GO on the toilet

episode #12

Select is not only SQL thing

The **select** statement in go gives the ability to listen on many channels at the same time. Let's have a look.

```
func main() {
    ch := make(chan int)
    ch2 := make(chan int)
    go func() { ch <- 1 }() // sending on separate goroutine to avoid deadlock

    select {
    case v := <-ch:
       fmt.Printf("received %d on channel 1", v)
    case v := <-ch2:
       fmt.Printf("received %d on channel 2", v)
    }
}
```

This piece of code executes only the first **case** statement because only the first channel has something that can be received. In scenarios where at the execution time, **select** statement receives values on multiple channels simultaneously, the **case** statement will be selected randomly. The example above may not look complex, neither impressive, so let's have a look at something more useful. The **select** enables us to build a simple timeout mechanism.

```
func main() {
    ch := make(chan int)
    go func() {
       time.Sleep(5 * time.Second)
       ch <- 1
    }()

    select {
    case v := <-ch:
       fmt.Printf("received %d on channel", v)
    case <-time.After(1 * time.Second):
       fmt.Printf("timed out")
    }
}
```

In order to understand this example you need to know that **time.After** returns a channel, which yields after time specified in the parameter. Since we send to the channel **ch** after 5 seconds, first case will never execute, because after 1 second, second case will run.