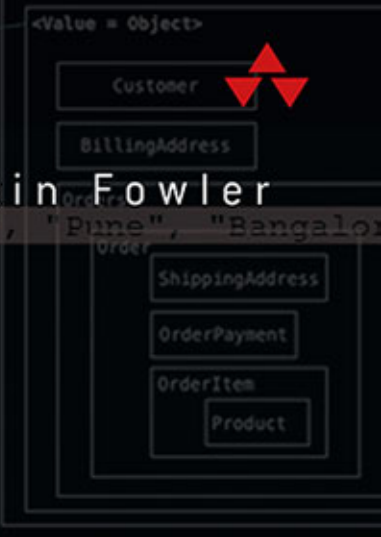


Pramod J. Sadalage | Martin Fowler

```
{  
  "first_name": "Prasad",  
  "citiesvisited": [ "Chicago", "London", "Pune", "Bangalore" ],  
  "addresses": [  
    {  
      "state": "AK",  
      "city": "DILLINGHAM",  
      "type": "R"  
    },  
    {  
      "state": "MH",  
      "city": "PUNE",  
      "type": "R"  
    }  
  ],  
  "lastcity": "Chicago"  
}
```



NoSQL

Kompendium wiedzy

Poznaj fascynujący świat baz danych NoSQL!

Tytuł oryginału: NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence

Tłumaczenie: Jakub Hubisz

ISBN: 978-83-246-9905-6

Authorized translation from the English language edition, entitled: NOSQL DISTILLED: A BRIEF GUIDE TO THE EMERGING WORLD OF POLYGLOT PERSISTENCE; ISBN 0321826620; by Pramodkumar J. Sadalage; and by Martin Fowler; published by Pearson Education, Inc, publishing as Addison Wesley.
Copyright © 2013 Pearson Education, Inc.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education, Inc.
Polish language edition published by HELION S.A. Copyright © 2015.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie bierze jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Wydawnictwo HELION nie ponosi również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: helion@helion.pl
WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<http://helion.pl/user/opinie/nosqlk>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

Spis treści

Przedmowa	11
Część I. Zrozumienie	17
Rozdział 1. Dlaczego NoSQL?	19
1.1. Wartość baz relacyjnych	19
1.1.1. Przechowywanie trwałych danych	19
1.1.2. Współbieżność	20
1.1.3. Integracja	20
1.1.4. Ustandaryzowany (przeważnie) model	20
1.2. Niezgodność impedancji	21
1.3. Bazy aplikacji i integracji	22
1.4. Atak klastrów	23
1.5. Pojawienie się baz NoSQL	25
1.6. Najważniejsze kwestie	27
Rozdział 2. Agregacyjne modele danych	29
2.1. Agregacje	29
2.1.1. Przykłady relacji i agregacji	30
2.1.2. Konsekwencje orientacji na agregacje	34
2.2. Modele klucz – wartość i dokumentów	35
2.3. Magazyny rodziny kolumn	36
2.4. Podsumowanie baz zorientowanych na agregacje	38
2.5. Dalsza lektura	39
2.6. Najważniejsze kwestie	39
Rozdział 3. Więcej szczegółów na temat modeli danych	41
3.1. Relacje	41
3.2. Bazy grafowe	42
3.3. Bazy danych bez schematu	44

3.4. Widoki zmaterializowane	46
3.5. Modelowanie z myślą o dostępie do danych	47
3.6. Najważniejsze kwestie	51
Rozdział 4. Modele dystrybucyjne	53
4.1. Pojedynczy serwer	53
4.2. Współdzielenie	54
4.3. Replikacja master-slave	56
4.4. Replikacja peer-to-peer	57
4.5. Łączenie shardingu i replikacji	59
4.6. Najważniejsze kwestie	59
Rozdział 5. Spójność	61
5.1. Spójność aktualizacji	61
5.2. Spójność odczytu	63
5.3. Rozluźnianie spójności	66
5.3.1. Teoria CAP	66
5.4. Rozluźnianie trwałości	69
5.5. Kwora	70
5.6. Dalsza lektura	72
5.7. Najważniejsze kwestie	72
Rozdział 6. Stemple wersji	73
6.1. Transakcje biznesowe i systemowe	73
6.2. Stemple wersji na wielu serwerach	75
6.3. Najważniejsze kwestie	76
Rozdział 7. Map-reduce	77
7.1. Podstawy map-reduce	78
7.2. Partycjonowanie i łączenie	79
7.3. Tworzenie obliczeń map-reduce	81
7.3.1. Przykład dwuetapowego map-reduce	82
7.3.2. Inkrementacyjny map-reduce	85
7.4. Dalsza lektura	86
7.5. Najważniejsze kwestie	86
Część II. Implementacja	89
Rozdział 8. Bazy klucz – wartość	91
8.1. Czym jest magazyn klucz – wartość?	91
8.2. Funkcjonalności magazynów klucz – wartość	93
8.2.1. Spójność	93
8.2.2. Transakcje	94
8.2.3. Możliwości zapytań	94
8.2.4. Struktura danych	95
8.2.5. Skalowanie	96

8.3. Pasujące przypadki użycia	96
8.3.1. Przechowywanie informacji o sesjach	96
8.3.2. Profile i preferencje użytkownika	97
8.3.3. Dane koszyka zakupów	97
8.4. Kiedy nie stosować	97
8.4.1. Relacje pomiędzy danymi	97
8.4.2. Transakcje dla wielu operacji	97
8.4.3. Zapytania na danych	97
8.4.4. Operacje na zestawach	98
Rozdział 9. Bazy dokumentów	99
9.1. Czym jest baza dokumentów?	99
9.2. Funkcjonalności	100
9.2.1. Spójność	101
9.2.2. Transakcje	102
9.2.3. Dostępność	102
9.2.4. Możliwości zapytań	103
9.2.5. Skalowanie	105
9.3. Pasujące przypadki użycia	107
9.3.1. Logowanie zdarzeń	107
9.3.2. Systemy zarządzania zawartością i platformy blogerskie	107
9.3.3. Analizy stron internetowych lub analizy w czasie rzeczywistym	107
9.3.4. Aplikacje e-commerce	107
9.4. Kiedy nie stosować	107
9.4.1. Złożone transakcje obejmujące różne operacje	107
9.4.2. Zapytania na zmiennej strukturze agregacji	108
Rozdział 10. Bazy rodziny kolumn	109
10.1. Czym jest magazyn rodziny kolumn?	109
10.2. Funkcjonalności	110
10.2.1. Spójność	112
10.2.2. Transakcje	113
10.2.3. Dostępność	113
10.2.4. Możliwości zapytań	114
10.2.5. Skalowanie	116
10.3. Pasujące przypadki użycia	116
10.3.1. Logowanie zdarzeń	116
10.3.2. Systemy zarządzania treścią i platformy blogowe	117
10.3.3. Liczniki	117
10.3.4. Wygasające dane	117
10.4. Kiedy nie stosować	118

Rozdział 11. Bazy grafowe	119
11.1. Czym jest baza grafowa?	119
11.2. Funkcjonalności	120
11.2.1. Spójność	122
11.2.2. Transakcje	122
11.2.3. Dostępność	123
11.2.4. Możliwości zapytań	123
11.2.5. Skalowanie	126
11.3. Pasujące przypadki użycia	127
11.3.1. Dane połączone	127
11.3.2. Wytaczanie trasy, wysyłka i usługi oparte o położenie	127
11.3.3. Silniki rekomendacji	128
11.4. Kiedy nie stosować	128
Rozdział 12. Zmiany schematów	129
12.1. Zmiany schematu	129
12.2. Zmiany schematu w bazach transakcyjnych	129
12.2.1. Zmiany w projektach budowanych od podstaw	130
12.2.2. Zmiany w projektach zastanych	132
12.3. Zmiany schematu w magazynach danych NoSQL	133
12.3.1. Zmiany inkrementacyjne	135
12.3.2. Zmiany w bazach grafowych	136
12.3.3. Zmiana struktury agregacji	137
12.4. Dalsza lektura	137
12.5. Najważniejsze kwestie	137
Rozdział 13. Poliglotyczne przechowywanie danych	139
13.1. Odmienne potrzeby przechowywania danych	139
13.2. Poliglotyczne wykorzystanie magazynu danych	140
13.3. Usługi a bezpośrednio przechowywanie danych	142
13.4. Rozszerzanie dla polepszenia funkcjonalności	142
13.5. Wybór odpowiedniej technologii	143
13.6. Problemy korporacyjne przy poliglotycznym przechowywaniu danych	144
13.7. Złożoność wdrożenia	145
13.8. Najważniejsze kwestie	145
Rozdział 14. Poza NoSQL	147
14.1. Systemy plików	147
14.2. Event sourcing	148
14.3. Obraz w pamięci	150
14.4. Kontrola wersji	151
14.5. Bazy XML	151
14.6. Bazy obiektowe	152
14.7. Najważniejsze kwestie	152

Rozdział 15. Wybór bazy danych	153
15.1. Wydajność programistów	153
15.2. Wydajność dostępu do danych	155
15.3. Trzymanie się standardów	156
15.4. Odwoływanie przypuszczeń	156
15.5. Najważniejsze kwestie	157
15.6. Końcowe przemyślenia	157
Bibliografia	159
Skorowidz	163

Rozdział 4

Modele dystrybucyjne

Główną cechą napędzającą zainteresowanie bazami NoSQL jest ich zdolność do działania w dużych klastrach. W miarę jak ilość danych rośnie, skalowanie w górę (kupno większego serwera, na którym działa baza danych) staje się coraz trudniejsze i droższe. Wygodniejszą opcją jest skalowanie w bok — uruchomienie bazy na klastrze serwerów. Orientacja na agregacje daje dobre podstawy do skalowania w bok, ponieważ agregacja jest idealną jednostką do dystrybucji.

Zależnie od Twojego modelu dystrybucyjnego możesz wykorzystać magazyn danych, który da Ci możliwość obsługi większych ilości danych, możliwość przetworzenia większego ruchu odczytu i zapisu lub większą niezawodność w dostępie do danych. Zazwyczaj są to ważne cechy, jednak wszystko ma swoją cenę. Praca w klastrze wprowadza złożoność — nie jest to więc coś, co powinieneś rozważać, jeżeli nie ma prawdziwej konieczności.

Są dwie główne ścieżki dystrybucji: replikacja i współdzielenie. W przypadku replikacji te same dane kopiowane są na wiele serwerów. Współdzielenie to umieszczenie różnych danych na różnych serwerach. Replikacja i współdzielenie to techniki ortogonalne — możesz korzystać tylko z jednej lub z obu. Replikacja przyjmuje dwie formy: *master-slave* lub *peer-to-peer*. Omówimy powyższe techniki, rozpoczynając od najprostszej, a następnie przechodząc do bardziej złożonych: najpierw pojedynczy serwer, potem replikacja *master-slave*, następnie współdzielenie i w końcu replikacja *peer-to-peer*.

4.1. Pojedynczy serwer

Pierwszą i najprostszą opcją dystrybucji jest ta, którą przeważnie polecam — brak dystrybucji, czyli uruchomienie bazy danych na jednej maszynie, która obsługuje wszystkie odczyty i zapisy do magazynu danych. Polecamy tę opcję, ponieważ pozwala wyeliminować problemy istniejące przy innych formach dystrybucji. Taki model nie sprawia problemów ani osobom zajmującym się serwerem, ani programistom korzystającym z bazy.

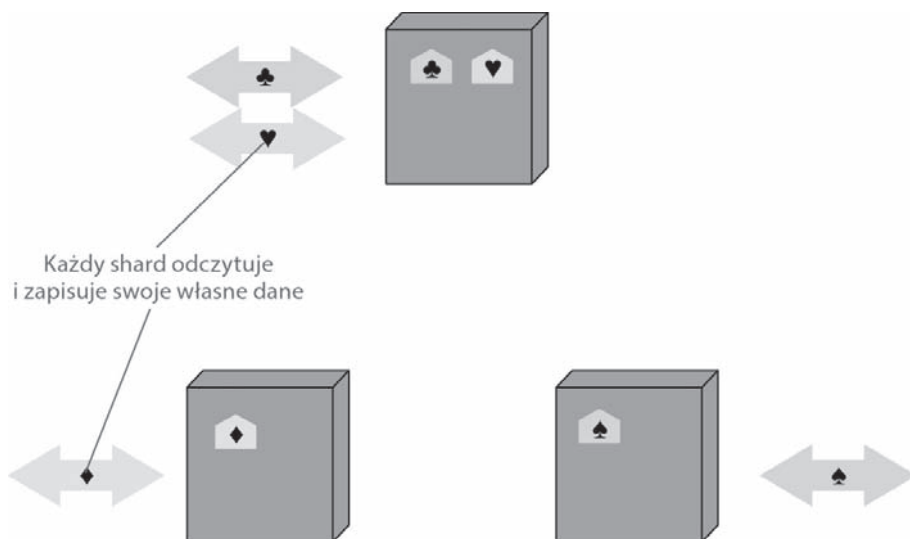
Mimo że wiele baz NoSQL zaprojektowanych jest z myślą o pracy w klastrach, sensowne jest też używanie NoSQL na pojedynczym serwerze, jeżeli model danych oferowany przez NoSQL jest bardziej przystosowany do danych zastosowań. Bazy grafowe są tutaj oczywistym przykładem — działają najlepiej na jednym serwerze. Jeżeli Twoje potrzeby skupiają się na

przetwarzaniu agregacji, jednoserwerowy magazyn dokumentów lub baza klucz – wartość mogą być warte rozważenia, ponieważ ułatwią pracę programistom.

Przez resztę tego rozdziału będziemy rozważać zalety i problemy związane z bardziej rozbudowanymi modelami dystrybucyjnymi. Nie myśl jednak, że ilość poświęconego miejsca oznacza, że preferujemy te opcje. Jeżeli możemy poradzić sobie bez konieczności dystrybuowania danych, zawsze wybieramy rozwiązanie z pojedynczym serwerem.

4.2. Współdzielenie

Zajęty magazyn danych jest często zajęty, ponieważ różne osoby uzyskują dostęp do różnych części danych. W takich przypadkach możemy wykonać skalowanie poziome, umieszczając różne części danych na różnych serwerach — technika ta nazywana jest **shardingiem** (rysunek 4.1).



Rysunek 4.1. Sharding pozwala umieścić różne dane na osobnych serwerach; każdy z tych serwerów wykonuje własne odczyty i zapisy

W idealnym przypadku różni użytkownicy komunikowaliby się z różnymi serwerami. Każdy użytkownik musiałby komunikować się z tylko jednym serwerem, dzięki czemu odpowiedzi na zapytania byłyby bardzo szybkie. Ruch rozkładałby się na wszystkie serwery — na przykład jeżeli mamy dziesięć serwerów, każdy musiałby obsłużyć tylko 10% ruchu.

Oczywiście sytuacja idealna występuje bardzo rzadko. Aby się do niej zbliżyć, musimy zadbać, aby dane, które pobierane są wspólnie, były przechowywane razem oraz aby organizacja danych była maksymalnie zoptymalizowana pod względem dostępu do danych.

Pierwsze pytanie brzmi, jak zgrupować dane tak, aby użytkownik, uzyskując do nich dostęp, odwoływał się w większości do jednego serwera. Tutaj z pomocą przychodzi orientacja na agregacje. Cała idea agregacji polega na budowaniu ich w taki sposób, aby dane, które prze-

ważnie obsługiwane są wspólnie, były też wspólnie przechowywane — agregacje stają się więc naturalną jednostką dystrybucji.

Jeżeli chodzi o organizację danych na serwerze, jest kilka czynników, które mogą pomóc poprawić wydajność. Jeżeli wiesz, że większość dostępu do danych agregacji wychodzi z jednego fizycznego miejsca, możesz umieścić te dane blisko odbiorców. Jeżeli realizujesz zamówienia dla kogoś mieszkającego w Londynie, możesz umieścić dane w centrum danych na Wyspach Brytyjskich.

Kolejnym czynnikiem jest rozłożenie obciążenia. Oznacza to, że powinieneś tak organizować agregacje, aby były rozłożone na serwerach równomiernie, dzięki czemu poszczególne serwery będą obciążone w podobnym stopniu. Ten czynnik może być zmienny w czasie, na przykład jeżeli niektóre dane używane są tylko w niektóre dni tygodnia — w tym przypadku mogą więc występować reguły zależne od zastosowania bazy.

W niektórych przypadkach warto umieszczać agregacje razem, jeżeli myślisz, że mogą być pobierane w sekwencjach. Dokument o BigTable [Chang i inni] opisał przechowywanie wierszy w porządku leksykograficznym i sortowanie adresów na podstawie odwróconych domen (np. `com.martinfowler`). Dzięki temu dane z różnych stron mogły być odczytywane razem, co pozwoliło podnieść wydajność przetwarzania.

Większość projektantów uwzględniła lub uwzględniła sharding w warstwie logiki aplikacji. Możesz umieścić wszystkich klientów o nazwiskach zaczynających się od A do D na jednym serwerze, a tych, których nazwiska zaczynają się od E do G, na innym. To komplikuje model programistyczny, ponieważ kod aplikacji musi zadbać, aby zapytania były dystrybuowane na różne serwery. Co więcej, zmiana rozłożenia danych na serwerach wiąże się ze zmianami w kodzie aplikacji oraz migracją danych. Wiele baz NoSQL udostępnia opcję *auto-sharding*, która pozwala przenieść odpowiedzialność za rozdzielanie danych i przekierowywanie zapytań do odpowiedniego serwera na bazę. Takie rozwiązanie jest znacznie prostsze niż sharding za pośrednictwem aplikacji.

Sharding jest bardzo pomocny w poprawianiu wydajności, ponieważ zwiększa zarówno szybkość odczytu, jak i szybkość zapisu danych. Korzystając z replikacji, szczególnie przy wykorzystywaniu buforowania, można znacznie poprawić wydajność operacji odczytu, jednak replikacja nie poprawi wydajności w przypadku bazy, do której wykonywane jest dużo zapisów.

Sam sharding poprawia niezawodność w bardzo niewielkim stopniu. Mimo że dane są na różnych serwerach, awaria serwera spowoduje, że dane te będą niedostępne, tak samo jak w przypadku rozwiązania z pojedynczym serwerem. Jedyna poprawa polega na tym, że w przypadku awarii serwera jej skutki odczują tylko użytkownicy tych danych; mimo wszystko jednak niedobrze jest mieć bazę danych posiadającą tylko część danych. Mając pojedynczy serwer, łatwiej jest poświęcić uwagę i pieniądze w celu zapewnienia jego działania; w klastrach pracują przeważnie bardziej zawodne maszyny i awaria serwera w klastrze jest bardziej prawdopodobna. To wszystko powoduje, że w praktyce sam sharding wpływa na osłabienie niezawodności.

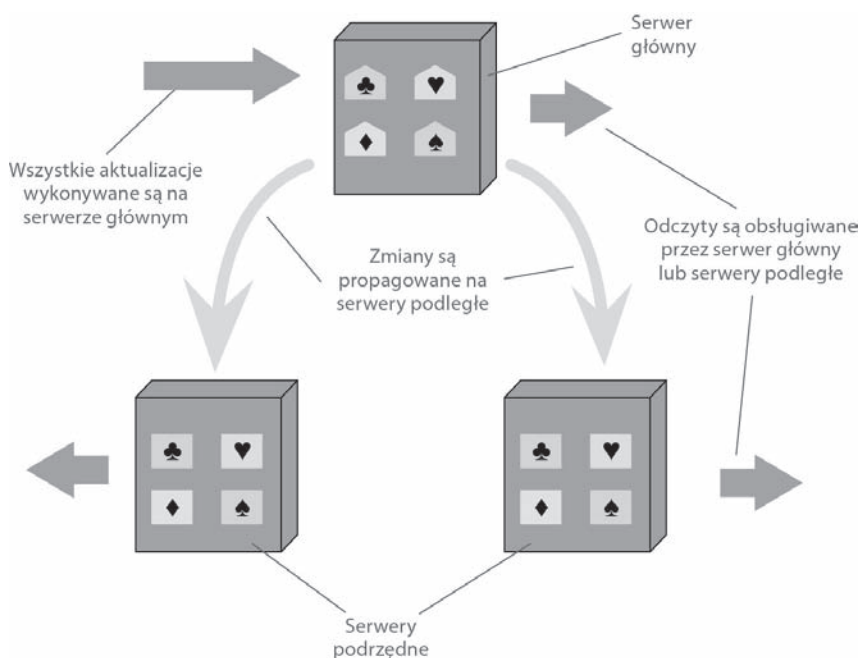
Mimo że sharding jest łatwiejszy do zaimplementowania w przypadku agregacji, nie jest to krok, który powinniśmy wykonywać pochopnie. Niektóre bazy danych są przeznaczone do wykorzystania shardingu od samego początku; w takim przypadku warto uruchamiać je na klastrze już w czasie prac nad programem i koniecznie od początku działania aplikacji produkcyjnej. Inne bazy pozwalają na wykorzystanie shardingu w formie rozbudowy

rozwiązania jednoserwerowego; wtedy klastr powinien być zakładany dopiero wtedy, gdy wydajność pojedynczego serwera przestanie być wystarczająca.

W obu przypadkach przejście z jednego serwera do klastra będzie wymagało uwagi. Znamy opowieści o zespołach, które napotkały problemy, ponieważ zbyt długo odkładały sharding, a kiedy włączyły go na środowisku produkcyjnym, baza przestała być dostępna, ponieważ wsparcie dla shardingu pochłonęło wszystkie dostępne zasoby na potrzeby przenoszenia danych na nowe serwery. Morał z tego taki, że powinieneś implementować sharding, zanim zaczniesz go potrzebować, kiedy jeszcze masz zasoby do przenoszenia danych.

4.3. Replikacja master-slave

Podczas dystrybucji *master-slave* replikujesz dane na wiele serwerów. Jeden z serwerów jest serwerem głównym (*master*). Serwer główny jest autorytatywnym źródłem danych i jest z reguły odpowiedzialny za przetwarzanie aktualizacji tych danych. Pozostałe serwery to serwery podległe (*slave*). Proces replikacji synchronizuje serwery podrzędne z serwerem głównym (rysunek 4.2).



Rysunek 4.2. Dane są replikowane z serwera głównego do serwerów podległych. Główny serwer obsługuje zapis danych, a odczyt może być obsługiwany przez serwer główny lub któryś z serwerów podległych

Replikacja *master-slave* jest pomocna w przypadku systemów, w których dane są głównie odczytywane. Można skalować w bok w celu poprawienia wydajności odczytu danych poprzez dodanie większej liczby serwerów i zapewnienie rozproszczenia po nich ruchu.

W przypadku zapisu jesteś jednak nadal ograniczany przez możliwości głównego serwera. W konsekwencji nie jest to rozwiązanie dobre dla baz obsługujących duży ruch, chociaż rozłożenie odczytów pozwoli też nieznacznie poprawić wydajność zapisu.

Drugą zaletą replikacji *master-slave* jest **niezawodność odczytu**: jeżeli główny serwer zawiedzie, serwery podrzędne nadal będą odpowiadały na żądania odczytu. Jest to oczywiście użyteczne, jeżeli większość ruchu stanowi odczyt. Awaria serwera głównego eliminuje możliwość zapisu danych, dopóki serwer główny nie zostanie przywrócony lub nie zostanie wyznaczony nowy serwer główny. Mając serwery podległe, na które replikowane są dane z serwera głównego, możemy szybko przywrócić system do działania, ponieważ serwer podległy może być szybko wyznaczony na serwer główny.

Możliwość wyznaczenia serwera podległego na nowy serwer główny oznacza, że ten typ replikacji może być przydatny nawet wtedy, kiedy nie musisz skalować w bok. Cały ruch może być kierowany na serwer główny, a serwer poboczny może działać jako wciąż aktualizowana kopia zapasowa. W takim przypadku najłatwiej myśleć o systemie jako o systemie z pojedynczym serwerem z automatycznie aktualizowaną kopią zapasową. Wygoda systemu jednoserwerowego łączy się z poprawą niezawodności — jest to szczególnie wygodne, jeżeli chcesz z łatwością radzić sobie z awariami systemu.

Serwery główne mogą być wyznaczane ręcznie lub automatycznie. Ręczne wyznaczanie serwera głównego oznacza zazwyczaj, że podczas konfiguracji klastra określasz jeden z serwerów jako główny. W wyznaczaniu automatycznym tworzysz klastr serwerów, a serwery same wybierają jeden z nich na serwer główny. Poza łatwiejszą konfiguracją wyznaczanie automatyczne oznacza, że klastr może automatycznie wyznaczyć nowy serwer główny w przypadku awarii poprzednika.

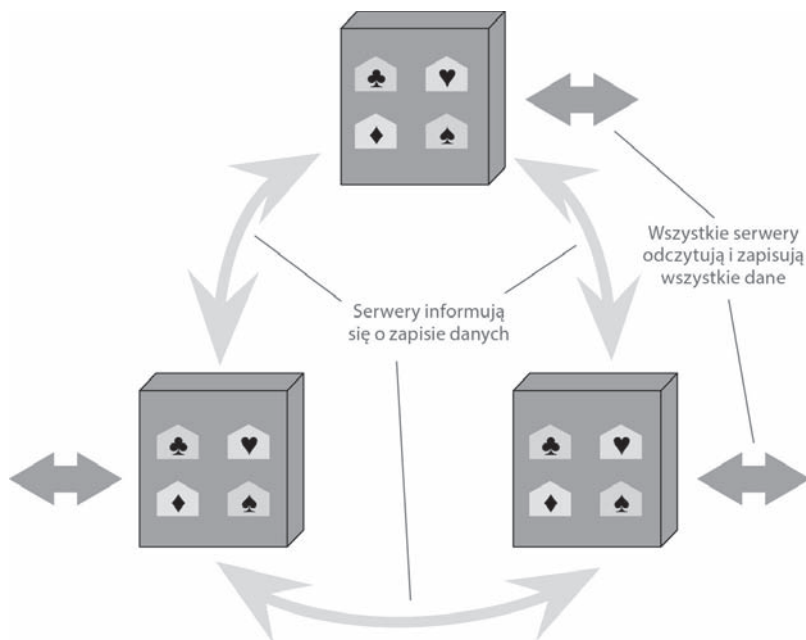
W celu uzyskania niezawodności odczytu musisz dopilnować, aby ścieżki odczytu i zapisu w Twojej aplikacji były różne, tak abyś mógł obsłużyć awarię ścieżki zapisu i móc nadal odczytywać dane. Łączy się to z koniecznością zapisywania i odczytywania danych za pomocą różnych połączeń — jest to funkcjonalność nieczęsto wspierana przez biblioteki do interakcji z bazami. Jak w przypadku wszystkich funkcjonalności, nie możesz mieć pewności, że odczyt danych jest niezawodny, bez dobrych testów wyłączających możliwość zapisu i sprawdzających odczyt.

Replikacja ma na pewno pociągające zalety, ma jednak także ciemną stronę — brak spójności. Istnieje niebezpieczeństwo, że różni klienci odczytujący różne serwery podrzędne zobaczą różne wartości, ponieważ wszystkie zmiany nie zostały rozpropagowane po wszystkich serwerach. W najgorszym przypadku może to oznaczać, że klient nie zobaczy danych, które właśnie zapisał. Nawet jeżeli korzystasz z replikacji *master-slave* wyłącznie do tworzenia kopii zapasowej danych, nadal możesz mieć problem, ponieważ jeżeli serwer główny ulegnie awarii, zmiany nieprzekazane do serwera podrzędnego zostaną utracone. Porozmawiamy o tym, jak radzić sobie z tym problemem, w rozdziale 5., „Spójność”.

4.4. Replikacja peer-to-peer

Replikacja *master-slave* pomaga w skalowaniu możliwości odczytu danych, nie pomaga jednak w przypadku zapisu. Poprawia niezawodność pod względem awarii serwerów podległych, ale nie serwera głównego. W gruncie rzeczy serwer główny jest nadal słabym ogniwem

systemu i miejscem podatnym na awarię. Replikacja *peer-to-peer* (rysunek 4.3) radzi sobie z tym problemem, eliminując serwer główny. Wszystkie repliki mają równe prawa i wszystkie mogą wykonywać zapis, a utrata którejś z nich nie ogranicza dostępu do magazynu danych.



Rysunek 4.3. W replikacji *peer-to-peer* wszystkie serwery obsługują odczyt i zapis danych

Perspektywa wygląda pięknie. Z klastrem replikującym się w trybie *peer-to-peer* możesz radzić sobie z awariami serwerów bez utraty dostępu do danych. Co więcej, możesz z łatwością dodawać nowe serwery, poprawiając wydajność. Taki system ma wiele zalet — powoduje też jednak pewne komplikacje.

Największym problemem jest, po raz kolejny, spójność. Jeżeli możesz zapisywać w dwóch różnych miejscach, występuje ryzyko, że dwie osoby spróbują jednocześnie zapisać te same dane: konflikt zapis – zapis. niespójność przy odczycie prowadzi do problemów, ale niespójność taka jest chwilowa. niespójność przy zapisie jest natomiast trwała.

W dalszej części wyjaśnimy, jak radzić sobie z niespójnością danych, w tej chwili jednak omówimy z grubsza kilka opcji. Z jednej strony możemy zapewnić, że kiedy dane zostaną zapisane, repliki skoordynują się w celu zapobieżenia konfliktom. Może nam to dać tak dobrą gwarancję jak baza główna, jednak kosztem ruchu sieciowego występującego podczas koordynacji. Nie potrzebujemy, aby wszystkie repliki zgadzały się na zapis, wystarczy większość, dzięki czemu będziemy mogli przeżyć, jeżeli stracimy mniejszość serwerów.

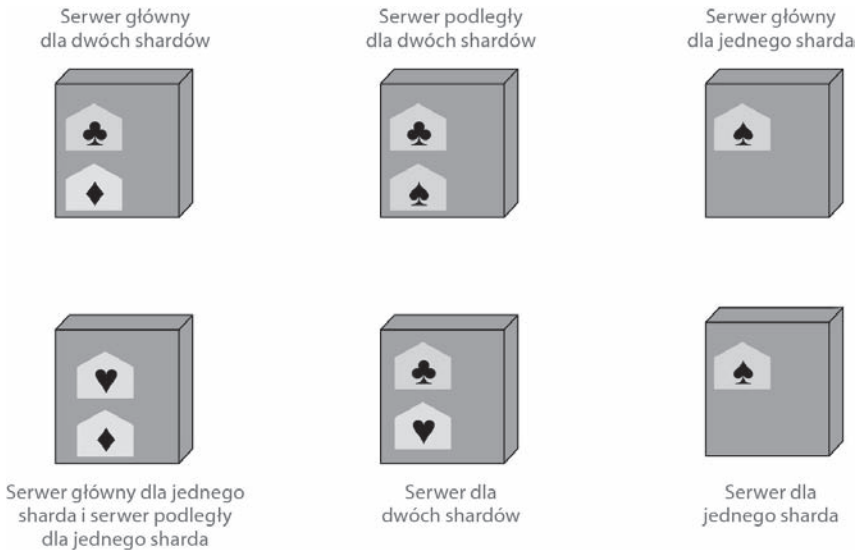
Z drugiej strony możemy zdecydować, aby zezwolić na niespójny zapis. W niektórych przypadkach możemy znaleźć sposób na scalenie niespójnych danych. W takim przypadku możemy korzystać z pełni wydajności zapewnianej przez repliki.

Te dwa sposoby są po przeciwnych stronach spektrum — zamieniamy spójność na dostępność.

4.5. Łączenie shardingu i replikacji

Replikacja i sharding to strategie, które możemy połączyć. Jeżeli wykorzystamy zarówno replikację *master-slave*, jak i sharding (rysunek 4.4), będziemy mieli wiele serwerów głównych, jednak każda część danych będzie miała tylko jeden serwer główny. Zależnie od konfiguracji możesz wybrać, aby serwer był serwerem głównym dla niektórych danych, a serwerem podległym dla innych, lub możesz wyznaczyć serwery dedykowane do bycia serwerami głównymi i pobocznymi.

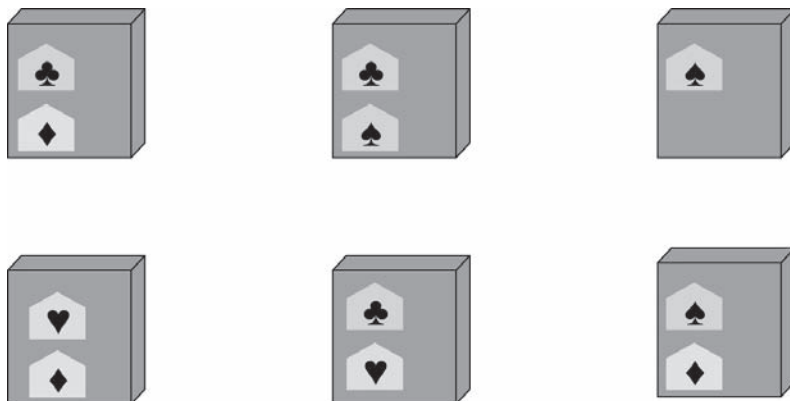
Wykorzystanie replikacji *peer-to-peer* i shardingu jest częste w przypadku baz rodziny kolumn. W takim przypadku możesz mieć dziesiątki lub setki serwerów w klastrze. Dobrym punktem startu w replikacji *peer-to-peer* jest współczynnik powtórzenia równy 3, czyli każda część danych reprezentowana jest na trzech serwerach. Jeżeli któryś z serwerów ulegnie awarii, shardy z tego serwera będą działały na pozostałych serwerach (rysunek 4.5).



Rysunek 4.4. Wykorzystanie replikacji *master-slave* razem z shardingiem

4.6. Najważniejsze kwestie

- Istnieją dwa sposoby obsługi danych rozproszonych:
 - Sharding rozdziela dane pomiędzy wiele serwerów, tak aby każdy serwer działał jako źródło dla podzbioru danych.
 - Replikacja kopiuje dane pomiędzy wieloma serwerami, tak aby każda część danych znajdowała się w wielu miejscach.



Rysunek 4.5. Wykorzystanie replikacji *peer-to-peer* razem z *shardingiem*

- Replikacja przyjmuje dwie postacie:
 - W replikacji *master-slave* jeden serwer odgrywa rolę serwera głównego obsługującego zapis danych i rozgłaszającego dane do serwerów podległych, które mogą obsługiwać odczyt danych.
 - W replikacji *peer-to-peer* zapis możliwy jest do dowolnego serwera; serwery koordynują synchronizację danych.

Replikacja *master-slave* redukuje ryzyko wystąpienia konfliktu zapisu, ale replikacja *peer-to-peer* pomaga rozładować ruch związany z zapisem danych.

Skorowidz

A

ACID, 35
agregacje, 12, 29–35, 48
aktualizacja warunkowa, 62
algorytm Dijkstry, 125
analiza danych w czasie rzeczywistym, 48
auto-sharding, 55

B

BASE, 69
baza
 aplikacji, 22, 23
 BigTable, 36, 55
 danych
 typologia, 15
 danych bez schematu, 44, 45
 dokumentów, 35, 99–108
 grafowa, 42, 43, 51, 119–128
 integrująca, 22, 23
 klucz – wartość, 35, 91–98
 NoSQL
 a bazy relacyjne, 28
 informacje ogólne, 11, 12, 25–28
 przechowywanie danych, 44
 wybór, 153–158
 obiektowa, 152
 relacyjna, 19–24
 rodziny kolumn, 36–38, 50, 109–118
 XML, 151
 zorientowana na agregacje, 29–39, 42

biblioteka map-reduce, 79
BigTable, 36, 55
branching, 151
BREADTH_FIRST, 124

C

CAP, 66–69
CAS, 74
Cassandra, 109–118
Cassandra Query Language, *Patrz* CQL
chudy wiersz, 38
CQL, 115, 116
CQRS, 149
Cypher, 123, 125, 126

D

dane
 niestandardowe, 44
 nieświeże, 64
DBDeploy, 131
DEPTH_FIRST, 124
domniemany schemat, 45
dostępność
 klastra w Cassandrze, 113, 114
 w Neo4J, 123
 w teorii CAP, 67

E

enkapsulacja bazy danych, 156
event sourcing, 148–150

F

faza przejściowa
 przy zmianie schematu, 132, 133
 FlockDB, 120
 funkcja
 combine, 80
 map, 79
 reduce, 79

G

graf
 struktura, 119, 120
 Gremlin, 123
 GUID, 74

H

hasz zawartości zasobu, 74

I

identyfikator GUID, 74
 integracja poprzez współdzieloną bazę danych, 20

K

klastry
 a bazy relacyjne, 24
 klucz sessionid, 96
 kolekcje w MongoDB, 100
 kolumna
 w Cassandrze, 110, 111
 wygasająca, 117
 kompaktowanie, 112
 koncepcja kworum, 94
 konflikt
 odczyt – zapis, 63
 zapis – zapis, 61, 62
 kontrola wersji, 151
 krawędzie, 43, 119
 krotka, 21, 29
 kwora, 70–72
 kworum zapisu, 71

L

log zdarzeń, 148–150

Ł

łączenie
 a map-reduce, 79–81
 łuki, *Patrz* krawędzie

M

magazyn
 danych, 36
 wykorzystanie poliglotyczne, 140
 klucz – wartość, 91–98
 rodziny kolumn, 36–38, 109–118
 wspierający, 19, 20
 mapowanie relacyjne, 34
 map-reduce, 77–87
 dwuetapowe, przykład, 82–85
 inkrementacyjny, 85, 86
 podstawy, 78, 79
 tworzenie obliczeń, 81, 82
 master, *Patrz* serwer główny
 migawka, 148, 149
 model
 danych, 29
 dla bazy relacyjnej, 30
 oparty o agregacje, 32
 dystrybucyjny, 53–60
 ignorujący agregacje, 35
 przechowywania, 29
 relacyjny, 21, 29, 42
 spójności ostatecznie spójny, 93
 modelowanie z myślą o dostępie do danych,
 47–51
 MongoDB, 100–106

N

naprawa odczytu, 112
 Neo4J, 120–127
 niezgodność\ impedancji, 21, 22
 NoSQL
 a bazy relacyjne, 28
 informacje ogólne, 11, 12, 25–28
 przechowywanie danych, 44
 wybór bazy danych, 153–158

O

obiekt Traverser, 124
 obraz w pamięci, 150
 odczyt niespójny, 63
 okno niespójności, 64
 operacja
 CAS, 74
 porównaj-i-ustaw, 74
 optymistyczna blokada offline, 74

P

pamięć
 główna, 19
 memtable, 112
 parametr
 slaveOk, 101
 WriteConcern, 102
 partycjonowanie a Map-reduce, 79–81
 podzielony umysł, 67
 pokrewieństwo sesji, 65
 pole blob, 91
 polecenie
 db.runCommand, 101
 DEL, 115
 GET, 115
 SET, 115
 poliglotyczne przechowywanie danych, 139
 porównaj-i-ustaw, 74
 programowanie poliglotyczne, 140
 przechowywanie danych
 poliglotyczne, 139
 przekazanie ze wskazaniem, 113
 przestrzenie kluczy, 112
 przyklejona sesja, 65

Q

QUORUM, 112

R

Redis, 92
 reduktor łączący, 80
 regiony, 79
 relacja, 41, 42
 w bazach grafowych, 121
 w modelu relacyjnym, 21

replikacja, 53, 59
 master-slave, 56, 57, 102
 peer-to-peer, 57, 58
 Riak, 92
 Riak Search, 94
 rodzina
 kolumn, 36–38, 109
 standardowa, 110
 superkolumn, 111, 112
 rozluźnianie
 spójności, 66
 trwałości, 69, 70

S

serwer
 główny, 56, 57
 podległy, 56, 57
 sharding, 54–56, 59, 96, 126
 w MongoDB, 106
 sieć zależności, 86
 skalowanie, 105
 baz grafowych, 126
 horyzontalne, 105
 w bazach klucz-wartość, 96
 w Cassandra, 116
 w MongoDB, 105, 106
 slave, *Patrz* serwer podległy
 spójność
 aktualizacji, 61, 62
 logiczna, 63
 odczytu, 63–65
 podejście optymistyczne, 61
 podejście pesymistyczne, 61
 replikacji, 64
 w bazach grafowych, 122
 w Cassandra, 112, 113
 w MongoDB, 101
 w sesji, 65
 SSTable, 112
 stempel
 czasowy ostatniej aktualizacji, 74
 wektor, 76
 wersji, 74, 75
 na wielu serwerach, 75, 76
 stracona aktualizacja, 61
 struktura danych w bazach klucz-wartość, 95
 superkolumna, 111
 systemy plików, 147
 szeroki wiersz, 38

T

tasowanie, 79
 teoria CAP, 66–69, 102
 testowanie bazy danych, 155
 tolerancja na partycjonowanie, 67, 68
 transakcja
 ACID, 35
 atomowa, 102
 biznesowa, 73–75
 systemowa, 73–75
 w Cassandra, 113
 w magazynie klucz–wartość, 94
 w MongoDB, 102
 w Neo4J, 122
 trawersowanie, 119, 120
 trwałość replikacji, 70
 TTL, 117
 typy baz danych, 15

W

wartość baz relacyjnych, 19–21
 wdrożenie poliglotyzmu, 145
 Webber Neo4J Scaling, 126
 węzły, 43, 119
 wiadra, 79, 93, 94
 domeny, 92, 93

widok w bazie relacyjnej, 46
 widoki zmaterializowane, 46, 47
 właściwości BASE, 69
 współbieżność, 20
 współczynnik replikacji, 71, 113
 współdzielenie, 53–56
 wydajność dostępu do danych w bazach
 NoSQL, 155
 wyjątek `NotInTransactionException`, 122

Z

zapytania
 dla magazynu klucz–wartość, 94, 95
 w Cassandra, 114–116
 w MongoDB, 103–105
 w Neo4J, 123–126
 zestawy replik w MongoDB, 101–103
 zmiana struktury agregacji, 137
 zmiany
 inkrementacyjne, 135
 schematów, 129–137
 schematu w bazach transakcyjnych, 129–133
 schematu w magazynach danych NoSQL,
 133–137
 w bazach grafowych, 136
 zmienna `WriteConcern`, 101

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



- 1. ZAREJESTRUJ SIĘ**
- 2. PREZENTUJ KSIĄŻKI**
- 3. ZBIERAJ PROWIZJĘ**

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

GRUPA WYDAWNICZA

 **Helion SA**

Zmień sposób myślenia na nierelacyjny!

Bazy danych NoSQL są coraz popularniejsze. Pozwalają na przechowywanie gigantycznych ilości informacji, a przy tym zachowują cały czas najwyższą wydajność. Sprawdzają się doskonale wszędzie tam, gdzie konieczne są: wysoka skalowalność systemu, elastyczne przechowywanie często zmieniających się danych lub inne specyficzne zastosowania. Jeżeli jesteś zagorzałym użytkownikiem relacyjnych baz danych SQL, jeżeli słyszałeś o bazach NoSQL i chcesz je poznać, trafiłeś na doskonałą książkę!

Stanowi ona świetne wprowadzenie do świata baz danych NoSQL. Na własnej skórze przekonasz się, w jakich zastosowaniach sprawdzą się one dobrze, a w jakich lepiej ich nie używać. W kolejnych rozdziałach poznasz stosowane modele danych oraz dowiesz się, co to jest map-reduce. Część druga książki została poświęcona konkretnym implementacjom — zaznajomisz się z bazami klucz-wartość, bazami dokumentów oraz bazami grafowymi. Sprawdź, które najlepiej rozwiążą Twoje problemy! Sięgnij po tę książkę i śmiało wkrocz w świat baz danych NoSQL!

PRAMOD J. SADALAGE — pracownik firmy ThoughtWorks, rozpoznawanej przez specjalistów branży IT na całym świecie. Doradza klientom i pomaga rozwiązywać problemy z zakresu baz danych i programowania. Pomysłodawca ewolucyjnego sposobu przechowywania informacji.

MARTIN FOWLER — legenda branży IT. Pracuje dla cenionej firmy ThoughtWorks i zajmuje się najlepszymi sposobami projektowania systemów informatycznych. W obszarze jego zainteresowań znajdują się również zagadnienia związane z refaktoryzacją kodu oraz pracą nad produktywnością programistów.

helion.pl
księgarnia internetowa

Nr katalogowy: 25880



Księgarnia internetowa:
<http://helion.pl>



Zamówienia telefoniczne:
0 801 339900



0 601 339900

Addison-Wesley
Pearson Education



Helion

Sprawdź najnowsze promocje:
• <http://helion.pl/promocje>
Książki najchętniej czytane:
• <http://helion.pl/bestsellery>
Zamów informacje o nowościach:
• <http://helion.pl/nowosci>

Helion SA
ul. Kosciuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
<http://helion.pl>

Dzięki tej książce:

- poznasz nierelacyjne modele przechowywania danych
- przekonasz się, gdzie najlepiej stosować bazy danych NoSQL
- nauczysz się korzystać z mechanizmu map-reduce
- wybierzesz bazę, która zaspokoi Twoje potrzeby
- zrozumiesz nierelacyjny świat baz danych NoSQL

sięgnij po **WIĘCEJ**



KOD KORZYŚCI

ISBN 978-83-246-9905-6



Cena: 39,00 zł