goose Database migrations

START HERE

`goose create <name> sql`
New migration file

`goose <driver> <dbstring> up`
Apply all pending

`goose <driver> <dbstring> redo`
Down, then up again

`goose <driver> <dbstring> version`
Current DB version

`goose <driver> <dbstring> status`
Applied or pending?

`goose env`
Show effective config

`goose <driver> <dbstring> down`
Undo the last one

`goose validate`
Lint files, no DB

create new .sql file → **Up / Down** SQL annotations → `goose up` → **Database** goose_db_version → `goose status` → **Verified** applied list

CONCEPTS

Set GOOSE_DRIVER, GOOSE_DBSTRING and GOOSE_MIGRATION_DIR (or a .env file) and drop the driver and dbstring from every command.

Each file needs exactly one '-- +goose Up' marker; Down is optional but must come after Up.

Timestamp names are handy in development; run 'goose fix' once migrations are final to renumber them sequentially.

CONNECT

`goose <driver> <dbstring> <command>`
Full invocation form

`goose -dir <path> <command>`
Migration directory (default .)

`export GOOSE_DRIVER=<driver>`
Driver from environment

`export GOOSE_DBSTRING=<dbstring>`
Connection string from environment

`export GOOSE_MIGRATION_DIR=<path>`
Directory from environment

`goose -env <file> <command>`
Load variables from another env file
Default is .env in the current directory; -env=none turns loading off.

`goose -table <name> <command>`
Custom version table (schema.table ok)

`goose -timeout <duration> <command>`
Cap query duration, e.g. 1h13m

`goose -v <command>`
Verbose output

`goose -no-color <command>`
Plain output (or NO_COLOR)

`goose -version`
Print goose version

`goose -h`
Help and driver list

DBSTRING EXAMPLES

Driver and connection string

driver	dbstring
sqlite3	./app.db
postgres	"user=postgres dbname=app sslmode=disable"
mysql	"user:pass@/app?parseTime=true"
mssql	"sqlserver://user:pass@host:1433?database=app"
clickhouse	"tcp://127.0.0.1:9000"
redshift	"postgres://user:pass@host:5439/db"

Other drivers: tidb, starrocks, spanner, azuresql, ydb, turso. MySQL needs parseTime=true for status, and multiStatements=true to run several statements per file.

`goose -ssl-cert <file> -ssl-key <file> <command>`
Client TLS files (MySQL)

`goose -certfile <file> <command>`
Root CA PEM (MySQL)

CREATE

`goose create <name> sql`
New SQL migration

`goose create <name> go`
New Go migration

`goose -s create <name> sql`
Sequential number, not timestamp

`goose -dir <path> create <name> sql`
Create in a given directory

`goose fix`
Renumber timestamps to sequential
Order is preserved. Run it in CI only once migrations are ready to ship.

`goose validate`
Check files without running them

SQL ANNOTATIONS

`-- +goose Up`
Start of forward statements

`-- +goose Down`
Start of rollback statements

`-- +goose NO TRANSACTION`
Run outside a transaction (file top)
Needed for e.g. CREATE INDEX CONCURRENTLY in Postgres.

SQL ANNOTATIONS (cont.)

`-- +goose StatementBegin`
Open a block with inner semicolons

`-- +goose StatementEnd`
Close that block

`-- +goose ENVSUB ON`
Expand ${VAR} in the SQL below

`-- +goose ENVSUB OFF`
Stop expanding variables

`${VAR:-<default>}`
Env substitution with a fallback

Every statement must end with a semicolon. Functions and triggers need a StatementBegin/End pair.

MIGRATE

`goose up`
Apply all pending migrations

`goose up-by-one`
Apply just the next one

`goose up-to <version>`
Migrate up to a version

`goose down`
Roll back the latest one

`goose down-to <version>`
Roll back to a version

`goose down-to 0`
Roll back everything

`goose redo`
Re-run the latest migration

`goose reset`
Roll back all migrations

`goose -allow-missing up`
Apply out-of-order migrations
By default a file older than the applied head is rejected.

`goose -no-versioning up`
Run files without tracking

INSPECT

`goose status`
List migrations and applied time

`goose version`
Print current DB version

`goose env`
Print goose environment

GO MIGRATIONS

`goose.AddMigration(Up, Down)`
Register in init()

GO MIGRATIONS (cont.)

`func Up(tx *sql.Tx) error`
Migration function signature

`goose.Up(db, "migrations")`
Run pending from library code

`goose.UpByOne(db, "migrations")`
Library: apply one

`goose.SetBaseFS(embedFS)`
Use embedded migrations

`goose.SetDialect("postgres")`
Pick a dialect in code

`goose.WithAllowMissing()`
Library: allow out-of-order

Go migrations need your own goose binary that imports the migrations package; the stock CLI only runs SQL files.

INSTALL

`go install github.com/pressly/goose/v3/cmd/goose@latest`
Install with Go

`brew install goose`
Install with Homebrew

`go build -tags='no_mysql no_sqlite3' ./cmd/goose`
Build a lite binary
Tags: no_postgres, no_mysql, no_sqlite3, no_clickhouse, no_mssql, no_vertica, no_ydb, no_libsql, no_azuresql.

RECIPES

New Postgres migration

```
export GOOSE_DRIVER=postgres
export GOOSE_DBSTRING="user=app
dbname=app sslmode=disable"
goose -dir db/migrations create
add_users sql
# edit the file, then:
goose -dir db/migrations up
```

Migration with a function

```
-- +goose Up
-- +goose StatementBegin
CREATE FUNCTION touch() RETURNS
trigger AS $$
BEGIN NEW.updated = now();
RETURN NEW; END;
$$ LANGUAGE plpgsql;
-- +goose StatementEnd
-- +goose Down
DROP FUNCTION touch();
```

Local SQLite loop

```
goose sqlite3 ./app.db up
goose sqlite3 ./app.db status
goose sqlite3 ./app.db redo
```

Pre-release cleanup

```
goose -dir db/migrations
validate
goose -dir db/migrations fix
git add db/migrations
```