

Davide Imola • Ludovico Russo

Level UP your RDBMS Productivity in GO



Davide Imola

Software Engineer @ RedCarbon
CM @ Schrödinger Hat



Ludovico Russo

VP of Engineering @ RedCarbon



▶ A little Disclaimer about this talk

All the things inside this talk are highly opinionated and they are not a standard. They are simply things we have discovered by working with DataBases and Go.

Examples are in PostgreSQL, but all the stuff works for other SQL DBs.

No DB have been harmed for the purpose of this talk, but a couple was truncated 🤔

Davide Imola • Ludovico Russo

The Actual Status

What might go(lang) wrong this time?



▶ Handling the DB Code

Let's be honest!
Who loves writing again and
again all this code?

No, Copilot (or any generative
AI 🤖) is not the answer.

```
main.go

func ListUsers(ctx context.Context) ([]User, error) {
    rows, err := db.Query(ctx, "SELECT * FROM users")
    if err != nil {
        return nil, err
    }
    defer rows.Close()
    var users []User
    for rows.Next() {
        var u User
        if err := rows.Scan(&u.ID, &u.Name); err != nil {
            return nil, err
        }
        users = append(users, u)
    }
    if err := rows.Err(); err != nil {
        return nil, err
    }
    return users, nil
}
```

▶ Finding hidden errors in SQL

```
main.go

func ListUsers(ctx context.Context) ([]User, error) {
    rows, err := db.Query(ctx, "SELECT * FROM upserts")
    if err != nil {
        return nil, err
    }
    defer rows.Close()
    var users []User
    for rows.Next() {
        var u User
        if err := rows.Scan(&u.ID, &u.Name); err != nil {
            return nil, err
        }
        users = append(users, u)
    }
    if err := rows.Err(); err != nil {
        return nil, err
    }
    return users, nil
}
```

▶ SQL Injection

```
func GetUser(ctx context.Context, id string) (*User, error) {  
    row := db.QueryRow(ctx, fmt.Sprintf("SELECT * FROM users WHERE id = %s", id))  
  
    var u User  
  
    err := row.Scan(&u.ID, &u.Name)  
    return u, err  
}
```

5 OR 1=1

▶ Code and DB out of sync

ALTER TABLE users ADD
column email varchar;


error: sql: expected 3 destination arguments
in Scan, not 2

```
main.go

func ListUsers(ctx context.Context) ([]User, error) {
    rows, err := db.Query(ctx, "SELECT * FROM users")
    if err != nil {
        return nil, err
    }
    defer rows.Close()
    var users []User
    for rows.Next() {
        var u User
        if err := rows.Scan(&u.ID, &u.Name); err != nil {
            return nil, err
        }
        users = append(users, u)
    }
    if err := rows.Err(); err != nil {
        return nil, err
    }
    return users, nil
}
```

- ▶ Manual type sync and possible downtimes

The **problem** is always **between**
the **chair** and the **monitor**



▶ Automated Testing (Why mocking your DB will make you cry 😭)

You don't want to bring up a DB? Ok let's mock it.

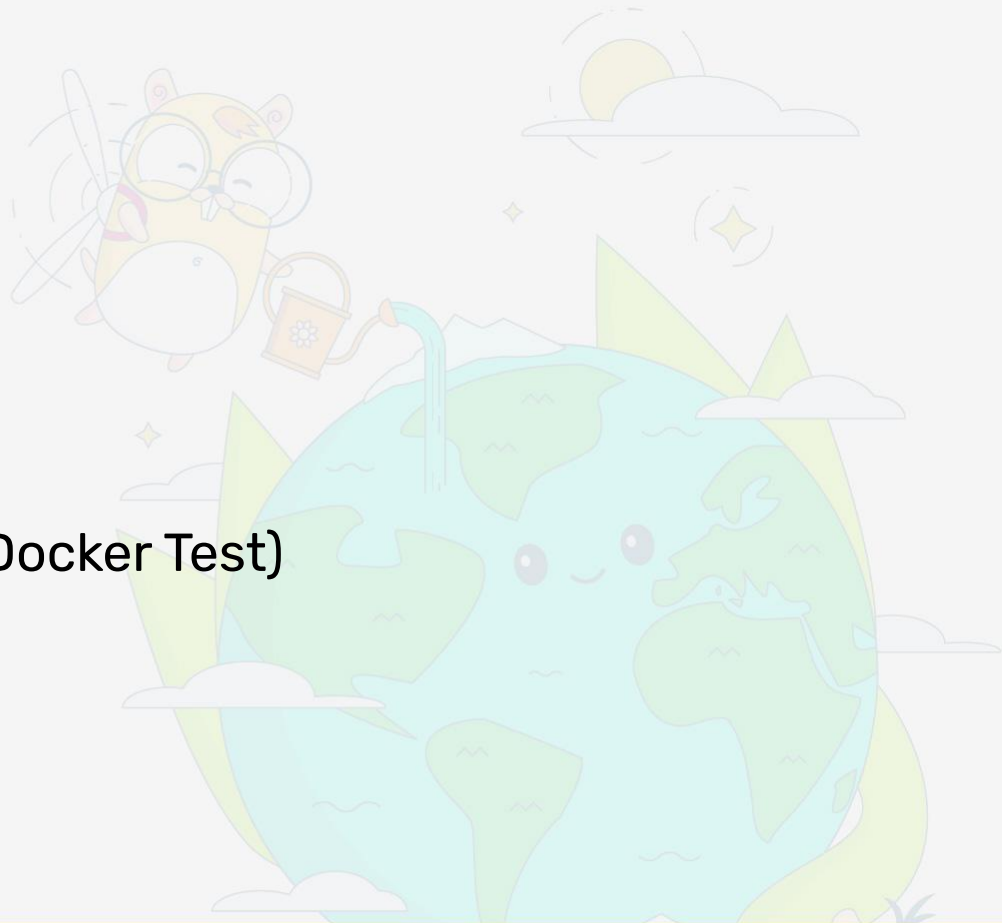
But, if you decide to mock `db.Query` or all the `ListUsers` function you can't see the bug inside this Query.

```
main.go

func ListUsers(ctx context.Context) ([]User, error) {
    rows, err := db.Query(ctx, "SELECT * FROM users")
    if err != nil {
        return nil, err
    }
    defer rows.Close()
    var users []User
    for rows.Next() {
        var u User
        if err := rows.Scan(&u.ID, &u.Name); err != nil {
            return nil, err
        }
        users = append(users, u)
    }
    if err := rows.Err(); err != nil {
        return nil, err
    }
    return users, nil
}
```

▶ So... what we are covering today...

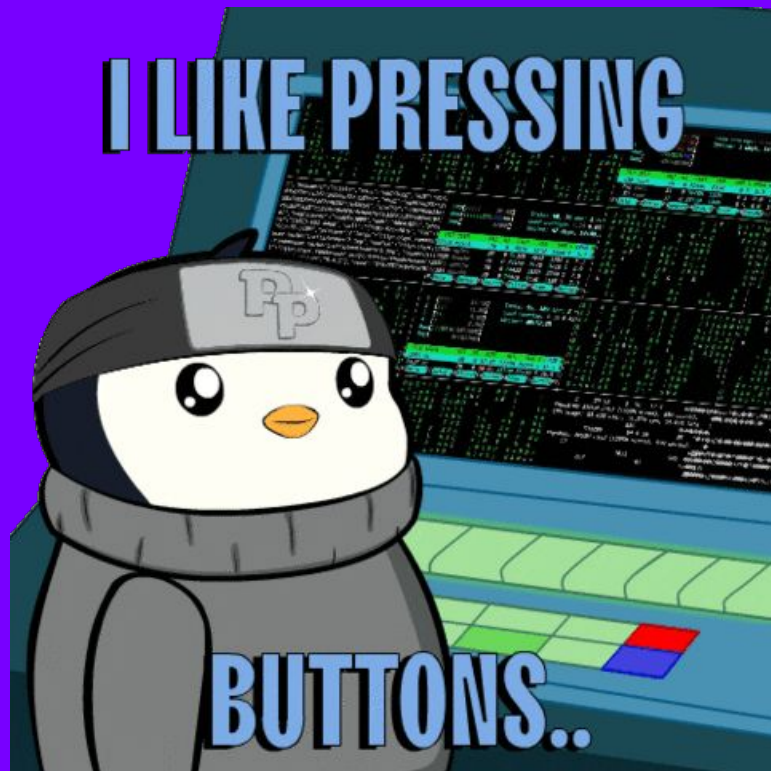
- SQL-first paradigm
- Migrations
- Test Containers (or Docker Test)



Davide Imola • Ludovico Russo

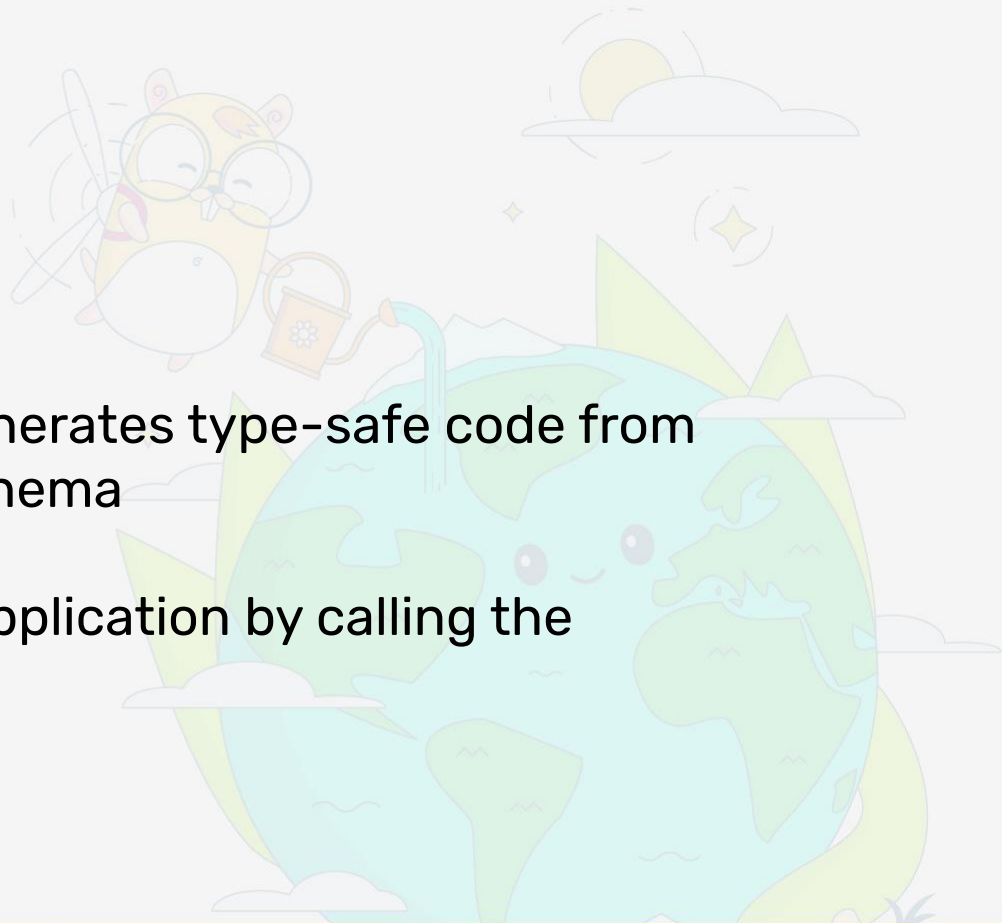
SQL-first

Compile SQL to type-safe code



▶ SQL-first

1. Write a SQL Query
2. Run a tool which generates type-safe code from your queries and schema
3. Enjoy writing your application by calling the generated methods!



▶ Other approaches

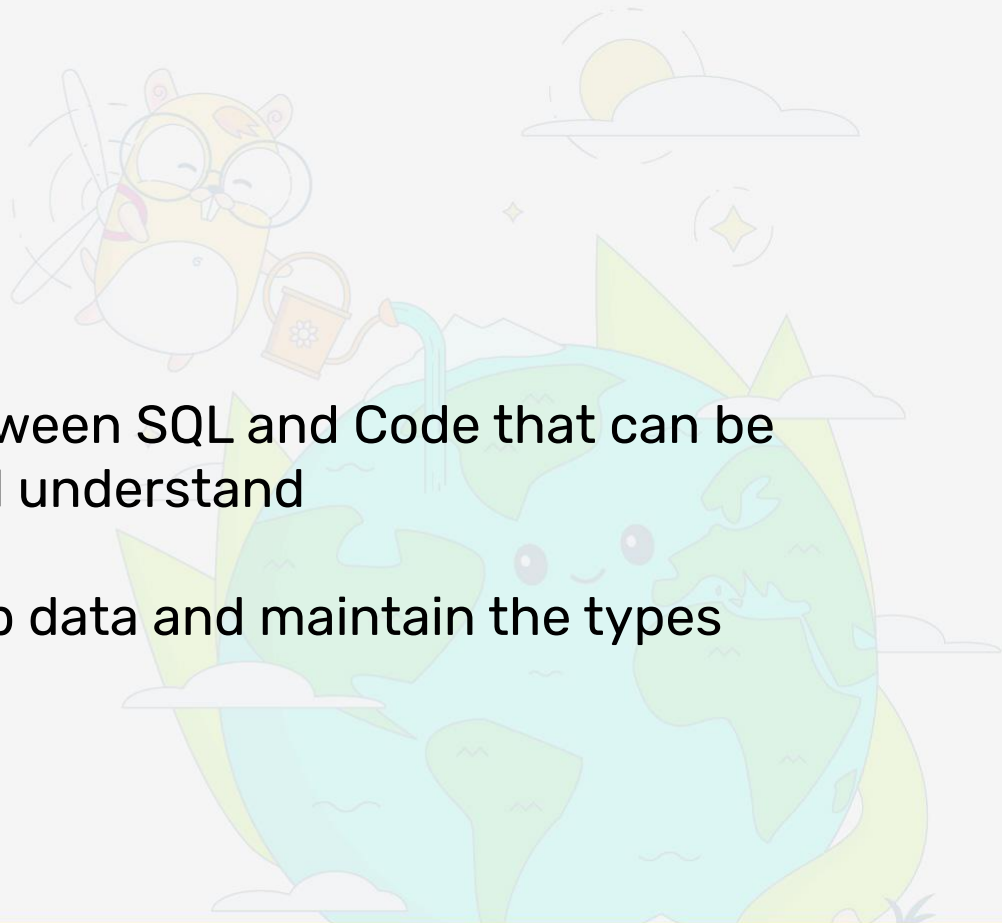
- **ORM (Object-Relational Mapping)**
Programming technique that enables seamless interaction between object-oriented code and relational databases
- **Query Builders**
Tools or libraries that provide a programmatic and often fluent interface for generating db queries

► Why we don't like ORM in Go?

- You don't have the chance to use SQL
- We don't like so much the APIs built for Go
- You can't use all the DBs features (e.g., JSONB filtering in Postgres)

► Why we don't like Query Builders?

- Not type-safe
- Mixed approach between SQL and Code that can be complex to read and understand
- You still need to map data and maintain the types manually



▶ SQL-first vs ORM vs Query Builders

	SQL First	ORM	Query Builder
Type-safe Code	✓	✓	✗
Learn SQL	✓	✗	✓
Protect from SQL Injection	✓	✓	✗
All DB features	✓	✗	✓
Clean Api	✓	✗ (in Go)	✗
Code Generation 🤖	✓	✗	✗
We like it 🤔	✓ ✓ ✓ ✓ ✓ ✓ ✓	✗	✗

▶ **Don't hate ORM or Query Builders! Use it as fallback!**

SQL-first may be an interesting approach. But it has major downsides when it comes to complex queries.

In this case using an ORM/Query Builders as a fallback for some specific cases might work very well!

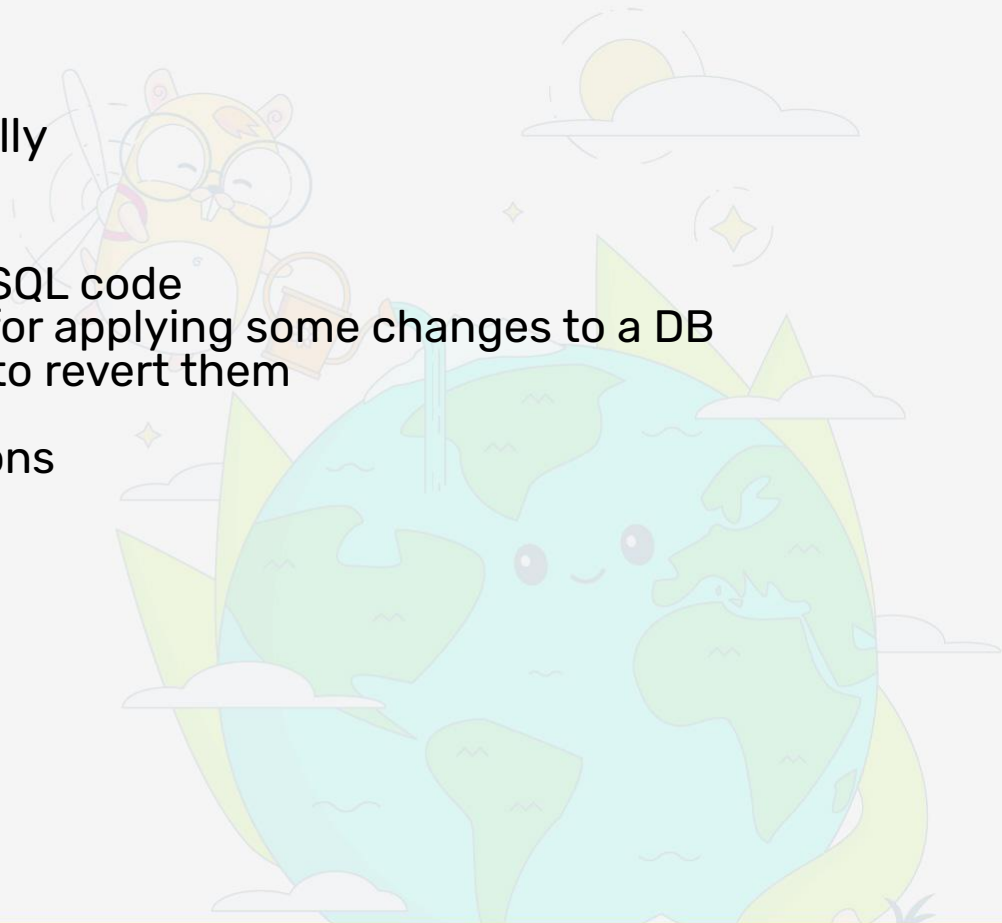
Migrations

dbmate up your DB!



▶ UP and DOWN your DB automatically

- Write your migration in SQL code
 - UP: queries to run for applying some changes to a DB
 - DOWN: if you want to revert them
- Commit in your migrations
- Run it
 - Manually
 - In a Job
 - In your CD pipeline
- That's all

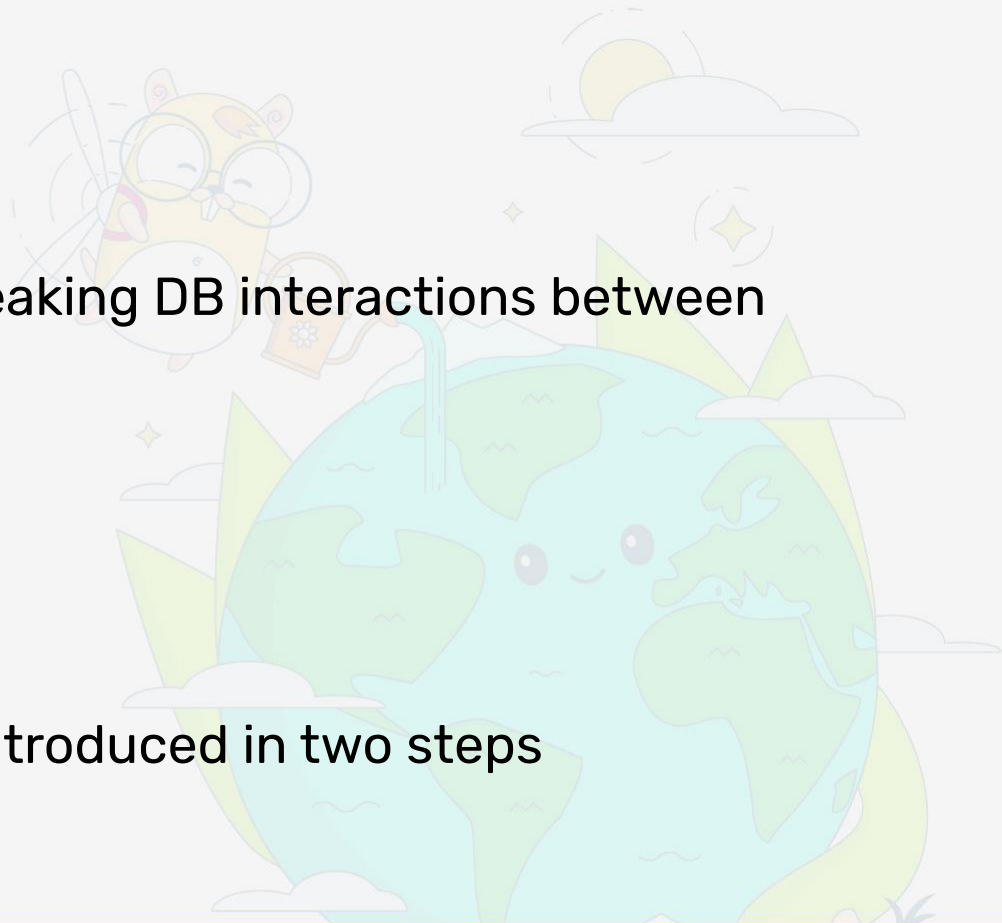


▶ Evolutionary Database Design

A technique to avoid breaking DB interactions between releases.

Rules:

- Automated testing
- Small migrations
- Breaking changes introduced in two steps

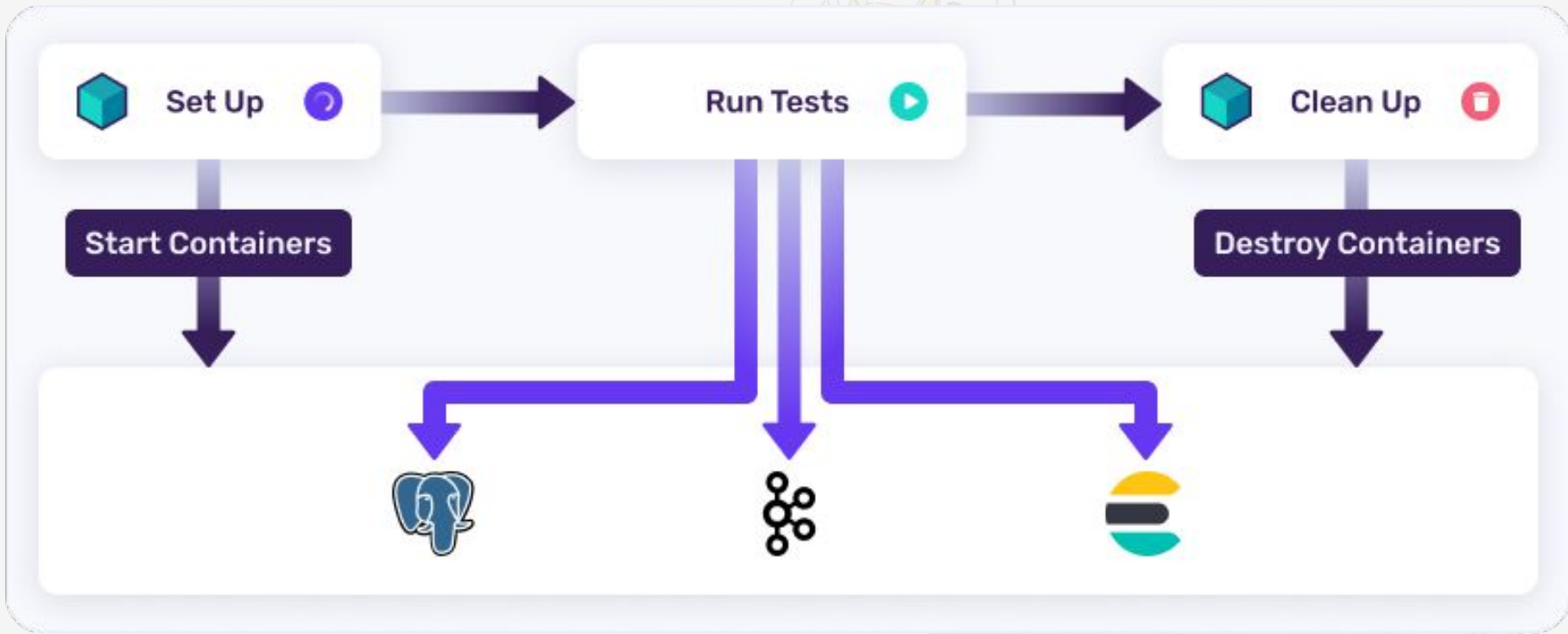


Test Containers (or Docker Test)

Dockerize your Integration tests



▶ Containerize your Infrastructure into Testing



Davide Imola • Ludovico Russo

Demo Time

May the Demo God be with US ♥



Davide Imola • Ludovico Russo

The New Status

What might we have fixed?



▶ Handling the DB Code

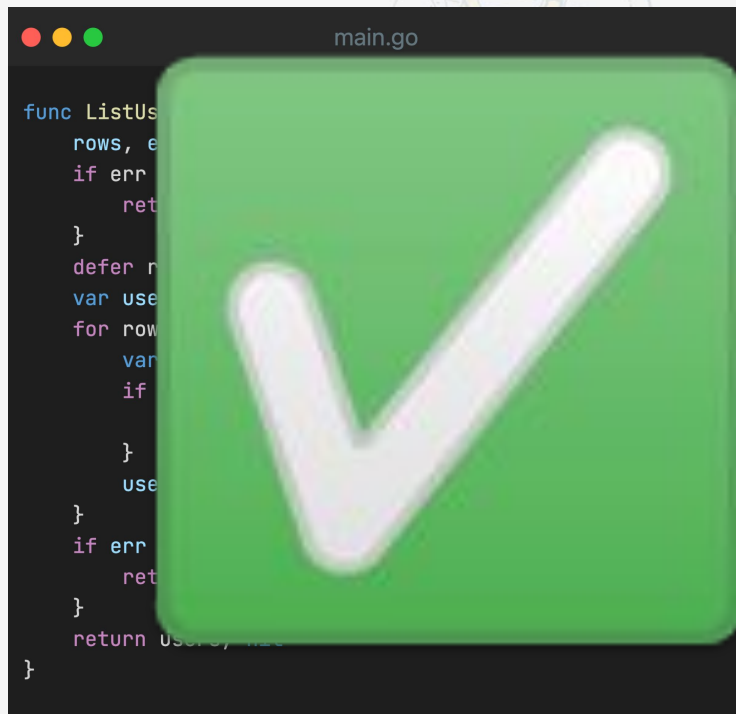
Let's be honest!
Who loves writing again
again all this code?

No, Copilot (or any general
AI 🤖) is not the answer



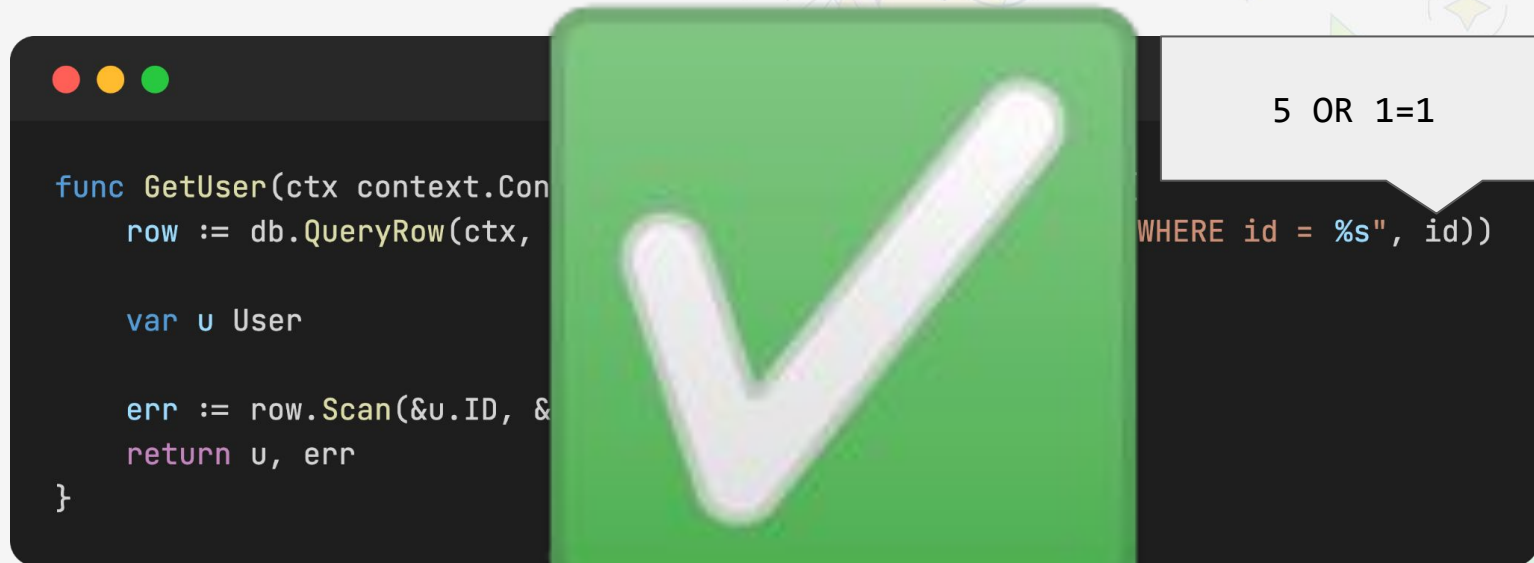
```
main.go
context.Context) ([]User, error) {
    db.Query(ctx, "SELECT * FROM users")
    {
        err
    }
    use()
    er
    ) {
        rows.Scan(&U.ID, &U.Name); err != nil {
            nil, err
        }
        pend(users, u)
    }
    .Err(); err != nil {
        , err
    }
    nil
}
```

▶ Errors in SQL syntax



```
func ListUsers(
    rows, err error) ([]User, error) {
    if err != nil {
        return nil, err
    }
    defer r.Close()
    var users []User
    for row := range rows {
        var user User
        if err := r.Scan(&user); err != nil {
            return nil, err
        }
        users = append(users, user)
    }
    if err := r.Err(); err != nil {
        return nil, err
    }
    return users, nil
}
```

▶ SQL Injection



▶ Code and DB out of sync

ALTER TABLE users ADD
column email varchar;

error: sql: expected 3 destination
in Scan, not 2



```
main.go
context.Context) ([]User, error) {
    b.Query(ctx, "SELECT * FROM users")
    {
        , err
    }
    se()
    er
    ) {
        rows.Scan(&u.ID, &u.Name); err != nil {
            nil, err
        }
        pend(users, u)
    }
    .Err(); err != nil {
        , err
    }
    nil
}
```

- ▶ Manual type sync and possible downtimes

The problem between the chain monitor



▶ Automated Testing (Why mocking your DB will make you cry 😭)

You don't want to bring DB? Ok let's mock it.

But, if you decide to mock `db.Query` or all the `ListUsers` function you can't see the result inside this Query.



```
main.go
context.Context) ([]User, error) {
    db.Query(ctx, "SELECT * FROM users")

    err

    func()
    r
    {
        rows.Scan(&u.ID, &u.Name); err != nil {
            nil, err

        }
        return users, u
    }

    Err(); err != nil {
        err

        nil
    }
}
```


Open Source Day

7-8 March 2024 @ Florence



Ludovico Russo

in/ludusrusso

@ludusrusso

Davide Imola

in/davideimola

@davideimola

@davideimola@fosstdon.org

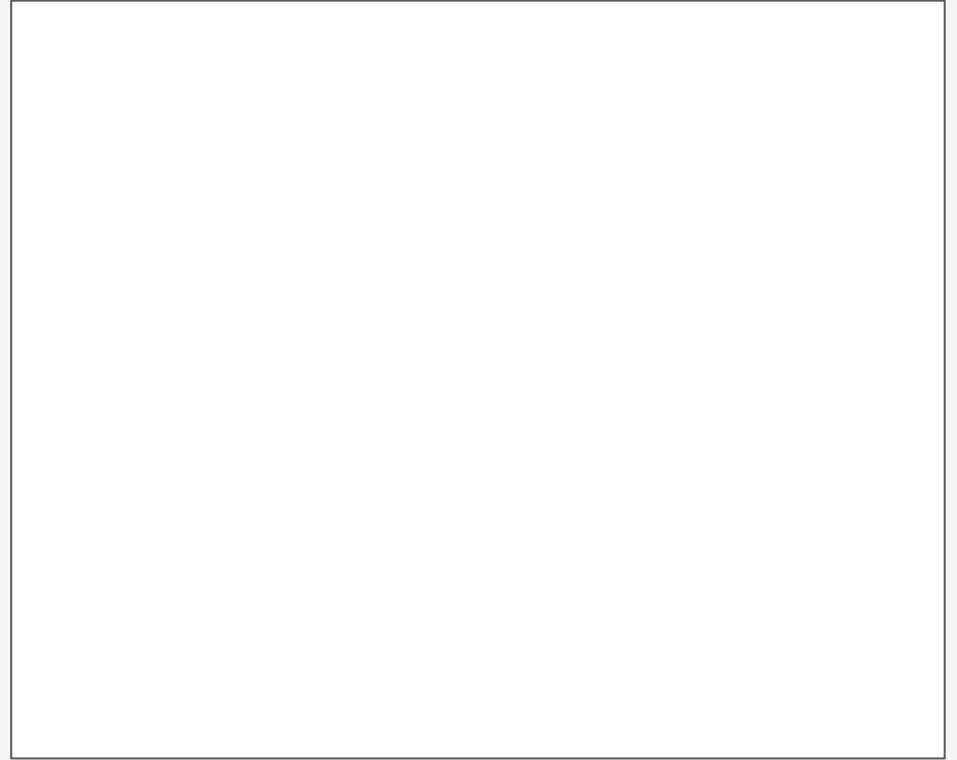


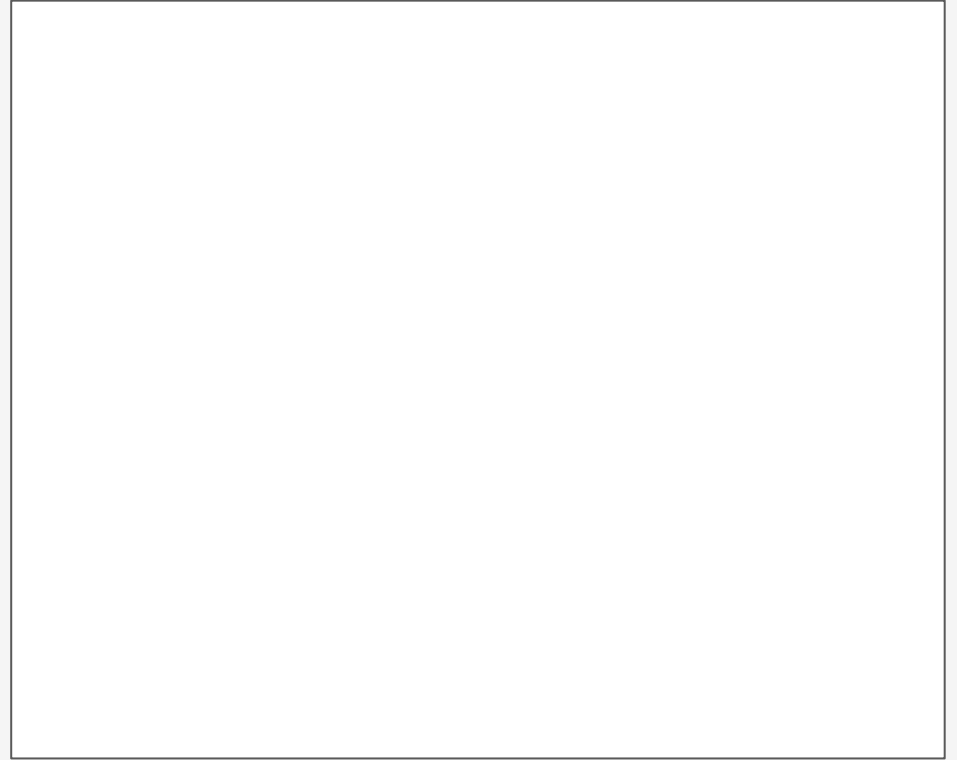
Q&A

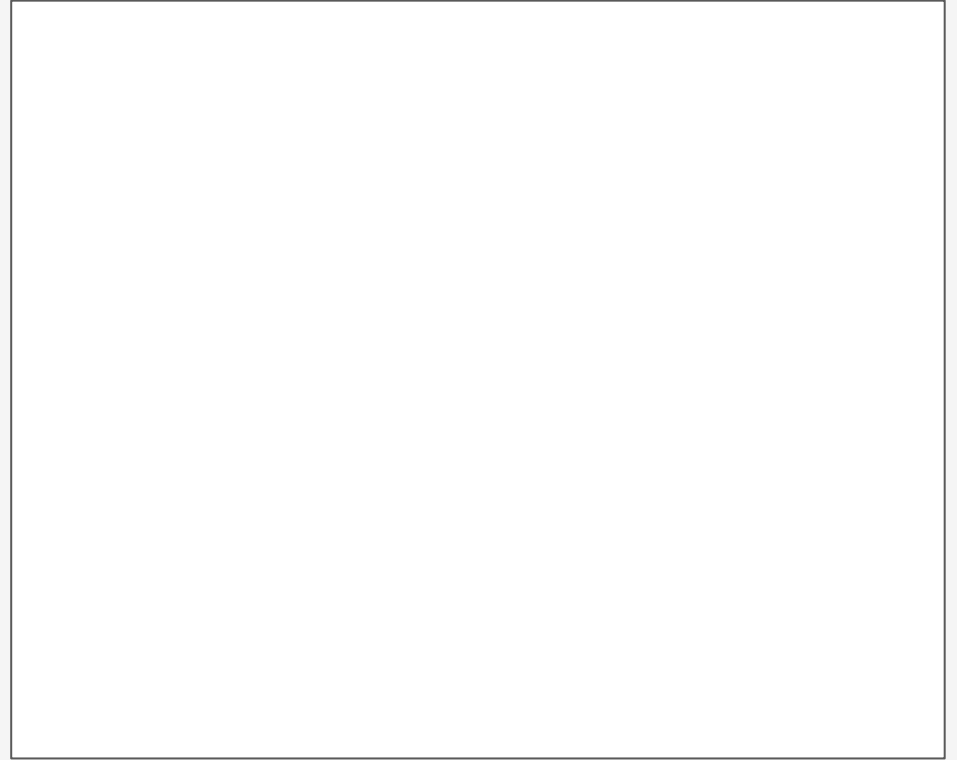
Don't be shy! We don't bite!

Please consider to give us a feedback!















So long, and
Thanks for all
the 

