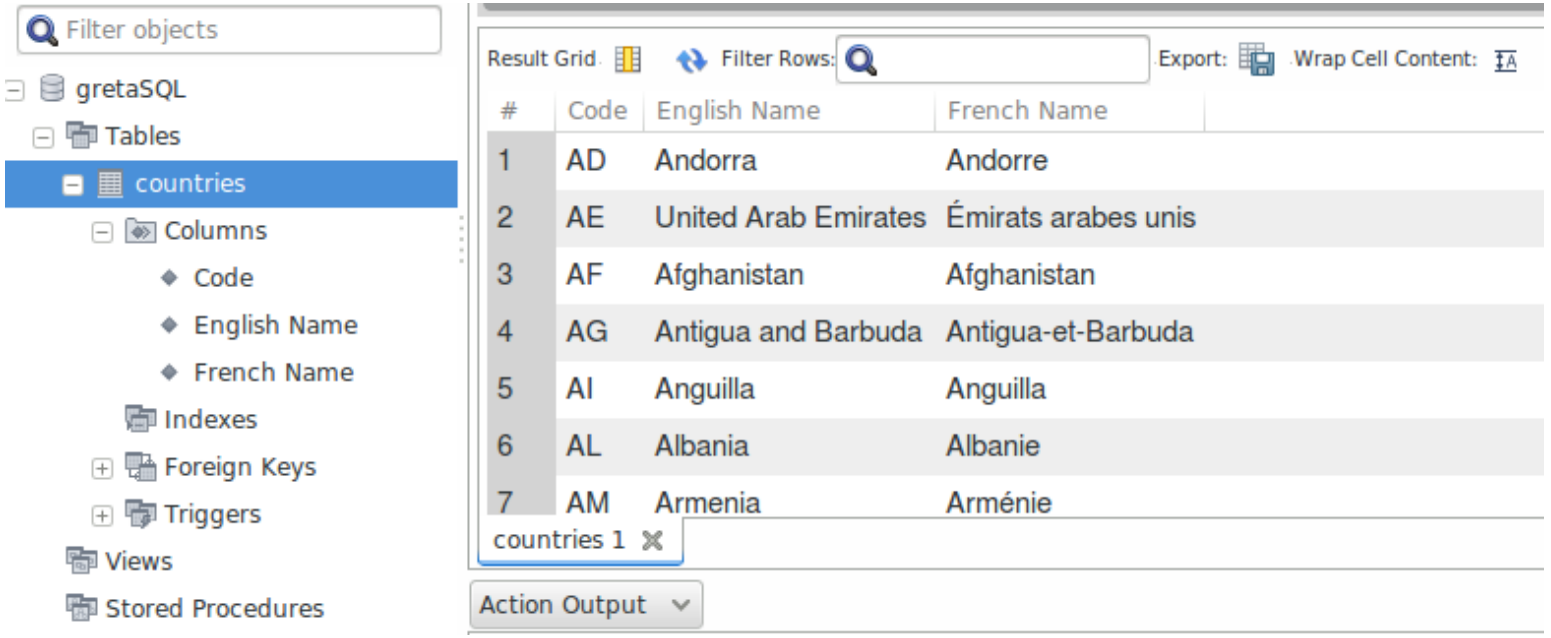
- **Connectivité SQL : Introduction.**
  - Il est possible de se connecter à des bases de données SQL en utilisant les pilotes natifs ou JavaScript comme pour **postgreSQL** ou **MySQL, MariaDB** ou **SQLite :**
    - **node-postgres**
    - **pg**
    - **pg-native**
    - **node-mysql**
    - **node-sqlite (sqlite)**
    - **…**
  - Nous utiliserons ensuite un ORM qui permet de se connecter à l'ensemble de ces bases de données citées plus haut.

- **Ajouter une table dans notre base de données : la table “countries”**
  - Dont les données en csv sont récupérables ici :

    http://mascaron.net/greta/countries_gretasql.csv
  - Vous pouvez utiliser l’importation de données pour la table countries via un logiciel de gestion de votre SGBD : pour MySQL il existe **MySQL Workbench**



Node, Express, MongoDB, et Baseq SQL

- **Pilote MySQL : le module “mysql”**
  - Installez le module pour votre base de données :

```
$ npm install mysql   // driver natif MySQL
```

  - Nous allons créer un fichier nommé “**testDriverSQL.js**” à la racine de notre projet puis l’exécuter dans un terminal : **node ./testDriverSQL.js**

```
var mysql = require('mysql');

var config = {host: 'localhost', user: 'stephane', password: 'azerty', database: 'gretaSQL'};

var connection = mysql.createConnection(config);
connection.connect(function(err) {
    if (err) { console.error('error connecting: ' + err.stack); return; }
    console.log('connected as id ' + connection.threadId);
});

connection.query('SELECT * from countries', function (error, results, fields) {
    if (error) throw error;
    console.log('The first record is: ', results[0]); // results is an array of records
});

connection.end(function (err) {
    console.log('connection mysql closed !');
});
```

- **Connectivité SQL via ODM : Sequelize**
  - Vous trouverez de la documentation sur le site officiel : http://sequelize.readthedocs.io/en/latest/
  - L'installation se fait via npm :

```
$ npm install --save sequelize
// Puis au choix en fonction de la base sur laquelle vous devez vous connecter
$ npm install --save pg pg-hstore
$ npm install --save mysql2
$ npm install --save sqlite3
$ npm install --save tedious // MSSQL
```

  - Configurer une connexion :

```
var sequelize = new Sequelize('database', 'username', 'password', {
  host: 'localhost',
  dialect: 'mysql'|'sqlite'|'postgres'|'mssql',

  pool: {
    max: 5,
    min: 0,
    idle: 10000
  }});
```

- **Connectivité SQL : Sequelize**
  - Nous allons créer un fichier **testSequelize.js** à la racine de notre projet :

```
var Sequelize = require ("sequelize");

var sequelize = new Sequelize('gretaSQL', 'stephane', 'azerty', {
  host: 'localhost',
  dialect: 'mysql',

  pool: {
    max: 5,
    min: 0,
    idle: 10000
  }
});
sequelize  // la syntaxe est celle d'une Promise
  .authenticate()
  .then(function(err) {
    console.log('Connection has been established successfully.');
  })
  .catch(function (err) {
    console.log('Unable to connect to the database:', err);
  });
```

NB : exécutez ce fichier dans un terminal :

$ node ./testSequelize.js *// vous devez voir le log ...**successfully***

Node, Express, MongoDB et Bases SQL

- **Nous allons réaliser une requête SQL simple sur notre table countries :**
  - Ajoutez ce code pour exécuter une requête SQL :

```
(…)
// Connexion effective à la base de données via la méthode authenticate()
// qui retourne une Promise (.then, .catch)
sequelize
    .authenticate()
    .then(function(err) {
        console.log('Connection has been established successfully.');
        // Requête SQL via l'instance sequelize
        sequelize.query("SELECT * FROM countries", {
                  type: sequelize.QueryTypes.SELECT
             })
             .then(function(countries) {
                  console.log('listes des pays : ', countries);
             })
             .catch(function(err) {
                  console.log('error select', err);
             });
    })
    .catch(function(err) {
        console.log('Unable to connect to the database:', err);
    });
```

- **Nous allons ajouter la capacité de connexion à la base MySQL pour notre Express App:**
  - Pour ce faire nous avons déjà intégré les modules via : **npm install --save sequelize mysql**.
  - Il nous faut donc juste créer la configuration Sequelize dans **appdyn.js** (avant cnx mongoose) et définir la config sequelize en GLOBAL pour pouvoir y accéder dans nos modules:

```
var Sequelize = require("sequelize");

// configuration des paramètres de la connexion
GLOBAL.sequelize = new Sequelize('gretaSQL', 'stephane', 'azerty', {
  host: 'localhost',
  dialect: 'mysql',
  pool: { max: 5, min: 0, idle: 10000 }
});
```

Node, Express, MongoDB et Bases SQL

- **Le code du contrôleur : countriesSQL.js**

```
var express = require('express');
var router = express.Router();

/* GET countriesSQL page. */
router.get('/', function(req, res, next) {
  if ((req.session.passport) && (req.session.passport.user != null)) {
        // Requête SQL via l'instance sequelize
    GLOBAL.sequelize.query("SELECT * FROM countries", {
        type: sequelize.QueryTypes.SELECT
    }).then(function(countries) { // sql query success
        console.log('listes des pays : ', countries);
        res.render('countriesSQL', {
            title: 'List countries from SQL postgreSQL',
            country: countries
        });
    }).catch(function(err) { // sql query error
        console.log('error select', err);
    });
  } else res.redirect('/');
});
module.exports = router;
```

  - A partir d'une structure classique de contrôleur pour notre application ajoutez le code en bleu.

Node, Express, MongoDB et Bases SQL

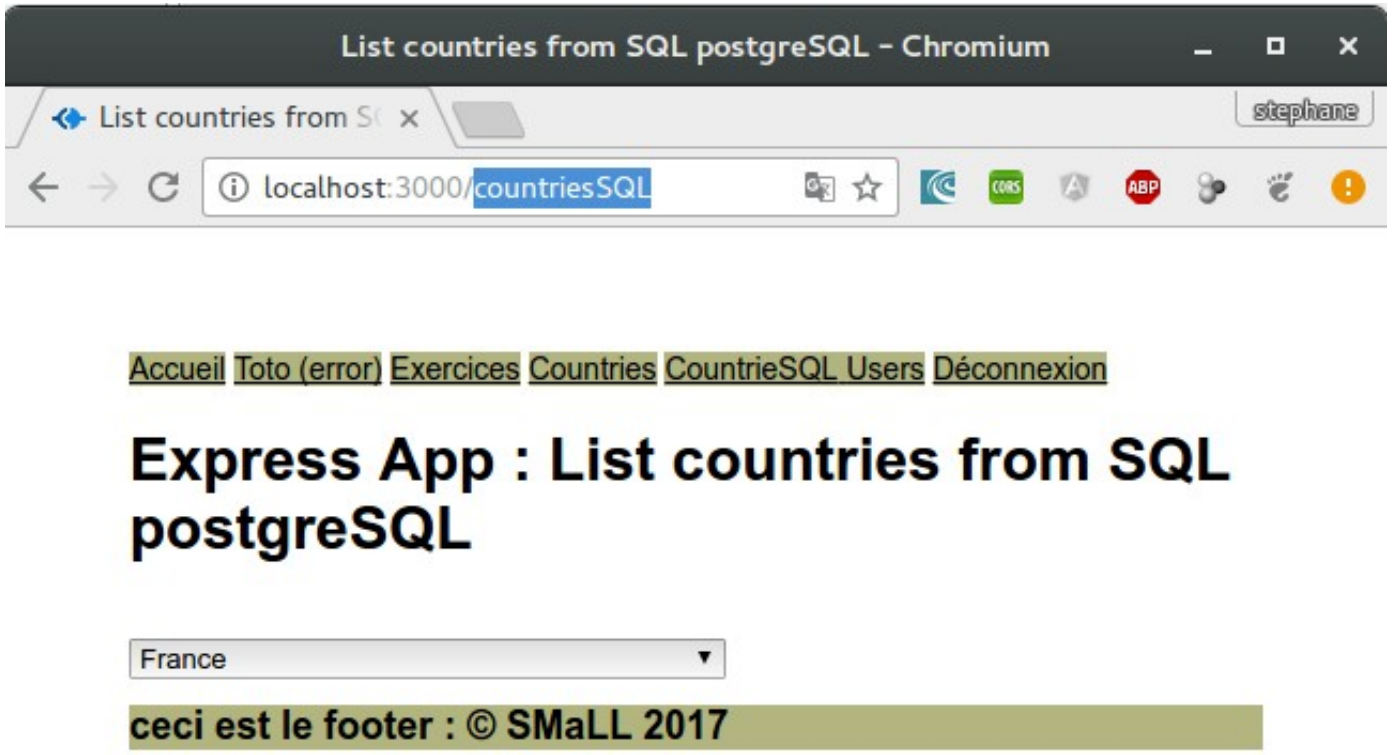
- **Avant de pouvoir tester il faut ajouter la route dans notre config_actions.json :**

```
(…)
"GET/logout": {
    "controler": "logout"
  },
  "GET/countriesSQL": {
    "controler": "countriesSQL"
  }
}
```

- **Puis dans notre barre de navigation des menus “nav.hbs” nous devons ajouter le lien pour exécuter l’action “/countriesSQL” :**

```
<nav class="barnav">
    <ul id="nav">
        <li><a href="/index">Accueil</a></li>
        <li><a href="/toto">Toto (error)</a></li>
        <li><a href="/exos">Exercices</a></li>
        <li><a href="/countries">Countries</a></li>
        <li><a href="/countriesSQL">CountrieSQL</li>
        <li><a href="/users">Users</a></li>
        <li><a href="/logout">Déconnexion</a></li>
    </ul>
</nav>
```

- **Voyons le résultat à l'exécution de notre App Express :**



- **Architecture logicielle :**
  - Nous allons rendre générique le contrôleur qui permet de réaliser les requêtes SQL SELECT :
    - Pour ce faire nous devons modifier le code précédent, et passer en paramètres les chaînes de caractères nécessaires à l'excution d'une requête :
      - **Le nom du contrôleur** est déjà paramétrée via le fichier config_actions.json
      - **La requête SQL** elle aussi est à passer en paramètre
      - **Le nom de la vue** (view handlebars) aussi doit être paramétrée.
    - La question suivante c'est où que nous devons définir ces 2 paramètres supplémentaires, la réponse est assez évidente dans le fichier : **config_actions.json**
    - Nous appellerons "**selectAllSQL.js**" notre contrôleur.

- **Architecture logicielle (suite) :**
  - Voyons comment modifier notre fichier d'actions :

```
(…)
"GET/logout": {
    "controler": "logout"
  },
  "GET/countriesSQL": {
    "controler": "selectAllSQL",
    "view": "countriesSQL",
    "sql_query": "SELECT * FROM countries"
  },
}
```

  - Nous allons devoir récupérer cette varaible "**sql_query**", comme on les charge déjà pour récupérer le controler, la vue, le modèle Mongoose, nous pouvons les ajouter à l'objet requête "**req**", en ajoutant une propriété, nous allons le faire dans le **dynamicRouter.js** :
    - En ajoutant une propriété "**message**" à "**req**"
    - En ajoutant l'**action** au message (= **type**+**path**) :
    - En ajoutant "**view**" et "**sql_query**" dans le message.

- **Voici le code modifié du dynamicRouteur :**

```
(…)
  path = url.parse(req.url).pathname;
  type = req.method;
  // Il faut supprimer pour le routage le param après l'action
  if (path.split('/').length > 0) path = '/' + path.split('/')[1]
  // configuration du message pour les contrôleurs génériques
  req.message = {};
  req.message.action = type + path;
  if (GLOBAL.actions_json[type + path].view)
    req.message.view = GLOBAL.actions_json[type + path].view;
  else
    req.message.view = null;
  if (GLOBAL.actions_json[type + path].sql_query)
    req.message.sql_query = GLOBAL.actions_json[type + path].sql_query;
  else
    req.message.sql_query = null;
  if (typeof GLOBAL.actions_json[type + path] == 'undefined') {
    console.log("Erreur pas d'action " + path + " dans l'annuaire");
    next();
  } else {
    instanceModule = require('./routes/' + GLOBAL.actions_json[type + path].controler);
    router.use(path, instanceModule);
    next();
  }
}
module.exports = dynamicRouter;
```

NB : Mettez en pratique cette factorisation de paramètres.

- **Exercice : dupliquez le contrôleur countriesSQL.js en le nommant “selectAllSQL.js”**
  - Vous devez remplacer les chaînes de caractères de la requête SQL et du nom de la vue par leur équivalent dans **req.message** ajouté par le **dynamicRouteur**.
  - Lancer le serveur et exécutez le code.
  - Pour des raisons de compréhension fonctionnelle de votre code vous devrez changer le noms de certaines variables en les rendant plus génériques.

- **Correction de l'exercice :**
  - Le contrôleur "**selectAllSQL.js**" :

```
var express = require('express');
var router = express.Router();

/* GET database datas from SQL query. */
router.get('/', function(req, res, next) {
  if ((req.session.passport) && (req.session.passport.user != null)) {
    GLOBAL.sequelize.query(req.message.sql_query, {
        type: sequelize.QueryTypes.SELECT
      }) // SQL query success return datas into callback
      .then(function(datas) {
        console.log('listes des datas : ', datas);
        res.render(req.message.view, {
          title: 'List from SQL postgreSQL',
          result: datas
        });
      }) // SQL query error return error into callback
      .catch(function(err) {
        console.log('error select', err);
      });
  } else res.redirect('/');
});
module.exports = router;
```

NB : En bleu les modifications, on place les variables à la place des chaînes de caractères et on renomme les variables spécifiques. Pensez à modifier le template handlebars et utiliser result comme données dans le {{#each result}}...{{/each}}.

- **Vous allez coder un contrôleur nommé “selectByIdSQL.js” :**
  - Il permettra d’ajouter un paramètre dans une clause Where d’une requête. Le paramètre est passé dans l’URL.
  - Nous allons utiliser le champ code de la table countries comme _id. Notre requête SQL sera :

```
SELECT * FROM countries where code = :_id
```

  - Nous pouvons nous inspirer du contrôleur “**modifyUsers.js**” pour le type de route :

```
router.route('/:_id').get(function(req, res) {
```

  - Une requête paramétrée Sequelize nécessite un objet “**options**” avec l’attribut “**replacements**”.

- **Requête paramétrée :**
  - Voici comment initialiser notre objet replacements dans le module contrôleur “**selectByIdSQL.js**” :

```
(…)
    if ((req.session.passport) && (req.session.passport.user != null)) {
        var options = {};
        options.replacements = req.params;
        options.type = sequelize.QueryTypes.SELECT;
(…)
```

  - Nous pouvons maintenant exécuter la requête sur la base de données via la variable globale “**sequelize**” :

```
(…)
GLOBAL.sequelize.query(req.message.sql_query, options)
            // SQL query success
            .then(function(datas) {
                console.log('listes des datas : ', datas);
                res.render(req.message.view, {
                    title: 'List from SQL postgreSQL',
                    result: datas
                });
            }) // SQL query error
            .catch(function(err) { console.log('error select', err);});
(…)
```

- **Contrôleur selectByIdSQL.js :**

```
var express = require('express');
var router = express.Router();

/* GET one record from SQL query by one parameter which it
can be _id or other like field code into countries table. */
router.route('/:_id').get(function(req, res, next) {
    if ((req.session.passport) && (req.session.passport.user != null)) {
        var options = {};
        options.replacements = req.params;
        options.type = sequelize.QueryTypes.SELECT;
        GLOBAL.sequelize.query(req.message.sql_query, options)
            // SQL query success
            .then(function(datas) {
                console.log('listes des datas : ', datas);
                res.render(req.message.view, {
                    title: 'List from SQL postgreSQL',
                    result: datas
                });
            }) // SQL query error
            .catch(function(err) {
                console.log('error select', err);
            });
        // Not authentified redirect login
    } else res.redirect('/');
});
module.exports = router;
```

NB : On remarquera l'enchaînement des fonctions : .then et .catch cela car la query Sequelize est une Promise.

Node, Express, MongoDB et Bases SQL

- **Exercice : utiliser selectByIdSQL.js pour une autre table : “companies” :**

```
CREATE TABLE public.companies
( -- Identifiant en auto incrément smallserial
  id smallint NOT NULL DEFAULT nextval('companies__id_seq'::regclass),
  name text, -- Nom de l'entreprise
  age smallint, -- nombre d'année d'existence de l'entreprise
  adress1 text, -- première ligne d'adresse de l'entreprise
  adress2 text, -- deuxième ligne d'adresse de l'entreprise
  postal_code character(5), -- Code postal de l'entreprise
  city text -- nom de la ville dans laquelle est installée l'entreprise
)
WITH (
  OIDS=TRUE
);
ALTER TABLE public.companies
  OWNER TO <votre_user>;
     COMMENT ON COLUMN public.companies.id IS 'Identifiant en auto incrément smallserial ';
     COMMENT ON COLUMN public.companies.name IS 'Nom de l''entreprise';
     COMMENT ON COLUMN public.companies.age IS 'nombre d''année d''existence de l''entreprise';
     COMMENT ON COLUMN public.companies.adress1 IS 'première ligne d''adresse de l''entreprise';
     COMMENT ON COLUMN public.companies.adress2 IS 'deuxième ligne d''adresse de l''entreprise';
     COMMENT ON COLUMN public.companies.postal_code IS 'Code postal de l''entreprise';
     COMMENT ON COLUMN public.companies.city IS 'nom de la ville dans laquelle elle est installée';
```

Node, Express, MongoDB et Bases SQL

- **Exercice suite : SQL pour injecter des données dans la table companies.**

```
INSERT INTO companies VALUES (3,'SAS Machin Truc',5,'20 rue des bidules','','40000','MONT-DE-MARSAN');
INSERT INTO companies VALUES (4,'SARL Huile de Code',3,'236 route de Canenx',
                              'Parc techno.SO WATT!-La Fabrik', '40000','MONT-DE-MARSAN');
INSERT INTO companies VALUES (5, 'SMALL : Stéphane MASCARON Architecte Logiciels Libres',23,
                              '260 avenue des arènes', '','40090','SAINT-PERDON');
INSERT INTO companies VALUES (6, 'SP Dev : Stéphane PONTEINS',10,'Rue des angles V2','','40100','DAX');
INSERT INTO companies VALUES (7, 'Agence Esens : Laurence MARTIN',15,'13 Rue Léon Bidulle','',
                              '40000','MONT-DE-MARSAN');
```

NB : Utilisez le **code SQL du slide précédent** pour créer une nouvelle table qui va stocker des entreprises, que l'on va appeler "**companies**". A partir des exemples Sequelize précédents, créer un affichage de toutes les "**companies**" et un affichage d'une seule companie filtrer par son "**id**". Puis ajoutez des données dans la nouvelle table companies via les **INSERT** ci-dessus.

Vous devrez créer les actions dans le config_actions.json avec les bons paramètres (view, sql_query et controler)

Node, Express, MongoDB et Bases SQL

- **Sequelize permet également de créer des schémas comme Mongoose.**
  - A partir d'un schéma il est possible d'obtenir une vue objet de notre base de données comme le ferait Hibernate en Java.
  - Pour créer un schéma Sequelize, voici un exemple avec notre table countries.

```
module.exports = function(sequelize, DataTypes) {
  return sequelize.define('countries', {
    code: {
      type: DataTypes.TEXT,
      allowNull: false
    },
    libelle_us: {
      type: DataTypes.TEXT,
      allowNull: false
    },
    libelle_fr: {
      type: DataTypes.TEXT,
      allowNull: true
    }
  }, {
    tableName: 'countries'
  });
};
```

- **Question : Si j'ai une base de données existante de 50 tables pour créer les Schémas je dois le faire à la Main ?**
  - Oui ! ;-) Meuh non !! ;-) il existe un module appelé :

    Sequelize-Auto
  - Installez le en global (-g) dans votre ordinateur

    ```
    npm install -g sequelize-auto
    ```
  - Testez son installation dans un terminal

    ```
    $ sequelize-auto -o "./models" -d gretajs -h localhost -u steph -p 5432 -x azerty -e postgres
    ```

NB : Attention si vous n'avez pas accès, il suffit de créer un lien, raccourcis vers /usr/bin/ de sequelize-auto

- **Dans notre base PostgreSQL nous avons 2 tables : “countries” et “companies” :**
  - Nous allons générer les Schémas via ce module sequelize-auto, pour ce faire placez vous dans le dossier du projet :

```
$ sequelize-auto -o "./models" -d gretajs -h localhost -u steph -p 5432 -x azerty -e postgres
```

  - Cette commande a créée un sous-dossier dans le répertoire du projet nommé “models” qui contient les fichier JavaScript des modèles :

```
stephane@UX303UB:~/workspaceGreta/gretaexos2/models$ ll
total 16
drwxrwxr-x  2 stephane stephane 4096 mars  21 14:42 ./
drwxrwxr-x 11 stephane stephane 4096 mars  21 14:40 ../
-rw-rw-r--  1 stephane stephane  718 mars  21 14:42 companies.js
-rw-rw-r--  1 stephane stephane  378 mars  21 14:42 countries.js
```

Node, Express, MongoDB et Bases SQL

- **Voici le modèle “companies” généré par sequelize-auto**

```
module.exports = function(sequelize, DataTypes) {
  return sequelize.define('companies', {
    _id: { type: DataTypes.INTEGER,
           allowNull: false,
           defaultValue: "nextval(companies__id_seq::regclass)"
    },
    name: { type: DataTypes.TEXT,
            allowNull: true
    },
    age: { type: DataTypes.INTEGER,
           allowNull: true
    },
    adress1: { type: DataTypes.TEXT,
               allowNull: true
    },
    adress2: {
      type: DataTypes.TEXT,
      allowNull: true
    },
    postal_code: {
      type: DataTypes.CHAR,
      allowNull: true
    },
    city: {
      type: DataTypes.TEXT,
      allowNull: true
    }
  }, {
    tableName: 'companies'
  });
};
```

- **Nous devons écrire un chargeur de modèles Sequelize :**
  - Il faut instancier un objet pour chaque schémas générés, comme on l'a fait pour les schémas mongoose. Ajouter ce code dans "**appdyn.js**" :

```
GLOBAL.modelsSeq = {};
// Loader Sequelize models into GLOBAL.modelsSeq
fs.readdirSync(__dirname + '/models')
    .filter(function(file) {
        return (file.indexOf(".") !== 0);   // on filtre les fichiers ils doivent contenir un . (.js)
     .forEach(function(file) {
        var model = GLOBAL.sequelize.import(path.join(__dirname + '/models', file));
        GLOBAL.modelsSeq[model.name] = model;
        console.log('file read : ' + file);
    });
(…)
```

  - Il nous reste maintenant à utiliser un model, comme ils sont stockés dans la variable "**GLOBAL.modelsSeq**", il est possible de l'utiliser dans un contrôleur pour réaliser une requête.

Node, Express, MongoDB et Bases SQL

- **Ecrire un contrôleur qui utilise un model Sequelize pour faire une requête dans la base SQL : “selectAllFromModelSQL.js” :**

```
var express = require('express');
var router = express.Router();

/* GET companies list into a web page. */
router.route('/').get(function(req, res, next) {
    if ((req.session.passport) && (req.session.passport.user != null)) {

        GLOBAL.modelsSeq["companies"].findAll().then(function(datas) {
            res.render(req.message.view, {
                title: 'List from SQL postgreSQL',
                result: datas
            });

        });
    } else res.redirect('/');
});
module.exports = router;
```

NB : Pensez à modifier la déclaration de Sequelize dans appdyn.js avec PostgreSQL si vous n’avez pas de champ createAt, updateAt il faut désactiver les “timestamps”.

- **Créer une action dans le config_actions.json**

```
  },
   "GET/companiesModel": {
       "controler": "selectAllFromModelSQL",
       "view": "companiesSQL"
  },
```

- **Puis ajouter une lien dans la barre de navigation**

```
<nav class="barnav">
    <ul id="nav">
        <li><a href="/index">Accueil</a></li>
        <li><a href="/toto">Toto (error)</a></li>
        <li><a href="/exos">Exercices</a></li>
        <li><a href="/countries">Countries</a></li>
        <li><a href="/countriesSQL">CountrieSQL</a></li>
        <li><a href="/countrySQL/FR">Country by code SQL</a></li>
        <li><a href="/companiesSQL">CompaniesSQL</a></li>
        <li><a href="/companySQL/7">Company by _id SQL</a></li>
        <li><a href="/companiesModel">Company by Model</a></li>
    </ul>
</nav>
```

- **Si vous avez l'erreur suivante :**
  - CreateAt and modifyAt not exist, vous devez modifier la configuration de l'objet sequelize dans appdyn.js (ajoutez le code en bleu)

```
// configuration des paramètres de la connexion SQL Sequelize
var sequelize = new Sequelize('gretajs', 'steph', 'azerty', {
    host: 'localhost',
    dialect: 'postgres',
    define: {
        timestamps: false
    },

    pool: {
        max: 5,
        min: 0,
        idle: 10000
    }
});
```

- **Voyons comment réaliser des jointures avec Sequelize :**
  - Il existe dans les API des fonctions pour réaliser des relations entre les tables :
    - Lorsque vous appelez une méthode telle que **Companies.belongsTo (Countries)**, vous dites que le modèle **Companies** (le modèle **sur lequel la fonction est appelée**) est la source et le modèle **Countries** (le modèle passé comme argument) **est la cible**.
    - Voyons un exemple de code de jointure entre les deux schémas. Pour ce faire nous allons créer un nouveau contrôleur que l'on va appeler :

      selectFromModelWithJoin.js

Node, Express, MongoDB et Bases SQL

- **Contrôleur selectFromModelWithJoin.js**

```
var express = require('express');
var router = express.Router();

/* GET liste of companies with link to countries. */
router.route('/').get(function(req, res, next) {
    if ((req.session.passport) && (req.session.passport.user != null)) {

        GLOBAL.modelsSeq["Companies"].findAll({
            include: [{
                model: GLOBAL.modelsSeq["Countries”],
                keyType: GLOBAL.db.Sequelize.INTEGER
            }]
        }).then(function(datas) {
             console.log(‘Data from Sequelize Model Jointure : ‘, datas);
             res.render(req.message.view, {
                 title: 'List from SQL postgreSQL',
                 result: datas
            });
        });

    } else res.redirect('/');
});
module.exports = router;
```

Node, Express, MongoDB et Bases SQL

- **Modifier appdyn.js pour rendre global la variable Sequelize et sequelize via une variable que l'on appelera db :**

```
/** todo : ici initialiser la connexion à la base postgreSQL via Sequelize */
// Chargement du module sequelize
GLOBAL.db = {};
var Sequelize = require("sequelize");
db.Sequelize = Sequelize;

GLOBAL.modelsSeq = {};
// configuration des paramètres de la connexion SQL Sequelize
var sequelize = new Sequelize('gretajs', 'steph', 'azerty', {
    host: 'localhost',
    dialect: 'postgres',
    define: {
        timestamps: false
    },

    pool: {
        max: 5,
        min: 0,
        idle: 10000
    }
});
db.sequelize = sequelize;
```

Node, Express, MongoDB et Bases SQL

- **Il nous faut créer la relation avec les outils de Sequelize, on va dans appdyn.js ajouter du code :**

```
(…)
// Loader Sequelize models into GLOBAL.db
fs.readdirSync(__dirname + '/models')
    .filter(function(file) {
// Pour exclure du chargement un fichier dans le dossier && (file !== "index.js");
        return (file.indexOf(".") !== 0);
    })
    .forEach(function(file) {
        var model = sequelize.import(path.join(__dirname + '/models', file));
        db[model.name] = model;
        console.log('file read : ' + file);
    });
// CREATION DE L'ASSOCIATION
GLOBAL.modelsSeq["Companies"].belongsTo(GLOBAL.modelsSeq["Countries"], {
    foreignKey: "countrieId",
    keyType: GLOBAL.db.Sequelize.INTEGER
});
(…)
```

NB : Cela suppose que vous avez bien une clé étrangère dans companies nommée : “countrieId”