

**What’s the Main Idea?**

Oracle’s **Database 23c AI** is a type of database designed to handle many types of data and tasks, such as text, graphs, and vectors, all in one place. This is referred to as a **converged database** because it brings multiple data types and functions together, unlike **single-purpose databases** that are only designed for one task (like only handling vectors or text).

**What is AI Vector Search?**

**AI Vector Search** is a technology that allows the database to search based on the **meaning** or **semantics** of data, not just simple keywords or exact matches. This is very useful for **modern AI systems** (like Generative AI) because these systems need to find connections and similarities between pieces of data efficiently, even if they aren’t exact matches.

* **Vectors**: In AI, text or images can be converted into numerical **vectors** (long lists of numbers) that represent their meaning. Searching using vectors allows you to find results based on how similar the meanings of two pieces of data are, rather than just exact words or data points.

**How Does Oracle’s Approach Help?**

1. **Multi-Purpose Database**:
   * Unlike single-purpose databases (that handle only one type of data), Oracle’s **converged database** handles many different kinds of data, including:
     + **JSON** (for structured data)
     + **XML** (for hierarchical data)
     + **Graph** (for networks of data)
     + **Spatial** (for geographic data)
     + **Text** (for full-text search)
     + **Relational** (for tables and relationships)
     + **Vectors** (for AI embeddings and similarity search)
2. **Vector Search**:
   * **Vector embeddings** are numeric representations of data (text, images, etc.) that capture their underlying meaning or features.
   * Oracle’s database has built-in support for storing and generating these **vector embeddings** using the **VECTOR datatype**.
   * This means that you can search for similar items based on their meaning or features, not just exact matches.

**Key Features in the Database:**

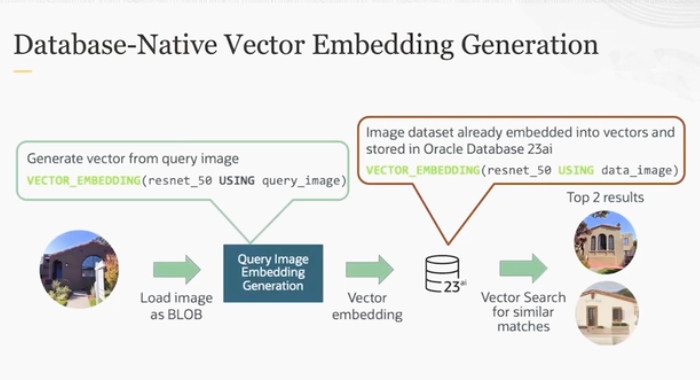
* **SQL Support**: You can use standard **SQL commands** (a language used to interact with databases) to generate vector embeddings and perform vector searches.
* **VECTOR Datatype**: There’s a **VECTOR datatype** specifically designed to store these **vector embeddings** within the database, so you don’t need to store them elsewhere.
* **Similarity Search**: You can use SQL syntax and special functions to perform **similarity searches** easily. This means you can search for items that are most similar to a given input, based on the **vector representation** of the data.
* **High-Performance Search Index**: Oracle’s database includes an **approximate search index**, which helps speed up searches while maintaining high quality and relevance in the results.

**Why is This Important?**

* **Efficient AI Systems**: With the ability to handle various data types and perform fast, meaningful searches, Oracle’s database is well-suited for powering **AI systems** like chatbots, recommendation engines, and semantic search.
* **Handling Complex Data**: By supporting many different data formats (JSON, XML, vectors, etc.), Oracle’s converged database is highly versatile and can handle complex, data-driven tasks more effectively than a single-purpose database.

**Conclusion:**

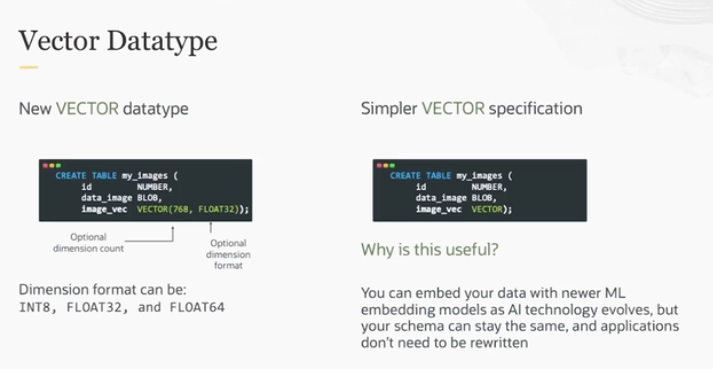
Oracle’s **Database 23c AI** is designed to handle multiple types of data and advanced tasks like **vector-based similarity searches**, making it an ideal tool for modern **AI-driven systems**. It supports high-performance searches, stores vector embeddings natively, and allows easy integration with standard SQL commands.



The database has SQL support for database-native vector generation, VECTOR datatype for storing vector embeddings, SQL syntax and function to perform similarity search with ease, and approximate search index packaged and tuned for high performance and quality.

Oracle Database 23ai allows you to use models with API calls or load an ONNX model into the database. In this example, resnet\_50 has been loaded into the database. Then you can pass the model you want to use into the VECTOR\_EMBEDDING function along with the query image.

The function returns a vector that you can pass into your database. The database has already stored other embeddings of other images. And using the same VECTOR\_EMBEDDING function and model, AI Vector Search can perform similarity searches to return the top matches of the images that look like the query image.



The **VECTOR datatype** in databases allows you to create a table that can store **vector embeddings** alongside traditional relational data. Let’s break it down step by step:

**What is the VECTOR Datatype?**

* The **VECTOR datatype** is a special format used to store **vector embeddings** in a database table.
* **Vector embeddings** are numerical representations of data, like text or images, that capture their meaning or features in a form that machines (AI models) can easily process.

**How It Works in Tables:**

* You can now create a table in a database that includes columns for vectors, just like you would for traditional data (like numbers or text).
* For example, in a single table, you can store both **relational data** (like names, IDs, etc.) and **vectors** (like AI-generated embeddings of text or images) in the same place.

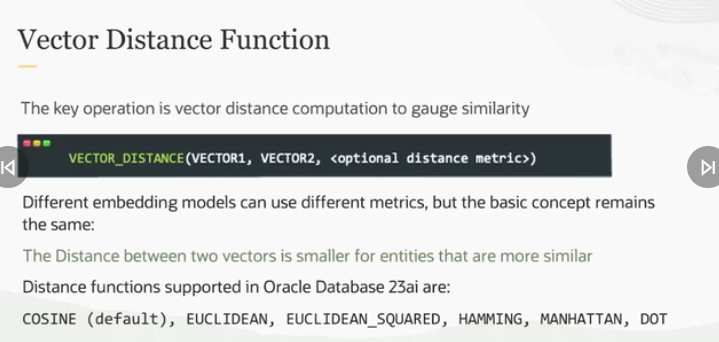
**Flexibility of the VECTOR Datatype:**

* **With Dimension Format**: You can specify the **dimension** (size) of the vector in the table, meaning how many numbers the vector will contain (e.g., a vector with 128 dimensions).
* **Without Dimension Format**: You can also choose a simpler version of the **VECTOR specification** without explicitly mentioning the dimension, making it more flexible.

**Why Is This Useful?**

1. **Future-Proofing Your Database**:
   * As **machine learning models** evolve and newer models generate different types of **embeddings**, your database schema (the way your tables are structured) doesn’t need to change.
   * For example, if you're using one type of vector embedding today (say 128 dimensions), but a newer model with 256-dimensional embeddings becomes available tomorrow, you don’t have to redesign your entire table. The VECTOR datatype can adapt to this easily.
2. **Combining Relational and Vector Data**:

* You can store traditional relational data (e.g., names, IDs, prices) alongside vector embeddings. This is important when you want to combine structured data with AI-powered features like **semantic search** or **recommendation systems**.



**What is a Similarity Search?**

In a **similarity search**, we try to find items (e.g., images, texts, or data points) that are **most similar** to a given input. This is useful in many AI applications, such as:

* Finding similar products in an online store.
* Searching for similar sentences or documents.
* Recommending similar movies or songs.

**VECTOR\_DISTANCE Function:**

* The **VECTOR\_DISTANCE function** is used to measure how close or far apart two vectors are. The closer two vectors are, the more **similar** they are to each other.
* Vectors represent items (like text or images) as **numerical embeddings**, and the **distance** between these vectors tells us how similar these items are.

**How Distance Works:**

* **Smaller Distance = More Similar**: If the distance between two vectors is **small**, it means the items they represent are very similar.
* **Larger Distance = Less Similar**: If the distance is **large**, the items are less similar.

**Distance Metrics:**

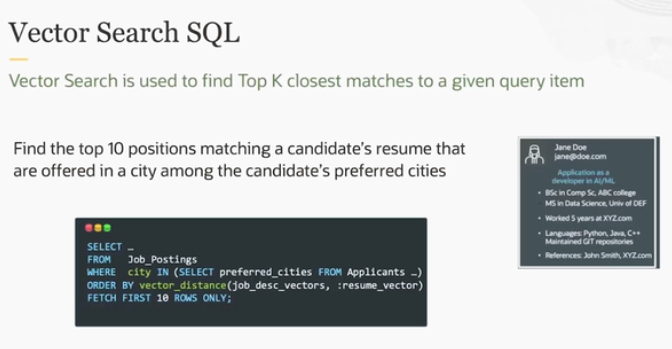
* **Cosine Distance (Default)**: The **default metric** is typically **cosine distance**, which measures the angle between two vectors. Cosine distance is often used when we want to know how similar the **direction** of two vectors is, regardless of their magnitude (size).
* **Other Distance Metrics**: You can choose different distance metrics depending on what your machine learning model uses. For example, some models might perform better with **Euclidean distance** (which measures the straight-line distance between points) or **Manhattan distance** (which measures distance in grid-like movements).

**Why Choosing the Right Distance Metric Matters:**

* The choice of **distance metric** depends on how your **vector embeddings** were generated. Some models work better with cosine distance, while others might work better with Euclidean distance. Choosing the right distance metric ensures your similarity search is accurate and gives you the best results.

**Summary:**

* The **VECTOR\_DISTANCE function** helps calculate how similar two vectors are based on their distance.
* **Smaller distances** indicate more similar items, while **larger distances** indicate less similarity.
* The **distance metric** (like cosine distance or Euclidean distance) should be chosen based on how the vector embeddings were created, to ensure you get the most accurate similarity matches.

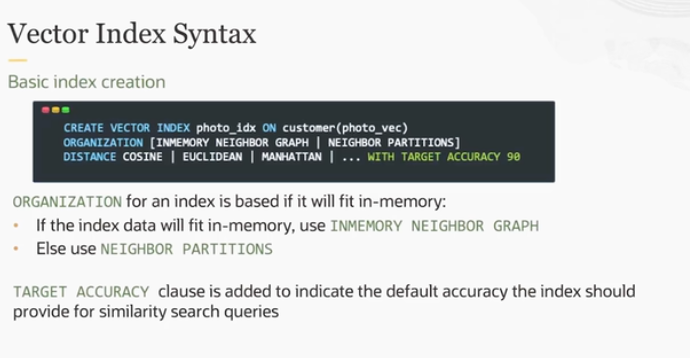


**Vector Search in SQL:**

* **Vector Search** is a way to search for **similar items** by comparing the **vector embeddings** (numerical representations) of data.
* In this case, it's used to find **job postings** that best match a **candidate's resume** based on **vector similarity**.

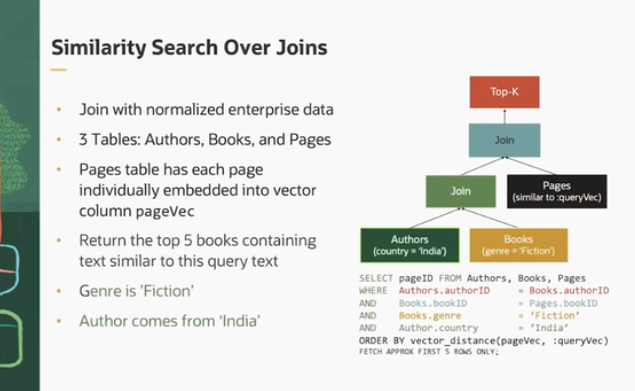
**What’s Happening in the Example:**

1. **Goal**:
   * The goal is to find the **top 10 job positions** that match a candidate’s resume and are offered in the **candidate’s preferred cities**.
2. **SQL Query**:
   * The SQL query retrieves job postings based on **vector distance** between the **job description vectors** (representing the job details) and the **resume vector** (representing the candidate's resume).
   * This means the system is comparing how similar the job descriptions are to the candidate’s resume using **vector embeddings**.
   * The ORDER BY vector\_distance function orders the results based on how **close** the vectors are (the smaller the distance, the more similar the job is to the resume).
   * **Fetch first 10 rows**: This limits the result to the **top 10 matches**, ensuring that the best matches are shown first.
3. **How It’s Used**:
   * In this case, **applicant data** (like preferred cities) is combined with **job data** (job descriptions), and then AI **Vector Search** is applied to find the best matches.
   * The whole process is done with just **five lines of SQL**, which makes it simple and efficient to use. It integrates multiple pieces of data—**resume, job descriptions, and location preferences**—into a single search.



A screenshot of a search engine

Description automatically generated



**What is Similarity Search Over Joins?**

This example shows how **vector-based similarity searches** can be combined with traditional SQL **joins**. In a real-world **enterprise** environment, data is often **normalized**, meaning it’s split across multiple tables (like **Authors**, **Books**, and **Pages**) to make it more efficient to store and manage.

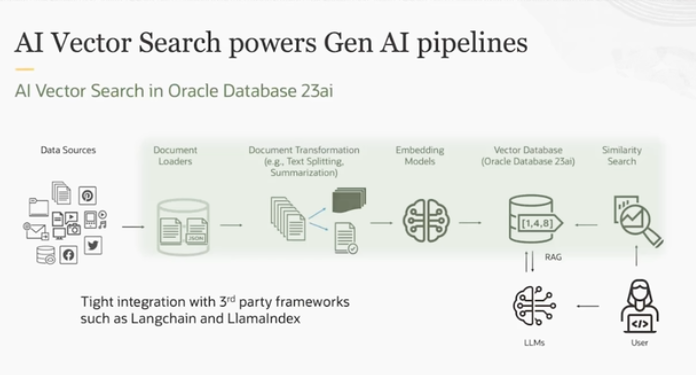
Oracle's ability to perform **similarity searches over joins** makes it possible to search across **multiple tables** for the most relevant matches based on vector embeddings, which isn’t something many databases offer easily.

**Breaking Down the Example:**

1. **Three Tables**:
   * **Authors**: Contains information about the authors (e.g., country).
   * **Books**: Contains details about the books (e.g., genre).
   * **Pages**: Contains text from each page of the books, and each page’s text is represented as a **vector embedding** (stored in the pageVec column).
2. **Goal**:  
   The aim is to find the **top 5 books** that:
   * Contain pages with text similar to a **query text** (stored as **qVec**).
   * Have the genre **“Fiction.”**
   * Are written by authors from **India**.
3. **How It Works**:
   * The SQL query **joins** the three tables: **Authors**, **Books**, and **Pages**.
   * **Similarity Search**: The core part of the query is based on the **vector\_distance** function, which compares the **vector embeddings** of the pages (pageVec) with the vector representation of the **query text** (qVec).
   * The query ranks the pages based on their similarity to the query using the **vector distance**, and it orders the results to show the closest (most similar) matches.
4. **Additional Filters**:
   * Only books with the **Fiction** genre are considered.
   * The authors of the books must be from **India**.

**Key Features:**

* **Vector Embeddings**: Each page's content is converted into a vector, allowing you to search by **semantic meaning**, not just by keywords.
* **Joins**: This query uses SQL joins to combine multiple tables (Authors, Books, Pages), allowing for a richer, more complex query.
* **Vector Search**: The **vector\_distance** function is used to compare the vectors of the pages with the query, helping find the most similar content.
* **Top K Results**: It limits the results to the **top 5** most relevant books.



The image and explanation describe how **AI Vector Search** powers a **Generative AI (Gen AI) pipeline** using **Oracle Database 23c AI**. Let’s break this down in simple terms:

**Steps in the Gen AI Pipeline:**

1. **Data Sources**:
   * The process starts with **data sources**, which can come from anywhere. It could be:
     + A database
     + CSV files
     + Social media posts
     + Websites, etc.
   * Essentially, any type of **structured** or **unstructured data** can be used in this pipeline.
2. **Document Loaders**:
   * The next step is using **document loaders** to bring that data into the system. This step allows the data to be loaded into the database so it can be processed.
3. **Document Transformation**:
   * Once the documents are loaded, you can **transform** them. Some common transformations include:
     + **Summarizing** the documents
     + **Splitting** them into smaller chunks
     + **Generating vector embeddings** for the content.
   * This step prepares the data for AI processing by turning it into a format that the AI model can understand.
4. **Embedding Models**:
   * An **embedding model** is applied to the data to generate **vector representations** (vectors are a way to numerically represent text or other data).
   * These vectors capture the meaning and relationships of the text in a way that the AI system can understand.
5. **Vector Database**:
   * The vectors are stored in the **vector database** (in this case, Oracle Database 23c AI). This specialized database stores the vector embeddings, making them easily searchable for similarity searches or other AI tasks.
6. **Similarity Search**:
   * Users can perform **similarity searches** on the vector database to find data that is **semantically similar** to a given query. For example, if a user searches for a topic or a document, the system can find other documents or data that have a similar meaning, not just exact keyword matches.
7. **RAG (Retrieval-Augmented Generation) Pattern**:
   * The pipeline can also use a **RAG design pattern**, which means that the **Large Language Model (LLM)** can pull relevant information from the vector database in real-time during the generation process. This makes the output of the LLM more accurate and relevant by leveraging real data stored in the database.
8. **Integration with Third-Party Frameworks**:
   * Oracle Database 23c AI supports tight integration with **third-party frameworks** like **LangChain** and **LlamaIndex**. These are powerful tools that help developers build more complex and customized AI applications by providing pre-built components and workflows for interacting with AI models and data.

**Why is This Important?**

* **AI Vector Search** allows for more advanced data retrieval because it searches based on meaning (semantics) rather than simple keyword matching.
* This pipeline supports **complex Generative AI workflows**—from transforming raw data to running searches or interacting with AI models like **LLMs**.
* The **integration with frameworks** like LangChain and LlamaIndex makes it easier for developers to create AI-powered applications.