
Python

unknown

Nov 26, 2022

DOCUMENTATION

1	Introduction	3
1.1	Packages	3
2	API	5
2.1	AttributeDefinition	5
2.2	AttributeOptions	5
2.3	AttributesDefinition	6
2.4	ForeignKeyActions	6
2.5	ForeignKeyOptions	6
2.6	IndexAttributes	7
2.7	IndexDefinition	7
2.8	IndexOptions	7
2.9	IndexesDefinition	8
2.10	Method	8
2.11	Methods	8
2.12	Model	9
2.13	ModelAttribute	9
2.14	ModelOptions	9
2.15	Sedentary	11
2.16	SedentaryOptions	13
2.17	Type	14
2.18	TypeDefinition	14
3	Development	15

The ORM which **never** needs to migrate.

INTRODUCTION

This package is designed **to make easy the process of applying changes to the database after model definition changes**, more than offer a quick and easy *database* access interface. Applying changes to the *database* after releasing a new version of the application is often a frustrating problem, usually solved with migration systems. Applying changes to the *database* during the development stage, often results in a complex sequence of backward and forward steps through the migrations; this process is complicated more and more especially when working in team with concurrent changes to the models (or the *database* schema). This package tries to solve these problems all in once.

1.1 Packages

The project is organized with a base package *sedentary* itself plus a package specialized for each *database* engine. Nowadays only the *PostgreSQL* specialization package is provided.

1.1.1 *sedentary*

The base package. **It must not be used directly**: it does not support any DB engine.

- NPM [sedentary](#) package
- GitHub [sedentary](#) repository

1.1.2 *sedentary-mysql*

NOT SCHEDULED YET

The *MySQL* implementation package.

- NPM [sedentary-mysql](#) package
- GitHub [sedentary-mysql](#) repository

1.1.3 sedentary-pg

The PostgreSQL implementation package.

- NPM [sedentary-pg package](#)
- GitHub [sedentary-pg repository](#)

1.1.4 sedentary-sqlite

NOT SCHEDULED YET

The SQLite implementation package.

- NPM [sedentary-sqlite package](#)
- GitHub [sedentary-sqlite repository](#)

2.1 AttributeDefinition

```
type AttributeDefinition = type AttributeDefinition = TypeDefinition | AttributeOptions;
```

Defines an *attribute*, it can be either a *TypeDefinition* or an *AttributeOptions*. If an *TypeDefinition* is used, the *default* value is used for all the not specified *AttributeOptions* attributes.

2.2 AttributeOptions

```
interface AttributeOptions {  
  defaultValue?: Natural;  
  fieldName?: string;  
  notNull?: boolean;  
  type: TypeDefinition;  
  unique?: boolean;  
}
```

Specifies the *options* of an *attribute*.

2.2.1 AttributeOptions.defaultValue

- default: undefined

If specified, defines the *default value* of the *field* at *database* level. See Entries initialization for details.

2.2.2 AttributeOptions.fieldName

- default: undefined

If specified, defines the name of the *field* at *database* level, otherwise the *field* has the same name of the *attribute*. This *option* is useful when a *fields* needs to have a reserved name.

2.2.3 AttributeOptions.notNull

- default: undefined

If `true`, specifies to set a `NOT NULL CONSTRAINT` on the *field* at *database* level.

2.2.4 AttributeOptions.type

- required

Specifies the *type* of the *field* at *database* level. Accepts a *TypeDefinition*.

2.2.5 AttributeOptions.unique

- default: undefined

If `true`, specifies to set a `UNIQUE CONSTRAINT` on the *field* at *database* level.

2.3 AttributesDefinition

```
type AttributesDefinition = { [key: string]: AttributeDefinition };
```

Defines the *attributes* of a *Model*. It is an *Object* where each *key* is the name of the *attribute* and the relative *AttributeDefinition* value is the definition of the *attribute*.

2.4 ForeignKeyActions

```
type ForeignKeyActions = "cascade" | "no action" | "restrict" | "set default" | "set null" | "set on delete cascade";
```

The possible *actions* the database engine has to take in case of deletion or update of the target *record* of the *foreign key*.

2.5 ForeignKeyOptions

```
interface ForeignKeyOptions {  
  onDelete?: ForeignKeyActions;  
  onUpdate?: ForeignKeyActions;  
}
```

Specifies the *options* for a *foreign key*.

2.5.1 ForeignKeyOptions.onDelete

- default: "no action"

The *action* the database engine has to take in case of deletion of the target *record* of the *foreign key*. Accepts a *ForeignKeyActions*.

2.5.2 ForeignKeyOptions.onUpdate

- default: "no action"

The *action* the database engine has to take in case of update of the target *field* of the *foreign key*. Accepts a *ForeignKeyActions*.

2.6 IndexAttributes

```
type IndexAttributes = string[] | string;
```

Specifies the *attributes* of an *index*. Accepts an *Array* of *strings* where each element is the name of an *attribute* of the same *Model*. If the index is on a single *attribute*, its name can be provided as a *string* instead of an *Array*.

2.7 IndexDefinition

```
type IndexDefinition = IndexAttributes | IndexOptions;
```

Defines an *index*, it can be either an *IndexAttributes* or an *IndexOptions*. If an *IndexAttributes* is used, the *default* value is used for all the not specified *IndexOptions* attributes.

2.8 IndexOptions

```
interface IndexOptions {
  attributes: IndexAttributes;
  type?: "btree" | "hash";
  unique?: boolean;
}
```

Specifies the *options* of an *index*.

2.8.1 IndexOptions.attributes

- required

Defines the *attributes* of the *index*. See *IndexAttributes* for details.

2.8.2 IndexOptions.type

- default: "btree"

Defines the *type* of the *index*. Accepted values are: "btree" and "hash".

2.8.3 IndexOptions.unique

- default: false

Defines if the *index* must be a *unique index* or not.

2.9 IndexesDefinition

```
type IndexesDefinition = { [key: string]: IndexDefinition };
```

Specifies the *indexes* on the *table* relative to the *Model*. It is an *Object* where each *key* is the name of the *index* and the relative *IndexDefinition* value is the definition of the *index*.

2.10 Method

```
type Method = () => unknown;
```

Is a *Function* which is mounted as **JavaScript** *method* of the *class Model*.

Warning: Do not use *Arrow Functions* to not override the **this** argument provided by the scope.

2.11 Methods

```
type Methods = { [key: string]: Method };
```

Specifies the **JavaScript** *methods* of the *class Model*. It is an *Object* where each *key* is the name of the *method* and the relative *value* is a *Function* which is the *body* of the *method*.

Note: Some *methods*, when provided, are called by **Sedentary** at specific events. Please check Special methods for more details.

2.11.1 ModelOptions.init

- default: undefined

If provided, it works as the *constructor* [Function](#) does. It will be called when a new `Model()` is created.

Note: **TODO** It is not called for loaded Entries.

2.12 Model

TODO

2.13 ModelAttribute

```
interface ModelAttribute {}
```

This type is only used to reference *attributes* for Foreign Keys. If we write following *model*:

```
const db = new Sedentary();
const Foo = db.model("Foo", { bar: db.INT });
```

the newly created *Foo model* has the `Foo.bar` **ModelAttribute** to be used later to specify a *foreign key* referencing the *bar attribute*:

```
const Baz = db.model("Baz", { bar: db.FKEY(Foo.bar) });
```

2.14 ModelOptions

```
interface ModelOptions {
  indexes?: IndexesDefinition;
  int8id?: boolean;
  parent?: Model;
  primaryKey?: string;
  sync?: boolean;
  tableName?: string;
}
```

Specifies the *options* for the *Model*.

2.14.1 ModelOptions.indexes

- default: {}

Defines the *indexes* of the *Model*. See *IndexesDefinition* for details.

2.14.2 ModelOptions.int8id

- default: false

If `true`, the implicit `id` attribute used as primary key is of type `INT8`, see *Data types* for details.

Note: This option conflicts with *ModelOptions.parent* and *ModelOptions.primaryKey* ones.

2.14.3 ModelOptions.parent

- default: undefined

If provided, defines the *parent* of the *Model*. This reflects both on *classes hierarchy* at **JavaScript** level and on *tables hierarchy* at *database* level. The primary key is inherited as well: neither an implicit `id` attribute is added nor can be specified through *ModelOptions.primaryKey* option.

Warning: Not all the *database engine specialized packages* may support this option.

Note: This option conflicts with *ModelOptions.int8id* and *ModelOptions.primaryKey* ones.

2.14.4 ModelOptions.primaryKey

- default: undefined

The value must be the name of an attribute. If provided, defines the primary key of the *Model*. The implicit `id` attribute is not added to the *Model*.

Note: This option conflicts with *ModelOptions.int8id* and *ModelOptions.parent* ones.

2.14.5 ModelOptions.sync

- default: *SedentaryOptions.sync*

If `false`, *Sedentary* does not sync the *table* associated to the *Model*, it simply checks if the *Model* is compliant with the *table* at *database* level.

2.14.6 ModelOptions.tableName

- default: undefined

If not provided, the name of the *table* is the name of the *Model* (i.e. the `name` argument of the *sedentary.model()* call), otherwise it overrides the default *table* name.

2.15 Sedentary

The base ORM class.

TODO

2.15.1 new Sedentary([options])

- `options?: SedentaryOptions` - default `{}` - The global options.
- returns the *Sedentary* object to interact with the *database*.

Warning: Do not use this constructor directly.

2.15.2 new SedentaryPG(config[, options])

- `config: pg.PoolConfig` - required - The connection configuration object.
- `options?: SedentaryOptions` - default `{}` - The global options
- returns the *SedentaryPG* object to interact with the *database*.

SedentaryPG uses `pg.Pool` to connect to the *database*; please refer to `pg` and its [pg-documentation](#) for details about the `config` object.

2.15.3 sedentary.connect([sync])

- `sync: boolean` - default `false` - Specifies whether to execute the **sync process** or not.
- returns a `Promise` which resolves with `void`.

Connects to the *database* and eventually syncs the schema. The value of the `sync` argument is ignored unless the *autoSync* option was set to `false` when *new Sedentary* was called.

Note: Must be called only once.

2.15.4 `sedentary.end()`

- returns a [Promise](#) which resolves with `void`.

Closes the connection with the *database*.

Note: Must be called only once, after `sedentary.connect()`.

2.15.5 `sedentary.model(name, fields[, options [, methods]])`

- **name:** [string](#) - required - The name of the model.
- **fields:** [AttributesDefinition](#) - required - The object with the fields definitions.
- **options?:** [ModelOptions](#) - default `{}` - The options of the model.
- **methods?:** [Methods](#) - default `{}` - The **JavaScript** level *methods* of the model.
- returns a new [class Model](#) to interact with the TABLE.

Defines one model. Should be called once for each model/TABLE to be configured.

Note: Must be called before `sedentary.connect()`.

2.15.6 `sedentary.DATETIME()`

- returns a [DATETIME Type](#).

It is the Type function to specify DATETIME as type for a *field*.

2.15.7 `sedentary.FKEY(attribute, options)`

- **attribute:** - [Model](#) | [ModelAttribute](#) - required - The *foreign key* target *attribute*.
- **options:** - [ForeignKeyOptions](#) - default `{}` - The *foreign key* options.
- returns the [Type](#) of the target *attribute*.

It is the Type function to specify a foreign key. It can be either [Model](#) or a [ModelAttribute](#). If a [Model](#) is provided, its primary key is the target *attribute*.

2.15.8 `sedentary.INT(size)`

- **size:** [number](#) - default: 4 - The *size* of the *field* at *database* level.
- returns an [INT Type](#).

It is the Type function to specify INT as type for a *field*. If the value of the *size argument* is 2, a 16 bit [INT Type](#) is returned; if 4, a 32 bit [INT Type](#) is returned; no other values are accepted.

2.15.9 sedentary.INT8

- returns an INT *Type*.

It is the Type function to specify 64 bit INT as type for a *field*. It is a distinct Type function from *sedentary.INT* to give the *attribute* a specific type at **TypeScript** level. **TODO**

2.15.10 sedentary.VARCHAR(size)

- **size**: *number* - default undefined- The *size* of the *field* at *database* level.
- returns an VARCHAR *Type*.

It is the Type function to specify VARCHAR as type for a *field*. If a value of the *size argument* is provided, it is the maximum allowed string size at *database* level.

2.16 SedentaryOptions

```
interface SedentaryOptions {
  autoSync?: boolean;
  log?: ((message: string) => void) | null;
  sync?: boolean;
}
```

Specifies the options for the *Sedentary* object.

2.16.1 SedentaryOptions.autoSync

- default: `true`

If `false`, the *sedentary.connect* method does not perform the **sync process** by default. This is usefull for distributed environments where we probably don't want to run the **sync process** at each *sedentary.connect* call, but we want to run it only once.

2.16.2 SedentaryOptions.log

- default: `console.log`

The *Function* which *Sedentary* will use to log its messages. If `null`, logging is disabled.

log(message)

- **message**: *string* - required - The message *Sedentary* needs to log.
- returns `void`.

2.16.3 SedentaryOptions.sync

- default: true

If false, *Sedentary* will not sync the *database*, it simply checks if the configured *Models* are compliant to the *tables* at *database* level.

2.17 Type

```
interface TypeDefinition {  
    // black box  
}
```

Defines a *type* for a *field* at *database* level. It is the return value of Type Functions. See Data types for details.

2.18 TypeDefinition

```
type TypeDefinition = (() => Type) | Type;
```

Defines a *type* for a *field* at *database* level, it can be either a Type Function (a *Function* which returns a *Type*) or a *Type* (the return value of a Type Function). If a Type Function is used, the *default* value is used for all the not specified *arguments*. See Data types for details.

DEVELOPMENT

Due to the organization of the *Packages*, probably any change will requires appropriate changes on the *sedentary package* itself and on some *DB engine dedicated extension* as well.

In order to do that, the *database engine dedicated extensions* repositories are added as `git` submodule of the *sedentary package* repository.

Some `make` target have been added to support development of the packages together:

- `make [all]` - performs the basic setup (`npm install`, `npm link`, and so on ...) on all the packages
- `make clean` - removes TypeScript produced files
- `make commit MESSAGE=""` - performs `git add .` and `git commit -m $MESSAGE` in all the git repositories
- `make coverage` - performs `npm coverage` on all the packages
- `make diff` - performs `git diff` in all the git repositories
- `make doc` - builds this documentation locally: requires `sphinx`
- `make outdated` - runs `npm outdated` on all the packages
- `make pull` - performs `git pull` in all the git repositories
- `make push` - performs `git push` in all the git repositories
- `make status` - performs `git status` in all the git repositories
- `make test` - performs `npm test` on all the packages
- `make version VERSION=""` - changes the versions, commits, tags and publishes everything

Both the `test` and the `coverage` targets require to access a *database*: depending on the packages in the development workspace a connection parameter may be required. The connection parameters are the string representation of the JSON object that should be passed to the `connect` method.

- **sedentary-pg: SPG**
 - `make coverage SPG='{"user":"postgres","password":"postgres"}'`
 - `make test SPG='{"user":"postgres","password":"postgres"}'`