Relacyjne, dokumentowe, kolumnowe, klucz-wartość: mapa rodzin

Poziom: podstawy **Dotyczy: wszystkie silniki** **Czas: ~15 min**

„NoSQL" sugeruje jedną kategorię, a w rzeczywistości kryje pod tą nazwą trzy albo cztery zupełnie różne sposoby organizowania danych, które łączy właściwie tylko to, że żaden z nich nie jest modelem relacyjnym.

Skróty użyte na tej stronie (7)

Skrót	Co znaczy
ASE	Adaptive Server Enterprise: nazwa silnika Sybase od 1996 roku, dziś rozwijanego przez SAP jako SAP ASE
BSON	Binary JSON: binarny format zapisu dokumentów używany wewnętrznie przez MongoDB, w tym w archiwach mongodump
CAP	Client Access Point: punkt dostępu klienta, czyli nazwa i adres IP roli klastra, pod którymi użytkownicy łączą się z usługą niezależnie od tego, który węzeł ją prowadzi
CQL	Cassandra Query Language: język zapytań Cassandry, podobny do SQL
JSON	JavaScript Object Notation: tekstowy format danych z klamrami i cudzysłowami
SAP	Systeme, Anwendungen und Produkte in der Datenverarbeitung: niemiecki producent oprogramowania klasy ERP, właściciel silnika HANA i przejętego wcześniej Sybase ASE
SQL	Structured Query Language: język zapytań do relacyjnych baz danych

Po co to istnieje

Wybór silnika bazodanowego bywa traktowany jako wybór między „SQL" a „NoSQL", jakby to były dwie równoważne opcje. W praktyce „SQL" oznacza tu jeden konkretny model danych — relacyjny — a „NoSQL" jest workiem, do którego wrzucono wszystko, co tym modelem nie jest, choć te „wszystko inne" różni się między sobą bardziej, niż różnią się od modelu relacyjnego.

Rodziny alternatywne wobec modelu relacyjnego nie są nowością ostatnich lat — wyrosły w drugiej połowie lat dwutysięcznych z konkretnego problemu: firmy w rodzaju Google i Amazon obsługiwały ruch, przy którym pojedynczy serwer relacyjny przestawał wystarczać, a jednoczesne wymaganie ścisłej spójności i pracy na tysiącach maszyn naraz okazało się trudne do pogodzenia. Publikacje takie jak artykuł Google o systemie Bigtable (2006) czy Amazona o systemie Dynamo (2007) opisały architektury zaprojektowane pod kątem skalowania poziomego kosztem części gwarancji, które model relacyjny traktował jako oczywiste — i to na tych architekturach wprost wzorują się dzisiejsze Cassandra i Redis.

> **Pojęcie: model danych.** Sposób, w jaki system organizuje dane koncepcyjnie i w jaki się je odpytuje — nie fizyczny format pliku na dysku, tylko kształt, w którym programista myśli o danych: jako o tabelach powiązanych kluczami, jako o samodzielnych dokumentach, jako o parach klucz-wartość. Wybór modelu danych determinuje, jakie pytania da się zadać łatwo, a jakie — trudno albo wcale.

Rodzina relacyjna

Opisana w szczegółach na stronie Model relacyjny: tabela, wiersz, kolumna, dziedzina — dane w tabelach powiązanych kluczami, odpytywane językiem SQL, z silnym naciskiem na spójność i integralność referencyjną wymuszaną przez sam silnik. Przedstawiciele na tej półce: PostgreSQL, Oracle Database, Microsoft SQL Server, MySQL, Sybase (SAP ASE). Sensowna, gdy dane mają jasną, powtarzalną strukturę i gdy zależy nam na tym, żeby silnik sam pilnował spójności między powiązanymi tabelami — księgowość, zamówienia, dane kadrowe, wszystko, gdzie „ten wiersz nie może istnieć bez tamtego" jest regułą biznesową, nie sugestią.

Rodzina dokumentowa

> **Pojęcie: dokument.** Samoopisujący się rekord, zwykle w formacie JSON albo jego binarnej odmianie (BSON w MongoDB), który trzyma wszystkie dane dotyczące jednej encji w jednej strukturze, łącznie z zagnieżdżonymi listami i obiektami — bez rozbijania ich na osobne tabele połączone kluczami obcymi. Przedstawiciel: MongoDB. Sensowna, gdy struktura danych jest zmienna między rekordami (nie każdy produkt w katalogu ma te same atrybuty) albo gdy dana encja i tak jest zawsze czytana i zapisywana w całości — profil użytkownika z całą jego konfiguracją w jednym dokumencie, zamiast złączenia pięciu tabel za każdym odczytem.

Rodzina kolumnowa (szeroko-kolumnowa)

> **Pojęcie: magazyn szeroko-kolumnowy.** Model, w którym dane grupowane są wierszami zidentyfikowanymi kluczem, ale w przeciwieństwie do tabeli relacyjnej każdy wiersz może mieć inny zestaw kolumn, a dane fizycznie są przechowywane i kompresowane kolumna po kolumnie, nie wiersz po wierszu — co czyni odczyt jednej kolumny z milionów wierszy bardzo tanim, kosztem odczytu całego wiersza naraz. Przedstawiciel: Cassandra. Sensowna przy bardzo dużym wolumenie zapisów rozłożonych na wiele serwerów naraz, gdzie odporność na awarię pojedynczego węzła i skalowanie poziome liczą się bardziej niż złożone zapytania łączące wiele encji — telemetria z urządzeń, logi zdarzeń, dane czasowe o wysokiej częstotliwości zapisu.

Rodzina klucz-wartość

> **Pojęcie: magazyn klucz-wartość.** Najprostszy z modeli — baza to w istocie ogromna mapa, gdzie każdemu kluczowi odpowiada jedna wartość, odczytywana i zapisywana wyłącznie po tym kluczu, bez języka zapytań w stylu SQL. Przedstawiciel: Redis. Sensowna jako warstwa bardzo szybkiego dostępu do danych o prostej strukturze dostępu — cache wyników zapytań, sesje użytkowników, liczniki, kolejki — nie jako główne, trwałe źródło prawdy dla danych o złożonych powiązaniach.

Czwarta, mniej liczna na tej półce rodzina to bazy grafowe (np. Neo4j), gdzie węzły i krawędzie między nimi są danymi pierwszej klasy, a nie czymś symulowanym kluczami obcymi — sensowne tam, gdzie pytanie brzmi „jak te rzeczy są ze sobą powiązane" (sieci społecznościowe, wykrywanie oszustw, rekomendacje), a nie „ile mamy tych rzeczy". W modelu relacyjnym odpowiedź na pytanie „znajomi znajomych znajomych" wymaga kilku złączeń tej samej tabeli z samą sobą i szybko robi się kosztowna; w bazie grafowej to przejście po krawędziach, które silnik wykonuje jako swoją podstawową operację.

Kiedy która ma sens

Rodzina	Mocna strona	Cena

Relacyjna	integralność wymuszana przez silnik, elastyczne zapytania ad hoc (JOIN, agregacje)	schemat trzeba zaprojektować z góry, skalowanie poziome jest trudniejsze
Dokumentowa	elastyczna struktura, wygodna gdy encja jest czytana/zapisywana w całości	złączenia między dokumentami są niewygodne lub niemożliwe, spójność między dokumentami zwykle po stronie aplikacji
Kolumnowa	bardzo wysoka przepustowość zapisu, skalowanie poziome na wielu węzłach	ograniczony język zapytań, złączenia praktycznie nie istnieją
Klucz-wartość	ekstremalna szybkość dla prostego dostępu po kluczu	brak zapytań po czymkolwiek innym niż klucz, brak relacji między wartościami

Jak to działa: schemat wymuszony a schemat zakładany

> **Pojęcie: schema-on-write.** Podejście, w którym silnik wymusza zgodność ze strukturą (typy kolumn, ograniczenia) w chwili zapisu — `INSERT` niezgodny ze schematem tabeli jest odrzucany, zanim dane trafią na dysk. Tak działa każdy silnik relacyjny na tej półce.

> **Pojęcie: schema-on-read.** Podejście, w którym silnik przy zapisie nie narzuca żadnej konkretnej struktury — przyjmie dowolny dokument JSON — a to, czy struktura jest zgodna z oczekiwaniami, ujawnia się dopiero w momencie, gdy aplikacja próbuje odczytać konkretne pole. Tak działa MongoDB: dwa dokumenty w tej samej kolekcji mogą mieć zupełnie inny zestaw pól, i silnik nie ma nic przeciwko.

Różnica ta nie jest kwestią „lepiej" czy „gorzej" — to przesunięcie w czasie momentu, w którym niezgodność strukturalna staje się widoczna. Schema-on-write wychwytuje błąd danych natychmiast, kosztem sztywności. Schema-on-read pozwala na ewolucję struktury bez migracji całej kolekcji naraz, kosztem tego, że błąd (brakujące pole, zły typ) ujawnia się dopiero w kodzie aplikacji, czasem długo po zapisaniu danych.

Twierdzenie CAP w jednym akapicie

> **Pojęcie: twierdzenie CAP.** Dla systemu rozproszonego, w którym dane są replikowane na kilku węzłach, twierdzenie CAP mówi, że w chwili, gdy sieć między węzłami się przerywa (partycja), system musi wybrać między dalszym odpowiadaniem na zapytania z potencjalnie nieaktualnymi danymi (dostępność) a odmową odpowiedzi, dopóki nie ma pewności co do spójności danych (spójność) — nie może mieć jednocześnie pełni obu. To nie jest wybór, którego dokonuje się raz na zawsze przy projektowaniu systemu — dobre systemy rozproszone pozwalają dostroić to zachowanie per operację, a odporność na partycję sieci nie jest w praktyce opcją do odrzucenia, tylko warunkiem brzegowym każdej architektury rozproszonej, więc realny wybór dotyczy tego, co system robi w chwili partycji, a nie tego, którą z trzech liter „wybiera raz na stałe" — wbrew popularnemu uproszczeniu „pick two of three".

Dlaczego „NoSQL" to zła nazwa

Nazwa definiuje kategorię przez negację — mówi, czym coś nie jest, a nie czym jest. MongoDB, Cassandra i Redis różnią się między sobą modelem danych, sposobem skalowania i typowym zastosowaniem bardziej, niż niektóre z nich różnią się od baz relacyjnych w konkretnych aspektach — Cassandra ma własny język zapytań (CQL) składniowo bardzo zbliżony do SQL, a MongoDB od pewnego czasu oferuje interfejs SQL-podobny do swojej agregacji. Trzymanie się nazwy „NoSQL" sugeruje istnienie jednej alternatywy dla SQL, podczas gdy w rzeczywistości istnieje kilka niezależnych modeli danych, z których każdy trzeba oceniać osobno pod kątem konkretnego problemu — nie da się wybrać „NoSQL" tak, jak wybiera się jeden konkretny silnik.

Częste błędy

Wybór modelu dokumentowego z powodu mody, nie potrzeby. Dane, które mają stałą, powtarzalną strukturę i wymagają częstych złączeń między encjami (zamówienia i ich pozycje, użytkownicy i ich role), w modelu dokumentowym kończą albo jako zagnieżdżone, zduplikowane dane w wielu dokumentach naraz, albo jako ręcznie odtworzone złączenia po stronie aplikacji — czyli dokładnie to, co model relacyjny robi za darmo.

Traktowanie „schema-on-read" jako braku schematu w ogóle. Baza dokumentowa nie wymusza jednej struktury na poziomie silnika, ale to nie znaczy, że struktury nie ma — aplikacja, która czyta dokumenty, wciąż zakłada konkretne pola i typy. Różnica jest w momencie, kiedy niezgodność struktury ujawnia się jako błąd: przy zapisie (relacyjna) czy dopiero przy odczycie, w kodzie aplikacji (dokumentowa).

Odczytanie twierdzenia CAP jako „trzeba wybrać dwie z trzech cech na zawsze". To popularne uproszczenie ignoruje, że tolerancja na partycję sieci nie jest opcjonalna w systemie

rozproszonym — pytanie brzmi wyłącznie, co system robi w chwili awarii sieci, nie które dwie litery z trzech ktoś sobie wybrał przy projektowaniu.

□□ Ćwiczenia

> Poniższe zadania nie wymagają zainstalowanego żadnego konkretnego silnika — chodzi o rozpoznanie modelu, nie o składnię.

Pytania kontrolne

1. System rejestruje dziesiątki tysięcy odczytów z czujników IoT na sekundę, a typowe zapytanie to „pokaż wszystkie odczyty czujnika X z ostatniej godziny". Która rodzina pasuje najlepiej i dlaczego?

Odpowiedź

Rodzina kolumnowa (np. Cassandra) — bardzo wysoka przepustowość zapisu rozłożona na wiele węzłów, a typowe zapytanie jest proste (jeden klucz identyfikujący czujnik, zakres czasu), więc nie potrzeba bogatego języka złączeń, za to potrzeba skalowania poziomego, którego model relacyjny na pojedynczym serwerze nie daje równie łatwo.

2. Dlaczego nazwanie Cassandry i MongoDB „tym samym typem bazy" jest błędem, mimo że obie często pojawiają się pod wspólną etykietą „NoSQL"?

Odpowiedź

Bo reprezentują dwa różne modele danych — Cassandra jest magazynem szeroko-kolumnowym zoptymalizowanym pod masowy, rozproszony zapis i proste zapytania po kluczu, a MongoDB jest bazą dokumentową zoptymalizowaną pod przechowywanie i odczyt złożonych, samodzielnych rekordów. Łączy je wyłącznie to, czym nie są — modelem relacyjnym — co jest dokładnie tym problemem z nazwą „NoSQL", o którym mówi ta strona.

Powiązane: Model relacyjny: tabela, wiersz, kolumna, dziedzina · Replikacja: synchroniczna, asynchroniczna, logiczna, fizyczna

Źródła: E. Brewer, prezentacja o twierdzeniu CAP (PODC 2000) · dokumentacja MongoDB, rozdz. „Data Modeling" · dokumentacja Apache Cassandra, rozdz. „Data Modeling"

Revision #47
Created 2026-09-24 22:41:41 UTC by Tomasz Konieczny
Updated 2026-10-26 14:10:39 UTC by Tomasz Konieczny