Go with sqlc Project Structure

Type-safe SQL with sqlc. Write queries in SQL, get generated Go code with full type safety.

#go #golang #sqlc #sql #postgres #database #backend

PNG PDF Copy Prompt

Project Directory

```
myproject/
    main.go  Entry point
    go.mod
    go.sum
    sqlc.yaml  sqlc configurat…
    Makefile  generate, migra…
    .env.example
    .gitignore
  cmd/
    api/
      main.go
  internal/
    db/  sqlc generated …
      db.go  Generated inter…
      models.go  Generated struc…
      users.sql.go  Generated queri…
      querier.go  Query interface
    handler/
      handler.go
      users.go
    service/
      user_service.go
    router/
      router.go
  sql/  SQL source files
    queries/  Query definitio…
      users.sql  User queries
      orders.sql
    schema/  Table definitio…
      001_users.sql
      002_orders.sql
```

Why This Structure?

sqlc generates type-safe Go code from SQL queries. Write SQL in `.sql` files, run `sqlc generate`, and get Go functions with proper types. No runtime reflection, compile-time query validation, and you keep full control over SQL.

Key Directories

`sql/queries/` - Your SQL queries with sqlc annotations

`sql/schema/` - Table definitions for sqlc to parse

`internal/db/` - Generated Go code (do not edit)

`sqlc.yaml` - Configuration for code generation

sqlc Query Annotations

```
-- sql/queries/users.sql
-- name: GetUser :one
SELECT * FROM users WHERE id = $1;

-- name: ListUsers :many
SELECT * FROM users ORDER BY created_at;

-- name: CreateUser :one
INSERT INTO users (email, name)
VALUES ($1, $2)
RETURNING *;
```

Getting Started

1. `go install github.com/sqlc-dev/sqlc/cmd/sqlc@latest`
2. Create `sqlc.yaml` configuration
3. Write SQL in `sql/queries/` and `sql/schema/`
4. `sqlc generate`
5. Import generated code from `internal/db/`

When To Use This

- You prefer writing raw SQL over ORMs
- Need compile-time query validation
- Working with complex SQL (joins, CTEs, window functions)
- Want zero runtime reflection overhead
- PostgreSQL, MySQL, or SQLite projects

Trade-offs

SQL knowledge required - Must write actual SQL, not abstracted away

Regeneration step - Run sqlc generate after query changes

Migration tooling separate - Use goose, migrate, or similar for migrations

Testing Strategy

Generated interface - Querier interface enables mocking

testcontainers - Spin up real Postgres for integration tests

sqlc vet - Validates queries against actual DB schema