Complete VectorStore Setup Guide

Step 1: Create PostgreSQL Database and User

1.1 Connect to PostgreSQL as superuser

bash

```bash
sudo -u postgres psql
```

1.2 Create database and user (run these SQL commands in PostgreSQL)

sql

```sql
-- Create a new database
CREATE DATABASE vector_db;

-- Create a new user
CREATE USER vector_user WITH PASSWORD 'your_secure_password';

-- Grant privileges to the user
GRANT ALL PRIVILEGES ON DATABASE vector_db TO vector_user;

-- Connect to the new database
\c vector_db

-- Create required extensions
CREATE EXTENSION IF NOT EXISTS vector;
CREATE EXTENSION IF NOT EXISTS timescaledb;

-- Grant usage on schema to the user
GRANT USAGE ON SCHEMA public TO vector_user;
GRANT CREATE ON SCHEMA public TO vector_user;
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO vector_user;
GRANT ALL PRIVILEGES ON ALL SEQUENCES IN SCHEMA public TO vector_user;

-- Exit PostgreSQL
\q
```

Step 2: Test Database Connection

2.1 Test the connection with your new credentials

bash

```bash
psql -h localhost -p 5432 -U vector_user -d vector_db
```

2.2 Verify extensions are installed

```sql
SELECT extname, extversion FROM pg_extension WHERE extname IN ('vector', 'timescaledb');
\q
```

Step 3: Set Environment Variables

```bash
# Set your Google API key
export GOOGLE_API_KEY="your_actual_gemini_api_key_here"

# Optional: Set database URL as environment variable
export DATABASE_URL="postgresql://vector_user:your_secure_password@localhost:5432/vector_db"
```

Step 4: Install Required Python Packages

```bash
pip install timescale-vector google-generativeai python-dotenv psycopg2-binary
```

Step 5: Fixed Python Code

Save this as `vector_store_fixed.py`:

python

```python
import google.generativeai as genai
from typing import List, Dict, Any
import logging
import time
import os
from datetime import timedelta
import psycopg2
from urllib.parse import urlparse

# Try to import the vector database client
try:
    from timescale_vector import client
    VECTOR_CLIENT_AVAILABLE = True
    print("✅ Timescale Vector client imported successfully")
except ImportError as e:
    VECTOR_CLIENT_AVAILABLE = False
    print(f"❌ Timescale Vector not available: {e}")
    print("Install with: pip install timescale-vector")


def get_settings():
    """Get database settings from environment or use defaults"""

    # Get from environment variable or use default
    database_url = os.getenv("DATABASE_URL", "postgresql://vector_user:your_secure_password@loc

    return {
        "database": {
            "service_url": database_url
        },
        "vector_store": {
            "table_name": "vector_embeddings",
            "embedding_dimensions": 768,
            "time_partition_interval": timedelta(days=7)  # Use timedelta object instead of str
        }
    }


def test_database_connection(database_url: str) -> bool:
    """Test if we can connect to the database"""
    try:
        # Parse the database URL
        parsed = urlparse(database_url)

        # Connect to the database
        conn = psycopg2.connect(
            host=parsed.hostname,
            port=parsed.port or 5432,
            user=parsed.username,
            password=parsed.password,
            database=parsed.path[1:]  # Remove leading slash
        )

        # Test the connection
        cursor = conn.cursor()
        cursor.execute("SELECT version();")
        version = cursor.fetchone()
        print(f"✅ Database connection successful: {version[0][:50]}...")

        # Check if required extensions exist
        cursor.execute("SELECT extname FROM pg_extension WHERE extname IN ('vector', 'timescale
        extensions = cursor.fetchall()
        ext_names = [ext[0] for ext in extensions]

        if 'vector' in ext_names:
            print("✅ Vector extension is installed")
        else:
            print("❌ Vector extension is NOT installed")

        if 'timescaledb' in ext_names:
            print("✅ TimescaleDB extension is installed")
        else:
            print("❌ TimescaleDB extension is NOT installed")

        cursor.close()
        conn.close()
        return True

    except Exception as e:
        print(f"❌ Database connection failed: {e}")
        return False


class VectorStore:
    """A class for managing vector operations and database interactions."""

    def __init__(self):
        """Initialize settings and Gemini embedding client."""
        self.settings = get_settings()
        self.vector_settings = self.settings["vector_store"]

        # Gemini embedding model
        self.embedding_model = "models/embedding-001"
        # Set Gemini API key
        api_key = os.getenv("GOOGLE_API_KEY")
        if not api_key:
            raise ValueError("GOOGLE_API_KEY environment variable is not set")

        genai.configure(api_key=api_key)
        print("✅ Gemini API configured")

        # Test database connection first
        db_url = self.settings["database"]["service_url"]
        if not test_database_connection(db_url):
            print("❌ Database connection test failed")
            self.vec_client = None
            return

        # Initialize vector database client
        if VECTOR_CLIENT_AVAILABLE:
            try:
                print(f"🔄 Initializing vector client...")
                print(f"🔄 Database URL: {db_url}")
                print(f"🔄 Table name: {self.vector_settings['table_name']}")
                print(f"🔄 Embedding dimensions: {self.vector_settings['embedding_dimensions']
                print(f"🔄 Time partition: {self.vector_settings['time_partition_interval']}")

                self.vec_client = client.Sync(
                    service_url=db_url,
                    table_name=self.vector_settings["table_name"],
                    num_dimensions=self.vector_settings["embedding_dimensions"],
                    time_partition_interval=self.vector_settings["time_partition_interval"],
                )
                print("✅ Vector database client initialized successfully")

            except Exception as e:
                print(f"❌ Failed to initialize vector database client: {e}")
                print(f"❌ Error type: {type(e).__name__}")
                print(f"❌ Error details: {str(e)}")
                self.vec_client = None
        else:
            self.vec_client = None
            print("⚠️  Vector database client not available.")

    def get_embedding(self, text: str) -> List[float]:
        """Generate embedding for the given text using Gemini."""
        text = text.replace("\n", " ").strip()
        if not text:
            return []
        start_time = time.time()

        try:
            response = genai.embed_content(
                model=self.embedding_model,
                content=text,
                task_type="retrieval_document"
            )
            embedding = response["embedding"]

            elapsed_time = time.time() - start_time
            print(f"✅ Embedding generated in {elapsed_time:.3f} seconds")

            return embedding
        except Exception as e:
            print(f"❌ Failed to get embedding: {e}")
            return []

    def create_table(self) -> bool:
        """Create the vector table"""
        if not self.vec_client:
            print("❌ Vector database client not available")
            return False

        try:
            print("🔄 Creating vector table...")
            self.vec_client.create_tables()
            print("✅ Vector table created successfully")
            return True
        except Exception as e:
            print(f"❌ Failed to create table: {e}")
            return False

    def insert_vector(self, text: str, metadata: Dict[str, Any] = None, doc_id: str = None) ->
        """Insert a vector into the database."""
        if not self.vec_client:
            print("❌ Vector database client not available")
            return False

        print(f"🔄 Processing text: '{text[:50]}...'")
        embedding = self.get_embedding(text)
        if not embedding:
            print("❌ Failed to generate embedding")
            return False

        try:
```

```python
            # Prepare the record
            record_id = doc_id or f"doc_{int(time.time() * 1000)}"

            # Create record in the format expected by timescale-vector
            record = {
                "id": record_id,
                "metadata": metadata or {},
                "contents": text,
                "embedding": embedding
            }

            print(f"🔄 Inserting record with ID: {record_id}")

            # Insert into the vector database
            self.vec_client.upsert([record])
            print(f"✅ Successfully inserted vector")
            return True

        except Exception as e:
            print(f"❌ Failed to insert vector: {e}")
            print(f"❌ Error type: {type(e).__name__}")
            return False

    def search_similar(self, query_text: str, limit: int = 5) -> List[Dict]:
        """Search for similar vectors in the database."""
        if not self.vec_client:
            print("❌ Vector database client not available")
            return []

        print(f"🔄 Searching for: '{query_text}'")
        query_embedding = self.get_embedding(query_text)
        if not query_embedding:
            print("❌ Failed to generate query embedding")
            return []

        try:
            results = self.vec_client.search(
                query_embedding,
                limit=limit
            )
            print(f"✅ Found {len(results)} similar documents")
            return results
        except Exception as e:
            print(f"❌ Search failed: {e}")
            return []

    def test_functionality(self):
        """Test the complete functionality"""
        print("\n🔬 Testing VectorStore functionality...")

        # Test 1: Embedding generation
        test_text = "Machine learning is transforming how we process data."
        print(f"\n1. Testing embedding generation...")
        embedding = self.get_embedding(test_text)

        if not embedding:
            print("❌ Embedding test failed")
            return

        print(f"✅ Generated {len(embedding)} dimensional embedding")

        # Test 2: Database operations
        if not self.vec_client:
            print("\n❌ Skipping database tests - client not available")
            return

        print(f"\n2. Creating table...")
        if not self.create_table():
            print("❌ Table creation failed")
            return

        print(f"\n3. Testing vector insertion...")
        success = self.insert_vector(
            text=test_text,
            metadata={"source": "test", "category": "ml"},
            doc_id="test_doc_1"
        )

        if not success:
            print("❌ Vector insertion failed")
            return

        print(f"\n4. Testing similarity search...")
        results = self.search_similar("data processing", limit=3)

        if results:
            print("✅ All tests passed!")
            for i, result in enumerate(results[:2]):
                print(f"   Match {i+1}: {result}")
        else:
            print("⚠️  No search results found")


def main():
    """Main function to test the VectorStore"""
    print("🚀 Starting VectorStore setup and testing...")

    try:
        # Check environment variables
        if not os.getenv("GOOGLE_API_KEY"):
            print("❌ GOOGLE_API_KEY environment variable not set")
            print("Set it with: export GOOGLE_API_KEY='your_api_key'")
            return

        # Initialize and test VectorStore
        vector_store = VectorStore()

        if vector_store.vec_client:
            vector_store.test_functionality()
        else:
            print("❌ VectorStore initialization failed")

    except Exception as e:
        print(f"❌ Error in main: {e}")


if __name__ == "__main__":
    main()
```

Step 6: Usage Instructions

1. **Update database credentials** in the code or set environment variable:

   bash

   ```bash
   export DATABASE_URL="postgresql://vector_user:your_secure_password@localhost:5432/vector_db
   ```

2. **Set your Google API key**:

   bash

   ```bash
   export GOOGLE_API_KEY="your_actual_gemini_api_key"
   ```

3. **Run the script**:

   bash

   ```bash
   python vector_store_fixed.py
   ```

Common Issues and Solutions

Issue 1: "relation does not exist"

Solution: Run the table creation step in the code

Issue 2: "permission denied"

Solution: Check user privileges in PostgreSQL

Issue 3: "connection refused"

Solution: Ensure PostgreSQL is running:

```bash
sudo systemctl status postgresql
sudo systemctl start postgresql
```

Issue 4: Wrong database URL format

Solution: Use exact format:

```
postgresql://username:password@hostname:port/database_name
```

This complete setup should resolve all the issues you're experiencing!