

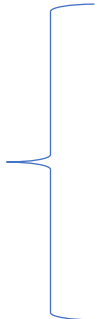
Introducción a GORM

Para instalar GORM:


```
go get -u gorm.io/gorm
```

Para instalar el driver correspondiente:

```
go get -u gorm.io/driver/*
```



- sqlite
- mysql
- postgres
- sqlserver
- clickhouse



Compatible
con SQL

Para importar en GO:

```
import (  
    "gorm.io/driver/mysql"  
    "gorm.io/gorm"  
)
```

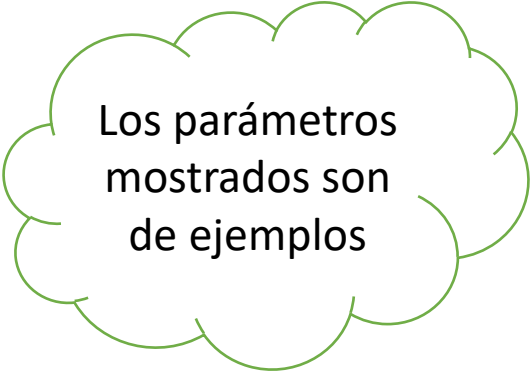
Para conectar

En Postgres

```
dsn := "host=localhost user=gorm password=gorm dbname=gorm port=9920 sslmode=disable TimeZone=Asia/Shanghai"
db, err := gorm.Open(postgres.Open(dsn), &gorm.Config{})
```

En MySQL

```
func main() {
// refer https://github.com/go-sql-driver/mysql#dsn-data-source-name for details
dsn := "user:pass@tcp(127.0.0.1:3306)/dbname?charset=utf8mb4&parseTime=True&loc=Local"
db, err := gorm.Open(mysql.Open(dsn), &gorm.Config{})
}
```



Los parámetros mostrados son de ejemplos

Conectar a una DB

- Usando MySQL o Posgres crear un usuario y luego crear la DB. Por ejemplo, usando postgres desde psql:
 - create user gorm with encrypted password 'gorm';
 - create database gorm_db;
 - grant all privileges on database gorm_db to gorm;
 - \c gorm_db
 - grant all privileges on all tables in schema public to gorm;
 - grant all privileges on all sequences in schema public to gorm;
 - GRANT ALL ON schema public TO gorm;

Recuerda crear un proyecto e inicializar el módulo antes .

Crear el modelo

```
type Student struct {  
    gorm.Model  
    Cod      uint8  
    Name     string  
    Last_Name string  
}
```

Dentro de la función main pondremos lo siguiente....

Recuerda crear un proyecto e inicializar el módulo antes .

Conectar a una DB

Con lo anterior el código de conexión queda:

```
dsn := "host=localhost user=gorm password=gorm dbname=gorm_db  
port=5432 sslmode=disable TimeZone=America/Bogota"  
db, err := gorm.Open(postgres.Open(dsn), &gorm.Config{})  
  
if err != nil {  
    panic("failed to connect database")  
}  
fmt.Println("Conectado exitosamente a la DB...")
```

```
db.AutoMigrate(&Student{})
db.Create(&Student{Cod: 190, Name: "Pepito",
Last_Name: "Doe"})
```

Migra todos los modelos dado como en este caso Student
Crea/Inserta un nuevo registro en la DB

	123 id	created_at	updated_at	deleted_at	123 cod	ABC name	ABC last_name
1	1	02:40:35.261 -0500	02:40:35.261 -0500	[NULL]	190	Pepito	Doe
2	2	02:40:46.038 -0500	02:40:46.038 -0500	[NULL]	190	Pepito	Doe

La migración no limpia la DB

Si se quisiera “limpiar” la tabla no se podría de la manera general

```
db.Delete(&Student{}).Error // gorm.ErrMissingWhereClause
```

Aquí 3 opciones

Analizar:
¿en qué se diferencian?

1

```
db.Where("1 = 1").Delete(&Student{})
// DELETE FROM `students` WHERE 1=1
```

2

```
db.Exec("DELETE FROM students")
// DELETE FROM students
```

3

```
db.Session(&gorm.Session{AllowGlobalUpdate: true}).Delete(&Student{})
// DELETE FROM students
```

Convenciones de las etiquetas de campos

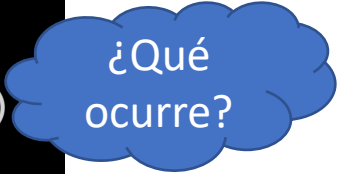
```
type Student struct {  
    Cod      uint8 `gorm:"primaryKey"`  
    Name     string `gorm:"not null"`  
    SecondName string  
    LastName string `gorm:"not null; default:'NN'"`  
    Visited_at time.Time  
}
```

Ver algunos ejemplos de etiquetas:

https://gorm.io/es_ES/docs/models.html#Fields-Tags

Creamos tres registros

```
db.Create(&Student{Cod: 190, Name: "Pepito"})  
db.Create(&Student{Cod: 190, Name: "Anita", Visited_at: time.Now()})  
db.Create(&Student{Cod: 192, Name: "Paquita", Visited_at: time.Now()})
```



¿Qué
ocurre?

consultamos

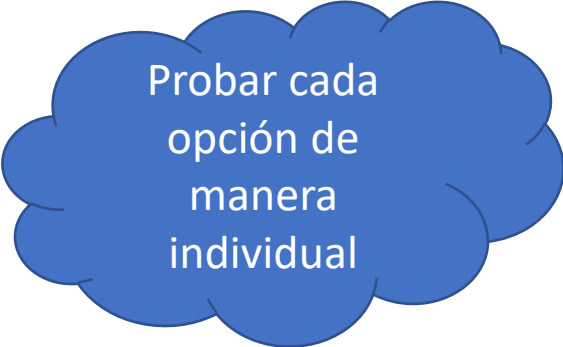
```
var student, student2, student3 Student
db.First(&student, 1) // find product with integer primary key
db.First(&student2, "cod = ?", 192) // find product with code 192
db.First(&student3, "cod = ?", 191) // find product with code 191
fmt.Println(student)
fmt.Println(student2)
fmt.Println(student3)
```

actualizamos

```
// Update - update student's Name to Marcos
db.Model(&student).Update("name", "Marcos")
// Update - update multiple fields
db.Model(&student2).Updates(Student{Cod: 200, Name: "Margarita", LastName: "Rosa"})
db.Model(&student3).Updates(map[string]interface{}{"Cod": 202, "Name": "Francisca",
"LastName": "Rosas"})
```

Y por último, eliminamos...

```
// Delete - delete student
db.Delete(&student)
db.Delete(&student2, 3)
db.Delete(&Student{}, 1)
students := []Student{student, student2, student3}
db.Delete(&students, []int{1,3})
db.Delete(&Student{}, []int{1,3})
```



Probar cada
opción de
manera
individual