Vector Database Use Case & Test Document

EverythingBlockchain, Inc
https://builddb.io

Disclaimer

- This document and its associated content are prepared by EverythingBlockchain, Inc (EBI).
- This document and its associated content are protected trade secrets and MAY NOT BE SHARED with anyone outside of the following party(s) specified in this disclaimer without written EBI approval.
- This document and its associated content are prepared exclusively for the benefit of the following party(s): Decentralized Labs, Inc.
- This document and its associated content are provided under a 30-day proof of concept (POC) evaluation of BuildDB as a cloud database alternative using an airline-specific use case.
- Use of this POC grant EverythingBlockchain, Inc the right to use and publish performance results.

Introduction

If your cloud services require storage, then avoid the common technical debt that accompanies the sprawl of multiple databases. The BuildDB™ database stack has evolved into a Software as a Service (SaaS) solution where one database solution can outperform in cost and performance against the following competitors!

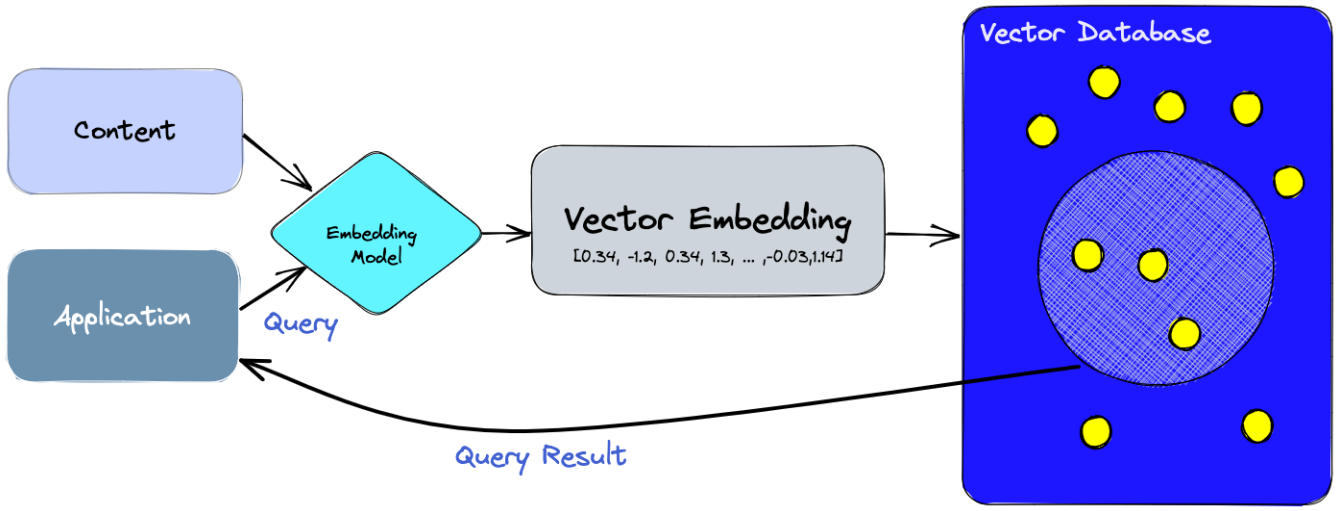
1. Supabase
2. PostgreSQL
3. AWS DynamoDB
4. Azure CosmoDB
5. Google Cloud Datastore
6. Oracle NoSQL Database
7. MongoDB
8. RavenDB
9. Redis
10. Couchbase
11. MongoDB

.

Vector Database Use Cases

Operating within the cloud is the prevailing trend across all industries, including those related to AI. Unfortunately, another undeniable trend is the escalating cost of maintaining AI models in the cloud. One of the challenges of AI is in how to efficiently handle the aggregation and querying of the vast amounts of data embeddings generated from the generated AI models. To solve this problem of dealing with the embeddings, vector databases were developed.

A vector database provides a superior solution for handling vector embeddings by addressing the limitations of standalone vector indices, such as scalability challenges, cumbersome integration processes, and the absence of real-time updates and built-in security measures, ensuring a more effective and streamlined data management experience



1. First, an applications use the embedding model to create vector embeddings for the content we want to index.

2. Then, the vector embedding is inserted into the vector database, with some reference to the original content the embedding was created from.

3. Finally, when the application issues a query, we use the same embedding model to create embeddings for the query and use those embeddings to query the database for similar vector embeddings. As mentioned before, those similar embeddings are associated with the original content that was used to create them.

What's the benefits of a vector database Like BuildDB™?

Standalone vector indices like FAISS (Facebook AI Similarity Search) can significantly improve the search and retrieval of vector embeddings, but they lack capabilities that exist in any database. Vector databases, like BuildDB™ on the other hand, are purpose-built to manage vector embeddings, providing several advantages over using standalone vector indices:

1. **Data management:** Vector databases offer well-known and easy-to-use features for data storage, like inserting, deleting, and updating data. This makes managing and maintaining vector data easier than using a standalone vector index like FAISS, which requires additional work to integrate with a storage solution.

2. **Metadata storage and filtering:** Vector databases can store metadata associated with each vector entry. Users can then query the database using additional metadata filters for fine-grained queries.

3. **Scalability:** Vector databases are designed to scale with growing data volumes and user demands, providing better support for distributed parallel processing. Standalone vector indices may require custom solutions to achieve similar levels of scalability (such as deploying and managing them on Kubernetes clusters or other similar systems).

4. **Real-time updates:** Vector databases often support real-time data updates, allowing for dynamic changes to the data, whereas standalone vector indexes may require a full re-indexing process to incorporate new data, which can be time-consuming and computationally expensive.

5. **Backups and collections:** Vector databases handle the routine operation of backing up all the data stored in the database. Pinecone also allows users to selectively choose specific indexes that can be backed up in the form of "collections," which store the data in that index for later use.

6. **Ecosystem integration:** Vector databases can more easily integrate with other components of a data processing ecosystem, such as ETL pipelines (like Spark), analytics tools (like Tableau and Segment), and visualization platforms (like Grafana) – streamlining the data management workflow. It also enables easy integration with other AI related tools like LangChain, LlamaIndex and ChatGPT's Plugins.

7. **Data security and access control:** Vector databases typically offer built-in data security features and access control mechanisms to protect sensitive information, which may not be available in standalone vector index solutions.

How does a vector database work?

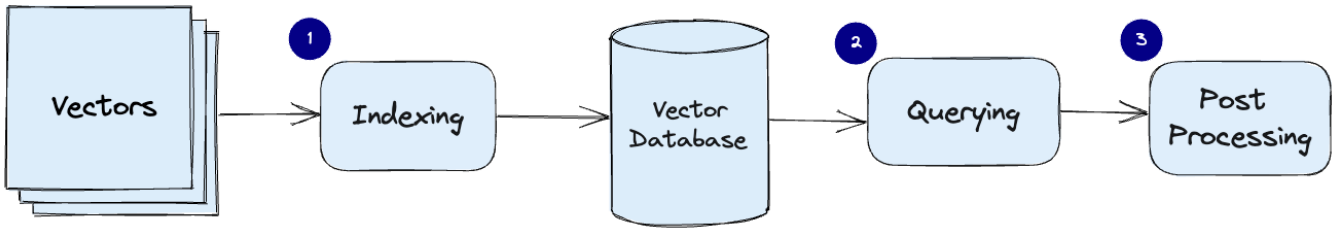
We all know how traditional databases work (more or less)—they store strings, numbers, and other types of scalar data in rows and columns. On the other hand, a vector database operates on vectors, so the way it's optimized and queried is quite different.

In traditional databases, we are usually querying for rows in the database where the value usually exactly matches our query. In vector databases, we apply a similarity metric to find a vector that is the most similar to our query.

A vector database uses a combination of different algorithms that all participate in Approximate Nearest Neighbor (ANN) search. These algorithms optimize the search through hashing, quantization, or graph-based search.

These algorithms are assembled into a pipeline that provides fast and accurate retrieval of the neighbors of a queried vector. Since the vector database provides approximate results, the main trade-offs we consider are between accuracy and speed. The more accurate the result, the slower the query will be. However, a good system can provide ultra-fast search with near-perfect accuracy.

Here's a common pipeline for a vector database:



1. **Indexing**: The vector database indexes vectors using an algorithm such as PQ, LSH, or HNSW. This step maps the vectors to a data structure that will enable faster searching.

2. **Querying**: The vector database compares the indexed query vector to the indexed vectors in the dataset to find the nearest neighbors (applying a similarity metric used by that index)

3. **Post Processing**: In some cases, the vector database retrieves the final nearest neighbors from the dataset and post-processes them to return the final results. This step can include re-ranking the nearest neighbors using a different similarity measure.

What is the BuildDB™ advantage?

While other relational databases have incorporated vector database functionality through plugins and libraries, these are bolt-on features that are severely limited by architecture. BuildDB™ combines its architecture with relational database functionality and vector database features as a first-class citizen without the inherent latency of a RDBMS and its indexes, resulting in the following advantages:

1. Superior speed
2. Superior storage savings
3. Superior scalability
4. Superior features

The key to mining massive amounts of data to find value not humanly possible lay in the ability to leverage AI in the cloud while managing costs efficiently. Achieving this balance depends upon identifying and evaluating and adopting technologies that provide a competitive advantage.

These competitive advantages take the form of:

1. Personalized Medicine

    - **Patient Stratification**: Text embeddings and vector databases can be used to analyze patient data and identify subgroups of patients who are likely to respond similarly to a particular treatment. This is crucial for the development of personalized treatment plans.
    - **Drug Response Prediction**: By analyzing genetic and clinical data, these tools can be used to predict how individual patients are likely to respond to specific drugs, allowing for more personalized and effective treatment

2. Regulatory Compliance and Document Management

    - **Deduplication:** Alleviate high data dimensionality from duplicate data
    - **Document Similarity and Classification**: Pharmaceutical and biotechnology companies deal with vast amounts of regulatory documents. Text embeddings can be used to classify and manage these documents effectively. For example, they can be used to identify documents that are relevant to a particular regulatory submission.

3. Competitive Intelligence

    - **Market and Competitor Analysis**: Text embeddings can be used to analyze vast amounts of data from patents, publications, and clinical trial databases to assess the competitive landscape, helping companies to make informed business decisions.

4. Large Language Model Marketplace

    - **Large Language Model SaaS**: Monetizing large language models through a Software as a Service (SaaS) model open opportunities to capitalize on seemingly worthless data from disparate sources.

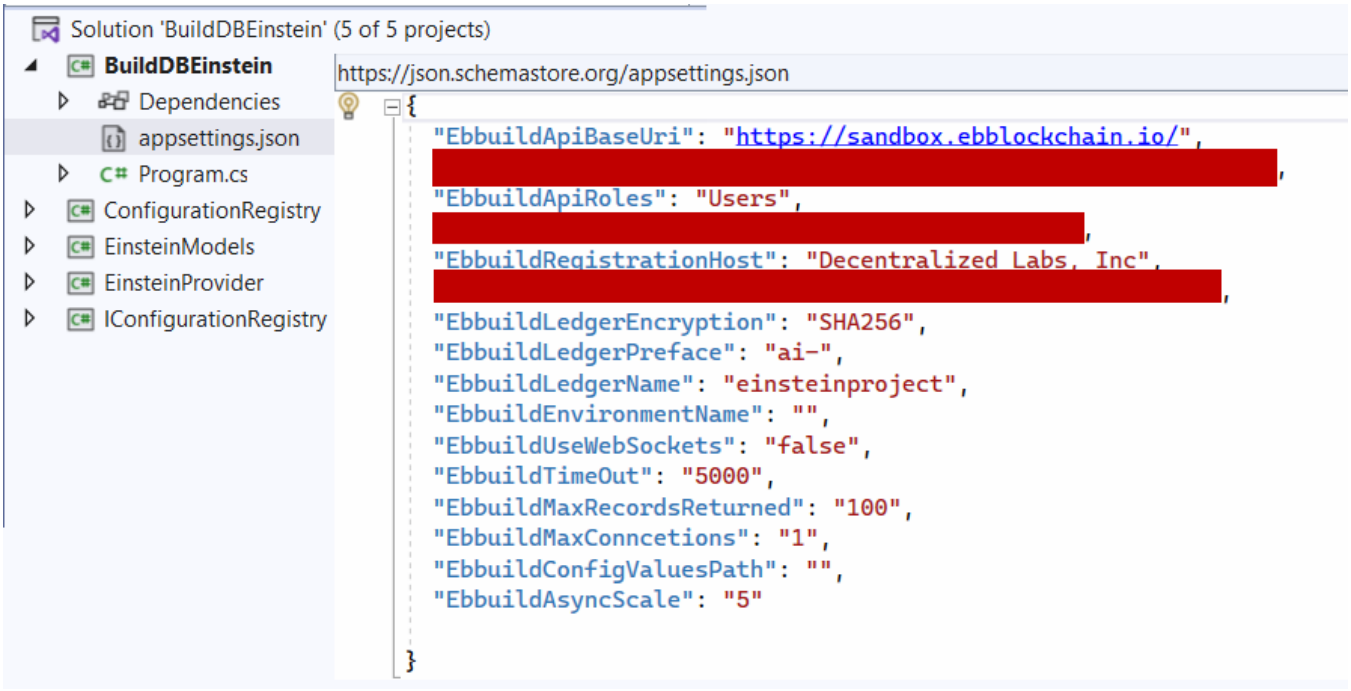
The Demo Project Structure

The associated demo is written in Microsoft C# under .NET Core 7.0 and designed via SOLID design principles. As such this project leverages common interfaces.
The Solution consists of five (5) projects which includes:

1. BuildDBEinstein.csproj - The main entry point project where the testing begins.
2. ConfigurationRegistry.csproj - The supporting project used to simulate .NET ConfigurationRegistry.
3. IConfigurationRegistry.csproj - The interface contract for the ConfigurationRegistry project.
4. EinsteinModels.csproj - The C# classes used to represent the table schemas within the database.
5. EinsteinProvider.csproj - The airline domain-specific logic for flight recent searches and hot markets.

The Demo Project Credentials

The project was built custom for Turkish Airlines, Inc and contains all credentials within the appsettings.json file located under the BuildDBEinstein.csproj.
Everything needed to run the demo is already contained within the projects and ready to run, test and modify as needed.

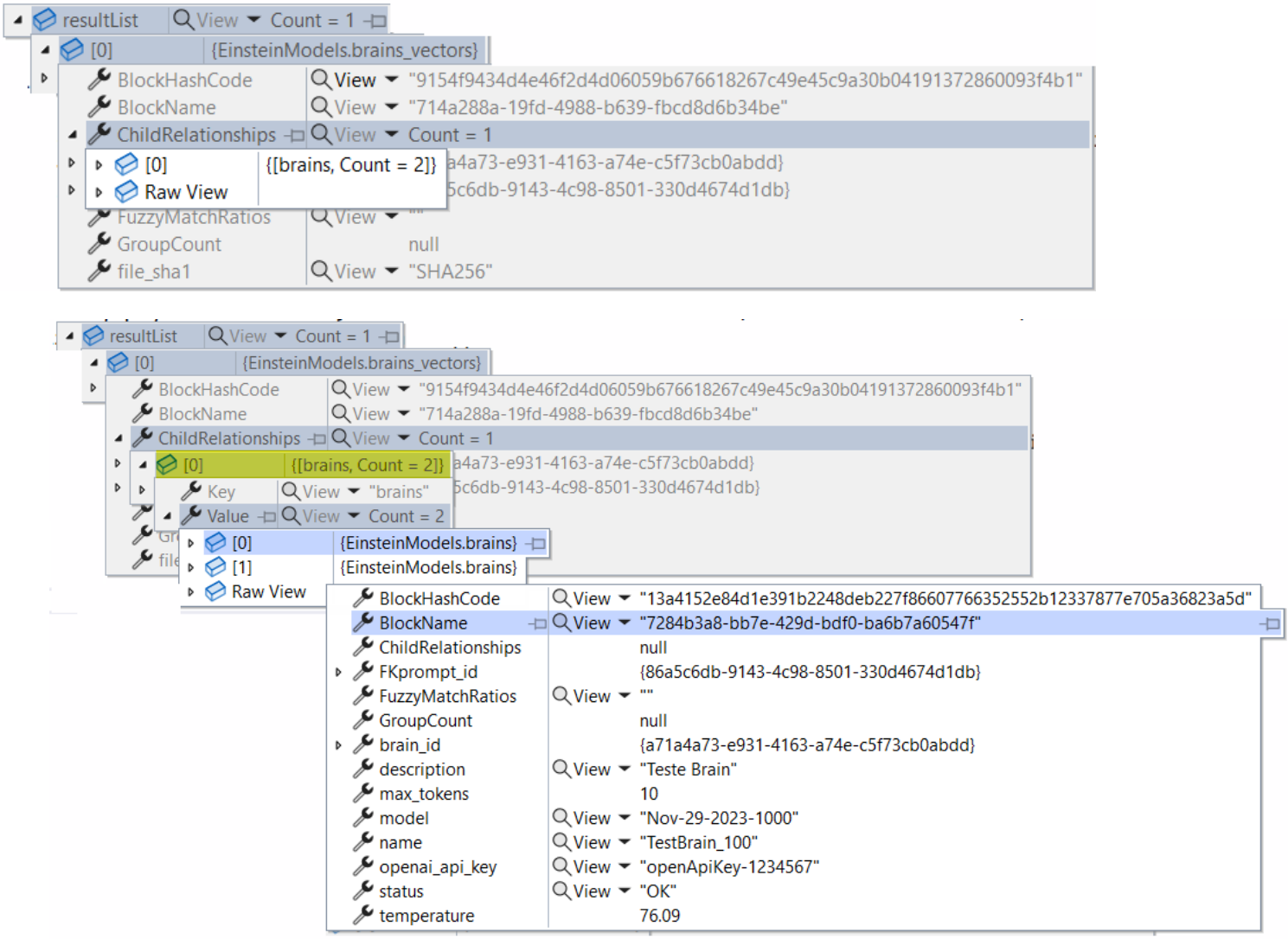


The Demo - BuildDBEinstein.csproj

This demonstration is a port of the Einstein database project to showcase both the requisite CRUD functions common to a RDBMS and the extended vector database functionality needed to store and search vectors and embeddings associated with AI models.

This demonstration will highlight how BuildDB™ provides superior RDBMS search on metadata used to significantly enhance the matching of AI models in a fraction of the time of traditional databases.

Additionally, BuildDB™ simulates user registration and identity management similar to that of an identity server.



The Demo Project Testing Instructions

In-order to demonstrate this use case BuildDB will showcase several key features not available on other document databases. To test the following features simply open and run the Visual Studio Solution BuildDBEinstein.csproj project file.

Once you have opened the BuildDBEinstein.csproj project file you will locate the Program.cs file which lists functions called in-order to highlight the following features.

To see the actual BuildDB code open the EinsteinProvider.csproj project file and open the EinsteinProvider.cs file. This file contains the actual implementation used to call the BuildDB database consisting of CRUD functionality common to most developers.

1. **BuildDB demos high data throughput**
    a. BuildDB can handle complex CRUD functions at a high throughput.

2. **BuildDB demos hierarchical relationships (joins) without the aid of costly table indexes.**
    a. All data stored with BuildDB is distributed and stored without indexes using a patent pending architecture.
    b. BuildDB will conduct hierarchical relationships between parent and child entities using complex joins and filtering without the use of any costly indexes.

3. **BuildDB demos the ability to handle complex scalar types like vectors and embeddings**
    a. BuildDB provides internal classes to handle the formatting of vectors and matrixes with high dimensionality.
    b. BuildDB can conduct advanced vector search against massive amounts of records with large embeddings.

4. **BuildDB can combine both filter searches on metadata and complex scalars like vectors and embeddings**
    a. BuildDB can leverage filter functions on metadata to refine vector database searches on embeddings.

5. **BuildDB demos enforcement of RBAC to restrict access to vector search and filtering of data**
    a. BuildDB can search for user identity records and programmatically utilize metadata to enforce access rules to data.