DIGITALHOUSE

# Implementando nuestro CRUD

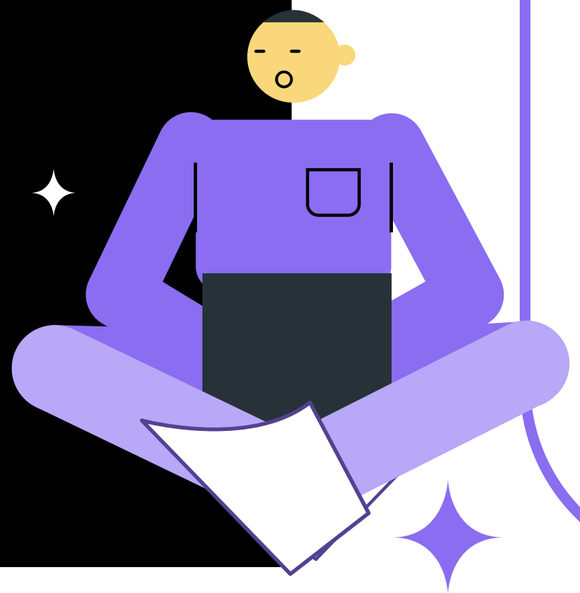
Certified Tech Developer
The Ultimate Degree

# Índice



# 01 GetOne

La función **GetOne** es similar a una función de búsqueda. Permite a las personas usuarias buscar y recuperar registros específicos en la tabla y leer sus valores. Los usuarios pueden encontrar los registros deseados utilizando palabras clave o filtrando los datos según criterios personalizados.



# GetOne

```
func (r *repository) GetOne(id int) models.User {
    var user models.User
    db := db.StorageDB
    rows, err := db.Query("select * from users where id = ?", id)
    if err != nil {
        log.Println(err)
        return user
    }
    for rows.Next() {
        if err := rows.Scan(&user.ID, &user.Name, &user.Telephone,
&user.Email, &user.Address); err != nil {
            log.Println(err.Error())
            return user
        }
    }
    return user
}
```

# GetOne

La implementación de **GetOne** se fundamenta en tres funciones principales:

01

**db.Query**

Devuelve las filas (objeto **rows**) encontradas para la consulta dada y los parámetros correspondientes. Devuelve también un error en los casos que corresponda.

02

**rows.Next**

Permite recorrer los resultados obtenidos.

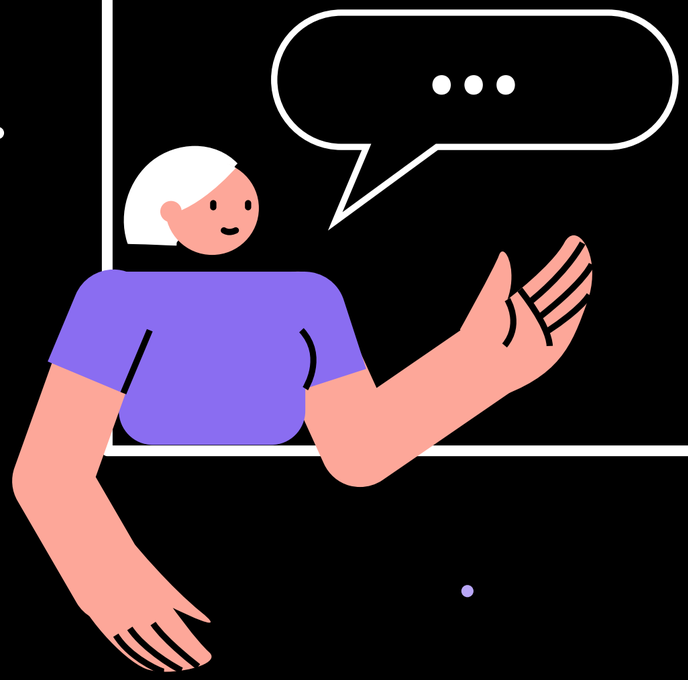
03

**rows.Scan**

Lee los campos de la fila obtenida y los copia o almacena en las posiciones de memoria de las variables que se indican por parámetros.

# 02 Store

La función **Store** permite a los usuarios crear un nuevo registro en la base de datos.
Recordemos que un registro es una fila y que las columnas se denominan atributos. Un usuario puede crear una nueva fila y completarla con datos que correspondan a cada atributo.



# Store

```go
func (r *repository) Store(user models.User) (models.User, error) {
    db := db.StorageDB // se inicializa la base
    stmt, err := db.Prepare("INSERT INTO users(name, email, telephone, address) VALUES( ?, ?, ?, ?)") // se prepara el SQL
    if err != nil {
        log.Fatal(err)
    }
    defer stmt.Close() // se cierra la sentencia al terminar. Si quedan abiertas, se generan consumos de memoria
    var result sql.Result
    result, err = stmt.Exec(user.Name, user.Email, user.Telephone, user.Address) // retorna un sql.Result y un error
    if err != nil {
        return models.User{}, err
    }
    insertedId, _ := result.LastInsertId() // del sql.Result devuelto en la ejecución obtenemos el ID insertado
    user.ID = int(insertedId)

    return user, nil
}
```

# Store

Los dos métodos elementales usados para interactuar con la base, en este caso, son:

01

**db.Prepare**

Devuelve el statement, o sentencia, listo para incorporar valores y ejecutar.

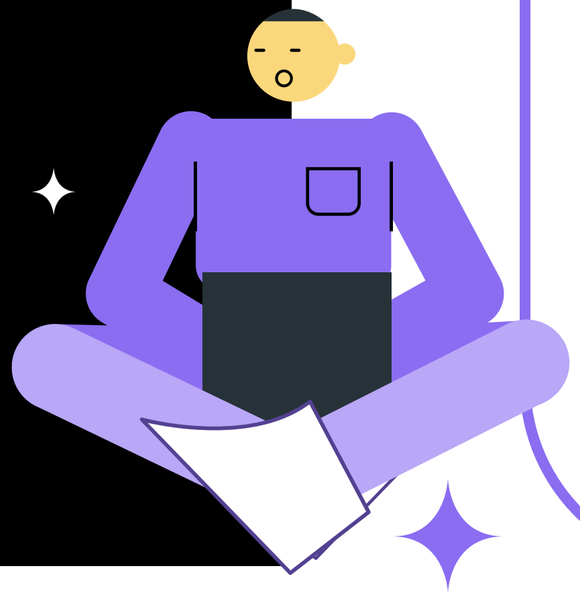
02

**stmt.Exec**

Este método pertenece al objeto statement y ejecuta la query preparada con los parámetros que recibe de entrada.

# 03 Update

La función **Update** se utiliza para modificar registros existentes que existen en la base de datos. Para cambiar completamente un registro, es posible que los usuarios tengan que modificar la información en varios campos.



# Update

```
func (r *repository) Update(user models.User) (models.User, error) {
    db := db.StorageDB       // se inicializa la base
    stmt, err := db.Prepare("UPDATE users SET name = ?, email = ?, telephone =
?, address = ? WHERE id = ?") // se prepara la sentencia SQL a ejecutar
    if err != nil {
        log.Fatal(err)
    }
    defer stmt.Close()     // se cierra la sentencia al terminar. Si quedan
abiertas, se generan consumos de memoria
    _, err = stmt.Exec(user.Name, user.Email, user.Telephone, user.Address,
user.ID)
    if err != nil {
        return models.User{}, err
    }
    return user, nil

}
```

# Update

De forma análoga al método **Store**, **Update** utiliza los métodos:

01

**db.Prepare**

Devuelve el statement, o sentencia, listo para incorporar valores y ejecutar.

02

**stmt.Exec**

Este método pertenece al objeto statement y ejecuta la query preparada con los parámetros que recibe de entrada

# 04 Delete

La función **Delete** permite a los usuarios eliminar registros de una base de datos que ya no se necesitan. Algunas aplicaciones de bases de datos relacionales pueden permitir a los usuarios realizar una eliminación definitiva o una eliminación temporal.

# Delete

```go
func (r *repository) Delete(user models.User)  error {
    db := db.StorageDB       // se inicializa la base
    stmt := "DELETE FROM users WHERE id = ?"

    _, err := db.Exec(stmt, user.ID)
    if err != nil {
        return err
    }
    return nil

}
```

# Delete

La implementación de **Delete**, al igual que el **Store** y el **Update**, hace uso del **Exec**:

01

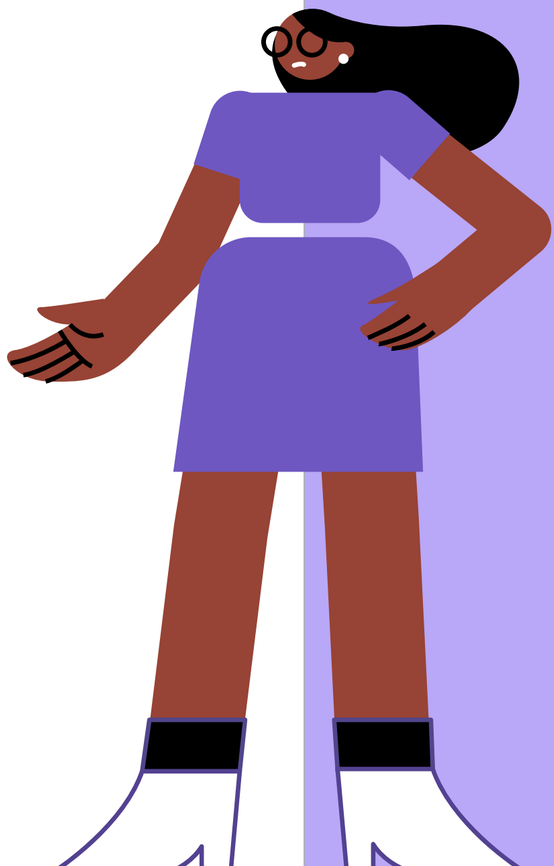
**stmt.Exec**

Este método pertenece al objeto statement y ejecuta la query preparada con los parámetros que recibe de entrada.

# Conclusiones

En esta clase hicimos un repaso del package **database/sql**, para poder conectar nuestra app a una base de datos.

También vimos cómo implementar los métodos **getOne**, **Store**, **Update** y **Delete** a nuestro repository para poder comunicarnos e interactuar con nuestra base de datos.



DIGITALHOUSE

# ¡Muchas gracias!