Drizzle ORM

Type-safe SQL schemas and queries in TypeScript.

01 Install & connect

```
npm i drizzle-orm pg
npm i -D drizzle-kit @types/pg
```

```
import * as schema from './db/schema';
const url = process.env.DATABASE_URL!;
const db = drizzle(url, { schema });
```

drizzle comes from drizzle-orm/node-postgres. schema enables db.query. With a pg Pool: drizzle(pool, { schema }).

02 Other drivers

node-postgres	pg, shown above
postgres-js	postgres.js
mysql2	MySQL
better-sqlite3	A local SQLite file path
libsql	Turso: url and authToken

Import from drizzle-orm/<driver>. mysql2 with a schema also needs mode: 'default' (or 'planetscale').

03 drizzle.config.ts

```
import { defineConfig } from 'drizzle-kit';
export default defineConfig({
  dialect: 'postgresql',
  schema: './src/db/schema.ts',
  out: './drizzle',
  dbCredentials: {
    url: process.env.DATABASE_URL!,
  },
});
```

dialect: postgresql, mysql, sqlite or turso.

04 Tables & column types

```
export const users = pgTable('users', {
  id: serial('id').primaryKey(),
  name: text('name').notNull(),
  email: text('email').notNull().unique(),
  active: boolean('active').default(true),
  createdAt: timestamp('created_at')
    .defaultNow().notNull(),
});
```

Builders come from drizzle-orm/pg-core.

05 Enums, JSON & auto-update

```
export const roleEnum = pgEnum('role',
  ['admin', 'user', 'moderator']);
export const users = pgTable('users', {
  role: roleEnum('role').default('user'),
  prefs: jsonb('prefs').$type<Prefs>(),
  updatedAt: timestamp('updated_at')
    .$onUpdate(() => new Date()),
});
```

pgEnum is a real Postgres ENUM. $type<T>() types JSON. $onUpdate runs in JS, not as a DB trigger.

06 Indexes & constraints

```
export const posts = pgTable('posts', {
  slug: text('slug').notNull(),
  authorId: integer('author_id').notNull(),
}, (t) => [
  uniqueIndex('slug_idx').on(t.slug),
  index('author_idx').on(t.authorId),
]);
```

Indexes go in the third argument. .on(a, b) is composite.

07 Foreign keys & types

```
export const posts = pgTable('posts', {
  authorId: integer('author_id').notNull()
    .references(() => users.id,
      { onDelete: 'cascade' }),
});
type Post = typeof posts.$inferSelect;
type NewPost = typeof posts.$inferInsert;
```

onDelete takes cascade, restrict, no action or set null. Insert types make defaults and nullable columns optional.

08 Select, sort & paginate

```
const page = await db.select().from(users)
  .where(eq(users.active, true))
  .orderBy(desc(users.createdAt))
  .limit(20).offset(40);
const names = await db.select({
  id: users.id, name: users.name,
}).from(users);
```

select() with no argument returns every column, typed. Operators, asc and desc import from drizzle-orm.

09 Filter operators

eq / ne	Equal, not equal
gt gte lt lte	Comparisons
like / ilike	ilike ignores case (Postgres)
inArray / between	In a list, in a range
isNull / isNotNull	NULL checks
and / or / not	Combine conditions

and() and or() take any number of conditions.

Drizzle ORM

Writes, relations, transactions and migrations.

10 Insert & upsert

```
const [user] = await db.insert(users)
  .values({ name: 'Bob', email: 'b@x.io' })
  .returning();
await db.insert(users).values(row)
  .onConflictDoUpdate({ target: users.email,
    set: { name: sql`excluded.name` } });
```

.returning() is Postgres and SQLite only. values([...]) inserts many rows. onConflictDoNothing() skips duplicates.

11 Update & delete

```
await db.update(users).set({ name: 'Ann' })
  .where(eq(users.id, 1));
await db.update(posts)
  .set({ views: sql`${posts.views} + 1` })
  .where(eq(posts.id, 1));
const [gone] = await db.delete(users)
  .where(eq(users.id, 1)).returning();
```

Without .where() every row changes. set() takes only the columns to change.

12 Joins & aggregates

```
const rows = await db.select().from(posts)
  .leftJoin(users,
    eq(posts.authorId, users.id));
const stats = await db.select({
  authorId: posts.authorId, total: count(),
}).from(posts).groupBy(posts.authorId)
  .having(gt(count(), 5));
```

leftJoin makes right-side columns nullable; innerJoin, rightJoin and fullJoin also exist. sum() and avg() return strings.

13 Relations

```
export const usersRel = relations(users,
  ({ many }) => ({ posts: many(posts) }));
export const postsRel = relations(posts,
  ({ one }) => ({
    author: one(users, {
      fields: [posts.authorId],
      references: [users.id],
    }),
  }));
```

ORM-only: no DB constraint. One-to-one uses one() on both sides; many-to-many, a junction table.

14 Relational queries

```
const list = await db.query.users.findMany({
  columns: { id: true, name: true },
  with: { posts: { limit: 5 } },
});
const one = await db.query.users.findFirst({
  where: (u, { eq }) => eq(u.id, 1),
});
```

Needs schema passed to drizzle(). Nested with can filter, order and limit. findFirst returns a row or undefined.

15 Conditional filters

```
const filters: SQL[] = [];
if (q) filters.push(ilike(users.name, q));
if (min) filters.push(gt(users.age, min));
await db.select().from(users)
  .where(and(...filters));
```

A second .where() replaces the first, even after $dynamic(). Collect conditions and pass and(...filters).

16 Transactions

```
await db.transaction(async (tx) => {
  await tx.insert(users).values(alice);
  if (!allowed) tx.rollback();
  await tx.transaction(async (tx2) => {
    await tx2.insert(posts).values(post);
  });
});
```

Any error rolls back; tx.rollback() throws to abort. A nested tx.transaction() is a savepoint.

17 Prepared statements & raw SQL

```
const byId = db.select().from(users)
  .where(eq(users.id, placeholder('id')))
  .prepare('user_by_id');
const user = await byId.execute({ id: 1 });
const res = await db.execute(
  sql`SELECT * FROM users WHERE id = ${id}`,
);
```

Interpolated values become bound parameters.

18 Migrations

generate	SQL files from schema changes
migrate	Apply pending SQL files
push	Sync the DB directly, no files
studio	Open the Drizzle Studio GUI
check	Check migration consistency

Run as npx drizzle-kit <command>. Use push for prototyping; in production commit the SQL and migrate, or call migrate() from drizzle-orm/node-postgres/migrator.