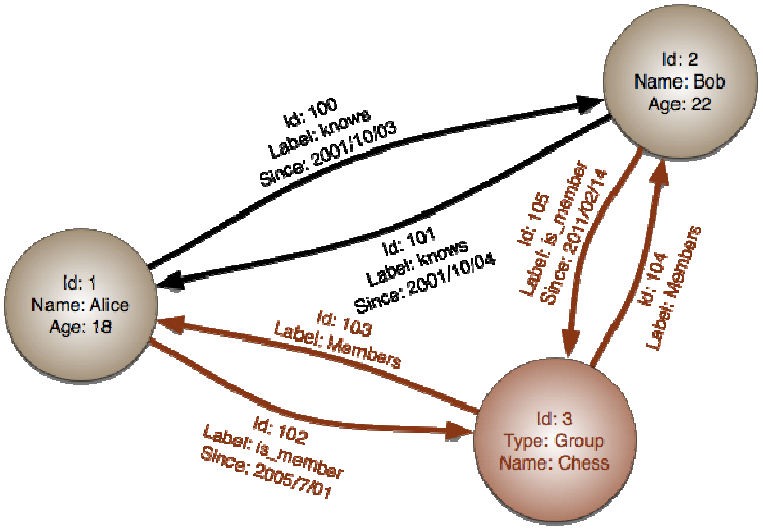
Bazy typu rodzina kolumn

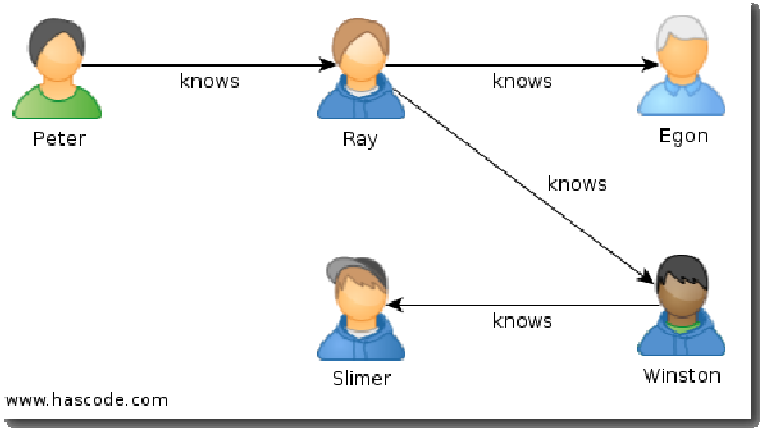
- Z powyższego wynika, że u podstaw baz typu RK leży ograniczenie do minimum ilość odczytywanych kolumn w zapytaniach (odczytujemy tylko te kolumny, które są potrzebne)
- Kompresja danych w bazach RK
  - każda kolumna może być kompresowana niezależnie
  - kompresja będzie bardzo efektywna (bo istnieje większe prawdopodobieństwo, że dane będą takie same w tej samej kolumnie niż z różnych)
  - przykład: jeżeli dane w kolumnie są posortowane, łatwo jest użyć algorytmu Run-Length Encoding (kodowanie długości serii)
    - AAA, AAA, AAA, BBB, BBB → AAA x 3, BBB x 2

Bazy grafowe

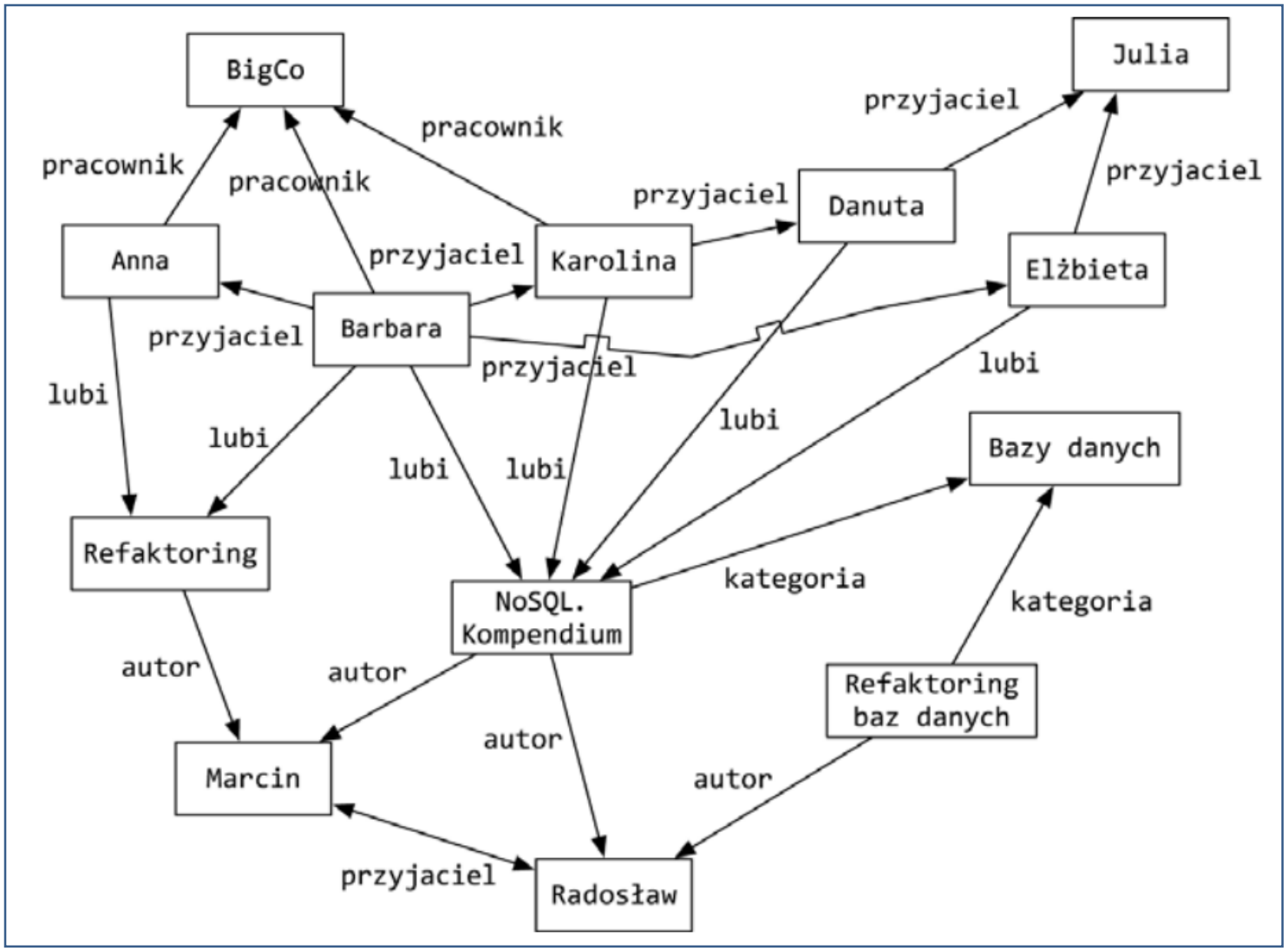
- Jeśli logika systemu wymaga algorytmów grafowych, wtedy bazy grafowe mogą znaczącą ułatwić oraz przyśpieszyć przetwarzanie
- Standardowym przykładem są portale społecznościowe , gdzie każda osoba może zostać opisana węzłem w grafie. Z kolei znajomi (kontakty) mogą być opisani za pomocą krawędzi (relacji)
  - w bazach grafowych mamy do czynienia z niewielkimi rekordami i rozbudowanymi połączeniami między nimi
  - węzły zawierają mało informacji, jest za to bardzo rozbudowana sieć połączeń
- Mając dane w formie grafu, bardzo łatwo znaleźć np. najkrótszą ścieżkę pomiędzy dwoma wierzchołkami
  - zbudowany graf (węzły i krawędzie) można odpytywać
  - Przykładowe zapytanie: znajdź książki z kategorii „bazy danych” napisane przez kogoś, kogo lubi ktoś z moich znajomych
- Najbardziej wyspecjalizowane ze wszystkich baz NoSQL

Bazy grafowe





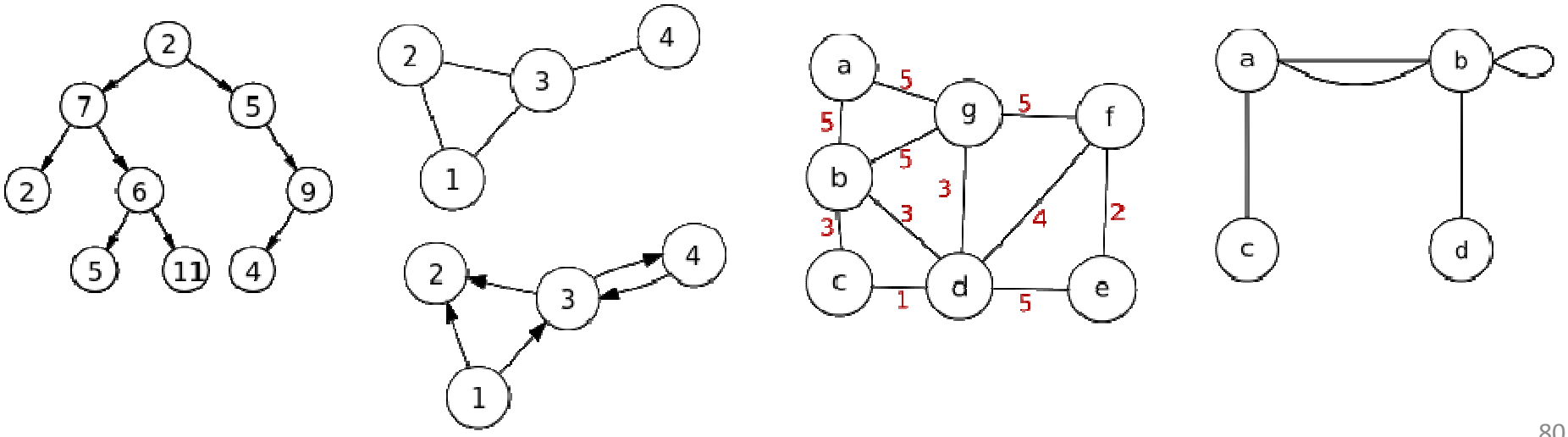
Bazy grafowe



Rysunek z książki: Sadalage P. J. oraz Fowler M. - NoSQL. Kompendium wiedzy

Trochę informacji na temat grafów

- Graf to podstawowy obiekt rozważań teorii grafów, struktura matematyczna służąca do przedstawiania i badania relacji między obiektami
  - w uproszczeniu graf to zbiór wierzchołków, które mogą być połączone krawędziami w taki sposób, że każda krawędź kończy się i zaczyna w którymś z wierzchołków
  - Najpopularniejsze rodzaje: graf typu drzewo, graf nieskierowany i skierowany, graf z wagami lub etykietami, multigraf i inne

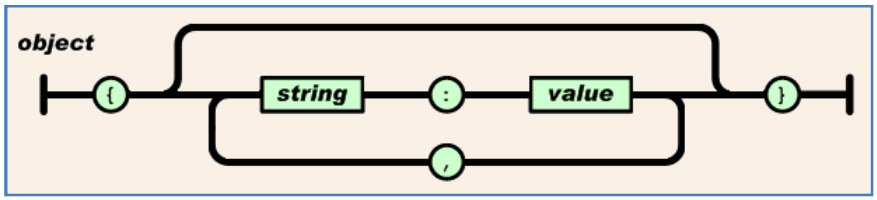


JSON

- JSON (JavaScript Object Notation)
- Lekki format wymiany danych komputerowych
  - tekstowy
  - pliki powinny być kodowane w UTF-8
- Podzbiór języka JavaScript
- Typ MIME dla formatu JSON to application/json
- Jeden z nieformalnych sposobów przekazywania danych do aplikacji opartych o AJAX
- Oparty o dwie struktury
  - zbiór par nazwa/wartość
  - uporządkowana lista wartości

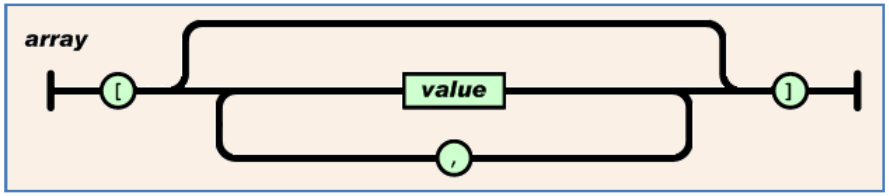
JSON

- **Obiekt** jest nieuporządkowanym zbiorem par nazwa/wartość



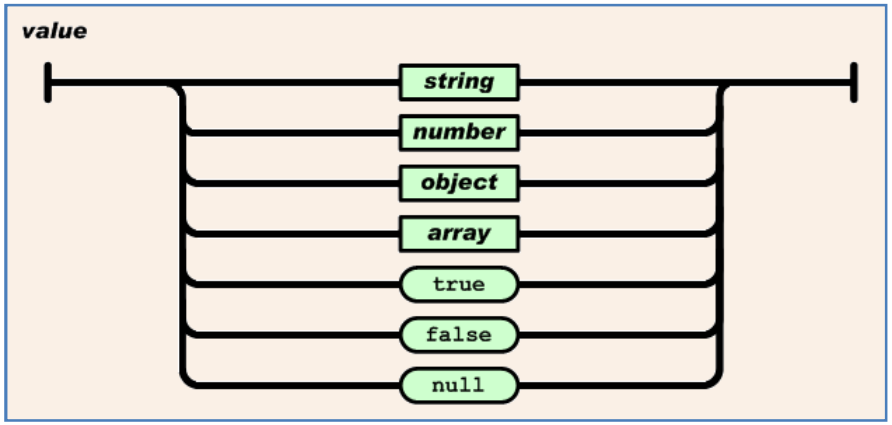
JSON

- **Tabela** jest uporządkowanym zbiorem wartości



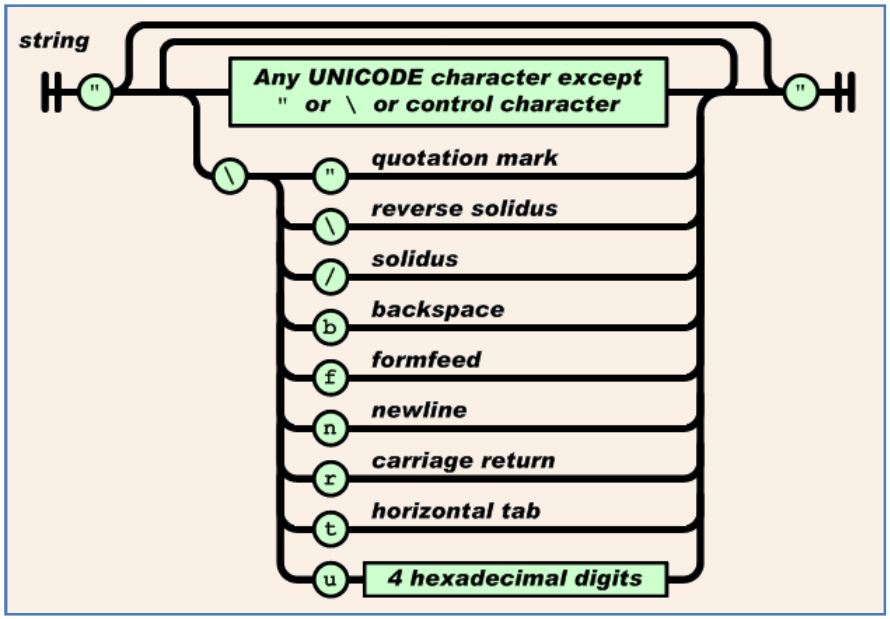
JSON

- **Wartość** to łańcuch znakowy



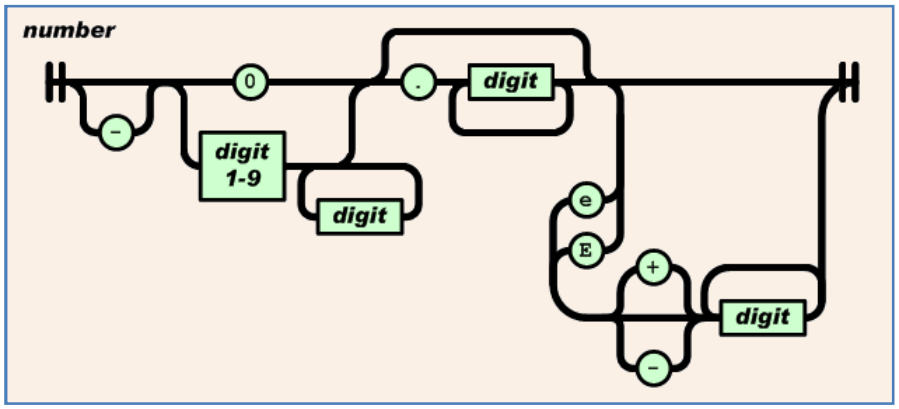
JSON

- **Łancuch znakowy** jest zbiorem zera lub większej ilości znaków Unicode



JSON

- **Liczby** zapisywane w formacie JSON są bardzo podobne do liczb w języku C lub Java, poza tym wyjątkiem, że nie używa się formatów ósemkowych i szesnastkowych

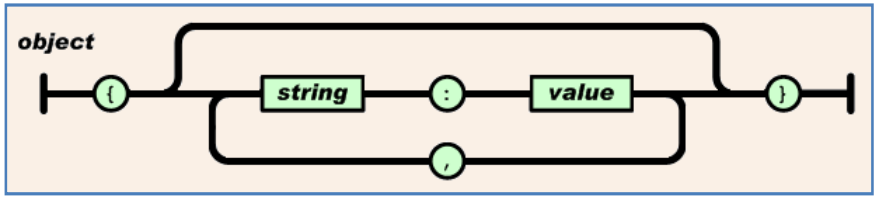


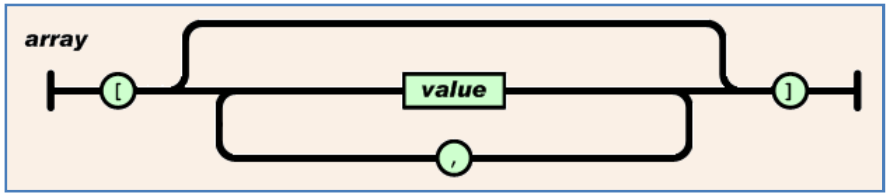
JSON

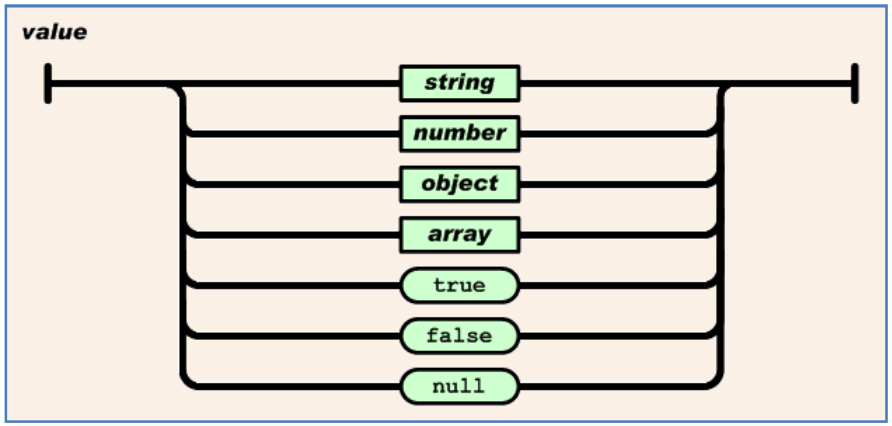
- **Obiekt** jest nieuporządkowanym zbiorem par nazwa/wartość.
- **Tabela** jest uporządkowanym zbiorem wartości
- **Wartość** to łańcuch znakowy
- **Łancuch znakowy** jest zbiorem zera lub większej ilości znaków Unicode
- **Liczby** zapisywane w formacie JSON są bardzo podobne do liczb w języku C lub Java, poza tym wyjątkiem, że nie używa się formatów ósemkowych i szesnastkowych

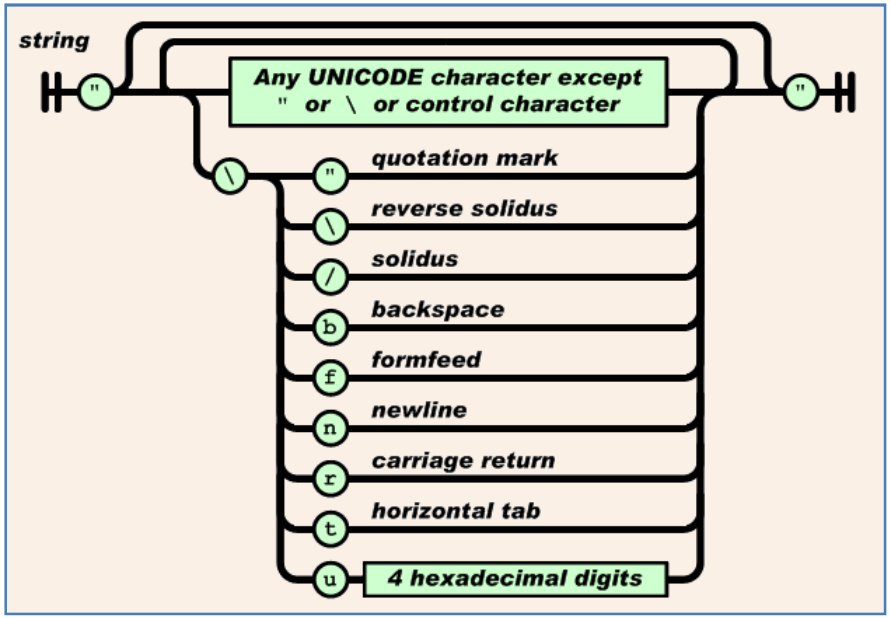
JSON

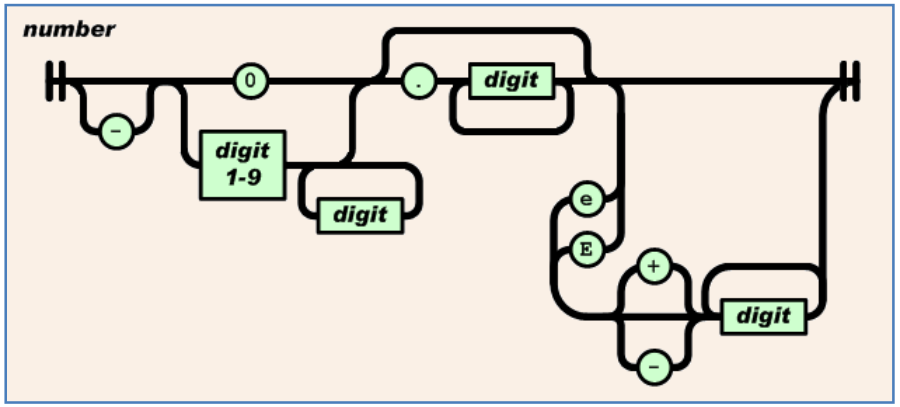
– https://www.json.org/json-pl.html











JSON vs XML

- Lżejszy, łatwiejszy do odczytu i zapisu
- Posiada typ tablicowy
- Pliki są bardziej "human readable"
- Nie ma wsparcia dla opisu sposobu wyświetlania
- Obsługuje skalarne typy danych
- Dzięki tablicom i obiektom można opisywać dane strukturalne

- Mniej prosty niż JSON
- Nie posiada typu tablicowego
- Pliki są mniej "human readable"
- Posiada wsparcie dla opisu sposobu wyświetlania (bo jest to język znaczników)
- Nie ma żadnego wsparcia dla opisu typów danych (informacje o typie przechowywane w XML Schema)

Oba formaty są:

- otwarte
- samodokumentujące
- hierarchiczne
- niezależne od języka programowania, który z nich korzysta
- łatwe do programowego parsowania
- mogą być źródłem danych w AJAX (korzystając z obiektu XMLHttpRequest)

JSON vs XML

JSON Example

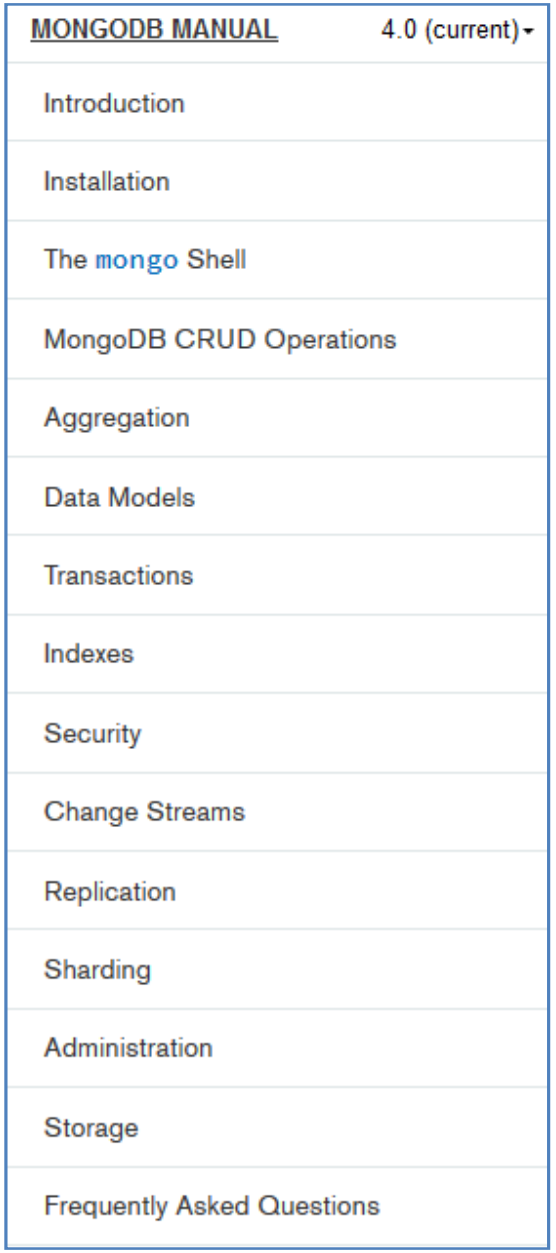
```
{"employees":[
    { "firstName":"John", "lastName":"Doe" },
    { "firstName":"Anna", "lastName":"Smith" },
    { "firstName":"Peter", "lastName":"Jones" }
]}
```

XML Example

```
<employees>
    <employee>
        <firstName>John</firstName> <lastName>Doe</lastName>
    </employee>
    <employee>
        <firstName>Anna</firstName> <lastName>Smith</lastName>
    </employee>
    <employee>
        <firstName>Peter</firstName> <lastName>Jones</lastName>
    </employee>
</employees>
```

MongoDB – materiały informacyjne

- MongoDB Tutorial
  - http://www.tutorialspoint.com/mongodb/index.htm
- MongoDB 3.0 Manual
  - https://docs.mongodb.org/manual/



MongoDB – podstawowe informacje

- Nazwa pochodzi od „humongous” (potężny)
  Open source
  - GNU AGPL wersja 3; Affero General Public License
- Zaimplementowana w C++
- Pierwsze wydanie stabilnej wersji w 2009 roku (prace rozpoczęły się w 2007 roku)
- Duża wydajność, skalowalność, niezawodność (poprzez replikację)
  - kosztem rezygnacji dla pełnego wsparcia transakcji. MongoDB oferuje atomowość na poziomie zapisu pojedynczego dokumentu. W przypadku modyfikacji wielu dokumentów w ramach jednej operacji istnieje możliwość modyfikacji przez inne wątki

MongoDB – podstawowe informacje

- Dane składowane są jako dokumenty w stylu BSON (binarny JSON)
  - zastosowanie elastycznego modelu danych opartego o standard BSON sprawia, że MongoDB jest idealnym rozwiązaniem w przypadkach, w których poszczególne obiekty różnią się zakresem przechowywanych atrybutów
  - do typowych zastosowań MongoDB należą katalogi produktów, platformy ankietowe, systemy zbierania i analizy logów czy zarządzania treścią (CMS)
- Prosty język zapytań – MongoDB Query
- Replikacja danych pozwalająca na łatwe skalowanie operacji odczytu

MongoDB – podstawowe informacje

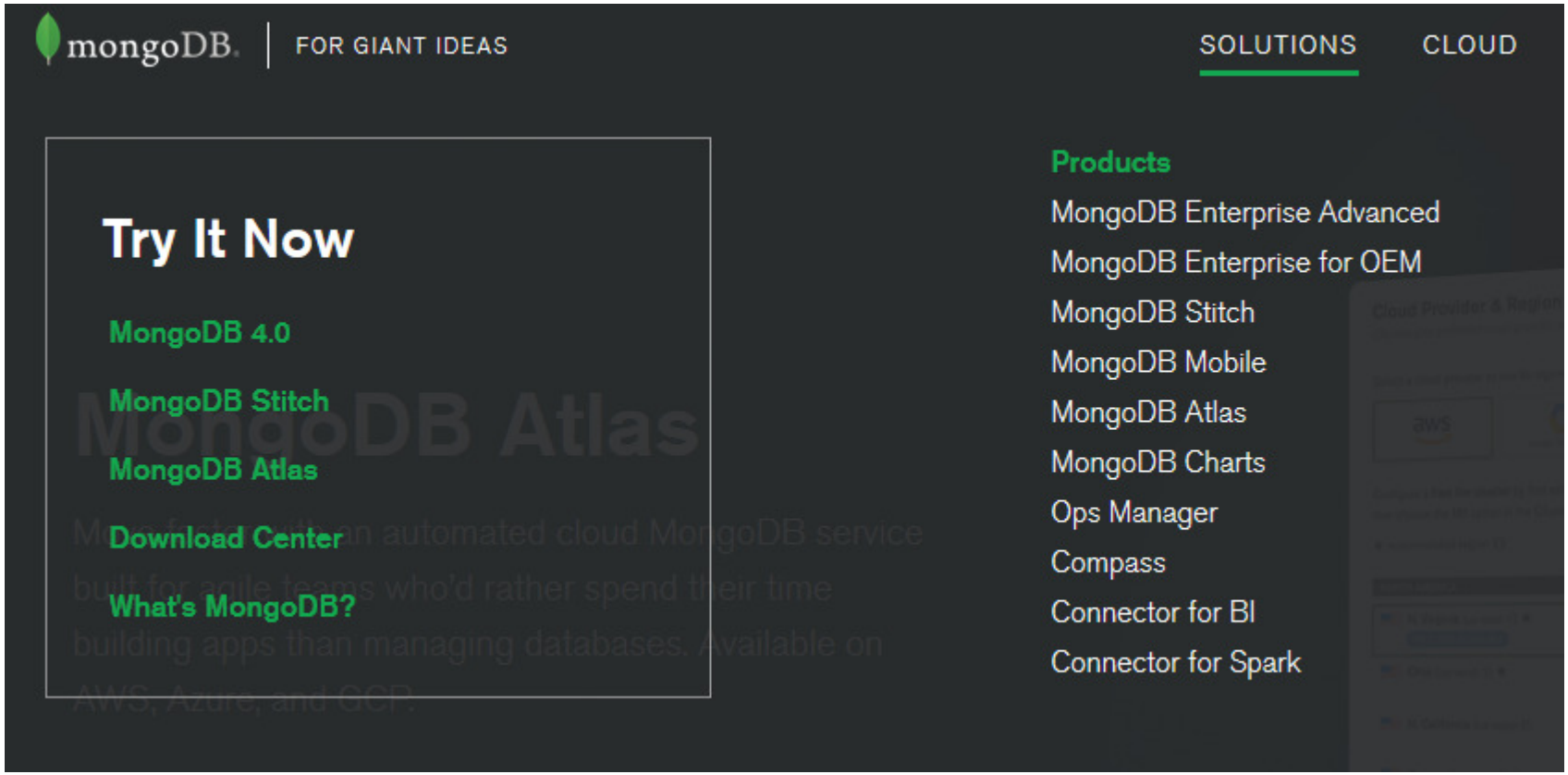
- Łatwe skalowanie poziome poprzez dodawanie kolejnych instancji
  - do budowy klastra MongoDB wykorzystywane są dwa mechanizmy: sharding (rodzaj partycjonowania) oraz replikacja
  - zastosowanie mechanizmów partycjonowania i replikacji umożliwia tworzenie w pełni skalowalnych oraz wysokodostępnych rozwiązań bez użycia kosztownego sprzętu
- Wiele dostępnych bibliotek dostępowych
  - pozwala to na integrację MongoDB z większością popularnych technologii używanych na rynku

MongoDB – narzędzia

- Sporo dostępnych narzędzi dostępowych
  - dostarczane przez zespół MongoDB
  - narzędzia innych dostawców (jest tego bardzo dużo, nie sposób wszystkie je omówić w tym miejscu)
- MongoDB Compass (dostępne na Windows, RedHat i Ubuntu, Mac OS)
  - graficzny przegląd danych
  - wykonywanie operacji CRUD na danych
  - szczegółowe informacje o indeksach
  - wykonywanie zapytań ad hoc
  - wyświetla graficzne plany wykonania zapytania aby ułatwić poprawę jego wydajności

MongoDB – narzędzia

- Inne narzędzia udostępniane przez twórców MongoDB



MongoDB – narzędzia

- MongoDB Atlas
  - użycie bazy danych jako usługi, konfigurację i zarządzanie klastrem
- MongoDB Cloud Manager
  - platforma do zarządzania MongoDB, działa w chmurze
- MongoDB Stitch
  - pozwala na wykorzystanie bazy MongoDB jako backendu w roli usługi, konfigurujemy bakcend naszej aplikacji w chmurze
- MongoDB Connector for BI (Business Intelligence)
  - podłączenie danych z bazy MongoDB do platformy analizy danych, do hurtowni danych (proces ETL)

MongoDB – narzędzia

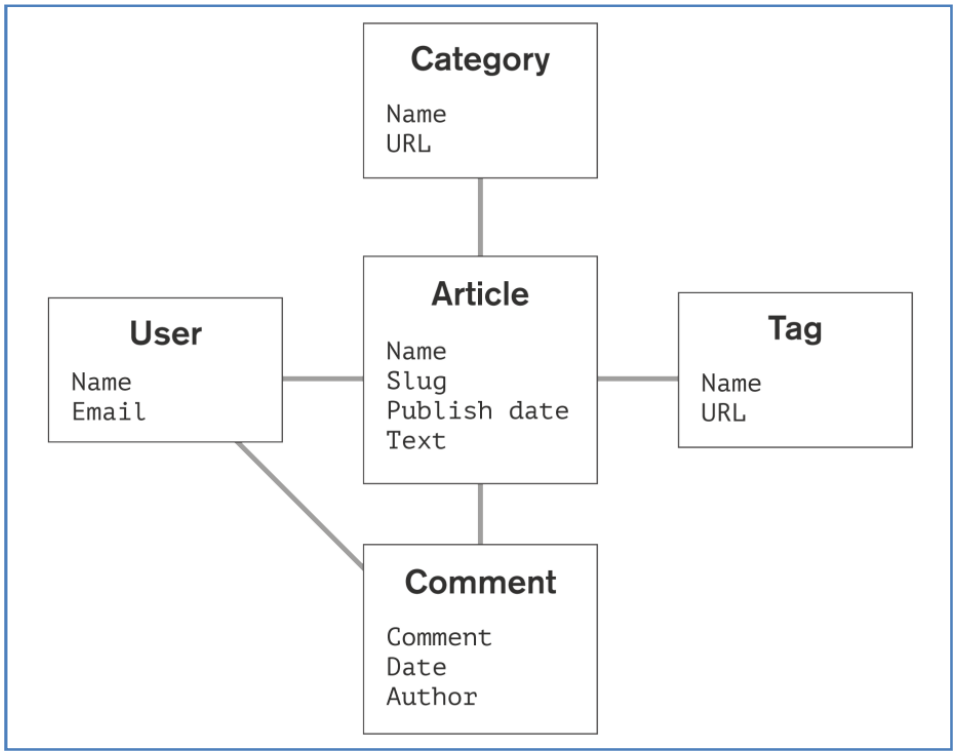
- Narzędzia od innych dostawców (bezpłatne, wybrane)
  - PHPMoAdmin - narzędzie do administracji MongoDB napisane w PHP
  - PHPmongoDB - zarządznie MongoDB w przeglądarce, napisane w PHP
  - Robomongo - dość proste ale wygodne GUI do zarządzania bazą MongoDB, jest zbudowane wokół mongo shell
  - NoSQL Viewer - tylko na windows, pozwala na przeglądanie danych ale na różne sposoby, wiele różnych baz NoSQL
  - Robo 3T (dawniej Robomongo) - przegląd i edycja danych
- Narzędzia od innych dostawców (płatne, wybrane)
  - Studio 3T (Free Edition) - wcześniej MongoChef Core, bardzo rozbudowane i bardzo funkcjonalne, trial na 30 dni

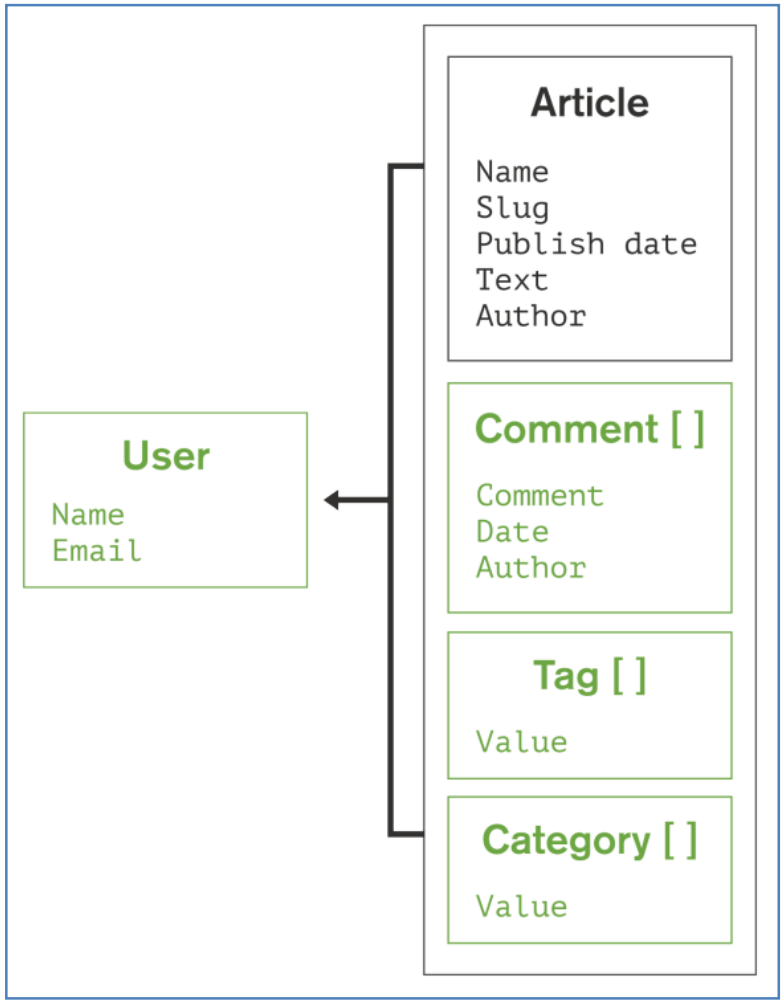
MongoDB – narzędzia

- Ops Manager
  - narzędzie do zarządzania MongoDB w centrum danych (monitorowanie, automatyzacja aktualizacji czy zarządzania indeksami, tworzenie backupu, optymalizacja zapytań),
- MongoDB Connector for Apache Spark
  - pozwala na podłączenie danych z MongoDB do Apache Spark i używanie wszystkich jego bibliotek (narzędzia analityczne: zapytania w SQL, uczenie maszynowe, przetwarzanie grafów, strumieniowanie danych)

MongoDB – podstawowe pojęcia

- Model relacyjny a model dokumentowy





MongoDB – podstawowe pojęcia

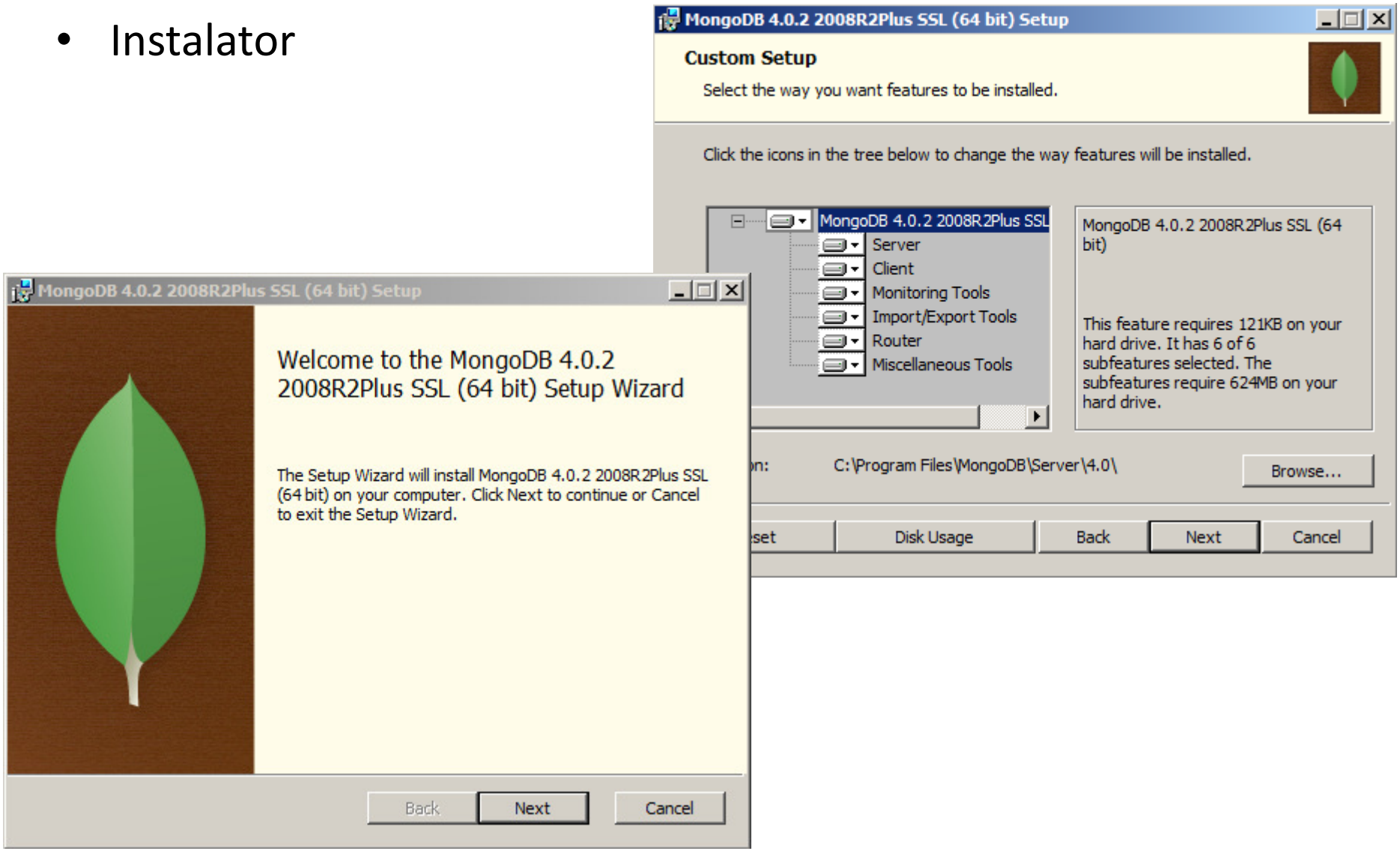
- Każda instancja → posiada wiele baz danych → a każda baza danych posiada wiele kolekcji → w kolekcji jest wiele dokumentów
- Baza danych
  - fizyczny kontener przechowujący kolekcje
  - pojedynczy serwer może obsługiwać wiele baz danych
  - każda baza danych posiada odrębne pliki
- Kolekcja
  - zbiór dokumentów (z grubsza odpowiednik tabeli w bazach relacyjnych)
- Dokument
  - zbiór par klucz:wartość (key-value)
  - Zapis w formacie JSON (BSON; binarny JSON). Nie wymusza określonego formatu dokumentów

MongoDB – terminologia

Oracle	MongoDB
instancja bazy danych	instancja MongoDB
schemat	baza danych
tabela	Kolekcja
wiersz	Dokument
rowid	Id
złączenie	DBRef

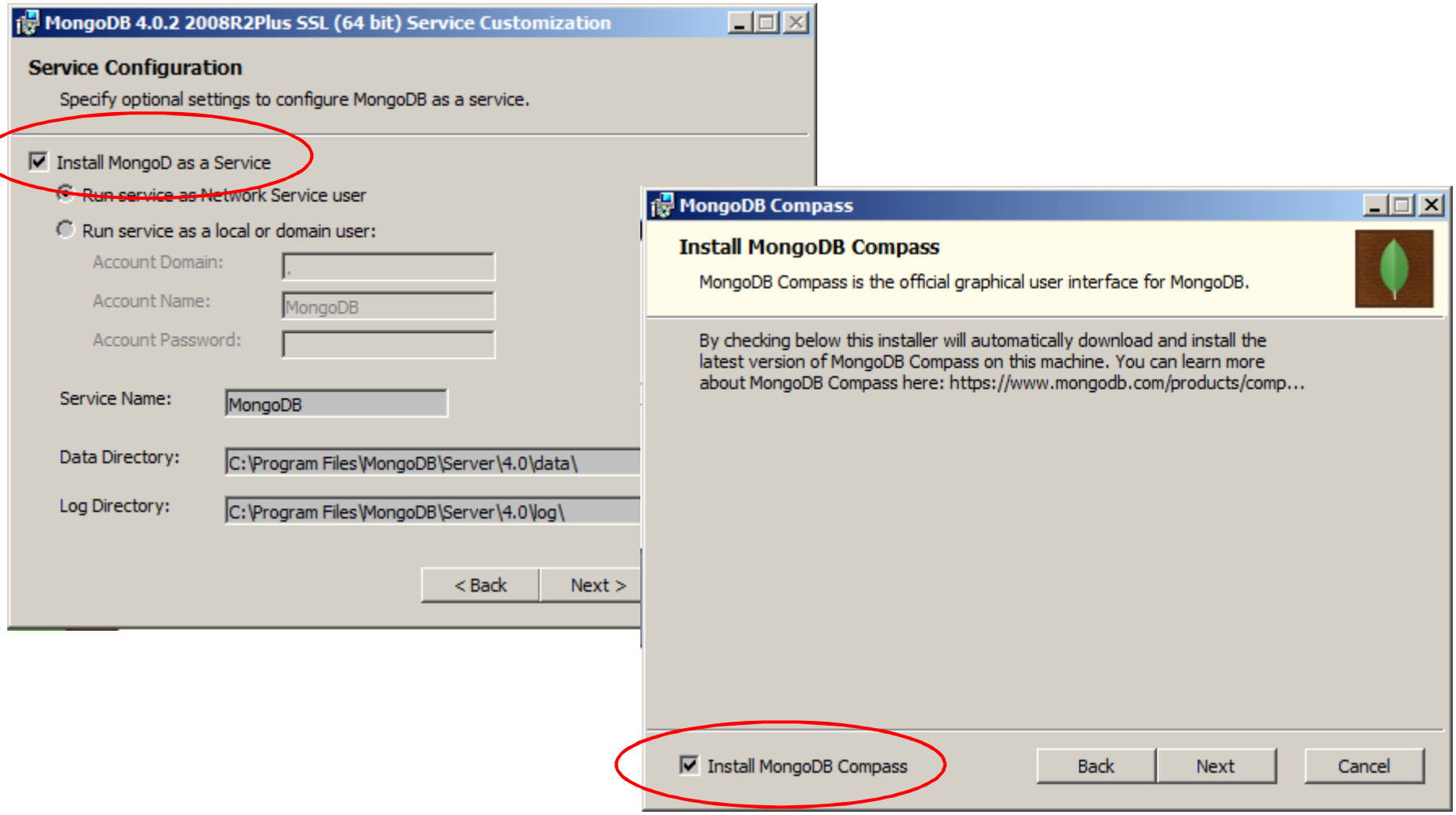
MongoDB – instalacja

- Instalator



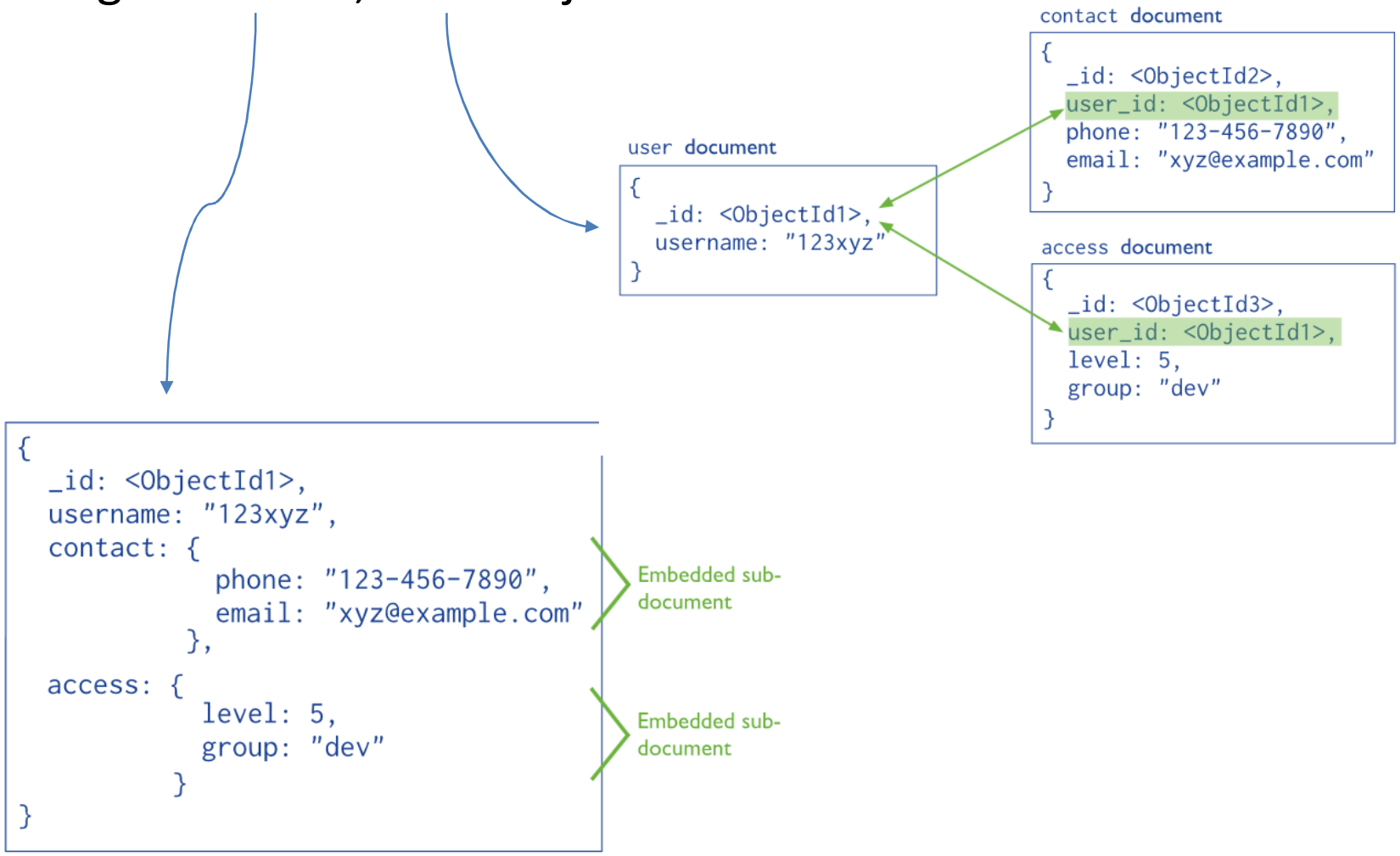
MongoDB – instalacja

- Instalator



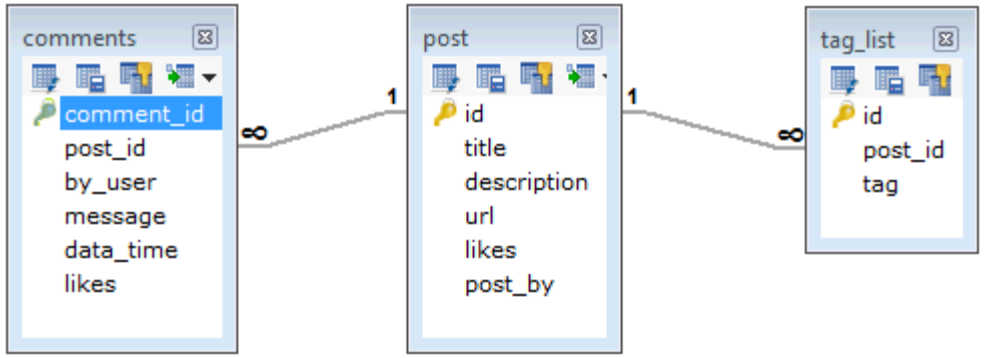
MongoDB – model danych

- Zagnieżdżanie, referencja



MongoDB – model danych

- przykład z https://www.tutorialspoint.com/mongodb/index.htm




```
{
   _id: POST_ID
   title: TITLE_OF_POST,
   description: POST_DESCRIPTION,
   by: POST_BY,
   url: URL_OF_POST,
   tags: [TAG1, TAG2, TAG3],
   likes: TOTAL_LIKES,
   comments: [
      {
         user:'COMMENT_BY',
         message: TEXT,
         dateCreated: DATE_TIME,
         like: LIKES
      },
      {
         user:'COMMENT_BY',
         message: TEXT,
         dateCreated: DATE_TIME,
         like: LIKES
      }
   ]
}
```

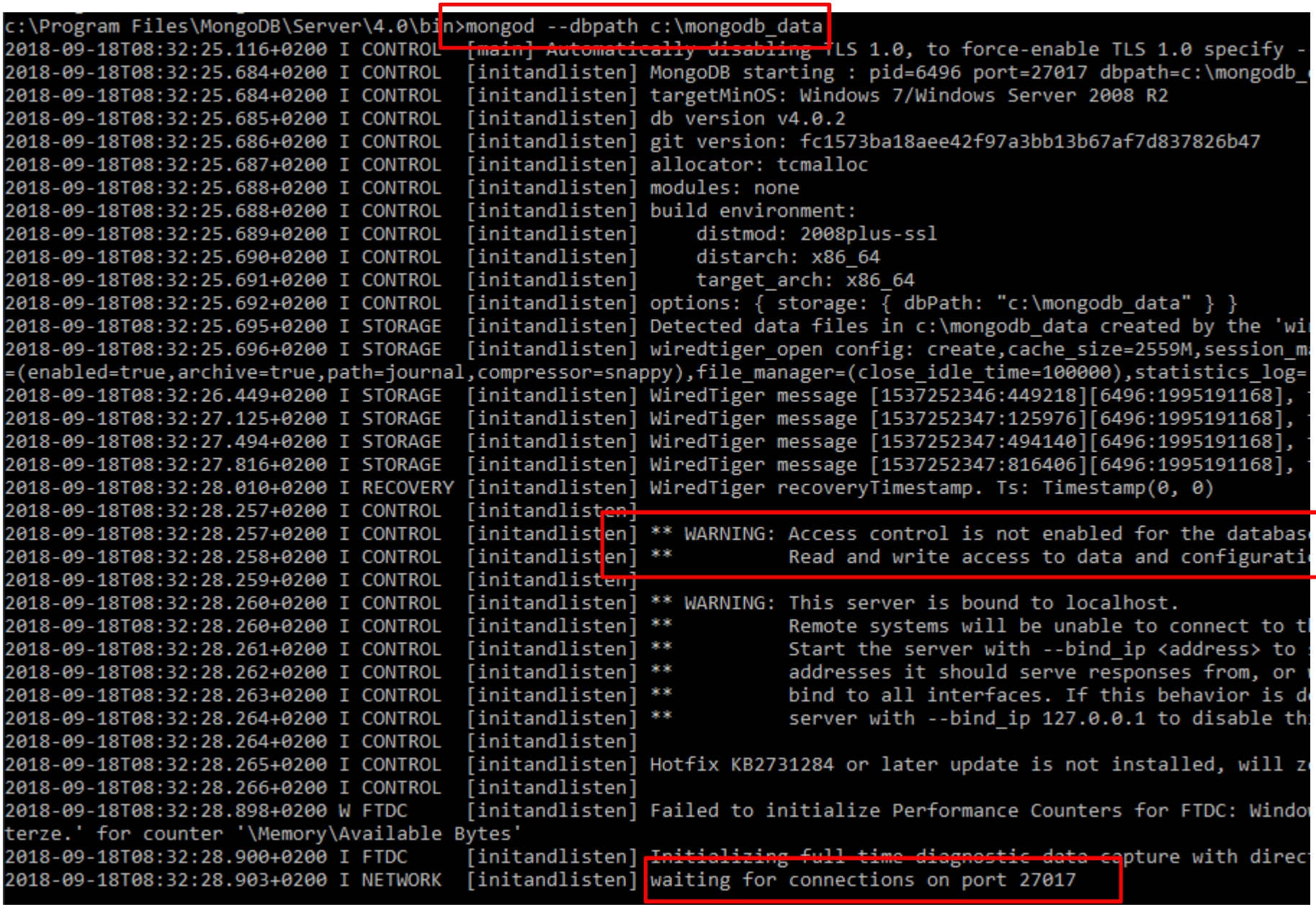
MongoDB – narzędzia, podstawy administracji

- Klient mongo
  - umożliwia połączenie się z serwerem MongoDB i wykonywanie poleceń/zapytań (obsługa danych, zadania administracyjne)
- Podłączenie się do procesu serwera mongod
  - wersja pełna (port domyślny to 27017)
    ```
    mongo --username <user> --password <pass> --host <host> --port NNNNN
    ```
  - wersja skrócona (port domyślny to 27017)
    ```
    mongo -u <user> -p <pass> -h <host> --port NNNNN
    ```
- Bardzo wiele parametrów uruchomieniowych dla procesu serwera oraz klienta, szczegóły patrz dokumentacja

MongoDB – narzędzia, podstawy administracji

- Zarządzanie procesem serwera mongod
  - serwer należy uruchomić wskazując katalog do składowania danych. Domyślny katalog to \data\db
  - serwer należy też uruchomić z aktywną kontrola dostępu (--auth), bez tego jest całkowicie "otwarty"
  - domyślnie nie ma utworzonego żadnego użytkownika, trzeba to zrobić samemu

```
mongod --port 27017 --dbpath /data/db1

use admin

db.createUser(
 {
   user: "myUserAdmin",
   pwd: "abc123",
   roles: [ { role: "userAdminAnyDatabase", db: "admin" } ]
  }
)
mongod --auth --port 27017 --dbpath /data/db1
```


c:\Program Files\MongoDB\Server\4.0\bin>mongod --dbpath c:\mongodb_data
2018-09-18T08:32:25.116+0200 I CONTROL [main] Automatically disabling TLS 1.0, to force-enable TLS 1.0 specify -
2018-09-18T08:32:25.684+0200 I CONTROL [initandlisten] MongoDB starting : pid=6496 port=27017 dbpath=c:\mongodb_
2018-09-18T08:32:25.684+0200 I CONTROL [initandlisten] targetMinOS: Windows 7/Windows Server 2008 R2
2018-09-18T08:32:25.685+0200 I CONTROL [initandlisten] db version v4.0.2
2018-09-18T08:32:25.686+0200 I CONTROL [initandlisten] git version: fc1573ba18aee42f97a3bb13b67af7d837826b47
2018-09-18T08:32:25.687+0200 I CONTROL [initandlisten] allocator: tcmalloc
2018-09-18T08:32:25.688+0200 I CONTROL [initandlisten] modules: none
2018-09-18T08:32:25.688+0200 I CONTROL [initandlisten] build environment:
2018-09-18T08:32:25.689+0200 I CONTROL [initandlisten] distmod: 2008plus-ssl
2018-09-18T08:32:25.690+0200 I CONTROL [initandlisten] distarch: x86_64
2018-09-18T08:32:25.691+0200 I CONTROL [initandlisten] target_arch: x86_64
2018-09-18T08:32:25.692+0200 I CONTROL [initandlisten] options: { storage: { dbPath: "c:\mongodb_data" } }
2018-09-18T08:32:25.695+0200 I STORAGE [initandlisten] Detected data files in c:\mongodb_data created by the 'wi
2018-09-18T08:32:25.696+0200 I STORAGE [initandlisten] wiredtiger_open config: create,cache_size=2559M,session_m
=(enabled=true,archive=true,path=journal,compressor=snappy),file_manager=(close_idle_time=100000),statistics_log=
2018-09-18T08:32:26.449+0200 I STORAGE [initandlisten] WiredTiger message [1537252346:449218][6496:1995191168],
2018-09-18T08:32:27.125+0200 I STORAGE [initandlisten] WiredTiger message [1537252347:125976][6496:1995191168],
2018-09-18T08:32:27.494+0200 I STORAGE [initandlisten] WiredTiger message [1537252347:494140][6496:1995191168],
2018-09-18T08:32:27.816+0200 I STORAGE [initandlisten] WiredTiger message [1537252347:816406][6496:1995191168],
2018-09-18T08:32:28.010+0200 I RECOVERY [initandlisten] WiredTiger recoveryTimestamp. Ts: Timestamp(0, 0)
2018-09-18T08:32:28.257+0200 I CONTROL [initandlisten]
2018-09-18T08:32:28.257+0200 I CONTROL [initandlisten] ** WARNING: Access control is not enabled for the databas
2018-09-18T08:32:28.258+0200 I CONTROL [initandlisten] ** Read and write access to data and configurati
2018-09-18T08:32:28.259+0200 I CONTROL [initandlisten]
2018-09-18T08:32:28.260+0200 I CONTROL [initandlisten] ** WARNING: This server is bound to localhost.
2018-09-18T08:32:28.260+0200 I CONTROL [initandlisten] ** Remote systems will be unable to connect to t
2018-09-18T08:32:28.261+0200 I CONTROL [initandlisten] ** Start the server with --bind_ip <address> to
2018-09-18T08:32:28.262+0200 I CONTROL [initandlisten] ** addresses it should serve responses from, or
2018-09-18T08:32:28.263+0200 I CONTROL [initandlisten] ** bind to all interfaces. If this behavior is d
2018-09-18T08:32:28.264+0200 I CONTROL [initandlisten] ** server with --bind_ip 127.0.0.1 to disable th
2018-09-18T08:32:28.264+0200 I CONTROL [initandlisten]
2018-09-18T08:32:28.265+0200 I CONTROL [initandlisten] Hotfix KB2731284 or later update is not installed, will z
2018-09-18T08:32:28.266+0200 I CONTROL [initandlisten]
2018-09-18T08:32:28.898+0200 W FTDC [initandlisten] Failed to initialize Performance Counters for FTDC: Windo
terze.' for counter '\Memory\Available Bytes'
2018-09-18T08:32:28.900+0200 I FTDC [initandlisten] Initializing full-time diagnostic data capture with direc
2018-09-18T08:32:28.903+0200 I NETWORK [initandlisten] waiting for connections on port 27017

MongoDB – narzędzia, podstawy administracji

- Uruchomienie klienta, gdy serwer uruchomiono bez opcji --auth

```
c:\Program Files\MongoDB\Server\4.0\bin>mongo
MongoDB shell version v4.0.2
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 4.0.2
Server has startup warnings:
2018-09-17T22:47:16.750+0200 I CONTROL  [initandlisten]
2018-09-17T22:47:16.750+0200 I CONTROL  [initandlisten] ** WARNING: Access control is not enabled for the database.
2018-09-17T22:47:16.750+0200 I CONTROL  [initandlisten] **          Read and write access to data and configuration
2018-09-17T22:47:16.751+0200 I CONTROL  [initandlisten]
2018-09-17T22:47:16.751+0200 I CONTROL  [initandlisten] ** WARNING: This server is bound to localhost.
2018-09-17T22:47:16.752+0200 I CONTROL  [initandlisten] **          Remote systems will be unable to connect to this
2018-09-17T22:47:16.753+0200 I CONTROL  [initandlisten] **          Start the server with --bind_ip <address> to spe
2018-09-17T22:47:16.753+0200 I CONTROL  [initandlisten] **          addresses it should serve responses from, or wit
2018-09-17T22:47:16.754+0200 I CONTROL  [initandlisten] **          bind to all interfaces. If this behavior is desi
2018-09-17T22:47:16.755+0200 I CONTROL  [initandlisten] **          server with --bind_ip 127.0.0.1 to disable this
2018-09-17T22:47:16.756+0200 I CONTROL  [initandlisten]
2018-09-17T22:47:16.757+0200 I CONTROL  [initandlisten] Hotfix KB2731284 or later update is not installed, will zero
2018-09-17T22:47:16.757+0200 I CONTROL  [initandlisten]
---
Enable MongoDB's free cloud-based monitoring service, which will then receive and display
metrics about your deployment (disk utilization, CPU, operation statistics, etc).

The monitoring data will be available on a MongoDB website with a unique URL accessible to you
and anyone you share the URL with. MongoDB may use this information to make product
improvements and to suggest MongoDB products and deployment options to you.

To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---

>
```

```
c:\Program Files\MongoDB\Server\4.0\bin>mongod --dbpath c:\mongodb_data --auth
2018-09-18T08:28:36.790+0200 I CONTROL  [main] Automatically disabling TLS 1.0, to force-enable TLS 1.0 spec
'
2018-09-18T08:28:37.353+0200 I CONTROL  [initandlisten] MongoDB starting : pid=5820 port=27017 dbpath=c:\mong
2018-09-18T08:28:37.353+0200 I CONTROL  [initandlisten] targetMinOS: Windows 7/Windows Server 2008 R2
2018-09-18T08:28:37.353+0200 I CONTROL  [initandlisten] db version v4.0.2
2018-09-18T08:28:37.353+0200 I CONTROL  [initandlisten] git version: fc1573ba18aee42f97a3bb13b67af7d837826b47
2018-09-18T08:28:37.353+0200 I CONTROL  [initandlisten] allocator: tcmalloc
2018-09-18T08:28:37.354+0200 I CONTROL  [initandlisten] modules: none
2018-09-18T08:28:37.354+0200 I CONTROL  [initandlisten] build environment:
2018-09-18T08:28:37.354+0200 I CONTROL  [initandlisten]     distmod: 2008plus-ssl
2018-09-18T08:28:37.354+0200 I CONTROL  [initandlisten]     distarch: x86_64
2018-09-18T08:28:37.354+0200 I CONTROL  [initandlisten]     target_arch: x86_64
2018-09-18T08:28:37.354+0200 I CONTROL  [initandlisten] options: { security: { authorization: "enabled" }, st
ta" } }
2018-09-18T08:28:37.357+0200 I STORAGE  [initandlisten] Detected data files in c:\mongodb_data created by the
o setting the active storage engine to 'wiredTiger'.
2018-09-18T08:28:37.358+0200 I STORAGE  [initandlisten] wiredtiger_open config: create,cache_size=2559M,sessi
min=4,threads_max=4),config_base=false,statistics=(fast),log=(enabled=true,archive=true,path=journal,compress
idle_time=100000),statistics_log=(wait=0),verbose=(recovery_progress),
2018-09-18T08:28:38.380+0200 I STORAGE  [initandlisten] WiredTiger message [1537252118:380859][5820:19951911
loop: starting at 21/7424
2018-09-18T08:28:39.034+0200 I STORAGE  [initandlisten] WiredTiger message [1537252119:34179][5820:199519116
21 through 22
2018-09-18T08:28:39.482+0200 I STORAGE  [initandlisten] WiredTiger message [1537252119:482421][5820:19951911
 22 through 22
2018-09-18T08:28:39.740+0200 I STORAGE  [initandlisten] WiredTiger message [1537252119:740234][5820:19951911
overy timestamp: 0
2018-09-18T08:28:40.473+0200 I RECOVERY [initandlisten] WiredTiger recoveryTimestamp. Ts: Timestamp(0, 0)
2018-09-18T08:28:41.935+0200 I CONTROL  [initandlisten]
2018-09-18T08:28:41.935+0200 I CONTROL  [initandlisten] ** WARNING: This server is bound to localhost.
2018-09-18T08:28:41.935+0200 I CONTROL  [initandlisten] **          Remote systems will be unable to connect
2018-09-18T08:28:41.935+0200 I CONTROL  [initandlisten] **          Start the server with --bind_ip <address>
2018-09-18T08:28:41.935+0200 I CONTROL  [initandlisten] **          addresses it should serve responses from
2018-09-18T08:28:41.935+0200 I CONTROL  [initandlisten] **          bind to all interfaces. If this behavior
2018-09-18T08:28:41.935+0200 I CONTROL  [initandlisten] **          server with --bind_ip 127.0.0.1 to disab
2018-09-18T08:28:41.935+0200 I CONTROL  [initandlisten]
2018-09-18T08:28:41.935+0200 I CONTROL  [initandlisten] Hotfix KB2731284 or later update is not installed, wi
2018-09-18T08:28:41.935+0200 I CONTROL  [initandlisten]
2018-09-18T08:28:43.101+0200 W FTDC     [initandlisten] Failed to initialize Performance Counters for FTDC: W
rPathW failed with 'Podanego obiektu nie znaleziono na komputerze.' for counter '\Memory\Available Bytes'
2018-09-18T08:28:43.101+0200 I FTDC     [initandlisten] Initializing full-time diagnostic data capture with d
ostic.data'
2018-09-18T08:28:43.108+0200 I NETWORK  [initandlisten] waiting for connections on port 27017
```

MongoDB – narzędzia, podstawy administracji

- Utworzenie użytkownika z uprawnieniami administratora

```
> use admin
switched to db admin
> db.createUser(
...   {
...     user: "myUserAdmin",
...     pwd: "abc123",
...     roles: [ { role: "userAdminAnyDatabase", db: "admin" } ]
...   }
... )
2018-09-17T22:11:17.408+0200 I NETWORK  [js] trying reconnect to 127.0.0.1:27017 failed
2018-09-17T22:11:17.421+0200 I NETWORK  [js] reconnect 127.0.0.1:27017 ok
Successfully added user: {
        "user" : "myUserAdmin",
        "roles" : [
                {
                        "role" : "userAdminAnyDatabase",
                        "db" : "admin"
                }
        ]
}
>
```

```
c:\Program Files\MongoDB\Server\4.0\bin>mongo --port 27017 -u "myUserAdmin" -p "abc123" --authenticationDatabase "admin"
MongoDB shell version v4.0.2
connecting to: mongodb://127.0.0.1:27017/
MongoDB server version: 4.0.2
>
```

MongoDB – narzędzia, podstawy administracji

- Utworzenie użytkownika testowego (bez uprawnień admin-a)

```
> use test
switched to db test
> db.createUser(
...   {
...     user: "myTester",
...     pwd: "xyz123",
...     roles: [ { role: "readWrite", db: "test" },
...              { role: "read", db: "reporting" } ]
...   }
... )
Successfully added user: {
        "user" : "myTester",
        "roles" : [
                {
                        "role" : "readWrite",
                        "db" : "test"
                },
                {
                        "role" : "read",
                        "db" : "reporting"
                }
        ]
}
>
```

MongoDB – narzędzia, podstawy administracji

- Bez opcji --auth każdy może zrobić wszystko, np. wykonać shutdown całego serwera!

```
> use admin
switched to db admin
> db.shutdownServer()
server should be down...
2018-09-17T22:53:15.917+0200 I NETWORK  [js] trying reconnect to 127.0.0.1:27017 failed
2018-09-17T22:53:16.919+0200 I NETWORK  [js] reconnect 127.0.0.1:27017 failed failed
>
```

```
2018-09-17T22:53:15.324+0200 I NETWORK  [conn1] shutdown: going to close listening sockets...
2018-09-17T22:53:15.326+0200 I CONTROL  [conn1] Shutting down free monitoring
2018-09-17T22:53:15.327+0200 I FTDC     [conn1] Shutting down full-time diagnostic data capture
2018-09-17T22:53:15.330+0200 I STORAGE  [conn1] WiredTigerKVEngine shutting down
2018-09-17T22:53:15.909+0200 I STORAGE  [conn1] shutdown: removing fs lock...
2018-09-17T22:53:15.910+0200 I CONTROL  [conn1] now exiting
2018-09-17T22:53:15.911+0200 I CONTROL  [conn1] shutting down with code:0
```

- Do pierwszych ćwiczeń z MongoDB nie musimy uruchamiać serwera z opcją --auth

MongoDB – narzędzia, podstawy administracji

- Uprawnienia, role
  - Database User Roles
    - read, readWrite
  - Database Administration Roles
    - dbAdmin, dbOwner, userAdmin
  - Cluster Administration Roles
    - clusterAdmin, clusterManager, clusterMonitor, hostManager
  - Backup and Restoration Roles
    - buckup, restore
  - All-Database Roles
    - readAnyDatabase, readWriteAnyDatabase, userAdminAnyDatabase, dbAdminAnyDatabase
  - Superuser Roles
    - root Provides access to the operations and all the resources of the
  - Internal Role
    - __system (Provides privileges to take any action against any object in the database. Do not assign this role to user objects representing applications or human administrators, other than in exceptional circumstances)

MongoDB – narzędzia, podstawy administracji

- Uprawnienia, role

Database User Roles

Every database includes the following roles:

Role	Short Description
`read`	Provides the ability to read data on all *non*-system collections and on the following system collections: `system.indexes`, `system.js`, and `system.namespaces` collections. For the specific privileges granted by the role see `read`.
`readWrite`	Provides all the privileges of the `read` role plus ability to modify data on all *non*-system collections and the `system.js` collection. For the specific privileges granted by the role, see `readWrite`.

MongoDB – narzędzia, podstawy administracji

- Uprawnienia, role

Database Administration Roles

Every database includes the following database administration roles:

Role	Short Description
`dbAdmin`	Provides the ability to perform administrative tasks such as schema-related tasks, indexing, and gathering statistics. This role does not grant privileges for user and role management. For the specific privileges granted by the role, see `dbAdmin`.
`dbOwner`	The database owner can perform any administrative action on the database. This role combines the privileges granted by the `readWrite`, `dbAdmin` and `userAdmin` roles.
`userAdmin`	Provides the ability to create and modify roles and users on the current database. Since the `userAdmin` role allows users to grant any privilege to any user, including themselves, the role also indirectly provides superuser access to either the database or, if scoped to the `admin` database, the cluster. For the specific privileges granted by the role, see `userAdmin`.

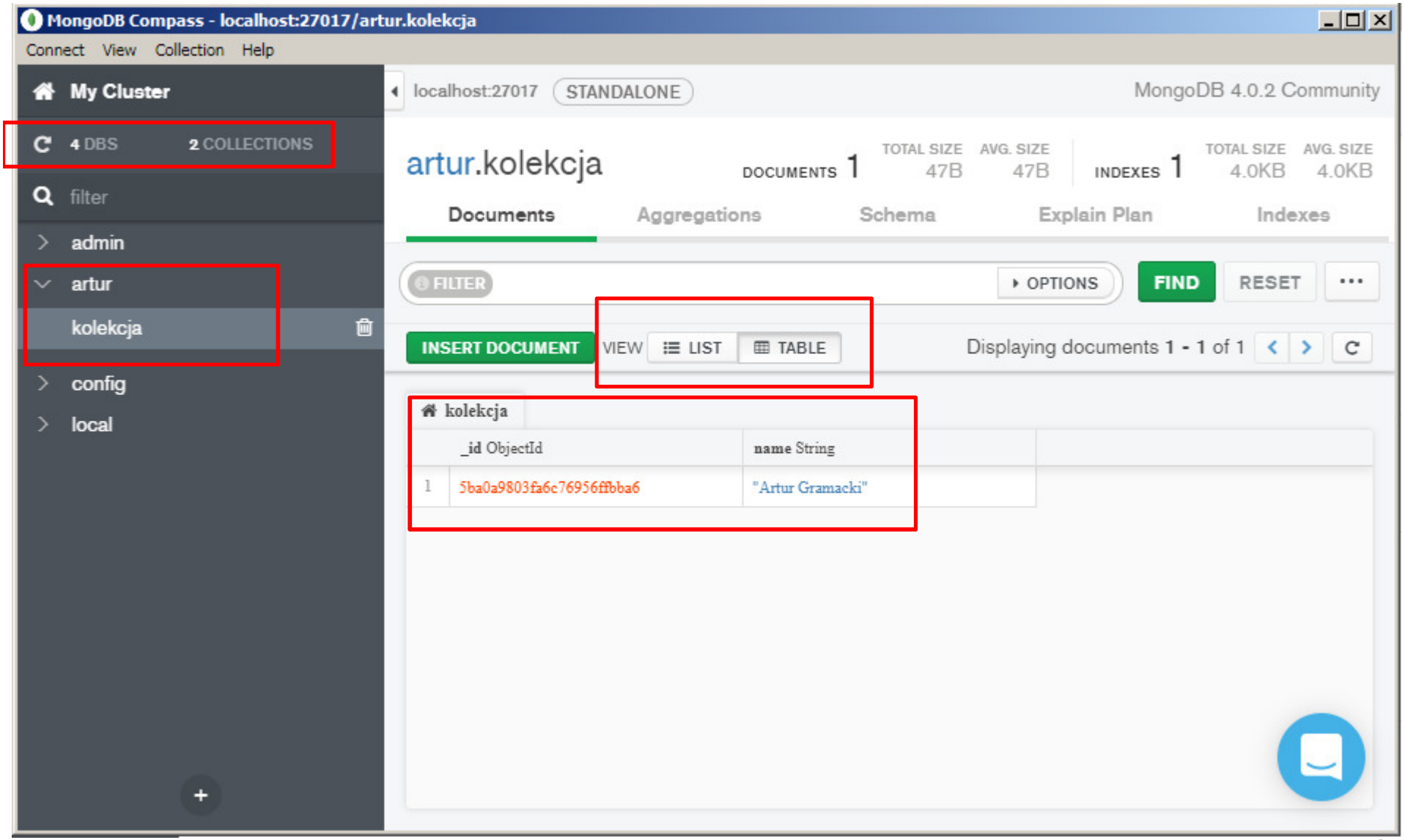
MongoDB – podstawy użytkowania

- Bazy danych
  - listowanie istniejących baz
  - wybieranie bazy
  - tworzenie nowej bazy danych
  - usuwanie bieżącej bazy danych

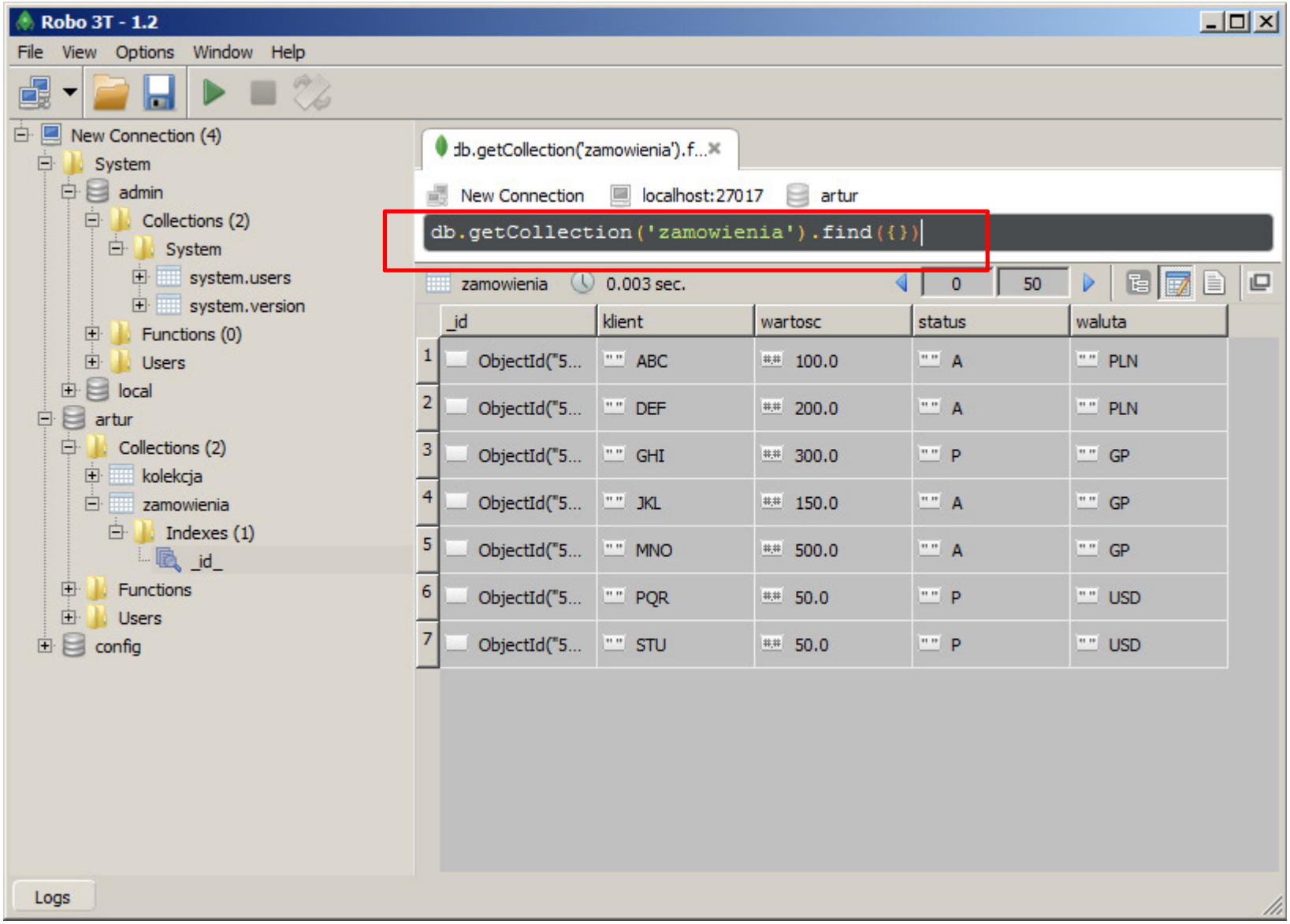
```
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB

> use artur
switched to db artur

> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB

> db.artur.insert({"name":"Artur Gramacki"})
WriteResult({ "nInserted" : 1 })

> show dbs
admin   0.000GB
artur   0.000GB
config  0.000GB
local   0.000GB

> use artur
switched to db artur

> db.dropDatabase()
{ "dropped" : "artur", "ok" : 1 }
>
```

Poleceniem use tworzona jest nowa baza danych. Nie widać jej jednak od razu na liście. Aby ją zobaczyć trzeba do niej wstawić przynajmniej jeden dokument (jakikolwiek).

MongoDB – narzędzie Compass



MongoDB – narzędzie Robo 3T



MongoDB – podstawy użytkowania

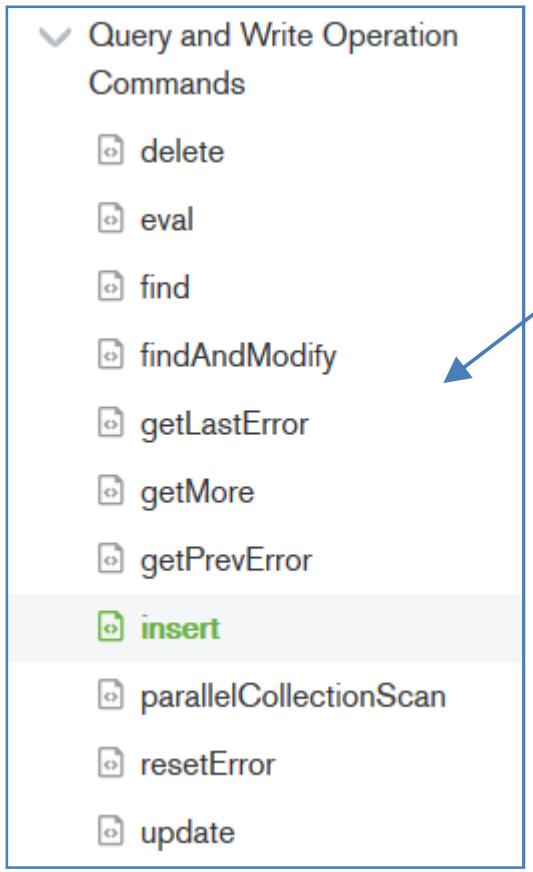
- Plik konfiguracyjny
  - odczytywany podczas startu instancji mongod
  - Linux: /etc/mongod.conf
  - Windows: <install directory>/bin/mongod.cfg
  - uruchamianie instacji z plikiem konfiguracyjnym
    `mongod --config /etc/mongod.conf`
  - bardzo dużo opcji, szczegóły patrz dokumentacja
  - używa formatu YAML
    - Uwaga: znak tabulatora nie jest obsługiwany w tym formacie, używać wyłącznie spacji

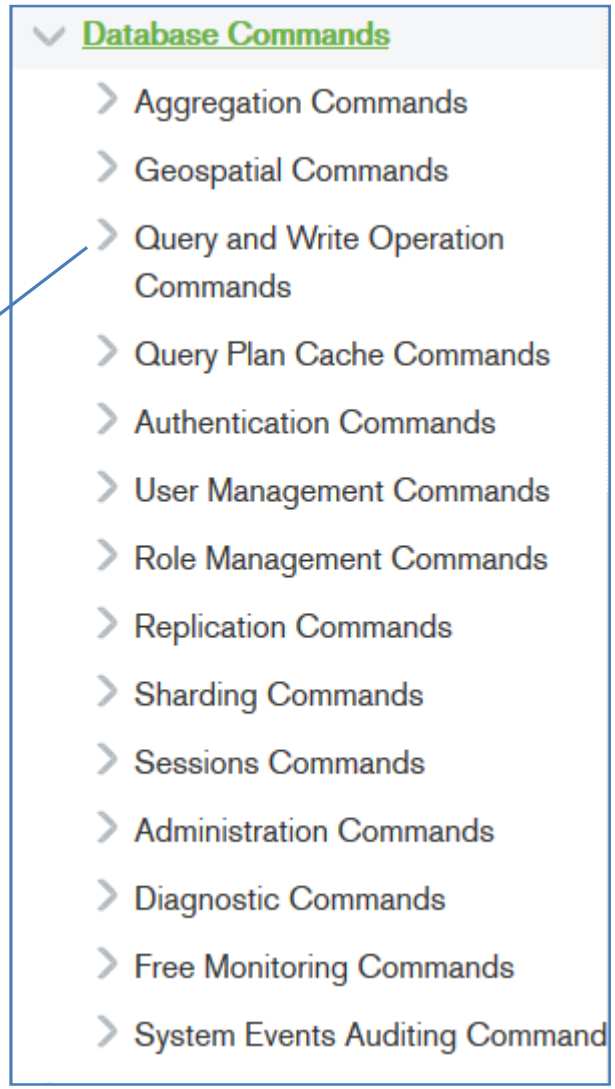
```
systemLog:
   destination: file
   path: "/var/log/mongodb/mongod.log"
   logAppend: true
storage:
   journal:
      enabled: true
processManagement:
   fork: true
net:
   bindIp: 127.0.0.1
   port: 27017
setParameter:
   enableLocalhostAuthBypass: false
...
```

MongoDB – podstawy użytkowania

- Komendy bazy danych

```
db.runCommand( { <command> } )
db.adminCommand( { <command> } )
```

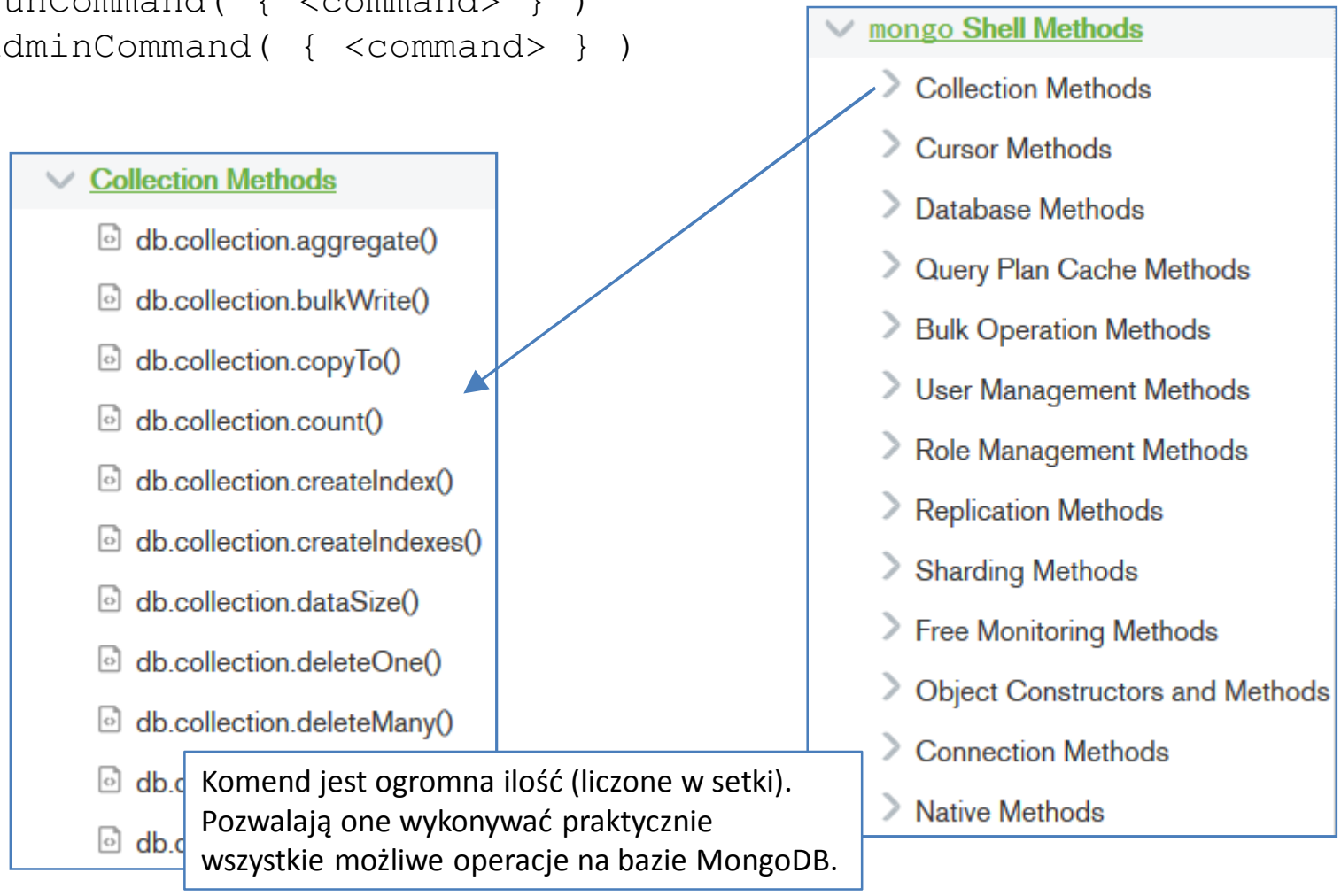




MongoDB – podstawy użytkowania

- Metody programu (powłoki; ang. shell) mongo

```
db.runCommand( { <command> } )
db.adminCommand( { <command> } )
```



MongoDB – podstawy użytkowania

- Tworzenie kolekcji i wstawianie do niej dokumentów

```
use artur

db.zamowienia.insert(
{
  klient: 'Jan Kowalski',
  opis: 'Zamówienie złożone przez internet',
  wartosc: '1024',
  waluta: 'PLN',
  data: new Date(2018,10,01,8,00),
  pozycje:
  [
    {nazwa: 'toner', wartosc: '100'},
    {nazwa: 'płyty DVD', wartosc: '56'}
  ]
}
)
```

```
> db.zamowienia.insert(
... {
...   klient: 'Jan Kowalski',
...   opis: 'Zamówienie złożone przez internet',
...   wartosc: '1024',
...   waluta: 'PLN',
...   data: new Date(2018,10,01,8,00),
...   pozycje:
...   [
...     {nazwa: 'toner', wartosc: '100'},
...     {nazwa: 'płyty DVD', wartosc: '56'}
...   ]
... }
... )
WriteResult({ "nInserted" : 1 })
>
```

```
> db.zamowienia.find()
{ "_id" : ObjectId("5ba0b4393fa6c76956ffbba8"), "klient" : "Jan Kowalski", "opis" : "Zamówienie złoż
one przez internet", "wartosc" : "1024", "waluta" : "PLN", "data" : ISODate("2018-11-01T07:00:00Z"),
 "pozycje" : [ { "nazwa" : "toner", "wartosc" : "100" }, { "nazwa" : "płyty DVD", "wartosc" : "56" }
 ] }
>
```

MongoDB – podstawy użytkowania

- Nadpisanie istniejącego dokumentu

Identyfikator istniejącego dokumentu.

```
use artur

db.zamowienia.save(
{
  _id: ObjectId('5ba0b4393fa6c76956ffbba8'),
  klient: 'Jan Kowalski',
  opis: 'Zamówienie złożone przez internet',
  wartosc: '1024',
  waluta: 'PLN',
  data: new Date(2018,10,01,8,
  data_wysylki: new Date(2018,
  pozycje:
  [
    {nazwa: 'toner', wartosc:
    {nazwa: 'płyty DVD', warto
  ]
}
)
```

```
> db.zamowienia.save(
... {
...   _id: ObjectId('5ba0b4393fa6c76956ffbba8'),
...   klient: 'Jan Kowalski',
...   opis: 'Zamówienie złożone przez internet',
...   wartosc: '1024',
...   waluta: 'PLN',
...   data: new Date(2018,10,01,8,00),
...   data_wysylki: new Date(2018,10,02,10,00),
...   pozycje:
...   [
...     {nazwa: 'toner', wartosc: '100'},
...     {nazwa: 'płyty DVD', wartosc: '56'}
...   ]
... }
... )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

MongoDB – podstawy użytkowania

- Tworzenie nowej kolekcji w bieżącej bazie danych
  `db.createCollection(<nazwa_kolekcji>,<opcje>)`
  - opcje:
    - capped:true – kolekcja o stałym rozmiarze nadpisująca swoje elementy po osiągnięciu zdefiniowanego rozmiaru (bufor cykliczny)
    - autoIndexID:true – automatycznie tworzy indeks na polu _id
    - size:<liczba> - maksymalny rozmiar kolekcji w bajtach
    - max:<liczba> - maksymalna liczba dokumentów
    - i sporo innych, szczegóły patrz dokumentacja
  - kolekcje nie muszą być tworzone jawnie – powstają automatycznie podczas próby wstawienia dokumentu
- Wyświetlanie listy kolekcji
  `show collections`
- Usuwanie kolekcji
  `db.<nazwa_kolekcji>.drop()`

MongoDB – podstawy użytkowania

- Wykonywanie zapytań
  - wyświetla wszystkie dokumenty z kolekcji w formie „surowej”
    `db.<nazwa_kolekcji>.find()`
  - wyświetla wszystkie dokumenty z kolekcji spełniające podany warunek selekcji
    `db.<nazwa_kolekcji>.find(<warunki_selekcji>)`
  - wyświetla wszystkie dokumenty z kolekcji w sposób sformatowany
    `db.<nazwa_kolekcji>.find().pretty()`
  - wyświetla pierwszy dokument z kolekcji
    `db.<nazwa_kolekcji>.findOne()`
  - wyświetla pierwszy dokument z kolekcji spełniający podany warunek selekcji
    `db.<nazwa_kolekcji>.findOne(<warunki_selekcji>)`
  - itd.

MongoDB – podstawy użytkowania

- Wykonywanie zapytań

```
> db.zamowienia.find()
{ "_id" : ObjectId("5ba0b4393fa6c76956ffbba8"), "klient" : "Jan Kowalski", "opis" : "Zamówienie złoż
one przez internet", "wartosc" : "1024", "waluta" : "PLN", "data" : ISODate("2018-11-01T07:00:00Z"),
 "data_wysylki" : ISODate("2018-11-02T09:00:00Z"), "pozycje" : [ { "nazwa" : "toner", "wartosc" : "1
00" }, { "nazwa" : "płyty DVD", "wartosc" : "56" } ] }
>
> db.zamowienia.find().pretty()
{
        "_id" : ObjectId("5ba0b4393fa6c76956ffbba8"),
        "klient" : "Jan Kowalski",
        "opis" : "Zamówienie złożone przez internet",
        "wartosc" : "1024",
        "waluta" : "PLN",
        "data" : ISODate("2018-11-01T07:00:00Z"),
        "data_wysylki" : ISODate("2018-11-02T09:00:00Z"),
        "pozycje" : [
                {
                        "nazwa" : "toner",
                        "wartosc" : "100"
                },
                {
                        "nazwa" : "płyty DVD",
                        "wartosc" : "56"
                }
        ]
}
>
```

MongoDB – podstawy użytkowania

- Wykonywanie zapytań
  - dokumenty, w których pole=wartość
    ```
    db.zamowienia.find({wartosc:1000})
    ```
  - dokumenty, w których pole<wartość
    ```
    db.zamowienia.find({wartosc:{$lt:1000}})
    ```
  - dokumenty, w których pole>wartość
    ```
    db.zamowienia.find({wartosc:{$gt:1000}})
    ```
  - dokumenty, w których pole<=wartość
    ```
    db.zamowienia.find({wartosc:{$lte:1000}})
    ```
  - dokumenty, w których pole>=wartość
    ```
    db.zamowienia.find({wartosc:{$gte:1000}})
    ```
  - dokumenty, w których pole!=wartość
    ```
    db.zamowienia.find({wartosc:{$ne:1000}})
    ```

MongoDB – podstawy użytkowania

- Wykonywanie zapytań
  - wyświetlanie tylko wybranych atrybutów
    `db.<nazwa_kolekcji>.find({},{atrybut:1,atrybut:1…})`
  - wyświetla tylko te atrybuty dokumentów, dla których podano „1”
  - atrybut „_id” jest wyświetlany domyślnie (aby go ukryć: „_id:0”)

```
> db.zamowienia.find(
...   {},
...   {klient:1,wartosc:1,_id:0}
... )
{ "klient" : "Jan Kowalski", "wartosc" : "1024" }
>
>
> db.zamowienia.find(   {},   {klient:1,wartosc:1,_id:1} )
{ "_id" : ObjectId("5ba0b4393fa6c76956ffbba8"), "klient" : "Jan Kowalski", "wartosc" : "1024" }
>
```

MongoDB – podstawy użytkowania

- Inne typowe i bardzo często wykonywane operacje
  - warunki logiczne AND, OR, AND+OR
  - ograniczanie liczby wyświetlanych dokumentów
    - wyświetl pierwszych N, pomiń pierwszych N, pomiń pierwszych N i potem wyświetl M itp.
  - sortowanie wyników zapytania
    - wg podanego atrybutu lub atrybutów
    - malejąco, rosnąco
  - wyświetlenie agregacji (podsumowań) danych
    - MIN, MAX, AVG, SUM, FIRST, LAST
  - modyfikowanie oraz usuwanie dokumentów
    - wszystkie
    - z możliwością podania warunku/warunków selekcji

MongoDB – podstawy użytkowania

- Wstawianie masowe

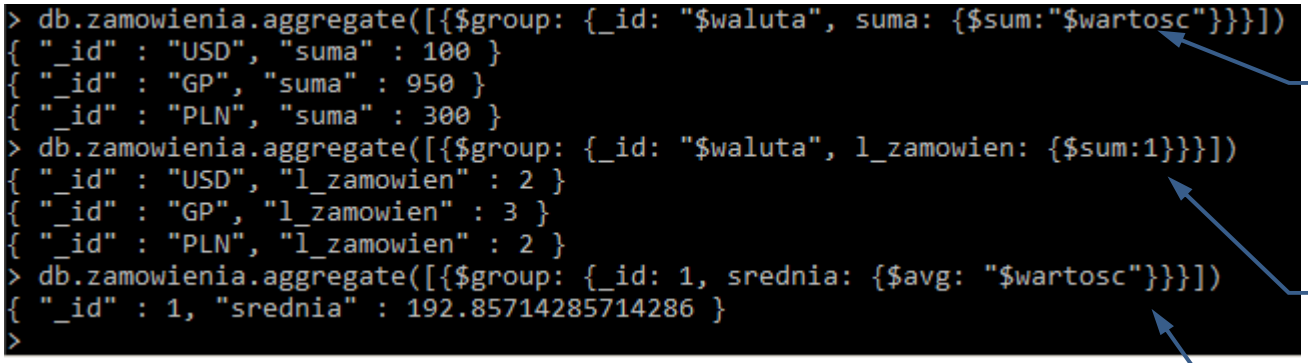
```
var bulk = db.zamowienia.initializeUnorderedBulkOp();
bulk.insert( { klient: "ABC", wartosc: 100, status: "A", waluta: 'PLN'} );
bulk.insert( { klient: "DEF", wartosc: 200, status: "A", waluta: 'PLN' } );
bulk.insert( { klient: "GHI", wartosc: 300, status: "P", waluta: 'GP' } );
bulk.insert( { klient: "JKL", wartosc: 150, status: "A", waluta: 'GP' } );
bulk.insert( { klient: "MNO", wartosc: 500, status: "A", waluta: 'GP' } );
bulk.insert( { klient: "PQR", wartosc:  50, status: "P", waluta: 'USD' } );
bulk.insert( { klient: "STU", wartosc:  50, status: "P", waluta: 'USD' } );
bulk.execute();
```

```
> var bulk = db.zamowienia.initializeUnorderedBulkOp();
> bulk.insert( { klient: "ABC", wartosc: 100, status: "A", waluta: 'PLN'} );
> bulk.insert( { klient: "DEF", wartosc: 200, status: "A", waluta: 'PLN' } );
> bulk.insert( { klient: "GHI", wartosc: 300, status: "P", waluta: 'GP' } );
> bulk.insert( { klient: "JKL", wartosc: 150, status: "A", waluta: 'GP' } );
> bulk.insert( { klient: "MNO", wartosc: 500, status: "A", waluta: 'GP' } );
> bulk.insert( { klient: "PQR", wartosc:  50, status: "P", waluta: 'USD' } );
> bulk.insert( { klient: "STU", wartosc:  50, status: "P", waluta: 'USD' } );
> bulk.execute();
BulkWriteResult({
        "writeErrors" : [ ],
        "writeConcernErrors" : [ ],
        "nInserted" : 7,
        "nUpserted" : 0,
        "nMatched" : 0,
        "nModified" : 0,
        "nRemoved" : 0,
        "upserted" : [ ]
})
```

MongoDB – podstawy użytkowania

- Przykłady agregacji

```
db.zamowienia.aggregate([{$group: {_id: "$waluta", suma: {$sum:"$wartosc"}}}])
db.zamowienia.aggregate([{$group: {_id: "$waluta", l_zamowien: {$sum:1}}}])
db.zamowienia.aggregate([{$group: {_id: 1, srednia: {$avg: "$wartosc"}}}])
```



Suma zamówień z podziałem na waluty.

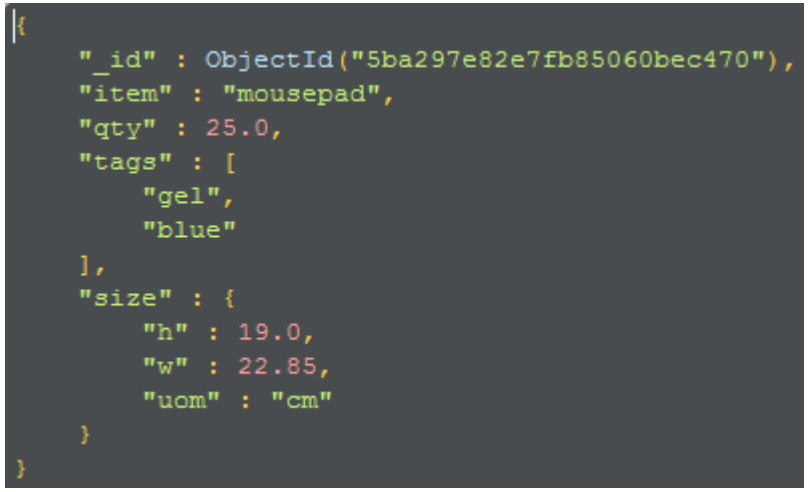
Suma jedynek (liczba dokumentów) z podziałem na waluty.

Średnia wartość wszystkich zamówień.

MongoDB – operacje CRUD

- CRUD – **C**reate, **R**ead, **U**pdate, **D**elete
- Tworzenie dokumentów
  - stosowane nazwy: create, insert, write
  - wszystkie operacje zapisu są atomowe na poziomie pojedynczego dokumentu

```
db.inventory.insertOne( {
      item: "canvas",
      qty: 100, tags: ["cotton"],
      size: {
        h: 28,
        w: 35.5,
        uom: "cm"
      }
  })
```




```
db.inventory.insertMany([
   { item: "journal", qty: 25, tags: ["blank", "red"], size: { h: 14, w: 21, uom: "cm" } },
   { item: "mat", qty: 85, tags: ["gray"], size: { h: 27.9, w: 35.5, uom: "cm" } },
   { item: "mousepad", qty: 25, tags: ["gel", "blue"], size: { h: 19, w: 22.85, uom: "cm" } }
])
```

MongoDB – operacje CRUD

- Update

  `db.collection.updateOne()`
  `db.collection.updateMany()`
  `db.collection.replaceOne()`

- Delete

  `db.collection.deleteOne()`
  `db.collection.deleteMany()`

```
db.restaurant.insertMany([
{ "_id": 1, "name": "Central Perk Cafe", "Borough": "Manhattan" },
{ "_id": 2, "name": "Rock A Feller Bar and Grill", "Borough": "Queens", "violations": 2 },
{ "_id": 3, "name": "Empire State Pub", "Borough": "Brooklyn", "violations": 0 }
])

db.restaurant.updateOne(
  { "name": "Central Perk Cafe" },
  { $set: { "violations" : 3 } }
)
```

```
db.restaurant.updateOne(
  { "name" : "Pizza Rat's Pizzaria" },
  { $set: {"_id" : 4, "violations" : 7, "borough" : "Manhattan" } },
  { upsert: true }
)
```

Wykona się insert, gdy takiego dokumentu nie ma w kolekcji.

MongoDB – operacje CRUD

- Read

```
db.inventory.insertMany([
   { item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status: "A" },
   { item: "notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" }, status: "A" },
   { item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "in" }, status: "D" },
   { item: "planner", qty: 75, size: { h: 22.85, w: 30, uom: "cm" }, status: "D" },
   { item: "postcard", qty: 45, size: { h: 10, w: 15.25, uom: "cm" }, status: "A" }
])
```

```
db.inventory.find( {} )
db.inventory.find( { status: "D" } )
db.inventory.find( { status: { $in: [ "A", "D" ] } } )
db.inventory.find( { status: "A", qty: { $lt: 30 } } )
db.inventory.find( { $or: [ { status: "A" }, { qty: { $lt: 30 } } ] } )
db.inventory.find( { status: "A", $or: [ { qty: { $lt: 30 } }, { item: /^p/ } ] } )
```

Wyrażenie regularne.

```
SELECT * FROM inventory
SELECT * FROM inventory WHERE status = "D"
SELECT * FROM inventory WHERE status in ("A", "D")
SELECT * FROM inventory WHERE status = "A" AND qty < 30
SELECT * FROM inventory WHERE status = "A" OR qty < 30
SELECT * FROM inventory WHERE status = "A" AND ( qty < 30 OR item LIKE "p%")
```

Funkcjonalne odpowiedniki w klasycznym SQL-u.

MongoDB – indeksy

- Unikamy konieczności pełnego odczytywania kolekcji w poszukiwaniu interesujących nas danych
  - bez indeksu trzeba sprawdzić każdy dokument, żeby określić czy spełnia on zapytanie czy nie
- Definiowane na poziomie kolekcji
- Działa bardzo podobnie, jak w klasycznych bazach relacyjnych
- Z punktu widzenia użytkownika są całkowicie "przeźroczyste"
- Mają strukturę B-drzew
- Wady
  - zajmują dodatkową przestrzeń na dysku
  - spowalniają operacje wstawiania i modyfikowania danych (konieczność przebudowy indeksu)

MongoDB – indeksy

- Typy indeksów
  - na polu _id, zakładany domyślnie, jest dostępny we wszystkich kolekcjach. Ten indeks jest unikalny (nie można dodać dwóch dokumentów o tym samym _id)
  - pojedynczy (na jednym polu, może być w kierunku rosnącym lub malejącym, np. od A do Z, od Z do A)
  - złożony (*compound*, zakładany na wielu polach, też może być rosnący lub malejący)
  - wielokluczowy (*multikey*), zakładany na polach typu tablicowego
  - Indeksy geoprzestrzenne
    - spcjalny rodzaj indeksu do operowania na danych przestrzennych
  - Indeksy TTL
    - indeksy tworzone na pojedynczym polu typu „Date”, powodujące automatyczne usuwanie dokumentów, dla których upłynął podany czas od zapisanej daty , np.
      `db.zamowienia.createIndex({"data":1},{expireAfterSeconds:36000})`

MongoDB – indeksy

- Składnia

  `db.<nazwa_kolekcji>.createIndex({atrybut:1, atrybut:-1,…}, {opcje})`

  (rosnąco: +1, malejąco: -1)

  opcje (przykładowe, więcej szczegółów w dokumentacji):

  background:false – tworzy indeks w tle, nie blokując sesji

  unique:true – tworzy indeks unikatowy

  name:<nazwa> - pozwala nazwać indeks

- Przykład

  `db.zamowienia.createIndex({klient:1})`

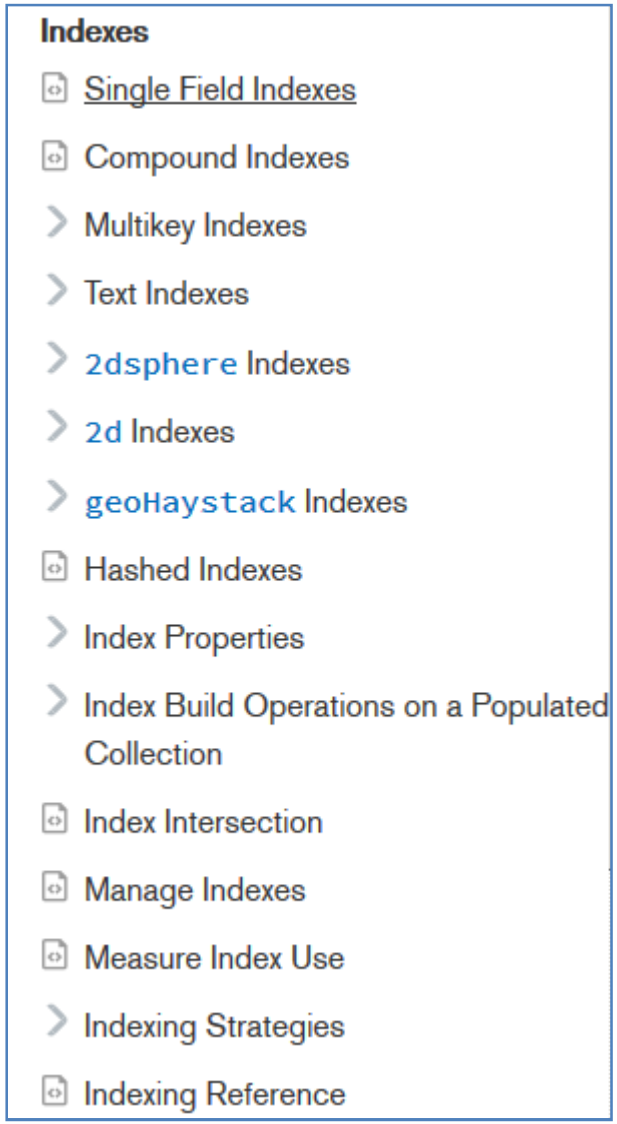
MongoDB – indeksy

- Indeksy tekstowe
  - MongoDB udostępnia indeksy tekstowe wspomagające przeszukiwanie danych tekstowych
  - mogą być tworzone dla pól zawierających dane tekstowe lub dla tablic z polami tekstowymi
  - słowa ze stoplisty nie są indeksowane (niestety na razie język polski nie jest wspierany)
  - przechowują tylko rdzenie zlaczeniowe słów (niestety na razie język polski nie jest wspierany)
  - ograniczenie: kolekcja może mieć założony tylko jeden indeks tekstowy
  - przykład

```
db.zamowienia.createIndex({klient:"text"})
```

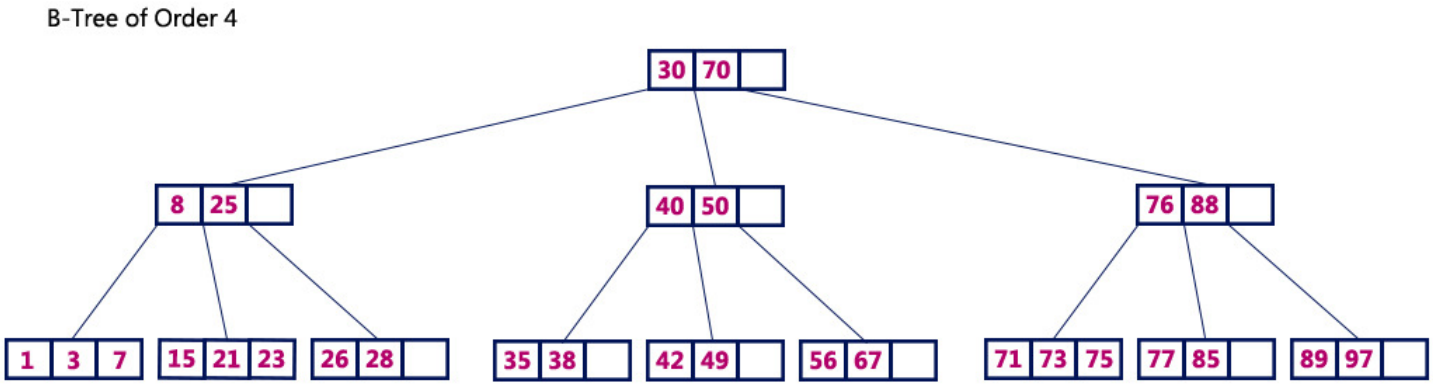
MongoDB – indeksy

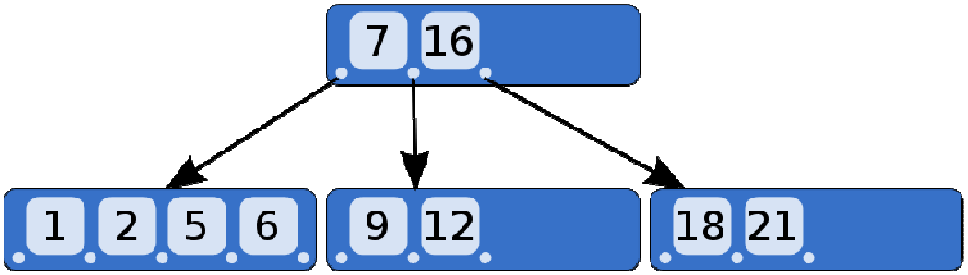
- Dokumentacja



MongoDB – indeksy

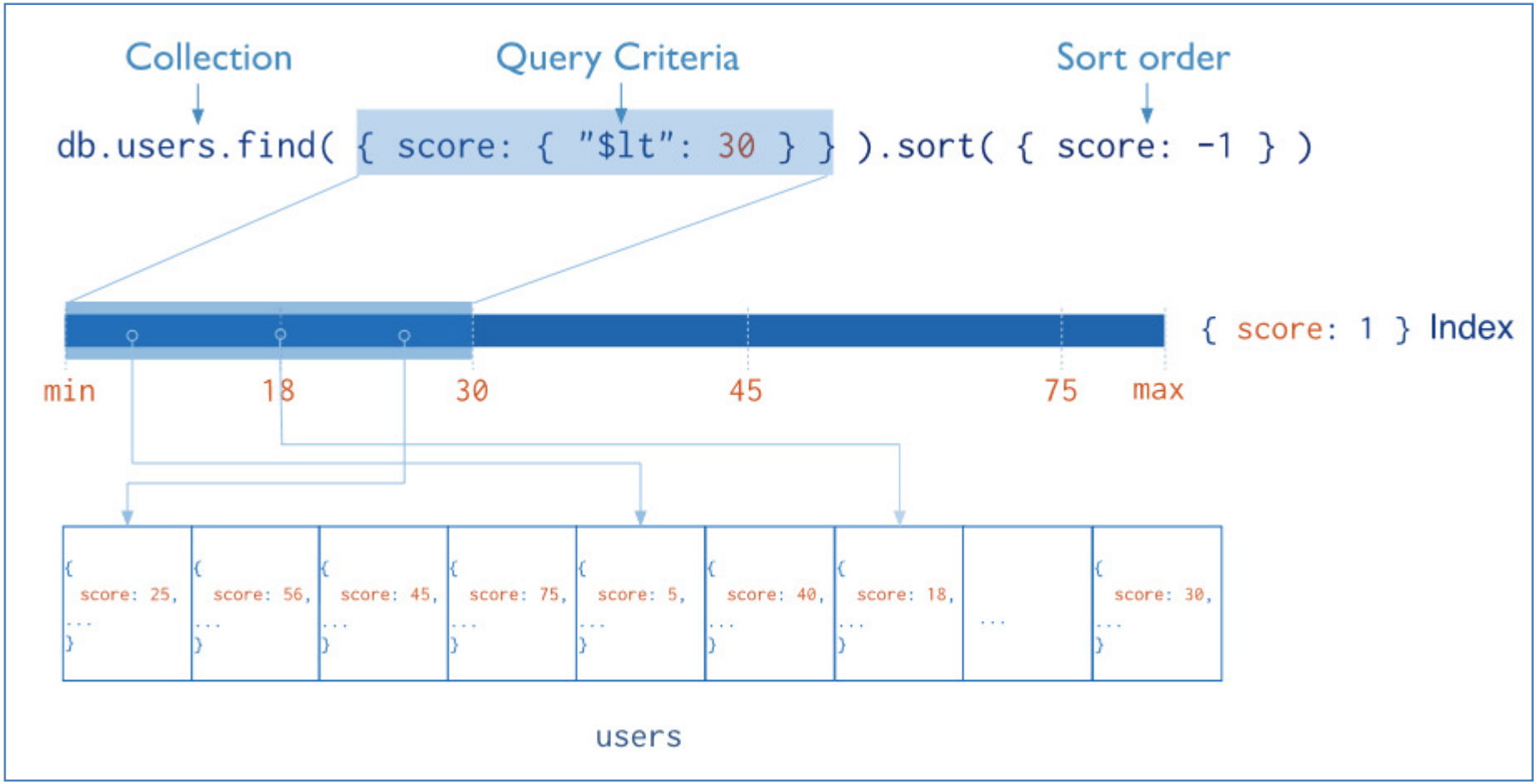
- B-drzewo (B-Tree)
  - drzewiasta struktura danych, przechowująca klucze w pewnym porządku i powiązane z nimi dane
  - szczególnym przypadkiem B-drzew są B+drzewa, które przechowującym dane tylko w liściach





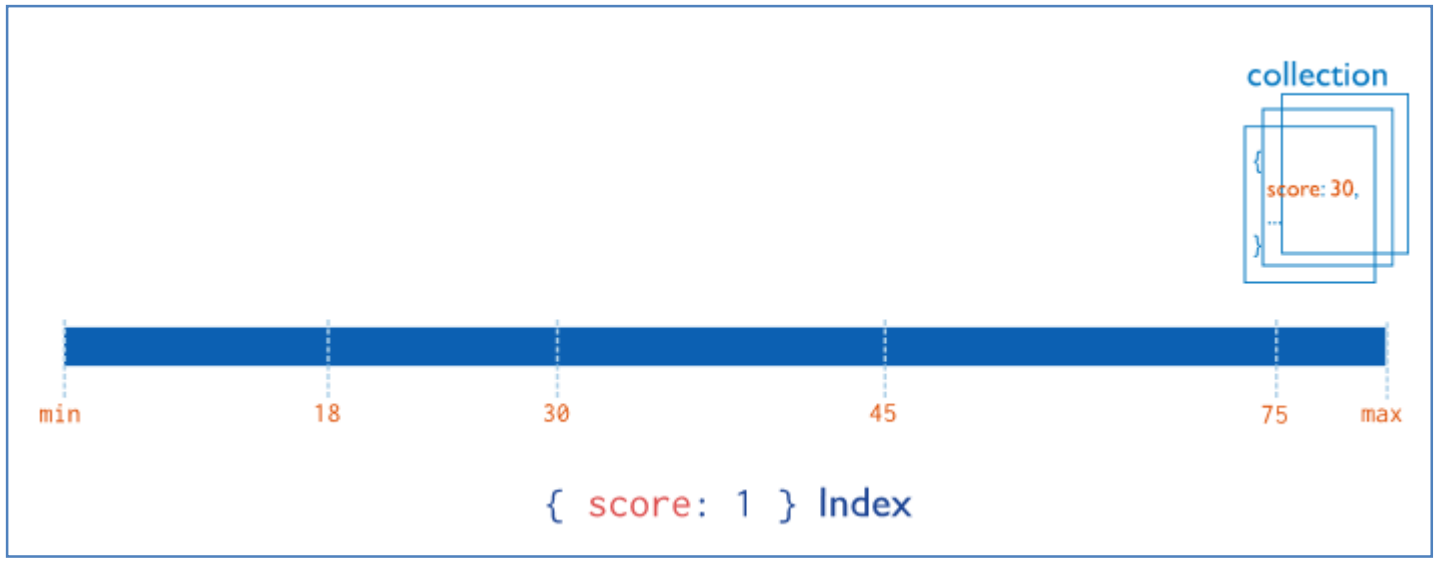
MongoDB – indeksy

- Istota indeksu graficznie



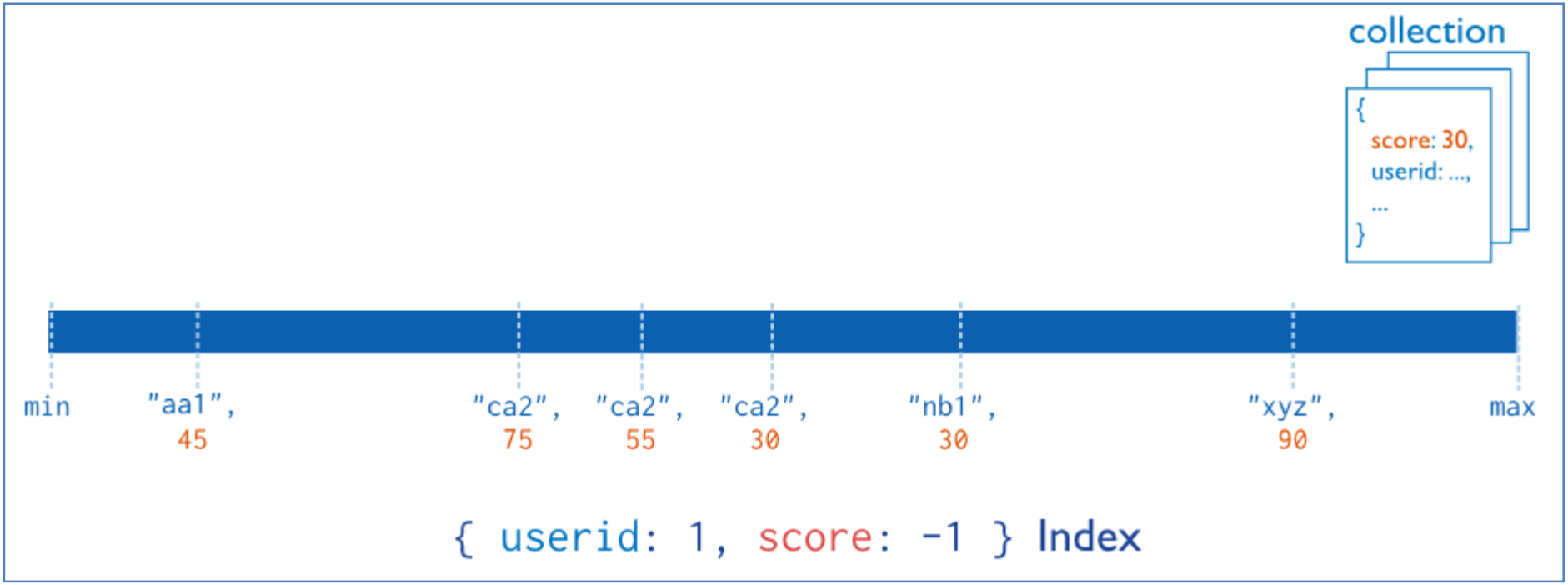
MongoDB – indeksy

- Indeks pojedynczy (Single field index)



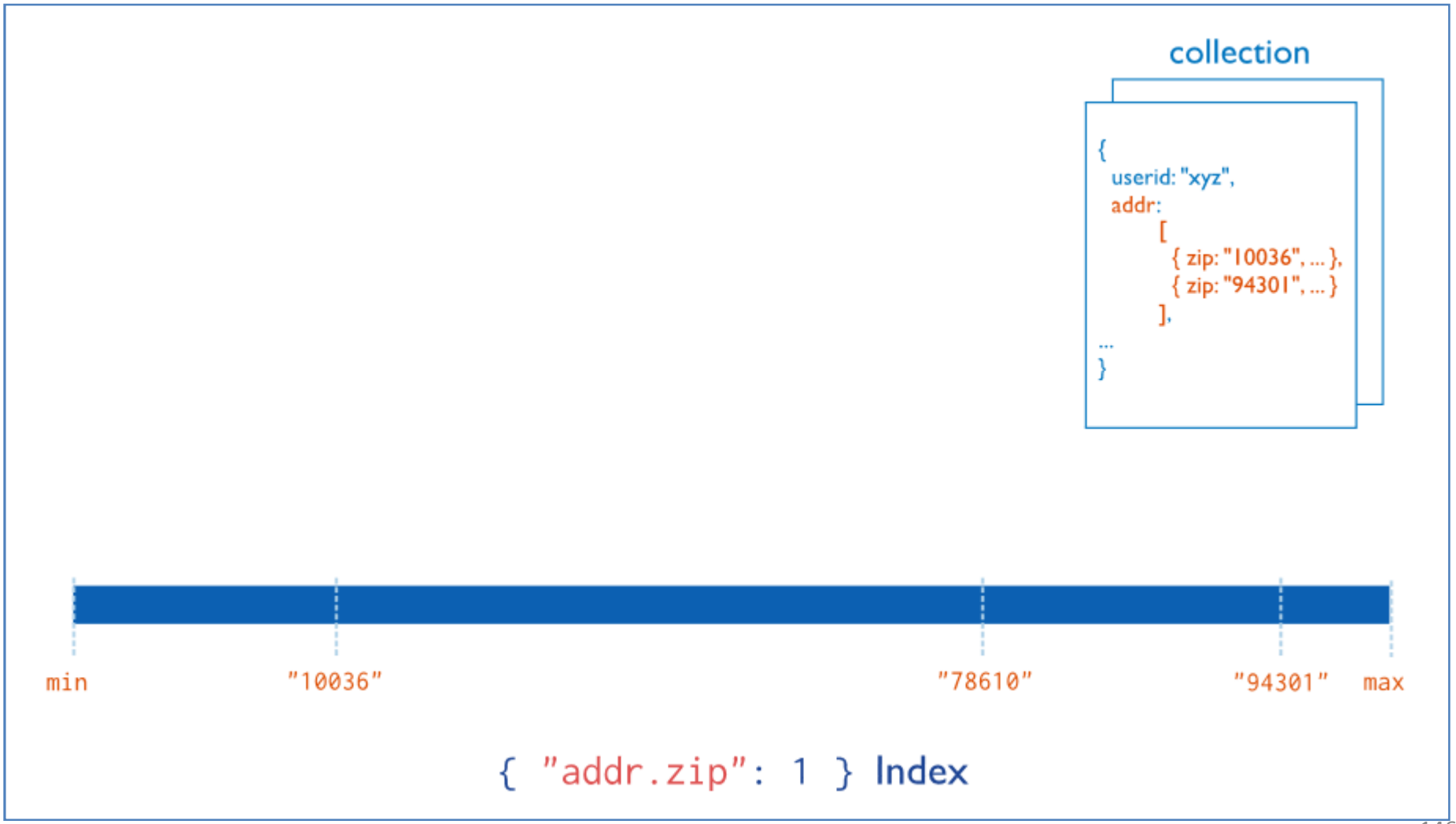
MongoDB – indeksy

- Indeks złożony (Compound index)



MongoDB – indeksy

- Indeks wielokluczowy (Multikey index)



MongoDB – replikacja

- Cel replikacji: poprawa niezawodności
- Zbiór replik (Replica Set) to zestaw serwerów MongoDB (instancji/procesów mongod) obsługujących ten sam zbiór danych (zwykle tych serwerów powinno być minimum 3)
- Jeden węzeł pełni zawsze rolę podstawowego (Primary Node), pozostałe węzły są wówczas węzłami wtórnymi (Secondary Nodes)
- Dane zawsze replikowane są od węzła podstawowego do węzłów wtórnych
- Po awarii serwera podstawowego trwającej ponad 10s, jeden z serwerów wtórnych zmienia rolę na podstawową. Sprawdzenie roli serwera: `db.isMaster()`
- Po usunięciu awarii serwera podstawowego pozostaje on nadal serwerem wtórnym

MongoDB – replikacja

- Konfiguracja
  - krok 1: zamknięcie aktualnie działającego serwera
  - krok 2: uruchomienie serwera z opcją `--replSet`

    ```
    mongod --port "PORT" --dbpath "YOUR_DB_DATA_PATH"
           --replSet "REPLICA_SET_INSTANCE_NAME"
    ```

    Przykład:

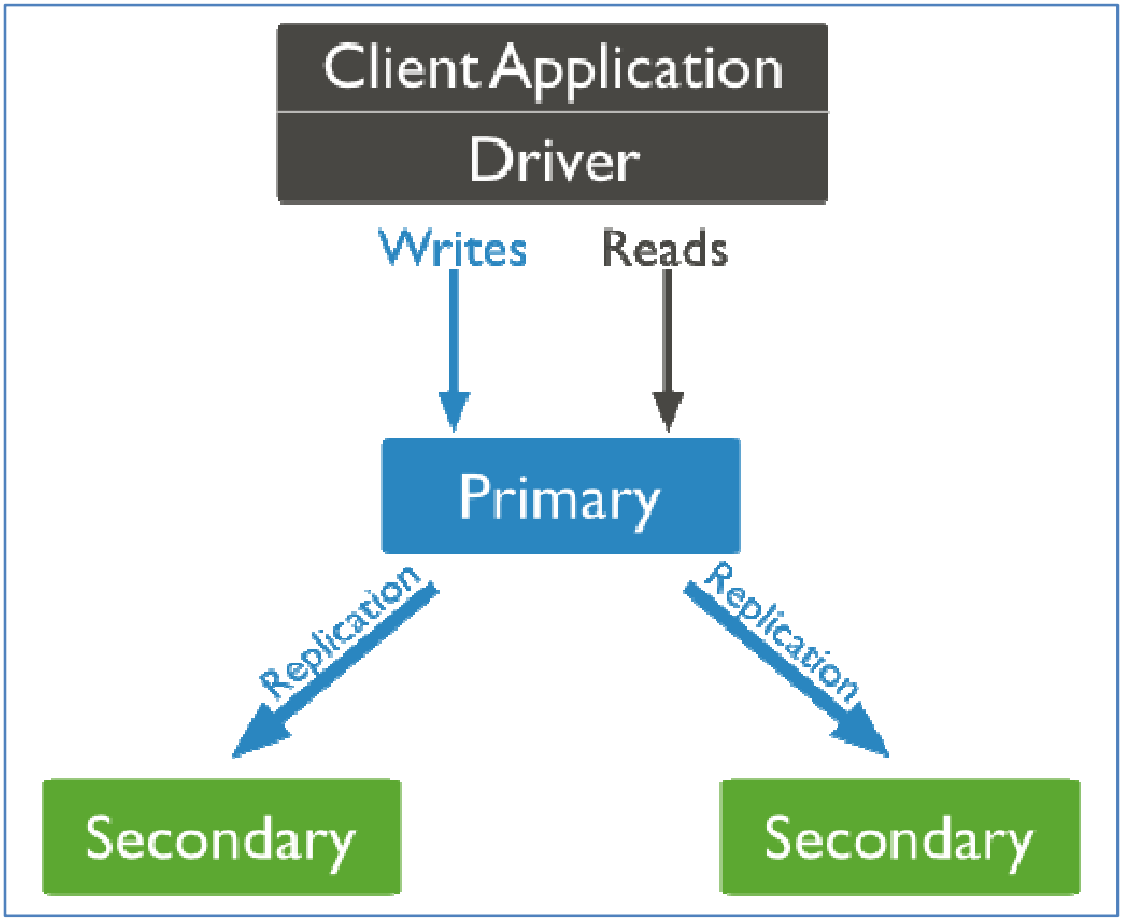
    ```
    mongod --port 27017 --dbpath "C:\data\db" --replSet rs0
    ```

    Powyższe spowoduje uruchomienie serwera mongodb i nadanie mu nazwy rs0
  - krok 3: z linii poleceń łączymy się do tej instancji
  - krok 4: wykonujemy komendę `rs.initiate()`, aby zainicjować nową replikę

MongoDB – replikacja

- Konfiguracja, c.d.
  - krok 5: uruchamiamy kolejne instancje MongoDB (procesy mongod) , które za chwilę będziemy chcieli uczynić węzłami wtórnymi, np.:
    ```
    mongod --port 27018 --dbpath "C:\data\db2" --replSet rs0
    mongod --port 27019 --dbpath "C:\data\db3" --replSet rs0
    ```
  - krok 6: dodajemy do repliki kolejne węzły za pomocą polecenia `rs.add()`, np: `rs.add("mongod1.net:27018")`
  - krok 7: aby sprawdzić konfigurację repliki wydajemy komendę `rs.conf()`. Aby sprawdzić aktualny status repliki wydajemy polecenie `rs.status()`
- Uwaga: kolejne węzły do stworzonej repliki można dodawać tylko z poziomu węzła podstawowego (trzeba być tam podłączonym). Poleceniem `db.isMaster()` sprawdzamy rodzaj węzła

MongoDB – replikacja

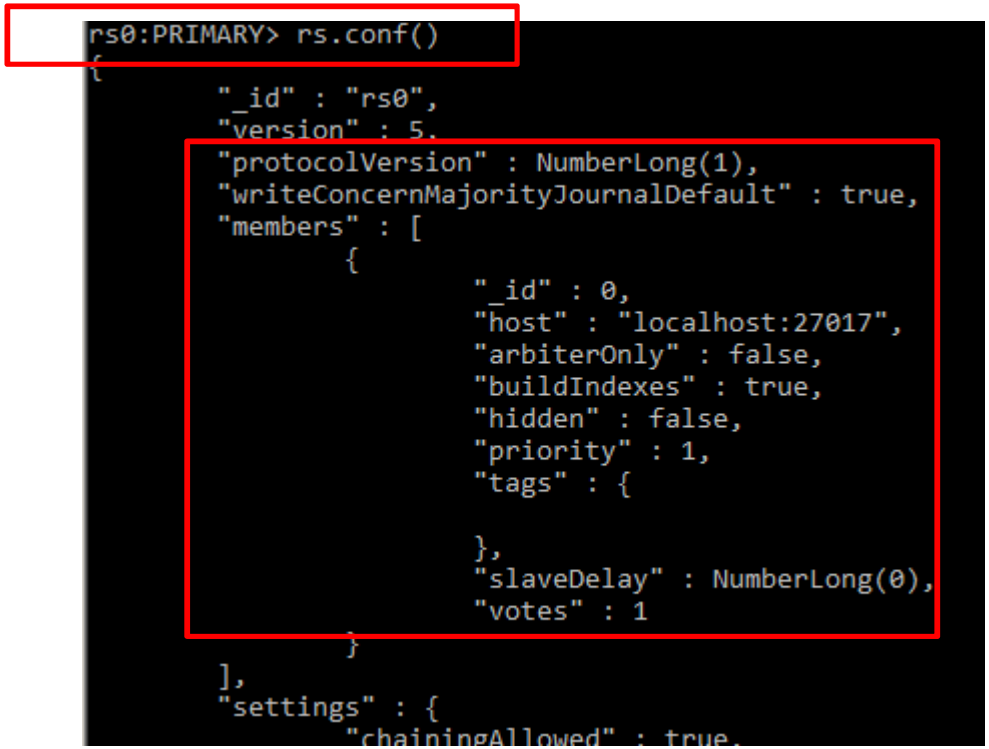


MongoDB – replikacja

- Demonstracja działania
  - uruchamiamy pierwszy serwer

```
mongod --port 27017 --dbpath "C:\data\db1" --replSet rs0
```

  - sprawdzamy aktualną konfigurację



MongoDB – replikacja

- Demonstracja działania
  - uruchamiamy dwa kolejne serwer i dodajemy je do repliki

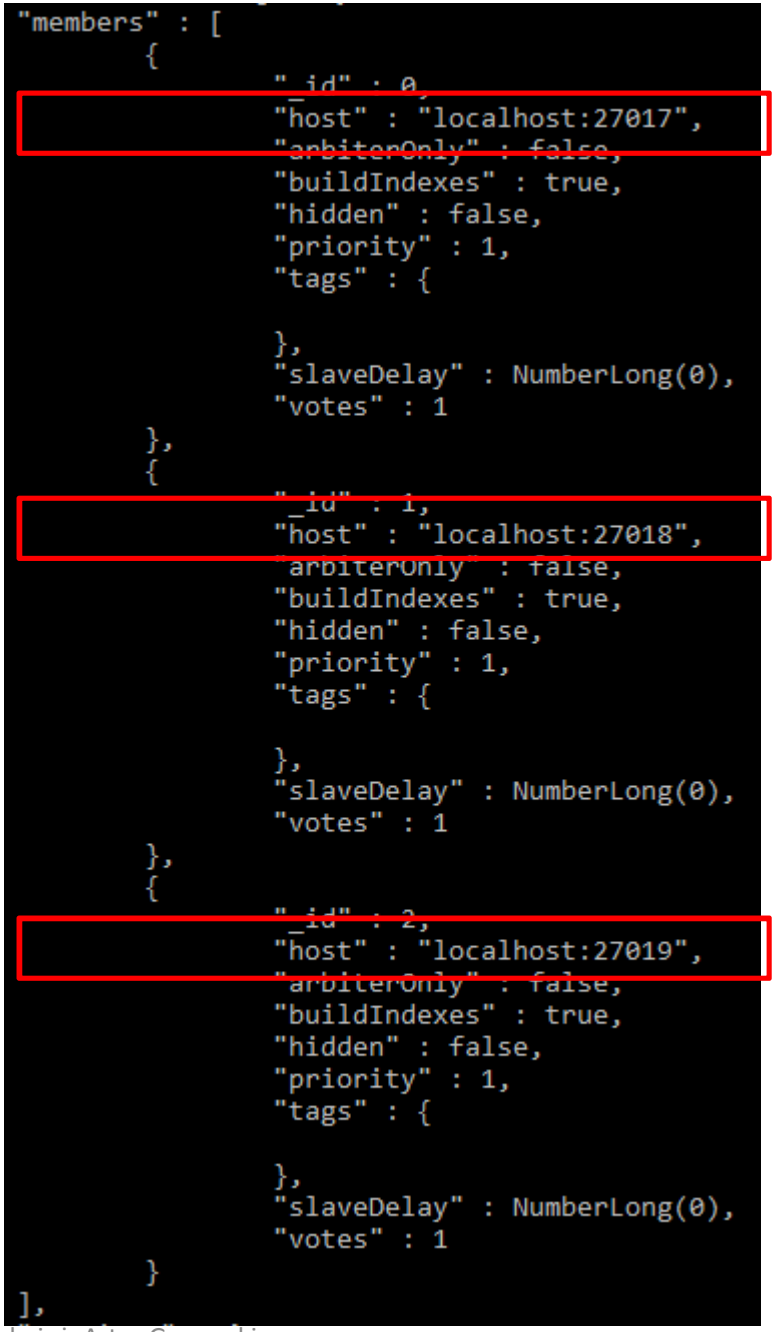
```
mongod --port 27018 --dbpath "C:\data\db2" --replSet rs0
mongod --port 27019 --dbpath "C:\data\db3" --replSet rs0
rs.add("localhost:27018")
rs.add("localhost:27019")
```

```
rs0:PRIMARY> rs.add("localhost:27018")
{
        "ok" : 1,
        "operationTime" : Timestamp(1537356581, 1),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1537356581, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
rs0:PRIMARY> rs.add("localhost:27019")
{
        "ok" : 1,
        "operationTime" : Timestamp(1537356584, 1),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1537356584, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
rs0:PRIMARY>
```

MongoDB – replikacja

- Demonstracja działania
  - sprawdzamy aktualną konfigurację klastra

```
rs.conf()
```



  - potwierdzamy, że dwa dodatkowe serwery pracują jako węzły wtórne

```
mongo --port 27018
rs0:SECONDARY>
```

```
mongo --port 27019
rs0:SECONDARY>
```

MongoDB – replikacja

- Demonstracja działania
  - zamykamy serwer podstawowy (działający na porcie 27017), w ten sposób symulujemy jego awarię

```
mongo --port 27017
```

```
c:\Program Files\MongoDB\Server\4.0\bin>mongo --port 27017
MongoDB shell version v4.0.2
connecting to: mongodb://127.0.0.1:27017/
2018-09-19T13:38:47.319+0200 E QUERY    [js] Error: couldn't connect to server 127.0.0.1:27017,
d: SocketException: Error connecting to 127.0.0.1:27017 :: caused by :: Nie mo□na nawi□za□ po□□□
 docelowy aktywnie go odmawia. :
connect@src/mongo/shell/mongo.js:257:13
@(connect):1:6
exception: connect failed
```

  - jeden z serwerów wtórnych zostaje promowany do funkcji serwera podstawowego

```
mongo --port 27018 rs0:PRIMARY>
```

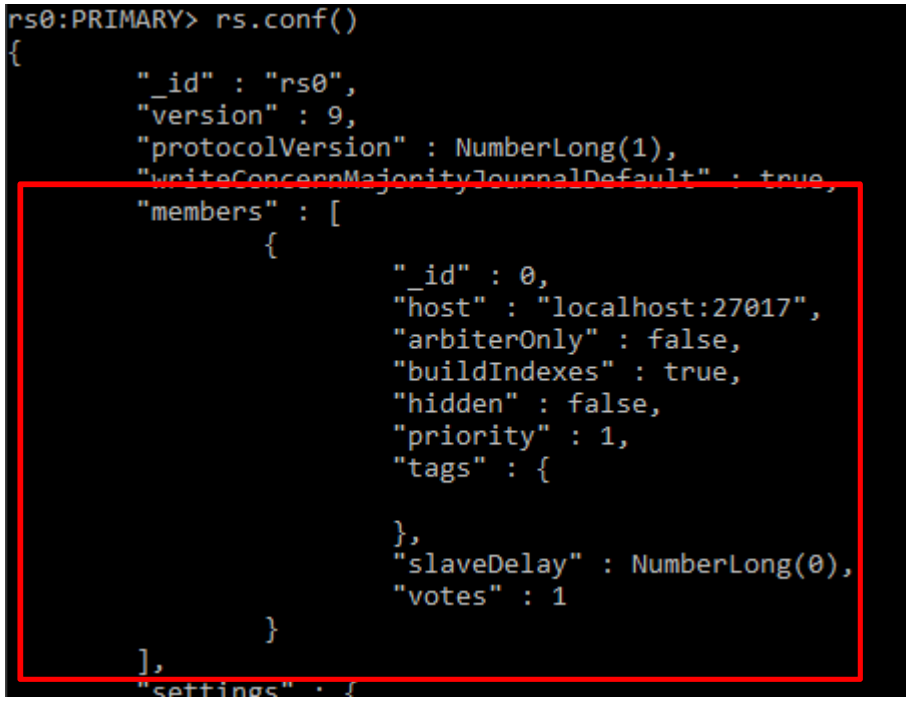
```
mongo --port 27019 rs0:SECONDARY>
```

MongoDB – replikacja

- Demonstracja działania
  - usuwamy dwa węzły z repliki

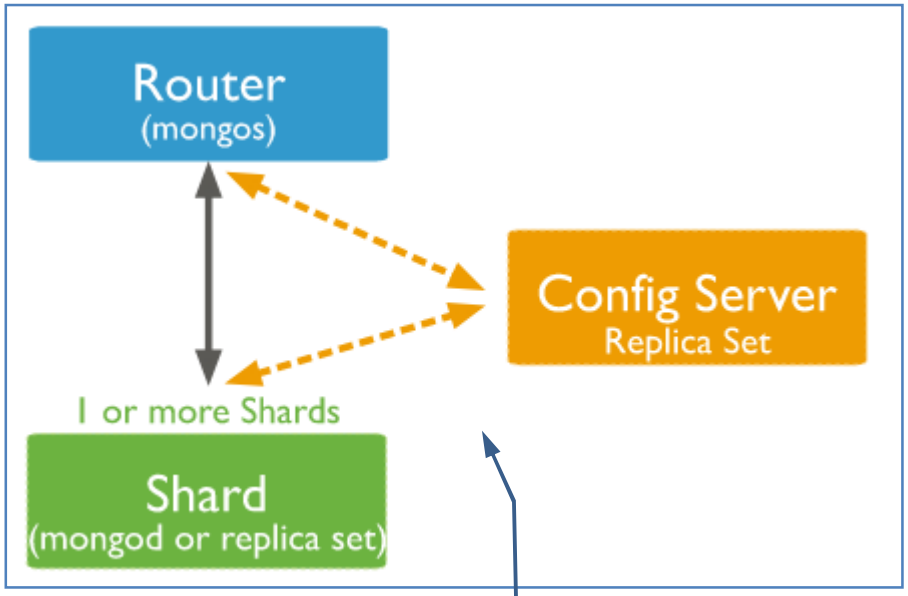
```
rs.remove("localhost:27018")
rs.remove("localhost:27019")
```

  - potwierdzamy, że nasza replika składa się teraz tylko z jednego węzła



MongoDB – sharding

- Sharding, rozdział danych
  - nie będziemy tłumaczyli tego słowa na polski (brak zgrabnego odpowiednika)
  - shard: odłamek, okruch, kawałek itp..
- Skalowanie poziome: rozdział dużych kolekcji danych pomiędzy wiele serwerów MongoDB
- Trzy role serwerów:
  - Shard – przechowuje fragment bazy danych (zwykle jako zbiór replik, aby zwiększyć bezpieczeństwo)
  - Router – przyjmuje zapytania użytkowników i kieruje je do serwerów Shard
  - Config Server – gromadzi metadane o odwzorowaniach danych na serwery Shard (zwykle jako zbiór replik, aby zwiększyć bezpieczeństwo)



W wersji minimalistycznej Config Server oraz Shard nie muszą koniecznie być konfigurowane jako repliki. Choc jest to zalecane.

https://www.bmc.com/blogs/mongodb-sharding-explained/

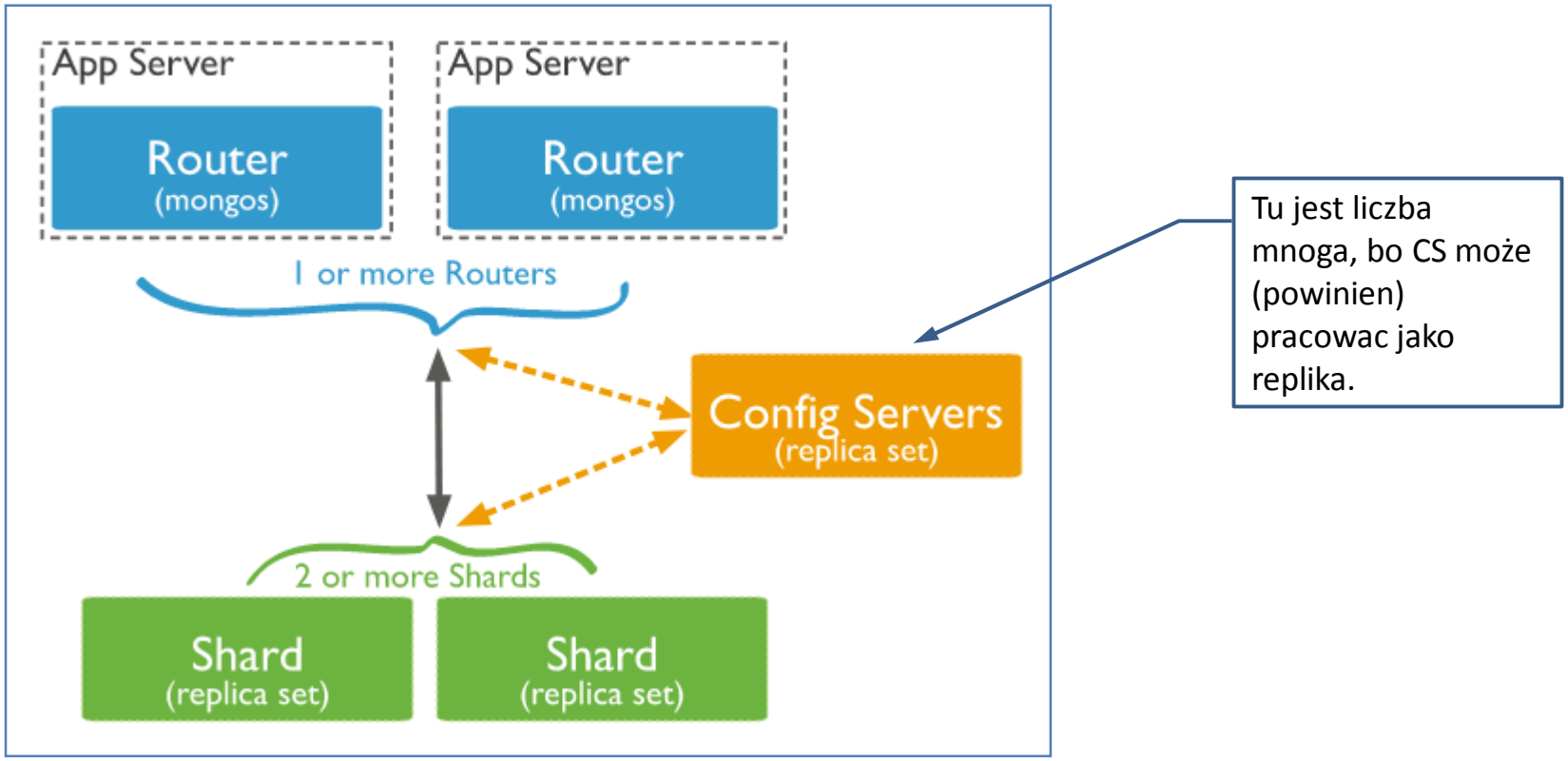
MongoDB – sharding

- Działanie środowiska, jak na rysunku powyżej
  1. Aplikacja użytkownika komunikuje się z router-em (mongos), gdy chce wykonać jakieś zapytanie do danych
  2. Router (mongos) komunikuje się z Configuration Server, aby sprawdzić w których shard-ie są poszukiwane dane i kieruje tam użytkownika
     - widać, jak ważna rolę pełni tutaj Configuration Server, tylko tam jest informacja, które dane są w którym shard-ie!
  3. Dane pobrane z shard-a są zwracane użytkownikowi

- Zwróćmy uwagę, że Configuration Server oraz shard-y pracują w konfiguracji replik
  - a przynajmniej powinny tak pracować w środowisku produkcyjnym

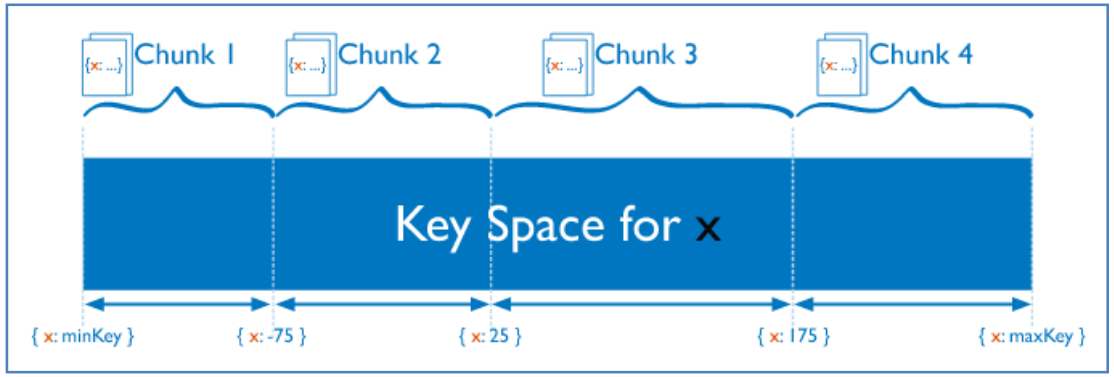
MongoDB – sharding

- Środowisko produkcyjne może wyglądać tak:
  - na poprzednim slajdzie pokazano środowisko minimalistyczne



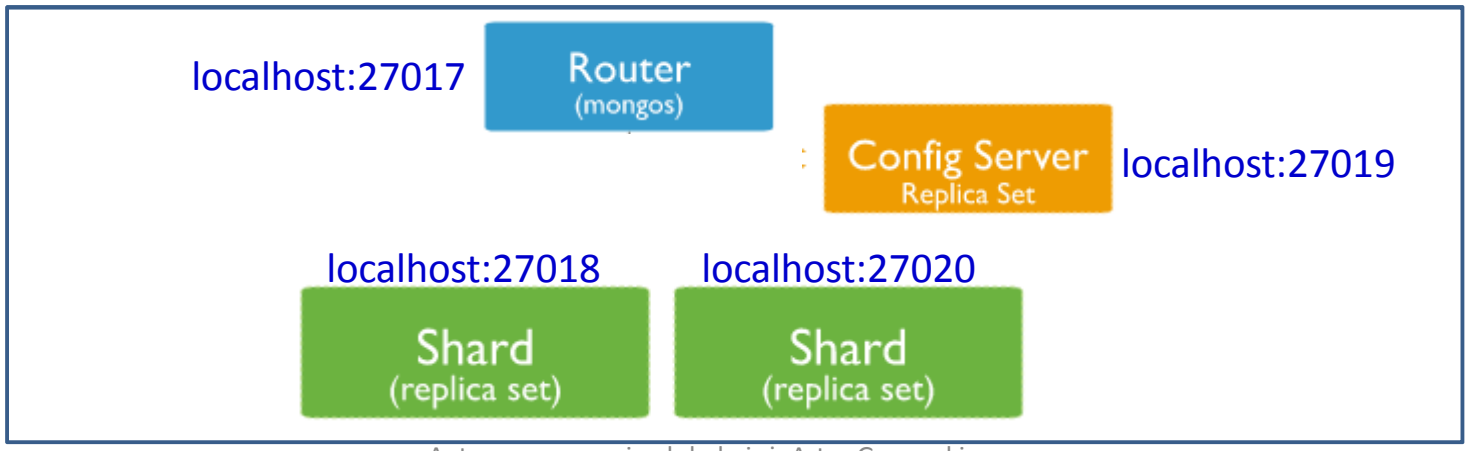
MongoDB – sharding

- Kolejne ważne pojęcie: Shard Keys
  - klucze używane do rozproszenia danych na poszczególne shard-y. Określa, do którego fizycznego shard-a trafią konkretne dane
  - ważne: po utworzeniu środowiska shard-owego dane nie mogą być już „od-shard-owane” (unshard)
- Kolejne ważne pojęcie: Chunks
  - chunk to dane z pewnego ciągłego zakresu Shard Keys. MongoDB może przesuwać chunks-y do innych shard-ów, aby balansować obciążenie



MongoDB – sharding

- Przykład uruchomienia środowiska shard-owego
  - https://docs.mongodb.com/manual/tutorial/deploy-shard-cluster/
  - https://www.bmc.com/blogs/mongodb-sharding-explained/
- Krok 1: tworzenie i konfiguracja Config Server (lub serwersów)
- Krok 2: tworzenie i konfiguracja Query Router (lub ruterów)
- Krok 3: tworzenie i konfiguracja Shard-a (lub Shard-ów)
- Krok 4: aktywacja środowiska shard-owego



MongoDB – sharding

- Krok 0: Utworzenie katalogów i plików konfiguracyjnych
  - w środowisku demonstracyjnym autora
  - oczywiście każdy może sobie to zrobić według własnych upodobań

```
c:\Programy\MongoDB>tree
├───Conf
├───ConfigServer
├───Data
│   ├───ConfigServer
│   ├───QueryRouter
│   ├───Shard1
│   └───Shard2
├───Logs
├───QueryRouter
├───Shard1
└───Shard2
```

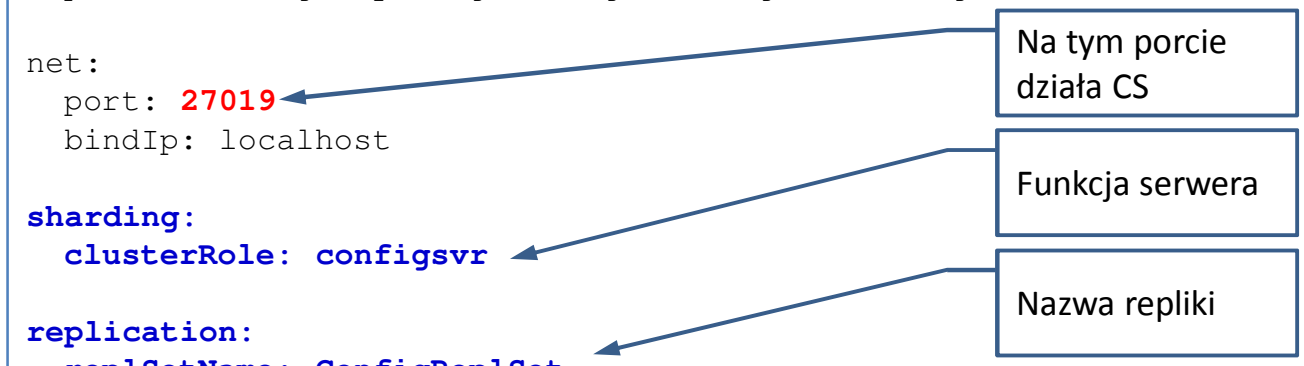
```
c:\Programy\MongoDB>dir Logs
2021-01-06  22:37    <DIR>          .
2021-01-06  22:37    <DIR>          ..
2021-01-06  22:37                 0 ConfigServer.log
2021-01-06  22:37                 0 QueryRouter.log
2021-01-06  22:37                 0 Shard1.log
2021-01-06  22:37                 0 Shard2.log
```

```
c:\Programy\MongoDB>dir Conf
2021-01-06  22:03    <DIR>          .
2021-01-06  22:03    <DIR>          ..
2021-01-06  00:46               396 ConfigServer.conf
2021-01-06  00:47               202 QueryRouter.conf
2021-01-06  01:53               381 Shard1.conf
2021-01-06  01:53               381 Shard2.conf
```

MongoDB – sharding

- Krok 1: tworzenie i konfiguracja Config Server-a
  - plik konfiguracyjny

```
# Uwaga: plik musi być w formacie YAML. Nie można
# używać tabulatorów, tylko spacje są dozwolone.
storage:
  dbPath: c:\Programy\MongoDB\Data\ConfigServer

systemLog:
  destination: file
  logAppend: true
  path: c:\Programy\MongoDB\Logs\ConfigServer.log

net:
  port: 27019
  bindIp: localhost

sharding:
  clusterRole: configsvr

replication:
  replSetName: ConfigReplSet
```



  - uruchomienie serwera mongod

```
mongod --config c:\Programy\MongoDB\Conf\ConfigServer.conf
```

MongoDB – sharding

- Krok 1: tworzenie i konfiguracja Config Server-a
  - podłączenie się klientem, inicjalizacja CS, sprawdzenie statusu

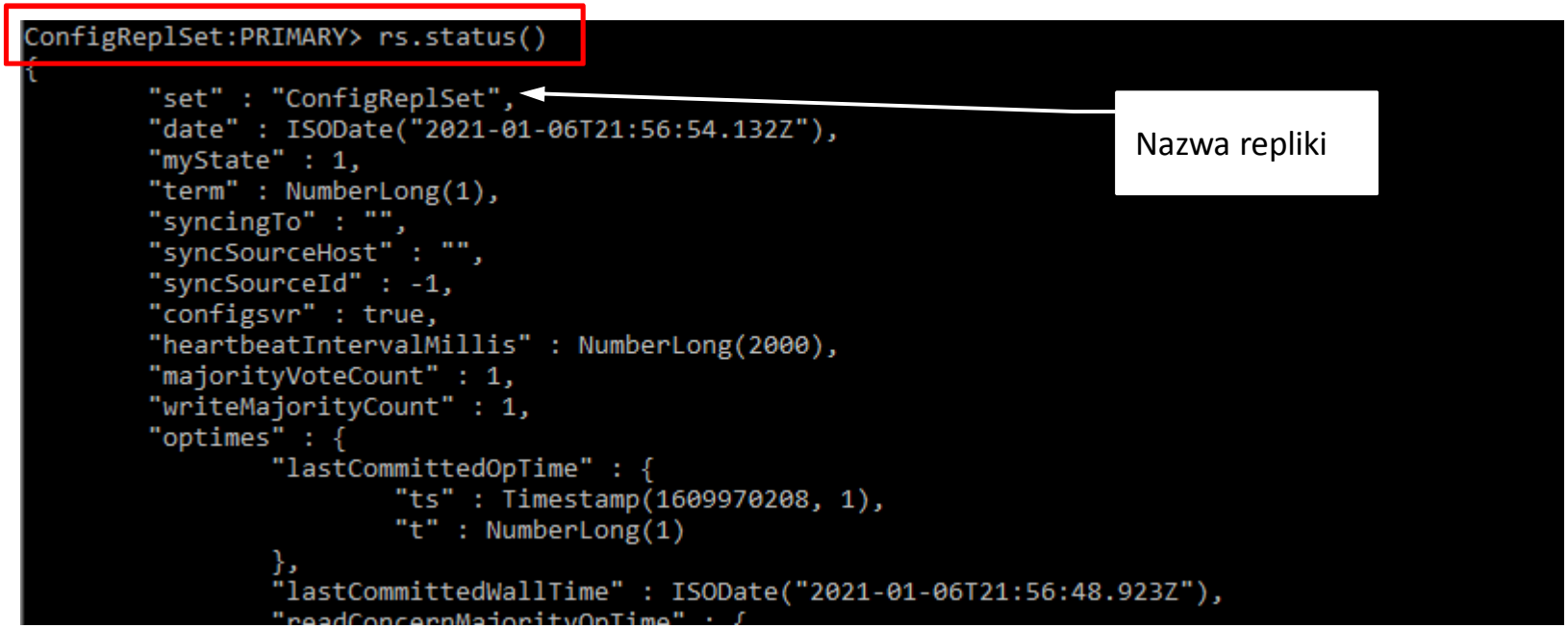
```
mongo localhost:27019
rs.initiate()
```

```
> rs.initiate()
{
        "info2" : "no configuration specified. Using a default configuration for the set",
        "me" : "localhost:27019",
        "ok" : 1,
        "$gleStats" : {
                "lastOpTime" : Timestamp(1609970131, 1),
                "electionId" : ObjectId("000000000000000000000000")
        },
        "lastCommittedOpTime" : Timestamp(0, 0),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1609970131, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        },
        "operationTime" : Timestamp(1609970131, 1)
}
ConfigReplSet:SECONDARY>
```

MongoDB – sharding

- Krok 1: tworzenie i konfiguracja Config Server-a
  - podłączenie się klientem, inicjalizacja CS, sprawdzenie statusu

```
rs.status()
```



MongoDB – sharding

- Krok 2: tworzenie i konfiguracja Query Router-a
  - plik konfiguracyjny

```
systemLog:
  destination: file
  logAppend: true
  path: c:\Programy\MongoDB\Logs\QueryRouter.log

net:
  port: 27017
  bindIp: localhost

sharding:
  configDB: ConfigReplSet/localhost:27019
```

Na tym porcie działa QR

Podajemy nazwę wcześniej zdefiniowanej repliki i listę hostów do niej należących (u nas jest tylko jeden host)

  - uruchomienie serwera mongod

```
mongod --config c:\Programy\MongoDB\Conf\ConfigServer.conf
```

MongoDB – sharding

- Krok 2: tworzenie i konfiguracja Query Router-a
  - uruchomienie QR za pomocą komendy mongos (literka s na końcu)

```
mongos --config c:\Programy\MongoDB\Conf\QueryRouter.conf
```

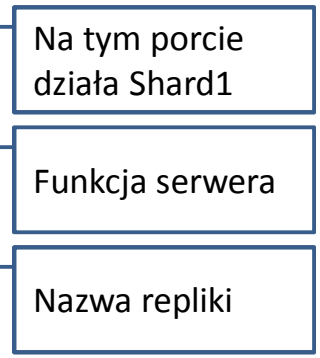
  - podłączenie się do QR

```
C:\Programy\MongoDB\Logs>c:\Programy\MongoDB\ConfigServer\bin\mongo localhost:27017
MongoDB shell version v4.2.9
connecting to: mongodb://localhost:27017/test?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("63740636-80b0-4aec-863c-7d02344f2c29") }
MongoDB server version: 4.2.9
Server has startup warnings:
2021-01-06T23:14:01.237+0100 I  CONTROL  [main]
2021-01-06T23:14:01.237+0100 I  CONTROL  [main] ** WARNING: Access control is not enabled for the data
2021-01-06T23:14:01.238+0100 I  CONTROL  [main] **          Read and write access to data and configur
2021-01-06T23:14:01.238+0100 I  CONTROL  [main]
mongos>
```

MongoDB – sharding

- Krok 3: tworzenie i konfiguracja Shard-a (lub Shard-ów)
  - plik konfiguracyjny, pierwszy shard

```
storage:
  dbPath: c:\Programy\MongoDB\Data\Shard1

systemLog:
  destination: file
  logAppend: true
  path: c:\Programy\MongoDB\Logs\Shard1.log

net:
  port: 27018
  bindIp: localhost

sharding:
  clusterRole: shardsvr

replication:
  replSetName: ShardReplSet
```



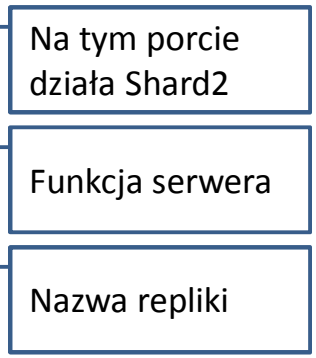
  - uruchomienie serwera mongod

```
mongod --config c:\Programy\MongoDB\Conf\Shard1.conf
```

MongoDB – sharding

- Krok 3: tworzenie i konfiguracja Shard-a (lub Shard-ów)
  - plik konfiguracyjny, drugi shard

```
storage:
  dbPath: c:\Programy\MongoDB\Data\Shard2

systemLog:
  destination: file
  logAppend: true
  path: c:\Programy\MongoDB\Logs\Shard2.log

net:
  port: 27020
  bindIp: localhost

sharding:
  clusterRole: shardsvr

replication:
  replSetName: ShardReplSet
```



  - uruchomienie serwera mongod

```
mongod --config c:\Programy\MongoDB\Conf\Shard2.conf
```

MongoDB – sharding

- Krok 3: tworzenie i konfiguracja Shard-a (lub Shard-ów)
  - inicjalizacja repliki ShardReplSet

```
mongo localhost 27018
rs.initiate()
```

```
> rs.initiate()
{
        "info2" : "no configuration specified. Using a default configuration for the set",
        "me" : "localhost:27018",
        "ok" : 1,
        "$clusterTime" : {
                "clusterTime" : Timestamp(1609972423, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        },
        "operationTime" : Timestamp(1609972423, 1)
}
```

```
rs.add("localhost:27020")
```

```
ShardReplSet:PRIMARY> rs.add("localhost:27020")
{
        "ok" : 1,
        "$clusterTime" : {
                "clusterTime" : Timestamp(1609972474, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        },
```

MongoDB – sharding

- Krok 3: tworzenie i konfiguracja Shard-a (lub Shard-ów)
  - sprawdzenie statusu repliki ShardReplSet

```
rs.status()
```

```
ShardReplSet:PRIMARY> rs.status()
{
        "set" : "ShardReplSet",
        "date" : ISODate("2021-01-06T22:34:43.333Z"),
        "myState" : 1,
        "term" : NumberLong(1),
        "syncingTo" : "",
        "members" : [
                {
                        "_id" : 0,
                        "name" : "localhost:27018",
                        "health" : 1,
                        "state" : 1,
                        "stateStr" : "PRIMARY",
                {
                        "_id" : 1,
                        "name" : "localhost:27020",
                        "health" : 1,
                        "state" : 2,
                        "stateStr" : "SECONDARY",
```

MongoDB – sharding

- Krok 4: aktywowanie shardingu
  - robimy to z poziomu Query Router-a (mongos)
  - stworzenie bazy danych i uczynienie jej „shardowaną”

```
use persons
sh.enableSharding("persons")
```

```
mongos> use database
switched to db database
mongos> sh.enableSharding("database")
{
        "ok" : 1,
        "operationTime" : Timestamp(1609973535, 5),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1609973535, 5),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
mongos>
```

MongoDB – sharding

- Krok 4: aktywowanie shardingu
  - sprawdzamy status Shard-a

```
mongos> sh.status()
--- Sharding Status ---
  sharding version: {
        "_id" : 1,
        "minCompatibleVersion" : 5,
        "currentVersion" : 6,
        "clusterId" : ObjectId("5ff631d7652906ea21e60ee8")
  }
  shards:
        {  "_id" : "ShardReplSet",  "host" : "ShardReplSet/localhost:27018,localhost:27020",  "state"
  active mongoses:
        "4.2.9" : 1
  autosplit:
        Currently enabled: yes
  balancer:
        Currently enabled:  yes
        Currently running:  no
        Failed balancer rounds in last 5 attempts:  0
        Migration Results for the last 24 hours:
                No recent migrations
  databases:
        {  "_id" : "config",  "primary" : "config",  "partitioned" : true }
                config.system.sessions
                        shard key: { "_id" : 1 }
                        unique: false
                        balancing: true
                        chunks:
                                ShardReplSet    1024
                        too many chunks to print, use verbose if you want to force print
        {  "_id" : "database",  "primary" :  "ShardReplSet",  "partitioned" : true,  "version" : {  "uu
```

MongoDB – sharding

- Krok 5: Demonstracja działania shardingu

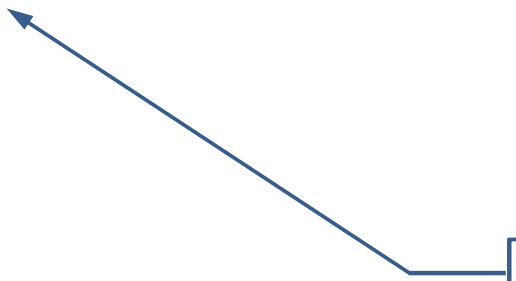
```
use persons

tworzymy przykładową kolekcję
db.createCollection("personscollection")

tworzymy indeks
db.personscollection.createIndex({personid: -1})

dodajemy dwa przykładowe dokumenty
db.personscollection.insertOne({personid: 10001})
db.personscollection.insertOne({personid: 10002})

Indeks musi być typu hashed. Sprawdzamy to
db.personscollection.ensureIndex({personid : "hashed"})

Jeżeli indeks jest w porządku, to możemy aktywować sharding, który będzie używał
personid jako tzw. shard key
sh.shardCollection("persons.personscollection", {personid : "hashed"})

Sprawdzamy status naszego shardingu
db.personscollection.getShardDistribution()
```

MongoDB – sharding

- Krok 5: Demonstracja działania shardingu

```
Sprawdzamy status naszego shardingu
db.personscollection.getShardDistribution()
```

```
mongos> db.personscollection.getShardDistribution()

Shard ShardReplSet at ShardReplSet/localhost:27018,localhost:27020
 data : 80B docs : 2 chunks : 1
 estimated data per chunk : 80B
 estimated docs per chunk : 2

Totals
 data : 80B docs : 2 chunks : 1
 Shard ShardReplSet contains 100% data, 100% docs in cluster, avg obj size on shard : 40B


mongos>
```



Pojawia się nazwa repliki, jakie węzły wchodzą w jej skład oraz ile mamy tzw. chunk-ów

MongoDB – sharding

- Krok 5: Demonstracja działania shardingu
  - przykładowy wynik polecenia getShardDistribution() dla bazy z większą ilością danych

```
Shard shard-a at shard-a/MyMachine.local:30000,MyMachine.local:30001,MyMachine.local:30002
data : 38.14Mb docs : 1000003 chunks : 2
estimated data per chunk : 19.07Mb
estimated docs per chunk : 500001

Shard shard-b at shard-b/MyMachine.local:30100,MyMachine.local:30101,MyMachine.local:30102
data : 38.14Mb docs : 999999 chunks : 3
estimated data per chunk : 12.71Mb
estimated docs per chunk : 333333

Totals
data : 76.29Mb docs : 2000002 chunks : 5
Shard shard-a contains 50% data, 50% docs in cluster, avg obj size on shard : 40b
Shard shard-b contains 49.99% data, 49.99% docs in cluster, avg obj size on shard : 40b
```

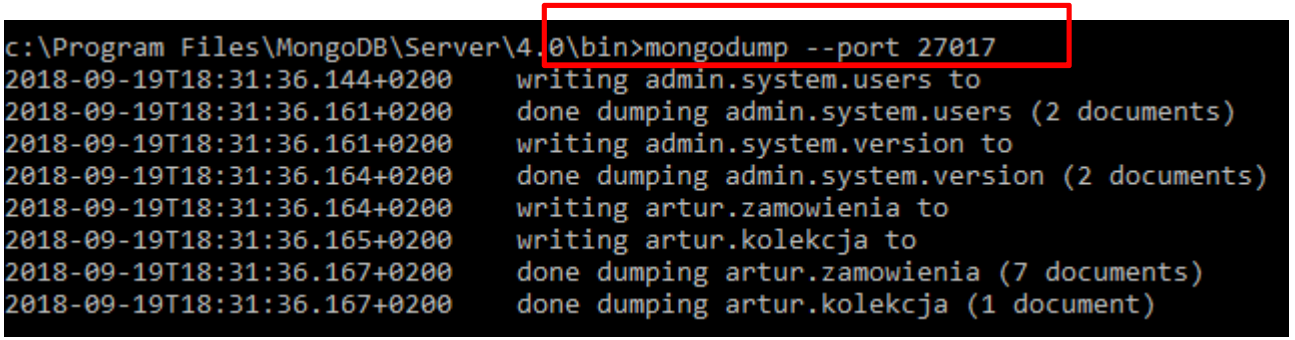
MongoDB – kopia bezpieczeństwa

- Program narzędziowy mongodump
  - tworzy zrzut zawartości wskazanych baz lub kolekcji w formacie binarnym BSON
  - domyślnie baza systemowa local nie jest kopiowana
  - kopiowane są dokumenty oraz ale nie indeksy, te muszą by po imporcie odtworzone (robi to program mongorest). Kopiowana jest natomiast informacja o tym, jakie są pozakładane indeksy
- Program narzędziowy mongorest
  - odtwarza dane z kopii wykonanej programem mongodump oraz odtwarza indeksy, jeżeli były takowe założone
  - gdy odtwarzamy dane do już istniejącej bazy danych to dokumenty, które mają te same wartości pól _id nigdy nie zostają nadpisywane
  - gdy odtwarzamy dane do nowej bazy danych, cała zawartość pliku eksportu jest ładowana do tej bazy
- Programy mongodump oraz mongorest uruchamiamy NIE z konsoli mongo ale z konsoli systemu operacyjnego (w Windows cmd)

MongoDB – kopia bezpieczeństwa

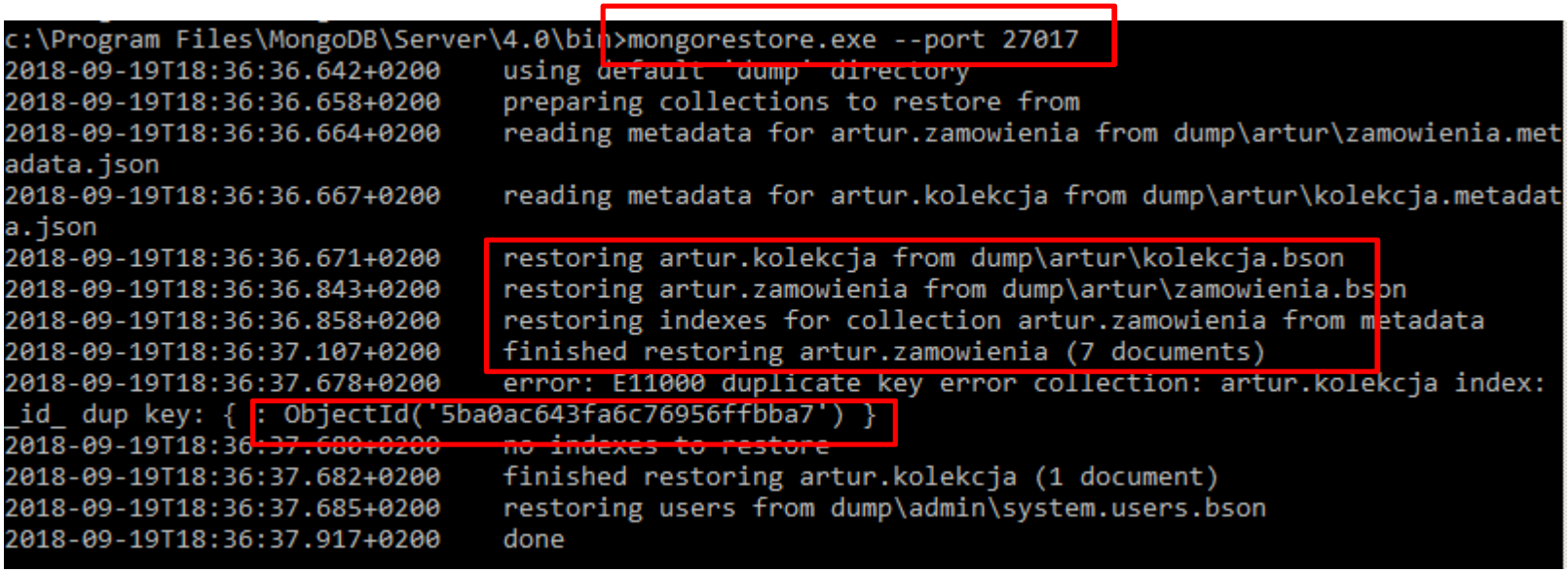
- Przykłady
  - wywołanie beż żadnych parametrów kopiuje wszystko
  - można wskazać konkretną bazę oraz kolekcję do skopiowania
  - użyto tylko najczęściej używanych opcji, jest ich więcej, szczegóły patrz dokumentacja

```
prompt> mongodump --port 27017 --db artur --collection zamowienia
                  --out c:\temp\dump
```



MongoDB – kopia bezpieczeństwa

- Przykłady
  - odtworzenie danych
  - zwróćmy uwagę, że dane w istniejących kolekcjach (u nas kolekcja "artur.kolekcja") nie zostają nadpisane zawartością pliku dump



MongoDB – import i eksport

- Program narzędziowy mongoimport
  - import danych w formatach: JSON, Extended JSON, CSV, TSV
- Program narzędziowy mongoexport
  - eksportuje dane zapisując je w formatach JSON lub CSV

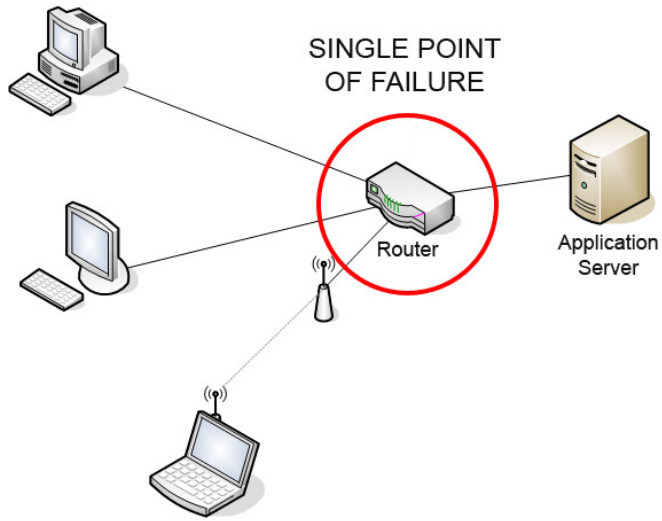
```
mongoimport --db users --collection contacts --file contacts.json

mongoexport --db users --collection contacts --type=csv
            --fields name,address --out /opt/backups/contacts.csv

mongoexport --db sales --collection contacts --out contacts.json

mongoexport --db sales --collection contacts --query '{"field": 1}'
```

Cassandra – podstawowe pojęcia

- Dane przechowywane są w rodzinach kolumn w formie wierszy, do których przypisany jest klucz
  - rodziny kolumn to grupy spokrewnionych danych, które przeważnie pobierane są razem
  - rodzina kolumn jest kontenerem zawierającym kolekcję wierszy
  - każdy wiersz zawiera uporządkowaną kolekcję kolumn
  - kolumna składa się z pary nazwa – wartość, gdzie nazwa odgrywa rolę klucza. Każda z par klucz – wartość jest pojedynczą kolumną i zawsze przechowywana jest wraz ze stemplem czasowym (czas ostatniej aktualizacji kolumny)
  - stempel czasowy jest wykorzystywany do ustalania daty ważności danych, rozwiązywania konfliktów zapisu, radzenia sobie z nieaktualnymi danymi i innych zadań
  - Istnieją też tzw. superkolumny. Jest to specjalny rodzaj kolumny, zawiera kolekcję zagnieżdżonych kolumn

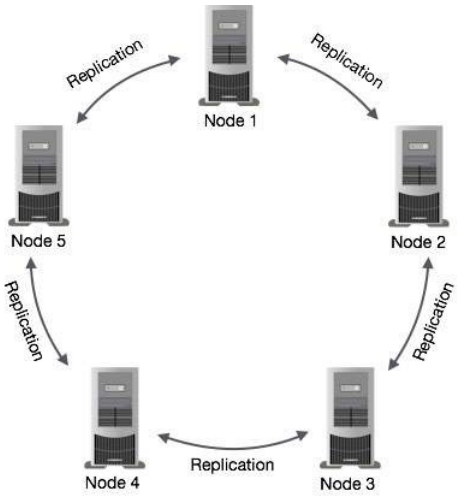
Apache Cassandra – podstawowe informacje

- Stworzony przez Facebook, pierwotnie do zadań przeszukiwania skrzynek pocztowych
- Udostępniony jako open-source w lipcu 2008, w marcu 2009 stała się częścią projektu Apache
- System klasy rodzina kolumn (ang. wide column store)
- Przeznaczony do obsługi bardzo dużych zbiorów danych, w architekturze rozproszonej
- Skalowalność, wysoka niezawodność, brak pojedynczego punktu awarii (ang. single point of failure; SPOF)



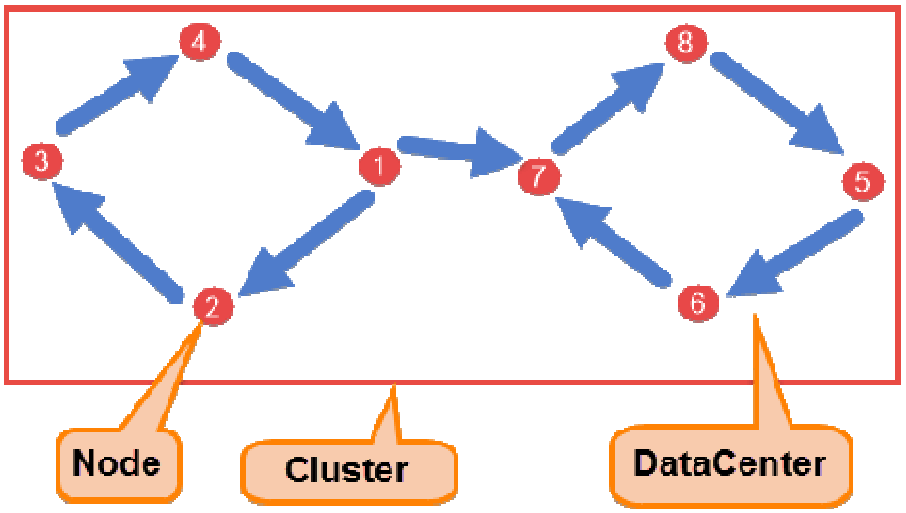
Apache Cassandra – podstawowe informacje

- Dane podlegają rozproszeniu i replikacji pomiędzy węzłami
- Wszystkie węzły odgrywają identyczną rolę, nie ma węzłów wyróżnionych
- Każdy węzeł działa niezależnie
- Po awarii węzła, jego zadania mogą realizować inne węzły
- Po wykryciu, że węzeł posiada nieaktualne dane, Cassandra odnajdzie i zwróci najbardziej aktualny obraz danych, a następnie przystąpi do zaktualizowania danych nieaktualnych (read repair)
- Węzły są zorganizowane w ring
- Konfiguracja w pliku `cassandra.yaml`



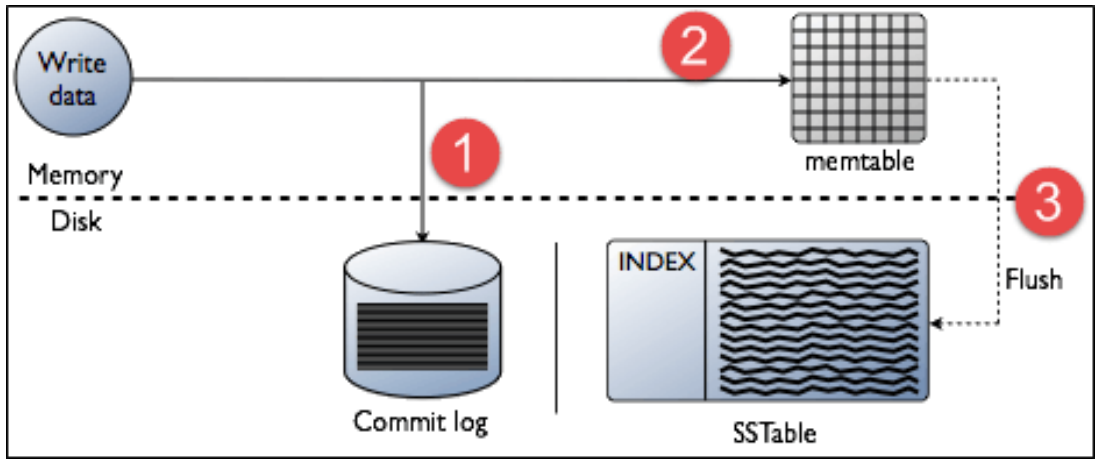
Apache Cassandra – komponenty

- Node – It is the place where data is stored
- Data center – It is a collection of related nodes
- Cluster – A cluster is a component that contains one or more data centers



Apache Cassandra – zapis danych

1. Gdy przychodzi żądanie zapisu danych do węzła, najpierw trafiają one do dziennika commit log
2. Następnie Cassandra zapisuje dane w mem-table. Mem-table to tymczasowy magazyn danych w pamięci operacyjnej (z założenia mniej pewnej niż dysk). Zabezpieczeniem jest równoległy zapis do commit log
3. Gdy rozmiar mem-table przekroczy zadany poziom, jego dane zostają zapisane w pliku dyskowym SSTable



Filtr Blooma

- Wymyślony przez Burtona H. Blooma w 1970 roku
- Filtr Blooma to struktura danych pozwalająca w sposób szybki i pamięciowo optymalny, pozwala wstępnie stwierdzić, czy element należy do zbioru
- Wykorzystywany przez Cassandrę do szybkiego sprawdzenia, czy plik SSTable zawiera wiersz o podanej wartości klucza
- Filtr Blooma ma jedną wadę. Jego wydajność (oszczędność pamięci) jest możliwa dzięki wprowadzeniu marginesu błędnych pozytywnych odpowiedzi
  - „element może jest w zbiorze” (false-positive, czyli że dany element nie istnieje w zadanym zbiorze, a jednak otrzymamy informację o jego prawdopodobnym istnieniu)
  - „elementu na pewno w zbiorze nie ma”
- W Cassandra można ten filtr stroić. Dokładność regulowana m.in. przez rozmiar wektora:

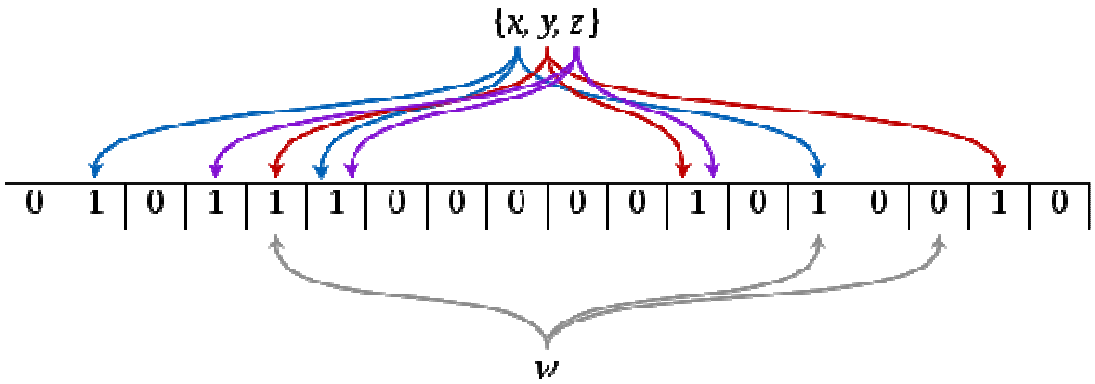
```
ALTER TABLE emp_data WITH bloom_filter_fp_chance = 0.1;
```

Rysunek i tekst: źródło Wikipedia.org

Filtr Blooma

- Filtr Blooma to tablica (wektor) m-bitów, który ma „przewidywać” istnienie *n* elementów. Elementy te zostały zakodowane *k* funkcjami haszującymi
  - funkcja haszująca (skrótu, mieszająca) – funkcja przyporządkowująca dowolnie dużej liczbie krótką, zawsze posiadającą stały rozmiar, niespecyficzną, quasi-losową wartość, tzw. skrót nieodwracalny
- Dobór tych parametrów wpływa na prawdopodobieństwo wystąpienia błędu (które chcemy minimalizować)

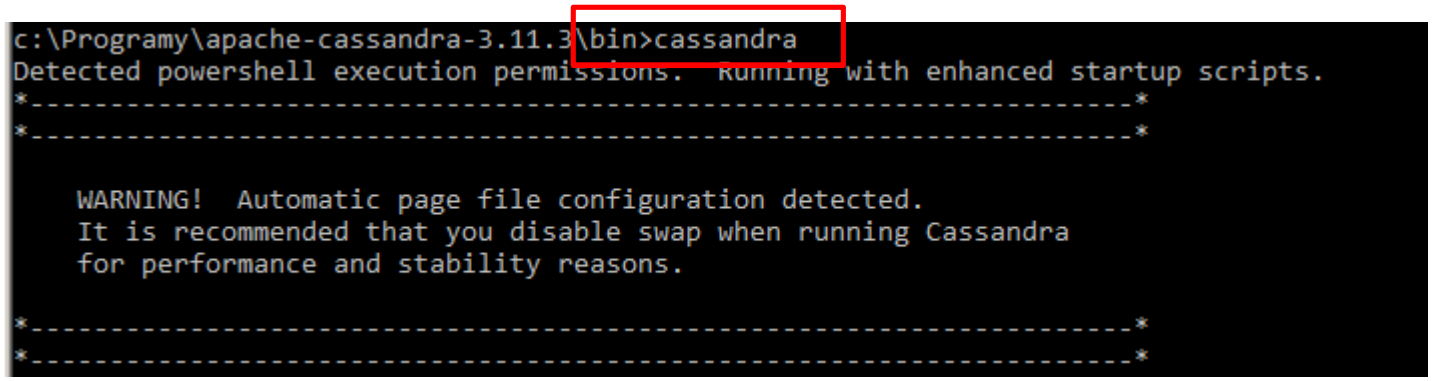
- Parametry: k=3, m=18.
- Kolorowe strzałki wskazują wartości funkcji haszujących dla dodanych elementów *x*, *y*, *z*; szare – dla szukanego elementu *w*.
- Dla szukanego elementu *w* jedna z funkcji haszujących wskazuje w tablicy na wartość 0, co oznacza że tego elementu nie ma w zbiorze.



Rysunek i tekst: źródło Wikipedia.org

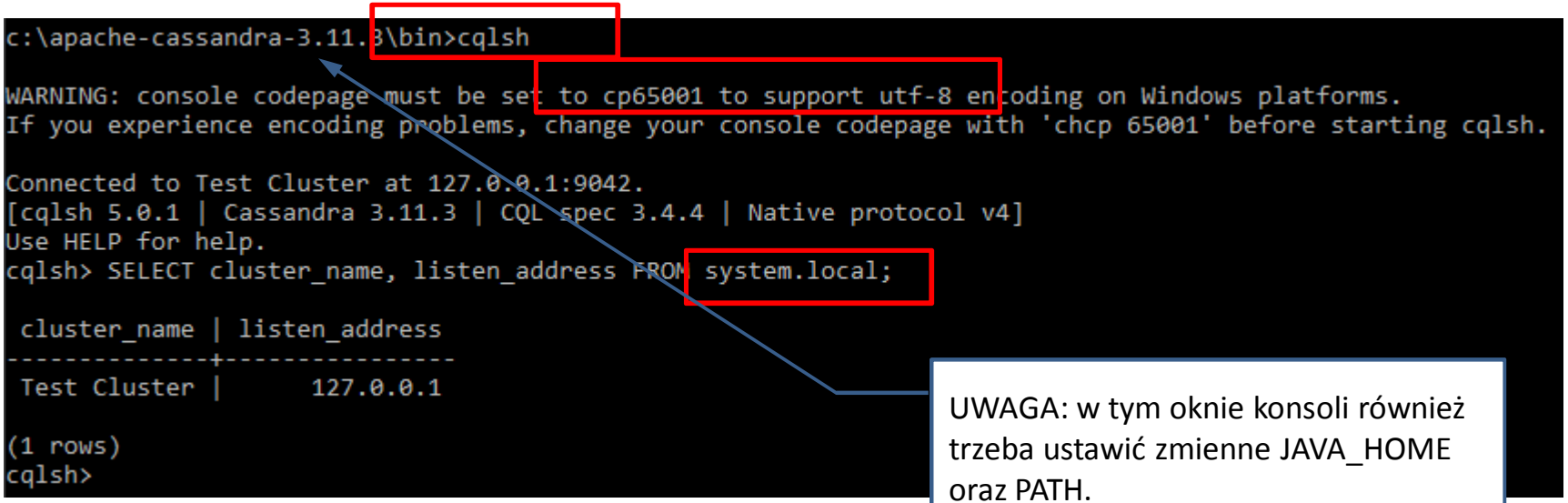
Apache Cassandra – instalacja

- Pobrać plik instalacyjny (np. apache-cassandra-3.11.3-bin.tar.gz) i go rozpakować
- Zainstalować interpreter języka Python 2.7
- Ustawić zmienne JAVA_HOME i PATH, np.:
  - set JAVA_HOME=c:\Program Files\Java\jre1.8.0_152
  - set PATH=c:\Programy\Portable_Python_2.7.6.1\App
  - set PATH=%PATH%;%SystemRoot%\system32\WindowsPowerShell\v1.0



Cassandra – instalacja

- Serwer bazy danych Cassandra o nazwie `cassanda`
- Program kliencki o nazwie `cqlsh` do wydawania poleceń w języku CSQL (podobny do klasycznego języka SQL)
- Do pracy wymagany jest Python wersja 2.7 (dla Cassandra wersja 3.x)



Cassandra – przestrzeń kluczy

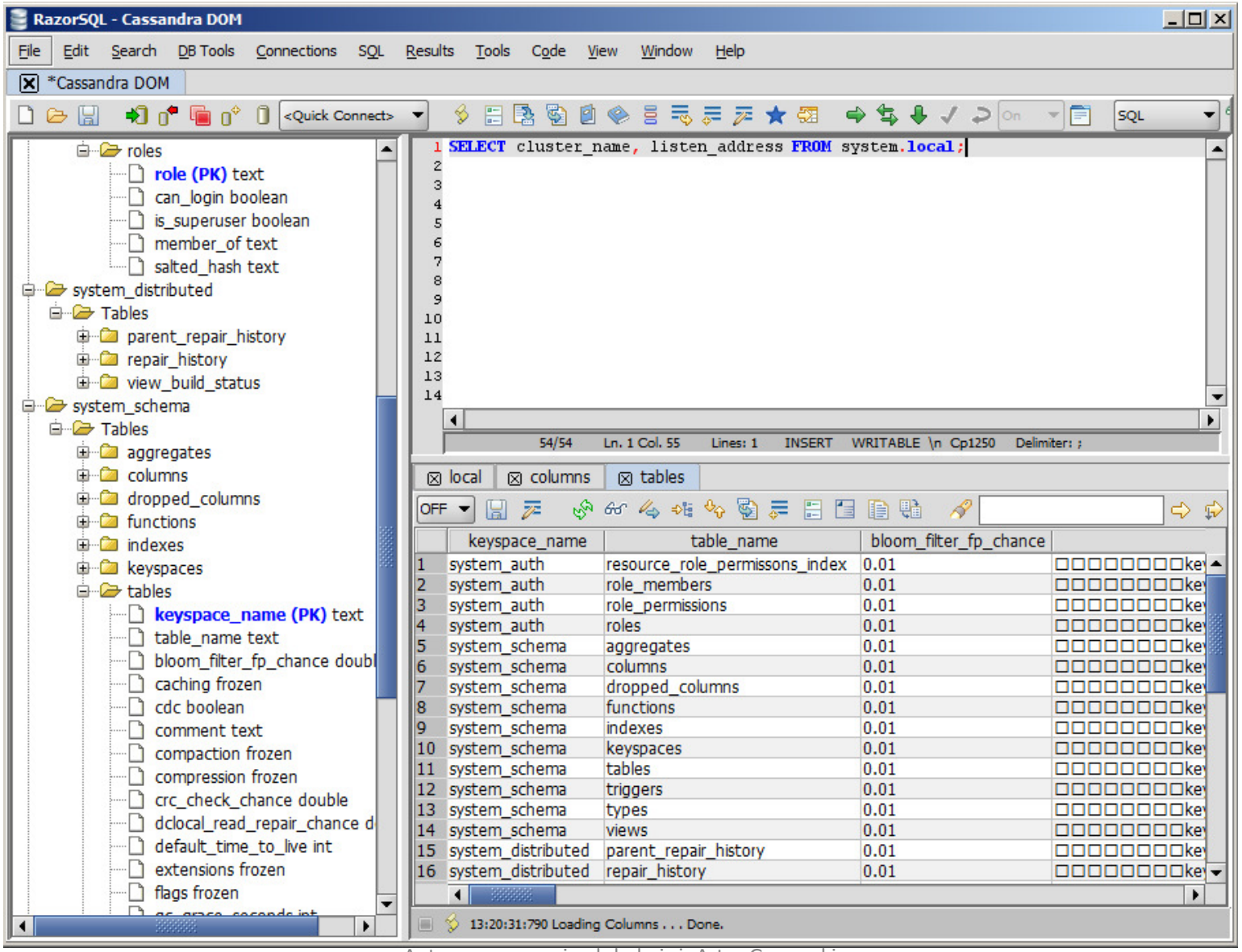
- Cassandra grupuje standardowe rodziny i rodziny superkolumn w przestrzenie kluczy
- Przestrzeń kluczy (Keyspace) – kontener przechowujący dane
  - koncepcja przestrzeni kluczy jest podobna do koncepcji baz danych w systemach transakcyjnych, gdzie przechowywane są wszystkie rodziny kolumn spokrewnione z aplikacją
  - aby rodziny kolumn mogły być przypisywane do przestrzeni kluczy, musimy je najpierw utworzyć
  - przestrzeń kluczy zawiera listę jednej lub wielu rodzin kolumn (ang. column families)

Cassandra – wbudowane typy danych

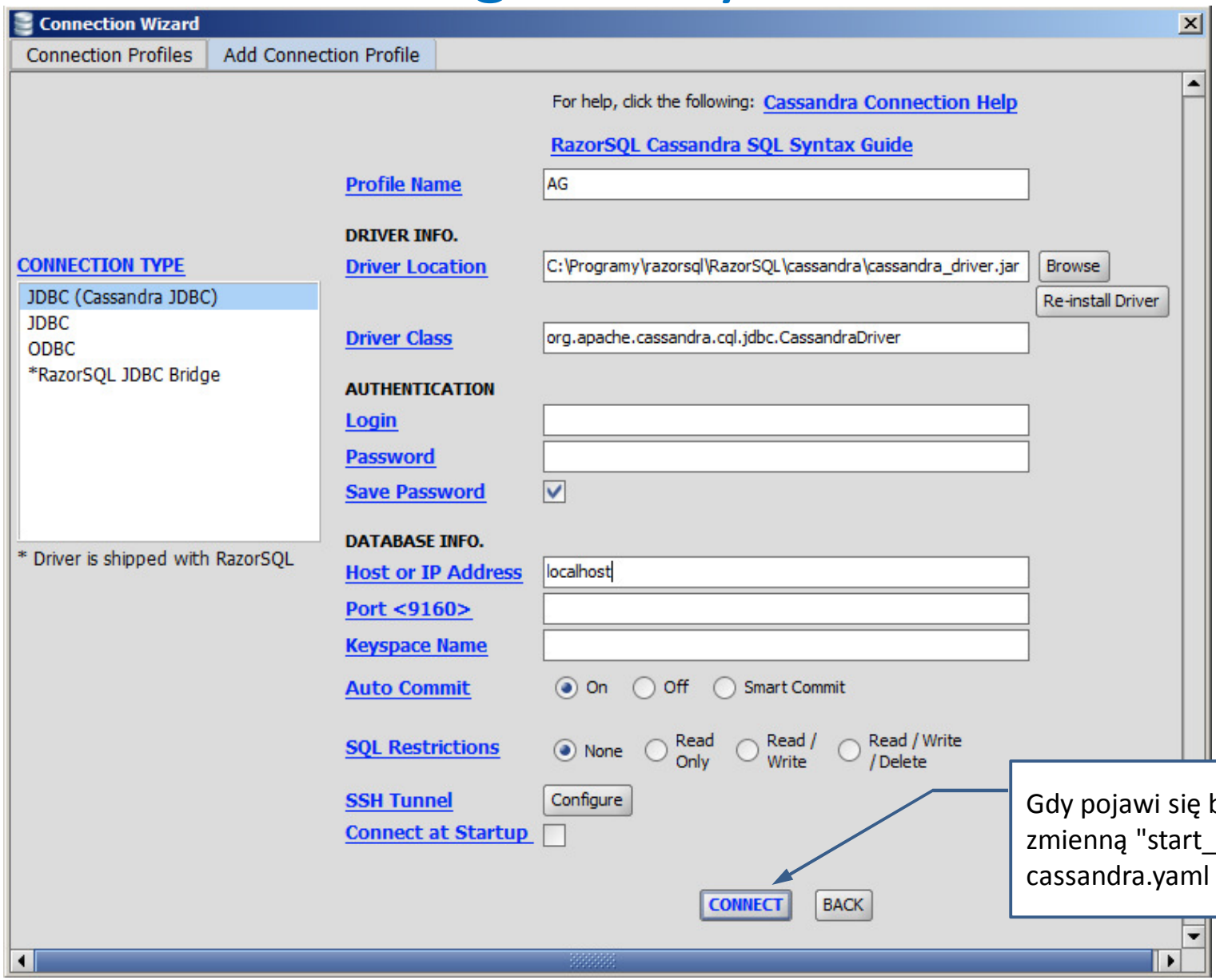
- Można też tworzyć swoje własne typy danych
  - CREATE TYPE
  - ALTER TYPE
  - DROP TYPE
  - DESCRIBE TYPE
  - DESCRIBE TYPES

Data Type	Constants	Description
ascii	strings	Represents ASCII character string
bigint	bigint	Represents 64-bit signed long
blob	blobs	Represents arbitrary bytes
Boolean	booleans	Represents true or false
counter	integers	Represents counter column
decimal	integers, floats	Represents variable-precision decimal
double	integers	Represents 64-bit IEEE-754 floating point
float	integers, floats	Represents 32-bit IEEE-754 floating point
inet	strings	Represents an IP address, IPv4 or IPv6
int	integers	Represents 32-bit signed int
text	strings	Represents UTF8 encoded string
timestamp	integers, strings	Represents a timestamp
timeuuid	uuids	Represents type 1 UUID
uuid	uuids	Represents type 1 or type 4
		UUID
varchar	strings	Represents uTF8 encoded string
varint	integers	Represents arbitrary-precision integer

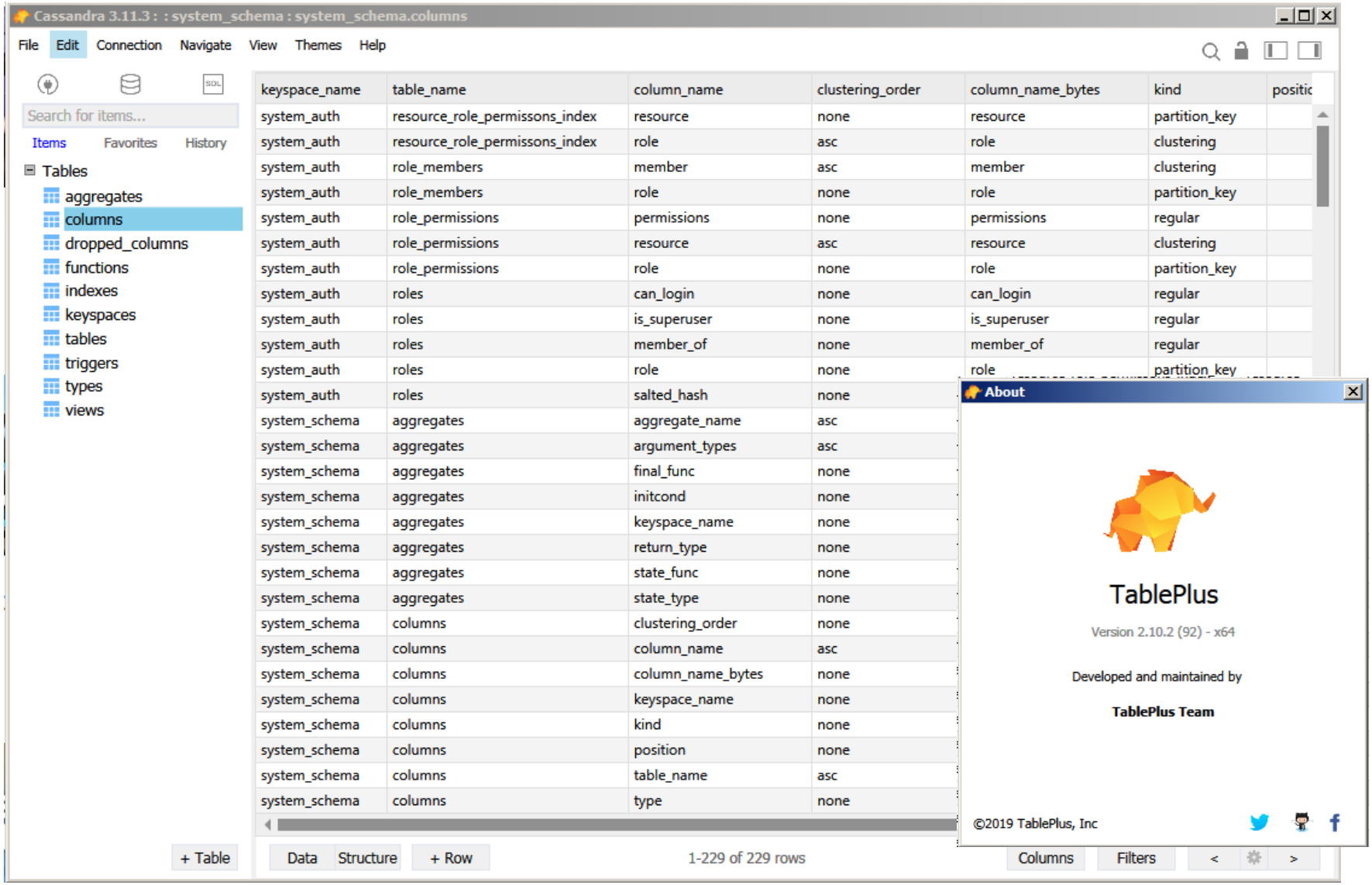
RazorSQL – graficzny klient dla Cassandry (trial na 30 dni)



RazorSQL – graficzny klient dla Cassandry



TablePlus – graficzny klient dla Cassandry



TablePlus – graficzny klient dla Cassandry

- https://tableplus.com/download
- Wspiera bazy Redis, MySQL, SQLite, Cassandra, Cockroach, Snowflake, PostereSQL, Amazon Redsfift, Microsoft SQL Server
- Wersja darmowa z pewnymi ograniczeniami
  - w danej chwili: max 2 otwarte okna, max 2 otwarte zakładki, max 2 zaawansowane filtry
  - do podstawowej pracy w zupełności wystarczy

Cassandra – podstawowe polecenia

- Tworzenie przestrzeni kluczy oraz rodziny kolumn (tabel)

```
DROP KEYSPACE firma;

CREATE KEYSPACE firma WITH REPLICATION = {'class' : 'SimpleStrategy', 'replication_factor' : 3};

USE firma;
DESC TABLES;

CREATE TABLE IF NOT EXISTS pracownik(
  id int,
  imie text,
  nazwisko text,
  data_ur date,
  PRIMARY KEY (id)
);

CREATE TABLE IF NOT EXISTS klient(
  id int,
  imie text,
  nazwisko text,
  PRIMARY KEY (id)
);

INSERT INTO pracownik (id, imie, nazwisko, data_ur) VALUES(1, 'Artur', 'Nowak', '1991-12-29');
INSERT INTO klient (id, imie, nazwisko) VALUES(1, 'Marek', 'Nowak');

SELECT * FROM pracownik;
SELECT * FROM klient
```

```
cqlsh:firma> SELECT * FROM pracownik ;

 id | data_ur    | imie  | nazwisko
----+------------+-------+----------
  1 | 1991-12-29 | Artur |    Nowak

(1 rows)
cqlsh:firma> SELECT * FROM klient;

 id | imie  | nazwisko
----+-------+----------
  1 | Marek |    Nowak

(1 rows)
```

Cassandra – kolekcje

- Kolekcji używamy do przechowywania małych zbiorów danych, np. telefony czy maile jednej osoby
- Dla zawartości, które mogą się w nieskończoność rozrastać, np. lista komentarzy użytkowników, kolekcje nie są dobrym rozwiązaniem; używamy wtedy specjalnej tabeli z kolumnami z grupowaniem
- Jest niedobrym zwyczajem używać kolekcji (pojedynczej) do przechowywania duży ilości danych

Collection	Description
list	A list is a collection of one or more ordered elements.
map	A map is a collection of key-value pairs.
set	A set is a collection of one or more elements.

Cassandra – kolekcje

- Listy
  - lista jest posortowaną kolekcją nieunikalnych wartości
  - elementy są uporządkowane położeniem na liście

```
CREATE KEYSPACE tutorial
  WITH replication = {'class':'SimpleStrategy', 'replication_factor' : 3};

CREATE TABLE data(name text PRIMARY KEY, email list<text>);
INSERT INTO data(name, email) VALUES ('AG', ['abc@gmail.com','def@yahoo.com']);
UPDATE data SET email = email + ['xyz@tutorial.com'] WHERE name = 'AG';
UPDATE data SET email = email + ['xyz@tutorial.com'] WHERE name = 'AG' IF EXISTS;

SELECT * FROM data;
 name | email
------+-----------------------------------------------------------
   AG | ['abc@gmail.com', 'def@yahoo.com', 'xyz@tutorialspoint.com‘]
```

Inaczej niż w klasycznym SQL, UPDATE nie sprawdza, czy aktualizowany wiersz istnieje (z wyjątkiem warunku IF). Wiersz jest tworzony, jeżeli nie istniał albo aktualizowany, jeżeli istniał.

Nie mamy informacji czy wiersz został utworzony czy zaktualizowany.

Cassandra – kolekcje

- Zbiory
  - zbiór jest posortowaną kolekcją unikalnych wartości

```
CREATE TABLE data2(name text PRIMARY KEY, phone set<varint>);
INSERT INTO data2(name, phone) VALUES ('AG', {9848022338, 9848022339});
UPDATE data2 SET phone = phone + {9848022330} WHERE name = 'AG';

SELECT * FROM data2;
   name | phone
--------+-----------------------------------
     AG | {9848022330, 9848022338, 9848022339}



-- Komentarz jednolinijkowy

// Druga możliwość komentarza jednolinijkowego

/*
Komentarz wielolinijkowy
*/
```

Cassandra – kolekcje

- Mapy
  - mapa to posortowany zbiór par klucz-wartość
  - klucze są unikalne
  - mapa jest posortowana po kluczach

```
DROP TABLE data3;
CREATE TABLE data3 (name text PRIMARY KEY, address map<text, text>);

INSERT INTO data3 (name, address)
VALUES ('AG', {'home': 'ZG' , 'office': 'UZ' } );

UPDATE data3 SET address = address + {'office':'WIEA'} WHERE name = AG';

SELECT * FROM data3;
  name | address
-------+-----------------------------------------
    AG | {'home': 'ZG', 'office': 'UZ'}
```

Cassandra – typy definiowane przez użytkownika

- Można je tworzyć, modyfikować i usuwać
- Po utworzeniu odwołujemy się do typu po nazwie
- Jest zbiorek pól z określonymi typami i nazwami, włącznie z kolekcjami oraz innymi typami zdefiniowanymi przez użytkownika
- Domyślnie tworzony jest w bieżącej przestrzeni kluczy, przynależy i jest dostępny tylko w niej
- Można stworzyć go w innej przestrzeni kluczy (trzeba wówczas podać prefiks przed nazwą)

Cassandra – typy definiowane przez użytkownika

- Przykład

```
DROP TABLE osoba;
DROP TYPE IF EXISTS adres;
DROP TYPE IF EXISTS telefon;

CREATE TYPE telefon(
  kod_kraju int,
  numer text,
);

CREATE TYPE adres(
  ulica text,
  miasto text,
  kod text,
  nr_tel map<text, frozen <telefon>>
);

CREATE TABLE osoba(
  imie_nazwisko text PRIMARY KEY,
  adresy map<text, frozen <adres>>
);
```

```
INSERT INTO osoba (imie_nazwisko, adresy)
VALUES ('Jak Kowalski',{
  'dom': {
    ulica: 'Agrestowa 1/1',
    miasto: 'Zielona Gora',
    kod: '65-780',
    nr_tel: {
      'komorka': {kod_kraju: 58, numer: '665032413'},
      'stacjonarny': {kod_kraju: 58, numer: '343276173'}
    }
  },
  'praca': {
     ulica:'Podgorna 50',
     miasto: 'Zielona Gora',
     kod: '65-246',
     nr_tel: {
       'fax': {kod_kraju: 58, numer: '343185692'}
     }
  }
});
```

Pola typu frozen serializują wiele pól w jedną wartość, nie można aktualizować poszczególnych wartości wewnątrz

Cassandra – typy definiowane przez użytkownika

- Przykład, wynik w konsoli programu cqlsh (niezbyt czytelny)

```
cqlsh:firma> SELECT * FROM osoba;

 imie_nazwisko | adresy
---------------+-------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
------
  Jak Kowalski | {'dom': {ulica: 'Agrestowa 1/1', miasto: 'Zielona Gora', kod: '
65-780', nr_tel: {'komorka': {kod_kraju: 58, numer: '665032413'}, 'stacjonarny':
 {kod_kraju: 58, numer: '343276173'}}}, 'praca': {ulica: 'Podgorna 50', miasto:
'Zielona Gora', kod: '65-246', nr_tel: {'fax': {kod_kraju: 58, numer: '343185692
'}}}}

(1 rows)
```

Cassandra – tworzenie klastra

- https://www.jamescoyle.net/how-to/2448-create-a-simple-cassandra-cluster-with-3-nodes
- https://docs.datastax.com/en/cassandra/2.1/cassandra/initialize/initializeSingleDS.html
- Wymagania
  - na każdym węźle mamy instalację Cassandry
  - żaden firewall nie blokuje wymaganych portów
    - 7000 TCP Non-encrypted inter-node cluster communication. Not used if SSL is in use
    - 7001 TCP Encrypted SSL inter-node cluster communication. Not used if SSL is not in use
    - 7199 TCP JMX monitoring port
    - 9042 TCP Client port used for native CQL
    - 9160 TCP Client port used for Thrift
  - znamy IP każdego węzła
  - na żadnym węźle (instancji Cassandry) nie ma danych. Gdy jakieś dane są, to trzeba wyczyścić na każdym węźle katalog `/data/system`

Cassandra – tworzenie klastra

- Edycja plików `cassandra.yaml` na każdym węźle

Example for node 1:

```
cluster_name: 'JC Cluster'
num_tokens: 256
seed_provider:
    - class_name: org.apache.cassandra.locator.SimpleSeedProvider
        - seeds: 10.0.0.1, 10.0.0.2
listen_address: 10.0.0.1
rpc_address: 10.0.0.1
endpoint_snitch: GossipingPropertyFileSnitch
```

Example for node 2:

```
cluster_name: 'JC Cluster'
num_tokens: 256
seed_provider:
    - class_name: org.apache.cassandra.locator.SimpleSeedProvider
        - seeds: 10.0.0.1, 10.0.0.2
listen_address: 10.0.0.2
rpc_address: 10.0.0.2
endpoint_snitch: GossipingPropertyFileSnitch
```

Example for node 3:

```
cluster_name: 'JC Cluster'
num_tokens: 256
seed_provider:
    - class_name: org.apache.cassandra.locator.SimpleSeedProvider
        - seeds: 10.0.0.1, 10.0.0.2
listen_address: 10.0.0.3
rpc_address: 10.0.0.3
endpoint_snitch: GossipingPropertyFileSnitch
```

Cassandra – tworzenie klastra

- Edycja plików `cassandra-rackdc.properties` na każdym węźle

Example for node 1:

```
dc=uk_dc
rack=rack1
```

Example for node 2:

```
dc=uk_dc
rack=rack1
```

Example for node 3:

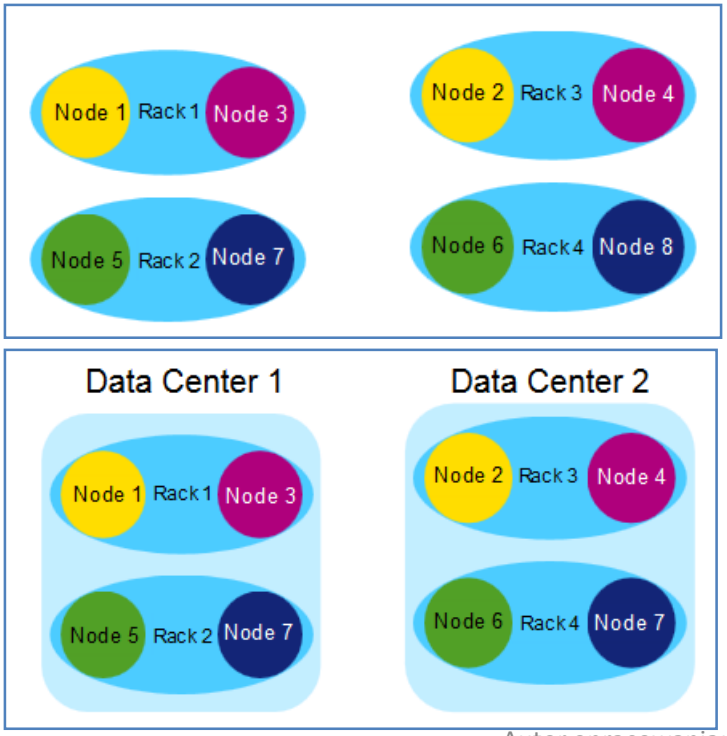
```
dc=uk_dc
rack=rack2
```

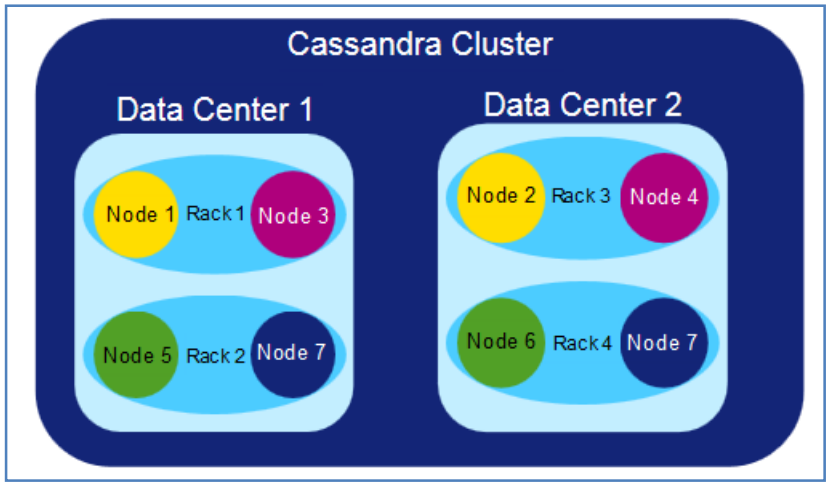
Cassandra – tworzenie klastra

- Wyjaśnienie skrótów dc oraz rack
  - Cassandra is built to be fault tolerant and will distribute data to try to minimize the risk of a failure causing loss of data or any downtime. Cassandra therefore has the understanding of a node, a rack and a data centre. Where possible, Cassandra will ensure that the data and it's backups are stored on a different rack and a different data centre to ensure that failure, even at a data centre level isn't catastrophic

Cassandra – pojęcia: node, rack, data center, cluster

- Node – serwer, z zainstalowaną Cassandrą
- Rack – logiczny zbiór węzłów (nodes)
- Data center – logiczny zbiór rack-ów
- Cassandra cluster = nodes + racks + data centers





Rysunki z: https://slideplayer.com/slide/9317099/

Cassandra – tworzenie klastra

- Usunąć pliki `cassandra-topology.properties` na każdym węźle
- Wystartować na każdym węźle instancję Cassandra
- Sprawdzić status klastra poleceniem *nodetool status*
  - polecenie to można uruchomić z dowolnego węzła
  - poniżej mamy widoczne 3 węzły tworzące data center o nazwie uk_dc, skonfigurowane są 2 rack-i

```
nodetool status

Datacenter: uk_dc
==========
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address      Load       Tokens       Owns    Host ID                               Rack
UN  10.0.0.3  124.64 KB  256          ?       bb57bbee-3fe4-47b1-9249-cd3f90cd9718  rack2
UN  10.0.0.2  124.7 KB   256          ?       6669bac4-52c5-49fb-a68a-da065f20ae2c  rack1
UN  10.0.0.1  106.45 KB  256          ?       ddee28dd-7909-4428-bebd-023e4e560db5  rack1

Note: Non-system keyspaces don't have the same replication settings, effective ownership information is meaningless
```

Cassandra – tworzenie klastra

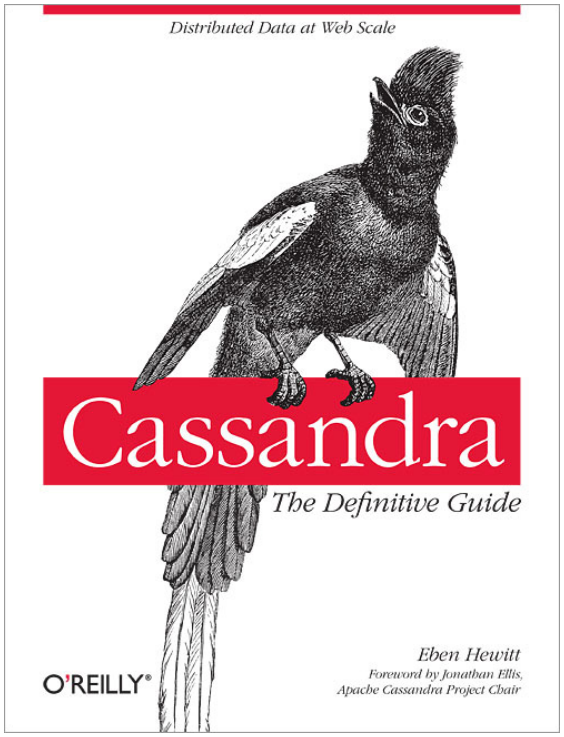
- Podłączenie się do klastra
  - z dowolnego węzła

```
cqlsh 10.0.0.1

Connected to Uptime2 at 10.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.2.1 | CQL spec 3.4.0 | Native protocol v4]
Use HELP for help.
cqlsh>
```

- Bardziej szczegółowe omówienie architektury i działania klastra można znaleźć tutaj: https://slideplayer.com/slide/9317099/

Cassandra - dokumentacja

- http://www.tutorialspoint.com/cassandra/
- http://www.planetcassandra.org/
- https://www.safaribooksonline.com/library/view/cassandra-the-definitive/9781449399764/



Redis – podstawowe informacje

- Redis (skrót od **Re**mote **Di**ctionary **S**ervice)
- Baza typu Key-Value
- Open Source
- Baza działa w pamięci operacyjnej a z powodów bezpieczeństwa dane są również zapisywane na dysku. Dzięki temu jest niezwykle szybka (czasy odczytu i zapisu są w zasadzie takie same)
  - stąd używana czasem nazwa: In-Memory, Key-Value Store
- Może działać w klastrze
- Oprogramowanie serwera (*redis-server*) oraz klienta (*redis-cli*)
- Wbudowany test wydajności (program *redis-benchmark*)
- Konfiguracja: plik *redic.conf*

Redis – podstawowe informacje

- Inaczej niż wiele innych baz key-value, Redis wspiera, oprócz stringów, też kilka bardziej dokładnie zdefiniowanych typów danych
  - String – łańcuch znakowy. Podstawowy typ danych w Redis. Ciągi tekstowe są binarnie bezpieczne (ang.Binary safe), co oznacza, że Redis nie analizuje zawartości przekazywanych mu danych. Mogą przechowywać do maks. 512 MB danych.
  - List – lista łańcuchów znakowych, $2^{32} - 1$ elementów. Zaletą list jest ten sam czas dodania elementu zarówno na początku jak i na końcu listy niezależnie od rozmiaru listy
  - Set – nieuporządkowany zbiór łańcuchów znakowych. Dodawanie, usuwanie czy wyszukiwanie elementu ma stałą złożoność obliczeniową równą O(1). Maks. 4 miliardy elementów
  - Sorted sets – jak Set ale wersja posortowana. Każdy element uporządkowanego zbioru ma przydzielony numer (score), który pozwala na posortowanie zbioru od najmniejszego do największego. Możemy szybko manipulować elementami posortowanego zbioru, ponieważ elementy są od razu wstawiane w odpowiedniej kolejności, zamiast sortowania po wstawieniu.

Redis – podstawowe informacje

- Hash (tablice asocjacyjne) – kolekcja par pole-wartość, maks. 4 miliardy par. Tablice mogą być wykorzystane do reprezentacji połączonych ze sobą struktur danych za pomocą referencji
- Bit-array (bitmapy) – za pomocą specjalnych komend można string traktować jako strumień bitów (można ustawić dowolny bit)
- Geospatial – obsługa danych przestrzennych
- HyperLogLogs – struktura pozwalająca szybko zliczać liczbę unikalnych elementów w zbiorze (przy użyciu niewielkich zasobów pamięciowych z błędem standardowym < 0.1 %. Struktura może przechowywać 2^{64} elementów)

Redis – podstawowe informacje

- Wspiera skrypty w języku Lua
- Oczywiście brak SQL (jak praktycznie każda baza NoSQL)
- Obsługuje transakcje. Baza gwarantuje obsługę własności atomowości oraz izolacji
  - wszystkie polecenia w transakcji są realizowane jako pojedyncza izolowana operacja. Jednoczesna modyfikacja nie jest możliwa przez inną operacje
  - transakcja jest atomowa, albo wykonane są wszystkie operacje, albo wszystkie są wycofane
- Polecenia wspierające transakcyjność:
  - MULTI (rozpoczęcie transakcji)
  - EXEC (zakończenie transakcji)
  - DISCARD (wycofanie transakcji)
  - WATCH (zakładanie blokady)
  - UNWATCH (zdejmowanie blokady)

Redis – podstawowe informacje

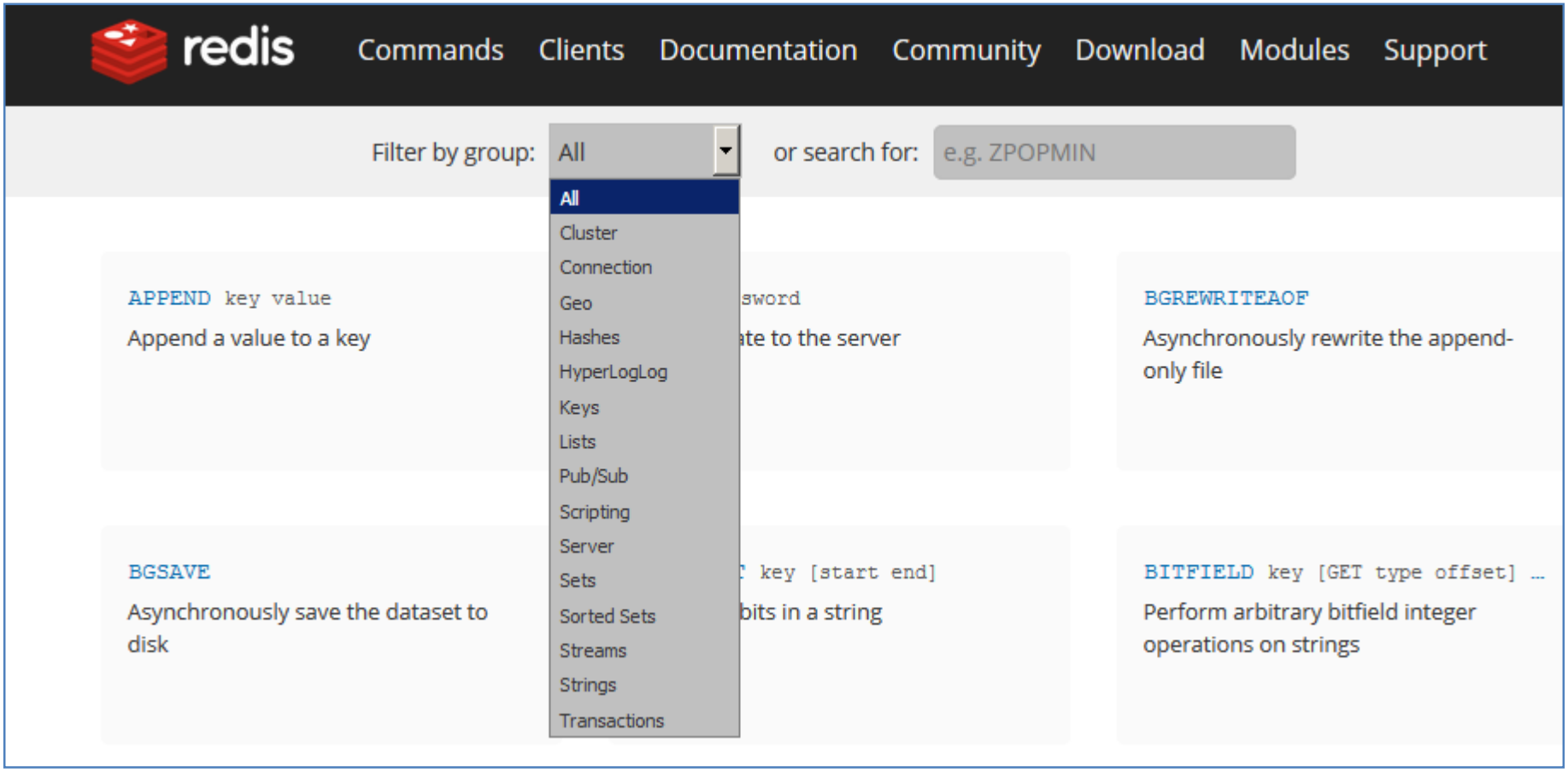
- Naturalnym środowiskiem pracy jest UNIX/Linux, ale też istnieje kompilacja dla Windows (stworzone i utrzymywane przez Microsoft)
  - https://github.com/MicrosoftArchive/redis/releases
- Ma wbudowane powielanie danych (replikację master-slave)
- Wspiera usuwanie najmniej używanych danych (LRU – least recently used)
- Dzięki Redis Cluser zapewnia automatyczne partycjonowanie danych
- Dzięki Redis Sentinel zapewnia wysoki poziom dostępności danych
- Ma wbudowane różne poziomy utrwalania danych na dysku
  - poprzez zrzucanie danych na dysk co jakiś określony czas
  - poprzez dodawanie każdej komendy do dziennika (log)
  - można ten mechanizm wyłączyć

Redis – podstawowe informacje

- Automatyczna obsługa pracy awaryjnej (serwer podrzędny staje się nadrzędnym)
- Obsługa kluczy z ograniczonym czasem życia
  - TTL (Time-To-Live)
- Nie zapewnia trwałości danych w przypadku wystąpienia awarii
  - ale jest obsługa migawek (snapshot)
  - oraz bezpieczniejsza alternatywa AOF (append-only file). Jest to dziennik (ang. journal), który zapisuje wszystkie zmiany wykonywane w bazie. Pozwala wrócić do jakiegokolwiek wcześniejszego stanu
- Ma wsparcie dla wielkiej liczby języków programowania (zbudowane na bazie firmowego API Redis)

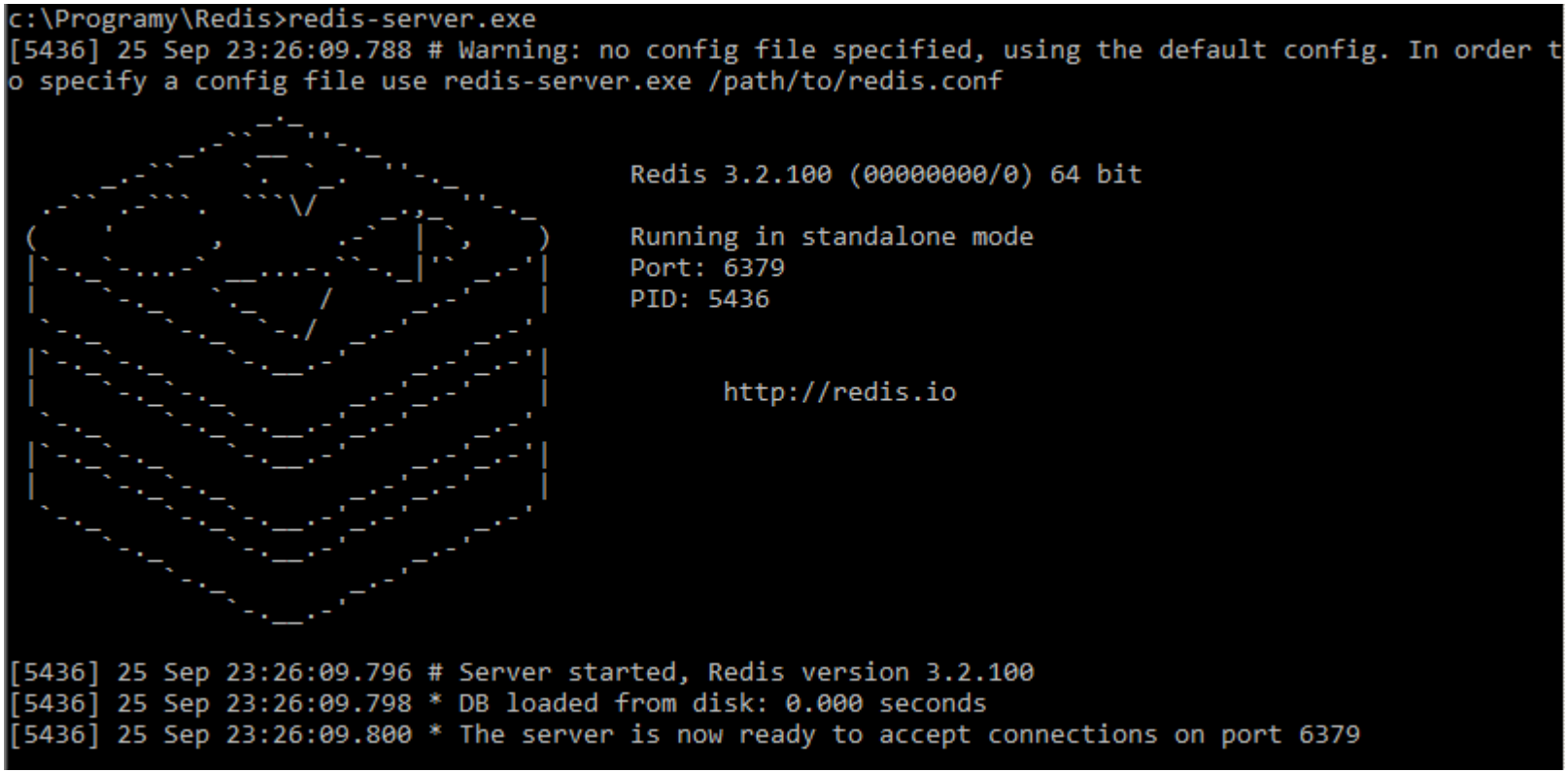
Redis – podstawowe informacje

- Jest ponad 200 poleceń, podzielonych na 15 grup



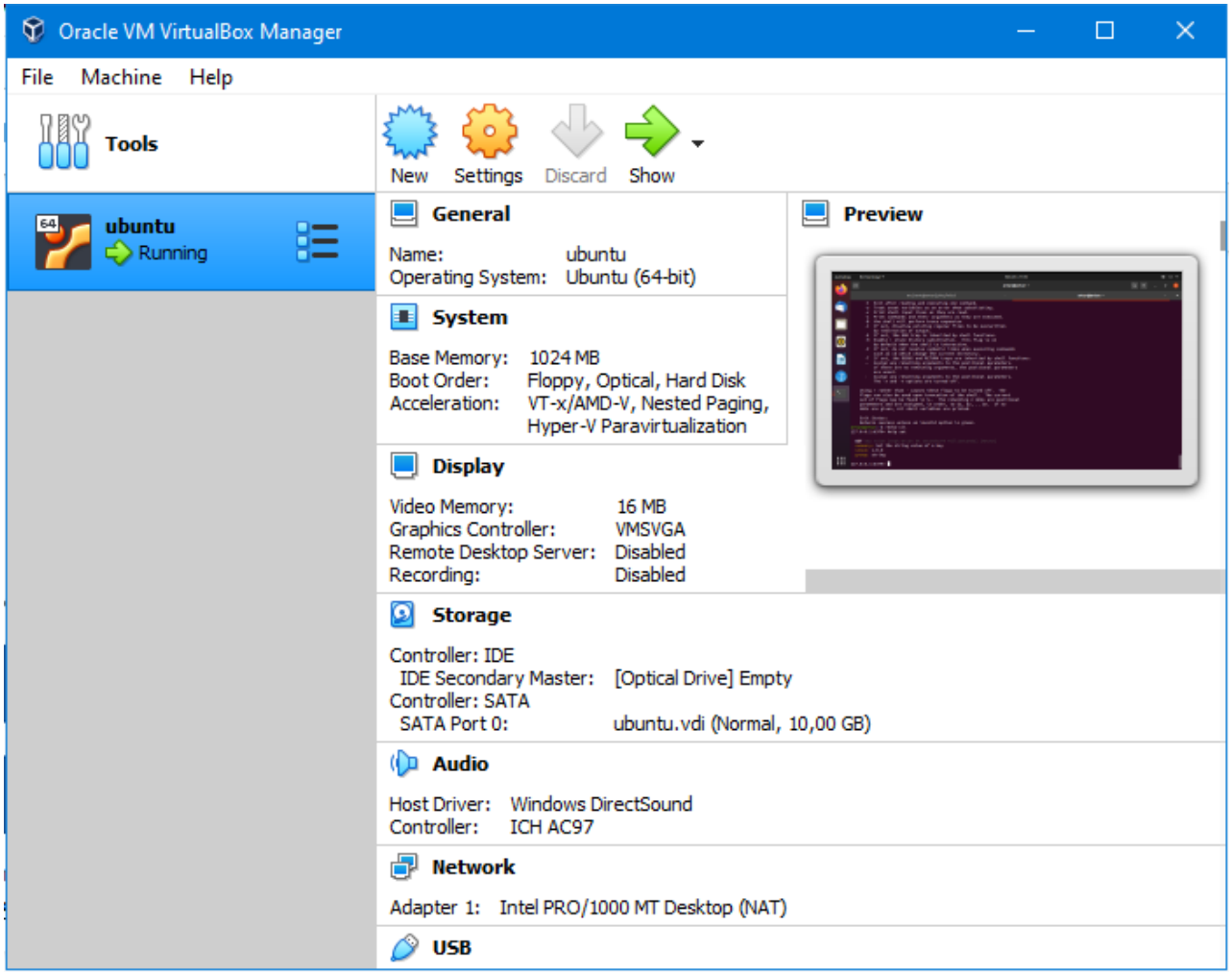
Redis – podstawy użytkowania

- Pobieramy oprogramowanie z: https://github.com/MicrosoftArchive/redis/releases
- Uruchamiamy serwer

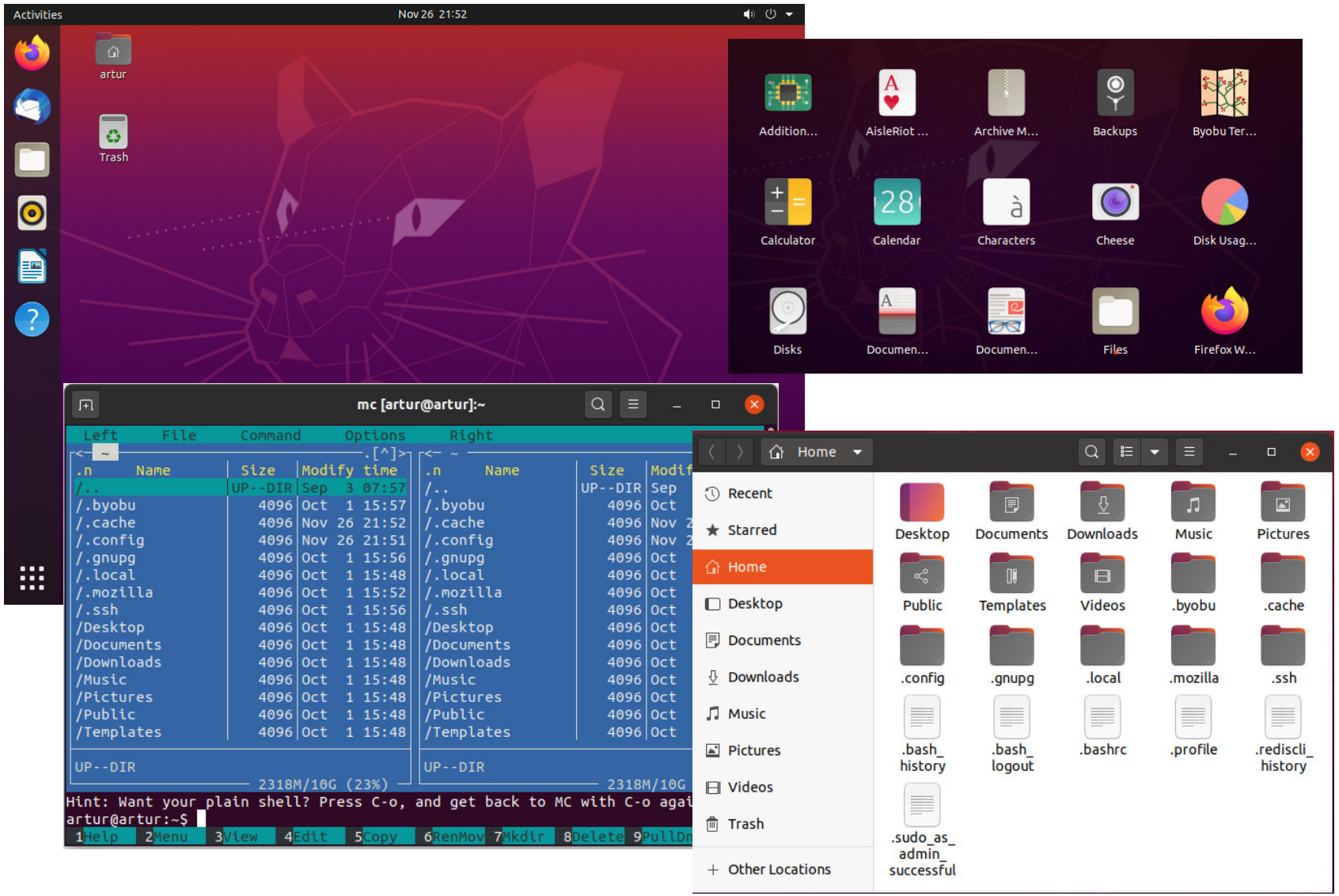


Redis – instalacja w Linux

- Oracle VM VirtualBox



Redis – instalacja w Linux (Ubuntu)



Redis – instalacja w Linux

- https://www.tutorialspoint.com/redis/redis_quick_guide.htm

Install Redis on Ubuntu

To install Redis on Ubuntu, go to the terminal and type the following commands –

```
$sudo apt-get update
$sudo apt-get install redis-server
```

This will install Redis on your machine.

Start Redis

```
$redis-server
```

Check If Redis is Working

```
$redis-cli
```

This will open a redis prompt.

```
redis 127.0.0.1:6379>
```

In the above prompt, **127.0.0.1** is your machine's IP address and **6379** is the port on which Redis server is running. Now type the following **PING** command.

```
redis 127.0.0.1:6379> ping
PONG
```

Redis – instalacja w Linux

- Otwarcie, zamknięcie serwera, test działania
  - przełączenie na root
  - start/stop usługi
  - uruchomienie klienta i test działania

Redis – podstawy użytkowania

- Podstawowe parametry uruchomieniowe

```
c:\Programy\Redis>redis-server.exe --help
Usage: ./redis-server [/path/to/redis.conf] [options]
       ./redis-server - (read config from stdin)
       ./redis-server -v or --version
       ./redis-server -h or --help
       ./redis-server --test-memory <megabytes>

Examples:
       ./redis-server (run the server with default conf)
       ./redis-server /etc/redis/6379.conf
       ./redis-server --port 7777
       ./redis-server --port 7777 --slaveof 127.0.0.1 8888
       ./redis-server /etc/myredis.conf --loglevel verbose

Sentinel mode:
       ./redis-server /etc/sentinel.conf --sentinel
```

- Serwer można zainstalować jako usługę Windows
  - patrz dokumentacja *Windows Service Documentation.docx*

Redis – podstawy użytkowania

- Program kliencki redis-cli
  - wysyłanie poleceń do serwera i odczytywanie odpowiedzi
  - dwa tryby pracy
    - interakcyjny
    - poleceniowy (polecenia jako argumenty redis-cli)
  - test działania (polecenie ping)

```
c:\Programy\Redis>redis-cli.exe
127.0.0.1:6379> ping
PONG
127.0.0.1:6379>
```

- Wstawianie danych, odczyty danych, kasowanie danych
  - polecenia SET, GET, DEL
  - bazy klucz-wartość przechowują konkretną pod wskazanym kluczem
  - umieszczoną daną można odczytać tylko, jeśli zna się dokładną wartość klucza

Redis-cli help

```
c:\Programy\Redis>redis-cli.exe --help
redis-cli 3.2.100

Usage: redis-cli [OPTIONS] [cmd [arg [arg ...]]]
  -h <hostname>      Server hostname (default: 127.0.0.1).
  -p <port>          Server port (default: 6379).
  -s <socket>        Server socket (overrides hostname and port).
  -a <password>      Password to use when connecting to the server.
  -r <repeat>        Execute specified command N times.
  -i <interval>      When -r is used, waits <interval> seconds per command.
                     It is possible to specify sub-second times like -i 0.1.
  -n <db>            Database number.
  -x                 Read last argument from STDIN.
  -d <delimiter>     Multi-bulk delimiter in for raw formatting (default: \n).
  -c                 Enable cluster mode (follow -ASK and -MOVED redirections).
  --raw              Use raw formatting for replies (default when STDOUT is
                     not a tty).
  --no-raw           Force formatted output even when STDOUT is not a tty.
  --csv              Output in CSV format.
  --stat             Print rolling stats about server: mem, clients, ...
  --latency          Enter a special mode continuously sampling latency.
  --latency-history  Like --latency but tracking latency changes over time.
                     Default time interval is 15 sec. Change it using -i.
  --latency-dist     Shows latency as a spectrum, requires xterm 256 colors.
                     Default time interval is 1 sec. Change it using -i.
  --lru-test <keys>  Simulate a cache workload with an 80-20 distribution.
  --slave            Simulate a slave showing commands received from the master.
  --rdb <filename>   Transfer an RDB dump from remote server to local file.
  --pipe             Transfer raw Redis protocol from stdin to server.
  --pipe-timeout <n> In --pipe mode, abort with error if after sending all data.
                     no reply is received within <n> seconds.
                     Default timeout: 30. Use 0 to wait forever.
  --bigkeys          Sample Redis keys looking for big keys.
  --scan             List all keys using the SCAN command.
  --pattern <pat>    Useful with --scan to specify a SCAN pattern.
  --intrinsic-latency <sec> Run a test to measure intrinsic system latency.
                     The test will run for the specified amount of seconds.
  --eval <file>      Send an EVAL command using the Lua script at <file>.
  --ldb              Used with --eval enable the Redis Lua debugger.
  --ldb-sync-mode    Like --ldb but uses the synchronous Lua debugger, in
                     this mode the server is blocked and script changes are
                     are not rolled back from the server memory.
  --help             Output this help and exit.
  --version          Output version and exit.
```

Redis – SET, GET, DEL

```
127.0.0.1:6379> SET "email:AG" "a.gramacki@issi.uz.zgora.pl"
OK

127.0.0.1:6379> GET email:AG
"a.gramacki@issi.uz.zgora.pl"

127.0.0.1:6379> GET email:AGG
(nil)

127.0.0.1:6379> EXISTS email:AG
(integer) 1

127.0.0.1:6379> EXISTS email:AGG
(integer) 0

127.0.0.1:6379> TYPE email:AG
string

127.0.0.1:6379> SETNX "email:AG" "a.gramacki@issi.uz.zgora.pl"
(integer) 0

127.0.0.1:6379> SETNX "email:JG" "j.gramacki@ck.uz.zgora.pl"
(integer) 1

127.0.0.1:6379> DEL email:AG
(integer) 1
```

Cudzysłowy nie są niezbędne ale poprawiają czytelność więc warto je stosować.

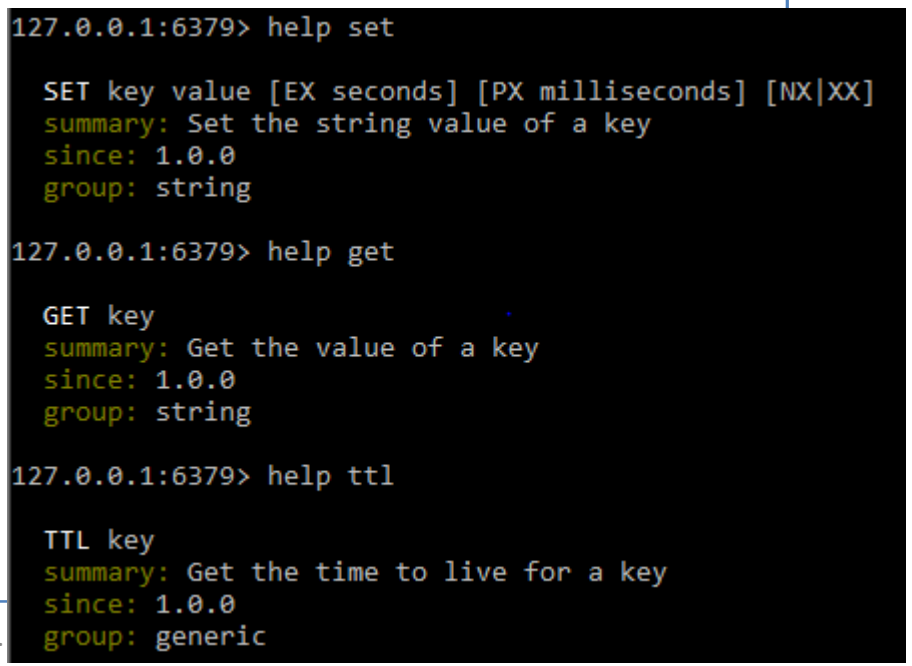
Wielkość liter dla poleceń nie ma znaczenia. My stosujemy duże litery, poprawia to czytelność.

SETNX (SET-if-Not-eXists): wstawia dane, jeśli wcześniej nie było ich w bazie.

Redis – zmienne czasowe

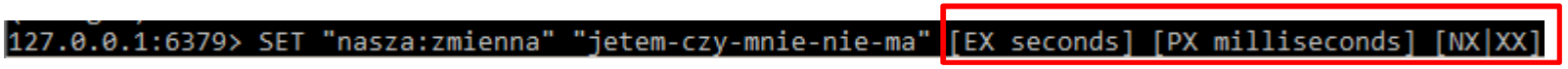
- Dane mogą mieć ustawiony określony czas trwania
  - TTL (Time-To-Live), polecenie pozwala odczytać pozostały czas
  - EXPIRE, ustawia czas ważności zmiennej
  - PTTL, zwraca pozostały czas życia zmiennej w milisekundach

```
127.0.0.1:6379> SET "nasza:zmienna" "jetem-czy-mnie-nie-ma"
OK
127.0.0.1:6379> EXPIRE nasza:zmienna 30
(integer) 1
127.0.0.1:6379> TTL nasza:zmienna
(integer) 24
127.0.0.1:6379> PTTL nasza:zmienna
(integer) 22737
127.0.0.1:6379> TTL nasza:zmienna
(integer) 20
127.0.0.1:6379> PTTL nasza:zmienna
(integer) 19242
127.0.0.1:6379> TTL nasza:zmienna
(integer) -2
127.0.0.1:6379> PTTL nasza:zmienna
(integer) -2
```



Redis – zmienne czasowe

- Opcje polecenia SET
  - EX, ustala czas życia zmiennej w sekundach
  - PX, ustala czas życia zmiennej w milisekundach
  - NX, ustawia zmienną tylko, jeśli jeszcze nie istnieje
  - XX, ustawia zmienną tylko, jeśli już istnieje



Redis – tryb autoryzowany

- Domyślnie serwer pracuje w trybie autoryzowanym
  - aby zawartość pliku `redis.conf` została odczytana, należy serwer uruchomić z parametrem (ścieżką i nazwą pliku konfiguracyjnego). W przeciwnym wypadku użyte zostaną wartości domyślne
    - `c:\redis\redis-server.exe redis.windows.conf`
  - zmienna `protected-mode yes` w pliku `redis.conf`
  - domyślnie hasło nie jest jednak ustawione (zmienna `requirepass` jest zakomentowana)
  - zmienna `requirepass` pozwala ustalić hasło dostępu do bazy. Będzie wówczas wymagane uwierzytelnianie poleceniem AUTH lub trzeba uruchomić klienta z parametrem `-a`
  - hasło powinno być bardzo silne, gdyż z godnie z dokumentacją w pliku redis.conf:

    "Warning: since Redis is pretty fast an outside user can try up to 150k passwords per second against a good box. This means that you should use a very strong password otherwise it will be very easy to break."

Redis – tryb autoryzowany

```
c:\Programy\redis-server.exe redis.windows.conf

c:\Programy\Redis>redis-cli.exe
127.0.0.1:6379> get email:AG
(error) NOAUTH Authentication required.

127.0.0.1:6379> AUTH moje-tajne-haslo
OK.

127.0.0.1:6379> GET email:AG
"a.gramacki@issi.uz.zgora.pl

c:\Programy\Redis>redis-cli.exe -a moje-tajne-haslo
127.0.0.1:6379> GET email:AG
"a.gramacki@issi.uz.zgora.pl"

127.0.0.1:6379> CONFIG GET requirepass
1) "requirepass"
2) "moje-tajne-haslo"
```

Jako parametr podano nazwę pliku konfiguracyjnego.

Wykonanie każdego polecenia wymaga wcześniejszej autoryzacji (autoryzacja raz na całą sesję)

Uruchamiamy klienta z opcją -a. Teraz nie trzeba już jawnie wykonywać polecenia AUTH.

Zawartość każdej zmiennej można wyświetlić w konsoli.

Redis – zmiana nazw komend

- Ciekawa funkcjonalność to zmienna `rename-command` w pliku konfiguracyjnym
  - pozwala zmienić nazwę wybranych poleceń na inne, trudne do zgadnięcia
  - np. `rename-command CONFIG b840fc02d524045429941cc15f59e41cb7be6c52`
  - można też wyłączyć wybrane polecenie (zwykle te newralgiczne) zmieniając nazwę na pusty napis,np.: `rename-command CONFIG ""`

Redis – wybrane polecenia dla zmiennej string

- ping – sprawdza połączenie z serwerem
- set – zapisanie klucza z wartością
- get – odczytanie klucza z wartością
- del – usunięcie klucza
- exists – sprawdzenie, czy klucz istnieje
- dump – pobranie serializowanej wersji wartości dla klucza
- expire, expireat, pexpire, pexpireat – ustawienie czasu wygaśnięcia klucza
- persist – wyłączenie wygasania klucza
- keys – znalezienie wszystkich kluczy pasujących do wzorca
- move – przeniesienie klucza do innej bazy danych
- rename – zmiana wartości klucza
- type – odczytanie typu wartości związanej z kluczem

Redis – wybrane polecenia dla zmiennej hash

- hmset – zapisanie klucza z kolekcją par pole-wartość (hash)
- hmget – odczytanie wartości dla wskazanego pola związanego z kluczem (hash)
- hgetall – odczytanie wszystkich par pole-wartość dla podanego klucza (hash)
- hdel – usunięcie jednej lub wielu par związanych z kluczem (hash)
- hexists – sprawdzenie, czy klucz zawiera podane pole
- hkeys – pobranie wszystkich pól dla podanego klucza
- hvals – pobranie wszystkich wartości dla podanego klucza

Redis – zmienne hash, przykłady

```
127.0.0.1:6379> HMSET AG imie "Artur" nazwisko "Gramacki" department "WIEA"
institute "ISSI"
OK
127.0.0.1:6379> HGETALL AG

1) "imie"
2) "Artur"
3) "nazwisko"
4) "Gramacki"
5) "department"
6) "WIEA"
7) "institute"
8) "ISSI"

127.0.0.1:6379> HGET AG institute
"ISSI"

127.0.0.1:6379> HKEYS AG
1) "imie"
2) "nazwisko"
3) "department"
4) "institute"

127.0.0.1:6379> HVALS AG
1) "Artur"
2) "Gramacki"
3) "WIEA"
4) "ISSI"
```

Redis – wybrane polecenia dla zmiennej set

- sadd – dodanie elementu do zbioru dla podanego klucza
- scard – pobranie liczby elementów zbioru dla podanego klucza
- sdiff – wyznaczenie różnicy zbiorów
- sinter – wyznaczenie części wspólnej zbiorów
- sunion – wyznaczenie sumy zbiorów
- smembers – pobranie wszystkich elementów zbioru dla podanego klucza
- spop – pobranie i usunięcie losowo wybranego elementu zbioru dla podanego klucza
- sismember – sprawdzenie, czy element należy do zbioru
- srem – usunięcie elementu(ów) zbioru dla podanego klucza

Redis – zmienne set, przykłady

```
127.0.0.1:6379> SADD NoSQLSet1 "Redis"
(integer) 1

127.0.0.1:6379> SADD NoSQLSet1 "MongoDB"
(integer) 1

127.0.0.1:6379> SADD NoSQLSet1 "Cassandra"
(integer) 1

127.0.0.1:6379> SADD NoSQLSet1 "Neo4J"
(integer) 1

127.0.0.1:6379> SMEMBERS NoSQLSet1
1) "Neo4J"
2) "Redis"
3) "Cassandra"
4) "MongoDB"

127.0.0.1:6379> SADD NoSQLSet2 "Redis"
(integer) 1

127.0.0.1:6379> SADD NoSQLSet2 "Oracle NoSQL Database"
(integer) 1
```

```
127.0.0.1:6379> SDIFF NoSQLSet1 NoSQLSet2
1) "Neo4J"
2) "Cassandra"
3) "MongoDB"

127.0.0.1:6379> SUNION NoSQLSet1 NoSQLSet2
1) "Cassandra"
2) "MongoDB"
3) "Neo4J"
4) "Redis"
5) "Oracle NoSQL Database"

127.0.0.1:6379> SINTER NoSQLSet1 NoSQLSet2
1) "Redis"

127.0.0.1:6379> SISMEMBER NoSQLSet1 "redis"
(integer) 0

127.0.0.1:6379> SISMEMBER NoSQLSet1 "Redis"
(integer) 1
```

Wielkość liter w danych jest istotna.

Redis – wybrane polecenia dla zmiennej list

- lpush – dodanie elementu na początek listy
- rpush – dodanie elementu na końcu listy
- lpop – pobranie i usunięcie pierwszego elementu listy
- rpop – pobranie i usunięcie ostatniego elementu listy
- lindex – odczyt elementu z podanej pozycji listy
- lrange – odczyt zakresu elementów listy
- lrem – usunięcie elementów listy

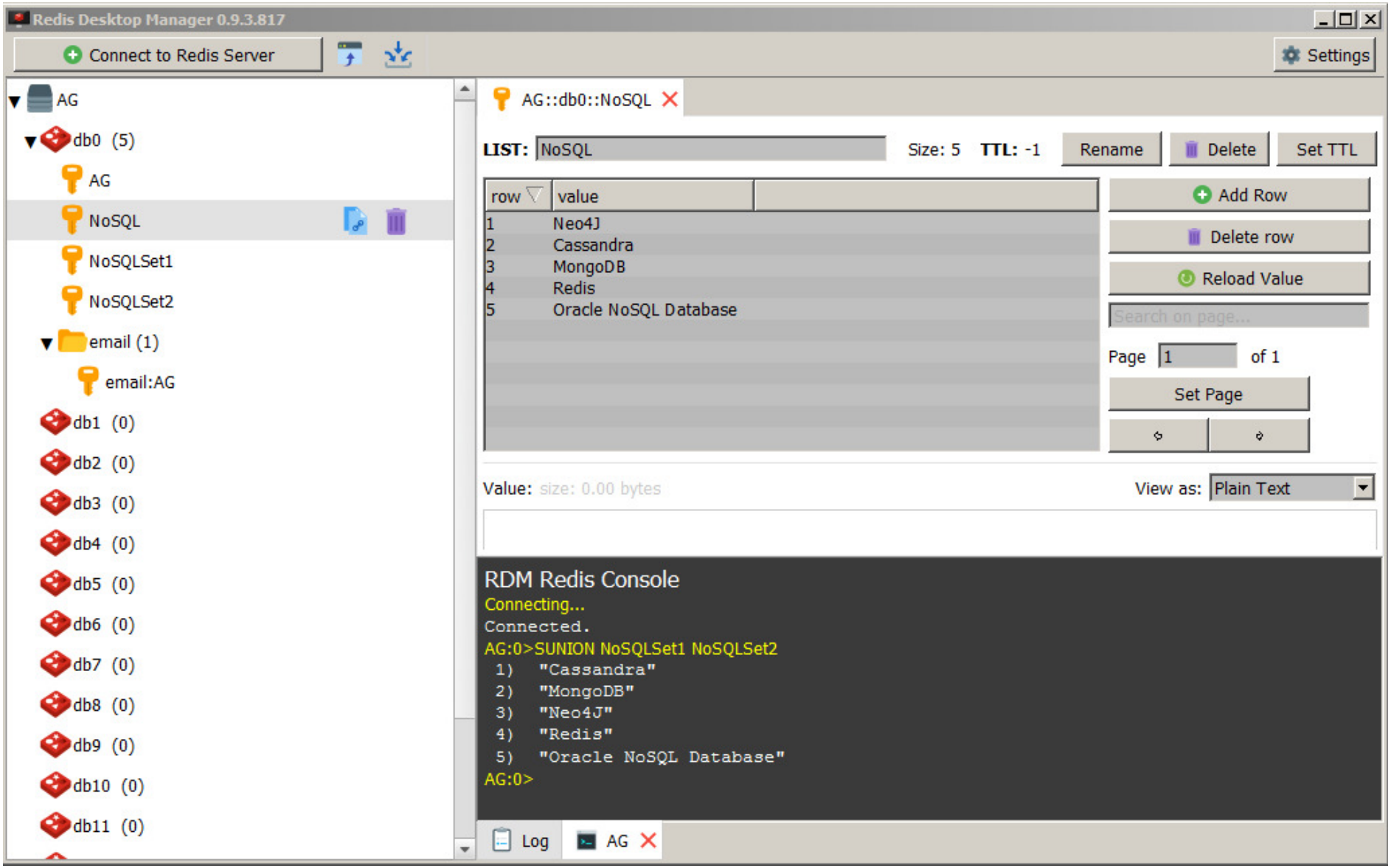
Redis – zmienne list, przykłady

```
127.0.0.1:6379> LPUSH NoSQL "Redis" "MongoDB" "Cassandra" "Neo4J"
(integer) 4

127.0.0.1:6379> LLEN NoSQL
(integer) 4

127.0.0.1:6379> RPUSH NoSQL "Oracle NoSQL Database"
(integer) 5

127.0.0.1:6379> LRANGE NoSQL 0 10
1) "Neo4J"
2) "Cassandra"
3) "MongoDB"
4) "Redis"
5) "Oracle NoSQL Database"
```

Redis Desktop Manager

- Wieloplatformowy (działa na bazie biblioteki Qt)
- Darmowy (open source)
  - na stronie projektu są do pobrania źródła, nie ma kompilacji pod Windows. Można ją jednak pobrać dość łatwo z innych źródeł
- Pozwala na proste zarządzanie bazą Redis
- Szybki, łatwy w użyciu
- Pozwala na podstawowe operacje: przeglądanie kluczy i ich drzew, operacje CRUD na kluczach, uruchamianie poleceń przez wbudowaną konsolę

Redis Desktop Manager



Redis – obsługa transakcji

Redis Transaction Commands

Following table shows some basic commands related to Redis transactions.

Sr.No	Command & Description
1	**DISCARD** Discards all commands issued after MULTI
2	**EXEC** Executes all commands issued after MULTI
3	**MULTI** Marks the start of a transaction block
4	**UNWATCH** Forgets about all watched keys
5	**WATCH key [key ...]** Watches the given keys to determine the execution of the MULTI/EXEC block

Redis – obsługa transakcji

- Transakcje z bazie Redis są rozumiane jak wykonanie grupy poleceń w jednym kroku
  - transakcja jest atomowa, albo wykonane są wszystkie operacje, albo wszystkie są wycofane
- Wspierane są tylko własności izolacji i atomowości transakcji (w praktyce to całkiem sporo)
- Wszystkie polecenia w transakcji są realizowane jako pojedyncza izolowana operacja. Jednoczesna modyfikacja nie jest możliwa przez inną operacje

Redis – obsługa transakcji

- Transakcje rozpoczynamy poleceniem **MULTI**, natomiast poleceniem **EXEC** kończymy. Do momentu pojawienia się polecenia **EXEC** wszystkie operacje dodawane są do kolejki zadań (status (**QUEUED**)
- Polecenie **EXEC** uruchamia wszystkie polecenia z kolejki i kończy transakcję. Jeśli chcielibyśmy wycofać wszystkie polecenia w transakcji (po poleceniu MULTI) możemy użyć komendy **DISCARD**
- W Redis nie ma możliwości wykonania operacji **rollback**, w takim sensie jak rozumieliśmy go z poziomu relacyjnej bazy danych
- Komenda **DISCARD** anuluje wszystkie polecenia wykonane przez transakcje, ale w przypadku gdy pojawi się błąd (awaria) podczas działania transakcji, to nie ma możliwości jej wycofania

Redis – obsługa transakcji

```
127.0.0.1:6379> 127.0.0.1:6379> MULTI
OK

127.0.0.1:6379> 127.0.0.1:6379> SET tutorial "redis"
QUEUED

127.0.0.1:6379> 127.0.0.1:6379> GET tutorial
QUEUED

127.0.0.1:6379> 127.0.0.1:6379> INCR visitors
QUEUED

127.0.0.1:6379> 127.0.0.1:6379> EXEC
1) OK
2) "redis"
3) (integer) 1
```

Redis – RDB, AOF

- Pojawia się tutaj hasło persistence, tutaj rozumiane jako wytrzymałość bazy danych na awarie
- Mamy dostępne 2 mechanizmy: RDB i AOF (Append Only File)
- Oba mechanizmy można włączać i wyłączać
  - RDB jest domyślnie włączony
  - AOF jest domyślnie wyłączony
- RDB polega na zrzucaniu bazy danych z pamięci RAM co pewien czas i co pewną ilość zmian w bazie danych. Domyślna konfiguracja zapisuję bazę danych do pliku dump.rdb
  - co 900 sekund (15 minut), jeżeli była co najmniej jedna zmiana (w konfiguracji to linijka: save 900 1),
  - co 300 sekund (5 minut), jeżeli było co najmniej 10 zmian (save 300 10),
  - co 60 sekund, jeżeli było co najmniej 10000 zmian (save 60 10000).
  - nazwę pliku do którego jest zrzucana baza danych jest zdefiniowana przez zmienną *dbfilename*.
  - aby wyłączyć RDB należy usunąć linijki save z konfiguracji.

Redis – RDB, AOF

- AOF polega na zapisywaniu poleceń write (czyli takich, które zmieniają dane).
  - domyślna konfiguracja nie korzysta z mechanizmu AOF
  - plik stworzony w ten sposób może być wielokrotnie większy od rozmiaru bazy danych
  - nazwa pliku AOF jest zdefiniowana prze zmienną *appendfilename*
  - jest możliwość ograniczania wielkości pliku przez przebudowanie pliku AOF bez przerwania działania Redisa
- Odtworzenie po awarii polega po prostu na skopiowaniu kopii bezpieczeństwa (plik dump.rdb) do katalogu roboczego

Redis – masowy import danych

- Cel: wczytanie dużej ilości danych od użytkownika w minimalnym czasie
- Generujemy plik w formacie zrozumiałym przez Redis

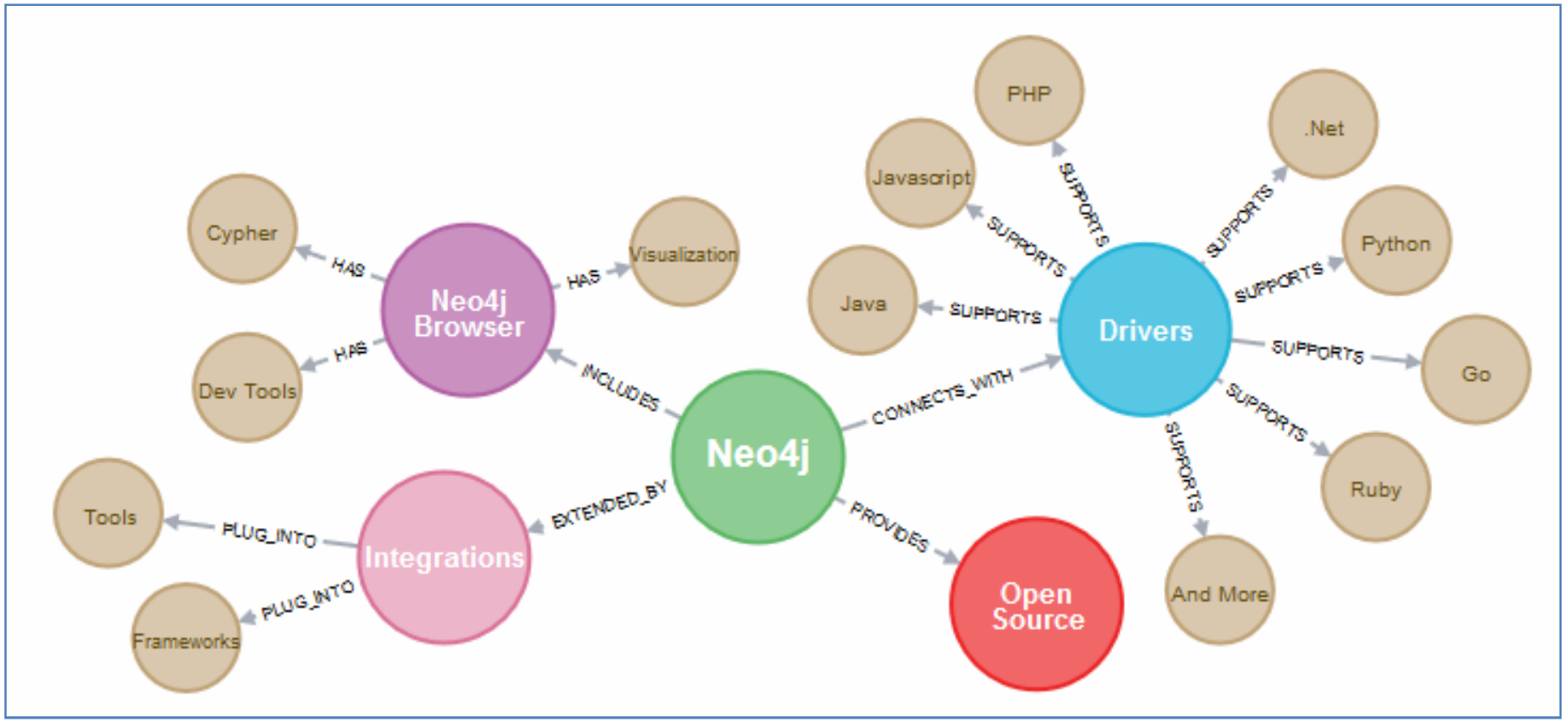
```
SET Key0 Value0
SET Key1 Value1
...
SET KeyN ValueN
```

- W środowisku Linux-a dane te załadujemy wykorzystując przełącznik --pipe klienta redis-cli oraz klasyczne potokowanie
  - `cat data.txt | redis-cli --pipe`
  - po zakończeniu importu otrzymamy komunikat

```
All data transferred. Waiting for the last
reply...
Last reply received from server.
errors: 0, replies: 1000000
```

Redis – tutorial online
http://try.redis.io/

Neo4j


Cypher
HAS
Neo4j Browser
HAS
Visualization
HAS
Dev Tools
INCLUDES
Neo4j
CONNECTS_WITH
Drivers
PHP
SUPPORTS
Javascript
SUPPORTS
Java
SUPPORTS
.Net
SUPPORTS
Python
SUPPORTS
Go
SUPPORTS
Ruby
SUPPORTS
And More
Tools
PLUG_INTO
Integrations
PLUG_INTO
Frameworks
EXTENDED_BY
PROVIDES
Open Source

Neo4j

- https://neo4j.com/
- Repozytorium: github.com/neo4j/neo4j
- Pierwsze wydanie w 2007 roku
- Obecnie stabilna wersja to 4.2.1 (26 listopad 2020)
- Napisana w języku Java, dla innych języków programowania baza jest dostępna poprzez język CQL (ang. *Cypher Query Language*)
- Dostępna na wielu platformach
- Edycje: Community oraz Enterprise
  - Community: może działać tylko na jednym węźle, brak wsparcia dla klastrów. Brak też wsparcia dla „hot backups” i zaawansowanego monitoringu

Neo4j

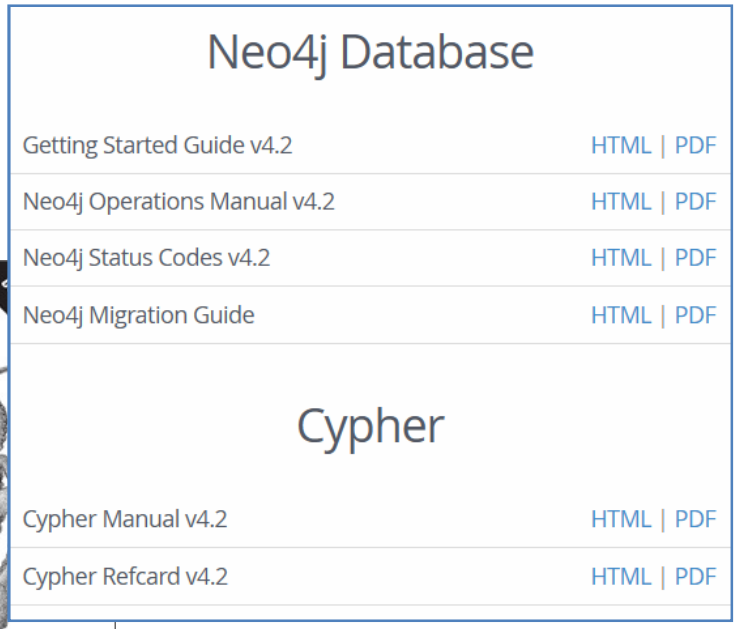
- System zarządzania grafową bazą danych
- Udostępnia transakcje zgodne z ACID
- Natywnie przechowuje i przetwarza dane w formie grafu
- Najbardziej popularna z baz grafowych na rynku według portalu https://db-engines.com
- Ma wszystkie cechy dojrzałej i solidnej bazy danych
- Dla wielu aplikacji Neo4J oferuje zysk wydajności mierzony w rzędach wielkości w porównaniu do baz relacyjnych (dziesiątki, setki, tysiące razy szybsza)
- Dostępna zarówno jako samodzielny serwer jak i komponent do osadzenia

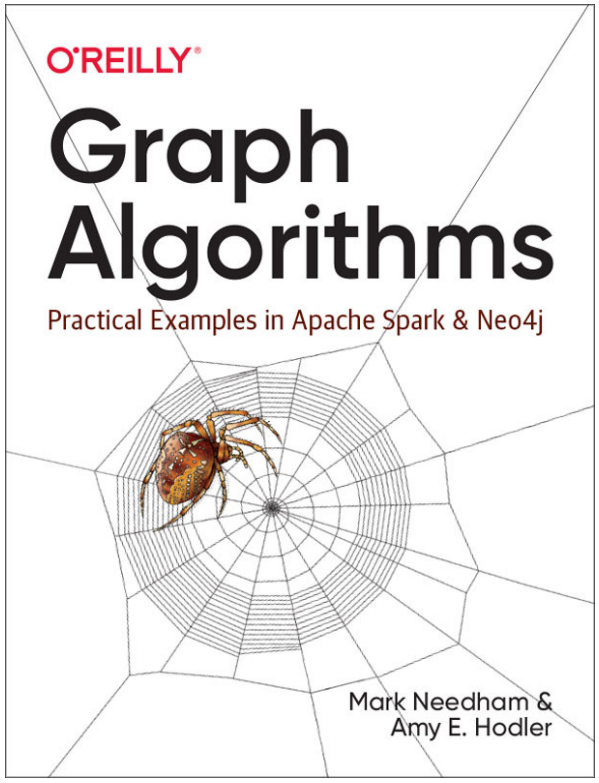
Neo4j, obszary zastosowań

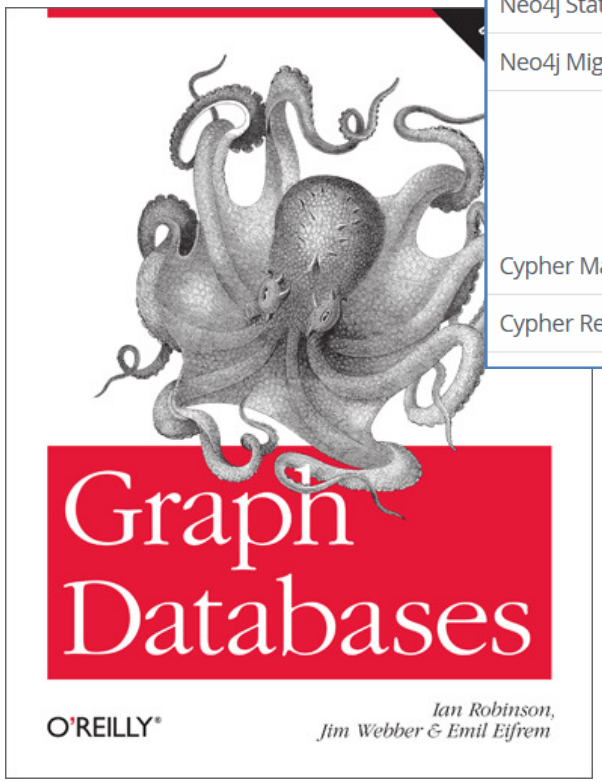
- W globalnym internecie naturalnym jest, że dane są ze sobą połączone (linki, polubienia, opinie itd.)
- Ilość połączeń w zasadzie ciągle się powiększa
  - media społecznościowe, urządzenia przenośne, przetwarzanie w chmurach, „automatyzacja wszystkiego”, internet rzeczy, głębokie sieci neuronowe
- Współczesne bazy relacyjne oraz bazy NoSQL (te agregacyjne) mają dużą trudność w określeniu, w jaki sposób powiązane są elementy danych.
  - w związku z tym trudno też w tych bazach danych tego typu dane analizować i przetwarzać
- Specjalnie zoptymalizowana do mapowania, analizy, przechowywania i przeszukiwania sieci połączonych danych
- Pomaga ujawniać/wykrywać ukryte zależności, niewidoczne konteksty

Neo4j, dokumentacja i darmowe książki

- https://neo4j.com/books/
- https://neo4j.com/docs/





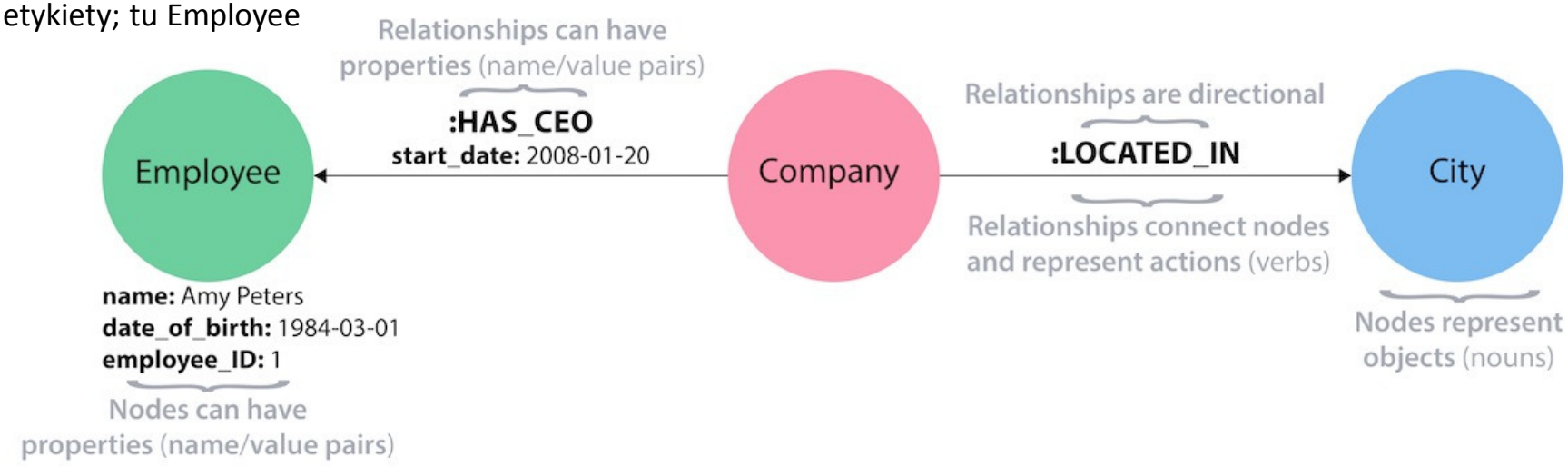


Struktura danych w Neo4j

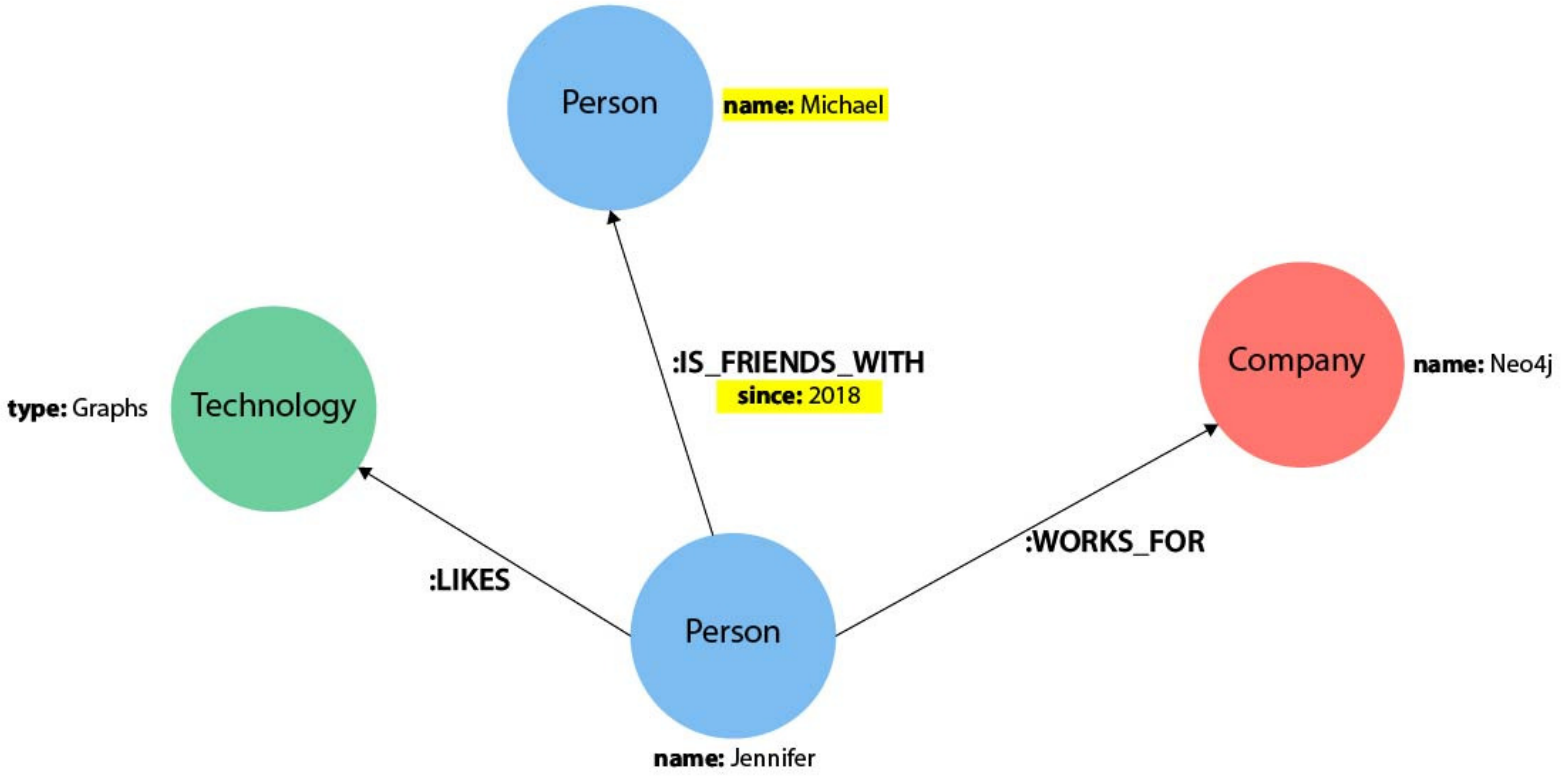
- Wszystkie dane są przechowywane w formie grafu skierowanego z węzłami, i krawędziami (node, edge)
  - czasami zamiast krawędzie używa się określenie relacje (relationships)
  - węzły mogą mieć wiele właściwości (properties) w formie par klucz-wartość
  - węzły mogą być znakowane (tagged) za pomocą etykiet (labels). Etykiety reprezentują różne role węzłów w modelu. Są używane do grupowania węzłów w zbiory
  - krawędzie mają swój kierunek (od węzła A do węzła B)
  - krawędzie też mogą mieć właściwości, zwykle wyrażone ilościowo (np. waga, koszt, odległość, rating, okres czasu, siła związku)

Struktura danych w Neo4j

Węzły mogą mieć etykiety; tu Employee

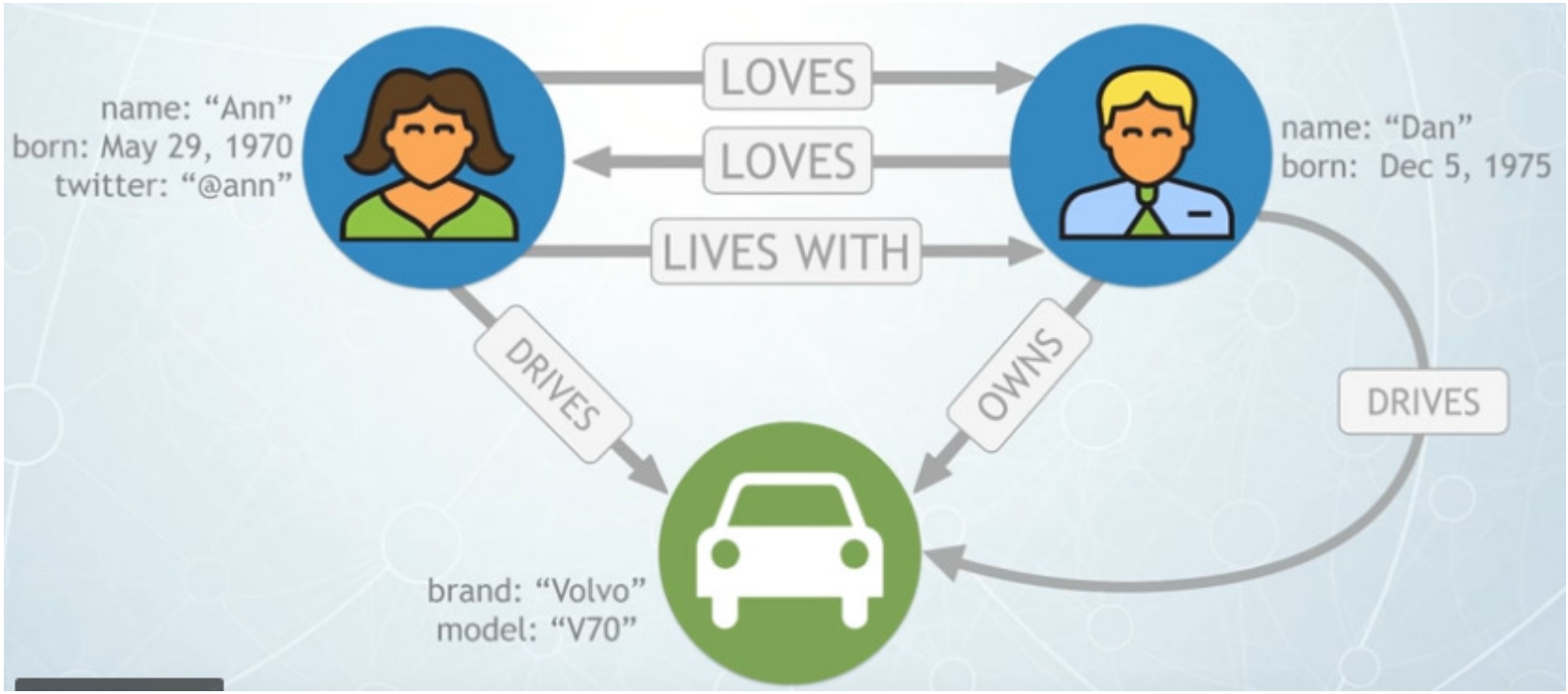


Struktura danych w Neo4j



- Node property: `(p:Person {name: 'Jennifer'})`
- Relationship property: `-[rel:IS_FRIENDS_WITH {since: 2018}]->`

Struktura danych w Neo4j



Na podstawie: https://neo4j.com/developer/cypher-basics-i/

Neo4j, współpraca z innymi językami

- https://neo4j.com/developer/language-guides/

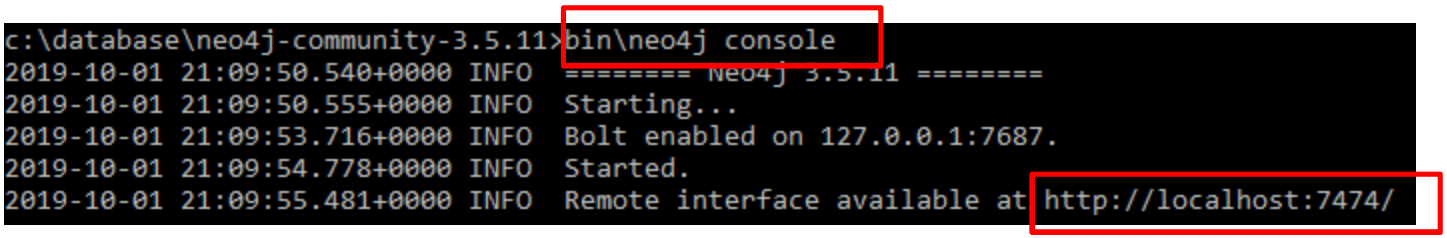
Community-Contributed Drivers

Thanks to the Neo4j contributor community, there are additionally drivers for almost every popular programming language, most of which mimic existing database driver idioms and approaches. See the dedicated pages linked below for more detail.

.NET	Java	Spring	Neo4j-OGM	JavaScript
Python	Ruby	PHP	R	Go
Erlang/Elixir	C/C++	Clojure	Perl	Haskell

Neo4j, instalacja, uruchomienie

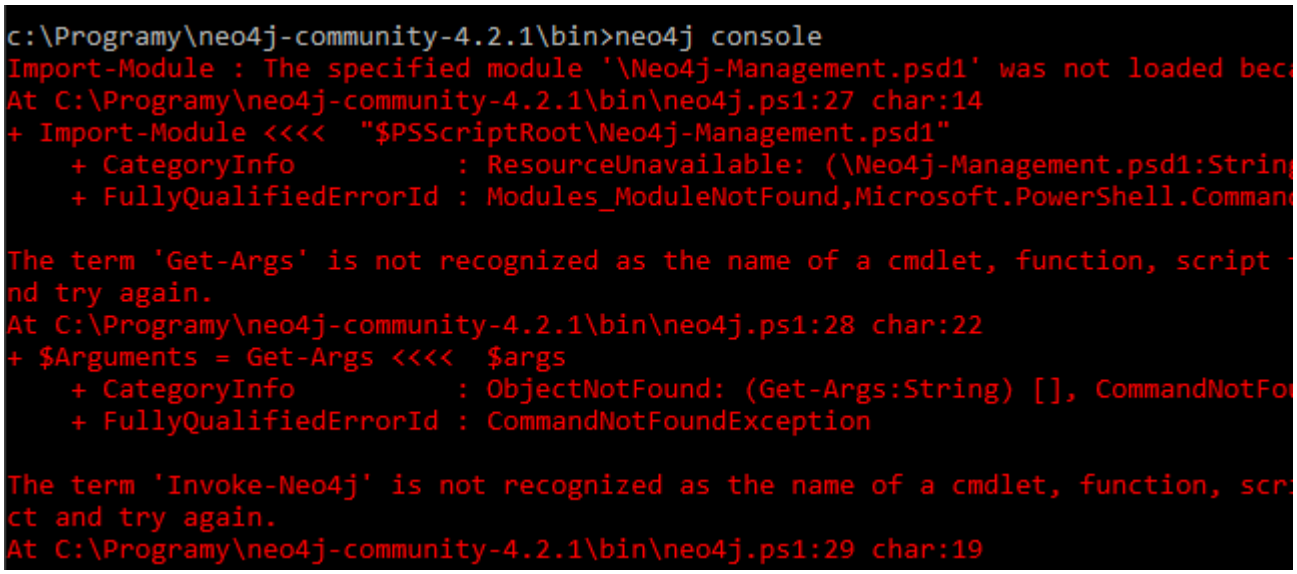
- Pobieramy najnowszą wersję ze strony producenta: https://neo4j.com/download/
- Rozpakowujemy, przechodzimy do utworzonego katalogu (np.: neo4j-community-3.5.11)
- Wystarczy uruchomić poleceniem
  `/bin/neo4j console`



W tym portalu w zasadzie możemy rozpocząć poznawanie neo4j. Nic więcej na początek nie jest konieczne

Neo4j, instalacja, uruchomienie

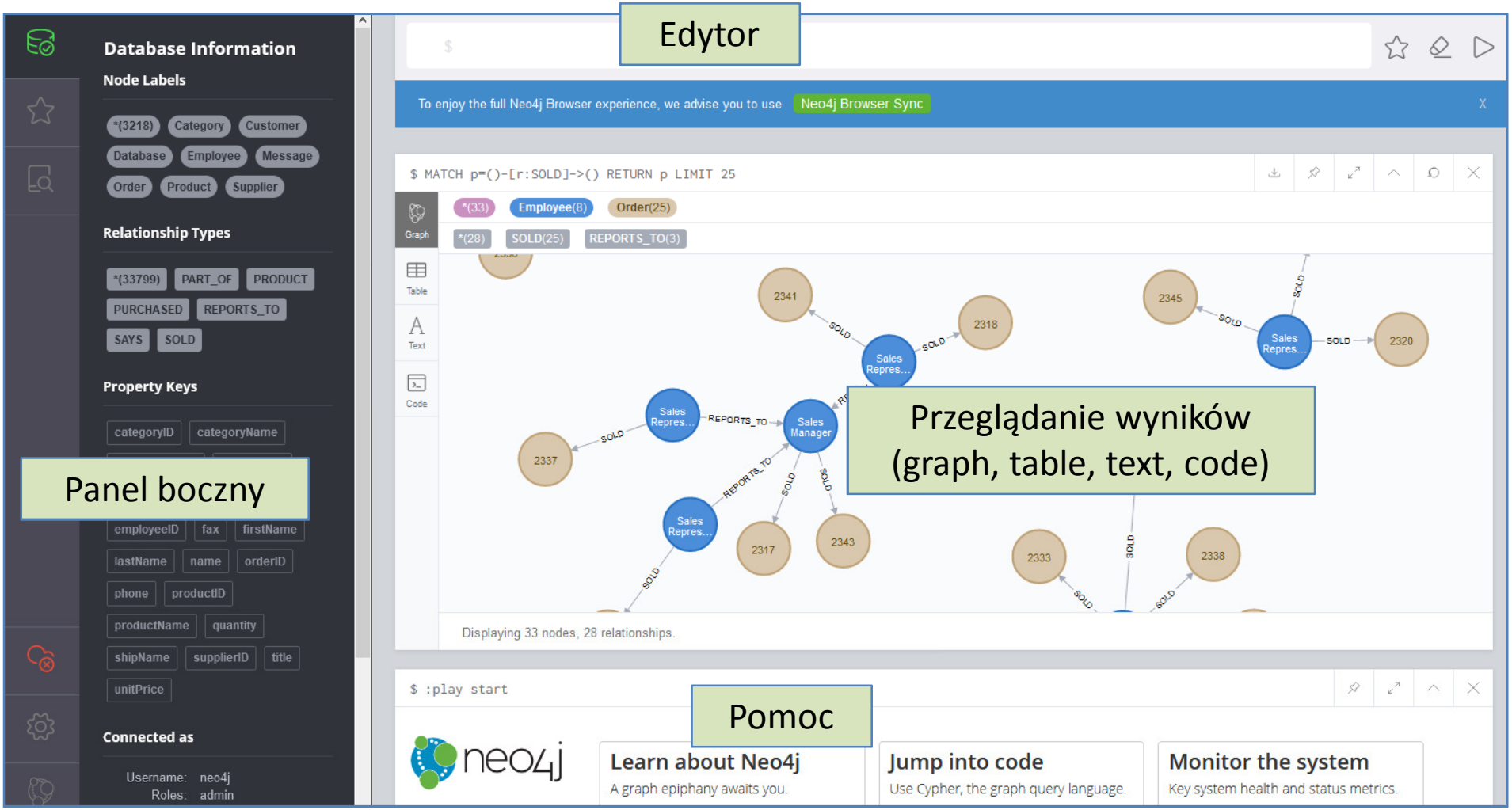
- Gdy pojawiają się błędy związane z PowerShell, patrz szczegółowe instrukcje tutaj https://neo4j.com/docs/operations-manual/current/installation/windows/



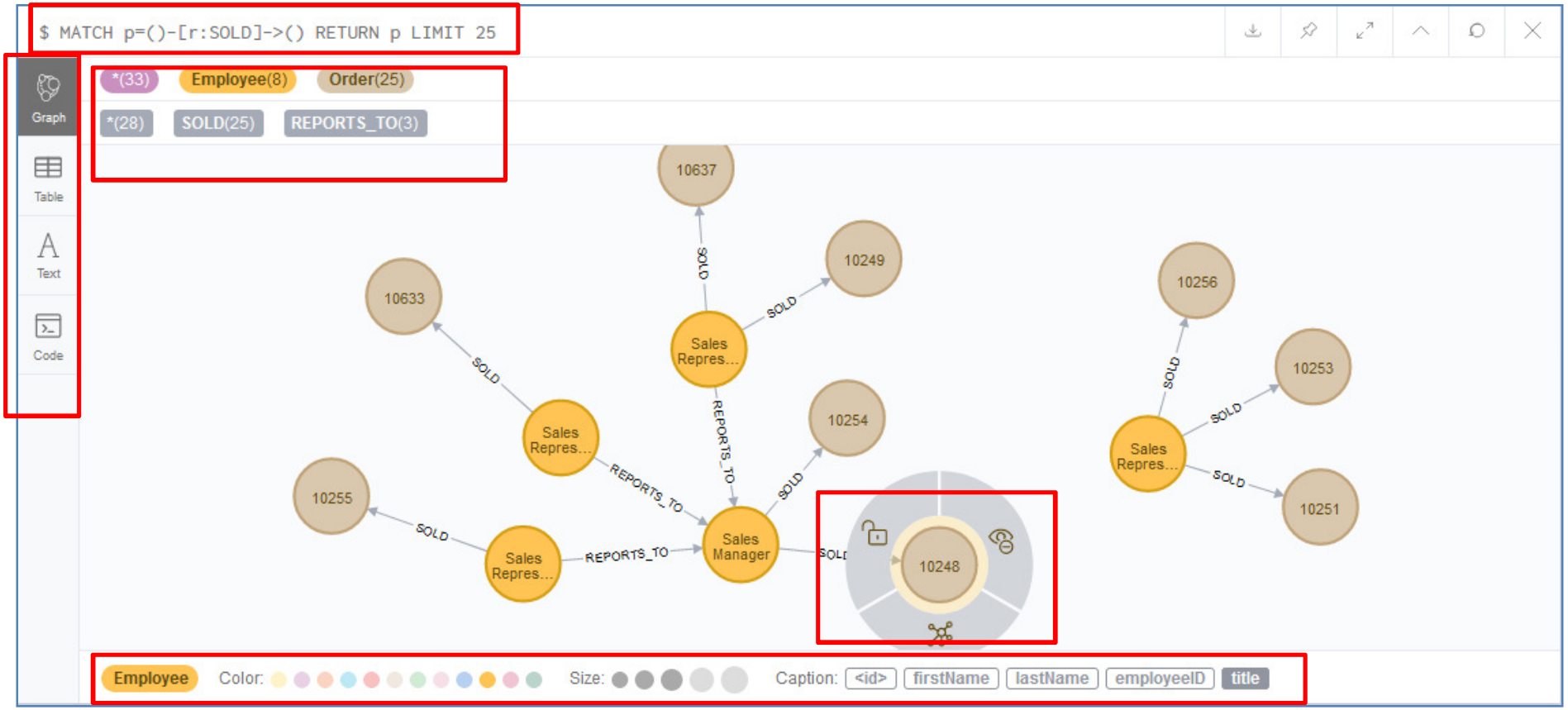
Neo4j, http://localhost:7474/

- Aplikacja webowa pozwalająca na pracę i zarządzanie bazą Neo4J
- Pozwala na wykonywanie zapytań języka CQL
- Pozwala łączyć się z dowolną bazą Neo4J (nie tylko lokalną)
- Zawiera wbudowane tutoriale i rozbudowaną pomoc
  - zwłaszcza tutoriale warto przejrzeć
- Udostępnia m.in. graficzną formę przeglądania wyników, węzłów i ich połączeń
  - raczej tylko do demonstracji, dla dużych baz danych praca z tym narzędziem może być niewygodna
- Jest częściowo konfigurowalna

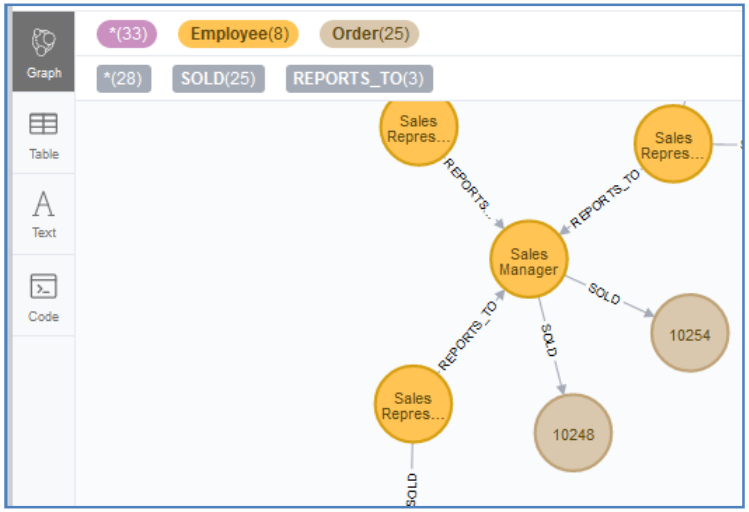
Neo4j, http://localhost:7474/

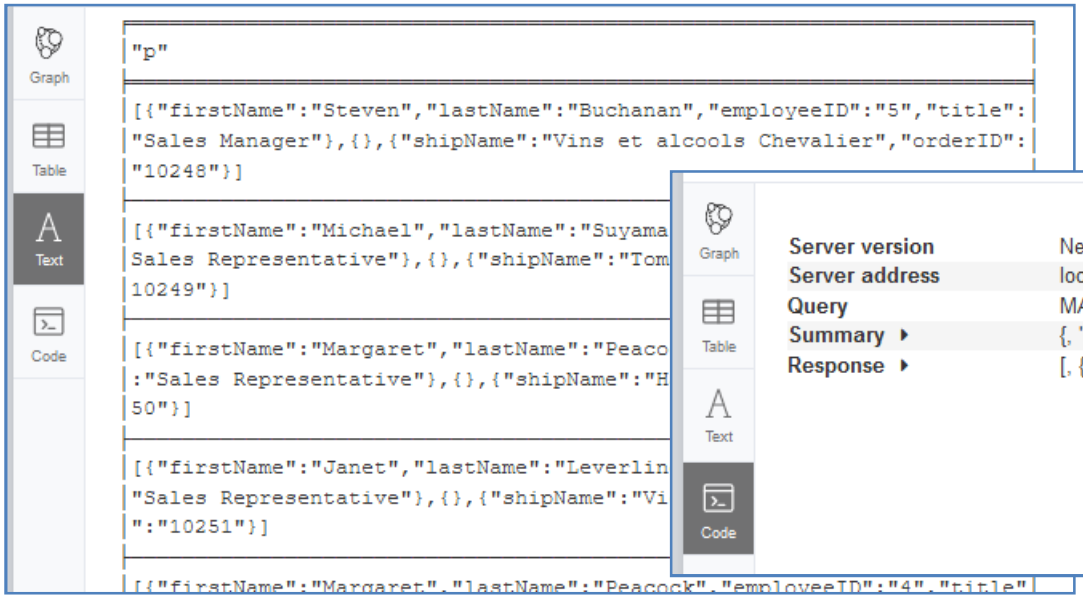


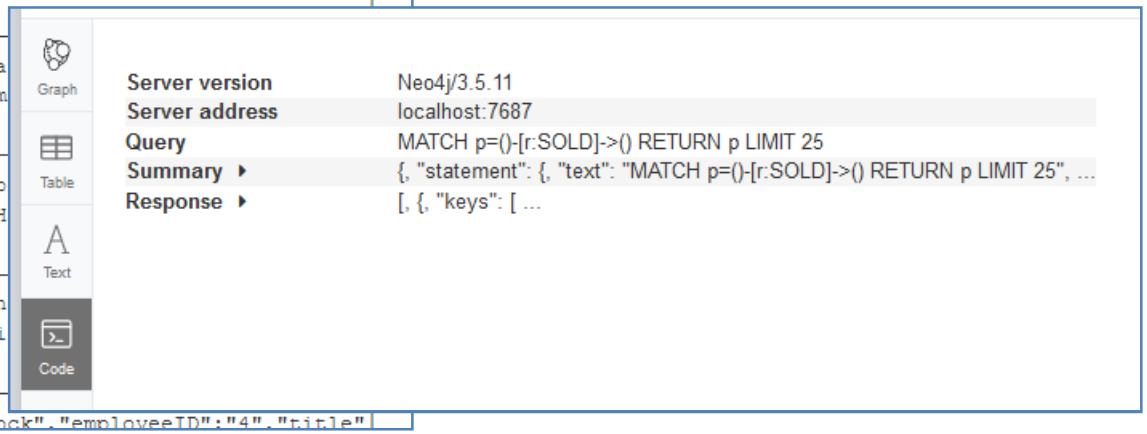
Neo4j, http://localhost:7474/



Neo4j, http://localhost:7474/







Neo4j, http://localhost:7474/

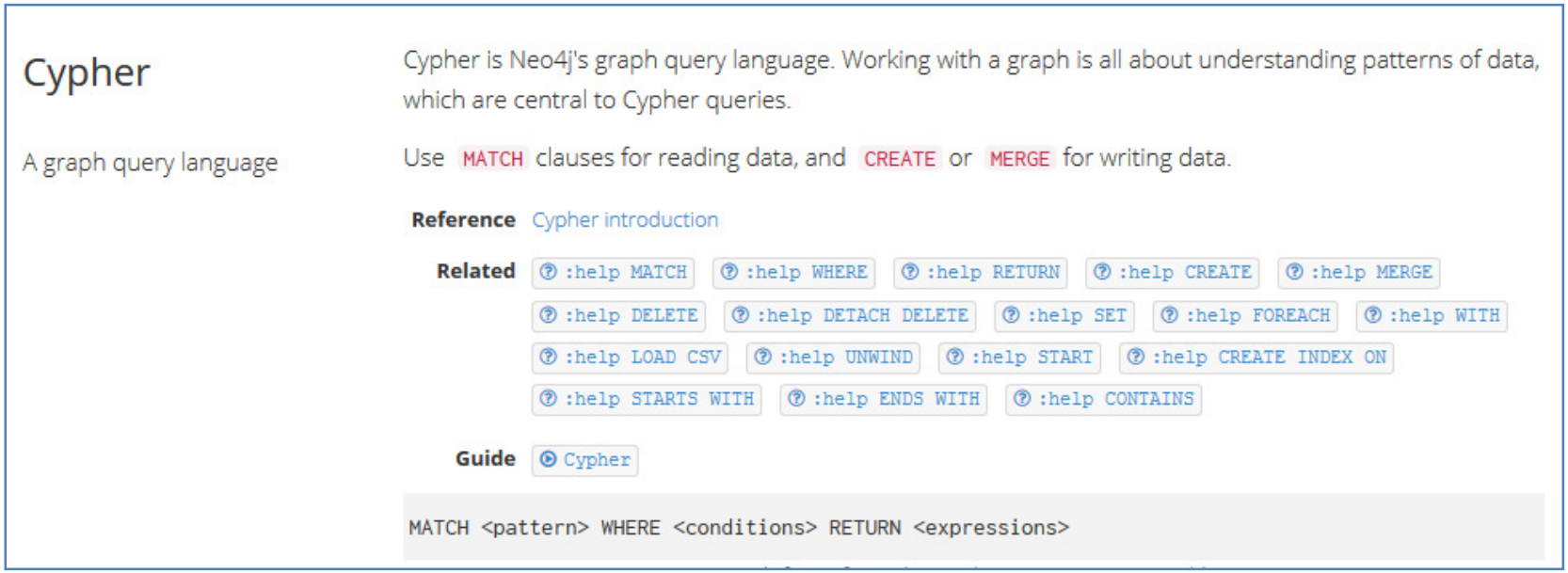
- Polecenia wydajemy w formacie: „dwukropek + polecenie”
  - **`:help`** – podstawowa pomoc uczenia się Neo4J, jest tutaj trochę różnych odnośników
  - **`:help commands`** – pomoc do konkretnie podanego polecenia
  - **`:help play`** – lista wszystkich tutoriali
  - **`:help keys`** – info o skrótach klawiaturowych,
  - **`:help cypher`** – okienko do pomocy nauki języka Cypher
  - **`:help REST GET`**, **`:help REST POST`**, **`:help rest-delete`**, … – pomoc do odsługi konkretnych metod protokołu HTTP
  - **`:GET /db/data/labels`**
  - **`:DELETE /db/data/transaction/2`**
  - **`:help param`**, **`:help params`** – obsługa parametrów wykorzystywanych w zapytaniach
  - **`:help queries`** – info o poleceniu **`:queries`** działającym tylko w Neo4J Enterprise

Neo4j, http://localhost:7474/

- Polecenia systemowe
  - **`:clear`** – usuwa wszystkie okienka z wynikami poleceń, tutorialami, itp.
  - **`:sysinfo`** – informacje o systemie, liczba węzłów, właściwości, relacji, typów relacji
  - **`:help server`** – pomoc do poleceń serwerowych
  - **`:server connect`** – łączenie z serwerem, wyświetla też informacje o połączeniu jeśli takie już jest
  - **`:server disconnect`** – rozłączanie się od serwera
  - **`:server status`** – informacja o połączeniu
  - **`:server change-password`** – zmiana hasła
  - **`:help server user`** – pomoc o poleceniach dla tworzenia użytkowników i wyświetlenia ich listy,
  - **`:server user add`** – dodawanie użytkownika
  - **`:server user list`** – lista utworzonych użytkowników

Neo4j, http://localhost:7474/

- Rodzaje komend



Neo4j, narzędzia

- neo4j-admin
  - główne narzędzie do zarządzania instancją Neo4J
  - dołączone do całego produktu Neo4J
  - użycie: `neo4j-admin polecenie`, dostępne polecenia (patrz następny slajd):

Neo4j, narzędzia

```
environment variables:
    NEO4J_CONF    Path to directory which contains neo4j.conf.
    NEO4J_DEBUG   Set to anything to enable debug output.
    NEO4J_HOME    Neo4j home directory.
    HEAP_SIZE     Set JVM maximum heap size during command execution.
                  Takes a number and a unit, for example 512m.

available commands:

General
    check-consistency
        Check the consistency of a database.
    help
        This help text, or help for the command specified in <command>.
    import
        Import from a collection of CSV files or a pre-3.0 database.
    memrec
        Print Neo4j heap and pagecache memory settings recommendations.
    push-to-cloud
        Push database to Neo4j cloud
    report
        Produces a zip/tar of the most common information needed for remote assessments.
    store-info
        Prints information about a Neo4j database store.

Authentication
    set-default-admin
        Sets the default admin user when no roles are present.
    set-initial-password
        Sets the initial password of the initial admin user ('neo4j').

Offline backup
    dump
        Dump a database into a single-file archive.
    load
        Load a database from an archive created with the dump command.
```

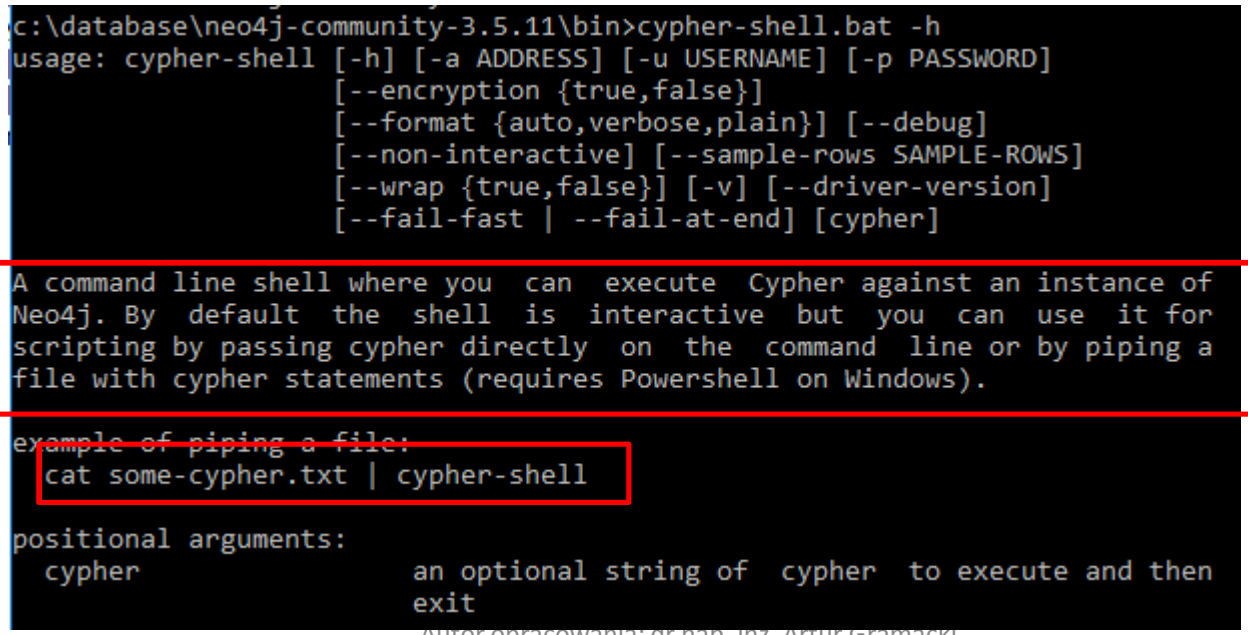
Neo4j, narzędzia

- neo4j-shell
  - narzędzie konsolowe do połączenia się z bazą i wykonania jednego polecenia albo listy poleceń zapisanych w pliku
  - uruchamia lokalnie bazę neo4j i wykonuje na niej polecenie/polecenia
  - komunikuje się z bazą poprzez **szyfrowany binarny protokół Bolt**
  - UWAGA: na chwilę obecną (01.10.2019) narzędzie to nie jest dostępne (deprecated), mimo info na stronie WWW:

> From Neo4j 3.0 access to `neo4j-shell` is no longer possible from the desktop-installers for Windows and OSX.
>
> To use `neo4j-shell`, you have to download the TAR/ZIP distribution from: http://neo4j.com/download/other-releases/

Neo4j, narzędzia

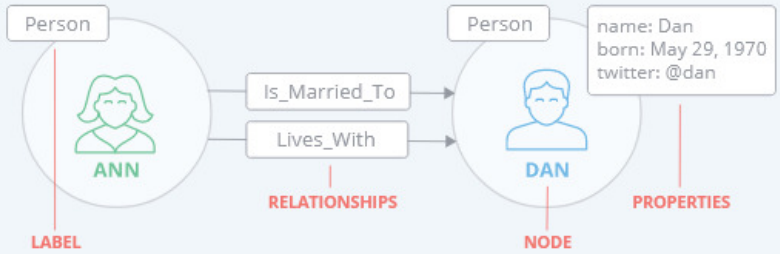
- cypher-shell
  - tekstowa linia poleceń, narzędzie dołączone do całego produktu Neo4J
  - pozwala połączyć się z dowolną bazą Neo4J, użyć języka Cypher do pobrania danych, zdefiniowania schematu czy zadań administracyjnych
  - pozwala na jawne używanie transakcji
  - komunikuje się z bazą poprzez szyfrowany binarny protokół Bolt



https://neo4j.com/product/#neo4j-desktop

Neo4j Basics

The Labeled Property Graph Model



Nodes

- Nodes are the main data elements
- Nodes are connected to other nodes via **relationships**
- Nodes can have one or more **properties** (i.e., attributes stored as key/value pairs)
- Nodes have one or more **labels** that describes its role in the graph
- Example: Person nodes vs Car nodes

Relationships

- Relationships connect two nodes
- Relationships are directional
- **Nodes** can have multiple, even recursive relationships
- Relationships can have one or more **properties** (i.e., attributes stored as key/value pairs)

Properties

- Properties are named values where the name (or key) is a string
- Properties can be indexed and constrained
- Composite indexes can be created from multiple properties

Labels

- Labels are used to group **nodes** into sets
- A node may have multiple labels
- Labels are indexed to accelerate finding nodes in the graph
- Native label indexes are optimized for speed

Neo4j, Cypher

- Deklaratywny język zaprojektowany do obsługi danych grafowych
  - deklaratywny: opisujemy co szukamy w danych a nie jak te dane wyszukać (taki jest też np. język SQL)
- Nieco podobny w swojej istocie do języka SQL
  - zawiera znane z SQL klauzule, słowa kluczowe, wyrażenia, np. WHERE, ORDER BY, SKIP LIMIT, AND, p.unitPrice > 10
  - ale jednak operuje na strukturach grafowych, więc siłą rzeczy musi różnić się od klasycznego SQL-a

Neo4j, Cypher

- Węzły są w nawiasach
  `()` lub `(p)`
- Etykiety (tagi) zaczynają się od znaku dwukropka (:) i grupują węzły podług spełnianych ról, typów itp.
  `(p:Person)`
- Węzły mogą posiadać właściwości
  `(p:Person {name: 'Gramacki'})`

```
()                  //anonymous node (no label or variable) can refer to any node in the database
(p:Person)          //using variable p and label Person
(:Technology)       //no variable, label Technology
(work:Company)      //using variable work and label Company
```

Neo4j, Cypher

- Relacje są otoczone znakami kreski albo nawiasami kwadratowymi

  `-->`

  lub, gdy chcemy jakoś dokładniej określić relację

  `-[h:HIRED]->`

- Kierunek relacji jest oznaczany znakami mniejszości i większości `< >`

  ```
  (p1)-[:HIRED]->(p2)
  ```

  lub

  ```
  (p1)<-[:HIRED]-(p2)
  ```

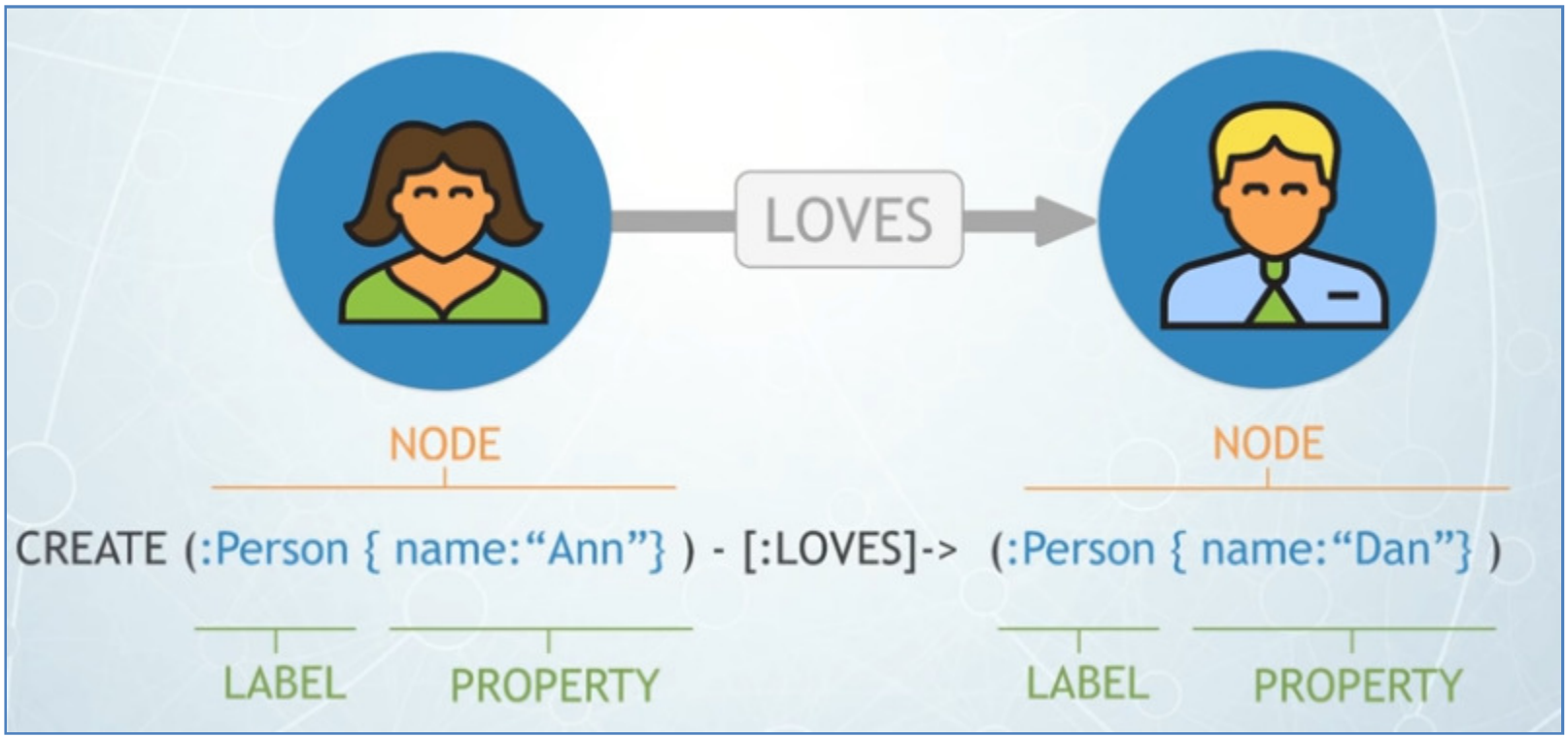
- Relacje też mogą mieć właściwości

  ```
  -[:HIRED {type: 'fulltime'}]->
  ```

Neo4j, Cypher, węzły, relacje, właściwości

- Zapamiętajmy więc:
  - (Nawiasy okrągłe) oznaczają węzły
  - [Nawiasy kwadratowe] określają relacje
  - {Nawiasy klamrowe} mówią nam o właściwościach

Neo4j, Cypher, tworzenie obiektów



Na podstawie: https://neo4j.com/developer/cypher-basics-i/

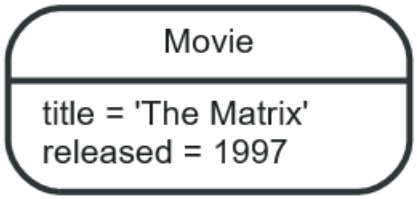
Neo4j, Cypher, wzorce (patterns)

- Cypher jest oparty na wzorcach (patterns)
  - pewien wizualny sposób reprezentowania ścieżek w grafie, którym de facto jest struktura danych w Neo4j
- Węzły i relacje są komponentami, z pomocą których tworzymy wzorce
- Wzorce pozwalają nam modelować zarówno proste, jak i bardzo złożone przejścia
- Przykład

```
(p:Person {name: "Jennifer"})-[rel:LIKES]->(g:Technology {type: "Graphs"})
```

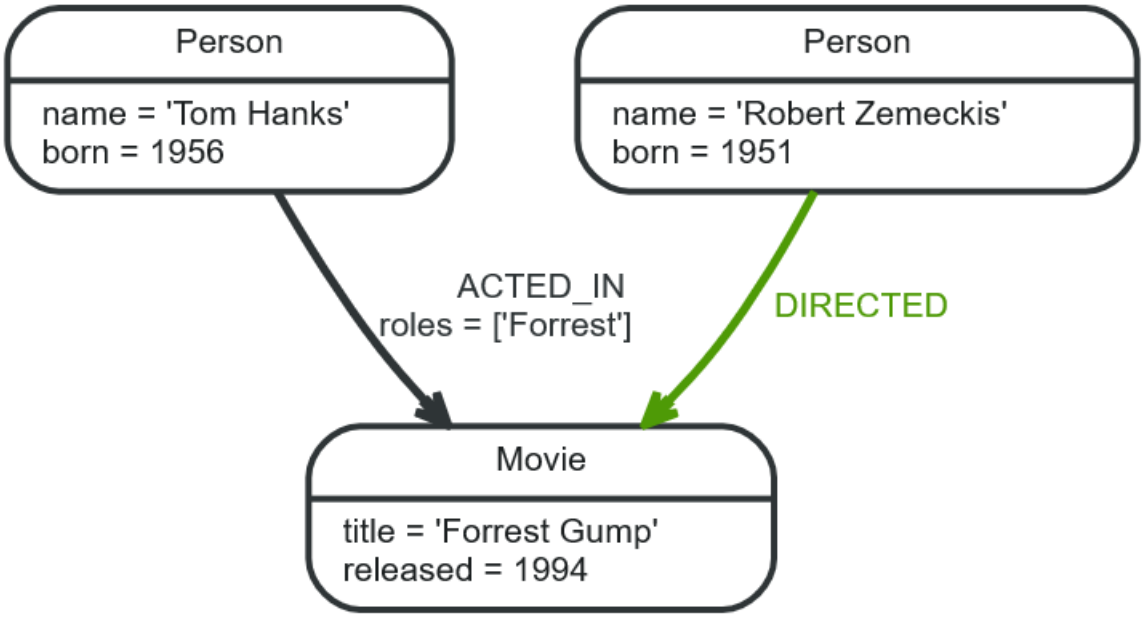
Neo4j, Cypher, wzorce (patterns)

```
CREATE (:Movie { title:"The Matrix",released:1997 })
```



CYPHER

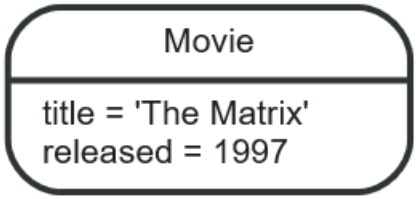
```
CREATE (a:Person { name:"Tom Hanks",
  born:1956 })-[r:ACTED_IN { roles: ["Forrest"]}]->(m:Movie {
title:"Forrest Gump",released:1994 })
CREATE (d:Person { name:"Robert Zemeckis", born:1951 })-[:DIRECTED]->
(m)
RETURN a,d,r,m
```

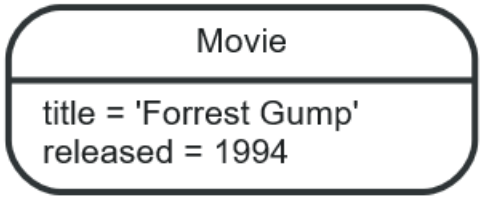


Neo4j, Cypher, zapytania

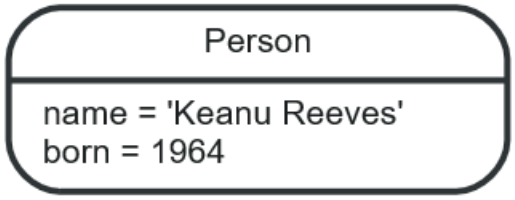
- Polecenie MATCH. Poszukuje wzorców, które specyfikujemy

```
MATCH (m:Movie)
RETURN m
```






```
MATCH (p:Person { name:"Keanu Reeves" })
RETURN p
```




```
MATCH (p:Person { name:"Tom Hanks" })-[r:ACTED_IN]->(m:Movie)
RETURN m.title, r.roles
```

```
+-----------------------------+
| m.title         | r.roles   |
+-----------------------------+
| "Forrest Gump" | ["Forrest"] |
+-----------------------------+
1 row
```

Neo4j, Cypher, zapytania

- Znajduje węzły z etykietą Person
  - Znajdź wszystkie węzły z etykietą Person. Zmienna p jest tutaj wymagana, jeżeli chcemy wynik zwrócić RETURN-em

```
MATCH (p:Person)
RETURN p
```

  - znajdź osobę z etykietą Person o imieniu Jennifer

```
MATCH (jenn:Person {name: 'Jennifer'})
RETURN jenn
```

  - znajdź, gdzie pracuje Jennifer

```
MATCH (:Person {name: 'Jennifer'})-[:WORKS_FOR]->(company:Company)
RETURN company
```

  - znajdź, gdzie pracuje Jennifer ale tym razem zwróć nazwę firmy

```
MATCH (:Person {name: 'Jennifer'})-[:WORKS_FOR]->(company:Company)
RETURN company.name
```

Neo4j, Cypher, aliasy

- Podobnie jak w SQL, AS umożliwia definiowane aliasów

```
//poorly-named property
MATCH (kristen:Customer {name:'Kristen'})-[rel:PURCHASED]-(order:Order)
RETURN order.orderId, order.orderDate, kristen.customerIdNo, order.orderTotalNoOfItems

//cleaner printed results with aliasing
MATCH (kristen:Customer {name:'Kristen'})-[rel:PURCHASED]-(order:Order)
RETURN order.orderId AS OrderID, order.orderDate AS `Purchase Date`,
       kristen.customerIdNo AS CustomerID, order.orderNumOfLineItems AS `Number Of Items`
```

Table | Text | Code

OrderID	Purchase Date	CustomerID	Number Of Items
975310086420	"2018-10-29"	987654	2
864200975310	"2018-10-20"	987654	2
13579024680	"2018-10-03"	987654	1
24680135790	"2018-09-15"	987654	3

Neo4j, Cypher, podobieństwo do SQL

- Jak odnaleźć samochód Anny?

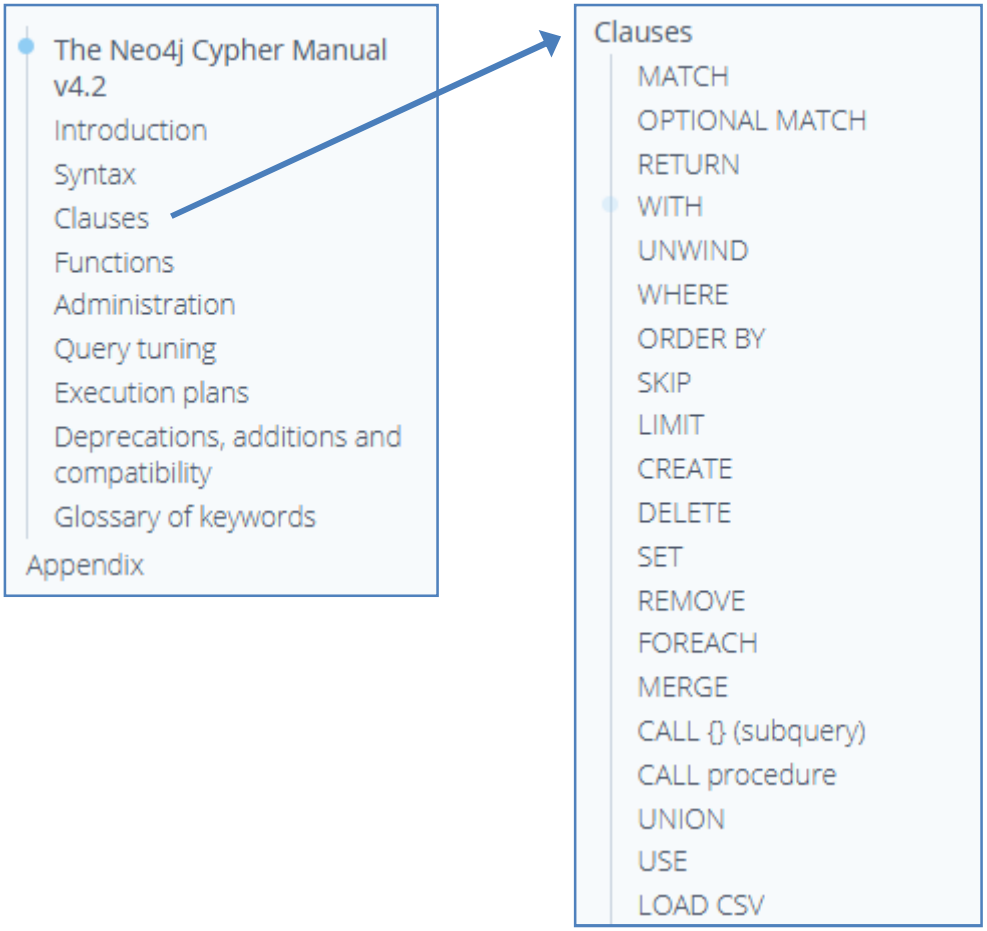
```
MATCH
  (:Person {name: 'Ann'})-[:DRIVES]->(c:Car)
RETURN
  c
```

- To samo ale inaczej zapisane (bardziej zbliżone do SQL-a)

```
MATCH
  (a:Person)-[:DRIVES]->(c:Car)
WHERE
  a.name='Ann'
RETURN
  c
```

Neo4j, Cypher, wszystkie klauzule

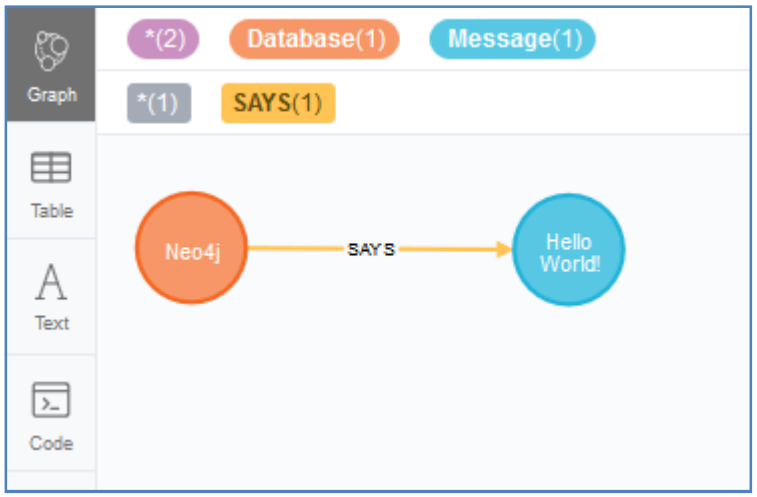
- https://neo4j.com/docs/cypher-manual/current/styleguide/



Neo4j , Cypher, proste przykłady

- Hello World

```
1 // Hello World!
2 CREATE (database:Database {name:"Neo4j"})-[r:SAYS]->(message:Message
  {name:"Hello World!"}) RETURN database, message, r
```



Neo4j, Cypher, proste przykłady

- Tworzenie węzła (`:play cypher`)

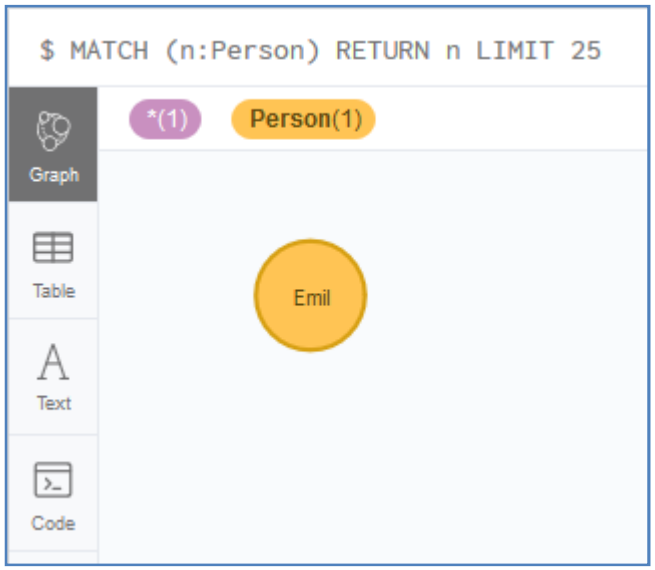
CREATE

Create a node

Let's use Cypher to generate a small social graph.

```
CREATE (ee:Person { name: "Emil", from: "Sweden", klout: 99 })
```

- `CREATE` clause to create data
- `()` parenthesis to indicate a node
- `ee:Person` a variable 'ee' and label 'Person' for the new node
- `{}` brackets to add properties to the node



Neo4j, Cypher, proste przykłady

- Znajdowanie węzłów

MATCH

Finding nodes

Now find the node representing Emil:

```
MATCH (ee:Person) WHERE ee.name = "Emil" RETURN ee;
```

- `MATCH` clause to specify a pattern of nodes and relationships
- `(ee:Person)` a single node pattern with label 'Person' which will assign matches to the variable 'ee'
- `WHERE` clause to constrain the results
- `ee.name = "Emil"` compares name property to the value "Emil"
- `RETURN` clause used to request particular results

Neo4j, Cypher, proste przykłady

- Więcej węzłów

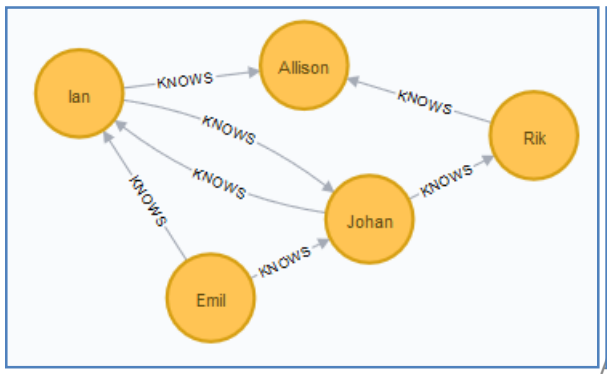
Added 4 labels, created 4 nodes, set 14 properties, created 7 relationships, completed after 302 ms.

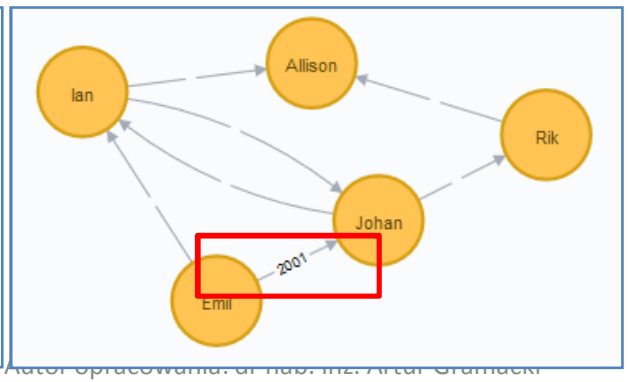
CREATE more

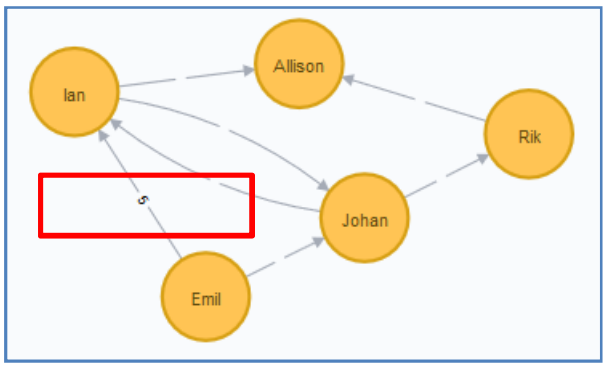
Nodes and relationships

`CREATE` clauses can create many nodes and relationships at once.

```
MATCH (ee:Person) WHERE ee.name = "Emil"
CREATE (js:Person { name: "Johan", from: "Sweden", learn: "surfing"
}),
(ir:Person { name: "Ian", from: "England", title: "author" }),
(rvb:Person { name: "Rik", from: "Belgium", pet: "Orval" }),
(ally:Person { name: "Allison", from: "California", hobby: "surfing"
}),
(ee)-[:KNOWS {since: 2001}]->(js),(ee)-[:KNOWS {rating: 5}]->(ir),
(js)-[:KNOWS]->(ir),(js)-[:KNOWS]->(rvb),
(ir)-[:KNOWS]->(js),(ir)-[:KNOWS]->(ally),
(rvb)-[:KNOWS]->(ally)
```







Neo4j, Cypher, proste przykłady

- Dopasowywanie do wzorców
  - odpowiednik klauzuli WHERE w języku SQL

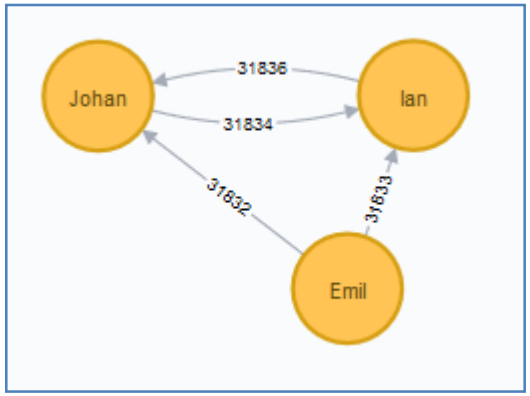
Pattern matching

Describe what to find in the graph

For instance, a pattern can be used to find Emil's friends:

```
MATCH (ee:Person)-[:KNOWS]-(friends)
WHERE ee.name = "Emil" RETURN ee, friends
```

- `MATCH` clause to describe the pattern from known Nodes to found Nodes
- `(ee)` starts the pattern with a Person (qualified by WHERE)
- `-[:KNOWS]-` matches "KNOWS" relationships (in either direction)
- `(friends)` will be bound to Emil's friends



Neo4j, Cypher, proste przykłady

- Wstawianie rekomendacji

Recommend

Using patterns

Pattern matching can be used to make recommendations. Johan is learning to surf, so he may want to find a new friend who already does:

```
MATCH (js:Person)-[:KNOWS]-()-[:KNOWS]-(surfer)
WHERE js.name = "Johan" AND surfer.hobby = "surfing"
RETURN DISTINCT surfer
```

- `()` empty parenthesis to ignore these nodes
- `DISTINCT` because more than one path will match the pattern
- `surfer` will contain Allison, a friend of a friend who surfs

surfer

```
{
  "name": "Allison",
  "from": "California",
  "hobby": "surfing"
}
```

Neo4j, Cypher, proste przykła

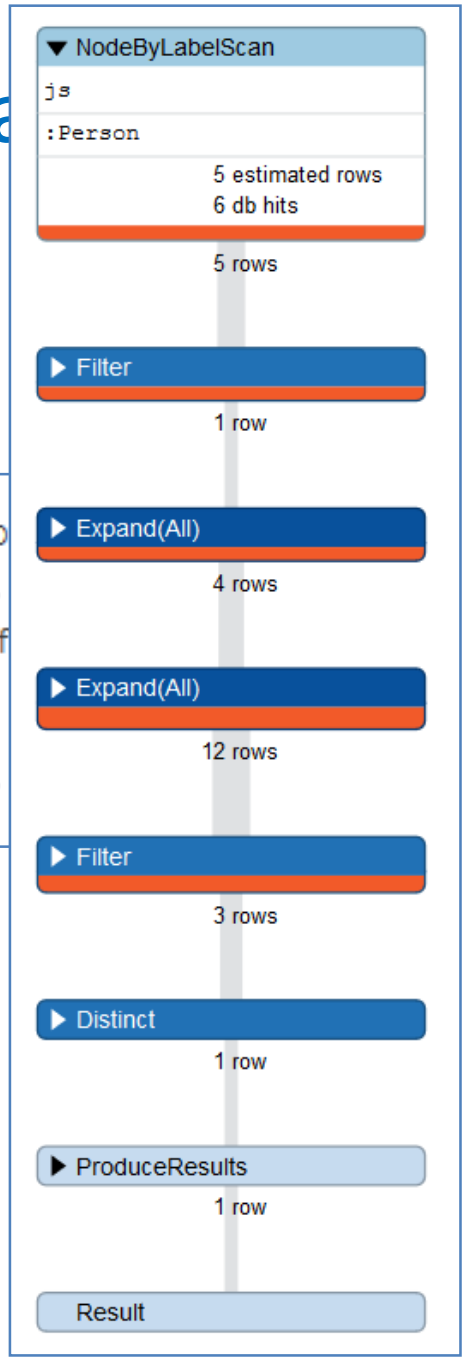
- Analizy
  - coś jak plan wykonania (execution plan) w tradycyjnych bazach relacyjnych

Analyze

Using the visual query plan

Understand how your query works by prepending `EXPLAIN` o

```
PROFILE MATCH (js:Person)-[:KNOWS]-()-[:KNOWS]-(surf
WHERE js.name = "Johan" AND surfer.hobby = "surfing"
RETURN DISTINCT surfer
```



Neo4J, przykład *Movie Graph*

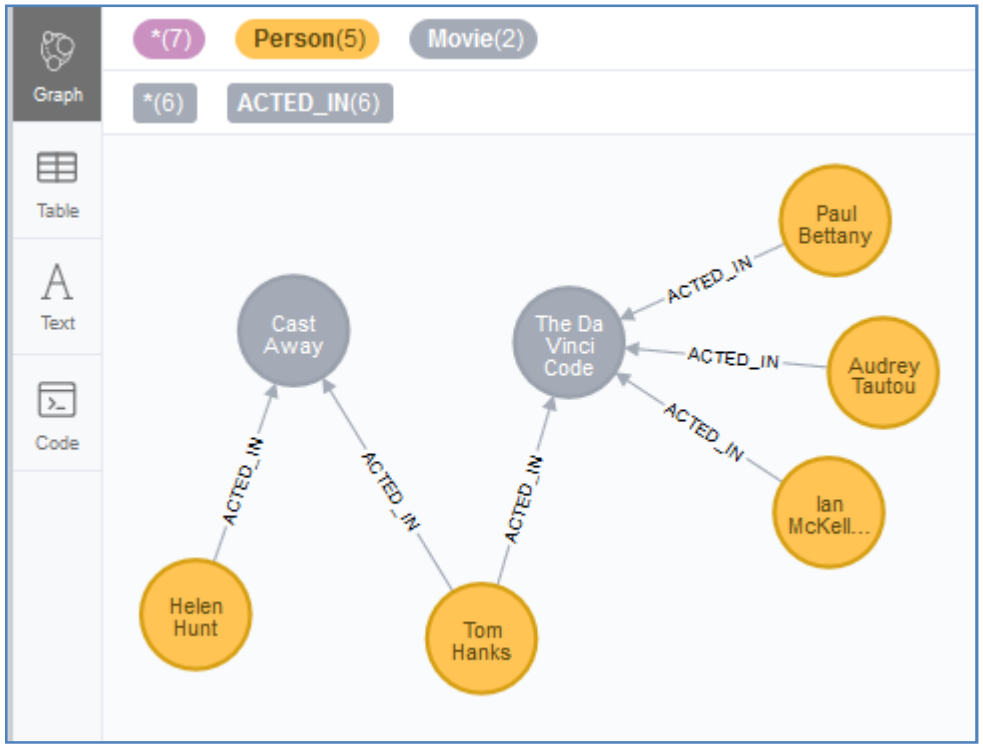
- **`:play movie-graph`**
- *The Movie Graph* is a mini graph application containing actors and directors that are related through the movies they've collaborated on
  - Create: insert movie data into the graph
  - Find: retrieve individual movies and actors
  - Query: discover related actors and directors
  - Solve: the Bacon Path
- Poniżej tylko fragmenty kodów Create oraz Query, całość do uruchomienia w systemie **`:play`**

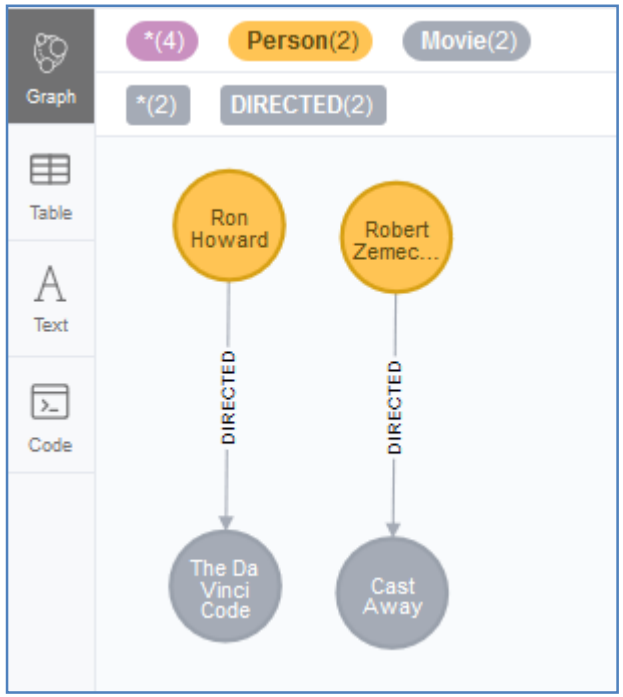
Neo4J, przykład *Movie Graph*

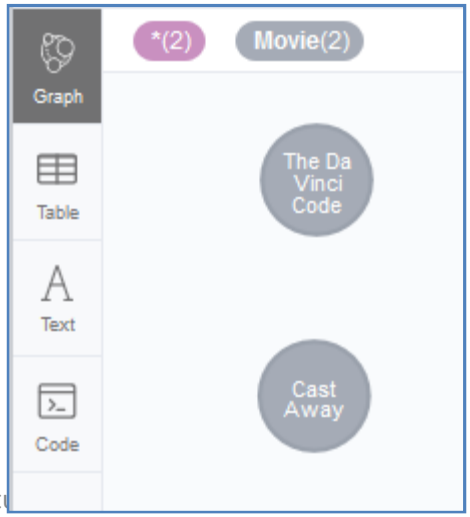
```
CREATE (TomH:Person {name:'Tom Hanks', born:1956})
CREATE (HelenH:Person {name:'Helen Hunt', born:1963})

CREATE (CastAway:Movie {title:'Cast Away',
                        released:2000,
                        tagline:'At the edge of the world, his journey begins.'})

CREATE (RobertZ:Person {name:'Robert Zemeckis', born:1951})
CREATE
  (TomH)-[:ACTED_IN {roles:['Chuck Noland']}]->(CastAway),
  (HelenH)-[:ACTED_IN {roles:['Kelly Frears']}]->(CastAway),
  (RobertZ)-[:DIRECTED]->(CastAway)

CREATE (TheDaVinciCode:Movie {title:'The Da Vinci Code',
                              released:2006,
                              tagline:'Break The Codes'})

CREATE (IanM:Person {name:'Ian McKellen', born:1939})
CREATE (AudreyT:Person {name:'Audrey Tautou', born:1976})
CREATE (PaulB:Person {name:'Paul Bettany', born:1971})
CREATE (RonH:Person {name:'Ron Howard', born:1954})
CREATE
  (TomH)-[:ACTED_IN {roles:['Dr. Robert Langdon']}]->(TheDaVinciCode),
  (IanM)-[:ACTED_IN {roles:['Sir Leight Teabing']}]->(TheDaVinciCode),
  (AudreyT)-[:ACTED_IN {roles:['Sophie Neveu']}]->(TheDaVinciCode),
  (PaulB)-[:ACTED_IN {roles:['Silas']}]->(TheDaVinciCode),
  (RonH)-[:DIRECTED]->(TheDaVinciCode)
```

Added 9 labels,
created 7 nodes,
set 26 properties,
created 8 relationships

Neo4J, przykład *Movie Graph*

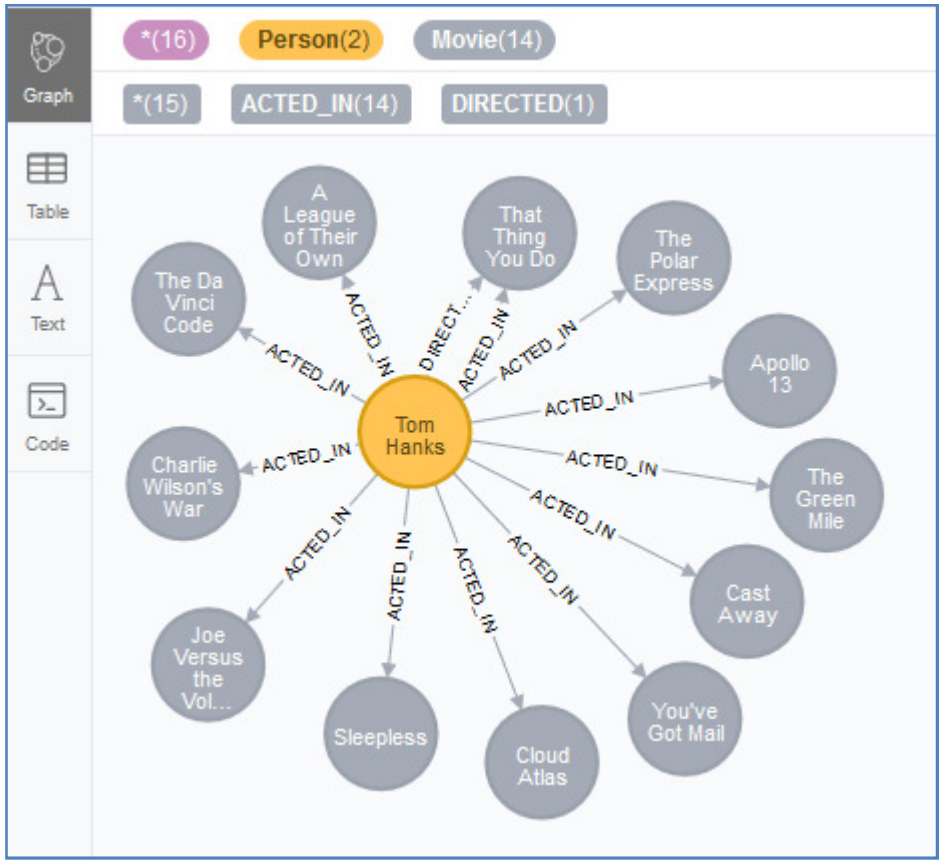






Neo4J, przykład *Movie Graph*

```
// List all Tom Hanks movies
MATCH (tom:Person {name: "Tom Hanks"})-[:ACTED_IN]->(tomHanksMovies)
RETURN tom,tomHanksMovies
```



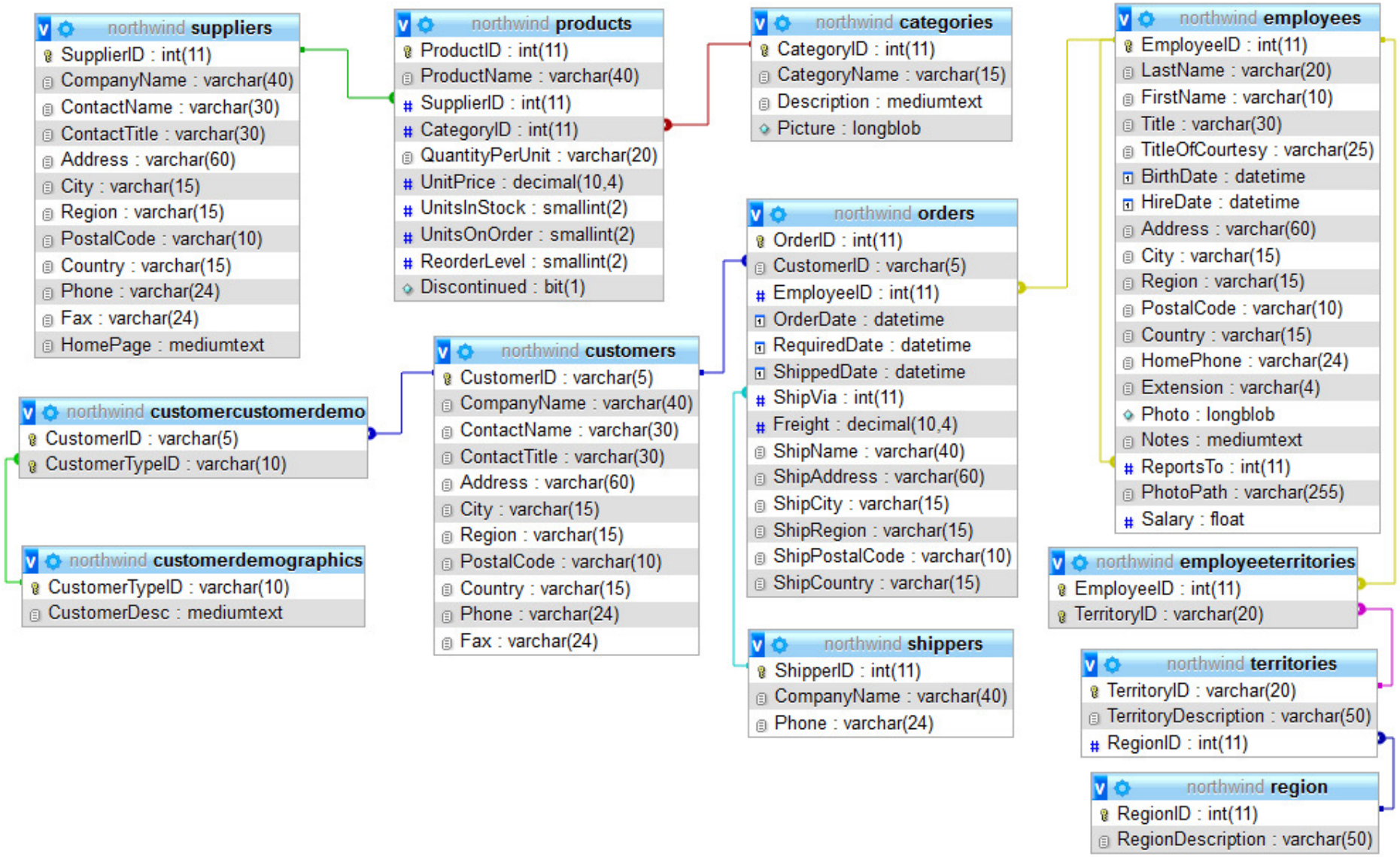
Neo4j, większy przykład

- Migracja bazy Northwind z bazy relacyjnej do Neo4j
  - https://code.google.com/archive/p/northwindextended/downloads

northwindextended

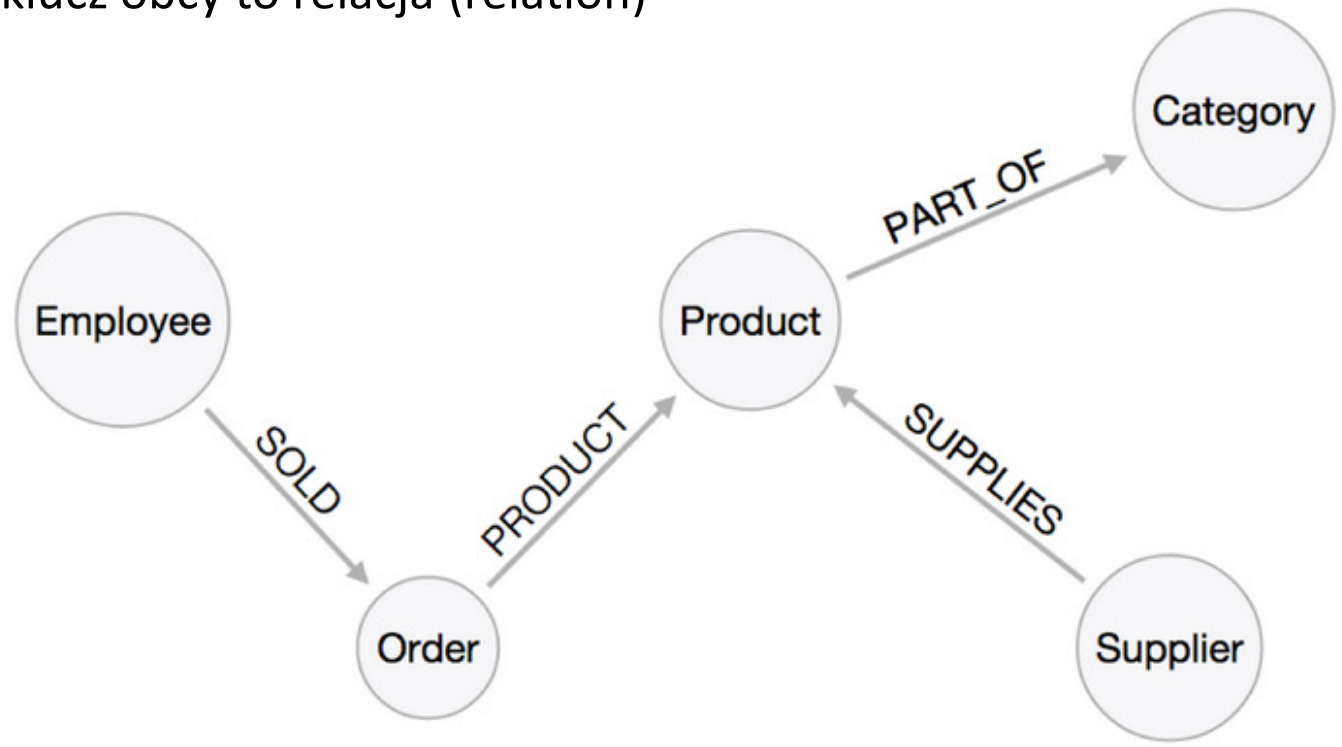
File	Summary + Labels	Uploaded	Size
Northwind.Sqlite3.sql	Sqlite 3 - Sample Featured Type-Source OpSys-All	Nov 21, 2013	886.28KB
Oracle NorthwindDB.sql	Oracle version of Northwind Type-Source OpSys-All Featured	Jun 8, 2012	902.43KB
northwind.xml.7z	Northwind in XML with Schema Type-Source OpSys-All	Jun 8, 2012	189.95KB
northwind.postgre.sql	Postgres SQL Type-Source OpSys-All Featured	Apr 12, 2012	340.75KB
Northwind.png	database schema - Northwind Featured Type-Docs OpSys-All	Jul 9, 2010	79.08KB
Northwind.VistaDB3.sql	VistaDB Featured Type-Source OpSys-All	Jul 9, 2010	1.21MB
Northwind.MySQL5.sql	MySQL 5.x Sample Featured Type-Source OpSys-All	Jul 9, 2010	794.88KB
Northwind.Ms.SQL.2005.sql	Northwind sample for MS SQL 2005+ Featured Type-Source OpSys-Windows	Jul 9, 2010	1.01MB

Neo4j, większy przykład



Neo4j, większy przykład

- Podstawowe zasady konwersji
  - wiersz staje się węzłem (node)
  - nazwa tabeli to etykieta (label)
  - klucz obcy to relacja (relation)

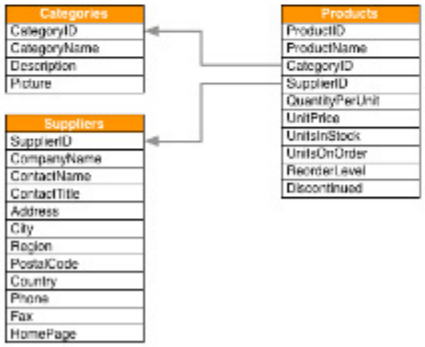


Neo4j, większy przykład

Product Catalog

Northwind sells food products in a few categories, provided by suppliers. Let's start by loading the product catalog tables.

The load statements to the right require public internet access. `LOAD CSV` will retrieve a CSV file from a valid URL, applying a Cypher statement to each row using a named map (here we're using the name `row`).



Load records

```
LOAD CSV WITH HEADERS FROM "http://data.neo4j.com/northwind/products.csv" AS row
CREATE (n:Product)
SET n = row,
  n.unitPrice = toFloat(row.unitPrice),
  n.unitsInStock = toInteger(row.unitsInStock), n.unitsOnOrder = toInteger(row.unitsOnOrder),
  n.reorderLevel = toInteger(row.reorderLevel), n.discontinued = (row.discontinued <> "0")
```

```
LOAD CSV WITH HEADERS FROM "http://data.neo4j.com/northwind/categories.csv" AS row
CREATE (n:Category)
SET n = row
```

```
LOAD CSV WITH HEADERS FROM "http://data.neo4j.com/northwind/suppliers.csv" AS row
CREATE (n:Supplier)
SET n = row
```

Create indexes

```
CREATE INDEX ON :Product(productID)
```

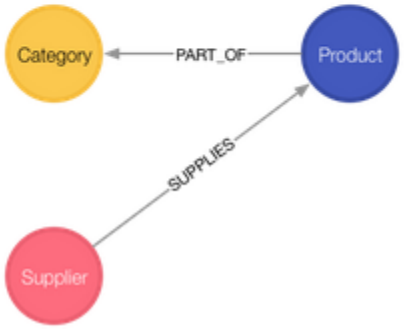
```
CREATE INDEX ON :Category(categoryID)
```

```
CREATE INDEX ON :Supplier(supplierID)
```

Neo4j, większy przykład

Product Catalog Graph

The products, categories and suppliers are related through foreign key references. Let's promote those to data relationships to realize the graph.



Create data relationships

```
MATCH (p:Product),(c:Category)
WHERE p.categoryID = c.categoryID
CREATE (p)-[:PART_OF]->(c)
```

Note you only need to compare property values like this when first creating relationships

Calculate join, materialize relationship. (See importing guide for more details)

```
MATCH (p:Product),(s:Supplier)
WHERE p.supplierID = s.supplierID
CREATE (s)-[:SUPPLIES]->(p)
```

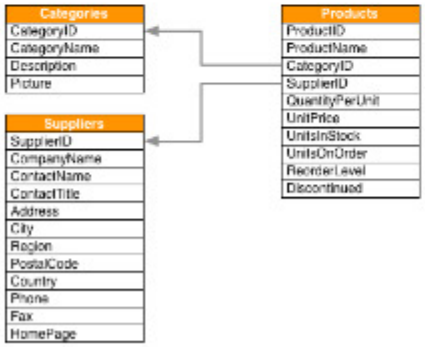
Note you only need to compare property values like this when first creating relationships

Neo4j, większy przykład

Product Catalog

Northwind sells food products in a few categories, provided by suppliers. Let's start by loading the product catalog tables.

The load statements to the right require public internet access. `LOAD CSV` will retrieve a CSV file from a valid URL, applying a Cypher statement to each row using a named map (here we're using the name `row`).



Load records

```
LOAD CSV WITH HEADERS FROM "http://data.neo4j.com/northwind/products.csv" AS row
CREATE (n:Product)
SET n = row,
  n.unitPrice = toFloat(row.unitPrice),
  n.unitsInStock = toInteger(row.unitsInStock), n.unitsOnOrder = toInteger(row.unitsOnOrder),
  n.reorderLevel = toInteger(row.reorderLevel), n.discontinued = (row.discontinued <> "0")
```

```
LOAD CSV WITH HEADERS FROM "http://data.neo4j.com/northwind/categories.csv" AS row
CREATE (n:Category)
SET n = row
```

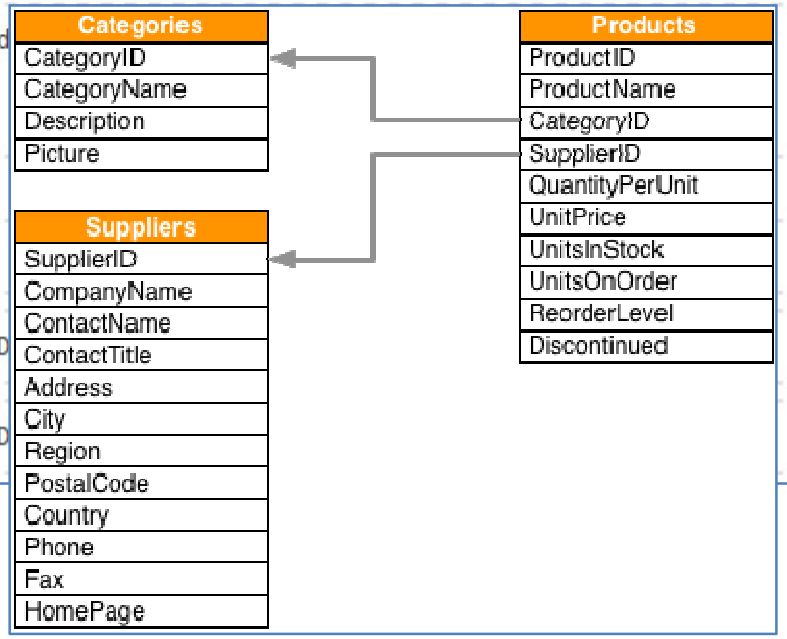
```
LOAD CSV WITH HEADERS FROM "http://d
CREATE (n:Supplier)
SET n = row
```

Create indexes

```
CREATE INDEX ON :Product(productID)
```

```
CREATE INDEX ON :Category(categoryID
```

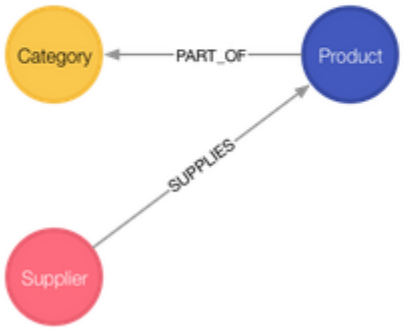
```
CREATE INDEX ON :Supplier(supplierID
```



Neo4j, większy przykład

Querying Product Catalog Graph

Lets try some queries using patterns.



Query using patterns

```
MATCH (s:Supplier)-->(:Product)-->(c:Category)
RETURN s.companyName as Company, collect(distinct c.categoryName) as Categories
```

List the product categories provided by each supplier.

```
MATCH (c:Category {categoryName:"Produce"})<--(:Product)<--(s:Supplier)
RETURN DISTINCT s.companyName as ProduceSuppliers
```

Find the produce suppliers.

Neo4j, większy przykład

Customer Order Graph

Notice that Order Details are always part of an Order and that they *relate* the Order to a Product — they're a join table. Join tables are always a sign of a data relationship, indicating shared information between two other records.

Here, we'll directly promote each OrderDetail record into a relationship in the graph.

Load and index records

```
LOAD CSV WITH HEADERS FROM "http://data.neo4j.com/northwind/order-details.csv" AS row
MATCH (p:Product), (o:Order)
WHERE p.productID = row.productID AND o.orderID = row.orderID
CREATE (o)-[details:ORDERS]->(p)
SET details = row,
  details.quantity = toInteger(row.quantity)
```

Note you only need to compare property values like this when first creating relationships

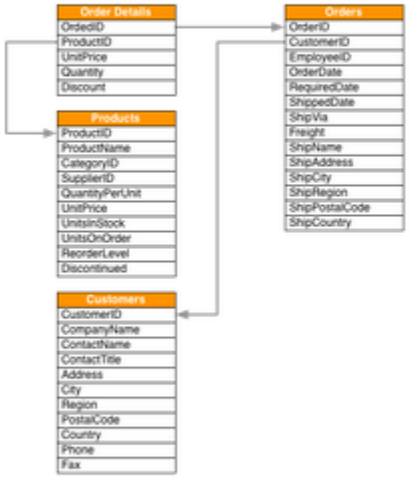
Query using patterns

```
MATCH (cust:Customer)-[:PURCHASED]->(:Order)-[o:ORDERS]->(p:Product),
      (p)-[:PART_OF]->(c:Category {categoryName:"Produce"})
RETURN DISTINCT cust.contactName as CustomerName, SUM(o.quantity) AS TotalProductsPurchased
```

Neo4j, większy przykład

Customer Orders

Northwind customers place orders which may detail multiple products.



:help ? cypher ? LOAD CSV

Load and index records

```
LOAD CSV WITH HEADERS FROM "http://data.neo4j.com/northwind/customers.csv" AS row
CREATE (n:Customer)
SET n = row
```

```
LOAD CSV WITH HEADERS FROM "http://data.neo4j.com/northwind/orders.csv" AS row
CREATE (n:Order)
SET n = row
```

```
CREATE INDEX ON :Customer(customerID)
```

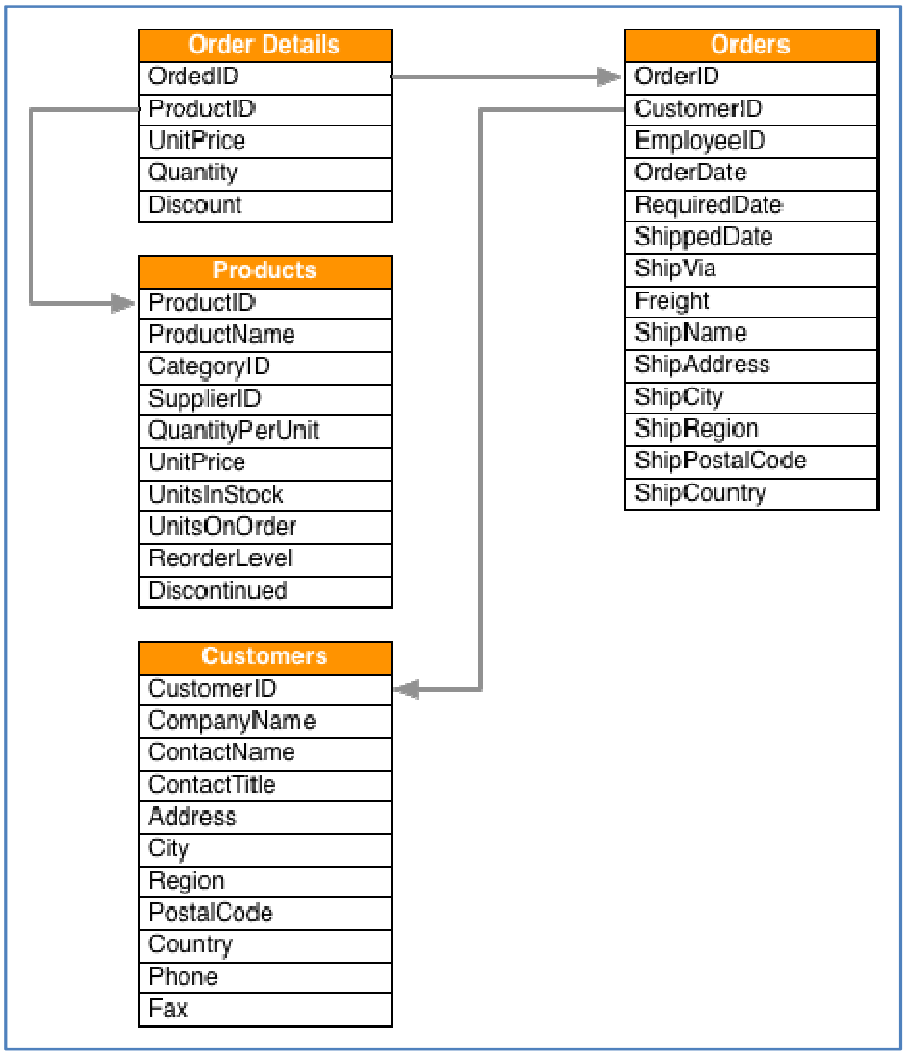
```
CREATE INDEX ON :Order(orderID)
```

Create data relationships

```
MATCH (c:Customer),(o:Order)
WHERE c.customerID = o.customerID
CREATE (c)-[:PURCHASED]->(o)
```

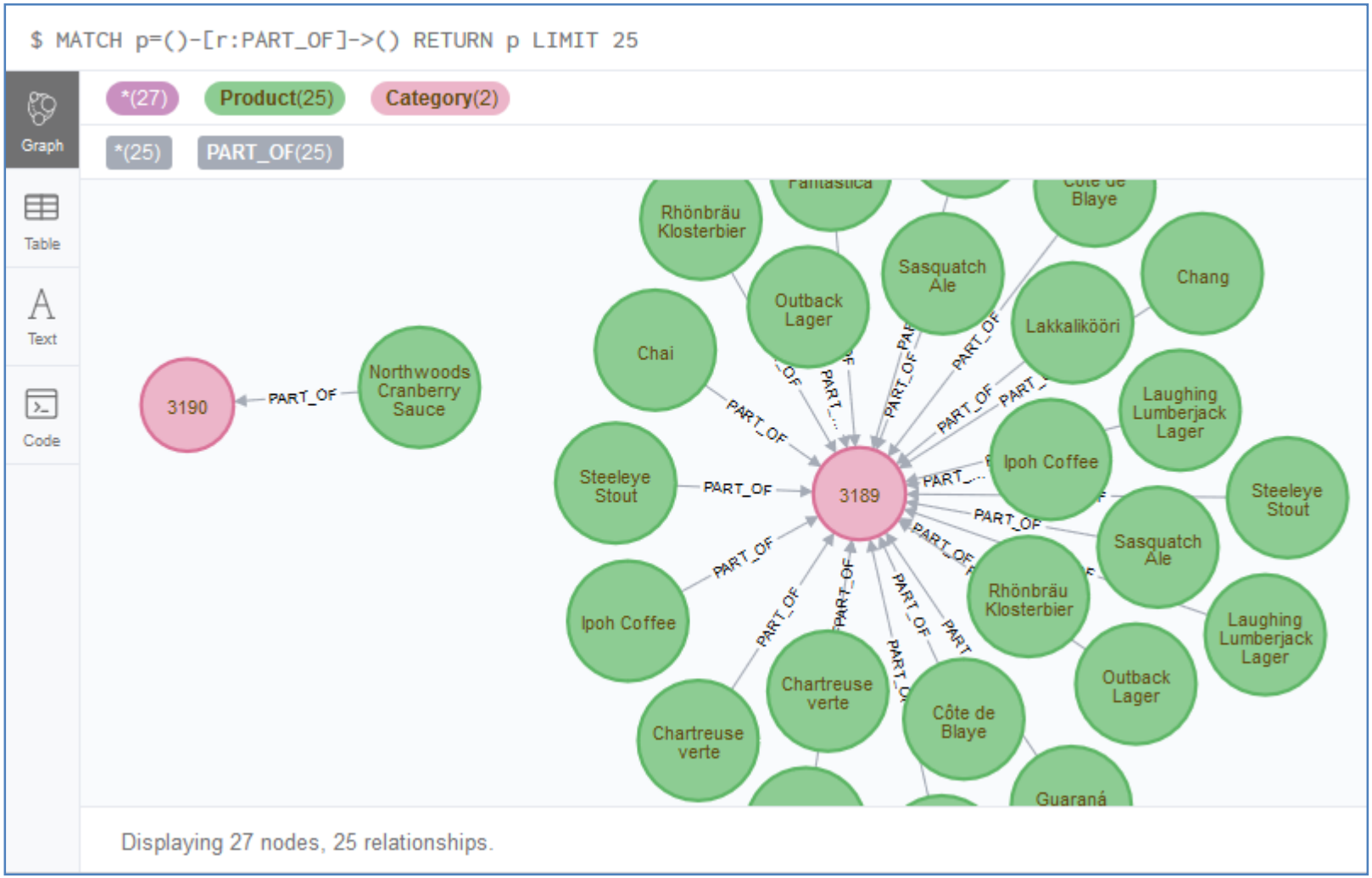
Note you only need to compare property values like this when first creating relationships

Neo4j, większy przykład



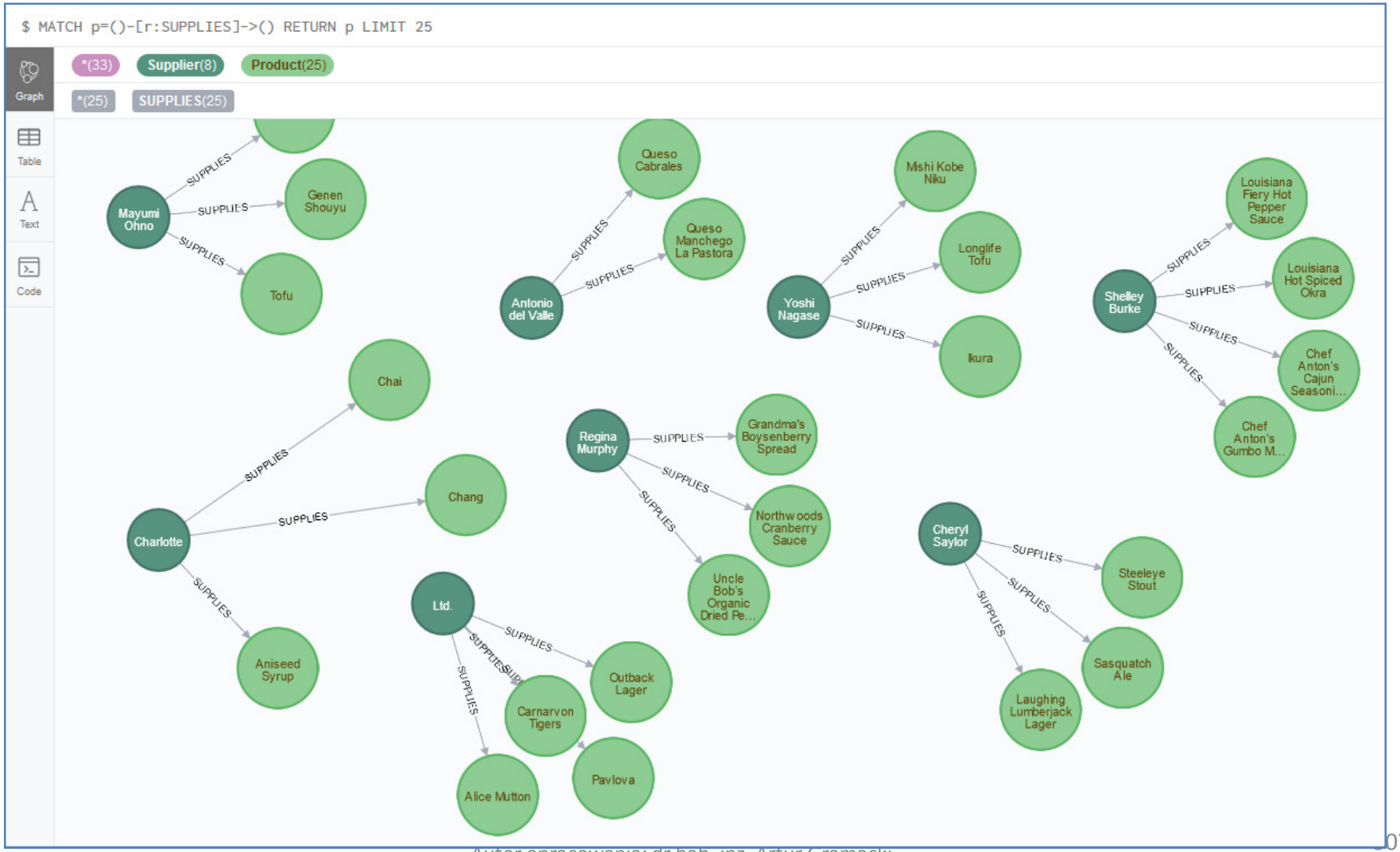
Neo4j, większy przykład

- PART_OF



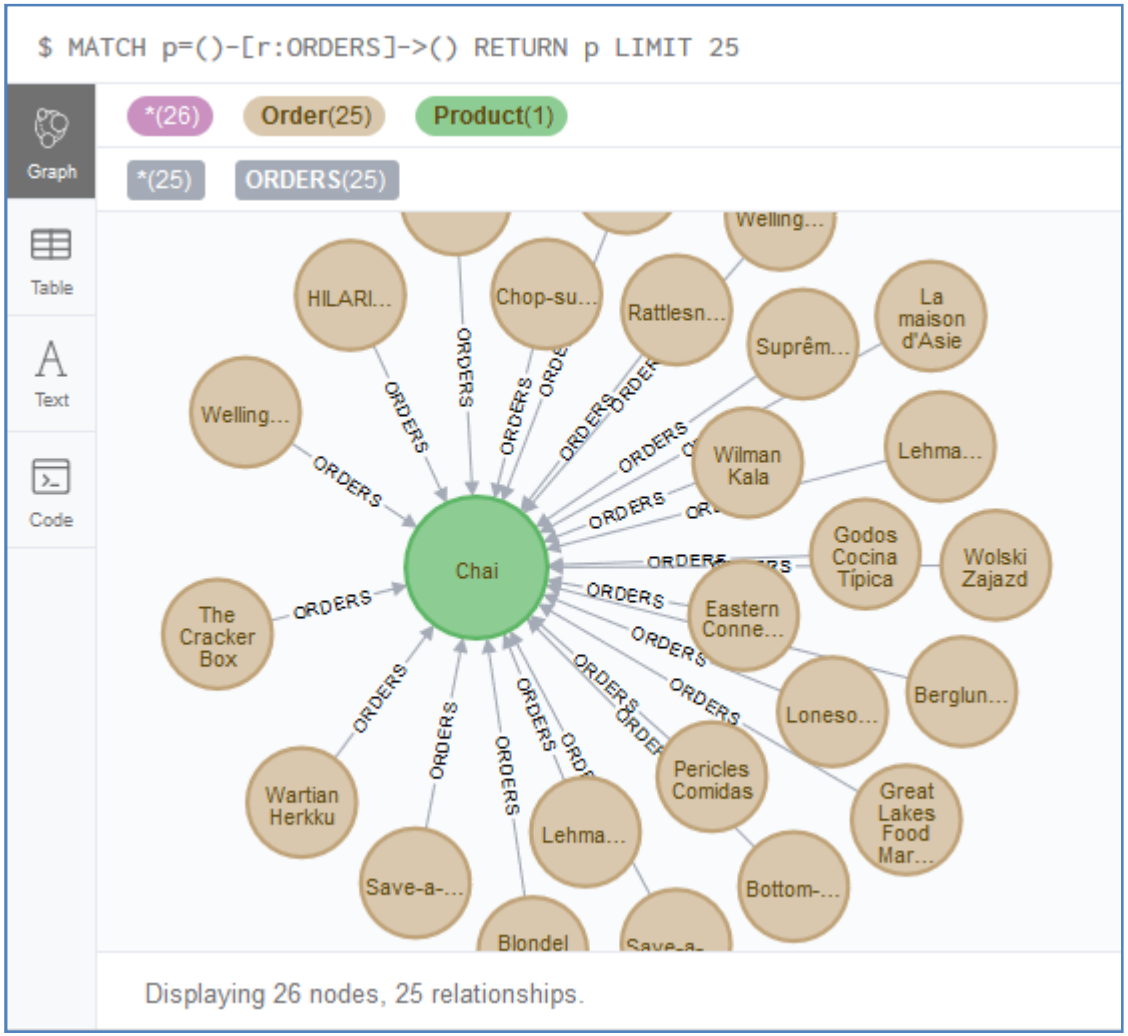
Neo4j, większy przykład

- SUPPLIES



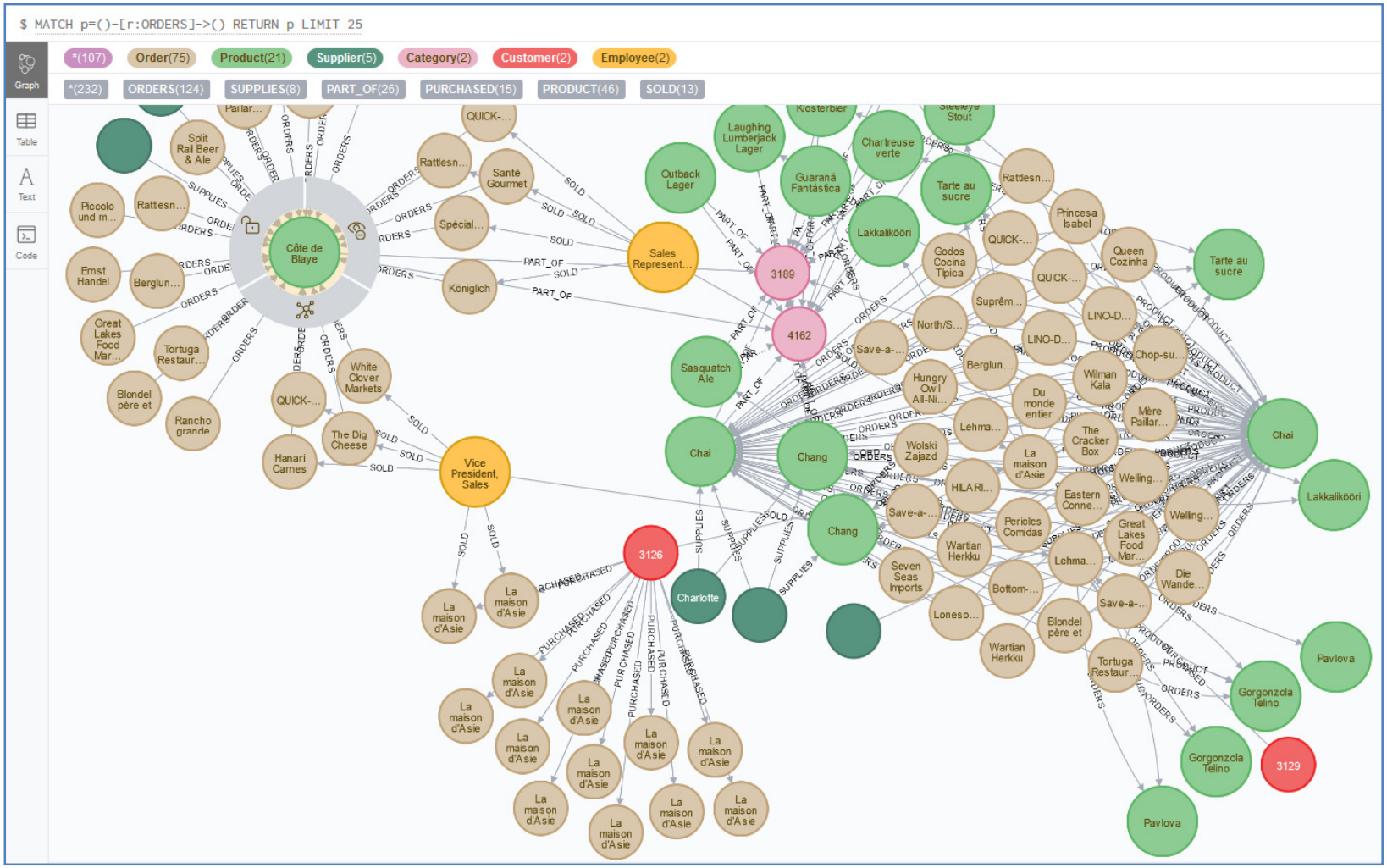
Neo4j, większy przykład

- ORDERS



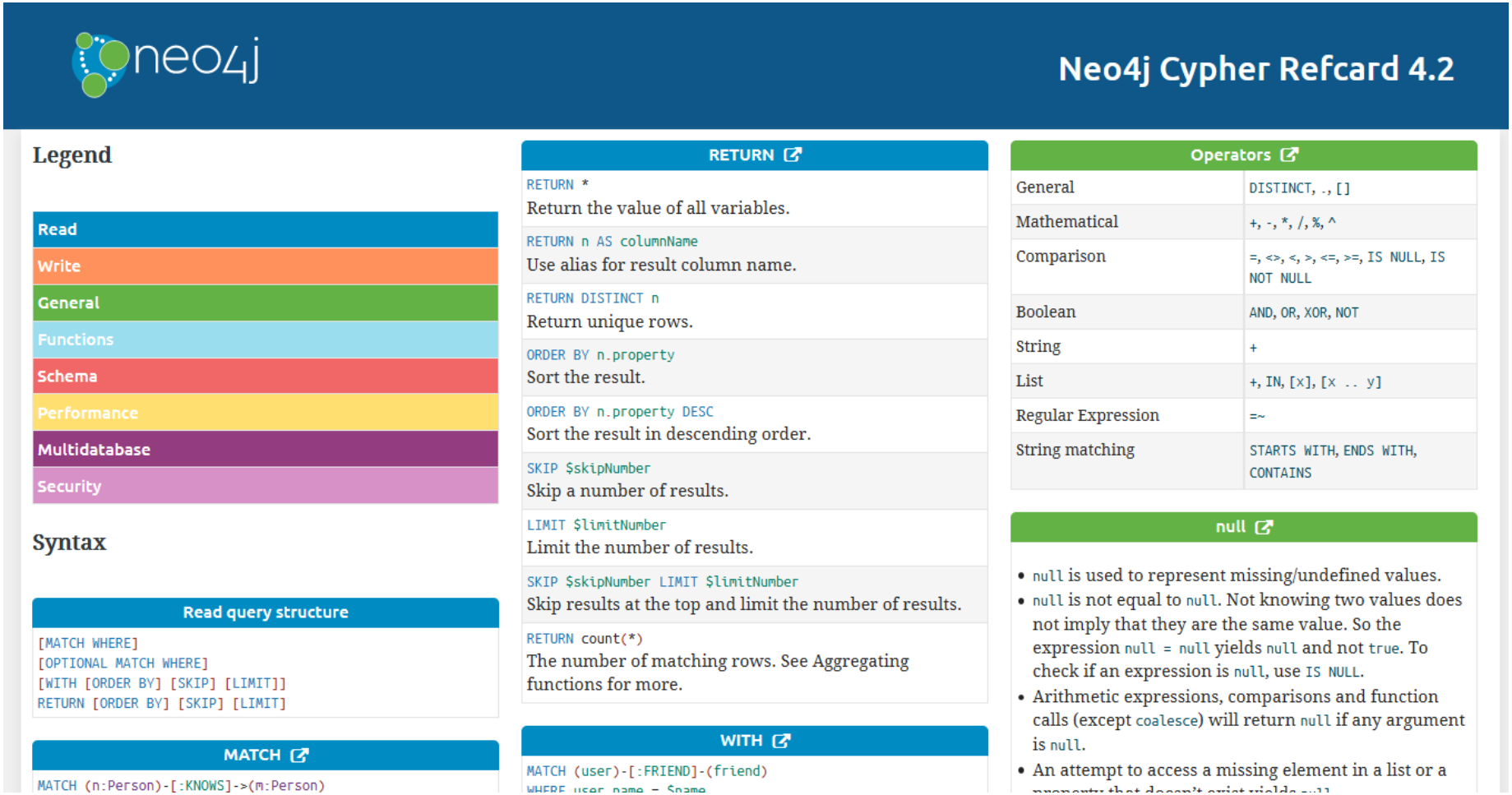
Neo4j, większy przykład

- „Sieczka” na ekranie ☺



Neo4j, refcard

- https://neo4j.com/docs/cypher-refcard/4.2/



neo4j

Neo4j Cypher Refcard 4.2

Legend

- Read
- Write
- General
- Functions
- Schema
- Performance
- Multidatabase
- Security

Syntax

Read query structure

```
[MATCH WHERE]
[OPTIONAL MATCH WHERE]
[WITH [ORDER BY] [SKIP] [LIMIT]]
RETURN [ORDER BY] [SKIP] [LIMIT]
```

MATCH

```
MATCH (n:Person)-[:KNOWS]->(m:Person)
```

RETURN

`RETURN *`
Return the value of all variables.

`RETURN n AS columnName`
Use alias for result column name.

`RETURN DISTINCT n`
Return unique rows.

`ORDER BY n.property`
Sort the result.

`ORDER BY n.property DESC`
Sort the result in descending order.

`SKIP $skipNumber`
Skip a number of results.

`LIMIT $limitNumber`
Limit the number of results.

`SKIP $skipNumber LIMIT $limitNumber`
Skip results at the top and limit the number of results.

`RETURN count(*)`
The number of matching rows. See Aggregating functions for more.

WITH

```
MATCH (user)-[:FRIEND]-(friend)
WHERE user.name = $name
```

Operators

General	`DISTINCT, ., []`
Mathematical	`+, -, *, /, %, ^`
Comparison	`=, <>, <, >, <=, >=, IS NULL, IS NOT NULL`
Boolean	`AND, OR, XOR, NOT`
String	`+`
List	`+, IN, [x], [x .. y]`
Regular Expression	`=~`
String matching	`STARTS WITH, ENDS WITH, CONTAINS`

null

- `null` is used to represent missing/undefined values.
- `null` is not equal to `null`. Not knowing two values does not imply that they are the same value. So the expression `null = null` yields `null` and not `true`. To check if an expression is `null`, use `IS NULL`.
- Arithmetic expressions, comparisons and function calls (except `coalesce`) will return `null` if any argument is `null`.
- An attempt to access a missing element in a list or a

Źródła

- Przygotowując ten wykład korzystałem z różnych źródeł, głównie internetowych. Czasami trudno jest dotrzeć do pierwotnego autora danego materiału, jeżeli więc kogoś pominąłem, proszę o informację. Wiele linków podaję na poszczególnych stronach