# DATABASE DESIGN:

This section outlines the detailed design of the database for our disaster management system

**Key Considerations**

The database was designed with the following priorities in mind:

* **Performance**: Deliver rapid data retrieval and manipulation for time-sensitive decision making.
* **Scalability**: To accommodate the surge in data volume during emergencies.
* **Offline Functionality**: Enable data submission and access even without an internet connection.
* **Accessibility**: To ensure real-time data access for citizens, and responders.
* **Security**: Safeguard sensitive information and maintain data integrity.
* **Real-time Communication**: Facilitate instant information sharing and collaboration among users.

1. **IDENTIFYING DATA ENTITIES AND THEIR ATTRIBUTES**

Here we outlined the various entities relevant to our disaster management system, such as users, responders, forums, announcements, incidents etc. and the attributes of each entity so as to capture relevant information of each entity.

**Entities and their attributes**:

### User: contains info about the various citizens including their forums and other relevant info

### Responders: contains info about the various responders

### Incident: to store info about the disasters which were reported and their status

### Help Requests: to store data about the help requests made by users/citizens

### Forums: to store info about the various forums in our system.

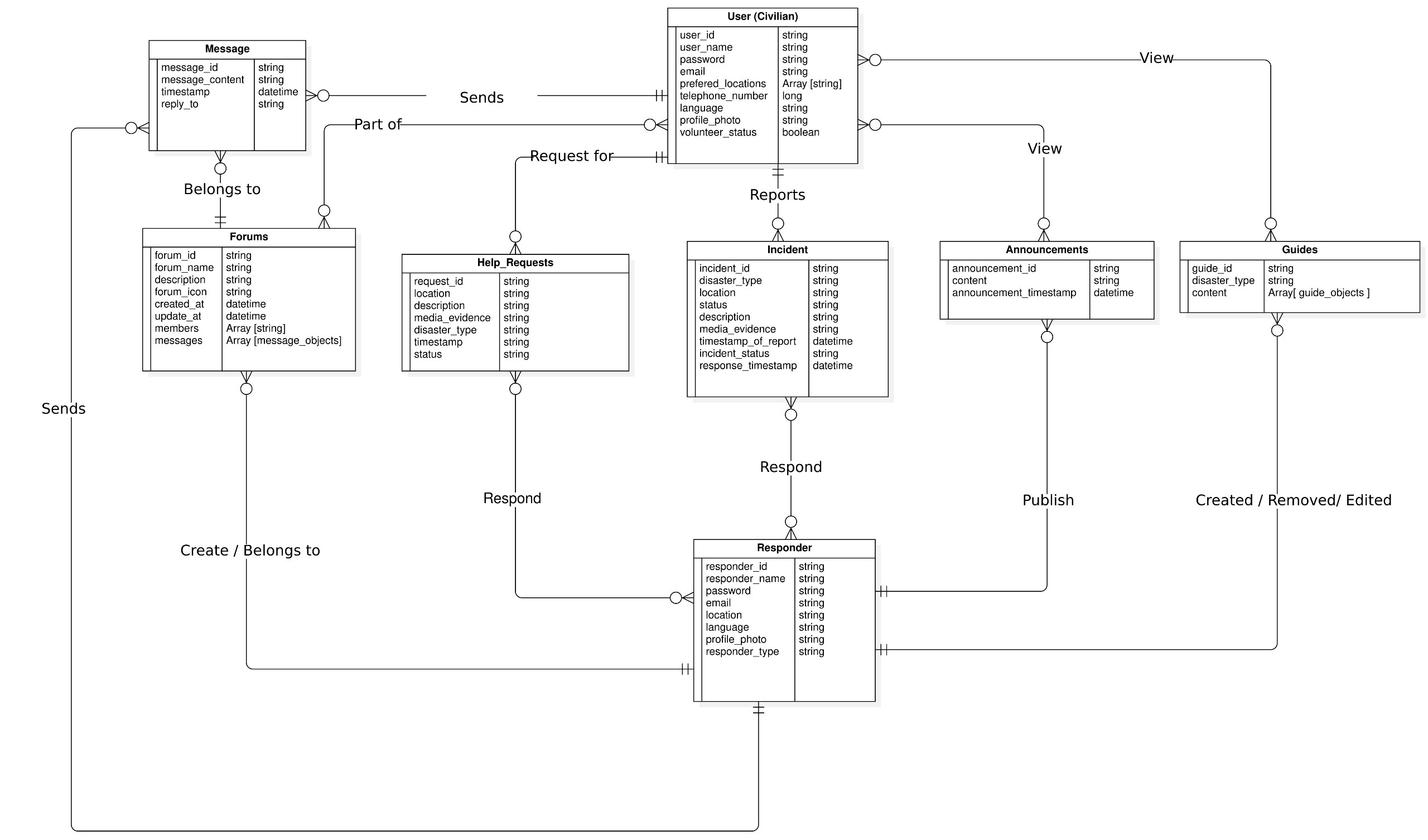
### Announcements: to store the various announcements passed out by the various responders.

### Guides: to store the guides for the various disasters

1. **DEFINE RELATIONSHIPS WITH REFERENCES**

Since relationships between entities are not explicitly defined using foreign keys, we outlined the relationships between various entities and stored references (document IDs) within documents to link them together by evaluating various criteria.

Here’s a visual representation of the system, including the entities, their attributes and their relationships/interactions with each other.



Since NoSQL databases have no specifically defined approach to producing a schema because the schema structure is solely dependent on the interactions and queries within the project. The multiplicities of the relationship between entities in the project also contribute to schema structure.

The following methods were adopted to create a schema with optimal performance

1. **Optimization structure for reading of frequently read data**

With this approach the NoSQL documents structure was in such a way that frequently read data or queried data were:

* **Denormalized**: With this approach frequently read data were made redundant across documents such that they are easily accessible, saving the overhead time for referencing.
* **Index**: Frequently queried fields were index so as to optimize the reading process.

1. **Optimization structure for writing of frequently written data**

With this approach the NoSQL documents structure was in such a way that frequently written data were:

* **Normalized:** With this approach frequently written data was made very less redundant such a single change in one document does force multiple updates in other documents containing same data to keep data consistency and integrity.

In cases where the data neither showed a frequent read or frequent write following was adopted

1. **One-to-one Relationships**

Reading was optimized here through denormalization at either end as this will reduce query complexity, shorten the fetch time and hence increase performance in the sight of the user.

1. **One-to-many Relationships**

With the one-to-many relationship denormalization was adopted at the many end such that data in the one end is made redundant at the many end. This will optimize reading.

* **Many-to-many Relationships**

With many-to-many relationship normalization was chosen as this will in a long run keep performance smooth even as data grows. In other words, too much data will not be stored in one document as the size of data grows.

* **Data size and growth**

Data with great anticipated growth sizes were either normalized if there was a need for optimizing write or no need for optimizing read. Also, where reading was still to be optimized regardless the size a sub collection embedded in another collection was used.

After following this approach, we came out with the various entities, their relationships and also the various references to other tables.