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

sqlc version
Print installed version

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

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

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

QUERY ANNOTATIONS (cont.)

-- name: *<Name>* **:copyfrom**
Bulk insert, driver specific

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

Placeholders per engine

Engine	Placeholder
`postgresql`	$1, $2
`mysql`	?
`sqlite`	?

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

overrides:
Map db types or columns to Go types

SQLC.YAML (cont.)

rules: [*<name>***]**
Enable vet rules for this package

database.uri: ${DB_URL}
Live database for vet; env vars expand

analyzer.database: false
Skip database-backed analysis

VET RULES

sqlc/db-prepare
Built-in: prepare every query on the DB

-- @sqlc-vet-disable *<rule>*
Skip a rule for one query

CEL variables in a rule

Variable	Holds
`query.sql`	SQL text
`query.name`	Query name
`query.cmd`	one, many, exec...
`query.params`	Parameter list
`config.engine`	Configured engine

RECIPES

Minimal sqlc.yaml

```
version: "2"
sql:
  - engine: "postgresql"
    schema: "schema.sql"
    queries: "query.sql"
    gen:
      go:
        package: "db"
        out: "db"
```

CRUD queries

```
-- name: GetAuthor :one
SELECT * FROM authors WHERE id =
$1;

-- name: ListAuthors :many
SELECT * FROM authors ORDER BY
name;

-- name: DeleteAuthor :exec
DELETE FROM authors WHERE id =
$1;
```

Custom vet rule

```
rules:
  - name: no-delete
    message: "don't use delete"
    rule:
query.sql.contains("DELETE")
# then add - no-delete under
sql[].rules
```

CI freshness check

```
sqlc compile
sqlc vet
sqlc diff   # non-zero if
generated code is stale
```