International Journal of Internet, Broadcasting and Communication Vol. 16 No.4 460-465 (2024)
http://dx.doi.org/10.7236/IJIBC.2024.16.4.460

IJIBC 24-4-50

Modeling with Design Patterns in MongoDB for Public Transportation Data

Meekyung Min

Professor, Dept. of Software, Seokyeong University, Seoul, Korea
mkmin@skuniv.ac.kr

Abstract

MongoDB, a document-based database, is suitable for distributed management environments of large-scale databases due to its high scalability, performance, and flexibility. Recently, as MongoDB has been widely used as a new database, many studies have been conducted including data modeling for MongoDB and studies on applying MongoDB to various applications. In this paper, we propose a data modeling method for implementing Seoul public transportation data with MongoDB. Seoul public transportation data is public data provided by the Korea Public Data Portal. In this study, we analyze the target data and find design patterns such as polymorphic pattern, subset pattern, computed pattern, and extended reference pattern in the data. Then, we present data modeling with these patterns. We also show examples of implementation of Seoul public transportation database in MongoDB. The proposed modeling method can improve database performance by leveraging the flexibility and scalability that are characteristics of MongoDB.

Keywords: *Data Modeling, MongoDB, Korea Public Data Portal, Seoul Public Transportation Data, Design Pattern*

1. INTRODUCTION

It is difficult to implement applications for very big data with relational database systems. A new database called NoSQL provides high scalability and flexibility in a distributed management environment of a large database. Among NoSQL databases, MongoDB, a document-oriented database, is widely used [1]. This is because MongoDB satisfies large-scale distributed databases, high availability, performance, and scalability, while also performing well the query functions for data manipulation used in relational databases. There has been a lot of research done on MongoDB. These include studies on migrating relational DBs to MongoDB [2-4] and studies on schema and modeling [5, 6].

The document data model is inherently flexible, allowing the data model to support application requirements well. This flexibility can lead to unnecessary complexity in the schema. Schema design requires consideration of simplicity as well as performance and scalability. In order to establish a schema design method that works well in MongoDB, twelve design patterns were presented in the article [7]. This method improves the performance of the database while maintaining scalability and simplicity. This paper presents the process of modeling Seoul public transportation data using these design patterns.

The Korea Public Data Portal is an integrated window that provides public data created or acquired and managed by government agencies in accordance with the government's public data opening policy [8]. The contents include various fields such as transportation, health, real estate, and medical care. The public data portal DB accumulates a huge amount of data and provides it to users in the form of CSV, JSON files, and open API [8]. This study targets Seoul public transportation data. Thousands of data are input from card readers

Manuscript Received: October. 25, 2024 / Revised: November. 1, 2024 / Accepted: November. 6, 2024
Corresponding Author: mkmin@skuniv.ac.kr
Tel: +82-2-940-7750, Fax: +82-2-919-0345
*Professor, Department of Software, Seokyeong University, Seoul, Korea

and sensors in the subway stations and buses every day, so a database like MongoDB that process big data is suitable for this large data. In addition, when analyzing the nature of the transportation data, a denormalized data model is more suitable than a normalized relational data model.

In this paper, we propose a data modeling with design patterns for implementing database in MongoDB. Among the twelve design patterns in [7], we model data using four design patterns that are suitable for the public transportation data. In section 2, we summarize the Seoul public transportation data of the Korea Public Data Portal. In section 3, we show the process of modeling data in MongoDB with the design patterns. In section 4, we discuss the results of the modeling and the direction of further research.

2. SEOUL PUBLIC TRANSPORTATION DATA

Table 1 summarizes Seoul public transportation data in the Korea Public Data Portal. [8, 9]. The format is file (CSV, JSON) and API. Table 1 mainly lists the data required for the pattern-oriented modeling presented in this paper. The data in the Seoul public transportation database are categorized into subway station information, train operation information, subway passenger statistics and bus information. Bus data mainly consists of bus passenger information and bus operation information.

Table 1. Seoul Public Transportation Data in Public Data Portal

Data	Category
Subway station information, Station address and phone number, Subway lines information, Transit information, Entrance information, Elevator location, Station facilities, Etc.	Subway Station Information
Number of passengers (by line, station, hour, day, month, year), Number of transit passengers (by station, year …), Number of free-fare / full-fare passengers (hour, day, month …), Number of tickets (by type), Ranking statistics of passengers, Etc.	Passenger Statistics
Train operation schedule, First/last train information Train arrival information, Train location information, Train time table, Etc.	Train Operations
Bus line Information, Bus station coordinates, Number of passengers (by station, bus number, hour…), Number of passengers of origin and destination station, Bus location, Number of transit passengers (by hour, by passenger type), Etc.	Bus information

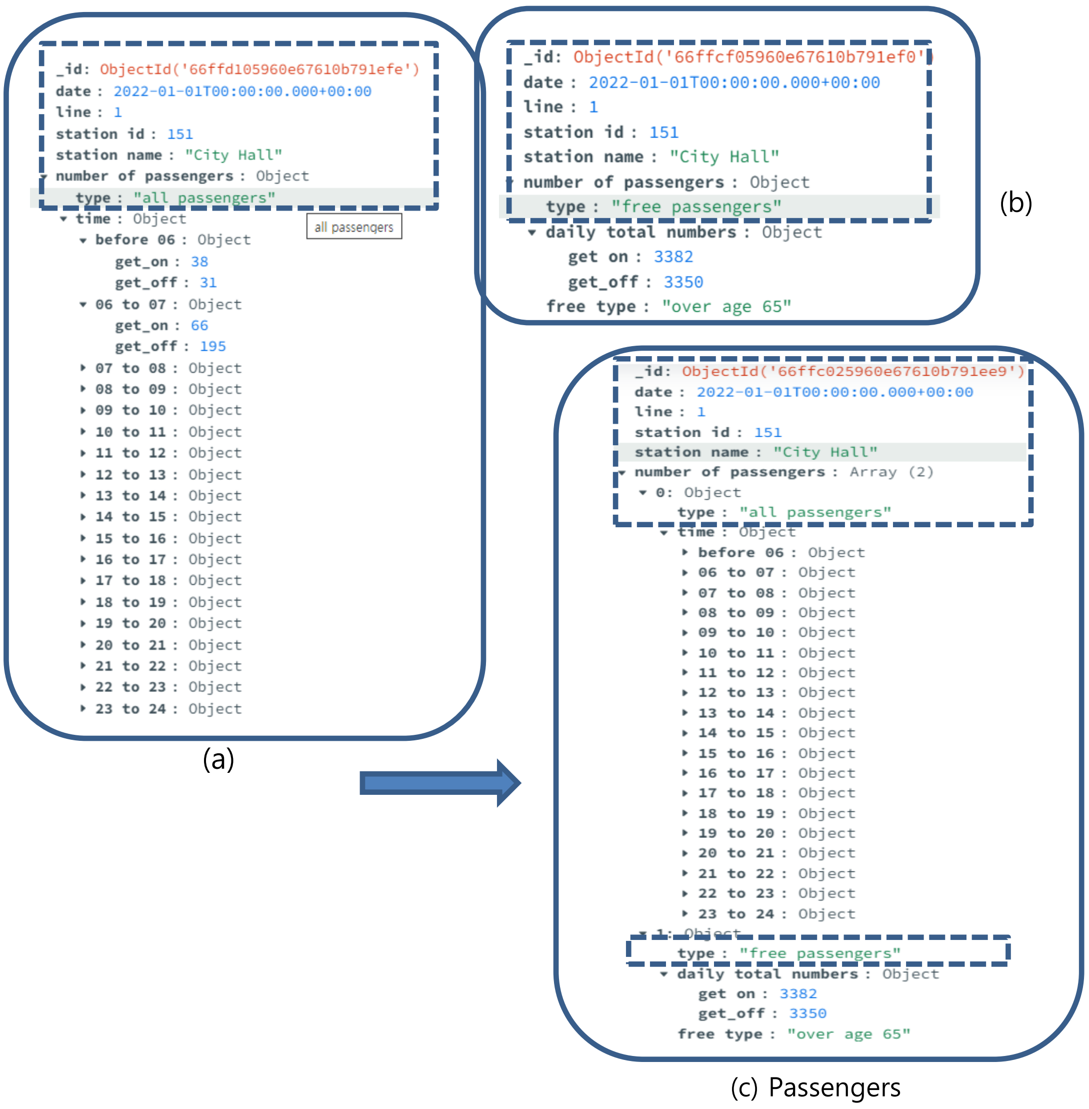
3. MODELING WITH DESIGN PATTERNS

In this section, we analyze the public transportation data in Table 1 to find design patterns. As a result of analyzing the data, we found the polymorphic pattern, subset pattern, computed pattern, and extended reference pattern among the design patterns suggested in [7]. We explain the modeling process using these patterns and show examples of collections and documents implemented in MongoDB.

3.1 Polymorphic Pattern

A polymorphic pattern means that documents in a collection are similar but not identical. For example, information on subway passengers contains statistics on the number of passengers. The types of passengers can be divided into full-fare passengers and free-fare passengers. These two types mainly have common information and store slightly different information depending on the type. These are polymorphic patterns. In this case, documents are stored in one *Passengers* collection without having to create two separate collections. Since we don't separate the two types of passenger collections, we can speed up when queries are performed on all passengers. This means that we can leverage the flexibility of MongoDB to improve performance.

Figure 1 shows an example of modeling with a polymorphic pattern. Fig. 1(a), (b), and (c) are part of screen captures of MongoDB Compass, respectively. Fig. 1(a) is a document showing the total number of passengers getting on and off at City Hall Station by hour. Fig. 1(b) shows the number of elderly free-fare passengers getting on and off at City Hall Station by day. Instead of separating the data into two collections Fig. 1(a) and Fig. 1(b), we can model it as a single *Passengers* Collection, as in Fig. 1(c). In Fig. 1(c), by using the "type" attribute, we modeled two types of documents as a single collection. There are many similar polymorphic pattern cases in Seoul public transportation data.



```
_id: ObjectId('66ffd105960e67610b791efe')
date : 2022-01-01T00:00:00.000+00:00
line : 1
station id : 151
station name : "City Hall"
number of passengers : Object
  type : "all passengers"
  time : Object
    before 06 : Object
      get_on : 38
      get_off : 31
    06 to 07 : Object
      get_on : 66
      get_off : 195
    07 to 08 : Object
    08 to 09 : Object
    09 to 10 : Object
    10 to 11 : Object
    11 to 12 : Object
    12 to 13 : Object
    13 to 14 : Object
    14 to 15 : Object
    15 to 16 : Object
    16 to 17 : Object
    17 to 18 : Object
    18 to 19 : Object
    19 to 20 : Object
    20 to 21 : Object
    21 to 22 : Object
    22 to 23 : Object
    23 to 24 : Object
```

```
_id: ObjectId('66ffcf05960e67610b791ef0')
date : 2022-01-01T00:00:00.000+00:00
line : 1
station id : 151
station name : "City Hall"
number of passengers : Object
  type : "free passengers"
  daily total numbers : Object
    get on : 3382
    get_off : 3350
  free type : "over age 65"
```

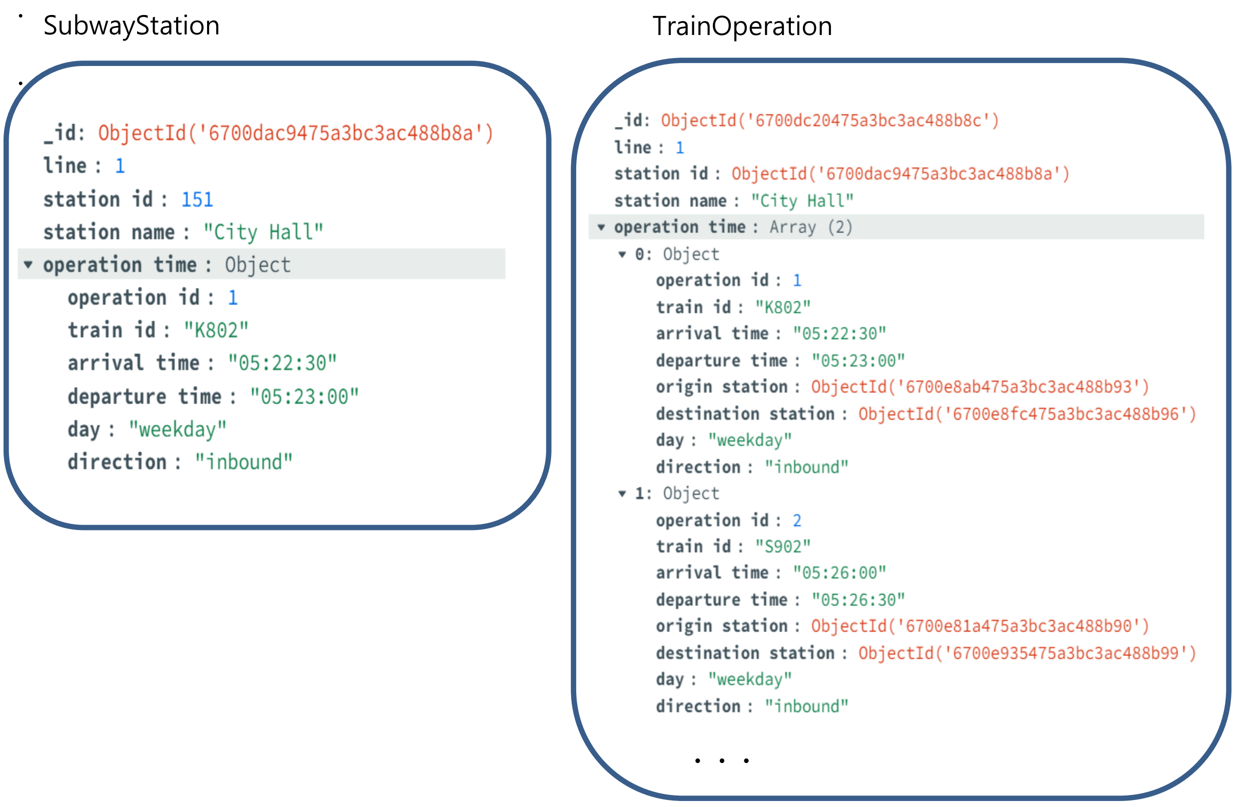
```
_id: ObjectId('66ffc025960e67610b791ee9')
date : 2022-01-01T00:00:00.000+00:00
line : 1
station id : 151
station name : "City Hall"
number of passengers : Array (2)
  0: Object
    type : "all passengers"
    time : Object
      before 06 : Object
      06 to 07 : Object
      07 to 08 : Object
      08 to 09 : Object
      09 to 10 : Object
      10 to 11 : Object
      11 to 12 : Object
      12 to 13 : Object
      13 to 14 : Object
      14 to 15 : Object
      15 to 16 : Object
      16 to 17 : Object
      17 to 18 : Object
      18 to 19 : Object
      19 to 20 : Object
      20 to 21 : Object
      21 to 22 : Object
      22 to 23 : Object
      23 to 24 : Object
  1: Object
    type : "free passengers"
    daily total numbers : Object
      get on : 3382
      get_off : 3350
    free type : "over age 65"
```

Figure 1. Modeling with Polymorphic Pattern

Another example is subway information and bus information. Both of them have common information such as date, station id, station name, number of passengers, etc. And they have some different information. If there are many query requests to search subways and buses at once, it would be efficient to model them as a single collection. However, from analyzing the transportation data, we can see that subway data and bus data are usually queried separately. Therefore, in this case, we do not model them as a polymorphic pattern.

3.2 Subset Pattern

The subset pattern is used to increase speed when the entire large document cannot be loaded into the main memory at once. The Seoul public transportation database contains a large amount of train operation information. It provides real-time train operation times for each station. If this large amount of train operation information were stored together with the station information, it would be too large to load into main memory. Therefore, we split the train operation information for each station into the *SubwayStation* collection and the *TrainOperation* collection. The *SubwayStation* collection only has the most frequently used information: information about the next train, the first train, and the last train. Since past train operation history is not of much use, the entire train operation information is stored separately in the *TrainOperation* collection. This is shown in Figure 2.



```
_id: ObjectId('6700dac9475a3bc3ac488b8a')
line : 1
station id : 151
station name : "City Hall"
▾ operation time : Object
    operation id : 1
    train id : "K802"
    arrival time : "05:22:30"
    departure time : "05:23:00"
    day : "weekday"
    direction : "inbound"
```

```
_id: ObjectId('6700dc20475a3bc3ac488b8c')
line : 1
station id : ObjectId('6700dac9475a3bc3ac488b8a')
station name : "City Hall"
▾ operation time : Array (2)
  ▾ 0: Object
      operation id : 1
      train id : "K802"
      arrival time : "05:22:30"
      departure time : "05:23:00"
      origin station : ObjectId('6700e8ab475a3bc3ac488b93')
      destination station : ObjectId('6700e8fc475a3bc3ac488b96')
      day : "weekday"
      direction : "inbound"
  ▾ 1: Object
      operation id : 2
      train id : "S902"
      arrival time : "05:26:00"
      departure time : "05:26:30"
      origin station : ObjectId('6700e81a475a3bc3ac488b90')
      destination station : ObjectId('6700e935475a3bc3ac488b99')
      day : "weekday"
      direction : "inbound"
        . . .
```

Figure 2. Modeling with Subset Pattern

Another example is about passenger statistics. Each station records the number of passengers getting on and off every hour. This number accumulates over time and becomes very large. Therefore, the *SubwayStation* collection is designed to contain only the data that is frequently used in queries, and *Passengers* collection stores detailed hourly data. *SubwayStation* may only store passenger statistics for a period of one week or one month, for example. As shown in Figure 1 in Section 3.1, the detailed passenger numbers by hour are in the *Passengers* collection.

3.3 Computed Pattern

Computed patterns are used when modeling data that is computed repeatedly. The computed values of attributes that are repeatedly used in queries are stored instead of being recalculated each time. The purpose is to eliminate the overhead of repeating computation every time a query occurs and to increase speed. Computed patterns are efficient when used on low-write and read-intensive data. There are many statistical and computational values in the public transportation database of this study. Among them, some major data are as shown in Table 2. The table shows the update frequency and write intensity for each data. Since read intensity varies depending on the application, it cannot be determined by the open data alone. Low-write attributes can be considered as read-intensive attributes with relatively many reads, so they are modeled as computed patterns.

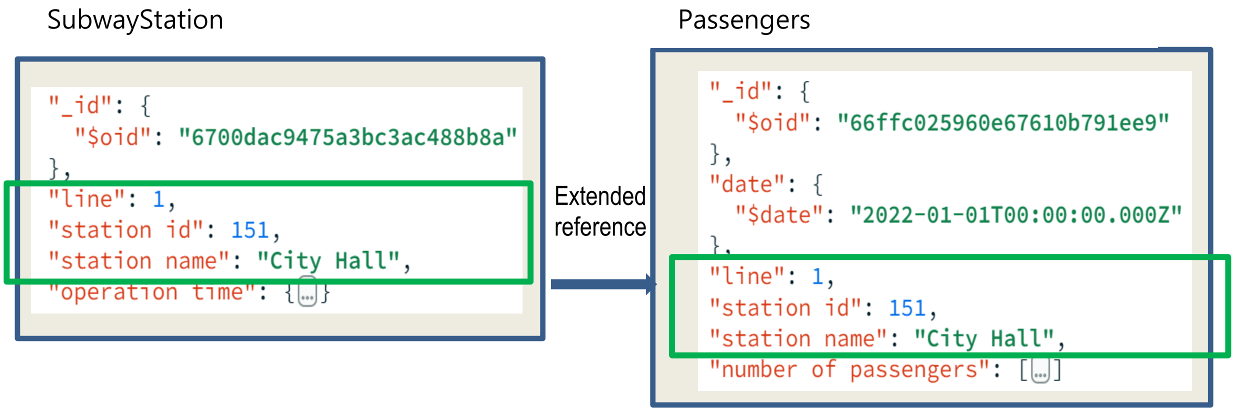
An example of a computed pattern is in *Passengers* in Figure 1 in Section 3.1. It is in the "free-fare passenger" type in the *Passengers* document. The "get-on" and "get-off" values of free-fare passenger are values computed on daily basis.

Table 2. Computed Patterns

Data	Update Frequency	Write intensity	Computed pattern
Number of boarding passengers (by station, day)	once/day	low	computed
Number of alighting passengers (by station, day)	once/day	low	computed
Number of free-fare passengers (by station, month)	once/month	low	computed
Number of free-fare boarding passengers (by station, day)	once/day	low	computed
Number of free-fare alighting passengers (by station, day)	once/day	low	computed
Number of boarding passengers (by station, ticket reader)	-	high	no
Number of alighting passengers (by station, ticket reader)	-	high	no

3.4 Extended Reference Attribute

In a relational database, relations are normalized and there is no duplication between different relations. When a relation references another relation, a join is performed. The join operation has the disadvantage of being slow when the data size increases. Speed is even more important in large databases such as MongoDB. Consider a case where a document references a document in another collection. Since each document has an object id (oid), you can reference another document using the oid. In this case, there is no duplication and you must do a join. However, unlike relational DBs, MongoDB can reference documents in other collections while including some duplication. This pattern is called the extended reference pattern. Since the duplicate attributes exist in both collections, a join is not necessary when searching for these duplicate attributes, and the query can be answered with only one collection. The downside is that if attributes are duplicated on both sides of the two collections, it takes time to maintain consistency when updating attributes. Therefore, attributes that rarely change are selected as duplicate attributes.



```
"_id": {
  "$oid": "6700dac9475a3bc3ac488b8a"
},
"line": 1,
"station id": 151,
"station name": "City Hall",
"operation time": {…}
```

```
"_id": {
  "$oid": "66ffc025960e67610b791ee9"
},
"date": {
  "$date": "2022-01-01T00:00:00.000Z"
},
"line": 1,
"station id": 151,
"station name": "City Hall",
"number of passengers": […]
```

Figure 3. Modeling with Extended Reference Pattern

Figure 3 shows two collections, *SubwayStation* and *Passengers*, as an example. *SubwayStation* is referenced in *Passengers*. Assume that the *Passengers* document references the *SubwayStation* document using the oid.

In order to retrieve the passenger information of the station, the documents of the two collections must be loaded into the main memory and joined. As the document size continues to grow, this takes a lot of time and reduces performance. Therefore, we duplicate frequently used but rarely updated attributes such as *station line number*, *station ID*, and *station name* in both *SubwayStation* and *Passengers*. In this case, when these frequently used attributes are in the query result, there is no need to join the two collections.

4. CONCLUSION

In this paper, we present data modeling using design patterns to implement Seoul public transportation data of the Korea Public Data Portal into MongoDB. We analyze the data to find cases corresponding to each design pattern, model them, and show examples of implementing a database in MongoDB. The first pattern, polymorphic pattern, allows documents with slightly different attributes to coexist in a single collection. The second pattern, subset pattern, reduces the size of the document by storing detailed information in a separate collection. The third pattern, computed pattern increases query execution speed by reducing repetitive calculations. The fourth pattern, extended reference pattern, allows duplicate attributes in multiple collections to reduce join operations. The advantage of modeling with design patterns presented in this paper is that it can improve database performance by utilizing the flexibility and availability of MongoDB.

We modeled and implemented a database based on open files and API's provided by the Korea Public Data Portal. In order to complete the modeling results, we need more information, such as how frequently actual data is read and written, which data is needed together from the database, what is the size and performance consideration of the document, and what is the growth rate of the data. These informations are determined and vary depending on the application. Therefore, for further work, when building a public transportation database application using MongoDB, the modeling method presented in this paper can be applied to complete the database design.

REFERENCES

[1] E. Plugge, D. Hows, P. Membrey, and T. Hawkins, *The Definitive Guide to MongoDB: A complete guide to dealing with Big Data using MongoDB*, Apress, 2015.

[2] A. Erraji, A. Maizate, and M. Ouzzif, "An Integral Approach for Complete Migration from a Relational Database to MongoDB," *Journal of the Nigerian Society of Physical Sciences*, Vol. 5, pp. 1089. DOI: https://doi.org/10.46481/jnsps.2023.1089

[3] L. Rocha, F. Vale, E. Cirilo, D. Barbosa, and F. Mourao, "A framework for migrating relational datasets to NoSQL," *Procedia Computer Science*, Vol. 51, pp. 2593-2602, 2015. DOI: https://doi.org/10.1016/j.procs.2015.05.367

[4] C. Győrödi, R. Győrödi, G. Pecherle, and A. Olah, "A comparative study: MongoDB vs. MySQL," *13th international conference on engineering of modern electric systems (EMES) IEEE*, pp. 1-6, June 2015. DOI: https://doi.org/10.1109/EMES.2015.7158433

[5] O. Alotaibi and E. Pardede, "Transformation of schema from relational database (RDB) to NoSQL databases," *Data*, Vol. 4, No. 4, pp. 148, 2019. DOI: https://doi.org/10.3390/data4040148

[6] P. Atzeni, F. Bugiotti, L. Cabibbo, and R. Torlone, "Data modeling in the NoSQL world," *Computer Standards & Interfaces*, Vol. 67, 2020. DOI: https://doi.org/10.1016/j.csi.2016.10.003

[7] D. Coupal and K. W. Alger, "Building with Patterns: A Summary," *https://www.mongodb.com/blog/post/building-with-patterns-a-summary*, April 2019, Updated: Sep 2024.

[8] Korea Public Data Portal. *https://data.go.kr*

[9] M. Min, "A Data Design for Increasing the Usability of Subway Public Data," *International Journal of Internet, Broadcasting and Communication (IJIBC)*, Vol. 11, No. 4, pp. 18-25, 2019. DOI: http://dx.doi.org/10.7236/IJIBC.2019.11.4.18