A MongoDB Document Definition for a Bicycle Ride Journal

This document discusses a possible data structure for a Javascript Object Notation ("JSON") record which can then be added to a MongoDB database collection. We will consider the need to store particular details of one bicycle rider's individual bicycle rides over a period of months. The rider has joined a contest event. The event takes place over a period of months. The event has many competing riders. Riders are organized into teams. Each team has a unique team number. During the contest period, a rider potentially could do thousands of bicycle rides. Each bicycle rider can be awarded points for each ride that he or she does during the contest period. The rider's points also aggregate to the rider's team. In order to know how many points a rider has earned over time, it is necessary to record each individual ride.

We will call riders "members" to be consistent with the "team" concept.

For each member, we would like to know:

- The team that person is assigned to.
- The member's name.
- An array containing a list of that member's rides.

Each row of the member's rides provides additional information:

- ✓ The day number of the contest, starting from 1 which means "day 1 of the contest".
- ✓ The calendar date and time of the member's ride.
- ✓ The starting location of the member's ride, in longitude, latitude coordinate format.
- ✓ The length of that ride, in miles, as a decimal value with 2 digits of precision. Example: 7.50.
- ✓ The number of points awarded to the rider for that ride as a decimal value with 2 digits of precision. Example: 7.50.

Let us see if we can organize this information in a possible JSON document. JSON documents consist of key-value pairs, and are able to represent arrays. JSON is easy to understand, or at least more comprehensible than XML. However, the MongoDB version 3.4 implementation supports more data types than standard JSON formats. MongoDB recognizes a superset of formats it terms "Extended JSON". When we discuss a JSON document in this article, we really refer to the formats recognized by MongoDB "Extended JSON".

Consider this possible representation of one single bicycle ride in a JSON document:

```
{
        "team": "18",
        member: "William Smith",
        "rides": [
                        { "day" : 136, "date" : new Date("2017-04-01T07:30:00"), "start_loc": [39.0019871,-76.8749491],
                                "end_loc": [38.975784,-76.9058972], "miles": NumberDecimal("5.00"), "points":
                                    NumberDecimal("5.00") }
                ]
}
```

In a flat file named "test_add_smith".
A data representation of one ride
by member William Smith,
belonging to Team 18,

showing that on contest day number 136, which is April 1, 2017, at 7:30:00 in the morning, Smith rode 5 miles from Greenbelt to Lanham and was awarded 5 points.

The field "rides" is an array, and it can hold any number of array elements up to the size limit for one MongoDB document. This member can have any number of rides in the same date, since the start time of each new ride will be different. So for example, Smith's 07:30 morning ride could be followed by a second ride on day 136 starting at time 17:00.

The input date string shown results in an output ISO8601 format date and includes the local time of the ride. The database engine converts localtime to UTC (Universal Time Coordinate).

start_loc and end_loc are both location coordinates expressed as longitude, latitude values.

Building A MongoDB Collection -- The input record is in a flat file which can be considered a script. It essentially executes the insert() method on a new object within the mongo shell. This yields an output document which is stored in a MongoDB "collection". The MongoDB version used here is "MongoDB Community Edition", version 3.4. The document is stored in a collection named "ride_journal1". The *mongo* command line client produces this representation of the document in response to a query:

```
> db.ride_journal1.findOne()
{
        "_id" : ObjectId("588277564814c35c2fba04b7"),
        "team" : "18",
        "member" : "William Smith",
        "rides" : [
                {
                        "day" : 136,
                        "date" : ISODate("2017-04-01T11:30:00Z"),
                        "start_loc" : [
                                39.0019871,
                                -76.8749491
                        ],
                        "end_loc" : [
                                38.975784,
                                -76.9058972
                        ],
                        "miles" : NumberDecimal("5.00"),
                        "points" : NumberDecimal("5.00")
                }
        ]
}
```

If we add a second ride, for the same day but at a later time, Smith's list of rides looks like this:

```
> db.ride_journal1.findOne()
{
	"_id" : ObjectId("588277564814c35c2fba04b7"),
	"team" : "18",
	"member" : "William Smith",
	"rides" : [
		{
			"day" : 136,
			"date" : ISODate("2017-04-01T11:30:00Z"),
			"start_loc" : [
				39.0019871,
				-76.8749491
			],
			"end_loc" : [
				38.975784,
				-76.9058972
			],
			"miles" : NumberDecimal("5.00"),
			"points" : NumberDecimal("5.00")
		},
		{
			"day" : 136,
			"date" : ISODate("2017-04-01T21:00:00Z"),
			"start_loc" : [
				38.97584,
				-76.9058972
			],
			"end_loc" : [
				39.0019871,
				-76.8749491
			],
			"miles" : NumberDecimal("5.00"),
			"points" : NumberDecimal("5.00")
		}
```

```
        ]
}
```

We can add a second rider to this collection, for the same team:

```
> db.ride_journal1.find().pretty()
{
        "_id" : ObjectId("588277564814c35c2fba04b7"),
        "team" : "18",
        "member" : "William Smith",
        "rides" : [
                {
                        "day" : 136,
                        "date" : ISODate("2017-04-01T11:30:00Z"),
                        "start_loc" : [
                                39.0019871,
                                -76.8749491
                        ],
                        "end_loc" : [
                                38.975784,
                                -76.9058972
                        ],
                        "miles" : NumberDecimal("5.00"),
                        "points" : NumberDecimal("5.00")
                },
                {
                        "day" : 136,
                        "date" : ISODate("2017-04-01T21:00:00Z"),
                        "start_loc" : [
                                38.97584,
                                -76.9058972
                        ],
                        "end_loc" : [
                                39.0019871,
                                -76.8749491
                        ],
                        "miles" : NumberDecimal("5.00"),
                        "points" : NumberDecimal("5.00")
                }
        ]
}
{
        "_id" : ObjectId("58828902ee45f15a1f1054cf"),
        "team" : "18",
        "member" : "Greta Garbuckle",
        "rides" : [
                {
                        "day" : 13,
                        "date" : ISODate("2017-01-13T12:30:00Z"),
                        "start_loc" : [
                                39.0019871,
                                -76.8749491
                        ],
                        "end_loc" : [
                                38.975784,
                                -76.9058972
                        ],
                        "miles" : NumberDecimal("7.00"),
                        "points" : NumberDecimal("7.00")
                }
        ]
}
```

So now the collection has 2 riders: Smith and Garbuckle. Smith has 2 rides, Garbuckle 1. Notice that within the list of rides, the earliest ride for each day number shows up first. The sort order could be changed to show the latest rides for each day first.

Let's add one more rider for a new team:

```
{
	"_id" : ObjectId("588295f90af5335fca58651e"),
	"team" : "19",
	"member" : "Milroy Fizzie",
	"rides" : [
		{
			"day" : 1,
			"date" : ISODate("2017-01-13T12:30:00Z"),
			"start_loc" : [
				39.0019871,
				-76.8749491
			],
			"end_loc" : [
				38.994095,
				-76.875786
			],
			"miles" : NumberDecimal("1.20"),
			"points" : NumberDecimal("1.00")
		}
	]
}
```

At this point, we have enough documents in the MongoDB collection to develop a web application with.

Important Database Administration Notes for MongoDB version 3.4

MongoDB Feature Compatibility

The developer who initially set up a MongoDB ride journal collection on his server ran into a number of challenges. The database platform had recently been upgraded from MongoDB version 3.2 to version 3.4.1. One of the new features of MongoDB 3.4 is the addition of a "NumberDecimal" data format. It is thought that using this data format to record rider miles and points will be extremely useful because of the mathematical precision it offers.

When upgrading from version 3.2 to 3.4.1, the mongod server does not automatically implement new features in 3.4.1 which are not backward-compatible to version 3.2. So when creating new databases and collections under this scenario, they implement 3.2 feature compatibility only. If you try to add a field to a collection in the NumberDecimal format, the mongo shell will generate an error and tell you that feature compatibility is set to version 3.2. To address this issue:

In the mongo shell, issue

```
db.adminCommand( { setFeatureCompatibilityVersion: "3.4" } )
```

Back up the collection and it's indexes.

Drop the collection and it's indexes.

Re-import the collection and rebuild the indexes. db.collection.getIndexes() should be queried to determine that the indexes are version 2 indexes.

In the mongo shell, insert documents which contain fields in NumberDecimal format. Inserts of such documents will now execute successfully.

mongoimport Utility Does Not Support NumberDecimal format

As of version 3.4.1, the mongoimport utility does not support the NumberDecimal format. So you can't mass-import a collection with mongoimport if it contains fields in NumberDecimal. To address this issue, use the mongo shell and db.collection.insert(). It is also likely that the Node.js and Java drivers provided by MongoDB natively support NumberDecimal.