endel / **zodgres** Public

Code | Issues 3 | Pull requests | Actions | Projects | Security | Insights

2 Branches | 0 Tags | Go to file | Add file | Code

endel refactor 'unique()' extension to … ✓

e13b0ba · 5 months ago

.github/…	docs	5 months ago
.vscode	wip using zod …	5 months ago
docs	improve .updat…	5 months ago
src	refactor 'uniqu…	5 months ago
test	refactor 'uniqu…	5 months ago
.gitignore	bump version	5 months ago
LICENSE	add README	5 months ago
READM…	missing PGLite…	5 months ago
TODO	add tests for J…	5 months ago
favicon.…	update README	5 months ago
package…	pglite test not p…	5 months ago
package…	refactor 'uniqu…	5 months ago
tsconfig.…	fix tsconfig	5 months ago
zodgres…	update README	5 months ago

About

Postgres.js + Zod

zodgres.dev

Readme

MIT license

Activity

19 stars

0 watching

0 forks

Report repository

Releases

No releases published

Packages

No packages published

Languages

TypeScript 100.0%

README | MIT license

TypeScript-first database collections with static type inference and automatic migrations. Built on top of Postgres.js and Zod.

- 🔒 **Type-safe** - Full TypeScript support with Zod schema validation
- 🚀 **Simple API** - Collection-based interface for common database operations
- 📦 **Flexible** - Works with Postgres or in-memory PGLite for testing
- ⚡ **SQL Templates** - Use template literals for complex queries
- 🔄 **Auto-migration** - Automatic table creation from Zod schemas

> ⚠️ **Disclaimer**: PGLite support is currently not working. See issue #1 for more details.

Installation

```
npm install zodgres
```

Quick Start

```
import { connect, z } from 'zodgres';

// Set-up database connection
const db = connect('postgres://user:password@localhost:5432/mydb');

// Define a collection with Zod schema
const users = db.collection('users', {
  id: z.number().optional(), // auto-incrementing
  name: z.string().max(100),
  age: z.number().min(0).max(100).optional(),
});

// Open the connection and run collection migrations
await db.open();

// Create records
const user = await users.create({ name: 'John Doe', age: 30 });
// Result: { id: 1, name: 'John Doe', age: 30 }

// Create multiple records
const newUsers = await users.create([
  { name: 'Alice' },
```

```
  { name: 'Bob', age: 25 }
]);
// Result: [{ id: 2, name: 'Alice' }, { id: 3, name: 'Bob', age: 25 }]

// Query records
const allUsers = await users.select(); // or users.select`*`
const adults = await users.select`* WHERE age >= ${18}`;

// Close connection
await db.close();
```

API Overview

Database Connection

`connect(uri, options?)`

Connect to a Postgres database or use in-memory storage for testing:

```
// Connect to Postgres
const db = connect('postgres://user:password@localhost:5432/mydb');

// Use in-memory database (great for testing)
const testDb = connect(':memory:');
```

> **Important:** After defining all your collections, you must call `await db.open()` to establish the database connection and run any necessary migrations. This ensures your database schema matches your collection definitions before performing any operations.

Collection Definition

`db.collection(name, schema, params?)`

Create a type-safe collection with Zod schema validation:

```
const items = db.collection('items', {
  id: z.number().optional(),          // auto-incrementing primary key
  name: z.string().max(100),          // required string with max length
  price: z.number().positive(),       // required positive number
  description: z.string().optional(), // optional string
});
```

Collection Operations

`create(data)` / `create(data[])`

Create single or multiple records:

```
// Single record
const item = await items.create({
  name: 'Widget',
  price: 19.99
});

// Multiple records
const newItems = await items.create([
  { name: 'Gadget', price: 29.99 },
  { name: 'Tool', price: 39.99, description: 'Useful tool' }
]);
```

`select()` / `select``query`

Query records using SQL template literals:

```
// Select all records
const all = await items.select();

// Select with conditions
const expensive = await items.select`* WHERE price > ${25}`;
const byName = await items.select`* WHERE name = ${'Widget'}`;

// Complex queries
const recent = await items.select`
  name, price
  WHERE created_at > ${new Date('2024-01-01')}
  ORDER BY price DESC
  LIMIT ${10}
`;
```

Testing

The library supports in-memory databases for fast testing:

> ⚠️ PGLite support is currently not working. See issue #1 for more details.

```
import { connect, z } from 'zodgres';

describe('My tests', () => {
  let db;

  before(async () => {
    db = await connect(':memory:').open(); // Uses PGLite
  });
```

```
  after(async () => {
    await db.close();
  });

  it('should create users', async () => {
    const users = db.collection('users', {
      id: z.number().optional(),
      name: z.string(),
    });

    const user = await users.create({ name: 'Test User' });
    assert.deepStrictEqual(user, { id: 1, name: 'Test User' });
  });
});
```

Schema Validation

All data is validated using Zod schemas before database operations:

```
const products = db.collection('products', {
  id: z.number().optional(),
  name: z.string().min(1).max(100),
  price: z.number().positive(),
  category: z.enum(['electronics', 'books', 'clothing']),
  tags: z.array(z.string()).optional(),
  metadata: z.record(z.any()).optional(),
});

// This will throw validation error
await products.create({
  name: '', // too short
  price: -10, // not positive
  category: 'invalid' // not in enum
});
```