For the CGD project, we need a database that is:

1. **Easy to query using Python or SQL.**
2. **Scalable (cloud-based preferred, but local is acceptable for initial stages).**
3. **Supports structured data (since we're dealing with sentences, word pairs, and similarity relationships).**

Given these requirements, PostgreSQL (a SQL database) is a strong choice because:

- It is highly reliable and supports complex queries.
- It has excellent Python support (via libraries like psycopg2 or SQL Alchemy).
- It can be deployed locally or on AWS (via Amazon RDS).

Database Schema Design:

Proposed Schema:

1. **Table: sentences**
    - id (Primary Key)
    - contestant ID (ID of the contestant who spoke the sentence)
    - sentence text (Array of words, stored as JSON or a dedicated table for words)
    - timestamp (When the sentence was recorded)
2. **Table: similar_pairs**
    - id (Primary Key)
    - word1 (First word in the similar pair)
    - word2 (Second word in the similar pair)
3. **Table: similarity_results (Optional)**
    - id (Primary Key)
    - sentence1_id (Foreign Key to sentences)
    - sentence2_id (Foreign Key to sentences)
    - is_similar (Boolean, result of the comparison)
    - timestamp (When the comparison was made)

Why This Schema?

- Normalized Structure: Separates sentences, word pairs, and results for clarity and scalability.
- Flexibility: Can easily extend to store additional metadata (e.g., contestant details).
- Efficient Queries: Indexes can be added to word1 and word2 for fast similarity checks.

Setting up the database

DATABASE DESIGN:

```
CREATE TABLE sentences (
  id SERIAL PRIMARY KEY,
  contestant_id INTEGER,
  sentence_text JSONB,
  timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE similar_pairs (
  id SERIAL PRIMARY KEY,
  word1 TEXT,
  word2 TEXT
);

CREATE TABLE similarity_results (
  id SERIAL PRIMARY KEY,
  sentence1_id INTEGER REFERENCES sentences(id),
  sentence2_id INTEGER REFERENCES sentences(id),
  is_similar BOOLEAN,
```

```
  timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

INSERTING SAMPLE DATA:

```
-- Insert sentences
INSERT INTO sentences (contestant_id, sentence_text) VALUES
(1, '["I", "love", "Python"]'),
(2, '["You", "hate", "Java"]');

-- Insert similar pairs
INSERT INTO similar_pairs (word1, word2) VALUES
('love', 'adore'),
('hate', 'dislike'),
('Python', 'Java');
```

QUERY SIMILARITY PYTHON EXAMPLE

```
import psycopg2
from psycopg2.extras import Json

def is_similar(sentence1, sentence2, similar_pairs):
  if len(sentence1) != len(sentence2):
    return False

  similar_set = {(x, y) for x, y in similar_pairs} | {(y, x) for x, y in similar_pairs}

  for word1, word2 in zip(sentence1, sentence2):
    if word1 == word2:
      continue
```

```
        if (word1, word2) not in similar_set:
            return False
    return True

# Connect to PostgreSQL
conn = psycopg2.connect(
    dbname="your_db",
    user="your_user",
    password="your_password",
    host="localhost"
)
cur = conn.cursor()

# Fetch sentences and similar pairs
cur.execute("SELECT sentence_text FROM sentences WHERE contestant_id IN (1, 2);")
sentence1, sentence2 = cur.fetchall()

cur.execute("SELECT word1, word2 FROM similar_pairs;")
similar_pairs = cur.fetchall()

# Check similarity
result = is_similar(sentence1[0][0], sentence2[0][0], similar_pairs)
print(f"Are the sentences similar? {result}")

# Optionally store the result
cur.execute(
```

```
    "INSERT INTO similarity_results (sentence1_id, sentence2_id, is_similar) VALUES (%s, %s, %s);",
    (1, 2, result)
)
conn.commit()
cur.close()
conn.close()
```