sqlc SQL to type-safe Go

CONCEPTS

> You write schema and queries in plain SQL; sqlc generates type-safe Go code, so the SQL is the source of truth.

schema.sql tables or migrations
↓ `sqlc generate`
query.sql named queries
↓ `parsed and typed`
Go package models, methods
↓ `you call`
Your app typed results

> Commit the generated code and run sqlc diff in CI to catch forgotten regeneration.

COMMANDS

`sqlc init`
Create an empty sqlc.yaml (v2 format)

`sqlc generate`
Generate code from schema and queries

`sqlc compile`
Check SQL for syntax and type errors

`sqlc vet`
Run lint rules over the queries

`sqlc diff`
Show how output differs from files on disk

`sqlc -f <file> generate`
Use another config file

QUERY ANNOTATIONS

`-- name: <Name> :one`
Return a single row

`-- name: <Name> :many`
Return a slice of rows

`-- name: <Name> :exec`
Run it, return only an error

`-- name: <Name> :execrows`
Return affected row count

QUERY ANNOTATIONS (cont.)

`-- name: <Name> :execresult`
Return the driver's sql.Result

`-- name: <Name> :execlastid`
Return the generated ID

PARAMETERS AND MACROS

`sqlc.arg(<name>)`
Name a parameter in the Go method

`sqlc.narg(<name>)`
Name a parameter and make it nullable

`sqlc.slice("<name>")`
Slice in IN (...) for MySQL/SQLite

`sqlc.embed(<table>)`
Return the table's model struct as a field

SQLC.YAML

`version: "2"`
Config format version

`engine: postgresql`
postgresql, mysql or sqlite

`schema: "<path>"`
Schema file, dir or migrations

`queries: "<path>"`
Query file or directory

`gen.go.package: <name>`
Go package name of the output

`gen.go.out: <dir>`
Output directory

`sql_package: pgx/v5`
Pick driver (default database/sql)

`emit_json_tags: true`
Add json tags to structs

`emit_interface: true`
Generate a Querier interface

`emit_empty_slices: true`
Return [] instead of nil for :many

`emit_pointers_for_null_types`
Use *T for nullable columns