Using Document Store for 3D Virtual Collections

Emanuela Mitreva[1], Vladimir Georgiev[2]

[1]Faculty of Mathematics and Informatics, Sofia University "St. Kliment Ohridski"
[2]Institute of Mathematics and Informatics – Bulgarian Academy of Sciences
emitreva@gmail.com, vlado80@gmail.com

Abstract. This paper describes the latest improvements to a recently developed online environment for managing virtual collections of 3D objects. As part of that effort we introduced a document-oriented NoSQL layer for storing the data describing the 3D objects and collections, provided by the MongoDB engine. This technology has many advantages for describing the type of data and metadata used in our application, which are also valid for and can be applied to the field of digital libraries in general.

Keywords: Document Store, 3D Objects, Virtual Museum, MongoDB

1 Introduction

In our previous work [1] we introduced the development of a 3D environment that can be used for making collections of objects for virtual museums and sharing those collections online. The system can be used in many different scenarios - from allowing the user to show a single big object to creating a collection of multiple smaller objects. It is a complete environment for creating, editing, publishing, unpublishing and deleting virtual collections, which can be examined in the customers web browser from a first-person perspective using the keyboard and the mouse.

One important area of improvement for the initial version of the application that we identified, was how the information for the objects and the collections, including how the object can be positioned and/or visualized, will be stored more efficiently. This includes the ability to store objects of all kinds, having any types of properties, defined on them in any structure, as well as being more scalable and reliable. Another thing we needed to do was to extend the structure of the objects to hold not only information about how they should be positioned in a collection, but also historical information about the specific object. Those two tasks are related to each other, because the information needed for the objects limits the possible types of databases that can be used. The initial version of our application uses the SQLite relational database engine for storing the information. Because the structure of the object data was initially static and predefined, to support new properties in the future we would need changes in the DB schema and the code and in the application. Another important problem was scalability, which is a crucial feature of a digital library web application used for storing large amounts of real-life data.

2 The Document Store Approach

In the initial implementation of our web application, the SQLite database engine was used to store the collections of objects and other information needed for the application. But even though the relational model is the most commonly used and a proven concept in terms of reliability and stability, we needed to extend the storage layer to support specific requirements of the application and the type of data it supports. The relational model has several limitations on the type of data that can be stored - the properties that are stored have to be atomic - i.e. simple types like integers, strings, etc. For example we cannot store effectively the whole JSON string, used for the visualization of an object in a collection. If we are using a relational database, the information for the object and the data, needed for the visualization of the item, have to be split in different tables. The database should be normalized, ideally in third normal form, to avoid any redundancies and inconsistencies. When rendering a collection for the virtual museum, all the information should be retrieved from the tables and formatted in the proper way to be visualized. If there are too many objects, this can be very slow and the user might have to wait some time for all the objects of the collection to be visualized. Another reason that a relational database is not be the optimal solution in our case is the fact that we need different properties to describe an item. When a relational database is used, all possible attributes, which might be needed for every type of object, have to be known ahead. If the different objects have many different attributes, there will be many columns for the objects, but only part of them will be filled out for a particular object, and the rest will be set to null values. Even if this is acceptable as the design of the table (many columns, but only some of them filled out will slow the retrieval of the data), if a new object property is introduced, the database schema would need to be changed. In the general case altering the schema of a table after it has some records in it (especially if there are many records) is difficult and will result in downtime of the application until the operation is complete.

To address the problems and the limitations of the relational databases, there are several alternative architectures that can be used [2]. Those include different storage types - column store, key-value store and document store, which are the different types of NoSQL solutions according to Cattel [3]. To meet our requirements, we needed a database store that can offer similar stability, but more flexibility and that offers the data to be stored in format similar to what is used for the visualization. The key-value store [4] approach was too simple to hold the type of data that was needed and the column store was too similar to the relational model, so we picked a document store solution for our case.

Document-oriented storage is used to store and manipulate data in structured or semi-structured format in a form of document [5]. The document is in a way similar to the records in relational stores, but the data is in a more rigid and flexible format - usually JSON, BSON or XML [6]. The stored documents have practically no limit to the number of levels that can be nested, allowing the information for an object - both the data used to visualize the item and information for the item itself, to be saved in one place. Later in this paper we will discuss whether this is the best approach on a logical level, but otherwise the document store allows it. The document store unlike

some other alternative solutions (key-value store for example) supports indexes - both simple and compound and even on inner levels of the document [3], and considering the amount of data that should be stored for the application, this is a definitive advantage.

The document store that we have chosen to use is mongodb. Mongodb is a document store, NOSQL solution that was created with the idea to handle big data, suggested by its name (huMONGOus) [7]. Mongodb data is stored as JSON [8] and its schema is not required to be defined prior to importing some data in it. Instead of tables, mongodb has collections, which store the documents, unlike the relational model, which has tables with rows and columns. As a document store it has flexible schema, i.e. the records or the documents that can be stored can differ in their attributes. This can be both an advantage and disadvantage, because the flexible schema also allows erroneous data and documents to be inserted. One significant difference between the relational model and mongodb is that the document-oriented store does not support join operations. However, this is not necessarily a disadvantage, because it depends on the data that needs to be stored. In the general case everything that needs to be stored can be fit into one or a few collections, thus making the joins unnecessary.

The document store has several advantages over the relational solutions - our application uses JSON and because the data in mongodb is stored in JSON format, it is easier for the application that is rendering a collection to gather all the needed information. It requires no effort to structure the data in the format that is used for the visualization of a collection of objects. Also mongodb has a flexible schema, thus allowing the records to have different properties - based on the type of object, we can have different attributes. This reduces the extra space to save for the attributes that are not needed for a specific type of item. Also if a different type of an object or attribute is needed for the new records, this will not require any changes to the schema - the records are simply added to the existing storage.

3 Extending the Storage Layer

The initial version of our application did not store data for its objects in a flexible way - it did not offer a way to store additional information about an object without changing the schema. That is why the current format of the stored data needed to be changed to support additional information and the way the objects were stored also had to be optimized to make the extraction of the information more effective. The extended structure of the items gives the authors the option to inspect the objects and learn more about them before they actually add them to a collection. This way the application can be used not only for making 3D collections of objects, but examining specific items.

The properties that can be added as additional characteristics can vary, based on the type of the object, but we have added the following properties, which we considered important for each object, regardless of its type:

- id - id of the item, this id also can be used to identify the object when it is part of a collection;
- origin - information about the origin of the item;
- year - year of origin for the item;
- provenance - information about the provenance of the item - where it originated, etc.;
- author - who is/are the authors of the item;
- location - where the item was found;
- year_found - in what year the item was found;
- type - painting, sculpture, etc.;
- country - country in which the object was currently situated;
- information - some additional information that was not covered by the other proper-ties.

Based on these additional properties, we divided the information into two collections. As we already mentioned mongodb does not support joins, hence the granularity of the information that is stored. This might lead to some inconsistencies of the data, because the database is not normalized as the relational databases usually are. The information that we are storing is several levels deep (objects list is three levels deep), but this is not a problem for mongodb - there is no limitation on the nesting levels. However in order to be able to search and retrieve the needed information effectively, we decided to put some indexes - on the id of the object, on the id of a collection. This speeds up the process of extracting the information for an object in a specific collection. Those indexes are enough to speed up the process of extraction of the information for the rendering, however if we want to expand even more the functionalities of the application, we can add some indexes on properties, that is most likely to be used to filter the data - e.g. search for item from a specific year/period, of a specific type, etc.

The information for the objects is split into two collections - one with the information for the rendering and one for the other information for the item. So we have the following properties for the collection that will be used for the rendering: collection id, collection name (same user defined name), list of objects that are part of the specific collection and each object will be defined by id, name, description (this is the description of the object that is shown when the user clicks on an item), properties for the format, object url, material url and texture, coordinates of the position, rotation information, scale and coordinate and dimension of the base. Most of these properties were already present in the previous version of the application, what we have changed is the way there are stored. The properties for the objects themselves is what is introduced by this paper - id of the object, origin, year, author, location year, when the item was found, type, country and some additional information, that is not already covered by the other properties.

All the information about the items - for visualizing and description - are to be kept in the document store, but the information about the users and groups will still be kept in the SQLite. Since the document store does not support transactions and does not

support the ACID properties as the relational database, it is better to keep that information in the relational store.

4 Conclusions and Future Work

The general purpose of the developed 3D environment is to create collections with 3D objects, that can be visualized, and although the initial implementation of the application serves this purpose, some improvements and extensions to it were needed. We have made improvements in two major areas - extending the data that was stored for the objects and change the storage type for the items' information. The initial version stores the data for the objects and the users in a relational database - SQLite. The user management was best stored in a relation store as was the initial idea, because of the stability and consistency of the relational model. However, for the objects' data, the relational store did not seem flexible enough. The data for the 3D objects was initially stored in a normalized relational database in columns and rows, and sent to the application in a JSON format several levels deep. The additional effort of formatting the data was an unwanted necessity, because the relational model allows only simple types of data, and no arrays and dictionaries. That was not the only disadvantage of the relational model for a primary storage of the objects' data. Any change to the properties required changes in the application and the database. Such changes can result in downtime for the whole application and such changes in the database can even break the existing data. When an additional property is needed, the schema of a table should be altered to adjust the new attribute and if there are specific limitations to the column the relational model might not allow such operation it before all the already inserted records comply with the limitation of the new column. Because of those restrictions and because of the ACID properties of the relational model, we decided that a store with more flexible schema is more appropriate for information about the items. Also since the format of the data that was sent to the application is JSON, a data store that can store the records in that same format or at least similar was needed. After researching alternative solutions, we have chosen to use the document-oriented mongodb. This database engine stores the data in a form of documents, defined in a JSON format, which was appropriate as the format of the data that our application uses was also JSON. The schema of the document store is flexible and the schema of a collection (i.e. tables in the relation model) can be defined by the records that are inserted, and any new record can potentially have completely different format. This is allowed, but is actually not advised, because the application will expect certain properties to be present, but fluctuation in what properties are present for a specific object is more than allowed and appropriate. Of course the application should be able to process all types of objects and all the different types of attributes each object has. Apart from the type of storage that was changed we have extended the information that was stored for the objects by including some important properties that can provide vital information for the item regardless of its type.

Both of these modifications provide significant enhancements to the initial version of the application and some new functionality that can be added will benefit from

those changes. One thing that can benefit from the extended model and the change of the data store is the implementation of a web services that could provide the users with advance search of the objects - searching by specific criterion and even visualizing the objects that match the filters. Another thing that we could do is to test whether the proposed arrangement of the data in the two collections is optimal for the efficiency and performance of the application. Based on the data that should be sent for the rendering of the 3D collection and the new web services, it might turn out that additional indexes will be needed to optimize the queries or a different split of the information.

References

1. Georgiev, V.: A Web Application for Creating and Presenting 3D Object Expositions. Proc. Fourth Int. Conf. Digit. Present. Preserv. Cult. Sci. Heritage, Veliko Tarnovo, Sept. 18-21, Bulg. (2014).
2. Mitreva, E., Kaloyanova, K.: NoSQL Solutions to Handle Big Data. Proc. Dr. Conf. MIE. 77–86 (2013).
3. Cattell, R.: Scalable SQL and NoSQL Data Stores. 39, (2010).
4. Seeger, M.: Key-Value stores : a practical overview. 1–21 (2009).
5. MongoDB Manual, http://docs.mongodb.org/manual/.
6. Padhy, R.P., Patra, M.R., Satapathy, S.C.: RDBMS to NoSQL: Reviewing Some Next-Generation Non-Relational Database's. 15–30 (2011).
7. Redmond, E., Wilson, J.R.: Seven Databases in Seven Weeks. (2012).
8. Zaki, A.K.: NoSQL Databases : New millenium database for Big data, big users, cloud computing and its security challenges. 403–409 (2014).