Dan Sullivan

NoSQL

PRZYJAZNY PRZEWODNIK

Helion

Tytuł oryginału: NoSQL for Mere Mortals®

Tłumaczenie: Jakub Hubisz

ISBN: 978-83-283-2488-6

Authorized translation from the English language edition, entitled: NOSQL FOR MERE MORTALS; ISBN 0134023218; by Dan Sullivan; published by Pearson Education, Inc, publishing as Addison-Wesley Professional.

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)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/nosqlp.zip

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/nosqlp
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

7
Terminologia baz dokumentów

Musimy mieć odwagę myśleć o tym, co „nie do pomyślenia".
Musimy nauczyć się eksplorować wszystkie opcje i możliwości,
które spotykają nas w skomplikowanym i gwałtownie zmieniającym się świecie.
— J. William Fulbright, były senator USA

Tematy omawiane w tym rozdziale:

- Terminy dotyczące dokumentów i kolekcji
- Typy partycji
- Modelowanie danych i przetwarzanie zapytań

W poprzednim rozdziale omówiliśmy podstawowe koncepcje związane z bazami dokumentów. W tym rozdziale skupimy się na zdefiniowaniu terminów powszechnie używanych w teorii i praktyce baz dokumentów.

Uwaga. Podobnie jak w przypadku innych rozdziałów dotyczących terminologii celem jest zdefiniowanie dokładnych opisów głównych terminów używanych w kontekście baz dokumentów. Część terminologii dotyczy również innych baz rozproszonych, podczas gdy niektóre pojęcia dotyczą wyłącznie baz dokumentów.

Pierwszy zestaw terminów jest związany z podstawowymi strukturami danych w bazie dokumentów. Powinieneś znać je z poprzedniego rozdziału, ale tutaj zostaną przedstawione w sposób bardziej formalny.

W drugim podrozdziale zdefiniowane zostaną terminy potrzebne podczas nauki architektury baz dokumentów. Terminy zdefiniowane w tym podrozdziale są używane

do ogólnego opisu baz dokumentów, ze szczególnym uwzględnieniem skalowania dużych baz. Będziesz się z nimi często spotykał podczas pracy z bazami NoSQL.

Ostatni podrozdział jest najbardziej heterogeniczny. Zawiera terminy związane z modelowaniem dokumentów i operacji. Terminy związane z modelowaniem są ogólnymi koncepcjami. W następnym rozdziale przedstawione zostaną bardziej szczegółowe wzorce projektowe. Na zakończenie podrozdziału omówię kilka ogólnych, ale ważnych terminów, które powinieneś znać, pracując z bazą dokumentów.

Terminy dotyczące dokumentów i kolekcji

Dokumenty i kolekcje to podstawowe struktury bazy dokumentów. Są analogiczne do rekordów i tabel w bazach relacyjnych. Nieformalne wprowadzenie do tych terminów w rozdziale 6. „Wprowadzenie do baz dokumentów" było wystarczające, aby przekazać podstawowe koncepcje baz dokumentów. Czas teraz na bardziej formalne definicje, które dokładniej opiszą zasady rządzące tymi strukturami.

Zdefiniowane zostaną następujące terminy:

- dokument,
- kolekcja,
- dokument osadzony,
- brak schematu,
- schemat polimorficzny.

Na końcu tego podrozdziału powinieneś dobrze rozumieć, w jaki sposób dokumenty są organizowane w kolekcje. Powinieneś również rozumieć ważne właściwości organizacji bazy dokumentów, które pozwalają na tworzenie elastycznych projektów. Jest to jeden z głównych powodów, dla których bazy dokumentów zyskały popularność pośród programistów.

Dokument

Dokument jest zestawem uporządkowanych par klucz-wartość. **Klucz-wartość** to struktura, która składa się z dwóch elementów, co oczywiste, klucza i wartości.

> **Uwaga.** Dla osób, które opuściły rozdziały poświęcone bazom klucz-wartość: **klucz** jest unikatowym identyfikatorem używanym do odnalezieniu wartości, **wartość** jest instancją dowolnego wspieranego typu danych, takiego jak ciąg znaków, numer, tablica lub lista.

Dokumenty: uporządkowany zestaw par klucz-wartość

Ponieważ dokument jest zestawem, posiada po jednej instancji każdego elementu. Elementy to pary klucz-wartość. Przykładowo poniżej znajduje się zestaw z trzema elementami: `'foo': 'a'`, `'bar': 'b'` i `'baz': 'c'`:

```
{ 'foo': 'a', 'bar': 'b', 'baz': 'c'}
```

Wystarczy drobna zmiana, aby zestaw przestał być zestawem (i stał się *torbą*):

```
{'foo': 'a', 'bar': 'b', 'baz': 'c', 'foo': 'a'}
```

Ta lista nie jest zestawem, ponieważ występują dwie instancje pary `'foo': 'a'`.

Zestawy nie są rozróżniane na podstawie kolejności. Poniższy zestaw:

```
{ 'foo': 'a', 'bar': 'b', 'baz': 'c'}
```

jest równoważny z:

```
{'baz': 'c', 'foo': 'a', 'bar': 'b'}
```

Jednak z punktu widzenia projektowania baz dokumentów są to różne dokumenty. Kolejność par klucz-wartość ma znaczenie w określaniu identyfikacji dokumentu. Dokument `{ 'foo': 'a', 'bar': 'b', 'baz': 'c'}` nie jest równoznaczny z dokumentem `{'baz': 'c', 'foo': 'a', 'bar': 'b'}`.

Klucze i typy wartości

Klucze to ciągi znaków. Niektóre bazy klucz-wartość wspierają bardziej rozbudowany zestaw typów danych dla kluczy, więc bazy dokumentów mogłyby również wspierać wiele typów danych dla kluczy.

Wartości mogą być różnego typu. Jak możesz się spodziewać, bazy dokumentów wspierają wartości liczbowe i ciągi znaków. Wspierają również bardziej strukturalne typy danych, takie jak tablice i inne dokumenty.

Tablice są użyteczne, kiedy musisz śledzić wiele instancji wartości, a wszystkie wartości są tego samego typu. Jeżeli na przykład musisz zamodelować pracownika i listę jego projektów, możesz skorzystać z dokumentu takiego jak poniżej:

```
{ 'nazwaPracownika' : 'Julia Kowalska',
  'dzial' : 'Wytwarzanie oprogramowania'
  'dataRozpoczecia' : '10/02/2010',
  'kodyZakonczonychProjektow' : [ 189847, 187731, 176533, 154812]
}
```

Klucz `kodyZakonczonychProjektow` zawiera listę numerów projektów. Wszystkie kody projektów to numery, dlatego użycie tablicy jest poprawne.

Ewentualnie, jeżeli wraz z danymi pracowników chcesz przechować lub osadzić więcej informacji o projektach, możesz w ramach dokumentu pracownika zamieścić inny dokument. Taki dokument mógłby wyglądać tak:

```
{ 'nazwaPracownika' : 'Julia Kowalska',
  'dzial' : 'Wytwarzanie oprogramowania',
  'dataRozpoczecia' : '10/02/2010',
  'zakonczoneProjekty' : {
  { 'kodProjektu' : 189847,
    'nazwaProjektu' : 'System rekomendacji produktów',
    'menadzerProjektu' : 'Jagoda Dulska' },
  { 'kodProjektu' : 187731,
    'nazwaProjektu' : 'Wymiana danych finansowych, wersja 3',
    'menadzerProjektu' : 'Jakub Rosiewicz'},
  { 'kodProjektu': 176533,
    'nazwaProjektu' : 'Uwierzytelnianie klientów',
    'menadzerProjektu' : 'Michał Krawiec'},
  { 'kodProjektu': 154812,
    'nazwaProjektu' : 'Miesięczny raport sprzedaży',
    'menadzerProjektu': 'Bożena Rondel'}
}
```

Podsumowując: dokumenty to uporządkowane zestawy par klucz-wartość. Klucze są wykorzystywane do odnajdywania konkretnych wartości. Wartości mogą być albo typu prostego, albo strukturalnego.

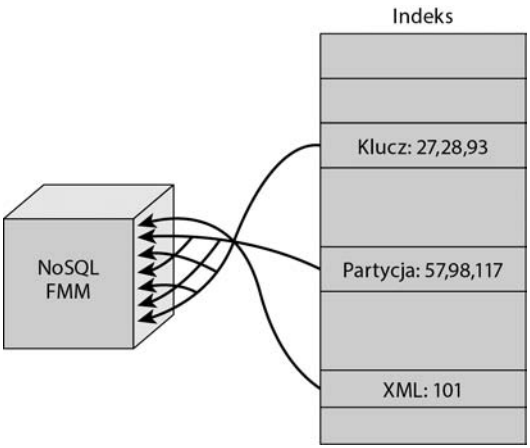
Kolekcja

Kolekcja to grupa dokumentów. Dokumenty w ramach kolekcji są zazwyczaj spokrewnione z tą samą encją obiektu, taką jak pracownicy, produkty, zalogowane zdarzenia lub profile użytkowników. Możliwe jest przechowywanie w kolekcji niezwiązanych ze sobą dokumentów, ale nie jest to polecane.

Mówiąc najbardziej ogólnie: kolekcje pozwalają operować na grupach spokrewnionych dokumentów. Jeżeli utrzymujesz kolekcję informacji o pracownikach, możesz przejrzeć wszystkie rekordy w kolekcji i odnaleźć konkretnych pracowników, na przykład takich, którzy rozpoczęli pracę przed 1 stycznia 2011 roku. Jeżeli masz dużą liczbę pracowników, takie przeszukiwanie może być niewydajne, ponieważ musiałbyś porównać dane każdego pracownika z kryteriami wyszukiwania.

Oprócz możliwości łatwego wykonywania operacji na grupach dokumentów kolekcje wspierają dodatkowe struktury, które sprawiają, że te operacje są bardziej wydajne. Przykładowo bardziej wydajne podejście do skanowania wszystkich dokumentów w kolekcji to wykorzystanie indeksów. Indeksy w kolekcjach są jak indeksy na końcu książki: zestaw informacji mapujących jeden atrybut, taki jak hasło **klucz**, do powiązanych z nim informacji, takich jak lista stron (patrz rysunek 7.1).

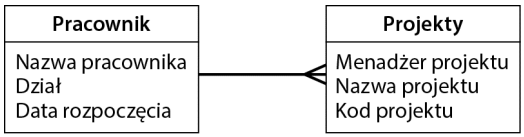
Kolekcje to grupy podobnych dokumentów, dzięki którym łatwo dotrzeć do wszystkich dokumentów w grupie lub wykonywać na nich operacje. Kolekcje wspierają dodatkowe struktury, takie jak indeksy, które poprawiają wydajność operacji na tych grupach dokumentów.



Rysunek 7.1. Indeksy mapują atrybuty, takie jak hasła, do powiązanych z nimi informacji, takich jak numery storn. Korzystanie z indeksu jest szybsze niż skanowanie całej książki

Dokument osadzony

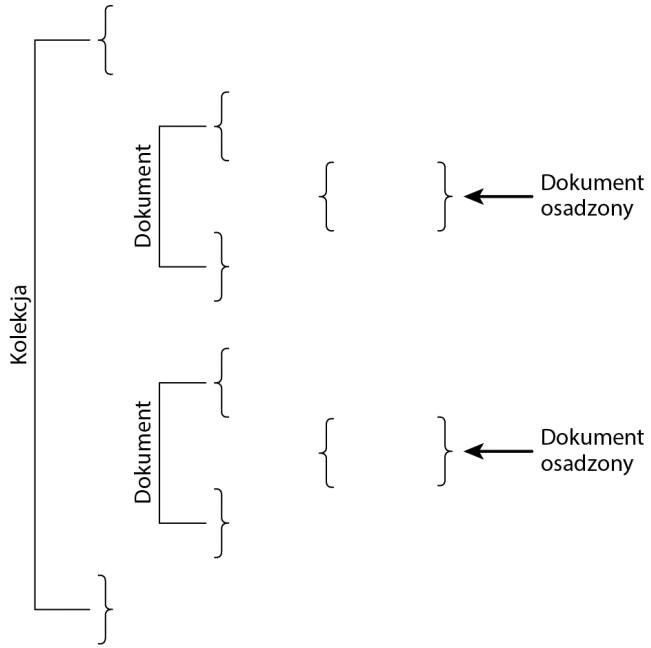
Jedną z zalet baz dokumentów jest to, że pozwalają programistom przechowywać spokrewnione dane w sposób bardziej elastyczny niż w przypadku baz relacyjnych. Gdybyś miał zamodelować pracowników i projekty, nad którymi pracują, prawdopodobnie stworzyłbyś dwie tabele: jedną przechowującą informacje o pracownikach i drugą przechowującą informacje o projektach (patrz rysunek 7.2).



Rysunek 7.2. Modele relacyjne dzielą dane różnych encji na osobne tabele. Wymaga to sprawdzania informacji w obu tabelach z wykorzystaniem procesu złączania

Dokument osadzony pozwala użytkownikom bazy dokumentów na przechowywanie spokrewnionych danych w jednym dokumencie. Dzięki temu baza dokumentów może uniknąć procesu złączania, polegającego na tym, że dane z jednej tabeli są używane do odnalezienia danych w innej tabeli (klucz obcy).

Złączenie dwóch dużych tabel może potencjalnie być czasochłonne i wymagać znacznej liczby odczytów z dysku. Dokumenty osadzone pozwalają na wspólne przechowywanie spokrewnionych danych. Kiedy dokument jest odczytywany z dysku, zarówno dane główne, jak i dane powiązane są odczytywane bez potrzeby ich złączania. Rysunek 7.3 przedstawia dokumenty osadzone w innym dokumencie.



Rysunek 7.3. Dokumenty osadzone to dokumenty zapisane w innym dokumencie. Osadzanie jest używane do efektywnego przechowywania i pobierania danych, które często są wykorzystywane razem

Dokumenty osadzone to dokumenty zapisane w innym dokumencie. Są używane do podnoszenia wydajności bazy przez wspólne przechowywanie danych, które często są wykorzystywane razem.

Brak schematu

Bazy dokumentów nie wymagają od osób modelujących dane formalnej specyfikacji struktury dokumentów. Formalna specyfikacja struktury nazywana jest **schematem**. Bazy relacyjne wymagają schematu. Schemat zawiera zazwyczaj specyfikacje:

- tabel,
- kolumn,
- kluczy głównych,
- kluczy obcych,
- ograniczeń.

Pozwalają one systemowi zarządzania bazą relacyjną na zarządzanie danymi w bazie. Pomagają również wykrywać błędy podczas dodawania danych do bazy. Jeżeli na przykład ktoś spróbuje dodać ciąg znaków w miejscu, w którym wymagana jest liczba, system zarządzania bazą danych zgłosi błąd.

Ograniczenia to reguły opisujące, jakie dane lub relacje pomiędzy danymi są dozwolone. W schemacie można wskazać, że kolumna zawsze musi posiadać wartość i nigdy nie może być pusta.

Wskazówka. Pusta kolumna w bazie relacyjnej przyjmuje wartość `NULL`.

Bazy dokumentów nie wymagają specyfikacji przed rozpoczęciem dodawania dokumentów do bazy. Z tego powodu bazy dokumentów są nazywane bazami bez schematu. Bazy bez schematu różnią się od baz relacyjnych w dwóch zasadniczych sprawach:

- dają większą elastyczność,
- wymagają większej odpowiedzialności.

Brak schematu oznacza większą elastyczność

W bazie bez schematu programiści i aplikacje mogą do dokumentów dodawać nowe pary klucz-wartość w dowolnym momencie. Po stworzeniu kolekcji możesz dodawać do niej dokumenty. Nie ma potrzeby informowania bazy dokumentów o strukturze dokumentów. Sama struktura dokumentów będzie często różna w różnych dokumentach w kolekcji. Poniższe dwa dokumenty:

```
{ 'nazwaPracownika' : 'Julia Kowalska',
  'dzial' : 'Wytwarzanie oprogramowania'
  'dataRozpoczecia' : '10/02/2010',
  'kodyZakonczonychProjektow' : [ 189847, 187731, 176533, 154812]
}
```

i

```
{ 'nazwaPracownika' : 'Robert Lucek',
  'dzial' : 'Finanse'
  'dataRozpoczecia' : '21/05/2009',
  'certyfikaty' : 'CPA'
}
```

opisują pracowników, ale pierwszy jest przystosowany dla osoby z działu wytwarzania oprogramowania, a drugi został zaprojektowany z myślą o pracowniku działu finansów.

Te dokumenty i ich wariancje mogą być po prostu dodawane do kolekcji. Nie ma potrzeby specyfikowania, że niektóre dokumenty będą posiadały klucz `kodyZakonczonychProjektow`, a inne klucz `certyfikaty`. Nie ma również potrzeby określania, że niektóre wartości będą ciągami znaków, a inne tablicami.

Uwaga. Baza dokumentów czerpie potrzebne jej informacje ze struktury dokumentów w kolekcji, nie z osobnej specyfikacji struktury.

Brak schematu oznacza większą odpowiedzialność

Bazy bez schematu są swego rodzaju bronią obosieczną. Z jednej strony elastyczność pracy bez schematu pozwala łatwo radzić sobie z różnicami w strukturach dokumentów. Z drugiej jednak strony baza dokumentów nie może narzucać reguł opartych na strukturze. Ponieważ nie ma możliwości określenia, że para klucz-wartość powinna zawsze istnieć w dokumencie, system zarządzania bazą dokumentów nie będzie tego sprawdzał.

Jeżeli system zarządzania bazą nie zapewnia reguł dotyczących danych, co ma się tym zająć? Odpowiedź brzmi — Twój kod.

Wskazówka. Wyjątkiem od tej reguły jest wykorzystanie unikatowych identyfikatorów. Jeżeli spróbujesz wstawić dokument bez unikatowego identyfikatora, baza dokumentów prawdopodobnie go uzupełni. Szczegółowe informacje na ten temat znajdziesz w dokumentacji swojej bazy.

Część kodu Twojej aplikacji powinna być dedykowana sprawdzaniu reguł dotyczących struktury danych. Jeżeli w dokumencie pracownika zawsze powinna być zawarta jego nazwa, Twój kod dodający pracowników powinien to sprawdzać podczas umieszczania dokumentów w bazie. Jest to prosty przykład walidacji danych, jednak nie wszystkie przypadki są tak proste.

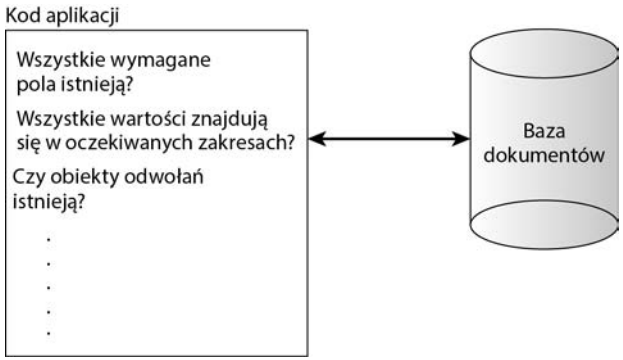
Z czasem klucze i wartości przechowywane w dokumentach w kolekcji mogą się zmieniać. Być może zacząłeś zbierać informacje o certyfikatach pracowników w zeszłym roku. Wszyscy pracownicy, którzy od tamtego czasu zostali dodani do bazy, mogą mieć klucz dotyczący certyfikatów. Pracownicy, którzy zostali zaktualizowani, również mogą mieć klucz i wartość dla certyfikatów. Pozostałe dokumenty nie będą miały tego klucza.

W tej sytuacji to kod, który używa i przetwarza dokumenty pracowników (nie kod dodający dokumenty), musi sprawdzać poprawność struktur danych albo przynajmniej obsłużyć przypadek, w którym oczekiwany klucz nie istnieje (patrz rysunek 7.4).

Bazy bez schematów nie wymagają formalnej specyfikacji struktury. Potrzebne informacje są pozyskiwane z dokumentów w kolekcji. To pozwala uzyskać większą elastyczność niż w bazach wymagających schematu, ale daje mniejsze możliwości automatycznego sprawdzania danych i integralności dokumentów.

Schemat polimorficzny

Innym terminem, z jakim możesz się spotkać w kontekście baz dokumentów, jest schemat polimorficzny. Może się wydawać dziwne, że bazy mogą nie mieć schematu (**brak schematu**) i jednocześnie mieć wiele schematów (**schemat polimorficzny**). Jest to jednak całkiem



Rysunek 7.4. Kod walidujący dane i obsługujący błędy jest używany w aplikacjach po to, aby zastąpić automatyczną, opartą na schemacie walidację

logiczne, jeżeli dostrzeżesz różnicę pomiędzy formalną specyfikacją struktury i strukturą wynikającą z dokumentów w kolekcji.

Wskazówka. Jeszcze raz. Bazy dokumentów nie mają schematu, ponieważ nie musisz specyfikować formalnej definicji struktury dokumentów, kluczy i wartości.

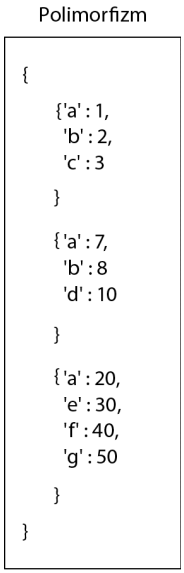
Baza dokumentów jest polimorficzna, ponieważ dokumenty istniejące w kolekcjach mogą mieć wiele różnych form (patrz rysunek 7.5).

Typy partycji

Partycjonowanie to słowo, które bardzo często jest używane w świecie NoSQL — być może za często.

W rozdziale 2. omówiliśmy teorię CAP — jak być może pamiętasz, teoria ta opisuje ograniczenia spójności, dostępności i tolerancji dla partycjonowania. W tym kontekście słowo **partycjonowanie** odnosi się do partycjonowania lub separowania sieci na odrębne części, które są niedostępne dla siebie nawzajem.

Jest to ważna koncepcja dla wszystkich rozproszonych baz danych, ale w kontekście baz dokumentów partycjonowanie oznacza coś innego. Kiedy podczas dyskusji na temat baz dokumentów ludzie używają pojęcia **partycjonowanie**, prawdopodobnie chodzi im o dzielenie bazy i dystrybuowanie różnych jej części na różne serwery.



Rysunek 7.5. Brak schematu oznacza, że nie ma formalnej definicji struktury. Schemat polimorficzny oznacza, że istnieje wiele struktur dokumentów wynikających z dokumentów występujących w kolekcji

Są dwa typy partycjonowania bazy: partycjonowanie pionowe i partycjonowanie poziome.

Należy zróżnicować znaczenie pojęcia **partycjonowanie** na podstawie kontekstu, w jakim zostanie użyte (patrz rysunek 7.6).

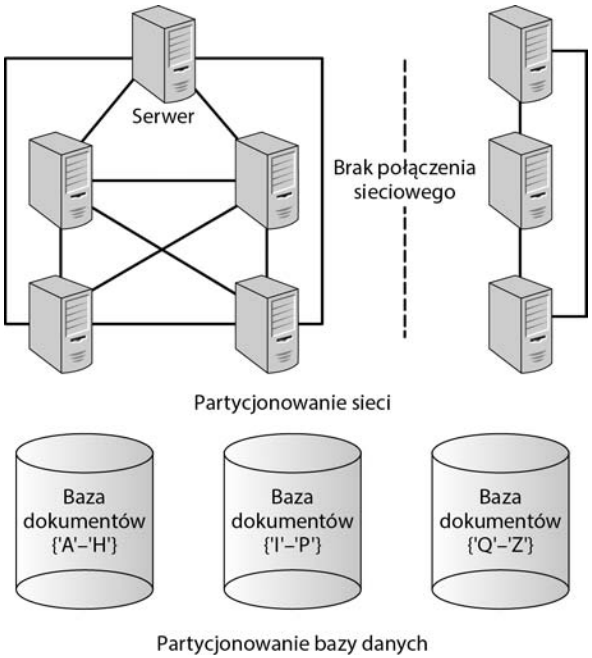
Partycjonowanie pionowe

Partycjonowanie pionowe jest techniką poprawiania wydajności bazy danych przez rozdzielenie kolumn tabeli relacyjnej do kilku odrębnych tabel (patrzy rysunek 7.7).

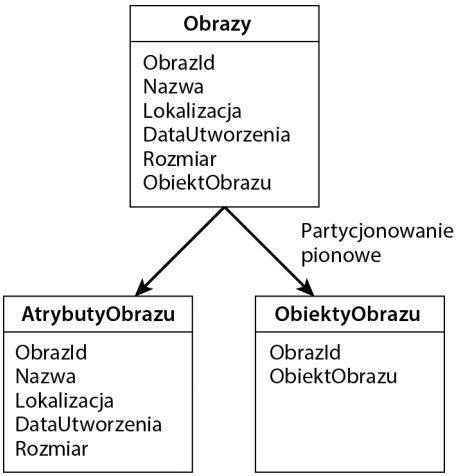
Ta technika jest szczególnie użyteczna, kiedy masz kolumny, które są często pobierane, podczas gdy inne są pobierane rzadziej. Rozważmy tabelę z obrazami i atrybutami tych obrazów, takimi jak nazwa, lokalizacja czy data utworzenia obrazu. Tabela może być wykorzystywana w aplikacji pozwalającej wyszukiwać obrazy na podstawie ich charakterystyk.

Ktoś może poszukiwać zdjęć Paryża zrobionych w ciągu ostatnich trzech miesięcy. System zarządzania bazą danych do odnalezienia rekordów spełniających kryteria wyszukiwania najprawdopodobniej wykorzystałby indeks. Jeżeli aplikacja w wynikach zwraca tylko listę atrybutów i przed wyświetleniem obrazu czeka, aż użytkownik wybierze konkretny rekord, nie ma potrzeby, aby z bazy pobierać obraz razem z atrybutami.

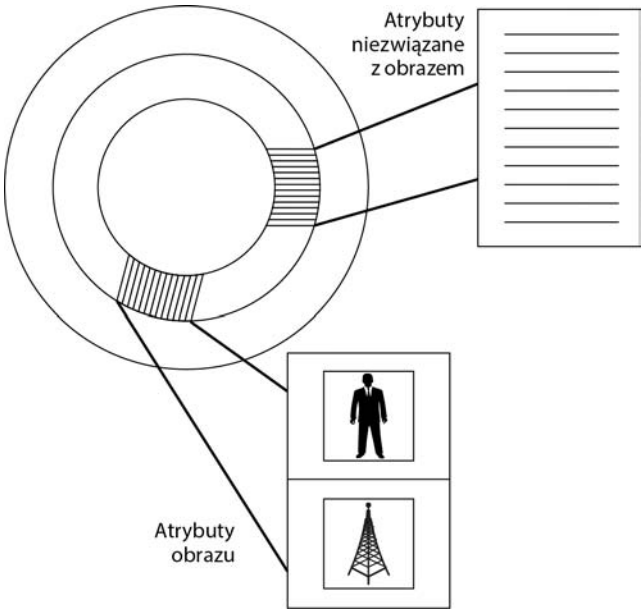
Jeżeli atrybuty obrazu i sam obraz byłyby przechowywane w tej samej tabeli, to z powodu rozkładu danych na dysku próba odczytania atrybutów mogłaby się skończyć odczytaniem także obrazu. Dzięki podzieleniu tabeli obrazu na tabelę zawierającą atrybuty i tabelę zawierającą obraz baza może efektywnie pobierać dane dla aplikacji (patrz rysunek 7.8).



Rysunek 7.6. Termin partycjonowanie ma wiele znaczeń, które są zależne od kontekstu, innego na przykład w dyskusji na temat sieci i w dyskusji na temat baz danych



Rysunek 7.7. Partycjonowanie pionowe jest zazwyczaj wykorzystywane w tabelach relacyjnych, ponieważ mają one ustaloną strukturę



Rysunek 7.8. Rozdzielanie kolumn na różne tabele może poprawić efektywność odczytów, ponieważ dane, które nie będą potrzebne (na przykład obiekt obrazu), nie są pobierane wraz z danymi, które są potrzebne (na przykład atrybuty obrazu)

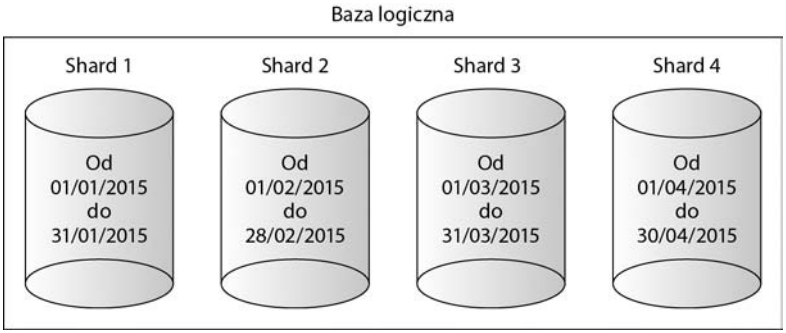
Partycjonowanie pionowe jest częściej wykorzystywane w bazach relacyjnych niż w bazach dokumentów. Są metody implementowania partycjonowania pionowego w bazach nierelacyjnych, ale partycjonowanie poziome (lub sharding) jest bardziej powszechne.

Uwaga. Przykład shardingu znajdziesz w artykule autorstwa J. Kaura i innych *A New and Improved Vertical Partitioning Scheme for Non-Relational Databases Using Greedy Method*, opublikowanym w numerze 8 „International Journal of Advanced Research in Computer and Communication Engineering" (sierpień 2013).

Partycjonowanie poziome, czyli sharding

Partycjonowanie poziome to proces dzielenia bazy danych na poziomie dokumentów w bazie dokumentów lub na poziomie wierszy w bazie relacyjnej.

Takie części bazy danych, nazywane shardami, są przechowywane na odrębnych serwerach (poziome partycjonowanie bazy dokumentów jest też nazywane **shardingiem**). Jeżeli baza zostanie skonfigurowana do replikowania danych, pojedynczy shard może być przechowywany na wielu serwerach. Niezależnie od tego, czy dane są replikowane, czy nie, pojedynczy serwer w klastrze będzie przechowywał tylko jeden shard (patrz rysunek 7.9).



Rysunek 7.9. Partycjonowanie poziome dzieli bazę na poziomie dokumentów lub wierszy i dystrybuuje części bazy, nazywane shardami, na różne serwery. Kiedy klaster zaimplementuje replikację, pojedynczy shard będzie dostępny na wielu serwerach

Sharding ma wiele zalet w przypadku implementacji dużych baz dokumentów. Znaczna liczba użytkowników lub inne pracochłonne zadania mogą mocno obciążać dostępne procesory, pamięć i łącze. Jednym ze sposobów radzenia sobie z taką sytuacją jest wykorzystanie większego serwera, z większą liczbą procesorów i ilością pamięci oraz lepszym łączem.

To rozwiązanie, nazywane skalowaniem pionowym, może wymagać znacznie większych nakładów finansowych i większej ilości czasu niż sharding. Dodatkowe serwery mogą być dodawane do klastra wraz z powiększaniem się bazy dokumentów. Istniejące serwery nie są zastępowane i pozostają w ciągłym użytku.

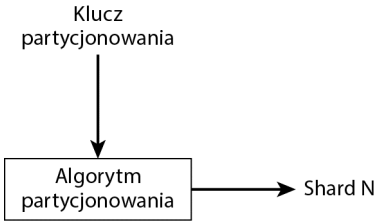
Aby zaimplementować sharding, projektanci baz danych muszą wybrać klucz i metodę partycjonowania. Tematy te zostaną omówione w następnych podrozdziałach.

Dzielenie danych za pomocą klucza partycjonowania

Klucz partycjonowania to jeden klucz lub więcej kluczy istniejących we wszystkich dokumentach w kolekcji i wykorzystywanych do podzielenia dokumentów. Klucz partycjonowania może być dowolnym atomowym polem dokumentu:

- unikatowym identyfikatorem dokumentu,
- nazwą,
- datą (na przykład datą utworzenia),
- kategorią lub typem,
- regionem geograficznym.

Klucz partycjonowania określa, jakie wartości mają być używane podczas grupowania dokumentów w shardy. Algorytm partycjonowania wykorzystuje klucz partycjonowania jako wsad i determinuje odpowiedni shard dla klucza (patrz rysunek 7.10).



Rysunek 7.10. Klucze partycjonowania to dane wsadowe dla algorytmu partycjonowania zwracającego shard

Uwaga. Pomimo tego, że bazy dokumentów nie mają schematu, niektóre elementy tych baz są podobne do schematu baz relacyjnych. Wykorzystanie indeksów jest jednym z takich podobieństw. Indeksy są częścią fizycznego modelu danych bazy relacyjnej, co oznacza, że w bazie danych istnieje struktura implementująca indeks. Indeksy są częścią schematu baz relacyjnych. W bazach bez schematu, takich jak bazy dokumentów, mogą występować obiekty podobne do obiektów schematu, takie jak indeksy. Indeksy pomagają poprawić szybkość operacji odczytu i są użyteczne podczas implementowania shardingu. Ponieważ wszystkie dokumenty w kolekcji muszą być umieszczone w shardzie, ważne jest, aby wszystkie miały klucz partycjonowania.

Dystrybuowanie danych przy pomocy algorytmu partycjonowania

Jest wiele sposobów na poziome partycjonowanie danych, włączając w to:

- zakresy,
- hasze,
- listy.

Partycjonowanie zakresami jest użyteczne, kiedy masz uporządkowany zestaw wartości dla kluczy partycjonowania, taki jak daty lub liczby. Przykładowo jeżeli wszystkie dokumenty w kolekcji mają pole z datą utworzenia, można wykorzystać to pole do partycjonowania danych na miesięczne shardy. Dokumenty stworzone pomiędzy 1 stycznia 2015 roku i 31 stycznia 2015 roku zostałyby zapisane w jednym shardzie, podczas gdy dokumenty stworzone pomiędzy 1 lutego 2015 roku i 28 lutego 2015 roku zostałyby zapisane w innym.

Uwaga. Systemy analizy biznesowej, które tworzą raporty dla określonego zakresu dat — na przykład raport porównujący sprzedaż z tego miesiąca ze sprzedażą z poprzedniego miesiąca — zazwyczaj wykorzystują partycjonowanie na podstawie czasu.

Partycjonowanie za pomocą haszy wykorzystuje funkcję haszującą do określenia, gdzie umieścić dokument. Funkcje haszujące zostały zaprojektowane tak, aby generowały wartości równomiernie rozłożone w ramach zakresu wartości funkcji haszującej. Jeżeli na przykład masz klaster złożony z ośmiu serwerów, a Twoja funkcja haszująca generuje wartości pomiędzy 1 a 8, powinieneś mieć prawie równą liczbę dokumentów na każdym z ośmiu serwerów.

Partycjonowanie na podstawie list wykorzystuje zestaw wartości do określenia lokalizacji danych. Możesz wyobrazić sobie bazę produktów z kilkoma typami, między innymi elektronika, drobne urządzenia, artykuły AGD, książki i ubrania. Te typy produktów mogą być wykorzystane jako klucz partycjonowania.

Jeżeli potrzebujesz więcej partycji, możesz połączyć typy produktów z innym polem, takim jak na przykład region sprzedaży, który jako wartości może przyjmować: północny wschód, południowy wschód, środkowy zachód, północny zachód i południowy zachód. Każdy z pięciu typów produktów może być wykorzystany z każdym z pięciu regionów sprzedaży, co daje dwadzieścia pięć możliwych partycji, w tym:

- elektronika-północny-wschód,
- elektronika-południowy-wschód,
- elektronika-środkowy-zachód,
- elektronika-południowy-zachód,
- elektronika-północny-zachód,
- urządzenia-północny-wschód,
- urządzenia-południowy-wschód,
- urządzenia-środkowy-zachód,
- i tak dalej…

Partycjonowanie to podstawowy proces pozwalający na skalowanie baz dokumentów wraz z rosnącym zapotrzebowaniem aplikacji, które mają dużą liczbę użytkowników lub wykonują dużo operacji. Partycjonowanie pionowe jest możliwe w przypadku baz dokumentów. Partycjonowanie poziome jest często stosowane.

Programiści korzystający z baz dokumentów mogą wybierać klucze używane podczas partycjonowania. Ale to programiści systemów zarządzania bazami dokumentów wybierają algorytmy partycjonowania wykorzystywane w bazie.

Ostatni podrozdział tego rozdziału wprowadza kilka nowych terminów niepasujących do żadnej z poprzednich kategorii. Powinieneś jednak zrozumieć te terminy, zanim przejdziemy do omawiania modelowania baz dokumentów w rozdziale 8.

Modelowanie danych i przetwarzanie zapytań

Bazy dokumentów są elastyczne. Mogą przechowywać szeroki wachlarz typów dokumentów i ich wariancji w ramach kolekcji dokumentów. Gdybyś miał teraz usiąść i zacząć projektować bazę dokumentów, prawdopodobnie zacząłbyś od listy zapytań, które chciałbyś uruchamiać w bazie (a przynajmniej powinieneś od tego zacząć). Gdybyś projektował bazę relacyjną, zacząłbyś prawdopodobnie od przemyślenia encji, które masz zamodelować, i ich wzajemnych relacji.

Kiedy już z grubsza zrozumiałbyś encje i relacje, zająłbyś się prawdopodobnie czymś, co nazywa się normalizacją. Jeżeli napotkasz problemy wydajnościowe z bazą, możesz zająć się procesem o nazwie denormalizacja. Ten proces byłby kierowany (przynajmniej do pewnego stopnia) informacjami uzyskanymi na podstawie niewydajnych zapytań z procesora zapytań.

Powinieneś znać zagadnienia normalizacji i denormalizacji, ponieważ najprawdopodobniej napotkasz je podczas pracy z bazami dokumentów. W przypadku baz dokumentów proces jest mniej formalny niż w przypadku baz relacyjnych, dlatego poniższe wyjaśnienia będą znacznie prostsze niż te, które znajdziesz w książkach poświęconych bazom relacyjnym.

Bazy dokumentów również implementują procesory zapytań, które próbują znaleźć optymalną sekwencję kroków prowadzących do pobrania szukanych danych.

Normalizacja

Normalizacja baz danych jest procesem organizacji danych w tabelach w taki sposób, aby ograniczyć potencjalne anomalie danych. **Anomalia** to niespójność danych. Załóżmy na przykład, że w swojej bazie przechowujesz tabelę przedstawioną w tabeli 7.1. Użytkownik odpytuje tę tabelę, szukając adresu klienta o nazwie Jolanta Wasilewska. Jakie adresy powinny zostać zwrócone?

Tabela 7.1. Adresy klientów

Numer zamówienia	Nazwa klienta	Adres klienta
9837373	Jolanta Wasilewska	ul. Parkowa 50, 81-549 Gdynia
9837374	Robert Daniel	os. Młodych, 58-200 Dzierżoniów
9837375	Tomasz Kowal	ul. Kościuszki 12/1, 63-900 Rawicz
9837376	Jolanta Wasilewska	ul. Stanisława Maczka 60/2, 81-417 Gdynia

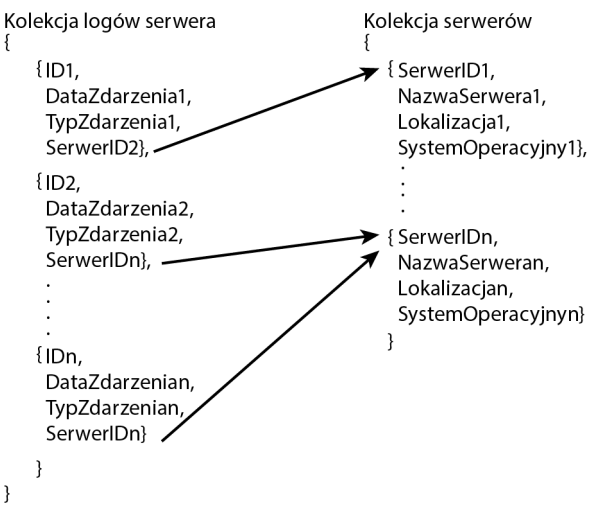
Zapytanie może zwrócić wartość *ul. Parkowa 50, 81-549 Gdynia* lub *ul. Stanisława Maczka 60/2, 81-417 Gdynia*, lub obie naraz. To możliwe, że Jolanta Wasilewska mieszka pod obydwoma adresami, ale również możliwe jest, że jeden z adresów jest aktualny, a drugi nie. W tabeli 7.1 nie ma na ten temat informacji.

Normalizacja redukuje ilość nadmiarowych danych w bazie. Nazwy klientów i ich adresy nie byłyby powtarzane przy każdym zamówieniu, tylko zostałyby umieszczone w osobnej tabeli. Zarówno do klienta, jak i do adresu mogłyby zostać przypisane dodatkowe atrybuty. W szczególności tabela z adresem mogłaby mieć flagę aktywnego adresu, pokazującą, który z wielu adresów jest aktualny.

Jest kilka zasad normalizacji baz danych. Bazy mogą występować w różnych formach normalnych, w zależności od tego, ile zasad normalizacyjnych jest przestrzeganych. Najczęściej projektanci dążą do tak zwanej trzeciej postaci normalnej, która oznacza spełnienie pierwszych trzech zasad normalizacji[1].

Normalizacja jest czasami używana do opisania sposobu projektowania dokumentów. Kiedy projektanci używają wielu kolekcji do przechowywania powiązanych ze sobą danych, mówi się, że dane są znormalizowane.

Dokumenty znormalizowane implikują to, że będziesz miał referencje do innych dokumentów, w których zapisane są dodatkowe dokumenty. Przykładowo dokument z logami serwera może posiadać pole z identyfikatorem serwera, który wygenerował te dane. Kolekcja dokumentów dla serwerów posiadałaby dodatkowe informacje o każdym serwerze, dzięki czemu te dane nie musiałyby być powtarzane w każdym dokumencie logów (patrz rysunek 7.11).



Rysunek 7.11. Dokumenty znormalizowane redukują nadmiarowość danych, przechowując referencje do pojedynczej kopii danych, zamiast powtarzać je w każdym dokumencie

[1] Wprowadzenie do normalizacji znajdziesz w artykule *A Simple Guide to Five Normal Forms in Relational Database Theory* Williama Kenta z września 1982 roku, *http://www.bkent.net/Doc/simple5.htm*.

Denormalizacja

Normalizacja pomaga uniknąć anomalii danych, ale może powodować problemy wydajnościowe. Może się tak zdarzyć, kiedy pobierasz dane z dwóch tabel lub więcej. Proces ten jest nazywany **złączaniem** i jest podstawową operacją w bazach relacyjnych. Dużo wysiłku zostało włożone w stworzenie efektywnych sposobów łączenia danych. Administratorzy baz danych i osoby modelujące dane mogą spędzać dużo czasu, próbując poprawić wydajność operacji złączania. Nie zawsze to się udaje.

Projektowanie baz danych wymaga kompromisów. Możesz zaprojektować wysoko znormalizowaną bazę danych bez żadnych zduplikowanych danych, ale za to mało wydajną. W takim wypadku wielu projektantów zwraca się w kierunku denormalizacji.

Jak sugeruje nazwa, denormalizacja jest odwrotnością normalizacji — wprowadza duplikację danych. Możesz się zastanawiać, po co wprowadzać nadmiarowe dane. Może to powodować anomalie danych, takie jak przedstawione w tabeli 7.1. Oczywiście wymaga to też większej powierzchni przechowywania. Ryzykuje się występowanie anomalii danych dlatego, że denormalizacja może znacząco poprawić wydajność.

Kiedy dane są zdenormalizowane, nie ma potrzeby odczytywania danych z wielu tabel i wykonywania złączeń z wielu kolekcji. Dane są pobierane z jednej kolekcji lub dokumentu. Może to być znacznie szybsze niż pobieranie z wielu kolekcji, w szczególności jeżeli dostępne są indeksy.

Procesor zapytań

Pobieranie danych z bazy dokumentów jest bardziej skomplikowane niż pobieranie ich z bazy klucz-wartość. Pamiętaj, że z bazy klucz-wartość, jeżeli masz klucz, możesz pobrać przypisaną do niego wartość.

Bazy dokumentów oferują większe możliwości pobierania danych. Przykładowo możesz pobrać dokumenty stworzone przed wskazaną datą, dokumenty, które są określonego typu, dokumenty zawierające ciąg znaków **bieganie długodystansowe** w opisie produktu, lub nawet kombinację wszystkich powyższych.

Procesor zapytań jest ważną częścią systemu zarządzania bazą danych. Przyjmuje zapytania oraz dane o dokumentach i kolekcjach w bazie, a następnie zwraca sekwencję operacji pobierających wybrane dane.

Bazy klucz-wartość nie potrzebują procesora zapytań; pobierają wartości za pomocą klucza. Nie ma potrzeby analizowania poleceń logicznych, takich jak poniżej:

```
(dataUtworzenia > '01/01/2015') AND (typProduktu = 'elektronika')
```

Jeżeli istnieje kilka warunków pobierania danych, procesor zapytań musi podjąć decyzję, na przykład które kryteria powinny zostać zaaplikowane jako pierwsze. Czy najpierw powinien znaleźć dokumenty z datą utworzenia po 1 stycznia 2015 roku czy wszystkie produkty elektroniczne?

Jeżeli dokumentów utworzonych po 1 stycznia 2015 roku jest mniej niż dokumentów dla urządzeń elektronicznych, sensowne będzie pobranie dokumentów po dacie utworzenia,

ponieważ wyników będzie mniej. To oznacza, że drugie kryterium będzie aplikowane do mniejszego zbioru dokumentów.

Jest to prosty przykład decyzji, jakie musi podejmować procesor zapytań podczas budowania planu zapytania.

Podsumowanie

Bazy dokumentów posiadają terminologię specyficzną dla tego typu bazy NoSQL, ale współdzielą również część pojęć z innymi bazami NoSQL i bazami relacyjnymi. Dokumenty są równoznaczne z wierszami w bazach relacyjnych, a kolekcje z tabelami. Partycjonowanie, w szczególności sharding, jest używane w bazach dokumentów do poprawiania wydajności i polega na dzieleniu dużych baz danych na wiele serwerów. Normalizacja, denormalizacja i procesory zapytań odgrywają znaczącą rolę w kontekście wydajności bazy dokumentów.

W rozdziale 8. zagłębimy się w problemy związane z projektowaniem konkretnych baz dokumentów.

Pytania kontrolne

1. Opisz, w jaki sposób dokumenty są analogiczne do wierszy w bazach relacyjnych.
2. Opisz, w jaki sposób kolekcje są analogiczne do tabel w bazach relacyjnych.
3. Zdefiniuj schemat.
4. Dlaczego bazy dokumentów są uznawane za nieposiadające schematu?
5. Dlaczego bazy dokumentów są uznawane za polimorficzne?
6. Czym partycjonowanie pionowe różni się od partycjonowania poziomego?
7. Czym jest klucz partycjonowania?
8. Jaki jest cel algorytmu partycjonowania w shardingu?
9. Czym jest normalizacja?
10. Po co denormalizować kolekcje w bazie dokumentów?

Odniesienia

Brewer Eric, *CAP Twelve Years Later: How the 'Rules' Have Changed*, „Computer", 2 (luty 2012), s. 23 – 29.

Brewer Eric, *Towards Robust Distributed Systems*, „PODC", 7, 2000.

Chodorow Kristina, *50 Tips and Tricks for MongoDB Developers*, O'Reilly Media, Inc., Sebastopol 2011.

Chodorow Kristina, *MongoDB: The Definitive Guide*, O'Reilly Media, Inc., Sebastopol 2013.

Copeland Rick, *MongoDB Applied Design Patterns*, O'Reilly Media, Inc., Sebastopol 2013.

Dokumentacja Couchbase, *http://docs.couchbase.com/*.

Han Jing i in., *Survey on NoSQL Database*, Pervasive computing and applications (ICPCA), VI Międzynarodowa Konferencja nt. IEEE, 2011.

Manual MongoDB 3.2, *http://docs.mongodb.org/manual/*.

O'Higgins Niall, *MongoDB and Python: Patterns and Processes for the Popular Document-Oriented Database*, O'Reilly Media, Inc., Sebastopol 2011.

Skorowidz

A

B

C

D

E

F

G

H

I

J

K

L

Ł

M

N

O

P

R

S

Ś

T

U

V

W

X

Z

Systemy do zarządzania danymi muszą dziś spełniać o wiele wyższe wymagania niż kiedyś. W wielu przypadkach nierelacyjne bazy danych, zwane NoSQL, są lepszym rozwiązaniem niż dominujące do niedawna bazy relacyjne. Projektant systemu ma więc do dyspozycji dobrze znane bazy relacyjne oraz systemy NoSQL, takie jak bazy klucz – wartość, bazy dokumentów, rodziny kolumn i bazy grafowe. Do rozstrzygnięcia pozostaje problem: którą technologię zarządzania danymi wybrać w danym przypadku.

Niniejsza książka jest przystępnym, pragmatycznym przewodnikiem po nierelacyjnych systemach bazodanowych. Pokazano w niej, czym różnią się NoSQL od baz relacyjnych. Szczególny nacisk położono na wyjaśnienie tych cech i funkcji, które powinny być uwzględniane podczas projektowania aplikacji i wybierania technologii bazodanowych. Przedstawiono wewnętrzne mechanizmy baz NoSQL i wyjaśniono, w jaki sposób zbudować za ich pomocą skalowalne, niezawodne aplikacje. Nie zabrakło przydatnych wskazówek, zasad projektowych i najlepszych praktyk.

W tej książce znajdziesz:

- podstawy relacyjnych baz danych w odniesieniu do systemów NoSQL
- bazy klucz – wartość — ich charakterystykę, zasady projektowania i słabe strony
- bazy dokumentów — koncepcje baz bez schematu, podstawowe operacje, najpowszechniejsze wzorce projektowe
- bazy rodzin kolumn — aplikacje Big Data, architekturę baz, zasady ich projektowania i wykorzystywania
- zasady dobierania technologii bazodanowej do konkretnych zastosowań

Dan Sullivan jest naukowcem i architektem danych. Od dwudziestu lat zajmuje się m.in. analizą biznesową, uczeniem maszynowym i data miningiem. Sullivan jest uznanym ekspertem w dziedzinie baz danych, zarówno relacyjnych, jak i NoSQL.

Addison-Wesley

Helion

46418 numer katalogowy

księgarnia internetowa

http://helion.pl

zamówienia telefoniczne

0 801 339900

0 601 339900

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. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

sięgnij po WIĘCEJ

KOD KORZYŚCI

ISBN 978-83-283-2488-6

9 788328 324886

Informatyka w najlepszym wydaniu

cena: 69,00 zł

ALWAYS LEARNING

PEARSON