

O'REILLY®

Helion 

Wektorowe bazy danych

Kompletny przewodnik
dla programistów AI i inżynierów danych



Nitin Borwankar

Tytuł oryginału: Vector Databases: A Practical Introduction

Tłumaczenie: Robert Górczyński

ISBN: 978-83-289-4206-6

© 2026 Helion S.A.

Authorized Polish translation of the English edition of *Vector Databases*

ISBN 9781098177591 © 2026 Nitin Borwankar.

This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same.

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 the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiejkolwiek 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 wydawca dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz wydawca nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

helion.pl/user/opinie/webada

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Helion S.A.

ul. Kościuszki 1c, 44-100 Gliwice

tel. 32 230 98 63

e-mail: helion@helion.pl

WWW: helion.pl (księgarnia internetowa, katalog książek)

Printed in Poland.

- Kup książkę
- Poleć książkę
- Oceń książkę

- Księgarnia internetowa
- Lubię to! » Nasza społeczność

Spis treści

Wprowadzenie	11
1. Wprowadzenie do wektorowych baz danych	19
Dlaczego potrzebujesz wektorowej bazy danych?	19
Wektor — nowy typ danych	20
Wyszukiwanie podobieństw	21
Czym charakteryzuje się typ danych Vector?	24
Gdzie wykorzystuje się wektorowe bazy danych?	26
SQL kontra wektorowe bazy danych	27
Podstawy matematyki biznesowej: arytmetyka księgowa	27
Reprezentacja wektorowa w relacyjnym systemie zarządzania bazą danych	28
Potrzeba funkcji specyficznych dla wektorów	29
NoSQL i wektorowe bazy danych	29
Bazy danych NoSQL i wektorowy magazyn danych	30
Ograniczenia rozszerzeń wektorowych w bazach danych NoSQL	30
Kiedy wybrać bazę danych NoSQL z rozszerzeniem wektorowym?	30
Podejścia hybrydowe: łączenie danych ustrukturyzowanych i wektorowych	31
Konieczność obsługi zarówno danych wektorowych, jak i metadanych	31
Ograniczenia czysto wektorowego magazynu danych	31
Hybrydowa architektura bazy danych	32
Przykład zapytania hybrydowego	32
Zalety podejścia hybrydowego	33
Podsumowanie	33
2. Osadzenia	35
Osadzenia wektorowe — dlaczego są potrzebne?	35
Word2Vec — przełom, który zmienił wszystko	36
Doc2Vec — od słów do dokumentów	38

Od osadzeń do nowoczesnych modeli językowych — związek z transformerami	41
Transformery oparte wyłącznie na koderze (BERT i jego warianty)	42
Transformery oparte wyłącznie na dekodrze (rodzina GPT)	42
Transformery typu koder-dekoder (T5, BART)	42
Modele osadzeń — wyspecjalizowane generatory wektorów	43
Różnice względem tradycyjnych modeli	44
Rola w nowoczesnych aplikacjach LLM	44
Praktyczne zastosowania i przypadki użycia	45
Prosty potok RAG	46
Biblioteka sentence-transformers —	
wszechstronne narzędzie do pracy z osadzeniami tekstu	49
Najlepsze praktyki korzystania z sentence-transformers —	
szczegółowy przewodnik	53
Warstwa osadzeń — brama do uczenia typu zero-shot	57
Anatomia osadzeń transformera	57
Połączenie z uczeniem typu zero-shot	59
Kluczowe cechy umożliwiające uczenie typu zero-shot	60
Ograniczenia i uwagi	61
Najnowsze osiągnięcia i trendy	62
Arytmetyka wektorowa z Word2Vec — praktyczny przewodnik	63
Krok 1. Konfiguracja i instalacja	63
Krok 2. Wczytanie wstępnie wytrenowanego modelu Word2Vec	63
Krok 3. Implementacja funkcji arytmetyki wektorowej	64
Krok 4. Klasyczna analogia król-królowa	65
Krok 5. Ciekawsze analogie	65
Krok 6. Interaktywne narzędzie do eksploracji	66
Ostatnie uwagi o arytmetyce wektorowej	67
Podsumowanie	67
3. Wyszukiwanie podobieństw za pomocą FAISS	68
Podstawy	68
Reprezentacje wektorowe	70
Miary odległości	71
Heurystyki selekcji	73
Indeksy FAISS	73
Indeksy płaskie (brute force)	73
Indeksy oparte na IVF	74
Indeksy oparte na LSH	75
Indeksy oparte na HNSW	76
Inne indeksy specjalistyczne	76
Indeksy złożone i transformujące	77
Wybór odpowiedniego indeksu	77

Kwantyzacja	80
Kwantyzacja skalarna	80
Kwantyzacja iloczynowa	82
Problem związany z wyszukiwaniem ANN	86
Problem	87
Unikaj kosztów obliczeniowych	87
Kluczowe techniki wyszukiwania ANN w FAISS	88
Wybór indeksu w FAISS	90
Przykładowy fragment kodu	90
Zrozumienie indeksów HNSW	91
Czym jest HNSW?	92
Jak działa HNSW?	93
Omówienie kluczowych parametrów	94
Przykład praktyczny: tworzenie systemu wyszukiwania podobieństw	95
Charakterystyka wydajności działania	96
Najlepsze praktyki	97
Architektura i komponenty FAISS	97
Podstawy	97
Podstawowe koncepcje	99
Kluczowe komponenty	100
Typowy przepływ pracy	101
Przykład	102
Najważniejsze wnioski	102
Dalsza eksploracja	103
Podsumowanie	103
4. Wyszukiwanie semantyczne za pomocą SQLite3	104
Rozszerzenie SQLite Vector Similarity Search	104
Podstawowe możliwości	105
Schemat architektury	106
Ograniczenia	106
Konfigurowanie środowiska programistycznego	107
Instalowanie zależności	107
Weryfikowanie instalacji	108
Instrukcje pragma	109
Projektowanie schematu bazy danych	109
Wymagania schematu	109
Definicje tabel	109
Decyzje projektowe dotyczące schematu	111
Łączenie z serwisem Reddit za pomocą interfejsu Python Reddit API Wrapper	112
Tworzenie poświadczeń dla API Reddit	112
Implementacja klienta PRAW	112
Przykład użycia	115

Ekstrakcja treści i jej wstępne przetwarzanie	115
Potok oczyszczania tekstu	115
Filtrowanie jakościowe	118
Generowanie i przechowywanie osadzeń	118
Generator osadzeń	119
Przechowywanie danych w bazie	120
Potok przetwarzania wsadowego	124
Tworzenie indeksu wektorowego	125
Indeksowanie za pomocą rozszerzenia VSS	125
Zarządzanie indeksami	126
Implementacja wyszukiwania semantycznego	128
Kontener wyników wyszukiwania	129
Silnik wyszukiwania	129
Połączenie wszystkiego w całość	135
Przykładowy przepływ pracy	135
Przykładowy wynik działania	137
Rozszerzenie — indeksowanie przyrostowe	138
Podsumowanie	139

5. Budowa systemu wyszukiwania publikacji z ArXiv z wykorzystaniem PostgreSQL 141

Wyzwanie przeszukiwania literatury naukowej	141
Dlaczego ArXiv jest idealnym źródłem danych?	141
Rzeczywiste przypadki użycia	142
Uzasadnienie wyboru stosu technologicznego	142
Schemat architektury	143
Komponenty systemu	143
Przepływ danych	144
Filozofia projektowania	145
Konfiguracja środowiska i zależności	145
Instalacja bazy danych PostgreSQL i rozszerzenia pgvector	146
Konfiguracja środowiska Pythona	146
Struktura katalogów i konfiguracja	147
Weryfikacja i testowanie	148
Projektowanie baz danych dla artykułów naukowych	149
Zasady projektowania schematu	149
Struktura głównych tabel schematu	150
Strategia przechowywania wektorów	153
Strategia indeksowania	154
Integracja z serwisem ArXiv i zarządzanie plikami PDF	155
Implementacja klienta API ArXiv	155
Potok pobierania plików PDF	156
System przetwarzania wsadowego	157

Wyodrębnianie tekstu z pliku PDF i jego przetwarzanie	159
Wyzwania związane z wyodrębnianiem tekstu z pliku PDF	159
Inteligentny podział tekstu na fragmenty	161
Generowanie i przechowywanie osadzeń	162
Strategia modelu osadzeń	162
Potok przetwarzania wsadowego	164
Implementacja wyszukiwania podobieństw	164
Aplikacja interaktywna i interfejs użytkownika	167
Pakowanie w Dockerze do lokalnego wdrożenia	169
Architektura kontenera	169
Konfiguracja narzędzia Docker Compose	169
Skrypty do inicjalizacji bazy danych	171
Programistyczny przepływ pracy	172
Projektowanie pod kątem chmury	172
Podstawy dostrajania wydajności	172
Konfiguracja indeksu	173
Wydajność działania zapytań	173
Zarządzanie zasobami	173
Kolejne kroki	173
Aktualne ograniczenia	173
Pomysły na ulepszenia	173
Co zrobiliśmy?	174
Osiągnięcia systemu	174
Zdobyte umiejętności techniczne	175
Praktyczne narzędzie badawcze	175
Podstawy systemów zaawansowanych	175
Przyszły potencjał	175
Podsumowanie	175
6. Budowa systemu RAG z wykorzystaniem SQLite VSS i Ollamy	176
Schemat architektury systemu	177
Podstawy baz danych z obsługą wektorów	178
Konfiguracja bazy danych z obsługą wektorów	178
Projektowanie schematu dla RAG	179
Tworzenie indeksów wyszukiwania	180
Przetwarzanie tekstu i generowanie osadzeń	181
Zarządzanie modelami osadzeń	181
Inteligentny podział tekstu na fragmenty	182
Przechowywanie treści za pomocą osadzeń	183
Implementacja wyszukiwania hybrydowego	184
Algorytm wyszukiwania hybrydowego	184
Komponent wyszukiwania semantycznego	185

Komponent wyszukiwania słów kluczowych	186
Fuzja i ranking wyników	187
Integracja LLM-a z Ollamą	188
Klient API Ollamy	188
Funkcja sprawdzania stanu Ollamy	188
Potok RAG	189
Formatowanie kontekstu	189
Potok udzielania odpowiedzi na pytanie	190
Prezentacja i testowanie	191
Wczytywanie przykładowych danych	191
Prezentacja możliwości systemu RAG	193
Interaktywny interfejs pytań i odpowiedzi	193
Narzędzie do szybkiego testowania	194
Kolejne kroki — rozbudowa systemu	195
Brakujące cechy danych z Reddita	195
Optymalizacje wydajności	196
Zagadnienia produkcyjne	197
Zaawansowane wzorce RAG	197
Podsumowanie	198
7. Budowa naukowego systemu RAG z wykorzystaniem PostgreSQL i pgvector	199
Cele i możliwości systemu	200
Schemat architektury	200
Baza danych z rozszerzeniem pgvector	202
Konfiguracja i przygotowanie bazy danych	202
Projektowanie schematu dla artykułów naukowych	203
Wysokowydajne indeksy wektorowe	204
Strategia generowania osadzeń	205
Integracja z serwisem ArXiv i przetwarzanie plików PDF	205
Znajdowanie publikacji za pomocą ArXiv API	206
Inteligentne wyodrębnianie tekstu z dokumentu PDF	207
Zaawansowany podział tekstu	209
Potok zapisu z osadzeniami	209
Wielopoziomowe wyszukiwanie semantyczne	211
Wyszukiwanie na poziomie abstraktu	212
Wyszukiwanie na poziomie sekcji	212
Potok RAG — dokładna analiza	214
Lokalna integracja LLM-a z Ollamą	214
Sprawdzanie stanu i wykrywanie modeli	215
Inteligentne wyszukiwanie kontekstu	215
Inżynieria promptów naukowych	216
Kompletny potok działania systemu RAG	217

Prezentacja i interfejs interaktywny	218
Główny przebieg prezentacji	218
Prezentacja operacji wyszukiwania	219
Prezentacja systemu RAG	220
Interfejs wyszukiwania interaktywnego	221
Punkt wejścia z wyborem trybu	222
Uwaga techniczna dotycząca HNSW	222
Jak ocenić swoje wyniki?	224
Kolejne kroki — rozbudowa naukowego systemu RAG	224
Podsumowanie	227
8. Budowa kompletnego systemu wyszukiwania rozmów i RAG	229
Cele i możliwości systemu	230
Schemat architektury systemu	231
Co wspólnie zbudujemy?	233
Baza danych do przechowywania rozmów	234
Projektowanie schematu rozmowy	234
Architektura trzech tabel dla zapewnienia optymalnej wydajności działania	235
Wysokowydajne indeksowanie wektorowe	236
Importowanie rozmów i potok przetwarzania danych	236
Niezawodne importowanie danych JSON z obsługą błędów	237
Przetwarzanie transakcji atomowych	237
Obsługa znaczników czasu i weryfikacja danych	238
Odzyskiwanie po błędach i rejestrowanie danych	239
Wydajne generowanie osadzeń i przetwarzanie wsadowe	240
Wzorzec singleton do zarządzania modelem	240
Strategia przetwarzania przyrostowego	240
Przetwarzanie wsadowe w celu osiągnięcia optymalnej wydajności działania	241
Obsługa konfliktów podczas wstawiania osadzeń do bazy danych	241
Wyszukiwanie kontekstowe ze zrozumieniem rozmów	242
Semantyczne wyszukiwanie podobieństw	243
Złączenia obejmujące wiele tabel w celu uzyskania rozbudowanego kontekstu	243
Formatowanie i struktura wyników	243
Pobieranie kontekstu rozmowy	244
Wyznaczanie granic okna kontekstowego	244
Integracja potoku RAG dla historii rozmowy	245
Ustrukturyzowane zarządzanie kontekstem	245
Lokalna integracja LLM-a z Ollamą	246
Monitorowanie stanu i odkrywanie modeli	246
Pobieranie i przygotowywanie kontekstu	247
Konwersacyjna inżynieria promptów	247
Pełny potok RAG wraz z monitorowaniem wydajności działania	248

Kompletne API internetowe z FastAPI	250
Struktura aplikacji FastAPI	250
Modele żądań z weryfikacją	250
Implementacja punktu końcowego wyszukiwania	251
Punkt końcowy RAG do udzielania odpowiedzi na pytania	251
Dane statystyczne i monitorowanie systemu	251
Konfiguracja serwera i jego uruchamianie	252
Prezentacja i przykładowe dane	253
Generowanie realistycznych danych przykładowych	253
Przykładowe pokrycie wielu tematów	254
Przykładowy potok przetwarzania danych	255
Kompleksowa prezentacja systemu	255
Stopniowa prezentacja funkcjonalności	256
Prezentacja systemu RAG z wykonaniem warunkowym	257
Funkcjonalność produkcyjnej operacji importowania	257
Punkty wejścia aplikacji	258
Podsumowanie — kompletny osobisty system wiedzy	258
9. Język zapytań wektorowych	259
Kluczowe koncepcje	260
Model danych	260
Podstawowa struktura składni	261
Operacje na wektorach	262
Wyszukiwanie podobieństw	262
Wyszukiwanie hybrydowe	263
Wyszukiwanie w zakresie	263
Operacje wsadowe	263
Funkcje i agregacje wektorowe	264
Funkcje wektorowe	264
Agregacje wektorowe	265

Wprowadzenie do wektorowych baz danych

Obecna fala generatywnej sztucznej inteligencji (w skrócie GenAI, ang. *generative AI*), szczególnie w aplikacjach opartych na języku naturalnym, wysunęła technologię wektorowych baz danych na pierwszy plan zainteresowań programistów. Nieustrukturyzowane dane służące do trenowania modeli są wektoryzowane, a dane wykorzystywane w procesie wnioskowania często trafiają do wektorowych baz danych. A jednak ta technologia wciąż nie jest dobrze znana ani w pełni zrozumiała dla przeciętnego twórcy aplikacji bazodanowych.

W tym rozdziale znajdziesz ogólny przegląd wektorowych baz danych, zarys ich głównych komponentów oraz zapowiedź materiału, który omówię w dalszej części książki. Dowiesz się przede wszystkim, czym jest ta technologia i do czego jest potrzebna. Po lekturze rozdziału będziesz wiedzieć, jak używać wektorowych baz danych, natomiast kluczowy temat tzw. osadzeń (ang. *embeddings*) omawiam szczegółowo w kolejnym rozdziale.

Dlaczego potrzebujesz wektorowej bazy danych?

Nawet przed nadejściem generatywnej sztucznej inteligencji zarządzanie danymi nieustrukturyzowanymi — takimi jak tekst, obrazy, dźwięk i wideo — było trudne. Wymagało specjalistycznej technologii i wiedzy, aby zarządzać tymi zasobami na dużą skalę. Indeksowanie i przeszukiwanie takich danych było możliwe tylko na poziomie metadanych — na przykład daty utworzenia, wielkości, autora itd. Jednak znalezienie obrazu podobnego do innego lub akapitu o podobnym znaczeniu nie było zadaniem, z którym poradziłyby sobie bazy danych SQL, ponieważ służy ona do dopasowywania składni, a nie semantyki.

Obecnie, wraz z wejściem wektorowych baz danych do głównego nurtu, zarządzanie danymi nieustrukturyzowanymi i wyszukiwanie oparte na *semantyce* — czyli znaczeniu danych — stało się znacznie prostsze. To właśnie świat wektorowych baz danych otwiera nowe możliwości aplikacjom opartym na wyszukiwaniu semantycznym lub wyszukiwaniu podobieństw. Wykracza to daleko poza wyszukiwanie słów kluczowych, które często prowadzi do nieporozumień, na przykład między angielskim słowem *apple* (jabłko) oznaczającym owoc, a słowem *Apple* jako nazwą firmy komputerowej. Wektorowe bazy danych stanowią fundamentalną

technologię dla sztucznej inteligencji w przedsiębiorstwach, podobnie jak relacyjne bazy danych są podstawą dla danych biznesowych. Obecnie powstają aplikacje, które łączą dane biznesowe z semantyką tekstu, obrazów, dźwięku i wideo, tworząc bogate, inteligentne i interaktywne systemy. Aby tworzyć tak rozbudowane aplikacje, konieczne jest dokładne zrozumienie wektorowych baz danych, na znacznie głębszym poziomie niż ten, który obecnie jest powszechnie dostępny.

Chociaż specjalistyczne wektorowe bazy danych są dostępne od mniej więcej 2025 roku, różnią się nieco od siebie i wymagają wiedzy specyficznej dla konkretnego rozwiązania. Wiele z nich jest dostępnych w chmurze — co wymaga znacznie mniej specjalistycznej konserwacji — podczas gdy wzbogacone o obsługę danych wektorowych bazy danych takie jak PostgreSQL są równie łatwo dostępne.

W książce skoncentrowałem się na technologii, w której możliwości wektorowych baz danych są dobrze zintegrowane ze światem relacyjnych baz danych. To pozwala na obsługę ustrukturyzowanych i nieustrukturyzowanych danych w znany sposób oraz zarządzanie nimi za pomocą dobrze znanych narzędzi bazodanowych, takich jak między innymi tworzenie kopii zapasowych, przywracanie, partycjonowanie i kontrola dostępu. Dane wektorowe (czyli nieustrukturyzowane) są zapisywane w standardowych tabelach bazy danych bez specjalnego traktowania ze względu na to, co te tabele mogą zawierać.

Przed pojawieniem się wektorowych baz danych dane nieustrukturyzowane były często przechowywane jako obiekty *BLOB* (ang. *binary large objects*) lub bezpośrednio w plikach, ze wskaźnikiem do pliku zapisanym w relacyjnej bazie danych. W obu przypadkach te dane uważano za nieprzezroczyste — stąd nazwa *BLOB*. Nie mogliśmy z nimi robić wiele więcej poza ich zapisywaniem i pobieraniem, z wyjątkiem być może tekstu, gdzie możliwe było wyszukiwanie słów kluczowych.

Wektor — nowy typ danych

Czym właściwie są „dane wektorowe” i dlaczego nie możemy po prostu użyć typu *BLOB* w jego obecnej postaci? Moglibyśmy — ale wektory dają supermoc, która pozwala dodać do danych komponent semantyczny, a także przygotować je pod kątem GenAI. W modelach językowych wykorzystywanych w GenAI słowa, zdania, akapity, a nawet całe dokumenty mogą być reprezentowane jako wektory. Dlatego aby realizować zaawansowane zadania w przestrzeni GenAI, zrozumienie danych wektorowych jest absolutnie kluczowe. Przyjrzyjmy się temu nieco bliżej.

Przed wszystkim, zaczynając od znanej nam dziedziny, potraktuj „wektor” jako nowy typ danych w systemie zarządzania bazą danych. Jako konkretny przykład weźmy PostgreSQL, który będzie naszą podstawą w tej książce — jest to dobrze znana i popularna baza danych.

Za pomocą systemu rozszerzeń w PostgreSQL do systemu typów możemy dodawać nowe typy. W przypadku typu wektorowego rozszerzenie *pgvector* daje możliwość tworzenia w tabeli kolumny typu *VECTOR* i oferuje natywną dla wektorów funkcję wyszukiwania — wyszukiwanie na podstawie podobieństwa (ang. *similarity search*) — dla tego typu danych. Jeśli spojrzysz

na zawartość kolumny VECTOR (zob. rysunek 1.1), zobaczysz listę liczb zmiennoprzecinkowych (float) — ale za tym kryje się znacznie więcej niż widać na pierwszy rzut oka.

ID	Nazwa	Osadzenie VECTOR(768)
1	Doc A	[0.023, -0.145, 0.892, 0.034, ..., -0.567, 0.123] 768 liczb zmiennoprzecinkowych

Każda liczba zmiennoprzecinkowa koduje określony aspekt znaczenia semantycznego

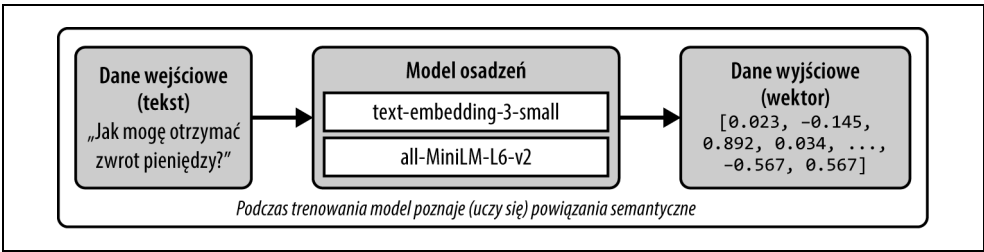
Rysunek 1.1. Zawartość typu danych Vector

Wyszukiwanie podobieństw

Funkcja wyszukiwania podobieństwa jest tym, co odróżnia typ danych VECTOR od zwykłego zapisywania i pobierania uporządkowanego zbioru liczb zmiennoprzecinkowych. To właśnie wyszukiwanie podobieństwa pozwala odszukać tekst o znaczeniu podobnym do wskazanego fragmentu. Możliwość określania podobieństwa wektorów stanowi fundament dla aplikacji bazodanowych, które są tworzone z użyciem dużych modeli językowych (LLM), takich jak ChatGPT i Claude, lub lokalnych LLM-ów, takich jak Llama3.x. Wyszukiwanie podobieństwa wektorów ma własny operator wyszukiwania, który rozszerzenie pgvector dodaje do systemu typów danych bazy PostgreSQL. To zagadnienie omówię bardziej szczegółowo w następnym rozdziale, poświęconym osadzeniom wektorowym.

Przykład wyszukiwania podobieństwa

Wyobraź sobie, że przeszukujesz firmową bazę wiedzy i wpisujesz jakieś zapytanie, na przykład: „Jak mogę otrzymać zwrot pieniędzy?”. Na rysunku 1.2 możesz zobaczyć, co się dzieje w takiej sytuacji.



Rysunek 1.2. Przeływ procesu osadzania

Wyszukiwanie słów kluczowych (dosłowne dopasowanie wyrazów). W przypadku wyszukiwania słów kluczowych system szuka dokładnie tych słów, które zostały wpisane (lub ich bliskich wariantów). Ta metoda sprawdza się dobrze, jeśli strona dosłownie zawiera słowa takie jak „pieniądze”

i „zwrot” lub frazę „chcę otrzymać zwrot pieniędzy”. Może jednak pominąć najlepszą odpowiedź, jeśli w dokumencie użyto innych słów lub fraz, takich jak:

- „polityka zwrotów”,
- „zwroty i wymiany”,
- „zwrot kosztów”,
- „anulowanie zamówienia”.

Zatem wyszukiwanie słów kluczowych może zwrócić:

- stronę o „gwarancji zwrotu pieniędzy” (dobrze, jeśli taka strona istnieje);
- nietrafione strony, w których słowa „pieniądze” i „zwrot” występują osobno (np. w kontekście usług „cash-back”)

Wyszukiwanie semantyczne (dopasowywanie znaczeniowe). *Wyszukiwanie semantyczne* próbuje zrozumieć, co użytkownik ma na myśli. Rozpoznaje, że fraza „chcę otrzymać zwrot pieniędzy” wyraża tę samą intencję, co:

- „prośba o zwrot kosztów”;
- „zwrot przedmiotu”;
- „anulowanie zamówienia i zwrot środków”;
- „cofnięcie obciążenia karty”.

Dlatego wyszukiwanie semantyczne z większym prawdopodobieństwem zwróci:

- stronę „Zwroty i reklamacje”,
- dokładną sekcję „Jak zażądać zwrotu pieniędzy?”.

Podsumowanie za pomocą jednego zdania. W gruncie rzeczy oba te rodzaje wyszukiwania sprawdzają się do następujących działań:

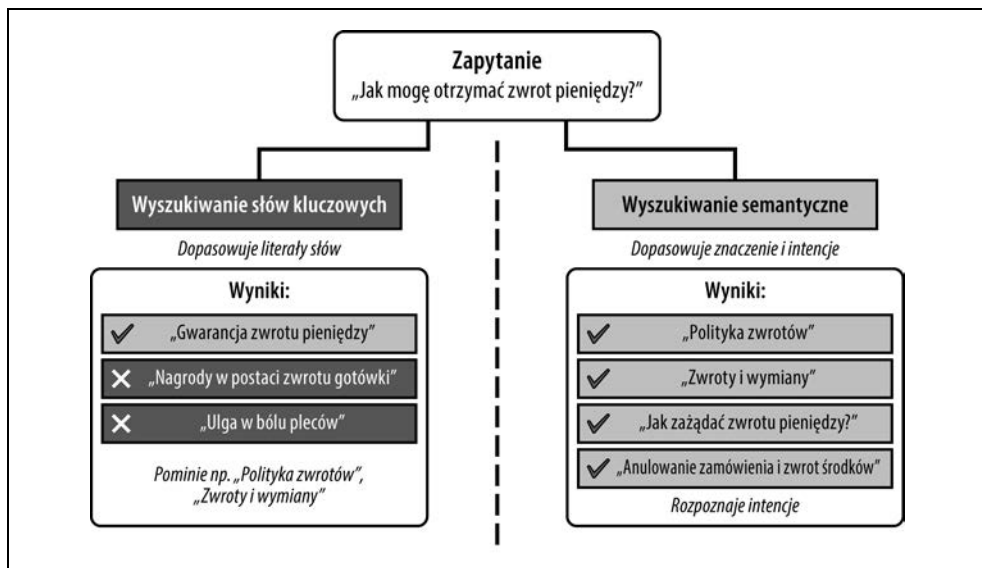
- Wyszukiwanie słów kluczowych: „Znajdź strony, które zawierają te słowa”.
- Wyszukiwanie semantyczne: „Znajdź strony, które odpowiadają na to pytanie, nawet jeśli użyto w nich innych sformułowań”.

Dodatkowe zapytanie

Przyjrzyjmy się jeszcze jednemu przykładowi. Wyobraź sobie, że szukasz frazy „Mój laptop nie chce się włączyć”.

- Wyszukiwanie słów kluczowych może pominąć artykuły zatytułowane np. „Urządzenie nie uruchamia się” lub „Problemy z zasilaniem”.
- Wyszukiwanie semantyczne najprawdopodobniej je znajdzie, ponieważ powiąże frazy „nie chce się włączyć” z „awarią zasilania/uruchamiania”.

Zobaczmy, jak wygląda działanie tej całej magii (zob. rysunek 1.3).



Rysunek 1.3. Porównanie wyszukiwania semantycznego i wyszukiwania opartego słów kluczowych

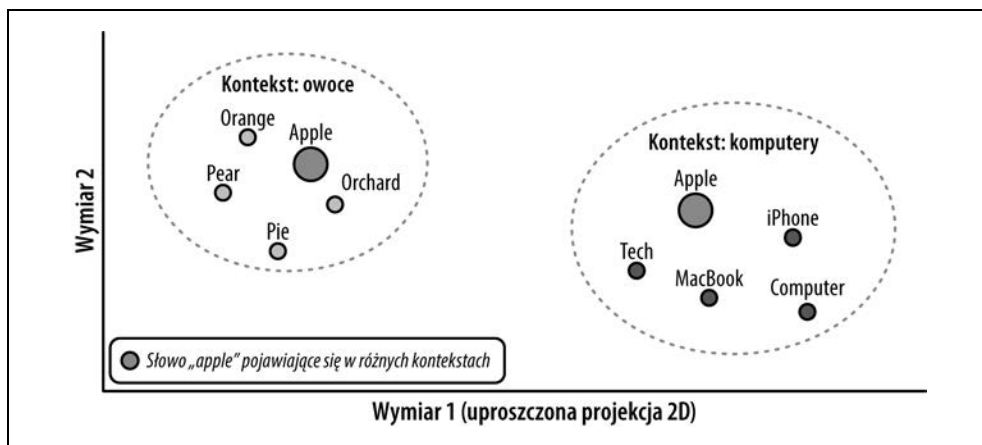
Osadzanie

Istnieje proces zwany *osadzaniem* (ang. *embedding*), który pozwala wziąć fragmenty danych nieustrukturyzowanych (czyli nietabelarycznych) i mapować je na wektor — listę liczb zmiennie-przecinkowych (zob. rysunek 1.3). Taki mapowanie zachowuje kontekst, w którym występuje dane słowo lub zdanie. Słowa o zbliżonym znaczeniu są mapowane na wektory, które leżą blisko siebie w przestrzeni wektorowej. Mówiąc bardzo ogólnie, znaczenie słowa jest silnie powiązane z kontekstem, w jakim zostało użyte.

Powróćmy do wcześniejszego przykładu ze słowem *apple* (jabłko): gdy to słowo lub zawierające je zdania pojawiają się w kontekście innych zdań o owocach, sadach czy pieczeniu ciasta, można stwierdzić, że w tym kontekście *apple* oznacza owoc. Jeśli zaś badany tekst występuje w kontekście opisów komputerów, można wywnioskować, że to słowo odnosi się do firmy Apple. Na tym etapie przedstawione wyjaśnienie jest nieco nieprecyzyjne, ale dobrze oddaje ogólną ideę. Bez obaw — w następnym rozdziale znacznie dokładniej omówię temat osadzeń. Na razie spójrz na rysunek 1.4, aby zorientować się, jak to działa.

W tym miejscu powinienem dodać, że słowo „osadzenie” jest używane w różnych znaczeniach, co może być mylące dla początkujących.

Po pierwsze, odnosi się ono do procesu, w którym dane są odwzorowywane na wektor — a co ważniejsze, w którym zbiór danych jest odwzorowywany w przestrzeń wektorową. W tym ujęciu pełni ono funkcję czasownika: proces osadzania. Na przykład kilka stron tekstu lub kilka książek, nazywanych łącznie *korpusem*, można przekształcić w procesie osadzania na zbiór wektorów podlegających regułom przestrzeni wektorowej.



Rysunek 1.4. Grupowanie w przestrzeni wektorowej

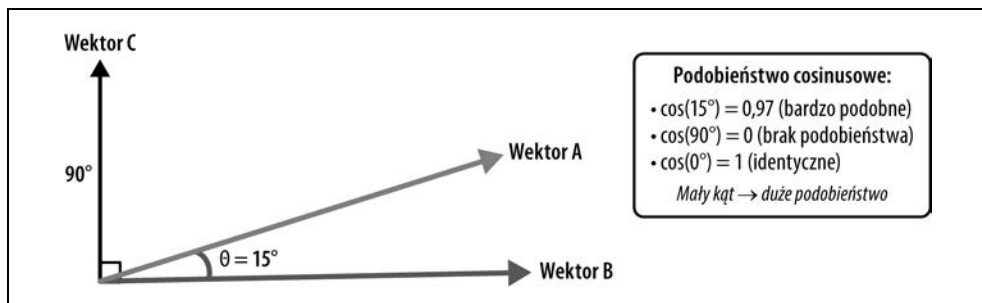
Po drugie, jest to również rzeczownik — *osadzenie*. W tym przypadku termin ten odnosi się do wyniku procesu osadzania — zbioru wektorów nazywanego łącznie *osadzeniem*. Zatem proces osadzania (czasownik) daje jako wynik osadzenie (rzeczownik). Dobrym sposobem na zapamiętanie tego rozróżnienia jest wyobrażenie sobie malarza tworzącego obraz: proces malowania tworzy obraz. Wreszcie „osadzenie” może odnosić się do pojedynczego wektora lub całej ich kolekcji — to, o które znaczenie chodzi, zazwyczaj jasno wynika z kontekstu.

Czym charakteryzuje się typ danych Vector?

Wraz z tym nowym typem danych pojawia się zestaw operacji przeznaczonych specjalnie do operowania wektorami. Operacje te mają fundamentalne znaczenie dla wielu zadań z dziedziny sztucznej inteligencji i uczenia maszynowego, takich jak systemy rekomendacyjne, rozpoznawanie obrazów czy rozumienie języka naturalnego. W kolejnych podpunktach nieco dokładniej przyjrzymy się tym operacjom.

Ocena podobieństwa

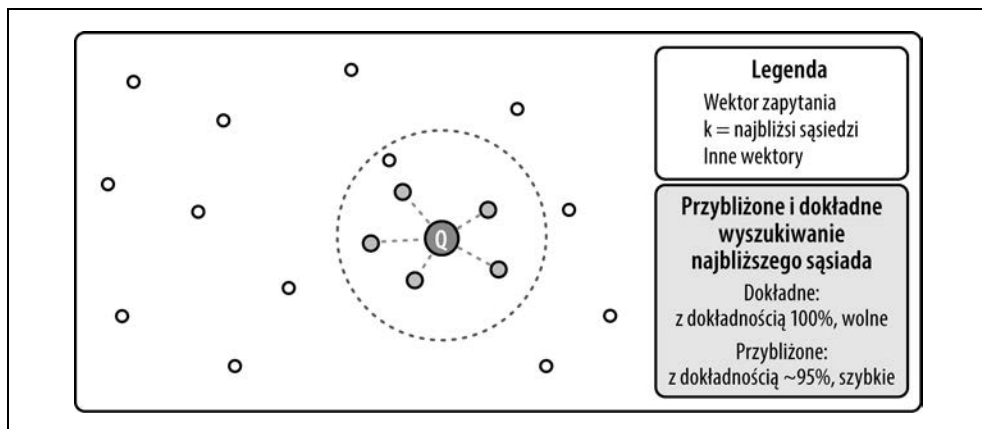
Pierwszą operacją jest ocena podobieństwa, w ramach której określa się, czy dany wektor jest znaczeniowo podobny do wektora referencyjnego. Powszechnie stosowaną metodą jest tutaj *podobieństwo cosinusowe* — obliczanie cosinusa kąta między dwoma wektorami w celu określenia ich podobieństwa (zob. rysunek 1.5). Wektory o zbliżonym znaczeniu mają wartość podobieństwa cosinusowego bliską 1 (wartość maksymalna dla identycznych wektorów), podczas gdy wektory znacznie od siebie oddalone cechują się niskim podobieństwem cosinusowym.



Rysunek 1.5. Geometria podobieństwa cosinusowego

Wyszukiwanie najbliższego sąsiada

Kolejną operacją jest *wyszukiwanie najbliższego sąsiada*, polegające na znalezieniu w zbiorze danych wielu wektorów, które znajdują się blisko wskazanego. W przestrzeni osadzeń wektorowych stosuje się metodę przybliżonego wyszukiwania najbliższego sąsiada (ang. *approximate nearest neighbor ANN*), która dostarcza przybliżone wyniki. Takie podejście jest w zupełności wystarczające dla aplikacji wykorzystujących GenAI (zob. rysunek 1.6), ponieważ z natury mają one charakter probabilistyczny. Dokładne dopasowanie nie jest tutaj ani wymagane, ani pożądane, gdyż mogłoby ograniczyć elastyczność niezbędną do zapewnienia wielu możliwych opcji wyboru na każdym etapie procesu generowania.



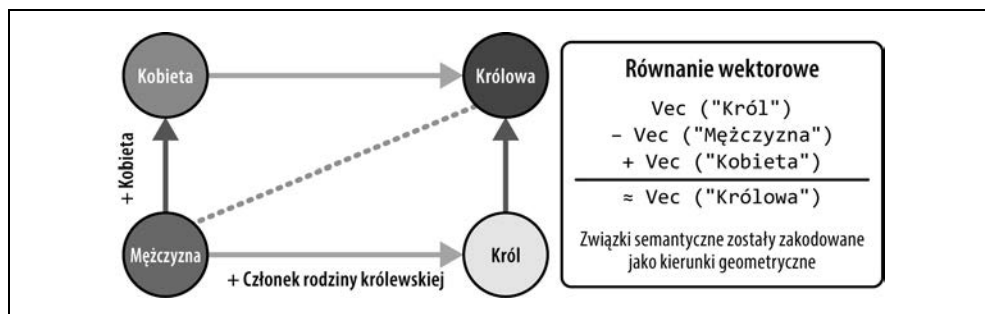
Rysunek 1.6. Wyszukiwanie najbliższego sąsiada

Dodawanie i odejmowanie wektorów

Ostatnią operacją jest *dodawanie i odejmowanie wektorów*, polegające na wykonywaniu operacji matematycznych na wektorach w celu odkrywania zależności lub tworzenia nowych reprezentacji. W jednej z przełomowych prac wprowadzających takie osadzenia wektorowe przedstawiono zaskakujący wynik:

$$\text{Vec}(\text{"królowa"}) \approx \text{Vec}(\text{"król"}) - \text{Vec}(\text{"mężczyzna"}) + \text{Vec}(\text{"kobieta"})$$

$\text{Vec}(\langle \text{słowo} \rangle)$ reprezentuje tutaj osadzenie wektorowe dla danego słowa (zob. rysunek 1.7). Więcej informacji na ten temat znajdziesz w publikacji *Efficient Estimation of Word Representations in Vector Space* (Mikolov i inni, 2013)¹.



Rysunek 1.7. Arytmetyka wektorowa: $\text{król} - \text{mężczyzna} + \text{kobieta} \approx \text{królowa}$

Gdzie wykorzystuje się wektorowe bazy danych?

Wektorowe bazy danych są wykorzystywane w wielu różnych aplikacjach i branżach. Sporo z tych zastosowań istniało jeszcze przed pojawieniem się GenAI i prawdopodobnie zdarzyło Ci się już korzystać z aplikacji opartej na wektorowej bazie danych, nawet o tym nie wiedząc. W miarę jak GenAI — a zwłaszcza aplikacje oparte na LLM-ach — staje się coraz powszechniejsza, wektorowe bazy danych coraz częściej wysuwają się na pierwszy plan.

Oto kilka przypadków użycia znanych większości programistów:

Wyszukiwanie semantyczne

Umożliwia inteligentne zapytania, które rozumieją znaczenie i kontekst, zamiast jedynie dopasowywać słowa kluczowe.

System rekomendacyjny

Dostarcza spersonalizowane rekomendacje produktów, treści lub usług na podstawie preferencji i zachowań użytkowników (np. w witrynach e-commerce).

Pobieranie obrazów i wideo

Wydajnie przechowuje i przeszukuje dane wizualne na potrzeby opartego na treści pobierania obrazów i wideo (np. Google Image Search).

Przetwarzanie języka naturalnego (NLP)

Obsługuje zadania takie jak klasyfikacja tekstu, analiza sentymentu i tłumaczenie języków poprzez reprezentowanie tekstu w postaci wektorów.

Wykrywanie anomalii

Identyfikuje nietypowe wzorce lub zachowania w kontekście cyberbezpieczeństwa lub nadużyć.

¹ <https://arxiv.org/abs/1301.3781>

Bioinformatyka

Analizuje sekwencje genetyczne i struktury białek w biologii obliczeniowej.

Rozpoznawanie obrazów

Wykonuje zadania związane z rozpoznawaniem obiektów, twarzy i scen.

Przetwarzanie audio

Obsługuje rekomendacje muzyczne, rozpoznawanie głosu oraz audio fingerprinting (tworzenie odcisków akustycznych).

Chatboty i konwersacyjna sztuczna inteligencja

Pomagają interpretować intencje użytkowników i dostarczają odpowiedzi, które kontekstowo są znacznie trafniejsze.

Odkrywanie leków

Pomaga w identyfikowaniu potencjalnych kandydatów na leki.

Analiza finansowa

Wspomaga analizę trendów rynkowych, ocenę ryzyka oraz handel algorytmiczny.

SQL kontra wektorowe bazy danych

Aby docenić wyjątkową wartość wektorowych baz danych, warto porównać je z tradycyjnymi bazami SQL (ang. *Structured Query Language*), które od dziesięcioleci stanowią podstawę zarządzania danymi w przedsiębiorstwach.

Podstawy matematyki biznesowej: arytmetyka księgowa

Relacyjne bazy danych — powszechnie nazywane *bazami danych SQL* — od dawna stanowią preferowane rozwiązanie w zakresie zarządzania danymi w przedsiębiorstwach, ponieważ od podstaw opierają się na matematycznych fundamentach biznesu: arytmetyce księgowej. Koncepcja ta znajduje odzwierciedlenie w strukturze i możliwościach baz SQL:

Dane tabelaryczne

Bazy SQL przechowują dane w tabelach złożonych z wierszy i kolumn, co odzwierciedla strukturę arkuszy kalkulacyjnych i ksiąg rachunkowych.

Typy danych

Obsługują one przede wszystkim proste typy danych, takie jak liczby całkowite (ang. *integers*), liczby zmiennoprzecinkowe (ang. *floats*), ciągi tekstowe (ang. *strings*) oraz liczby dziesiętne kodowane dwójkowo (ang. *binary-coded decimals*), które pozwalają na precyzyjne obliczenia walutowe.

Operacje CRUD

Bazy SQL obsługują operacje CRUD — tworzenie (ang. *create*), odczyt (ang. *read*), aktualizację (ang. *update*) i usuwanie (ang. *delete*) — które bezpośrednio odwzorowują typowe procesy biznesowe.

Agregacje i złączenia

Dostarczają potężnych narzędzi do sumowania wartości, łączenia powiązanych danych (tzw. złączenia) oraz raportowania i przeprowadzania analiz finansowych.

Hurtownie danych

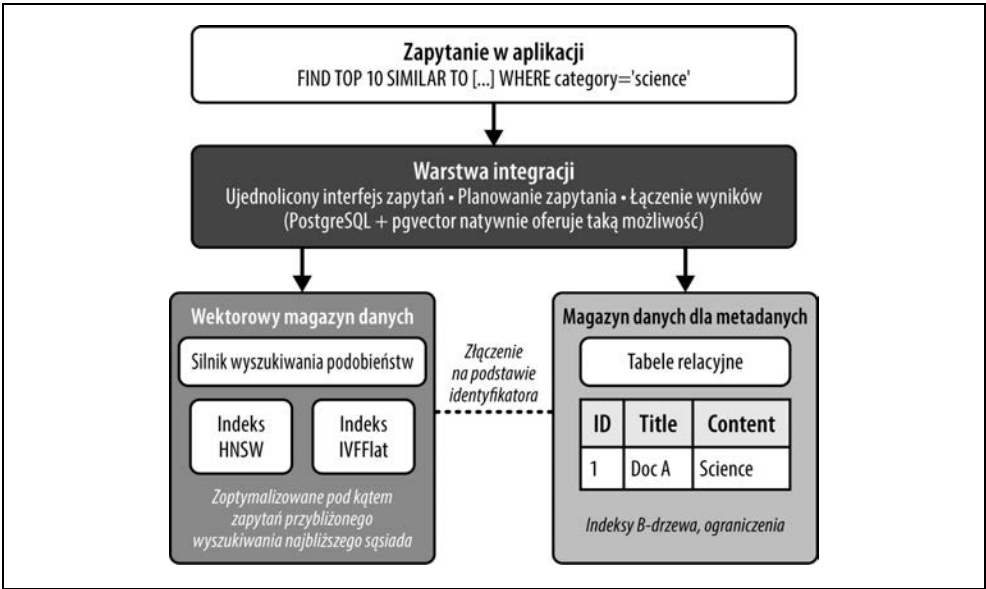
Nowoczesne bazy SQL radzą sobie z tabelami zdenormalizowanymi — tabelami faktów oraz schematami gwiazdy lub płotka śniegu — na potrzeby zapytań w systemach wspomagania decyzji.

Reprezentacja wektorowa w relacyjnym systemie zarządzania bazą danych

Choć formalnie rzecz biorąc, wektory można reprezentować w systemie zarządzania relacyjnymi bazami danych (ang. *relational database management system*, RDBMS), nie jest to idealne rozwiązanie. Wektor można na przykład zapisać jako listę liczb zmiennoprzecinkowych (zob. rysunek 1.8):

```
```sql
CREATE TABLE vector_data (
 id INT PRIMARY KEY,
 vector FLOAT[]
);

INSERT INTO vector_data (id, vector) VALUES (1, ARRAY[0.1, 0.2, ..., 0.768]);
```



Rysunek 1.8. Hybrydowa architektura bazy danych

Jednak takie podejście ma kilka ograniczeń:

#### *Wydajność*

Wykonywanie zapytań i przeprowadzanie operacji na tych wektorach za pomocą operacji SQL jest nieefektywne — zwłaszcza w przypadku danych o wielu wymiarach — ponieważ tradycyjne indeksy nie są zoptymalizowane pod kątem wyszukiwania wektorowego.

#### *Brak operacji wektorowych*

Bazy danych SQL nie obsługują natywnie operacji specyficznych dla wektorów, takich jak wyszukiwanie podobieństwa czy wyszukiwanie najbliższego sąsiada.

#### *Problemy ze skalowalnością*

Wraz ze wzrostem liczby wektorów i wymiarów wydajność zapytań SQL dotyczących wektorów gwałtownie spada.

## **Potrzeba funkcji specyficznych dla wektorów**

Aby w pełni wykorzystać potencjał reprezentacji wektorowych, potrzebujemy baz danych, które potrafią:

#### *Wydajnie indeksować dane wielowymiarowe*

Wektorowe bazy danych wykorzystują specjalistyczne techniki indeksowania, takie jak LSH (ang. *locality-sensitive hashing*) lub HNSW (ang. *hierarchical navigable small world*) — dzięki temu wyszukiwanie podobieństwa jest szybkie i skalowalne.

#### *Natywnie wykonywać operacje na wektorach*

Wektorowe bazy danych oferują wbudowane funkcje obliczania podobieństwa cosinusowego, odległości euklidesowej oraz  $k$  najbliższych sąsiadów.

#### *Optymalizować wyszukiwanie ANN*

W wielu zastosowaniach sztucznej inteligencji znalezienie przybliżonych najbliższych sąsiadów (ANN) jest wystarczające i znacznie szybsze niż przeprowadzanie dokładnego wyszukiwania. Wektorowe bazy danych są projektowane właśnie z myślą o takich zastosowaniach.

#### *Obsługiwać wektory rzadkie*

Wiele modeli AI generuje wektory rzadkie (wektory z wieloma wartościami zerowymi), a wektorowe bazy danych potrafią je wydajnie przechowywać i przeszukiwać.

## **NoSQL i wektorowe bazy danych**

Gdy bazy danych SQL borykały się z problemem danych nieustrukturyzowanych i częściowo ustrukturyzowanych, około 2005 roku pojawiły się bazy NoSQL jako częściowe rozwiązanie problemu. Zaprojektowano je z myślą o obsłudze formatów takich jak JSON i XML, które nie dawały się łatwo odwzorować w tradycyjnych tabelach SQL.

W obliczu rosnącej popularności GenAI warto zastanowić się, czy bazy NoSQL mogą pełnić funkcję wektorowych baz danych.

## Bazy danych NoSQL i wektorowy magazyn danych

Kilku dostawców baz NoSQL wprowadziło rozszerzenia przeznaczone do obsługi wektorów. Do dobrze znanych przykładów należą:

- Redis RediSearch z funkcją wyszukiwania podobnych wektorów;
- MongoDB Atlas, które obecnie zawiera funkcję wyszukiwania wektorowego;
- Elasticsearch, które oferuje gęste pola wektorowe i funkcje wektorowe;
- Apache Cassandra, które można rozszerzyć o możliwości wyszukiwania wektorów za pomocą wtyczek.

## Ograniczenia rozszerzeń wektorowych w bazach danych NoSQL

Chociaż bazy NoSQL potrafią obsługiwać podobieństwo wektorów, w porównaniu z dedykowanymi bazami wektorowymi pozostają one jedynie rozwiązaniami adaptowanymi:

### *Wydajność*

Operacje wektorowe są często słabiej zoptymalizowane, szczególnie w przypadku wielowymiarowych lub bardzo dużych zbiorów danych.

### *Funkcjonalność*

Dostępnych jest mniej operacji i metod indeksowania niż w wyspecjalizowanych systemach.

### *Skalowalność*

Niektóre bazy NoSQL mają trudności z utrzymaniem wydajności, gdy wielkość danych rośnie do poziomu miliardów wektorów.

### *Integracja*

Operacje wektorowe mogą nie współgrać płynnie ze standardowymi zapytaniami, co komplikuje tworzenie zapytań łączących dane wektorowe z niewektorowymi.

## Kiedy wybrać bazę danych NoSQL z rozszerzeniem wektorowym?

Bazy NoSQL z rozszerzeniami wektorowymi sprawdzają się dobrze w następujących scenariuszach:

### *Mieszane typy danych*

Gdy aplikacja przetwarza dane ustrukturyzowane, częściowo ustrukturyzowane i wektorowe, a same wektory nie są jej głównym elementem.

### *Istniejąca infrastruktura*

Gdy intensywnie korzystasz już z baz danych NoSQL i chcesz dodać obsługę wektorów bez konieczności przebudowy całego systemu.

### *Umiarkowana skala*

Gdy ekstremalna wydajność lub ogromna skala nie mają znaczenia kluczowego.

W większości tej książki będę korzystał z bazy danych PostgreSQL z rozszerzeniem pgvector dla aplikacji na poziomie przedsiębiorstwa. Takie rozwiązanie jest w zupełności wystarczające dla małych i średnich systemów, które nie wymagają skali na poziomie obsługi miliardów wektorów.

# Podejścia hybrydowe: łączenie danych ustrukturyzowanych i wektorowych

Wiele rzeczywistych systemów wymaga zarówno danych wektorowych, jak i ustrukturyzowanych danych biznesowych, co prowadzi do tworzenia rozwiązań hybrydowych, które łączą wektorowe bazy danych z tradycyjnymi bazami danych SQL.

## Konieczność obsługi zarówno danych wektorowych, jak i metadanych

Większość praktycznych systemów wymaga również dodatkowych, powiązanych metadanych — ustrukturyzowanych informacji opisujących wektory. Na przykład:

- W systemie wyszukiwania dokumentów każdy dokument może być przedstawiany w postaci wektora, ale potrzebujemy również metadanych, takich jak tytuł dokumentu, autor, data publikacji i kategorie.
- W systemie wyszukiwania obrazów każdy obraz (zdjęcie) może być przedstawiany w postaci wektora, ale potrzebujemy również metadanych, takich jak nazwa pliku, jego wielkość, data utworzenia i tagi. Do metadanych często należą też współrzędne GPS i informacje o aparacie.

## Ograniczenia czysto wektorowego magazynu danych

Przechowywanie metadanych w postaci wektorów jest niepraktyczne z kilku powodów:

### *Elastyczność zapytań*

Znacznie łatwiej jest wykonywać operacje filtrowania i sortowania na ustrukturyzowanych metadanych za pomocą tradycyjnych zapytań do bazy danych. Obecnie w wektorowych magazynach danych nie ma porównywalnych możliwości filtrowania i sortowania.

### *Efektywność przechowywania danych*

Przechowywanie prostych metadanych w postaci wektorów byłoby marnotrawstwem przestrzeni dyskowej.

### *Indeksowanie*

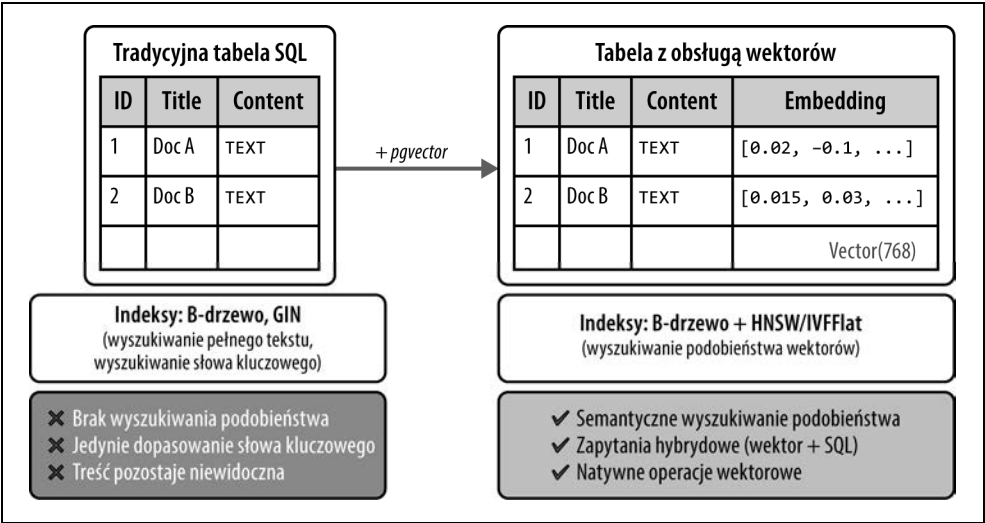
Tradycyjne bazy danych zapewniają wydajne mechanizmy indeksowania ustrukturyzowanych danych, które są kluczowe dla szybkiego wyszukiwania i filtrowania. Indeksowanie wektorowe jest zoptymalizowane pod kątem podobieństwa i klastrowania, a nie tworzenia podzbiorów i filtrowania na podstawie dowolnych predykatów.

### *Spójność danych*

Relacyjne bazy danych oferują funkcje takie jak więzy integralności i transakcje, które są niezbędne do utrzymania spójności ustrukturyzowanych danych.

# Hybrydowa architektura bazy danych

Aby sprostać tym potrzebom, większość systemów wektorowych baz danych przyjmuje obecnie architekturę hybrydową, która łączy wektorowy magazyn danych z możliwościami tradycyjnych baz danych (zob. rysunek 1.9).



Rysunek 1.9. Tabela SQL i tabela z obsługą wektorów

## Komponent wektorowy

Ta część bazy danych jest zoptymalizowana pod kątem przechowywania i odpytywania wektorów wielowymiarowych. Wykorzystuje wyspecjalizowane struktury indeksowania i algorytmy wydajnego wyszukiwania podobieństwa oraz innych operacji na wektorach.

## Komponent metadanych

Często jest to tradycyjna, relacyjna baza danych (np. SQLite) lub baza typu NoSQL, która przechowuje ustrukturyzowane metadane powiązane z poszczególnymi wektorami.

## Warstwa integracji

Zapewnia ujednolicony interfejs przeznaczony do odpytywania komponentu zarówno wektorowego, jak i metadanych, co z kolei umożliwia tworzenie złożonych zapytań, w których łączy się wyszukiwanie podobieństwa wektorów z filtrowaniem metadanych.

# Przykład zapytania hybrydowego

W hybrydowym systemie wektorowej bazy danych typowa kwerenda mogłaby wyglądać następująco:

```
FIND TOP 10 SIMILAR VECTORS TO [0.1, 0.2, ..., 0.5]
WHERE metadata.category = 'science'
AND metadata.publish_date > '2023-01-01'
ORDER BY similarity DESC, metadata.views DESC
```

To zapytanie łączy wyszukiwanie na podstawie podobieństwa wektorów z filtrowaniem i sortowaniem metadanych, co doskonale ukazuje możliwości podejścia hybrydowego. Przedstawiona tutaj składnia ma charakter nieformalny i służy jedynie zilustrowaniu rodzajów operacji, jakie można wykonać w zapytaniu hybrydowym. To nie jest oficjalna składnia żadnej konkretnej hybrydowej bazy danych. Jednak baza danych, która obsługuje zarówno SQL, jak i operacje wektorowe, będzie udostępniać formalną składnię pozwalającą na uzyskanie takiego wyniku. W kolejnych rozdziałach zobaczysz, jak to zrobić za pomocą PostgreSQL z rozszerzeniem pgvector.

## Zalety podejścia hybrydowego

Model hybrydowy oferuje kilka kluczowych korzyści, szczególnie w przypadku małej i średniej wielkości baz danych:

### *Elastyczność*

Architektura hybrydowa pozwala na złożone zapytania, które wykorzystują zarówno podobieństwo wektorów, jak i tradycyjne operacje na danych SQL, umożliwiając filtrowanie najpierw według jednego, a następnie drugiego kryterium.

### *Wydaźność*

Każdy komponent (wektory i metadane) może być zoptymalizowany pod kątem specyficznego typu danych i wzorców zapytań. Indeksy można dobierać oddzielnie dla danych tabelarycznych i wektorowych, aby zoptymalizować wydajność zapytań hybrydowych na podstawie wybieranych i filtrowanych kolumn.

### *Zarządzanie danymi*

Łatwiej jest zarządzać metadanymi, aktualizować je i utrzymywać oddzielnie od danych wektorowych. Tabela metadanych jest zazwyczaj oddzielona od tabeli danych wektorowych, a klucz obcy łączy każdy wektor z powiązanymi z nim metadanymi.

W przypadku baz danych zawierających miliardy obiektów bardziej odpowiednie mogą okazać się czysto wektorowe bazy danych.

## Podsumowanie

Wektorowe bazy danych stanowią ważny etap ewolucji technologii danych, napędzany potrzebami GenAI i uczenia maszynowego. Wprowadzają one koncepcje takie jak wyszukiwanie semantyczne i osadzenia wektorowe dla danych nieustrukturyzowanych — tekstu, dźwięku, obrazów i wideo. W rozwiązaniach hybrydowych uzupełniają one tradycyjne bazy danych, zamiast je zastępować.

Przyszłość zarządzania danymi prawdopodobnie leży w systemach hybrydowych, które wydajnie obsługują zarówno dane ustrukturyzowane, jak i wektorowe. W miarę jak sztuczna inteligencja będzie obejmować kolejne dziedziny, znaczenie wektorowych baz danych i architektur hybrydowych będzie tylko rosło.

Zrozumienie zalet i ograniczeń wektorowych baz danych — oraz tego, jak integrują się one z szerszym ekosystemem zarządzania danymi — ma znaczenie kluczowe dla badaczy danych, inżynierów i decydentów tworzących aplikacje AI nowej generacji. Ponieważ dziedzina ta stale się rozwija, można spodziewać się kolejnych innowacji, które wypełnią lukę między wektorowymi i tradycyjnymi podejściami do zarządzania danymi.

# PROGRAM PARTNERSKI

— GRUPY 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  
**Helion**

# Zbuduj inteligentne AI — opanuj wektorowe bazy danych!

Sztuczna inteligencja generatywna, duże modele językowe i systemy RAG zdominowały rynek IT, a u ich podstaw leży technologia, którą większość programistów wciąż słabo zna: wektorowe bazy danych. To właśnie one umożliwiają wyszukiwanie semantyczne, zasilają ChatGPT, Claude i lokalne modele LLM, pozwalają również budować inteligentne aplikacje operujące na nieustrukturyzowanych danych: tekście, obrazach i dźwięku. Zapotrzebowanie na specjalistów rozumiejących tę technologię rośnie lawinowo, a literatura fachowa w tym obszarze pozostaje rozproszona i fragmentaryczna. Książka Nitina Borwankara wypełnia tę lukę — to pierwsze tak kompleksowe, praktyczne źródło wiedzy o wektorowych bazach danych.

Publikacja prowadzi czytelnika od teorii osadzeń wektorowych (Word2Vec, Doc2Vec, transformery), przez bibliotekę FAISS i rozszerzenia SQLite VSS, a także PostgreSQL pgvector, aż po budowę kompletnych systemów RAG. Kolejne rozdziały to gotowe projekty: system wyszukiwania treści z Reddita, wyszukiwarka artykułów naukowych z ArXiv, lokalny asystent RAG oparty na Ollamie, narzędzie do przeszukiwania historii konwersacji z AI. Całość wieńczy autorska propozycja VQL zunifikowanego języka zapytań dla wektorowych baz danych, wzorowanego na SQL.

## Najważniejsze zagrożnienia

- Osadzenia wektorowe — teoria, ewolucja i praktyczne zastosowanie w modelach LLM
- Wyszukiwanie podobieństw z biblioteką FAISS — indeksy płaskie, IVF, HNSW i kwantyzacja
- Hybrydowe bazy danych SQLite VSS i PostgreSQL pgvector w projektach AI
- Budowa systemów RAG — od prostego prototypu po naukowy asystent badawczy
- Wyszukiwanie semantyczne kontra słów kluczowych — kiedy i jak stosować każde podejście
- Lokalne wdrożenia z Ollamą — prywatność, kontrola kosztów i pełna niezależność od chmury

**Nitin Borwankar** jest doświadczonym analitykiem danych i specjalistą do spraw baz danych z trzydziestoletnim stażem. Aktywnie działa na rzecz edukacji w obszarze danych i otwartych narzędzi uczenia maszynowego. Regularnie występuje na konferencjach branżowych, prezentując pragmatyczne podejście do AI i modeli LLM.

<b>Helion</b>	<b>KOD KORZYŚCI</b> Sięgnij po więcej! ▶	
 <b>helion.pl</b>	ISBN 978-83-289-4206-6	
 <b>HELION S.A.</b> ul. Kościuszki 1c 44-100 Gliwice tel.: 32 250 98 63 helion@helion.pl	 9 788328 942066	
<b>Cena: 79,00 zł</b>		