Back to Blog Home

What is a Vector Database? A Beginner's Guide

Yugabyte Team
August 11, 2025

Modern organizations use a variety of databases to manage information, from relational databases and NoSQL systems to distributed databases that scale across nodes. As artificial intelligence (AI) and machine learning (ML) models continue to grow in popularity, a new category has entered the mix: vector databases. These specialized databases are specifically designed to handle the unique data needs of AI, especially the high-dimensional numeric data (vectors) that encode unstructured content like text, images, and audio.

This blog provides a clear definition of vector databases and answers key questions about how they work, how they differ from regular databases, and when you should use them for your AI and ML projects.

What is a vector database?

A vector database is a type of database optimized to store data as mathematical vectors – essentially, fixed-length arrays of numbers that represent items or information. Unlike traditional databases (which store data in tables of rows and columns), modern vector databases organize information in an n-dimensional vector space. Each data item is encoded as a point in this space, allowing the database to compare items by their distance or similarity rather than by exact matching of text or values.

How does a vector database work?

In practical terms, a vector database enables efficient similarity search. You can query the database with a vector (such as the embedding of a sentence or an image), and it will retrieve records whose vectors are closest to that given query vector.

For example, imagine an e-commerce catalog. A relational database can easily find all products where the category is "sneakers" or the color is "red" – those are exact matches on structured attributes. But what if we want to find products similar to a given sneaker (in style or shape), or images that resemble a reference image? A vector database addresses this requirement by representing each product image (or description) as a high-dimensional vector, capturing its features.

When you search with a target item's vector, the database finds other items with vectors clustered near it, meaning they are similar in the encoded feature space. This ability to retrieve "things like this thing" makes advanced vector databases especially suited for AI tasks like image similarity search, recommendation systems, and semantic text search.

What is vector data and how is it stored?

Vector database tools store numeric vector representations (embeddings) of objects such as text, images, audio, or other complex data types. Rather than storing raw unstructured data directly, a vector search engine stores the embeddings produced by ML models – these are the high-dimensional vectors that encode the essential information or semantics of the data.

Search Blogs

Popular Topics

PostgreSQL Compatibility

Database Migration

Database Sharding

Geo-Distribution

Customer Stories

Change Data Capture

Categories

ACID Transactions
AI
Amazon Aurora
Amazo Dynam
Amazon Web Services
Apache Cassar
Careers
Change Captur
Cloud Providers
Cockro
Community News
Compa News
Compare and contrast
Contair

where the number of coordinates equals the embedding size. The idea is that semantically or conceptually similar data will produce vectors that are close together (low distance) in this space, while dissimilar data yields distant vectors.

Storing vectors in databases poses new challenges, because traditional indexing methods (like B-trees or hash indexes on single values) are not effective for high-dimensional similarity search. This means that vector databases need to use specialized data structures and indexes to store and retrieve vectors efficiently. Common approaches include graph-based indexes (e.g. HNSW – Hierarchical Navigable Small World graphs) and quantization or hashing techniques for Approximate Nearest Neighbor (ANN) search.

These indexes are designed to quickly approximate nearest neighbors without comparing a query to every single vector in the database (which would be prohibitively slow at scale). For example, HNSW organizes vectors in layers of small-world graphs, enabling sub-linear search time by navigating through "neighbor" links. IVF (Inverted File Index) then partitions the vector space into cells to narrow search scope.

Vector databases are typically powered by k-nearest neighbors (k-NN) indexes built on algorithms like HNSW or IVF, offering fast lookup of nearest neighbors in an N-dimensional space. Put simply, the storage of embeddings is optimized so that when you add new vectors, the database updates its structures to keep similar vectors grouped. Then when you search, it zeroes in on relevant regions of the vector space instead of scanning everything.

What is an example of a vector database?

Several vector database platforms have emerged to serve this growing need from AI and ML applications. One example is Pinecone, a fully managed vector database service. Pinecone's system indexes and stores vector embeddings for fast similarity search, and it supports familiar database operations like inserts, updates (CRUD), as well as metadata filtering and horizontal scaling.

To give a clearer picture, let's say you're building an AI-powered semantic search for support tickets. You might use an embedding model to convert each ticket (its text) into a vector, then store those in a vector database with an ID and perhaps metadata like {product: "XYZ", priority: "high"}. When a new customer query comes in (also converted to a vector), the database can quickly return, for example, the top 5 most similar existing tickets (by vector similarity) that match the query – and you could also filter that query to only search within product: "XYZ" tickets using the database's metadata filters.

This kind of use case is where vector database capabilites shine.

As well as Pinecone, there are now many other vector databases available.

- Milvus is an open source vector database known for high performance and the ability to handle billions of vectors (a popular choice for organizations that want to self-host).
- Weaviate is another open source vector database that includes additional features, including a built-in knowledge graph and hybrid search.
- Faiss (Facebook AI Similarity Search) is a library for vector indexing that can be used as a lower-level component (often embedded in other systems) On its own, Faiss is more of a library than a full database service.
- Chroma, Qdrant, and Annoy are other open source database examples, each with different strengths.

Cloud vendors also offer vector search capabilities. Amazon's OpenSearch service (a search engine based on Elasticsearch) now supports vector indices and is recommended as a vector database solution for certain AWS AI services.

- Database Migration
- Databa
- Distributed SQL
- Distribu SQL Su
- Docker
- Ecosys Integra
- Geo-Distribution
- Google Platfor
- Google Spanner
- GraphQ
- How It Works
- How To
- Integrations
- JavaSc
- Jepsen Tests
- Kafka
- Kubernetes
- Micros Azure
- Microsoft Azure Cosmos DB
- Mongo
- On-Premises
- Open S
- Partners
- Perfor Benchr
- Pivotal Container Service (PKS)
- Postgr
- Release Announcements
- Retail
- Securit
- Spring
- Stream Industr
- Ultra Resilience
- Use Ca
- Vector
- Week i Review
- YugabyteDB
- Yugaby Aeon
- YugabyteDB Training
- Yugaby Voyage

vector search natively with a PostgreS-compatible SQL interface. As it is fully PostgreSQL-compatible, YugabyteDB feels immediately familiar. Users can define vector columns, create vector indexes, and query them using standard SQL via the pgvector extension. Underneath this familiar interface lies a highly optimized, distributed vector storage engine, purpose-built for scale-out performance in a globally distributed database. Learn more about YugabyteDB's vector indexing architecture in this recent blog.

Is SQL a vector database?

No – SQL (Structured Query Language) itself is not a vector database. SQL is a language used to query relational databases like MySQL, PostgreSQL, Oracle, or SQL Server. These relational databases store structured data in tables and excel at operations like exact match queries, range queries, and joins across tables of data.

Traditional SQL-based databases (relational databases) were not built to handle the high-dimensional similarity searches that vector databases perform. In a vector database vs regular database comparison, one of the biggest differences is that regular relational databases do not natively support querying by vector similarity. If you ask a SQL database "Find records that are most similar to this target vector," it wouldn't know what to do — at least not without additional help.

However, the landscape is evolving. While a vanilla SQL database won't function as a vector database, some relational systems have begun adding extensions or features to accommodate vector data. A notable example is the PostgreSQL pgvector extension. pgvector introduces a new column type for vectors and related functions, allowing Postgres to store embedding vectors and perform nearest-neighbor searches on them. In essence, pgvector fills a gap because Postgres does not have native vector capabilities in core versions.

With the PostgreSQL pgvector extension, you can create a table that has a column of type VECTOR(768) (for a 768-dimensional vector, for example) and then create an index on that column to enable similarity queries. This means you can use SQL to insert and query vectors (e.g., using <-> operator in Postgres to get distance between vectors), bringing vector search capabilities into the relational world.

In practice, many companies do not need all of the capabilities of a standalone vector database. They need vector search functionality, which can be paired with more traditional SQL databases, offering a fully multi-modal solution for teams building modern applications. YugabyteDB is a fully distributed, AI-ready, multi-modal database that provides this type of flexibility.

What is the difference between a vector database and a regular database?

Differences between a vector database and a "regular" database (typically meaning a relational database) stem from the data model, query types, and use cases they are optimized for. In a vector database vs. relational database comparison, we find that each is suited to different kinds of data and workloads. Below is a comparision of some ey differences:

Data Model and Structure

Regular relational databases store structured data in tables with predefined schemas (rows and columns for each field). They're excellent for clearly defined data like numbers, dates, categories, and relationships (e.g., an e-commerce database with tables for Customers, Orders, Products linked by foreign keys).

A vector database is designed to store unstructured or semi-structured data as vectors – high-dimensional numerical representations (embeddings) of the data.

Query Mechanism

conditions specified. If you want to find similar entries in a relational database, you usually have to craft logic (like OR conditions or external application logic).

Vector databases use similarity queries. The core query in a vector database is something like: "Given this query vector, find the K nearest vectors in the database" using a distance metric (cosine similarity, Euclidean distance, etc.). Results are typically ordered by how similar they are to the query, not by an exact property match.

Indexing and Performance

Regular databases index data using structures like B-trees, hash indexes, etc., optimized for discrete values and range queries on one or a few columns. These work great for primary keys, sorted order retrieval, and so on.

Vector databases employ specialized vector indexes (as discussed earlier) such as HNSW graphs, IVF, or product quantization to handle the complexity of high-dimensional data and large datasets. These indexes make approximate nearest neighbor search dramatically faster.

The trade-off is that results might be approximate, but in practice a well-tuned vector index can retrieve very similar results with a tiny fraction of the computations.

In summary, regular databases are optimized for precise filtering and joining of structured records and vector databases are optimized for computing similarities among high dimensional points. If you try to replicate what a vector database does in a normal database (say, by storing vectors in a table and scanning through computing distances in SQL), it would be painfully slow on large data. The vector database's indexing makes those operations feasible in real-time.

In practice, a vector database and a regular relational database are complementary tools. The vector database versus regular database question is not really about which is universally better, but which is better for a specific task. If your data is highly structured and your queries are well-defined and exact, the traditional database wins. If your data is unstructured (text, images, complex signals) and you need to query by similarity or semantic content, a vector database might be the specialized solution you need.

Many modern architectures combine both — using vectors to handle the AI side (e.g., understanding user queries, content, recommendations) and relational tables to handle the transactional side (e.g., user accounts, logging actions, etc.). By understanding the core differences, you can choose the right tool (or a combination of tools) for your use case.

Why and when should you use a vector database?

The rise of AI applications has quickly moved vector databases from a niche technology to a critical component of the AI/ML stack.

But why are they so useful, and when should you introduce a vector database into your architecture?

Basically, you should consider a vector database if you are dealing with data that benefits from similarity-based retrieval, or when you need to manage unstructured data at scale for ML purposes.

Traditional databases struggle with unstructured data (like free text or multimedia) because they can't natively query the meaning or content of this data. By turning data into embeddings and storing those, vector databases allow applications to search by semantic similarity. This unlocks a variety of powerful capabilities and applications.

Semantic Search and Information Retrieval

efficient retrieval of information.

For example, with a vector database backing your search, a query like "how to reset my account password?" can return a relevant answer even if the document doesn't contain the exact words "reset" or "password," because the system compares the query's embedding to those of documents and finds ones that are conceptually similar. This is used in advanced document search, customer support bots, legal discovery tools, etc., to retrieve information that matches the intent behind a query. It's more robust to synonyms, paraphrasing, or typos than a traditional keyword search.

Recommendation Systems and Personalization

Vector databases are great for finding items similar to a given item or user preference vector. If you've seen "customers also bought..." or "you might like..." on e-commerce and streaming platforms, there's often vector similarity working behind the scenes. For instance, an online store can use vectors to represent each product's properties (possibly learned from user behavior or product descriptions) and learn each user's taste, then recommend new products by finding vectors close to the user's "interest vector".

Multimedia Search (Images, Audio, Video)

With vectors, you can search through images or audio in ways that are impossible with standard databases. For example, you could take a photo of a shoe and find visually similar shoes in a catalog (image-to-image search) by comparing image embeddings. Or find segments of audio that sound similar to a given clip (useful in music or speech processing domains). Vector databases can store embeddings for images and audio generated by deep learning models, enabling content-based retrieval. A traditional database would require manually annotated tags or metadata to do anything comparable, which is often labor-intensive and incomplete.

Retrieval Augmented Generation (RAG) For Natural Language Applications and LLMs (Large Language Models)

If you are building applications with large language models or other AI models, a vector database is essential for providing context and long-term memory. This is a big reason for the recent surge in popularity of vector databases. For example, in a chatbot or assistant, you might have a knowledge base of documents or facts. By embedding all those documents and storing them in a vector database, you can retrieve the documents most relevant to the user's current query in real time (by embedding the query and doing a similarity search).

This technique, known as retrieval-augmented generation (RAG), allows an LLM to return answers with real, up-to-date information instead of relying solely on trained knowledge. It also helps prevent AI "hallucinations" by grounding the responses in actual retrieved data.

If you're augmenting LLMs or doing any kind of question-answering over custom data, a vector database is the go-to solution for storing and retrieving those embeddings of your domain text. (This is what people mean by a vector database for LLM use case.)

What is the ideal use case for a vector database?

You should consider a vector database when your application or project involves unstructured data, AI models, or semantic search/retrieval needs that go beyond the capabilities of keyword search or simple SQL queries. Ideal use cases include:

- When you have a lot of **unstructured data to index** – for example, a large volume of documents, articles, customer support tickets, or research papers that you want to make searchable by meaning. If users need to find relevant text without exact matches, a vector database is ideal.

knowledge base, you'll want to vectorize that knowledge base and use a vector database to fetch relevant pieces as the conversation progresses (to feed into the model).

When implementing **recommendation or personalization features**. The moment you decide "we should suggest similar items to users" or "let's find users with similar behavior for marketing," you are talking about comparing vectors (because you'll likely use an ML model to encode item properties or user behavior into vectors).

When you require **hybrid search – combining traditional filtering with semantic search**. If you want to do queries like "find products similar to X that are under $50 and in stock" or "find documents about energy (not necessarily the keyword, but the concept) written after 2020," this mix of structured constraint and semantic similarity can be elegantly handled by a vector database with metadata filtering.

Consider using a vector database when you need to understand or retrieve data by meaning, similarity, or high-dimensional patterns. Vector features are required for AI-driven features like semantic search, recommendations, and AI assistance, especially at scale. Conversely, stick with traditional databases for transactional processing and strictly structured queries.

Often, the best solution is a combination of both. Keep your structured data and critical transactions in a relational database, and augment your system with a vector database for the intelligent, ML-driven components. Many enterprises are now incorporating vector databases alongside their existing data platforms to unlock insights from unstructured data that were previously out of reach, or choosing newer relational databases that offer the enhanced vector features they require.

Ready to build smarter AI applications?

Vector databases are transforming how AI and machine learning applications handle unstructured data. Whether you're implementing semantic search, powering recommendation engines, or giving your AI models long-term memory through RAG, the right vector database infrastructure is crucial for success.

YugabyteDB combines the power of distributed PostgreSQL with pgvector support, giving you the best of both worlds – enterprise-grade SQL capabilities alongside high-performance vector search. Scale from millions to billions of vectors while maintaining low-latency queries and strong consistency. Your AI applications deserve a database that can grow with them.

Start experimenting with YugabyteDB today for your vector search needs – available as open source for self-hosting or as a fully managed service. See how a distributed SQL approach to vector databases can take your AI and ML applications to the next level.

Find out more about vector search features and capabilities on our dedicated Key Concepts page.

AUGUST 11, 2025

AI, VECTOR, YUGABYTEDB

Yugabyte Team
August 11, 2025

Related Posts

Semantic AI Search with Vector Databases

Implementing Multi-Agent AI with YugabyteDB Vector

3 months ago

Explore YugabyteDB's Vector Indexing Architecture

7 months ago

Explore and YugabyteDB in Depth

Discover the future of data management.

Learn at Yugabyte University

Get Started

Browse Yugabyte Docs

Explore docs

PostgreSQL For Cloud Native World

Product

Company

Solutions

Community

Resources

FOLLOW US

RECEIVE OUR UPDATES

First name*

Last name*

Work Email*

Subscribe

Terms of Service Privacy Policy Cookie Policy Your California Privacy Choices