Converting your data from MySQL to MongoDB

03 SEPTEMBER 2013

As mentioned in my previous post I'm working together with a friend on a project that was originally written for LAMP and is now converted to the MEAN stack.

The very first exercise that had to be carried out was the conversion of data from MySQL to MongoDB - this is what I'm going to discuss on this post.

Note that some of the code snippets in this article use Italian variable names. We are currently working on renaming those to English as well to show better examples.

When converting the data bear in mind that there is a massive difference in how one would normalise the data in MySQL and in MongoDB - tables no longer exist but collections do. These collections can contain nested data; data which used to be in tables before and instead of making JOIN statements, JavaScript array/object operations will have to be utilised.

At a high level, these are the tables that currently make up the database in the project:

- **Services**: Holds information about Public Transportation services (buses, trams, underground, etc) such as Line number, Last stop, type of transport etc
- **Schedule**: lists the departure schedules for the services by each stop
- **Stops**: holds all the possible stops and stores various information, such as the GPS position of a stop
- **Route**: Stores the route of a particular transportation service, with ordinal stop numbers (1st to Nth)

There are various joins that are needed here in order to get the right information out from the system. For example, if I want to display the departure times for a certain service I need to join the Services and Schedule table together.

Here's how a "simple" query looks like at the moment to get a list of public transport vehicles that stop (well, go through really) at a given stop:

```
SELECT
    DISTINCT
    Linea as 'Line',
    Capolinea as 'Direction',
    Tipo as 'Vehicle'
FROM
    percorsi as routes,
    linee as lines
WHERE
    lines.Percorso = routes.Percorso
AND
    Palina = 73565
```

(Just for the super curious people, bus stop with

ID 73565 is Via Collatina, Rome)

The query may look short but it takes a reasonable time to execute and once it did, the dataset needs to be massaged into a JSON object.

I hear you ask: So how is the same information stored in MongoDB? Well, let me show you - instead of the 4 tables, the following collections exist:
Stops collection

```
{
    "stopID": "ID",
    "name": "name",
    "GPS": {
        "type": "Point",
        "coordinates": [
            "lat",
            "long"
        ]
    }
}
```

Note The `GPS` field has a `2dsphere` index instead of the `2d` index. (This also means that it only runs on MongoDB version 2.4 or newer)

Line collection

```
{
    "Route": "routeID",
    "Line": "line",
    "Description": "Description",
    "Vehicle": "vehicle",
    "Stops": [
        "1st",
        "Nth"
    ],
    "Departures": [
        "date",
        "date",
        "date"
    ]
}
```

As you can see the whole structure is nicer and of course faster. So the query that was displayed

above as an example now looks like this:

```
var lts = db.Lines.find({$and:
[{Stops: 73565}]}); //Lines through
stop
for (i=0;i<lts.length();i++){
    var direction =
db.Stops.find({stopID:
lts[i].Stops[lts[i].Stops.length-
1]},{name:1})[0].name;
    print({'Line': lts[i].Line,
'Vehicle': lts[i].Vehicle,
'Direction': direction});
}
```

It may seem complex at first sight but in fact it is a rather simple way of extracting the same information.

Finally, let's talk about the actual conversion. Unfortunately there is no straight way converting data from MySQL to MongoDB, but there are a few techniques that you can use.

If your database is simple and you only have individual tables that you want to import (meaning: there are no JOINs between the tables) you can very simply export your data form MySQL to a CSV file and then import that CSV file with `mongoimport` to MongoDB.

To import your data from MySQL to CSV you could use a query similar to this:

```
SELECT * FROM books INTO OUTFILE
'/tmp/books.csv'
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
```

If your MySQL server is on a remote machine and you want to copy the `books.csv` file onto your server, have a look at the article that I posted about automating such processes using `scp` and `expect`.

Once you have your CSV file, run this simple command (modify as needed):

```
mongoimport --db users --collection
books --type csv --file
/tmp/books.csv
```

The above works well for single tables or sets of tables with no relationship between them, however, as also the name suggests, relational databases usually have, well, yeah, relations. My advice is the following for converting such tables:

Write a script in your preferred language (PHP, Python, Perl, etc...) that queries these tables using the JOIN statements and returns the dataset in a JSON format, which you can save to disc and later import it using `mongoimport`. It's very hard to give an example as the datasets can be diverse but at a conceptual level, see the following:

Let's assume that I have a Books table with book titles, page numbers and an Author ID. I also have an Author table with a name, an author ID and a publisher ID and finally I have a Publisher table with a Publisher ID and a name. The

relationship is the following:
One Publisher can have multiple Authors. One Author can only belong to one Publisher. One Author can have multiple books. As per the explanation in the previous few paragraphs of this post our MongoDB could look like this:
Book collection

```
{
  "title": title,
  "pages": pagenumbers,
  "author": authorID
}
```

Author collection

```
{
  "name": author-name,
  "ID": author-id
}
```

Publisher collection

```
{
  "name": publisher-name,
  "authors": [{author-id1, author-
id2}]
}
```

Probably it's even possible to put all this information into one big collection called 'Books' as opposed to creating 3 separate ones:

```
{
 "title": title,
  "pages": pagenumbers,
  "author": [{
              "id": authod-id,
              "name": author-name
            }],
  "publisher": [
                 "name": publisher-name
            ]
}
```

But for the sake of this article we are assuming that there are 3 collections.

It is then the script's job to return the right data in the right format from MySQL and save that. Or even, take this to the next level, the very same script could generate a JSON file and immediately call `mongoimport`. (If you choose this approach, I would recommend that you do a few test runs to make sure that your JSON data structure is indeed correct before importing it into MongoDB.) Here's an example structure:

```
$my_file = 'data.json';
$con=mysqli_connect("localhost","root","
if (mysqli_connect_errno()) {
  echo "Failed to connect to MySQL: "
. mysqli_connect_error();
}

$result = mysqli_query($con,"SELECT *
FROM Books b JOIN Authors a ON
b.author_id = a.id");

$rows = mysqli_fetch_array($result);
print json_encode($rows);
```

```
mysqli_close($con);

$handle = fopen($my_file, 'w') or
die('Cannot open file:  '.$my_file);
fwrite($handle, $rows);
```

The example above is very crude and should only give you an idea how this could be achieved. You probably need to massage your JSON data further.

As a summary - if you are converting your database make sure that you do proper de-normalisation, you can probably store 2 or 3 different table's information in one collection without a problem in a form of an array or a nested object. If you are looking for automated conversions you'd have to code that on your own and format the CSV or JSON file to suit your needs.

In the next post I will write a bit about the geospatial features of MongoDB which are supercool! Stay tuned & thanks for reading.

Tamas Piros

Experienced software engineer, blogger, author and teacher & preacher of super-heroic web technologies.

http://fullstacktraining.com/

Share this post