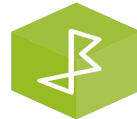
SQL en langage de prog

Introduction à Sequelize ORM

S3 - R3.07

UNIVERSITé DE FRANCHE-COMTé IUT Belfort-Montbéliard

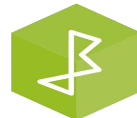


PostgreSQL node JS

Qu'est-ce que Sequelize?

- Sequelize (également connu sous le nom de SequelizeJS) est un framework ORM qui permet de connecter et de faire correspondre votre application Node.js à une base de données.
- Sequelize est en développement depuis 2010 par **Sascha Depold** et est largement utilisé au sein des entreprises Fortune 100.
- Au fil des ans, le framework est passé à près de 25 000 stargazers sur GitHub, avec plus de 900 contributeurs, et est utilisé par plus de 300 000 projets open source.
- Sequelize a été testé pour ses performances et sa sécurité pendant plus d'une décennie et a fonctionné sans problème pour les grands magasins de détail (retail stores) et les agences Web (telles que Walmart et Bitnami), même pendant les périodes de trafic les plus élevées de l'année.
- Ce qui a commencé comme une thèse de master s'est transformé en un élément constitutif majeur de l'écosystème de Node.js.

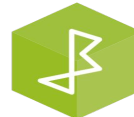
UNIVERSITé DE FRANCHE-COMTé IUT Belfort-Montbéliard



PostgreSQL node JS

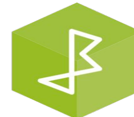
Object-Relational Mapping (ORM)

Une couche entre l'application et la persistance



Object-Relational Mapping (ORM)

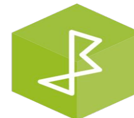
- Un ORM est une méthodologie d'association de structures de base de données et d'informations à l'aide de décorations et de modèles orientés objet (OO).
- Le but d'un ORM est d'aider à atténuer les différences entre les SGBD et d'offrir une certaine forme d'abstraction pour interroger et manipuler les données de manière plus ergonomique.
- En règle générale, un ORM est également livré avec des fonctions d'assistance pour aider à gérer l'état des connexions, la pré-validation des données et les flux de travail.



Avantages et inconvénients de l'ORM

Avantages des ORM :

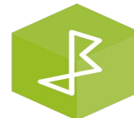
- Le modèle de données est défini en un seul endroit, ce qui est plus facile à mettre à jour et à maintenir, et est également propice à la réutilisation du code.
- ORM dispose d'outils prêts à l'emploi et de nombreuses fonctions peuvent être complétées automatiquement, telles que la validation des données, le prétraitement, les transactions, etc.
- Cela vous oblige à utiliser l'architecture MVC, ORM permet de rendre le code plus clair.
- Le code métier basé sur ORM est relativement simple, avec moins de code, une bonne sémantique et facile à comprendre.
- Vous n'êtes pas obligé d'écrire du SQL.



Avantages et inconvénients de l'ORM

Inconvénients des ORM :

- La bibliothèque ORM n'est pas un outil léger, son apprentissage et sa configuration demandent beaucoup d'efforts.
- Pour les requêtes complexes, ORM est soit incapable de fonctionner efficacement, soit les performances ne sont pas aussi bonnes que le SQL natif.
- ORM fait abstraction de la couche de base de données et les développeurs ne peuvent pas comprendre les opérations de base de données sous-jacentes, ni personnaliser certains SQL spéciaux.



Sequelize ORM

- Le framework suit une approche basée sur les promesses, qui permet aux programmeurs d'invoquer des données de manière asynchrone. L'approche basée sur les promesses offre un moyen plus pratique de gérer les valeurs renvoyées, ou les erreurs, dans votre application sans attendre que le ou les résultats reviennent immédiatement.

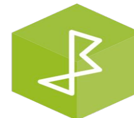
Évitez d'avoir à écrire du SQL brut

```
SELECT * FROM Users WHERE id = 4
```

```
Users.find(4)
  .then(user => {...})
  .catch(err => {...})
```

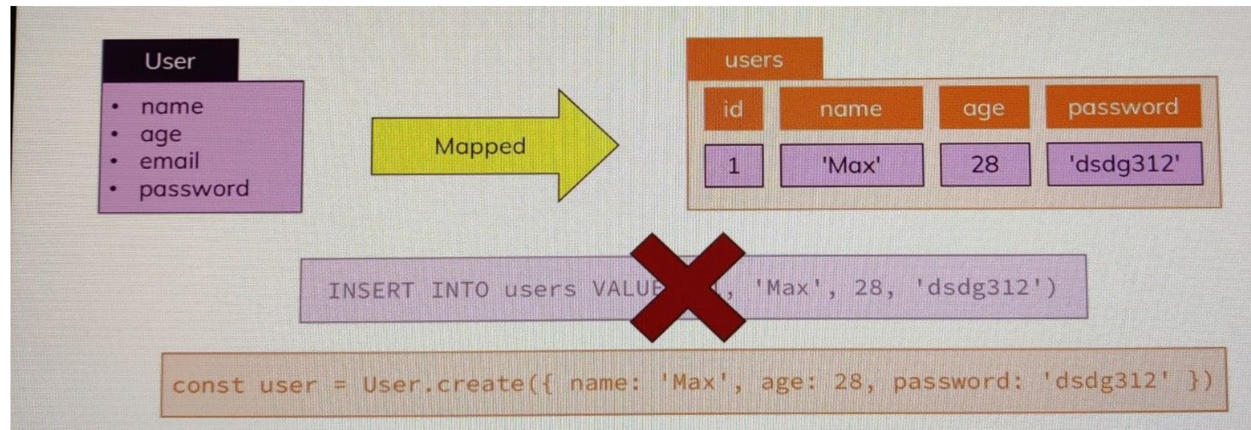
Sequelize ORM

- Entièrement écrit en JavaScript.
- Version actuelle : v7 - alpha.
- Sequelize prend en charge les SGBD suivants : MySQL, MariaDB, Postgres, Microsoft SQL Server (MSSQL), Snowflake, Database 2 (DB2) et SQLite. Un ORM offre plus qu'un simple connecteur à votre base de données. Les ORM offrent souvent des fonctionnalités telles que les suivantes :
  - Outils de migration de schémas et de données *(Tooling for migrating schemas and data)*
  - Prise en charge des adaptateurs/plugins *(Adapter/plugin support)*
  - Regroupement de connexions *(Connection pooling)*
  - Chargement rapide des données *(Eager loading of data)*
  - Transactions gérées *(Managed transactions)*



Sequelize ORM

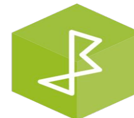
- Sequelize est une bibliothèque de mappage relationnel d'objets, ce qui signifie simplement qu'elle fait tout le code SQL en arrière-plan pour nous et le mappe en objets javascript avec des méthodes pratiques que nous pouvons appeler.
- Exécutez du code SQL en arrière-plan afin que nous n'ayons jamais à écrire de code SQL par nous-mêmes.
- Lorsque nous créons un nouvel utilisateur par exemple, nous appelons simplement une méthode sur cet objet javascript utilisateur et sequelize exécute la requête SQL.



UNIVERSITé DE FRANCHE-COMTé
IUT Belfort-Montbéliard
PostgreSQL
node JS

Approche code d'abord (**code-first**)

- Domain-Driven Design (DDD) est un ensemble de principes et de modèles qui aident les développeurs à créer des systèmes d'objets élégants.
  - Correctement appliqué, il peut conduire à des abstractions logicielles appelées modèles de domaine. Ces modèles encapsulent une logique métier complexe, comblant le fossé entre la réalité métier et le code.
- Code-First est principalement utile dans la conception pilotée par domaine (DDD). Dans l'approche Code-First, vous vous concentrez sur le domaine de votre application et commencez à créer des classes pour votre entité de domaine plutôt que de concevoir d'abord votre base de données, puis de créer les classes qui correspondent à la conception de votre base de données.



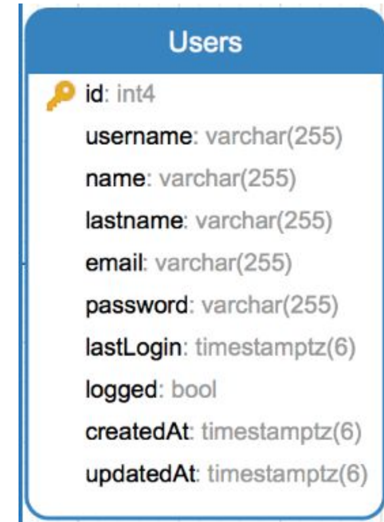
Approche code d'abord (**code-first**)

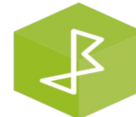
- Les principaux avantages de l'utilisation de code-first sont :
  - **Vitesse de développement** - Vous n'avez pas à vous soucier de la création d'une base de données, vous commencez simplement à coder. Bon pour les développeurs venant d'un milieu de programmation sans beaucoup d'expérience DBA (DataBase Administration). Il dispose également de mises à jour automatiques de la base de données, de sorte que chaque fois que vous modifiez le modèle, la base de données est également automatiquement mise à jour.

Approche code d'abord (**code-first**)

- Les principaux avantages de l'utilisation de code-first sont :
  - **POCO (Plain Old Class Object) -** Le code est beaucoup plus propre, vous ne vous retrouvez pas avec des charges de code généré automatiquement. Vous avez le contrôle total de chacune de vos classes.

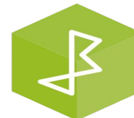
```javascript
module.exports = (sequelize, DataTypes) => {
  const User = sequelize.define('User', {
    username: DataTypes.STRING,
    name: DataTypes.STRING,
    lastname: DataTypes.STRING,
    email: DataTypes.STRING,
    password: DataTypes.STRING,
    lastLogin: DataTypes.DATE,
    logged: DataTypes.BOOLEAN,
    lastPassword: DataTypes.STRING
  }, {})

  return User
}
```





Approche base de données d'abord (**database-first**)

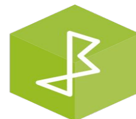
- Dans l'approche Database First, la base de données et les tables associées sont créées en premier. Après cela, vous pouvez créer des modèles de données d'entité à l'aide de la base de données. Il est plus facile de créer une base de données, car plusieurs options sont disponibles en utilisant des interfaces utilisateur graphiques.
- Elle est préférée pour les applications volumineuses et étendues axées sur les données.
- Plus facile de créer des clés et des relations sans écrire de code supplémentaire pour cela.
- Très populaire si vous avez une base de données conçue par des DBA, développée séparément ou si vous avez une base de données existante.
- S'il y a un changement dans la base de données, la classe de modèle doit être étendue avec les mêmes propriétés. ✖
- La création et la gestion de clés et de relations nécessitent plus de codage. ✖



Installation

- Sequelize est disponible via npm

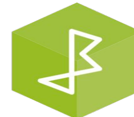
```
$ npm i -S sequelize

# And one of the following:
$ npm i -S pg pg-hstore
$ npm i -S mysql // For both mysql and mariadb dialects
$ npm i -S sqlite3
$ npm i -S tedious // MSSQL
```

Configuration d'une connexion

-

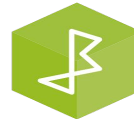
```
const sequelize = new Sequelize('database', 'username', 'password', {
  host: 'localhost',

  // choose anyone between them
  dialect: 'mysql'|'mariadb'|'sqlite'|'postgres'|'mssql',

  // To create a pool of connections
  pool: {
    max: 5,
    min: 0,
    idle: 10000
  },

  // For SQLite only
  storage: 'path/to/database.sqlite'
});
```

Modèles

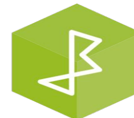
```
const User = sequelize.define('User', {
    username: DataTypes.STRING,
    name: DataTypes.STRING,
    lastname: DataTypes.STRING,
    email: DataTypes.STRING,
    password: DataTypes.STRING,
    lastLogin: DataTypes.DATE,
    logged: DataTypes.BOOLEAN,
    lastPassword: DataTypes.STRING
})
```

Représente des tableaux



Modèles

- Les modèles sont l'essence de Sequelize. Un modèle est une abstraction qui représente une table dans votre base de données. Dans Sequelize, c'est une classe qui étend **Model**.
- Le modèle indique à Sequelize plusieurs informations sur l'entité qu'il représente, telles que le nom de la table dans la base de données et les colonnes qu'elle contient (et leurs types de données).
- Un modèle dans Sequelize a un nom. **Ce nom ne doit pas nécessairement être le même que celui de la table qu'il représente dans la base de données**. Habituellement, les modèles ont des noms au singulier (tels que **User**) tandis que les tables ont des noms au pluriel (tels que **users**), bien que cela soit entièrement configurable.



Modèles

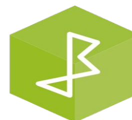
```
// Allow null, default value
flag: { type: Sequelize.BOOLEAN, allowNull: false, defaultValue: true}

// Unique values
someUnique: {type: Sequelize.STRING, unique: true}

// Primary Key
id: { type: Sequelize.INTEGER, primaryKey: true}

// autoIncrement
increment: { type: Sequelize.INTEGER, autoIncrement: true }

// comments - only for MySQL and PG
hasComment: { type: Sequelize.INTEGER, comment: "Comment!" }
```

Vous pouvez définir des options sur chaque colonne

Modèles

```
Sequelize.STRING(1234)              // VARCHAR(1234)
Sequelize.TEXT                      // TEXT
Sequelize.INTEGER                   // INTEGER
Sequelize.FLOAT(11, 12)             // FLOAT(11,12)
Sequelize.DECIMAL(10, 2)            // DECIMAL(10,2)
Sequelize.DATE                      // DATETIME for mysql/sqlite; TIMESTAMP
                                    // WTZ Postgres
Sequelize.BOOLEAN                   // TINYINT(1)
Sequelize.ENUM('value 1', 'value 2') // ENUM
Sequelize.UUID                      // UUID for PostgreSQL and
```

Types de données

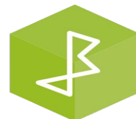


Association

- Vous pouvez spécifier des associations entre plusieurs modèles.
- Créer automatiquement des index pour contrôler la relation entre les tables.
- Sequelize prend en charge les associations standard :

One-To-One
One-To-Many
Many-To-Many

UNIVERSITÉ DE FRANCHE-COMTÉ IUT Belfort-Montbéliard



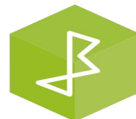
Association

One-To-One Association

- Connectez une source avec exactement une cible.
- Méthodes belongsTo() and hasOne().

```
const User = sequelize.define('User', {/* ... */});
const Project = sequelize.define('Project', {/* ... */});

// One-way associations, from Project to User
Project.hasOne(User);
// To get the association working from User to Project, we need to do this:
User.belongsTo(Project);
```

UNIVERSITé DE FRANCHE-COMTé IUT Belfort-Montbéliard



PostgreSQL node JS

Association

One-To-Many Association

- Connectez une source à plusieurs cibles.
- Les cibles sont cependant connectées à une seule source.

```
const Company = sequelize.define('Company', {/* ... */});
const Product = sequelize.define('Product', {/* ... */});

Company.hasMany(Product);
Product.belongsTo(Company);
```

Association

Many-To-Many Association

- Connectez plusieurs ressources à plusieurs cibles

```
Project.hasMany(User, { as: 'Workers' });
User.hasMany(Project);
```

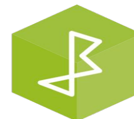
Sequelize créera la table de jointure ‘Workers’

Association

Many-To-Many Association

- Vous pouvez spécifier le nom d'une table de jointure existante dans votre base de données.

```
Project.hasMany(User, {through: 'project_has_users'});
User.hasMany(Project, {through: 'project_has_users'});
```

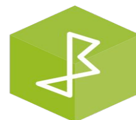



Association

Many-To-Many Association

- Vous pouvez également ajouter des attributs à la table de jointure.

```
UserProjects = sequelize.define('UserProjects', {status: DataTypes.STRING});

User.hasMany(Project, { through: UserProjects });
Project.hasMany(User, { through: UserProjects });
```

Association

Associer des objets

- Parce que Sequelize fait beaucoup de magie, vous devez appeler Sequelize.sync après avoir défini les associations !

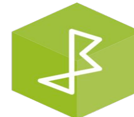
```
await sequelize.sync();
console.log("All models were synchronized successfully.");
```

```
Project.hasMany(Task)
Task.belongsTo(Project)

Project.create()...
Task.create()...
Task.create()...

// save them... and then:
project.setTasks([task1, task2]).then(() => {
  // saved!
})

// ok, now they are saved... how do I get them later on?
project.getTasks().then(associatedTasks => {
  // associatedTasks is an array of tasks
})

// You can also pass filters to the getter method.
// They are equal to the options you can pass to a usual finder method.
project.getTasks({ where: 'id > 10' }).then(tasks => {
  // tasks with an id greater than 10 :)
})

// You can also only retrieve certain fields of a associated object.
project.getTasks({attributes: ['title']}).then(tasks => {
  // retrieve tasks with the attributes "title" and "id"
})
```

Requêtes utilisant Sequelize

find() ==> renvoie une seule entrée

```
// search for a known id
Project.find(123).then((project) => {});

// search for attributes
Project.find({ where: {title: 'NuxtJS'} }).then((project) => {});

// only select some attributes
Project.find({
  where: {title: 'aProject'},
  attributes: ['id', 'title']
}).then((project) => {});
```

Requêtes utilisant Sequelize

findOrCreate() // Vérifie si un élément existe et sinon, crée-le

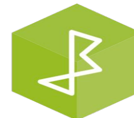
- La fonction de rappel de succès recevra l'utilisateur récupéré/créé.

```
User.findOrCreate(
    { username: 'khriztianmoreno' },
    { fullname: 'Khriztian Moreno' })
}).then((project) => {});
```

Requêtes utilisant Sequelize

findAndCountAll() // Renvoie un objet avec deux propriétés, le nombre total d'enregistrements et un tableau d'objets contenant les enregistrements

```
Product.findAndCountAll(
  { where: { name: {[Op.iLike]: `%${query}%`} } }
).then((result) => { /* result.count and result.rows */ });
```

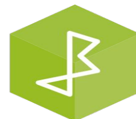
Requêtes utilisant Sequelize

Les opérateurs

- Sequelize fournit plusieurs opérateurs.
- L'option **where** est utilisée pour filtrer la requête. Il existe de nombreux opérateurs à utiliser pour la clause **where**, disponibles sous forme de symboles de l'**Op**.

```
Post.findAll({
  where: {
    [Op.and]: [{ a: 5 }, { b: 6 }],            // (a = 5) AND (b = 6)
    [Op.or]: [{ a: 5 }, { b: 6 }],             // (a = 5) OR (b = 6)
    someAttribute: {
      // Basics
      [Op.eq]: 3,                              // = 3
      [Op.ne]: 20,                             // != 20
      [Op.is]: null,                           // IS NULL
      [Op.not]: true,                          // IS NOT TRUE
      [Op.or]: [5, 6],                         // (someAttribute = 5) OR (someAttribute =
6)
      // Using dialect specific column identifiers (PG in the following example):
      [Op.col]: 'user.organization_id',        // = "user"."organization_id"

      // Number comparisons
      [Op.gt]: 6,                              // > 6
      [Op.gte]: 6,                             // >= 6
      [Op.lt]: 10,                             // < 10
      [Op.lte]: 10,                            // <= 10
      [Op.between]: [6, 10],                   // BETWEEN 6 AND 10
      [Op.notBetween]: [11, 15],               // NOT BETWEEN 11 AND 15
    }
  }
});
```

UNIVERSITé DE FRANCHE-COMTé IUT Belfort-Montbéliard



PostgreSQL

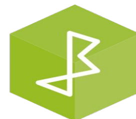
node js

Requêtes utilisant Sequelize

findAll() // Recherche plusieurs éléments dans la base de données

```
Product.findAll().then((products) => {
  // returns all instances of Products
});

Product.findAll({
  attributes: ['id', 'name', 'categories', 'price'],
  where: {
    companyId: company
  },
  limit: 5
}).then(products) => {  });
```

UNIVERSITé DE FRANCHE-COMTé IUT Belfort-Montbéliard



PostgreSQL node JS

Références

- https://sequelize.org/docs/v6/getting-started/
- 💻 https://github.com/khriztianmoreno/sequelize-101
- https://bestofjs.org/projects/sequelize