Question-Answering System

Database Design Document

Luana Batista

Version 1.0

03/22/2019

Table of Contents

1. Overview

In a Question-Answering problem, a question needs to be stored with its respective answers. When a new question arrives, ideally, we want to check if a similar question was already asked before. If so, the question will not be stored as a new one. Instead, the answers for the similar question will be output to the user. This is an NLP (Natural Language Processing) solution for avoiding storing duplicate questions in question-and-answer sites such Quora and Stack Overflow. Figure 1 shows a typical architecture of a Question-Answering system based on NLP.

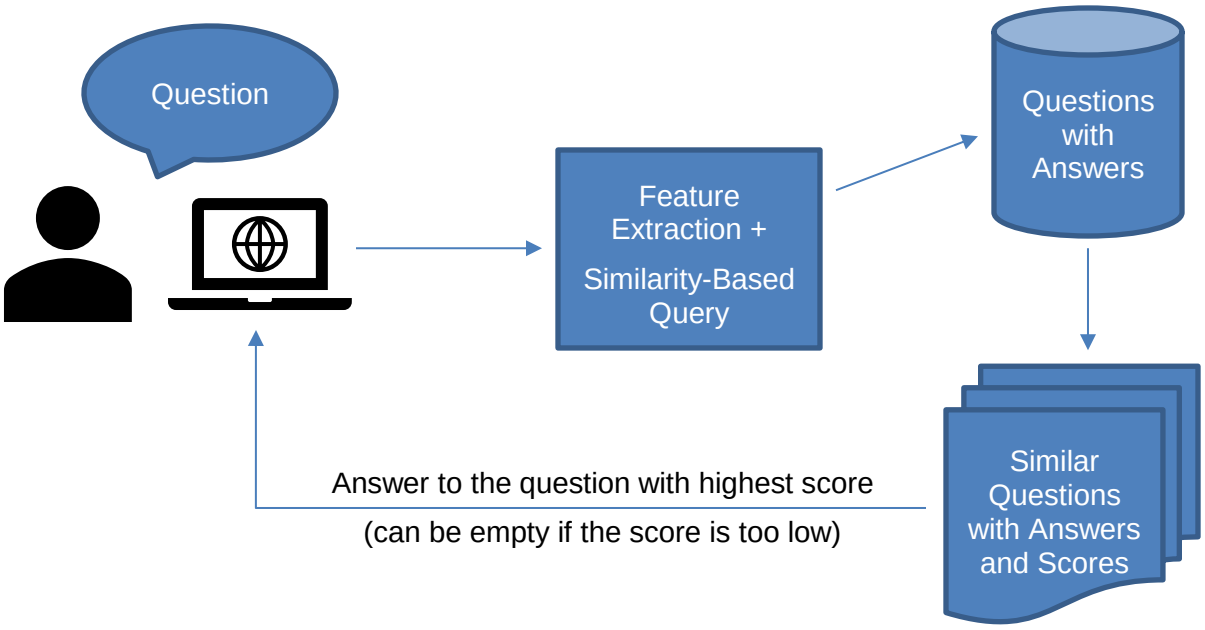

Figure 1. Typical architecture of a Question-Answering system based on NLP

Nevertheless, even using NLP, the set of questions/answers might become very big overtime. Therefore, a Big Data architecture would be the most suitable way of dealing with the increasing amount of data to be stored and accessed in this type of system.

From a question-and-answer website, when a user asks a specific question, this question is not immediately answered. In Quora's system for example, each question is reviewed and assigned to another user that can answer it according with his/her interests/knowledge. The assigned user can also re-assign the question to someone else that would be better suited to answer that question. By using NLP, a question could be answered immediately, as long as a similar question has been asked before.

The goal of this project is to model the question-and-answer pairs in a dedicated database considering that it will be interacting with a second database that models the users (who asked/answered questions), as well as their preferences, actions and relationships (like a social media system).

2. Assumptions/Constraints/Risks

In this project, we assume that target customer/company has (or has the means of obtaining) all the hardware and software resources necessary to the development of a free consumer internet product in which 10s of millions of users can connect.

Regarding the question-and-answer data to be stored, which is the main scope of this project, we need to foresee a scalable solution that expects to handle more than 10,000 questions per day.

As for the data properties, this system can behave well with a BASE (Basically Available, Soft state, Eventual) Consistency Model.

3. Design Decisions

Regarding the database choice, different options have been considered for this project.

A traditional Relational DB is an interesting option that allows us to model the entities-relationships of our system in the most natural way, that is, a *question* can have zero, one or many *answers* (see the Conceptual and Logical models in the next section). However, we can run into scalability issues that are better dealt with some noSQL solutions. The later can expand easily to take advantage of a distributed system ("scale out").

Among the noSQL DBs, Cassandra is a good option that has been considered (see QUESTION-ANSWERING.cql script). For the NLP subproblem of returning questions that are close to the asked question, the column "question" would be the only one employed for querying. Cassandra is well suited for this type of problem, since all questions would be stored in the same place and no other columns would be employed in a query – of course, the corresponding "answer" would be output in the end. However, we need to be sure that the system will always work this way, otherwise we could encounter some performance issues in the future.

Therefore, because of an eventual risk of data structure/behavior change, we opted for a document-database, MongoDB, in which we model each question-and-answer pair as a specific entry in a collection.

MongoDB has an efficient replication mechanism that provides redundancy and increases data availability with multiple copies of data on different database servers. The MongoDB replication mechanism is described in [2].

In addition to MongoDB, we decided to use the search engine ElasticSearch as a query layer, as it can offer much more text search capabilities than MongoDB alone. With ElasticSearch, we can perform "proximity search" with different similarity measures, which will be useful for our NLP module.

The connectivity between MongoDB and ElasticSearch is straightforward and can be performed by using the MongoConnector in combination with the ElasticDocManager (e.g., `mongo-connector -m localhost:27017 -t localhost:9200 -d elastic_doc_manager`).

Once the connection is stablished, we can query any data in MongoDB through ElasticSearch using the same MongoDB db/collection names, as presented in this example:

https://github.com/luana-be/CEB1250_repo/blob/master/elastic_mongo_connector.ipynb.

Note that, in order to use ElasticSearch together with MongoDB, the MongoDB server needs to be started as a replica set (e.g., `mongod --dbpath C:\data\db --replSet rs0`), as illustrated in Figure 2. In this schema, the documents are copied from MongoDB to ElasticSearch, which are constantly updated in order to keep MongoDB and on ElasticSearch in sync.

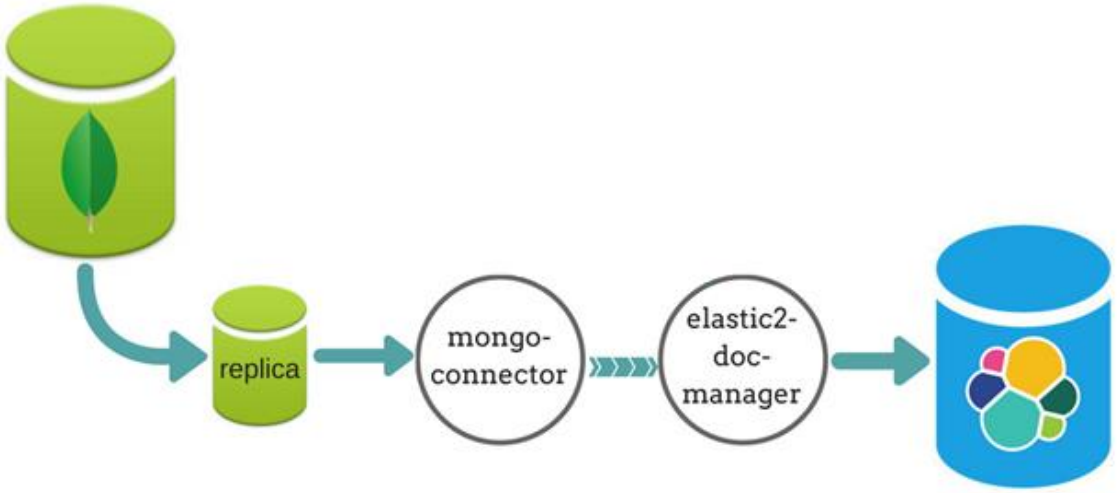

Figure 2. MongoDB-ElasticSearch connection and synchronization. Adapted from [9].

4. Detailed Database Design

As mentioned in the previous session, MongoDB has been chosen for our project. Nevertheless, in order to have a big picture on how the system entities interact with each other, we firstly defined the conceptual and logical models, as shows Figure 3 and Figure 4, respectively.

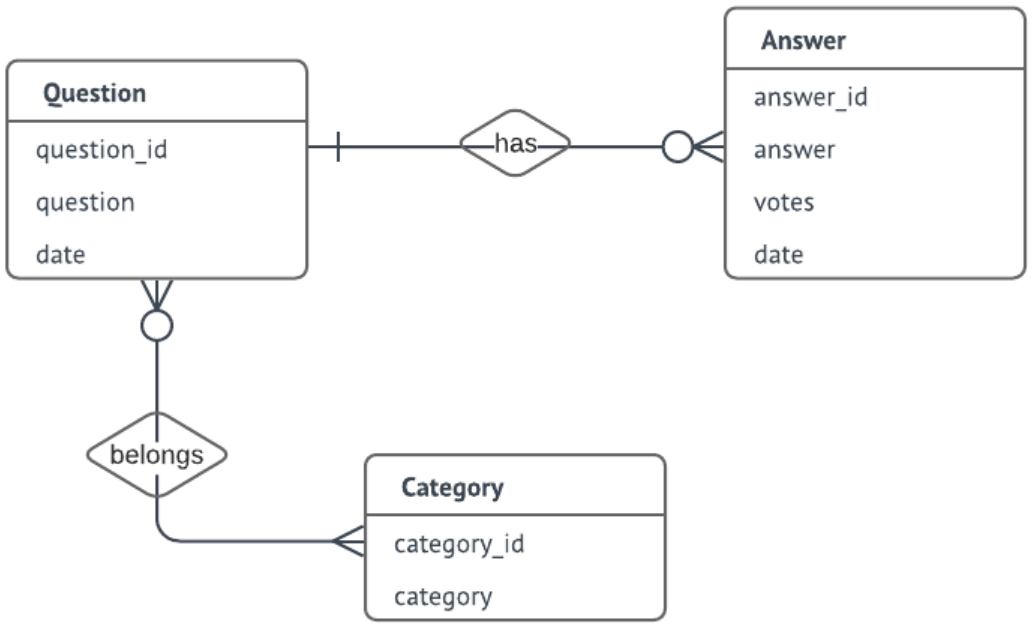

Figure 3 – Conceptual Model

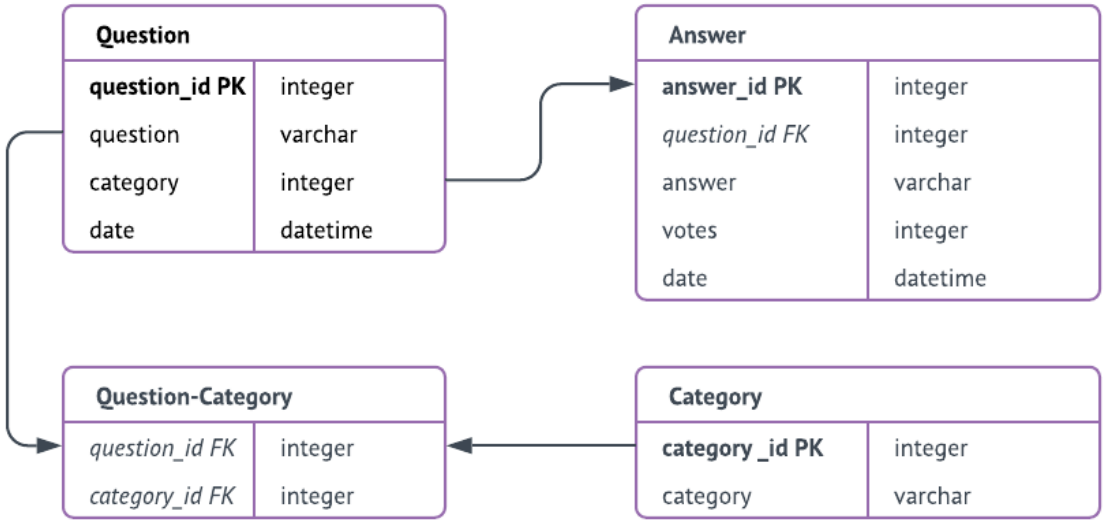

Figure 4 – Logical Model

The corresponding SQL DDL is presented in:

https://github.com/luana-be/CEB1250_repo/blob/master/QUESTION-ANSWERING.sql

On the other hand, for our MongoDB, only one collection needed to be defined, i.e., *question_and_answer_pais*. Below is an example of two entries in this database:

```
{
    question: "What is F1",
    question_date: new Date(),
    question_categories: ["all"],
    answer: "F1 is a sport",
    answer_date: new Date(),
    answer_votes: 0
}
{
    question: "What is F1",
    question_date: new Date(),
    question_categories: ["all", "computers"],
    answer: "F1 is a function key",
    answer_date: new Date(),
    answer_votes: 0
}
```

Please referrer to the corresponding DDL in:

https://github.com/luana-be/CEB1250_repo/blob/master/QUESTION-ANSWERING.mongo

It is worth noting that, when a question is asked for the first time, it is inserted into the database without an answer, for instance:

```
{
    question: "What is F1",
    question_date: new Date(),
    question_categories: ["all"]
}
```

Then, when an answer is available, the question is updated using the following command:

```
db.question_answer_pairs.updateOne(
    { question: "What is F1" },
    { $set:
          {
                answer: "F1 is a sport",
                answer_date: new Date(),
                answer_votes: 0
          }
    },
    { upsert: true }
)
```

Finally, when a second question is available for the same question, a new record is created as follows:

```
db.question_answer_pairs.insertOne(
    {
          question: "What is F1",
          question_date: new Date(),
          question_categories: ["all", "computers"],
          answer: "F1 is a function key",
          answer_date: new Date(),
          answer_votes: 0
    }
)
```

In the real-word scenario, the questions will be asked through a Web interface and inserted into the database using the PyMongo API.

References

MongoDB

[1] http://api.mongodb.com/python/current/tutorial.html

[2] https://docs.mongodb.com/manual/replication/

[3] https://api.mongodb.com/python/current/installation.html

ElasticSearch

[4] https://www.bitquabit.com/post/having-fun-python-and-elasticsearch-part-1/

[5] https://elasticsearch-dsl.readthedocs.io/en/latest/search_dsl.html

[6] https://marcobonzanini.com/2015/02/09/phrase-match-and-proximity-search-in-elasticsearch/

[7] https://www.elastic.co/guide/en/elasticsearch/reference/current/zip-windows.html

MongoDB + ElasticSearch

[8] https://medium.com/@xoor/indexing-mongodb-with-elasticsearch-2c428b676343

[9] https://code.likeagirl.io/5-different-ways-to-synchronize-data-from-mongodb-to-elasticsearch-d8456b83d44f

Source code snippets

https://github.com/luana-be/CEB1250_repo/blob/master/