Extracted from:

Effective Go Recipes

Fast Solutions to Common Tasks

The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

Effective Go Recipes

Fast Solutions to Common Tasks

Miki Tebeka

edited by Margaret Eldridge

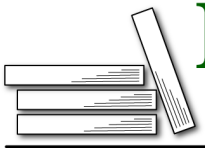
Effective Go Recipes

Fast Solutions to Common Tasks

Miki Tebeka

The Pragmatic Bookshelf

Raleigh, North Carolina



For our complete catalog of hands-on, practical, and Pragmatic content for software developers, please visit *https://pragprog.com*.

For sales, volume licensing, and support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

ISBN-13: 978-1-68050-846-8
Encoded using the finest acid-free high-entropy binary digits.
Book version: B1.0—August 16, 2023

Using Composite Keys in Maps

Task

As a hobby, you work on some algorithmic trading code. After a gaming night with to your friend Bob, who works in a bank you decide you need a fast in-memory database for the stock information. You're going to use this database to train stock trading algorithms that will make you rich!

Stock Trading

This is by no way a recommendation to trade the stock market. I've worked for several years in high-frequency trading companies, and the volume of information we had gave us a huge advantage over casual traders.

Automated trading is very risky, the Knight Capital Group lost $440 million dollars in 45 minutes due to a bug. Do you think you can stand such losses?

Solution

The information for a stock is the opening, low, high and closing price of the day (known as OHLC).

types/stock_db/stock_db.go

```
// StockInfo is information about stock at a given date
type StockInfo struct {
    Date   time.Time
    Symbol string
    Open   float64
    High   float64
    Low    float64
    Close  float64
}
```

You want fast access for stock information at a given date, you decide to use a map and have a struct as a composite key. The key is:

types/stock_db/stock_db.go

```
type key struct {
    year   int
```

```
    month  time.Month
    day    int
    symbol string
}
```

Then you write a InfoDB struct that will hold the map:

types/stock_db/stock_db.go

```
type InfoDB struct {
    m map[key]StockInfo
}
```

InfoDB provides a Get function that will return stock information for given stock at a given date:

types/stock_db/stock_db.go

```
// Get return information for stock at date. If information is not found the
// second return value will be false
func (i *InfoDB) Get(symbol string, date time.Time) (StockInfo, bool) {
    k := key{date.Year(), date.Month(), date.Day(), symbol}
    info, ok := i.m[k]
    return info, ok
}
```

Discussion

Most of the time, maps keys will be a simple type, but in some cases, you'll need a composite key. A composite key is a key that's consists of two or more attributes.

I've seen people that instead of using composite keys, create a string that represents the composite key. For example, for our database case you might construct a string key with values such as "AAPL:20200302". This approach is error prone; you are inventing a new serialization format, and might create the same string for two different keys.

For example, assume that a user is defined by its login name and a user ID. If you create the string "joe42" it might be user joe with user ID 42 or the user joe4 with the id 2.

We're taking advantage of the fact that a Go struct is comparable if all of its fields are comparable. You can use these structs as map key, without figuring out a bullet-proof way of creating strings to represent keys.

Some types, such as slices, are not comparable and can't be used in composite keys. If you need to use a slice as a map key, or a part of a map key - convert it to a string first. However, since a user might change the original slice, its better not to use slices as map keys at all.

In general, prefer types that can't be changed (immutable) such as string, int, rune as map keys.

You decided not to use time.Time as part of the compound key since comparing with time.Time can be tricky. See Recipe 35, Parsing Time Strings, on page ? for more information.