Model One-to-One Relationships with Embedded Documents

Embedded Document Pattern

```
// patron document
{
   _id: "joe",
   name: "Joe Bookreader"
}

// address document
{
   patron_id: "joe", // reference to patron document
   street: "123 Fake Street",
   city: "Faketon",
   state: "MA",
   zip: "12345"
}
```

```
{
   _id: "joe",
   name: "Joe Bookreader",
   address: {
               street: "123 Fake Street",
               city: "Faketon",
               state: "MA",
               zip: "12345"
             }
}
```

Subset Pattern

```
{
  "_id": 1,
  "title": "The Arrival of a Train",
  "year": 1896,
  "runtime": 1,
  "released": ISODate("01-25-1896"),
  "poster": "http://ia.media-imdb.com/images/M/MV5BMjEyNDk5MDYzOV5BMl5BanB
nXkFtZTgwNjIxMTEwMzE@._V1_SX300.jpg",
  "plot": "A group of people are standing in a straight line along the pla
tform of a railway station, waiting for a train, which is seen coming at s
ome distance. When the train stops at the platform, ...",
  "fullplot": "A group of people are standing in a straight line along the
platform of a railway station, waiting for a train, which is seen coming a
t some distance. When the train stops at the platform, the line dissolves.
The doors of the railway-cars open, and people on the platform help passen
gers to get off.",
  "lastupdated": ISODate("2015-08-15T10:06:53"),
  "type": "movie",
  "directors": [ "Auguste Lumière", "Louis Lumière" ],
  "imdb": {
    "rating": 7.3,
    "votes": 5043,
    "id": 12
  },
  "countries": [ "France" ],
  "genres": [ "Documentary", "Short" ],
  "tomatoes": {
    "viewer": {
      "rating": 3.7,
      "numReviews": 59
    },
    "lastUpdated": ISODate("2020-01-09T00:02:53")
  }
}
```

Currently, the `movie` collection contains several fields that the application does not need to show a simple overview of a movie, such as `fullplot` and rating information. Instead of storing all of the movie data in a single collection, you can split the collection into two collections:

The `movie` collection contains basic information on a movie. This is the data that the application loads by default:

```
// movie collection

{
  "_id": 1,
  "title": "The Arrival of a Train",
  "year": 1896,
  "runtime": 1,
  "released": ISODate("1896-01-25"),
  "type": "movie",
  "directors": [ "Auguste Lumière", "Louis Lumière" ],
  "countries": [ "France" ],
  "genres": [ "Documentary", "Short" ],
}
```

The `movie_details` collection contains additional, less frequently-accessed data for each movie:

```
// movie_details collection

{
  "_id": 156,
  "movie_id": 1, // reference to the movie collection
  "poster": "http://ia.media-imdb.com/images/M/MV5BMjEyNDk5MDYzOV5BMl5BanB
nXkFtZTgwNjIxMTEwMzE@._V1_SX300.jpg",
  "plot": "A group of people are standing in a straight line along the platform of a railway station, waiting for a train, which is seen coming at some distance. When the train stops at the platform, ...",
  "fullplot": "A group of people are standing in a straight line along the platform of a railway station, waiting for a train, which is seen coming at some distance. When the train stops at the platform, the line dissolves. The doors of the railway-cars open, and people on the platform help passengers to get off.",
  "lastupdated": ISODate("2015-08-15T10:06:53"),
  "imdb": {
    "rating": 7.3,
    "votes": 5043,
    "id": 12
  },
  "tomatoes": {
    "viewer": {
      "rating": 3.7,
      "numReviews": 59
    },
    "lastUpdated": ISODate("2020-01-29T00:02:53")
  }
}
```

Trade-Offs of the Subset Pattern

Using smaller documents containing more frequently-accessed data reduces the overall size of the working set. These smaller documents result in improved read performance and make more memory available for the application.

However, it is important to understand your application and the way it loads data. If you split your data into multiple collections improperly, your application will often need to make multiple trips to the database and rely on `JOIN` operations to retrieve all of the data that it needs.

In addition, splitting your data into many small collections may increase required database maintenance, as it may become difficult to track what data is stored in which collection.

Model One-to-Many Relationships with Embedded Documents

Embedded Document Pattern

```
// patron document
{
   _id: "joe",
   name: "Joe Bookreader"
}

// address documents
{
   patron_id: "joe", // reference to patron document
   street: "123 Fake Street",
   city: "Faketon",
   state: "MA",
   zip: "12345"
}

{
   patron_id: "joe",
   street: "1 Some Other Street",
   city: "Boston",
   state: "MA",
   zip: "12345"
}
```

```
{
   "_id": "joe",
   "name": "Joe Bookreader",
   "addresses": [
                 {
                   "street": "123 Fake Street",
                   "city": "Faketon",
                   "state": "MA",
                   "zip": "12345"
                 },
                 {
                   "street": "1 Some Other Street",
                   "city": "Boston",
                   "state": "MA",
                   "zip": "12345"
                 }
               ]
 }
```

Subset Pattern

```
{
  "_id": 1,
  "name": "Super Widget",
  "description": "This is the most useful item in your toolbox.",
  "price": { "value": NumberDecimal("119.99"), "currency": "USD" },
  "reviews": [
    {
      "review_id": 786,
      "review_author": "Kristina",
      "review_text": "This is indeed an amazing widget.",
      "published_date": ISODate("2019-02-18")
    },
    {
      "review_id": 785,
      "review_author": "Trina",
      "review_text": "Nice product. Slow shipping.",
      "published_date": ISODate("2019-02-17")
    },
    ...
    {
      "review_id": 1,
      "review_author": "Hans",
      "review_text": "Meh, it's okay.",
      "published_date": ISODate("2017-12-06")
    }
  ]
}
```

The reviews are sorted in reverse chronological order. When a user visits a product page, the application loads the ten most recent reviews.

Instead of storing all of the reviews with the product, you can split the collection into two collections:

The `product` collection stores information on each product, including the product's ten most recent reviews:

```
{
  "_id": 1,
  "name": "Super Widget",
  "description": "This is the most useful item in your toolbox.",
  "price": { "value": NumberDecimal("119.99"), "currency": "USD" },
  "reviews": [
    {
      "review_id": 786,
      "review_author": "Kristina",
      "review_text": "This is indeed an amazing widget.",
      "published_date": ISODate("2019-02-18")
    }
    ...
    {
      "review_id": 776,
      "review_author": "Pablo",
      "review_text": "Amazing!",
      "published_date": ISODate("2019-02-16")
    }
  ]
}
```

The `review` collection stores all reviews. Each review contains a reference to the product for which it was written.

```
{
  "review_id": 786,
  "product_id": 1,
  "review_author": "Kristina",
  "review_text": "This is indeed an amazing widget.",
  "published_date": ISODate("2019-02-18")
}
{
  "review_id": 785,
  "product_id": 1,
  "review_author": "Trina",
  "review_text": "Nice product. Slow shipping.",
  "published_date": ISODate("2019-02-17")
}
...
{
  "review_id": 1,
  "product_id": 1,
  "review_author": "Hans",
  "review_text": "Meh, it's okay.",
  "published_date": ISODate("2017-12-06")
}
```

By storing the ten most recent reviews in the `product` collection, only the required subset of the overall data is returned in the call to the `product` collection. If a user wants to see additional reviews, the application makes a call to the `review` collection.

TIP
When considering where to split your data, the most frequently-accessed portion of the data should go in the collection that the application loads first. In this example, the schema is split at ten reviews because that is the number of reviews visible in the application by default.

Trade-Offs of the Subset Pattern

Using smaller documents containing more frequently-accessed data reduces the overall size of the working set. These smaller documents result in improved read performance for the data that the application accesses most frequently.

However, the subset pattern results in data duplication. In the example, reviews are maintained in both the `product` collection and the `reviews` collection. Extra steps must be taken to ensure that the reviews are consistent between each collection. For example, when a customer edits their review, the application may need to make two write operations: one to update the `product` collection and one to update the `reviews` collection.

You must also implement logic in your application to ensure that the reviews in the `product` collection are always the ten most recent reviews for that product.

Other Sample Use Cases

In addition to product reviews, the subset pattern can also be a good fit to store:

- Comments on a blog post, when you only want to show the most recent or highest-rated comments by default.
- Cast members in a movie, when you only want to show cast members with the largest roles by default.

Model One-to-Many Relationships with Document References

```
{
   title: "MongoDB: The Definitive Guide",
   author: [ "Kristina Chodorow", "Mike Dirolf" ],
   published_date: ISODate("2010-09-24"),
   pages: 216,
   language: "English",
   publisher: {
              name: "O'Reilly Media",
              founded: 1980,
              location: "CA"
            }
}

{
   title: "50 Tips and Tricks for MongoDB Developer",
   author: "Kristina Chodorow",
   published_date: ISODate("2011-05-06"),
   pages: 68,
   language: "English",
   publisher: {
              name: "O'Reilly Media",
              founded: 1980,
              location: "CA"
            }
}
```

To avoid repetition of the publisher data, use *references* and keep the publisher information in a separate collection from the book collection.

When using references, the growth of the relationships determine where to store the reference. If the number of books per publisher is small with limited growth, storing the book reference inside the publisher document may sometimes be useful. Otherwise, if the number of books per publisher is unbounded, this data model would lead to mutable, growing arrays, as in the following example:

```
{
   name: "O'Reilly Media",
   founded: 1980,
   location: "CA",
   books: [123456789, 234567890, ...]
}

{
    _id: 123456789,
    title: "MongoDB: The Definitive Guide",
    author: [ "Kristina Chodorow", "Mike Dirolf" ],
    published_date: ISODate("2010-09-24"),
    pages: 216,
    language: "English"
}

{
   _id: 234567890,
   title: "50 Tips and Tricks for MongoDB Developer",
   author: "Kristina Chodorow",
   published_date: ISODate("2011-05-06"),
   pages: 68,
   language: "English"
}
```

To avoid mutable, growing arrays, store the publisher reference inside the book document:

```
{
   _id: "oreilly",
   name: "O'Reilly Media",
   founded: 1980,
   location: "CA"
}

{
   _id: 123456789,
   title: "MongoDB: The Definitive Guide",
   author: [ "Kristina Chodorow", "Mike Dirolf" ],
   published_date: ISODate("2010-09-24"),
   pages: 216,
   language: "English",
   publisher_id: "oreilly"
}

{
   _id: 234567890,
   title: "50 Tips and Tricks for MongoDB Developer",
   author: "Kristina Chodorow",
   published_date: ISODate("2011-05-06"),
   pages: 68,
   language: "English",
   publisher_id: "oreilly"
}
```

References

https://docs.mongodb.com/manual/applications/data-models-relationships/