CYBERTEC

DATA SCIENCE & POSTGRESQL

When it all GOes right

Writing applications with Go for PostgreSQL

Senior Consultant/Developer

Pavlo Golub

MAIL

pavlo.golub@cybertec.at

Twitter

@PavloGolub

WEB

www.cybertec-postgresql.com

CYBERTEC
DATA SCIENCE & POSTGRESQL
www.cybertec-postgresql.com

ABOUT **CYBERTEC**

Highly specialized,

fast growing

IT company

International Team

(10 countries),

six locations worldwide

PostgreSQL Services &

Support

Owner managed

since 2000

AUSTRIA (HQ)

CYBERTEC POSTGRESQL
INTERNATIONAL (HQ)

ESTONIA

CYBERTEC POSTGRESQL
NORDIC

SWITZERLAND

CYBERTEC POSTGRESQL
SWITZERLAND

POLAND

CYBERTEC POSTGRESQL
POLAND

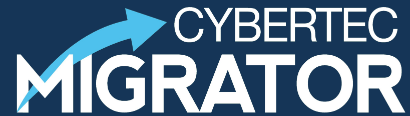
URUGUAY

CYBERTEC POSTGRESQL
SOUTH AMERICA

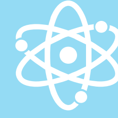
SOUTH AFRICA

CYBERTEC POSTGRESQL
SOUTH AFRICA

DATABASE - PRODUCTS



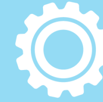
WHY PostgreSQL?



ADVANCED OPEN SOURCE DATABASE SYSTEM

25 YEARS OF DEVELOPMENT

NO LICENSE COSTS



EXTENSIVE FUNCTIONALITY

RELIABILITY

LOW SUPPORT COSTS

SCALABILITY

www.cybertec-postgresql.com

CYBERTEC
DATA SCIENCE & POSTGRESQL
www.cybertec-postgresql.com

- Intro to Go
- IDE’s and tools
- Drivers
- Useful extensions
- Testing
- CI/CD

CYBERTEC
DATA SCIENCE & POSTGRESQL
www.cybertec-postgresql.com

When it all GOes right

Intro to Go

Why Go?

Native binaries

Go compiles quickly and produces binary code fast.

Simplicity

It’s designed to be simple and easy to learn for developers of all levels.

Concurrency

Go provides simple and efficient support for concurrent programming.

Comprehensive tools

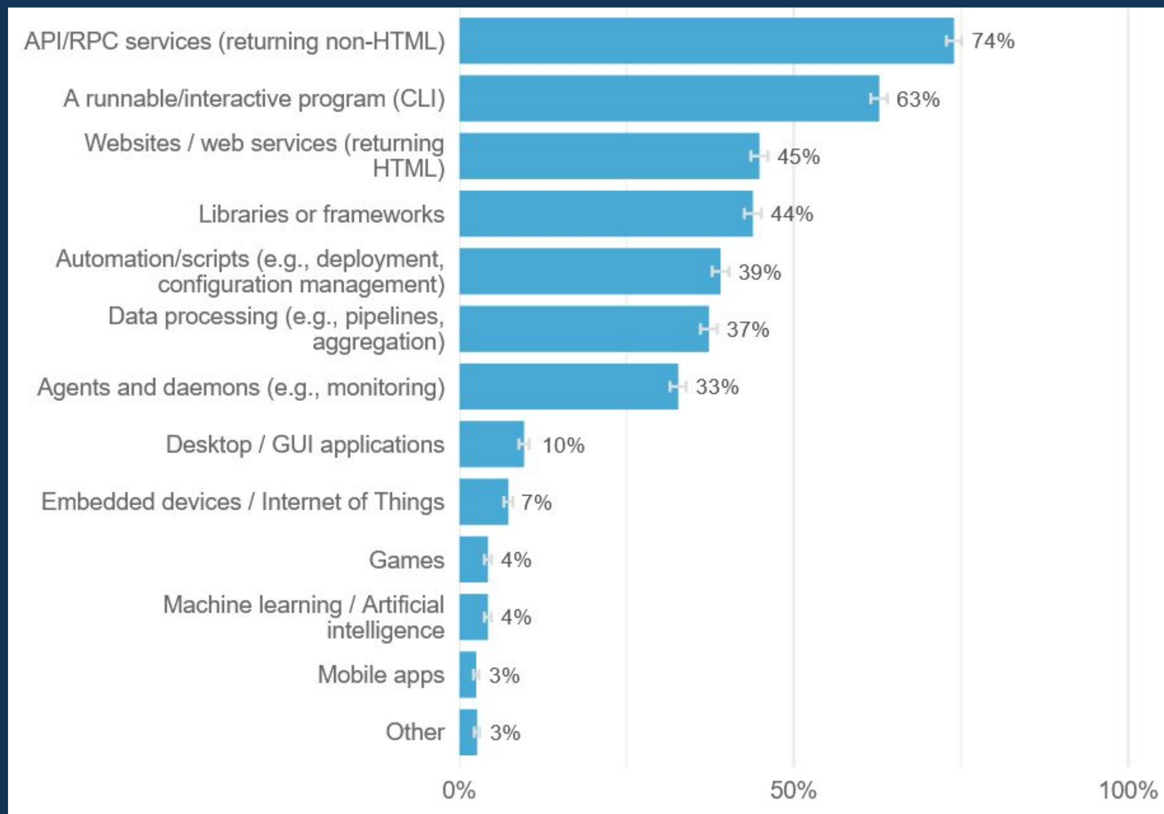
Go comes with a comprehensive set of tools and libraries.

Cross-platform

Go is designed to allow programs to be compiled and run on any platform.

Go Developer Survey 2024

How did survey respondents use Go?



https://go.dev/blog/survey2024-h1-results

CYBERTEC
DATA SCIENCE & POSTGRESQL
www.cybertec-postgresql.com

Go Developer Survey 2020

I work with Go

in the areas listed

Area	2020	2019
Web programming	68 %	68 %
Databases	46 %	47 %
DevOps	42 %	39 %
Network programming	41 %	43 %
Systems programming	40 %	39 %
Security	15 %	14 %
Finance / commerce	14 %	13 %
Desktop / GUI	9 %	9 %
Embedded devices / IoT	9 %	9 %
Data science	8 %	9 %
ML / AI	6 %	7 %
Gaming	5 %	5 %
Scientific / numeric	4 %	5 %
Mobile	4 %	4 %
Other	5 %	5 %

0% 20% 40% 60% 80%

% of respondents

https://go.dev/blog/survey2020-results

CYBERTEC
DATA SCIENCE & POSTGRESQL
www.cybertec-postgresql.com

Top products written in Go

- **Kubernetes (K8s)** - production-grade container management
- **Prometheus** - monitoring system and time series database
- **OpenShift** - a family of containerization software products
- **Moby** - a collaborative project for the container ecosystem
- **Hugo** - the world's fastest framework for building websites
- **Grafana** - observability and data visualization platform
- **frp** - a fast reverse proxy to expose a local server behind a NAT
- **Gogs** - painless self-hosted Git service
- **Etcd** - distributed reliable key-value store
- **Caddy** - fast, multi-platform web server with automatic HTTPS

Postgres-related Go products

- **CockroachDB** - a cloud-native distributed SQL database
- **pgweb** - a cross-platform client for PostgreSQL databases
- **stolon** - a cloud native PostgreSQL manager
- **postgres operators**
  - by Zalando, Cybertec, Crunchy, CloudNativePG
- **wal-g** - archives and restoration for Postgres
- **pgwatch2** - PostgreSQL metrics monitor/dashboard
- **pg_timetable** - advanced scheduling for PostgreSQL
- **pg_flame** - a flamegraph generator for EXPLAIN output

Top products written in Go

FIND MORE

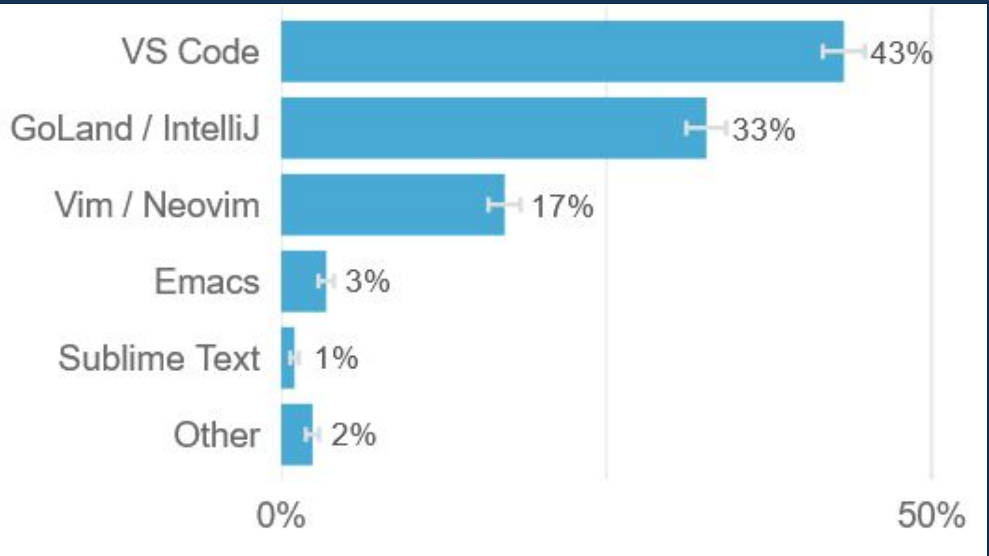
https://awesome-go.com/

https://github.com/avelino/awesome-go/

https://github.com/trending/go/

When it all GOes right

IDE’s and tools



https://go.dev/blog/survey2024-h1-results

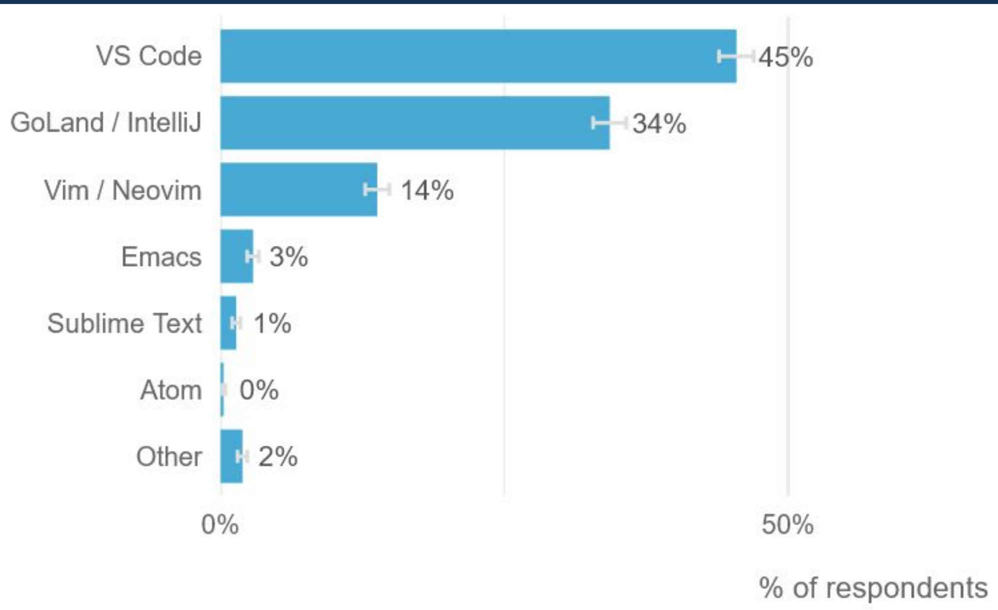
Go Developer Survey 2024

"My preferred editor for Go code is..."

CYBERTEC
DATA SCIENCE & POSTGRESQL
www.cybertec-postgresql.com

Go Developer Survey 2022

"My preferred editor for Go code is…"



https://go.dev/blog/survey2022-q2-results

CYBERTEC
DATA SCIENCE & POSTGRESQL
www.cybertec-postgresql.com

My environment

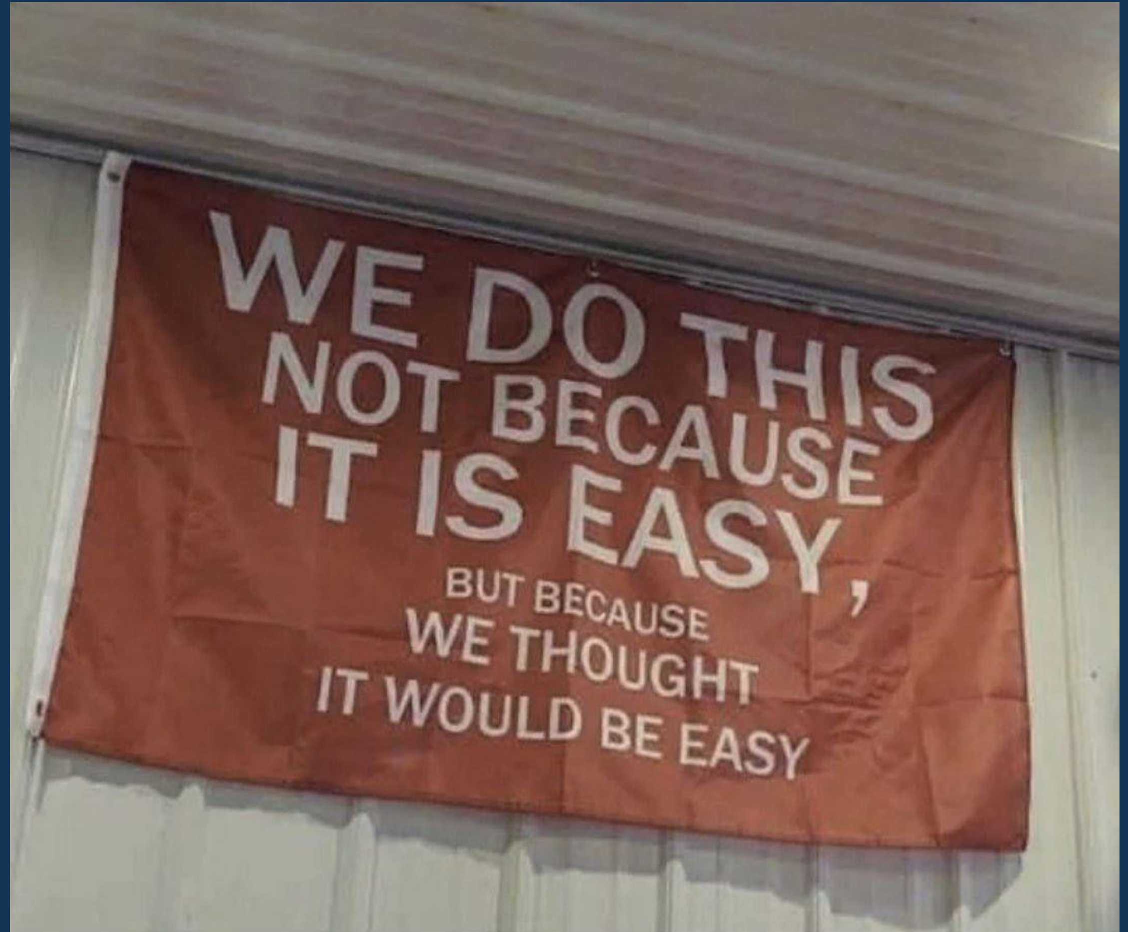
- **VSCode** with the official vscode-go plugin
- **mfridman/tparse** - summarizes go test output with color tables
- **golangci-lint** - fast linters Runner for Go
- **Tabnine** - the AI code completion tool
- **GitHub Copilot** - the AI developer tool
- **GoReleaser** - deliver Go binaries as quickly and easily as possible
- **PostgreSQL** - most advanced object-relational database
- **gitpod.io** - container-based ready-to-code developer environments

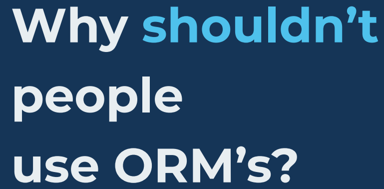
When it all GOes right

PostgreSQL Drivers

NO
ORM
ZONE

Why do people use ORM’s?





Why shouldn't people use ORM's?

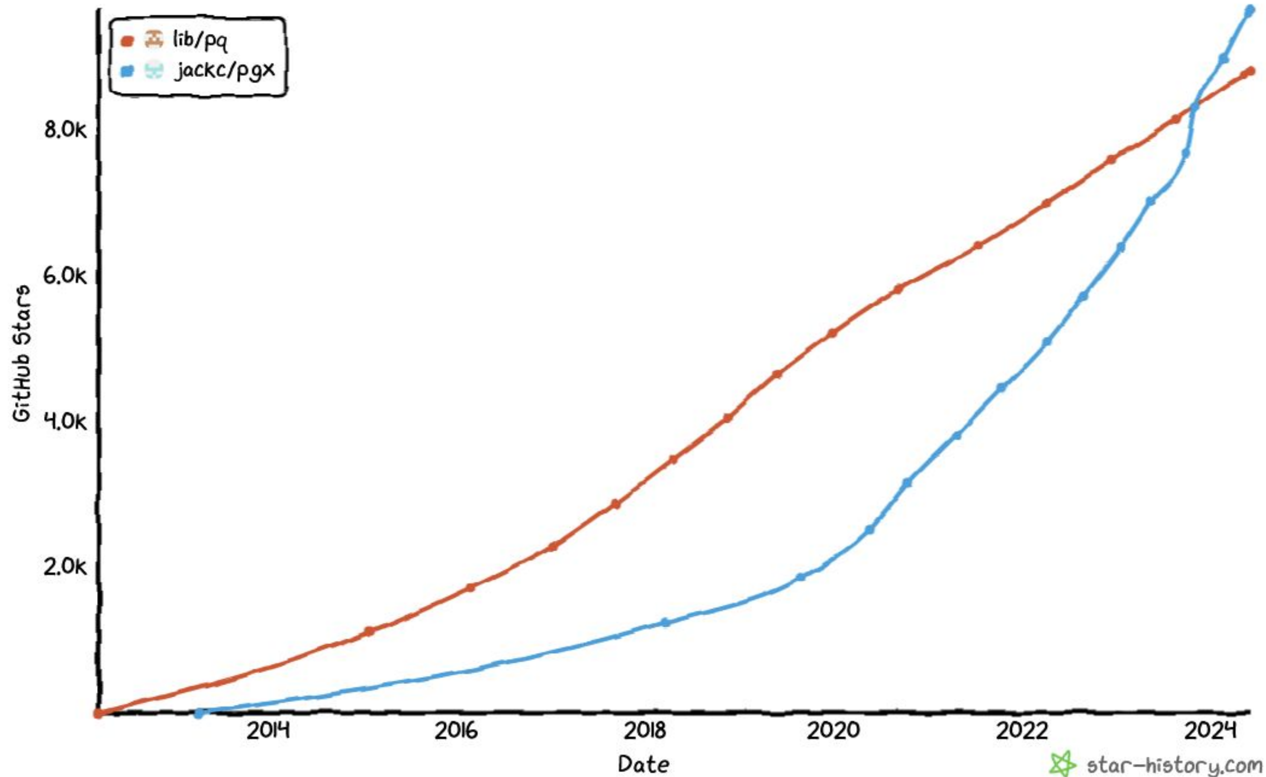
Pavlo Golub
@PavloGolub

If God wanted us to use ORMs, God would not have created SQL!

4:59 PM · Dec 14, 2022

Drivers’ availability

- **database/sql** is a set of database access interfaces
  - standard de facto creating database applications
  - needs implementation to work
  - read **go-database-sql.org** for more information
- **github.com/lib/pq** - pure Go Postgres driver for **database/sql**
  - is in maintenance mode for several years
- **github.com/jackc/pgx** — PostgreSQL driver and toolkit for Go
  - is the default choice nowadays
  - use this even with **database/sql**


Star History
lib/pq
jackc/pgx
GitHub Stars
8.0k
6.0k
4.0k
2.0k
2014
2016
2018
2020
2022
2024
Date
star-history.com

pgx vs database/sql

- Use **jackc/pgx** interface (not **database/sql**) when
  - The application only targets PostgreSQL.
  - No other libraries that require **database/sql** are in use.

- Otherwise use **database/sql** with **jackc/pgx/stdlib**
  - compatibility with non-PostgreSQL databases is required
  - when using other libraries that require **database/sql** such as **sqlx** or **gorm**

pgx unique features

- Support for approximately 70 different PostgreSQL types
- Automatic statement preparation and caching
- Batch queries
- Single-round trip query mode
- Full TLS connection control
- Binary format support for custom types
- COPY protocol support for faster bulk data loads
- Extendable tracing and logging support
- Connection pool with after-connect hook

pgx unique features

- Listen / Notify
- Conversion of PostgreSQL arrays to Go slice mappings
- `inet`, `cidr`, `hstore`, `json` and `jsonb` native support
- Large object support
- `NULL` mapping to `Null*` struct or pointer to pointer
- Supports `database/sql.Scanner` and `database/sql/driver.Valuer` interfaces for custom types
- Notice response handling
- Simulated nested transactions with savepoints

Hello World:
database/sql

```go
package main

import (
	"database/sql"
	"fmt"
	"os"

	_ "github.com/jackc/pgx/v5/stdlib"
)

func main() {
	db, err := sql.Open("pgx", os.Getenv("DATABASE_URL"))
	if err != nil {
		fmt.Fprintf(os.Stderr, "Unable to connect to database: %v\n", err)
		os.Exit(1)
	}
	defer db.Close()

	var greeting string
	err = db.QueryRow("select 'Hello, world!'").Scan(&greeting)
	if err != nil {
		fmt.Fprintf(os.Stderr, "QueryRow failed: %v\n", err)
		os.Exit(1)
	}

	fmt.Println(greeting)
}
```

Hello World: jackc/pgx

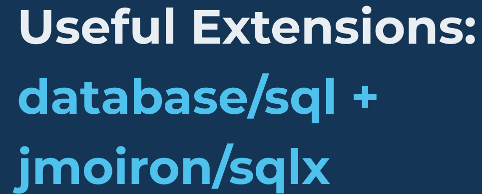
```go
package main

import (
	"context"
	"fmt"
	"os"

	"github.com/jackc/pgx/v5"
)

func main() {
	conn, err := pgx.Connect(context.Background(), os.Getenv("DATABASE_URL"))
	if err != nil {
		fmt.Fprintf(os.Stderr, "Unable to connect to database: %v\n", err)
		os.Exit(1)
	}
	defer conn.Close(context.Background())

	var greeting string
	err = conn.QueryRow(context.Background(), "select 'Hello, world!'").Scan(&greeting)
	if err != nil {
		fmt.Fprintf(os.Stderr, "QueryRow failed: %v\n", err)
		os.Exit(1)
	}

	fmt.Println(greeting)
}
```

Hello World: pgx/pgxpool

```go
package main

import (
	"context"
	"fmt"
	"os"

	"github.com/jackc/pgx/v5/pgxpool"
)

func main() {
	dbpool, err := pgxpool.New(context.Background(), os.Getenv("DATABASE_URL"))
	if err != nil {
		fmt.Fprintf(os.Stderr, "Unable to create connection pool: %v\n", err)
		os.Exit(1)
	}
	defer dbpool.Close()

	var greeting string
	err = dbpool.QueryRow(context.Background(), "select 'Hello, world!'").Scan(&greeting)
	if err != nil {
		fmt.Fprintf(os.Stderr, "QueryRow failed: %v\n", err)
		os.Exit(1)
	}

	fmt.Println(greeting)
}
```

When it all GOes right

Useful extensions



Useful Extensions: database/sql + jmoiron/sqlx

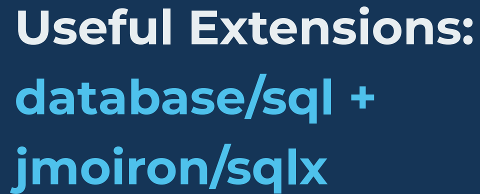
These extensions to the built-in verbs:

- `MustExec() sql.Result` -- Exec, but panic on error
- `Queryx(...) (*sqlx.Rows, error)` - Query, but return an sqlx.Rows
- `QueryRowx(...) *sqlx.Row` -- QueryRow, but return an sqlx.Row

And these new semantics:

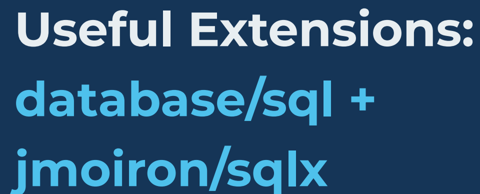
- `Get(dest interface{}, ...) error`
- `Select(dest interface{}, ...) error`

CYBERTEC
DATA SCIENCE & POSTGRESQL
www.cybertec-postgresql.com



Useful Extensions:
database/sql + jmoiron/sqlx

```go
type Place struct {
    Country       string
    City          sql.NullString
    TelephoneCode int `db:"telcode"`
}

rows, err := db.Queryx("SELECT * FROM place")
for rows.Next() {
    var p Place
    err = rows.StructScan(&p)
}
```



Useful Extensions: database/sql + jmoiron/sqlx

```go
p := Place{}
pp := []Place{}

// this will pull the first place directly into p
err = db.Get(&p, "SELECT * FROM place LIMIT 1")

// this will pull places with telcode > 50 into the slice pp
err = db.Select(&pp, "SELECT * FROM place WHERE telcode > ?", 50)

// they work with regular types as well
var id int
err = db.Get(&id, "SELECT count(*) FROM place")

// fetch at most 10 place names
var names []string
err = db.Select(&names, "SELECT name FROM place LIMIT 10")
```

Useful Extensions:
jackc/pgx

It's perfect!
No extensions
needed!

Useful Extensions: jackc/pgx

```go
type product struct {
	ID    int32
	Name  string
	Price int32
}

rows, _ := conn.Query(ctx, "select * from products where price < $1", 12)
products, err := pgx.CollectRows(rows, pgx.RowToStructByName[product])
if err != nil {
	fmt.Printf("CollectRows error: %v", err)
	return
}

for _, p := range products {
	fmt.Printf("%s: $%d\n", p.Name, p.Price)
}
```

Useful Extensions:

sqlc-dev/sqlc

sqlc generates **type-safe** code from SQL. Here's how it works:

1. You write queries in SQL.
2. You run sqlc to generate code with type-safe interfaces to those queries.
3. You write application code that calls the generated code.

Check out an **interactive example** to see it in action

When it all GOes right

Testing

Testing Approaches

- Real PostgreSQL server
  - running locally
  - **testcontainers/testcontainers-go**
- Mocking libraries
  - **DATA-DOG/go-sqlmock**
  - **pashagolub/pgxmock**
- Mock PostgreSQL wire protocol
  - **jackc/pgmock**
  - **cockroachdb/cockroach-go/testserver**
  - **cockroachdb/copyist**

Mocking Example: pgxmock

```go
type PgxIface interface {
	Begin(context.Context) (pgx.Tx, error)
	Close(context.Context) error
}

func recordStats(db PgxIface, userID, productID int) (err error) {
	if tx, err := db.Begin(context.Background()); err != nil {
		return
	}
	defer func() {
		switch err {
		case nil:
			err = tx.Commit(context.Background())
		default:
			_ = tx.Rollback(context.Background())
		}
	}()
	sql := "UPDATE products SET views = views + 1"
	if _, err = tx.Exec(context.Background(), sql); err != nil {
		return
	}
	sql = "INSERT INTO product_viewers (user_id, product_id) VALUES ($1, $2)"
	if _, err = tx.Exec(context.Background(), sql, userID, productID); err != nil {
		return
	}
	return
}
```

Mocking Example: pgxmock

```go
// a successful case
func TestShouldUpdateStats(t *testing.T) {
	mock, err := pgxmock.NewPool()
	if err != nil {
		t.Fatal(err)
	}
	defer mock.Close()

	mock.ExpectBegin()
	mock.ExpectExec("UPDATE products").
		WillReturnResult(pgxmock.NewResult("UPDATE", 1))
	mock.ExpectExec("INSERT INTO product_viewers").
		WithArgs(2, 3).
		WillReturnResult(pgxmock.NewResult("INSERT", 1))
	mock.ExpectCommit()

	// now we execute our method
	if err = recordStats(mock, 2, 3); err != nil {
		t.Errorf("error was not expected while updating: %s", err)
	}

	// we make sure that all expectations were met
	if err := mock.ExpectationsWereMet(); err != nil {
		t.Errorf("there were unfulfilled expectations: %s", err)
	}
}
```

Mocking Example: pgxmock

```go
// a failing test case
func TestShouldRollbackStatUpdatesOnFailure(t *testing.T) {
	mock, err := pgxmock.NewPool()
	if err != nil {
		t.Fatal(err)
	}
	defer mock.Close()

	mock.ExpectBegin()
	mock.ExpectExec("UPDATE products").
		WillReturnResult(pgxmock.NewResult("UPDATE", 1))
	mock.ExpectExec("INSERT INTO product_viewers").
		WithArgs(2, 3).
		WillReturnError(fmt.Errorf("some error"))
	mock.ExpectRollback()

	// now we execute our method
	if err = recordStats(mock, 2, 3); err == nil {
		t.Errorf("was expecting an error, but there was none")
	}

	// we make sure that all expectations were met
	if err := mock.ExpectationsWereMet(); err != nil {
		t.Errorf("there were unfulfilled expectations: %s", err)
	}
}
```

Coverage:
go test + tparse

```
 basic  master ↓2 ~2 1  1.19.5  go test ./... -coverprofile='c.out' -json | tparse -all
```

STATUS	ELAPSED	TEST	PACKAGE
PASS	0.00	TestShouldUpdateStats	basic
PASS	0.00	TestShouldRollbackStatUpdatesOnFailure	basic

STATUS	ELAPSED	PACKAGE	COVER	PASS	FAIL	SKIP
PASS	0.30s	github.com/pashagolub/pgxmock/v2/examples/basic	60.0%	2	0	0

```
 basic  master ↓2 ~2 1  1.19.5  go tool cover -func='c.out'
github.com/pashagolub/pgxmock/v2/examples/basic/basic.go:15:    recordStats     85.7%
github.com/pashagolub/pgxmock/v2/examples/basic/basic.go:39:    main            0.0%
total:                                                          (statements)    60.0%
 basic  master ↓2 ~2 1  1.19.5  go tool cover -html='c.out'
 basic  master ↓2 ~2 1  1.19.5
```

Coverage:
go tool cover

github.com/pashagolub/pgxmock/v2/examples/basic/basic.go (60.0%) not tracked not covered covered

```
package main

import (
	"context"

	pgx "github.com/jackc/pgx/v5"
	pgxpool "github.com/jackc/pgx/v5/pgxpool"
)

type PgxIface interface {
	Begin(context.Context) (pgx.Tx, error)
	Close()
}

func recordStats(db PgxIface, userID, productID int) (err error) {
	tx, err := db.Begin(context.Background())
	if err != nil {
		return
	}
	defer func() {
		switch err {
		case nil:
			err = tx.Commit(context.Background())
		default:
			_ = tx.Rollback(context.Background())
		}
	}()
	sql := "UPDATE products SET views = views + 1"
	if _, err = tx.Exec(context.Background(), sql); err != nil {
		return
	}
	sql = "INSERT INTO product_viewers (user_id, product_id) VALUES ($1, $2)"
	if _, err = tx.Exec(context.Background(), sql, userID, productID); err != nil {
		return
	}
	return
}

func main() {
	// @NOTE: the real connection is not required for tests
	db, err := pgxpool.New(context.Background(), "postgres://rolname@hostname/dbname")
	if err != nil {
		panic(err)
	}
	defer db.Close()

	if err = recordStats(db, 1 /*some user id*/, 5 /*some product id*/); err != nil {
		panic(err)
	}
}
```

When it all GOes right

CI/CD

GitHub Integration

- **Dependabot** - maintains repository's dependencies automatically
- **CodeQL** - action runs analysis engine to find security vulnerabilities
- **Build & Test** - action runs on each pull request, or manually
- **Release** - action runs on new tag, publishes release
- **Docker** - action runs on every commit, publishes Docker images

```yaml
version: 2
updates:

  # Maintain dependencies for Go modules
  - package-ecosystem: gomod
    directory: "/"
    schedule:
      interval: daily
      time: "04:00"
    open-pull-requests-limit: 10

    # Maintain dependencies for GitHub Actions
  - package-ecosystem: "github-actions"
    directory: "/"
    schedule:
      interval: "daily"
```

Github Actions:
CodeQL

```yaml
name: "CodeQL"

on:
  push:
    branches: [ master ]
  pull_request:
    branches: [ master ]
  schedule:
    - cron: '19 11 * * 6'

jobs:
  analyze:
    name: Analyze
    runs-on: ubuntu-latest
    permissions:
      actions: read
      contents: read
      security-events: write

    strategy:
      fail-fast: false
      matrix:
        language: [ 'go' ]

    steps:
    - name: Checkout repository
      uses: actions/checkout@v3

    # Initializes the CodeQL tools for scanning.
    - name: Initialize CodeQL
      uses: github/codeql-action/init@v2
      with:
        languages: ${{ matrix.language }}

    - name: Autobuild
      uses: github/codeql-action/autobuild@v2

    - name: Perform CodeQL Analysis
      uses: github/codeql-action/analyze@v2
```

Github Actions:

Build & Test

```
90   test-postgresql-ubuntu:
91     if: true # false to skip job during debug
92     name: Test and Build on Ubuntu
93     runs-on: ubuntu-latest
94     steps:
95
96     - name: Start PostgreSQL on Ubuntu
97       run: |
98         sudo systemctl start postgresql.service
99         pg_isready
100
101    - name: Create scheduler user
102      run: |
103        sudo -u postgres psql --command="CREATE USER scheduler PASSWORD 'somestrong'" --command="\du"
104
105    - name: Create timetable database
106      run: |
107        sudo -u postgres createdb --owner=scheduler timetable
108        PGPASSWORD=somestrong psql --username=scheduler --host=localhost --list timetable
109
110    - name: Check out code
111      uses: actions/checkout@v3
112
113    - name: Set up Golang
114      uses: actions/setup-go@v3
115      with:
116        go-version: '1.19'
117
118    - name: Get dependencies
119      run: |
120        go mod download
121        go version
122
123    - name: GolangCI-Lint
124      uses: golangci/golangci-lint-action@v3
125      with:
126        version: latest
127
128    - name: Test
129      run: go test -failfast -v -timeout=300s -p 1 -coverprofile=profile.cov ./...
130
131    - name: Coveralls
132      uses: shogo82148/actions-goveralls@v1
133      with:
134        path-to-profile: profile.cov
```

Github Actions:
Release

```yaml
name: Release
on:
  release:
    types: [created]

jobs:

  goreleaser:
    if: true # false to skip job during debug
    runs-on: ubuntu-latest
    name: goreleaser
    steps:

    - name: Set up Golang
      uses: actions/setup-go@v3
      with:
        go-version: '1.19'

    - name: Check out code into the Go module directory
      uses: actions/checkout@v3

    - name: Unshallow
      run: git fetch --prune --unshallow

    - name: Release via goreleaser
      uses: goreleaser/goreleaser-action@v4
      with:
        args: release
      env:
        GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
```

Github Actions:
Docker

```
16    docker:
17      if: true # false to skip job during debug
18      runs-on: ubuntu-latest
19      steps:
20
21      - name: Check out code
22        uses: actions/checkout@v3
23
24      - name: Set up Golang
25        uses: actions/setup-go@v3
26        with:
27          go-version: '1.19'
28
29      # despite the fact docker will build binary internally
30      # we want to stop workflow in case of any error before pushing to registry
31      - name: Get dependencies and Build
32        run: |
33          go version
34          go mod download
35          go build
36
37      - name: Version strings
38        id: version
39        run: |
40          echo "RELEASE_TIME=$(git show -s --format=%cI HEAD)" >> $GITHUB_OUTPUT
41
42      - name: Publish to Registry
43        uses: elgohr/Publish-Docker-Github-Action@v5
44        env:
45          VERSION: ${{ github.ref_name }}
46          COMMIT: ${{ github.sha }}
47          DATE: ${{ steps.version.outputs.RELEASE_TIME }}
48        with:
49          name: cybertecpostgresql/pg_timetable
50          username: ${{ secrets.DOCKER_USERNAME }}
51          password: ${{ secrets.DOCKER_PASSWORD }}
52          buildargs: VERSION,COMMIT,DATE
```

When it all GOes right

Takeaways

Takeaways

- Go language is popular, fast, easy and well-scaled
- 45% of developers use Go to work with databases (go.dev survey)
- You can use your preferred editor or use special IDE’s to work
- Kubernetes operators (including Postgres ones) are written in Go
- Go is flexible when working with Postgres: use **sql** or **pgx** interfaces
- Go has powerful programming tools and GitHub/GitLab integration
- Go is backwards-compatible. The API’s may grow but not in a way that breaks existing Go v1 code

CYBERTEC
DATA SCIENCE & POSTGRESQL
www.cybertec-postgresql.com

DON'T BE A STRANGER

PERSONAL GITHUB

pashagolub.github.io

CYBERTEC BLOG

www.cybertec-postgresql.com/en/blog/

CYBERTEC GITHUB

www.github.com/cybertec-postgresql

“I would rather have questions that can't be answered than answers that can't be questioned.”

Richard Feynman

#StandWithUkraine