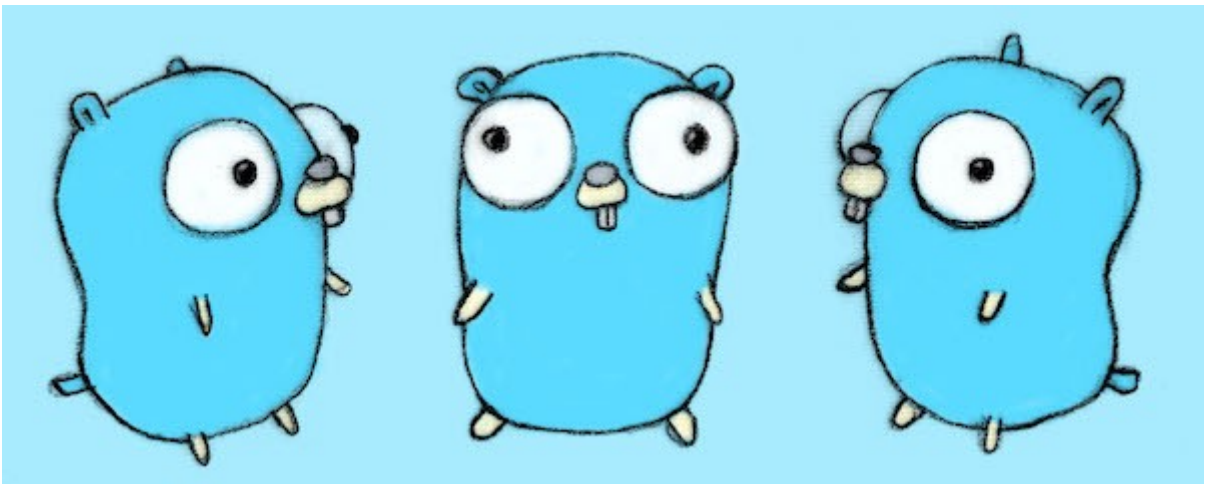
SQL Data Mapping With Protocol Buffers.

Jack J
Nov 6, 2019 · 5 min read



Background

With the rise of Big Data came increasing complexities of its management. Engineers often spend plenty of time trying to make their SQL data accessible in a convenient format for their apps. With complex schema and advanced app features come painstaking tasks of mapping the data onto well-defined and structured responses. This is especially true for large apps with sophisticated features and numerous microservices.

I would like to introduce a tool resolving this management issue. protoc-gen-map is a SQL data mapping framework implementing protocol buffers — Google's language-neutral data structuring method. Aside from defining proto messages and SQL queries, developers do not need to write any data retrieval or mapping code.

Why Proto Buffers?

Protocol buffers, or simply protobuf, is a mechanism for serializing structured data. Protobuf's key advantage is its platform and language independence.

A single definition of a proto message can be shared across multiple services. The structure of the data remains consistent regardless of language or platform.

Approach

Let's assume that we work for an online publishing company. We manage and host a complex database of blogs for our users.
One day, we receive a request to create a service which would allow our app to retrieve basic information about a blog. Assume that for now, we are using a very simple schema:

To create a microservice allowing such functionality, we can define proto messages with some remote procedure calls (gRPC). Code snippet below defines services, request and response.

```
service BlogService {
    rpc SelectBlog (BlogRequest) returns (BlogResponse) {}
    rpc SelectBlogs (BlogRequest) returns (stream BlogResponse) {}
}
message BlogRequest {
    uint32 id = 1;
    string author_id  = 2;
}
message BlogResponse {
    uint32 id = 1;
    string title  = 2;
    string author_id  = 3;
}
```

To retrieve necessary data, we can write SQL statement. However, by using Go's text/template syntax we can modify our query based on what our user requests. Below we are defining queries for two of our services.

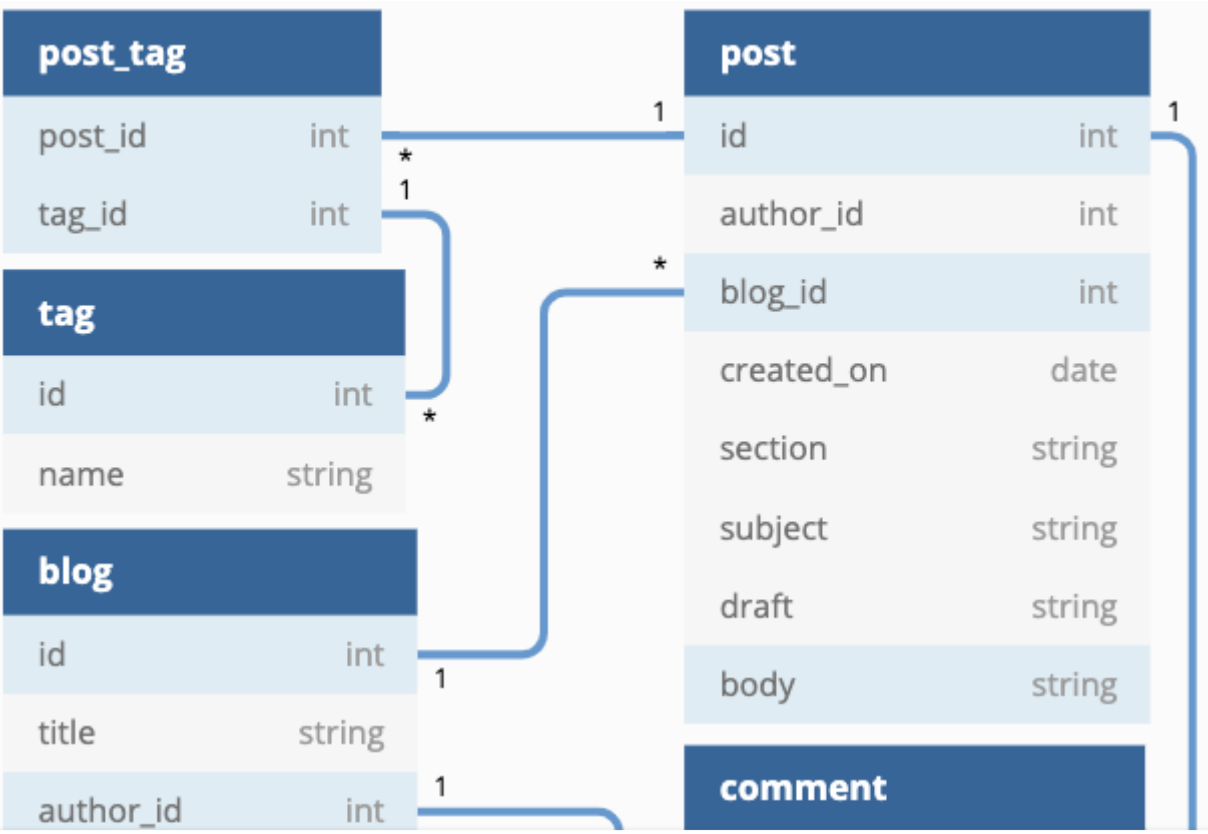
```
    blog where id = {{ .Id }} limit 1
{{ end }}

{{ define "SelectBlogs" }}
    select id, title, author_id from blog
    where author_id = {{ .AuthorId }}
{{ end }}
```

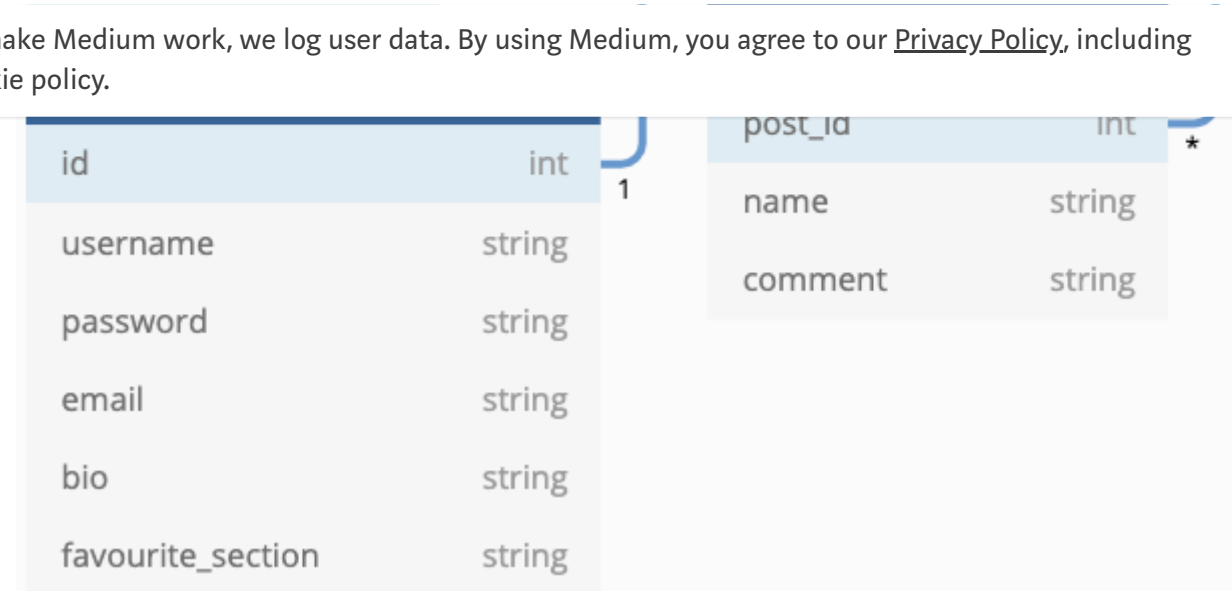
Now we would need to execute this SQL template based on an incoming request, map the retrieved SQL data to the response, and return the response. protoc-gen-map will take care of all that. It generates code that will execute the template and more importantly map the data onto the proto message. All developers have to do is define the query and the message.

When things get complex

The example above is quite trivial. However, things are rarely as simple. SQL statements often tend to be complex and lengthy. protoc-gen-map helps manage this issue.

Let's say we receive a task to create a service, which would allow our app to get a much more detailed information about a blog. Assume that we are now using a more complex schema, one with has-one and has-many relationships.





Our query would be as follows.

```
{{ define "SelectDetailedBlog" }}
select
       id                 as  blog_id,
       title              as  blog_title,
       A.id               as  author_id,
       A.username         as  author_username,
       A.password         as  author_password,
       A.email            as  author_email,
       A.bio              as  author_bio,
       A.favourite_section as  author_favourite_section,
       P.id               as  post_id,
       P.blog_id          as  post_blog_id,
       P.author_id        as  post_author_id,
       P.created_on       as  post_created_on,
       P.section          as  post_section,
       P.subject          as  post_subject,
       P.draft            as  draft,
       P.body             as  post_body,
       C.id               as  comment_id,
       C.post_id          as  comment_post_id,
       C.comment          as  comment_text,
       T.id               as  tag_id,
       T.name             as  tag_name
from blog
       left outer join author A    on  blog.author_id = A.id
       left outer join post P      on  blog.id = P.blog_id
       left outer join comment C   on  P.id = C.post_id
       left outer join post_tag PT on  PT.post_id = P.id
       left outer join tag T       on  PT.tag_id = T.id
where blog.id =  {{ .Id }}
{{ end }}
```

defined below. There is no need to write any data retrieval or mapping code.

```
service BlogQueryService {
  rpc SelectDetailedBlog (BlogRequest) returns
(DetailedBlogResponse) {}
}

message BlogRequest {
  uint32 id = 1;
}

message DetailedBlogResponse {
  uint32 blog_id = 1;
  string blog_title = 2;
  Author author = 3;
  repeated Post posts = 4;
}
message Author {
  uint32 author_id = 1;
  string author_username = 2;
  string author_password = 3;
  string author_email = 4;
  string author_bio = 5;
  Section author_favourite_section = 6;
}

message Post {
  uint32 post_id = 1;
  uint32 post_blog_id = 2;
  uint32 post_author_id = 3;
  google.protobuf.Timestamp post_created_on = 4;
  Section post_section = 5;
  string post_subject = 6;
  string draft = 7;
  string post_body = 8;
  repeated Comment comments = 9;
  repeated Tag tags = 10;
}

message Comment {
  uint32 comment_id = 1;
  uint32 comment_post_id = 2;
  string comment_name = 3;
  string comment_text = 4;
}

message Tag {
  uint32 tag_id = 1;
  string tag_name = 2;
}
```

```
    woodworking = 2;
    snowboarding = 3;
  }
```

Any client requesting a detailed blog information would receive a message with properly mapped data from the rows retrieved by the query.

The major advantage of using this workflow is that no matter what language out client is using, the response guarantees a consistent, well-known structure.

In addition, all the work put into mapping our SQL data has been taken care of by simply defining our messages.

Summary

protoc-gen-map allows developers to create database microservices quickly and efficiently. The developers not need to worry about the complexities and tediousness of retrieving and mapping their data.

For detailed information on protoc-gen-map framework visit
https://github.com/jackskj/protoc-gen-map

For sample case, head over to examples.
Comments and feature requests greatly appreciated.

Go Protobuf Data Programming Sql