**Documentation for Hybrid Search API Development**

**Main Task**

The task is to develop a Hybrid Search API that performs searches across a table of 1 million records. The search functionality should combine traditional keyword-based searches with vector-based searches. The data model involves two tables: one for magazine information and another for magazine content.

**Requirements**

**API Endpoint**

* **Create a single endpoint** that performs a hybrid search.
* **Combine traditional keyword-based search** with vector-based search.

**Data Model**

* **Table 1: Magazine Information**
  + **Fields:** id, title, author, publication\_date, category, etc.
* **Table 2: Magazine Content**
  + **Fields:** id, magazine\_id (foreign key to Magazine Information), content, vector\_representation, etc.

**Search Functionality**

* **Keyword Search:** Search based on keywords in the title, author, and content.
* **Vector Search:** Search based on vector similarity in the vector\_representation field.
* **Hybrid Search:** Combine results from both keyword and vector searches to return the most relevant results.

**Performance Considerations**

* **Optimize the search** to handle 1 million records efficiently.
* **Consider indexing strategies** and database optimizations.

**Technology Stack**

* **Backend Technology:** Node.js v18 or greater (Express)
* **Database:** Use a suitable database (e.g., PostgreSQL, Elasticsearch, ChromaDB, etc.) that supports vector searches.

**Issues Encountered**

During the development of the API, several issues related to inserting vector data into the database were encountered:

1. **Vector Dimension Mismatch:**
   * The database schema expected vectors with 300 dimensions. However, insertion commands contained vectors with fewer dimensions, leading to errors.
2. **SQL Syntax and Transaction Management:**
   * Errors such as "current transaction is aborted" occurred, indicating that a failure in a previous SQL command affected subsequent commands in the same transaction.
3. **Database Schema Constraints:**
   * Ensuring that the vector\_representation column was correctly defined to handle 300-dimensional vectors was critical. Mismatched schemas led to insertion errors.

**Example of SQL Command and Error Handling**

**Corrected SQL Command**

sql

BEGIN;

-- Insert data with the correct vector dimensions

INSERT INTO magazine\_content (magazine\_id, content, vector\_representation) VALUES

(1, 'This is some sample content related to Science.', ARRAY[

0.1, 0.1, 0.1, ..., 0.1 -- 300 values in total

]::vector);

COMMIT;

**Error Handling**

* **Ensure that the vector** being inserted contains exactly 300 values.
* **Review SQL commands** for syntax errors and proper transaction management.
* **Confirm that the database schema** supports vectors with 300 dimensions.
* **Handle transaction failures** by rolling back transactions and resolving root causes before retrying.

**Setup and Running the API**

1. **Setup Instructions:**
   * **Clone the Repository:**

bash

git clone <repository-url>

cd <repository-directory>

* + **Install Dependencies:**

bash

npm install

* + **Configure the Database:**
    - Edit the configuration file (e.g., config.js) to include your database connection details.
  + **Run Migrations:**

bash

npm run migrate

* + **Start the API Server:**

bash

npm start

1. **Accessing the API:**
   * **Endpoint URL:** http://localhost:5000/search
   * **Method:** POST
   * **Request Body Example:**

json

{

"query": "Science",

"vector": [0.1, 0.1, ..., 0.1] // 300-dimensional vector

}

**Examples of Search Queries and Expected Results**

1. **Keyword-Based Search Query:**
   * **Request Body:**

json

{

"query": "Science"

}

* + **Expected Results:**

json

[

{

"id": 1,

"title": "Exploring the Universe",

"author": "Jane Doe",

"publication\_date": "2023-01-15",

"category": "Science",

"content": "This article explores various aspects of the universe...",

"vector\_representation": [0.1, 0.1, ..., 0.1]

}

]

1. **Vector-Based Search Query:**
   * **Request Body:**

json

{

"vector": [0.1, 0.1, ..., 0.1] // 300-dimensional vector

}

* + **Expected Results:**

json

[

{

"id": 1,

"title": "Exploring the Universe",

"author": "Jane Doe",

"publication\_date": "2023-01-15",

"category": "Science",

"content": "This article explores various aspects of the universe...",

"vector\_representation": [0.1, 0.1, ..., 0.1]

}

]

1. **Hybrid Search Query:**
   * **Request Body:**

json

{

"query": "Science",

"vector": [0.1, 0.1, ..., 0.1] // 300-dimensional vector

}

* + **Expected Results:**

json

[

{

"id": 1,

"title": "Exploring the Universe",

"author": "Jane Doe",

"publication\_date": "2023-01-15",

"category": "Science",

"content": "This article explores various aspects of the universe...",

"vector\_representation": [0.1, 0.1, ..., 0.1]

}

]

**Deliverables**

1. **Source Code:**
   * Complete source code of the API implementing hybrid search functionality.
2. **Database Schema:**
   * SQL scripts or ORM models for creating the database tables (Magazine Information and Magazine Content).
3. **Documentation:**
   * Detailed documentation on how to set up and run the API.
   * Examples of search queries and expected results.
4. **Performance Report:**
   * Brief report on performance considerations and optimizations implemented to handle large datasets efficiently.

**Timelines**

* **Completion Deadline:** 7 days from the start of the task.

**Evaluation Criteria**

1. **Functionality:**
   * The API should meet all the specified requirements for hybrid search.
2. **Code Quality:**
   * The code should be clean, readable, and well-documented.
3. **Performance:**
   * The implementation should handle large datasets efficiently.
4. **Innovation:**
   * The solution should demonstrate creative approaches to hybrid search.
5. **Documentation:**
   * The documentation should be clear, comprehensive, and include setup instructions and usage examples.