Building With Patterns: The Attribute Pattern

Welcome back to the MongoDB Schema Design Patterns series. Last time we looked at the Polymorphic Pattern which covers situations when all documents in a collection are of similar, but not identical, structure. In this post, we'll take a look at the Attribute Pattern. The Attribute Pattern is particularly well suited when:

- We have big documents with many similar fields but there is a subset of fields that share common characteristics and we want to sort or query on that subset of fields, *or*
- The fields we need to sort on are only found in a small subset of documents, *or*
- Both of the above conditions are met within the documents.

For performance reasons, to optimize our search we'd likely need many indexes to account for all of the subsets. Creating all of these indexes could reduce performance. The Attribute Pattern provides a good solution for these cases.

THE ATTRIBUTE PATTERN

Let's think about a collection of movies. The documents will likely have similar fields involved across all of the documents: title, director, producer, cast, etc. Let's say we want to search on the release date. A challenge that we face when doing so, is *which* release date? Movies are often released on different dates in different countries.

```
{
    title: "Star Wars",
    director: "George Lucas",
    ...
    release_US: ISODate("1977-05-20T01:00:00+01:00"),
    release_France: ISODate("1977-10-19T01:00:00+01:00"),
    release_Italy: ISODate("1977-10-20T01:00:00+01:00"),
    release_UK: ISODate("1977-12-27T01:00:00+01:00"),
    ...
}
```

A search for a release date will require looking across many fields at once. In order to quickly do searches for release dates, we'd need several indexes on our movies collection:

```
{release_US: 1}
{release_France: 1}
{release_Italy: 1}
...
```

By using the Attribute Pattern, we can move this subset of information into an array and reduce the indexing needs. We turn this information into an array of key-value pairs:

```
{
    title: "Star Wars",
    director: "George Lucas",
    …
    releases: [
        {
        location: "USA",
        date: ISODate("1977-05-20T01:00:00+01:00")
```

```
        location: "France",
        date: ISODate("1977-10-19T01:00:00+01:00")
        },
        {
        location: "Italy",
        date: ISODate("1977-10-20T01:00:00+01:00")
        },
        {
        location: "UK",
        date: ISODate("1977-12-27T01:00:00+01:00")
        },
        …
    ],
    …
}
```

Indexing becomes much more manageable by creating one index on the elements in the array: `{ "releases.location": 1, "releases.date": 1}`

By using the Attribute Pattern we can add organization to our documents for common characteristics and account for rare/unpredictable fields. For example, a movie released in a new or small festival. Further, moving to a key/value convention allows for the use of non-deterministic naming and the easy addition of qualifiers. For example, if our data collection was on bottles of water, our attributes might look something like:

```
"specs": [
    { k: "volume", v: "500", u: "ml" },
    { k: "volume", v: "12", u: "ounces" }
]
```

Here we break the information out into keys and values, “k” and “v”, and add in a third field, “u” which allows for the units of measure to be stored separately.

SAMPLE USE CASE

The Attribute Pattern is well suited for schemas that have sets of fields that have the same value type, such as lists of dates. It also works well when working with characteristics of products. Some products, such as clothing, may have sizes that are expressed in small, medium, or large. Other products in the same collection may be expressed in volume. Yet others may be expressed in physical dimensions or weight.

A customer in the domain of asset management has recently deployed their solution using the Attribute Pattern. The customer uses the pattern to store all characteristics of a given asset that are seldom common across assets or were simply difficult to predict at design time. Relational models typically use a complicated design process to express the same idea in the form of user-defined fields.

While many of the fields in the product catalog are similar, such as name, vendor, manufacturer, country of origin, etc., the specifications, or attributes, of the item may differ. If your application and data access patterns rely on searching through many of these differences at once, the Attribute Pattern provides a good structure for the data.

CONCLUSION

The Attribute Pattern provides for the easier indexing the documents, targeting many similar fields per document. By moving this subset of data into a key-value sub-document, we can use non-deterministic field names, add additional qualifiers to the information, and more clearly state the relationship of the original field and value. When we use the Attribute Pattern, we need fewer indexes, our queries become simpler to write, and our queries become faster.

The next pattern we'll discuss is the Bucket Design Pattern.

If you have questions, please leave comments below.

This post was originally published on the MongoDB Blog.

 admin / February 7, 2019 / MongoDB / Attribute Pattern, schema design

One thought on “Building With Patterns: The Attribute Pattern”

 **Www.ncai.org**

March 8, 2019 at 1:41 pm

But wanna comment on few general things, The website style iis perfect, the content material is very superb
: D.