ГЛАВА 1

Знакомство с векторными базами данных

Захлестнувший весь современный мир поток генеративного искусственного интеллекта (GenAI), прежде всего систем обработки естественного языка (NLP), вынес технологию векторных БД в центр внимания разработчиков. Неструктурированные данные из обучающего датасета приводятся к векторной форме, а те, что используются моделями для работы, нередко хранятся в векторных БД. И все-таки эта технология еще не слишком известна и понятна среднему разработчику приложений на основе БД.

В этой главе мы вкратце обсудим векторные БД, их основные компоненты и назначение — то, что будем подробно рассматривать в дальнейшем. К концу главы вы поймете, как использовать эту технологию. Подробное обсуждение еще одной важнейшей темы этой книги — эмбеддингов — нам предстоит уже в следующей главе.

Для чего нужны векторные БД

Обработка неструктурированных данных — текста, изображений, аудио и видео — и раньше, до появления GenAI, была непростой задачей, требовала особых технологий и знаний. Индексация и поиск были возможны только по метаданным, к примеру дате создания, автору, размерам и т. д. А вот найти изображение, похожее на заданное, или отрывок текста, схожий по смыслу с другим, средствами реляционных баз данных было нельзя: SQL предназначен для поиска синтаксических, а не семантических (смысловых) соответствий.

Сейчас, когда векторные БД становятся привычным делом, работа с неструктурированными данными значительно упростилась. Векторные БД — это новые приложения, в основе которых лежит *семантический поиск*. Он расширяет возможности обычного поиска по ключевым словам, не позволяющего различить, где, например, слово «лук» означает овощ, а где — оружие.

Для систем искусственного интеллекта (ИИ) векторные БД важны настолько же, насколько реляционные — для систем обработки бизнес-данных. Сегодня мы можем создавать мощные интеллектуальные интерактивные системы, в которых такая обработка будет дополнена семантическим поиском по тексту,

изображениям, аудио и видео. Однако для этого необходимо хорошо знать принципы работы векторных БД: на данный момент общедоступной информации по этой теме недостаточно.

Доступные в настоящее время специализированные векторные БД несколько различаются по устройству, и для работы с ними необходимо знать их специфику. Часть из них работает в облаке, а значит, требует меньше усилий по обслуживанию; часть — как, например, Postgres с векторным расширением — можно запускать локально.

В этой книге мы будем обсуждать системы, где векторные возможности тесно интегрированы в реляционные БД. Благодаря этому с данными обоих видов — структурированными и неструктурированными — можно работать привычным образом и выполнять обычные для БД операции: резервное копирование, восстановление, партиционирование, управление доступом. Для сохранения векторных (то есть неструктурированных) данных в обычных таблицах любого назначения не требуется специальная обработка.

До появления векторных БД неструктурированные данные обычно хранились в полях типа BLOB (binary large object — большой бинарный объект) или в отдельных файлах, ссылки на которые заносились в таблицы БД. В обоих случаях данные оставались непрозрачными, недоступными для каких-либо операций помимо сохранения и выборки. Единственным исключением были тексты: в них можно было вести поиск по ключевым словам.

Новый тип данных: вектор

Что представляют собой «векторные данные» и почему нельзя использовать вместо них обычный BLOB? Можно. Однако именно векторы добавляют в данные смысловой компонент и готовят их к обработке при помощи GenAI. В языковых моделях слова, предложения, абзацы и даже целые документы представляются в виде векторов, а значит, для эффективной работы в пространстве ИИ необходимо хорошо понимать, как устроены данные этого типа. Давайте рассмотрим подробнее.

Прежде всего, чтобы начать с чего-то знакомого, представим векторы как новый тип данных в обычной СУБД. Возьмем, например, Postgres: поскольку она популярна и хорошо известна многим разработчикам, именно с ней в основном мы и будем работать в дальнейшем.

Благодаря системе расширений в Postgres можно вводить новые типы данных. В случае векторов нам потребуется расширение pgvector: оно позволяет добавлять в таблицы столбцы типа `VECTOR` и выполнять по ним векторный *поиск по сходству* (similarity search). Если посмотреть на содержимое, хранящееся в таком столбце (рис. 1.1), мы увидим длинные списки чисел с плавающей точкой. Однако за ними кроется намного больше, чем может показаться.

ID	Title	Embedding VECTOR(768)
1	Doc A	[0.023, –0.145, 0.892, 0.034, ..., –0.567, 0.123]

768 чисел с плавающей точкой

Каждое число определяет какую-то часть семантического значения

Рис. 1.1. Данные в столбце типа VECTOR

Поиск по сходству

Возможность поиска по сходству — то, что принципиально отличает данные типа `VECTOR` от набора упорядоченных списков чисел, с которыми, помимо сохранения и выборки, практически ничего сделать нельзя. Поиск по сходству позволяет находить фрагменты текста, близкие по смыслу к заданному, и следовательно, создавать приложения, в которых с базами данных работают большие языковые модели (LLM), подобные ChatGPT, Claude или локальной Llama3.x. В Postgres с расширением pgvector для поиска по сходству есть специальный оператор. Его мы рассмотрим в следующей главе, где речь пойдет об эмбеддингах.

Пример поиска по сходству

Представьте себе поиск по корпоративной базе знаний. Вы вводите, например, «Как вернуть деньги?». Что происходит дальше, показано на рис. 1.2.

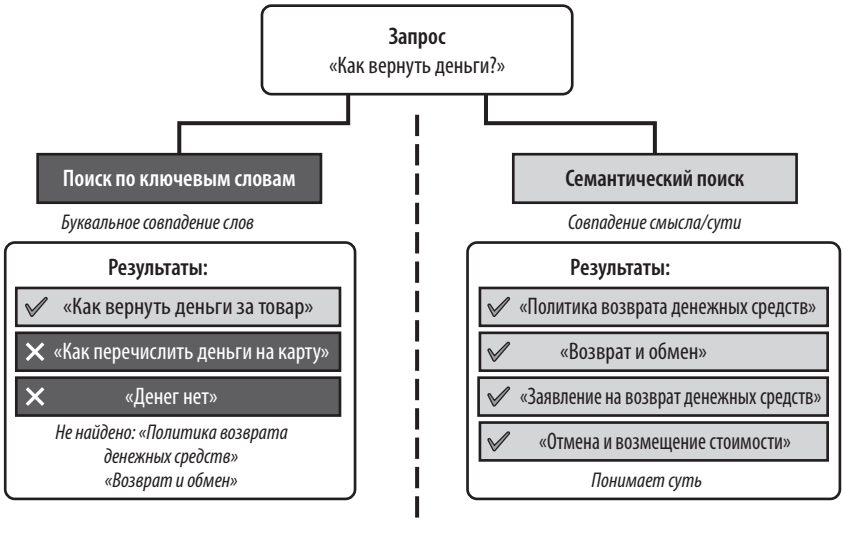


Рис. 1.2. Сравнение поиска по ключевым словам с семантическим поиском

Поиск по ключевым словам (буквальному совпадению слов). При *поиске по ключевым словам* (keyword search) система ищет точные вхождения набранных вами слов

(или же близких к ним словоформ), и это сработает, если такие слова («деньги», «назад») или же словосочетания («как получить») в тексте есть. Однако система пропустит страницы, в которых тот же смысл выражен другими словами, к примеру:

- «Политика возврата денежных средств»
- «Возврат и обмен»
- «Возмещение стоимости»
- «Отмена заказа»

Итак, при поиске по ключевым словам можно найти:

- страницу «Как вернуть деньги за товар» (хорошо, если такая есть);
- страницы, где говорится о деньгах, но совершенно в ином смысле (к примеру, «Как перечислить деньги на карту»).

Семантический поиск (по совпадению смыслов). При *семантическом поиске* (semantic search) система пытается понять запрос и отыскать что-то близкое по смыслу. Поэтому она найдет, например:

- «Заявление на возврат денежных средств»
- «Возврат товара»
- «Отмена и возмещение стоимости»
- «Отмена сделки»

Значит, при семантическом поиске, вероятно, найдутся:

- страница «Возврат средств и товаров»;
- ее подраздел «Как потребовать возврата денежных средств после покупки».

Краткий итог. Сравнивая два метода поиска, можно сказать, что:

- поиск по ключевым словам «находит страницы, содержащие эти слова»;
- семантический поиск «находит страницы, отвечающие на вопрос, независимо от формулировок».

Еще пример

Рассмотрим еще один небольшой пример. Допустим, вы вводите «Мой ноутбук не включается».

- При поиске по ключевым словам вы не найдете статьи «Устройство не загружается» и «Поломка блока питания».
- При семантическом поиске система с большой вероятностью свяжет слова «не включается» с проблемами при загрузке и поломкой блока питания.

Раскроем секрет этого волшебства.

Эмбеддинг

Благодаря процессу, который называется *эмбеддингом* (embedding), фрагменты неструктурированных (нетабличных) данных преобразуются в векторы — списки чисел с плавающей точкой (рис. 1.3). При этом контекст, в котором употребляются слова или предложения, сохраняется: близким по значению словам соответствуют векторы, расположенные рядом друг с другом в векторном пространстве. Грубо говоря, смысл слов четко ассоциируется с контекстом, в котором они употребляются.

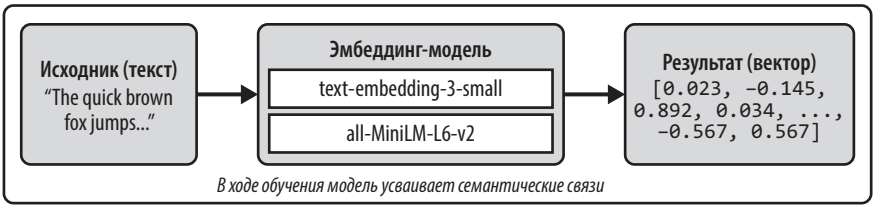


Рис. 1.3. Процесс эмбеддинга

Вернемся к примеру со словом «лук». Когда оно или предложение с ним находится в контексте других предложений об овощах, огороде или приготовлении супа, ясно, что «лук» в данном случае — это овощ. Если же перед нами текст об охоте, сражениях или воинах древности, то очевидно, что слово «лук» означает оружие. Выходит примерно так, как показано на рис. 1.4. Этот пример, хоть и весьма приблизительный, передает основную мысль. Пока что этого достаточно. Не бойтесь: подробный разговор об эмбеддингах ждет нас в следующей главе.

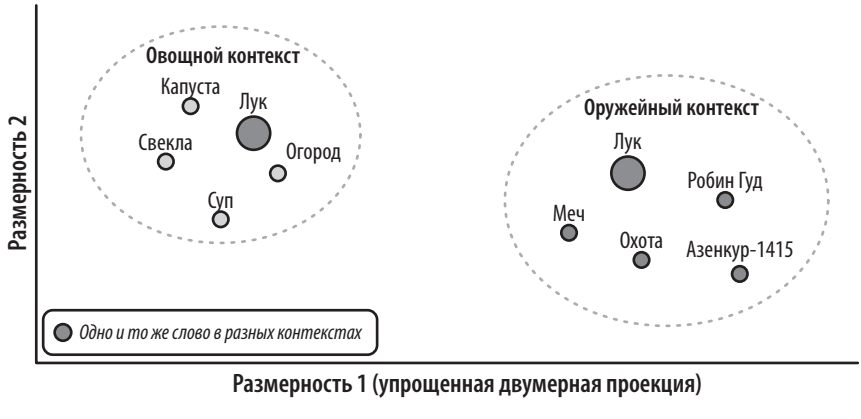


Рис. 1.4. Кластеризация векторного пространства

Необходимо пояснить, что слово «эмбеддинг» тоже имеет несколько значений, и это может немного запутать новичков.

Во-первых, эмбеддинг — процесс, в результате которого данным приводятся в соответствие векторы и, что более важно, набор данных отображается в векторном

пространстве. К примеру, несколько страниц текста или целых книг, собирательно называемых *корпусом*, в ходе эмбеддинга превращаются в набор векторов, подчиняющихся правилам векторного пространства.

Во-вторых, эмбеддингом называют сам результат процесса эмбеддинга — тот самый набор векторов или отдельный вектор из этого набора (что обычно нетрудно определить по контексту). Хорошая аналогия — слово «расчет», которое также обозначает и процесс, и его результат.

Особенности векторного типа данных

Новый тип данных сопровождается набором операций, которые необходимы для работы с векторами. Без этих операций немыслимо выполнение многих задач GenAI и систем машинного обучения — распознавания образов, понимания естественных языков, выдачи рекомендаций.

Сравнение на сходство

Прежде всего, нужно поговорить о сравнении на сходство, при помощи которого можно установить, насколько данный вектор похож по смыслу на эталонный. За меру чаще всего принимается *косинусное сходство* (cosine similarity) — косинус угла между двумя векторами (рис. 1.5). У близких по смыслу векторов его значение стремится к единице (максимум, означающий идентичность); чем ближе к нулю, тем меньше сходство.

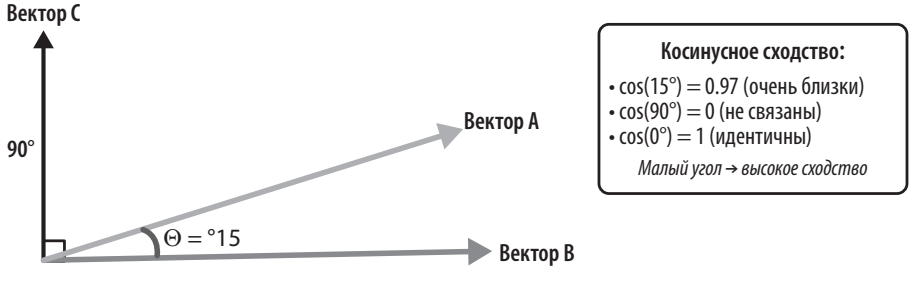


Рис. 1.5. Геометрия косинусного сходства

Поиск ближайшего соседа

Следующая операция — *поиск ближайшего соседа* (nearest-neighbor search, NN): с ее помощью можно найти векторы, расположенные в датасете недалеко от заданного. Однако в вероятностных по природе GenAI-системах точное нахождение ближайшего соседа не только не требуется, но и нежелательно, так как не позволяет на каждом этапе генерации получать по несколько вариантов продолжения. Поэтому применяется другой метод — *приближенный поиск ближайших соседей* (approximate nearest neighbors, ANN), который дает, соответственно, несколько приближенных результатов (рис. 1.6).

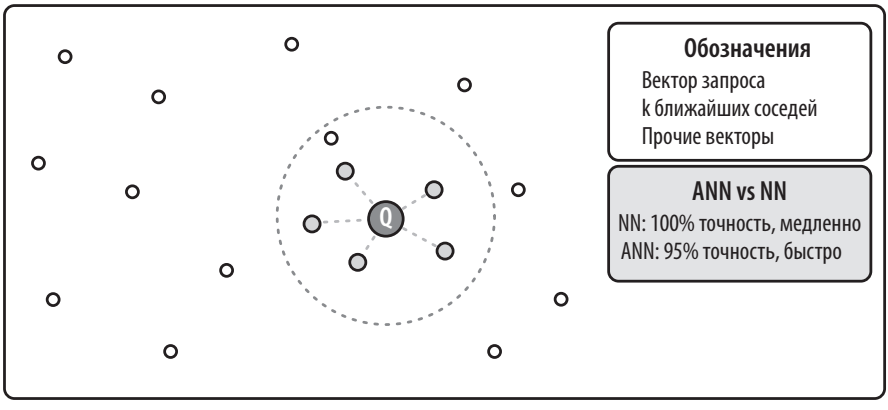


Рис. 1.6. Поиск ближайшего соседа

Сложение и вычитание векторов

Наконец, еще два действия — *сложение и вычитание*, в ходе которых посредством математических операций с векторами выявляются связи между ними или же строятся новые векторы. В одной оригинальной статье об эмбеддингах приводится такой пример:

```
Vec("Королева") ≈ Vec("Король") – Vec("Мужчина") + Vec("Женщина")
```

где `Vec(<слово>)` — эмбеддинг этого слова (рис. 1.7). Более подробные объяснения можно найти в статье https://oreil.ly/dtoM7 (Mikolov et al., 2013).

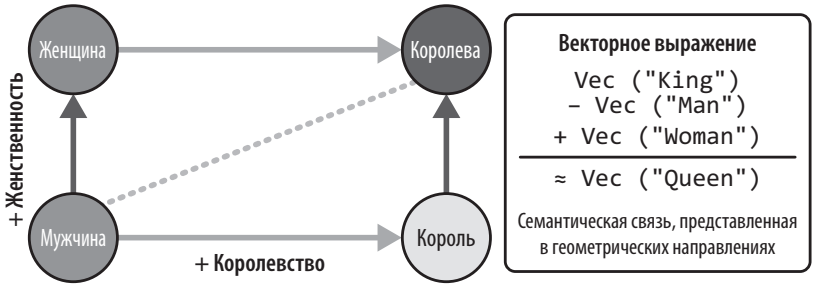


Рис. 1.7. Векторная арифметика: Король – Мужчина + Женщина ≈ Королева

Где применяются векторные БД

Векторные БД применяются во многих предметных областях и чаще всего в GenAI-системах. Чем больше ИИ, а особенно LLM-приложения, участвуют в нашей жизни, тем чаще и интенсивнее применяются векторные БД. Поэтому, вероятно, вам уже приходилось работать с ними, пусть даже опосредованно и не осознавая этого.

Можно назвать несколько областей применения, известных большинству разработчиков:

Семантический поиск

Интеллектуальные запросы, которые требуют понимания смысла и контекста, а не простого поиска совпадающих слов.

Рекомендательные системы

Персонализированный контент, продукты или услуги, рекомендуемые на основании предпочтений и поведения пользователей (к примеру, в интернет-магазинах).

Поиск изображений и видео

Эффективное хранение и поиск визуальных данных с последующей выдачей пользователю (например, Google Image Search).

Понимание естественных языков (NLP)

Классификация, анализ тональности и перевод текста на другие языки возможны благодаря его представлению в виде векторов.

Обнаружение аномалий

Выявление необычных паттернов и поведения в системах кибербезопасности и борьбы с мошенничеством.

Биоинформатика

Анализ генетических последовательностей и белковых структур в вычислительной биологии.

Компьютерное зрение

Распознавание объектов, лиц и мест.

Обработка звука

Рекомендации музыки, распознавание голоса, звуковые отпечатки пальцев.

Чатботы и диалоговые ИИ-системы

Помощь в интерпретации намерений пользователя, выдача ответов, соответствующих контексту.

Поиск новых лекарств

Помощь в разработке новых лекарственных препаратов.

Финансовый анализ

Помощь в анализе рыночных тенденций, оценка рисков, алгоритмический трейдинг.

Сравнение векторных и реляционных БД

Чтобы оценить уникальные достоинства векторных БД, полезно сравнить их с привычными реляционными.

Основы бизнес-математики: бухгалтерская арифметика

Реляционные БД, нередко называемые базами данных SQL, — проверенное решение для хранения корпоративных данных и управления ими. Математические принципы ведения бизнеса — бухгалтерская арифметика — закладывались в эти БД изначально, а потому повлияли на их структуру и возможности:

Табличные данные

В реляционных БД данные организованы в виде таблиц — строк и столбцов, — которые повторяют структуру бухгалтерских книг и учетных журналов.

Типы данных

В реляционных БД хранятся данные самых простых типов: строки, целые числа, числа с плавающей точкой, а также двоично-десятичные числа (BCD, binary-coded decimals), которые нужны для точных подсчетов при обмене валют.

CRUD-операции

Создание, чтение, обновление и удаление (create, read, update, delete — CRUD) — все эти операции в точности соответствуют общераспространенным бизнес-процессам.

Агрегация и соединение

Эффективные средства для суммирования значений, соединения связанных данных, а также поддержки финансовой отчетности и анализа.

Хранение данных

Современные реляционные БД поддерживают денормализованные таблицы — таблицы фактов, схемы «звезда» или «снежинка», — которые применяются в системах принятия решений.

Векторное представление в реляционных БД

В реляционной БД можно хранить и векторы: к примеру, представить их в виде массивов чисел с плавающей точкой (рис. 1.8). Однако такой подход далек от идеала.

```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]);
```

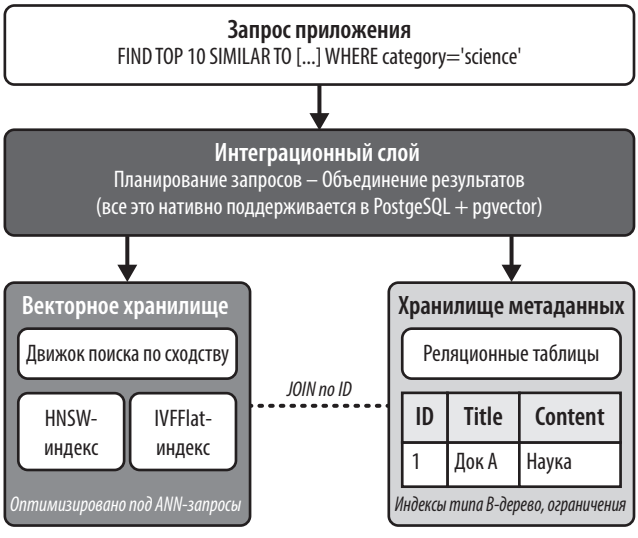


Рис. 1.8. Архитектура гибридной базы данных

У такого подхода существует ряд ограничений:

Производительность

Поскольку традиционные индексы не оптимизировались под поиск векторов, выборка и другие операции с ними при помощи обычных SQL-запросов неэффективны, особенно при высоких размерностях.

Отсутствие операций с векторами

В реляционных БД нет нативной поддержки векторных операций: поиска по сходству или поиска ближайшего соседа.

Проблемы с масштабируемостью

Чем больше векторов, чем выше их размерность, тем ниже производительность SQL при выполнении запросов.

Потребность в специфичных возможностях векторов

Чтобы в полной мере использовать возможности векторного представления данных, в БД должны поддерживаться соответствующие функции, а именно:

Эффективная индексация данных с высокой размерностью

В векторных БД используются особые методы индексации, например LSH (locality-sensitive hashing, локально-чувствительное хеширование) и HNSW (hierarchical navigable small world, индексы на основе иерархического маленького мира), благодаря которым поиск по сходству работает быстро и независимо от масштабов.

Нативная поддержка векторных операций

В векторные БД уже встроена поддержка вычисления косинусного сходства и евклидова расстояния, а также поиска *k* ближайших соседей.

Оптимизация под поиск ANN

Поиск нескольких ANN намного быстрее, чем точный поиск ближайшего соседа. При этом для большинства ИИ-приложений вполне достаточно ANN. Векторные БД проектируются с учетом этих фактов.

Работа с разреженными векторами

На выходе многих ИИ-моделей образуются разреженные векторы (sparse vectors), то есть векторы с большим количеством нулей. Векторные БД эффективно хранят их и работают с ними.

Сравнение векторных и NoSQL баз данных

Для реляционных БД полностью и частично неструктурированные данные представляют проблему. Чтобы хоть как-то ее решить, в 2005 году появились нереляционные (NoSQL) БД. Они способны работать с данными в форматах, подобных JSON или XML, которые трудно поместить в обычные SQL-таблицы.

Теперь, в эпоху GenAI, имеет смысл подумать: не будут ли NoSQL БД эффективным хранилищем для векторов?

NoSQL БД как векторные хранилища

Несколько провайдеров NoSQL предложили расширения для работы с векторами. Вот самые известные из них:

- Redis RediSearch поддерживает поиск по сходству;
- MongoDB Atlas также поддерживает векторный поиск;
- в Elasticsearch предусмотрены поля для хранения плотных векторов (dense vectors) и функции для работы с ними;
- Apache Cassandra предлагает набор плагинов для векторного поиска.

Ограничения векторных расширений для NoSQL БД

Несмотря на поддержку векторных операций, NoSQL БД изначально не были рассчитаны на их выполнение, а потому уступают векторным БД по ряду показателей:

Производительность

Векторные операции в NoSQL БД нередко плохо оптимизированы, особенно при больших объемах и высоких размерностях.

Функциональность

NoSQL БД поддерживают меньше операций и способов индексации, чем специализированные векторные БД.

Масштабируемость

Когда счет векторов идет на миллиарды, производительность некоторых NoSQL БД заметно снижается.

Интеграция

Векторные операции не всегда хорошо сочетаются с обычными, что усложняет написание запросов для работы с векторами и данными других типов.

Когда имеет смысл использовать NoSQL БД

NoSQL БД с векторными расширениями могут неплохо работать в следующих сценариях:

Смешанные типы данных

Если приложение работает со структурированными, полуструктурированными и векторными данными, однако последние не так важны.

Существующая инфраструктура

Если NoSQL БД уже активно используется, однако необходимо добавить в нее векторные функции без полной перестройки системы.

Средние масштабы

Когда экстремальная производительность или масштабы не требуются.

В этой книге по большей части будет использоваться PostgreSQL с расширением pgvector. Этого вполне достаточно для корпоративных приложений небольшого и среднего масштаба, в которых количество векторов не достигает миллиарда.

Гибридные подходы: объединение векторных и структурированных данных

Поскольку многим реальным приложениям приходится работать как с векторами, так и со структурированными корпоративными данными, появились гибридные решения, в которых векторная БД сочетается с традиционной.

Необходимость в векторах и метаданных

В большинстве реальных систем используются метаданные — структурированная информация, которая описывает векторы. К примеру:

- В системе поиска документации каждый документ описывается вектором. Однако нужны и метаданные: заголовок, автор, категория, дата публикации.
- В системе поиска изображений каждая картинка описывается вектором. Однако нужны и метаданные: имя файла, размер, дата создания, теги. Нередко сюда добавляются координаты GPS и сведения о камере.

Ограничения чисто векторных хранилищ

По ряду причин хранить метаданные в виде векторов непрактично:

Гибкость запросов

Фильтрацию и сортировку структурированных метаданных намного проще выполнять в традиционной БД. На данный момент ни в одном из векторных хранилищ нет столь же эффективных средств.

Эффективность хранения

Хранение простых метаданных в виде векторов — напрасная трата объема хранилища.

Индексация

Традиционные БД предлагают эффективные средства индексации структурированных данных, а это критически важно для поиска и фильтрации. Индексация векторов оптимизирована под поиск по сходству и кластеризацию, но не фильтрацию и выделение подмножеств по произвольным признакам.

Целостность данных

Целостность структурированных данных непросто обеспечить без ограничений и транзакций, активно поддерживаемых традиционными БД.

Гибридная архитектура БД

Чтобы решить все эти проблемы, большинство векторных БД сегодня имеют гибридную архитектуру, в которой сочетаются возможности векторного хранилища и традиционной БД (рис. 1.9):

Векторное хранилище

Часть базы данных, оптимизированная под хранение и выборку векторов высокой размерности. Используются алгоритмы и методы индексации, рассчитанные на эффективный поиск по сходству и другие векторные операции.

Хранилище метаданных

Нередко это традиционная (к примеру, SQLite) или NoSQL БД, в которой хранятся структурированные метаданные векторов.

Интеграционный слой

Унифицированный интерфейс для обращения к обоим хранилищам, который позволяет делать сложные запросы с использованием как поиска по сходству, так и фильтрации метаданных.

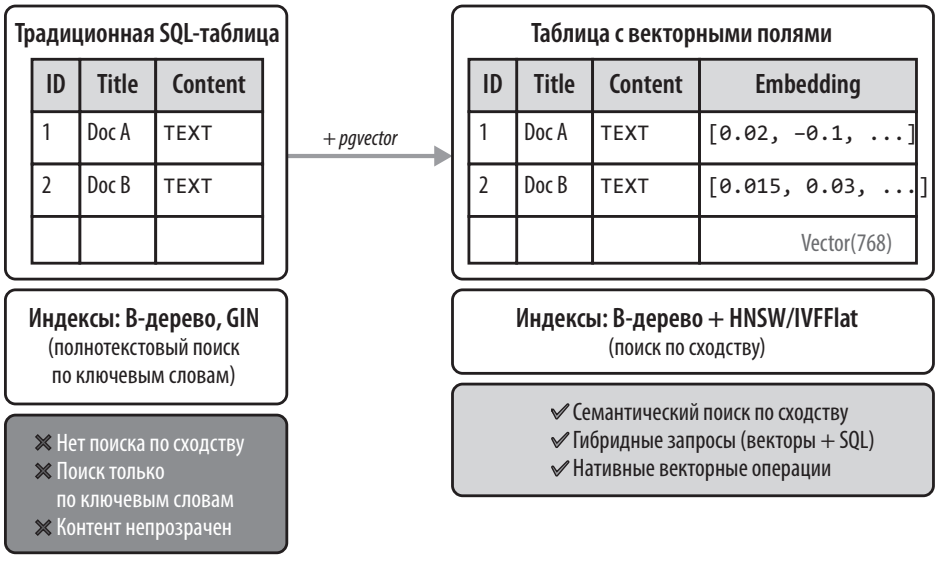


Рис. 1.9. Сравнение таблиц с векторными полями и SQL-таблиц

Пример гибридного запроса

Типовой запрос в гибридной векторной БД выглядит так:

```
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
```

Здесь векторный поиск по сходству объединен с фильтрацией и сортировкой метаданных. В этом и заключается преимущество гибридного подхода. Показанный выше синтаксис не является формальным, общим для всех гибридных БД: это лишь демонстрация возможностей. Однако в любой БД, которая их поддерживает, предусмотрен и соответствующий формальный синтаксис. В последующих главах я покажу, как писать подобные запросы в PostgreSQL с расширением pgvector.

Преимущества гибридного подхода

Гибридная модель имеет несколько важных преимуществ, особенно для БД небольших и средних размеров:

Гибкость

Гибридная архитектура позволяет выполнять сложные запросы, в которых поиск по сходству сочетается с традиционными SQL-операциями, включая фильтрацию.

Производительность

Оба хранилища (для векторов и метаданных) можно оптимизировать под соответствующие типы данных и шаблоны запросов. В целях эффективного выполнения гибридных запросов для векторных и табличных данных можно выбрать отдельные индексы.

Управление данными

Раздельное хранение векторов и метаданных значительно упрощает задачи управления, обновления и обслуживания. Метаданные хранятся в отдельных таблицах и связываются с соответствующими векторами при помощи внешних ключей.

Тем не менее хранить миллиарды объектов может быть удобнее в чисто векторной БД.

Выводы

Векторные БД — это большой шаг вперед в развитии технологий обработки данных, обусловленный потребностями ИИ и машинного обучения. Векторные БД реализуют возможности семантического поиска и перевода неструктурированных данных — текста, изображений, аудио и видео — в векторную форму. В гибридных системах векторные БД дополняют, но не заменяют традиционные.

Будущее обработки данных, вероятно, за гибридными системами, эффективно работающими как с векторными, так и со структурированными данными. По мере того как ИИ будет осваивать новые предметные области, векторные и гибридные БД будут приобретать все более важное значение.

При создании ИИ-систем нового поколения специалистам data science, инженерам и руководителям необходимо понимать сильные и слабые стороны векторных БД, а также особенности их интеграции в более широкую экосистему управления данными. По мере дальнейшего развития этой области можно ожидать появления новых решений, которые преодолеют разрыв между векторными и традиционными подходами к управлению данными.