

---

# Génération universelle de données pour tests logiciels

---

**Projet tuteuré - Rapport final**  
**Licence Professionnelle 2018 - 2019**



---

# Table des matières

<b>1. Sommaire .....</b>	<b>2</b>
<b>2. Introduction .....</b>	<b>3</b>
2.1. Cadre de travail .....	3
2.2. Équipe .....	3
<b>3. Remerciements .....</b>	<b>4</b>
<b>4. Présentation .....</b>	<b>4</b>
4.1. Description du projet .....	4
4.2 Objectif du projet .....	5
<b>5. Notre travail .....</b>	<b>5 - 19</b>
5.1 Analyse de l'existant .....	5 - 7
5.2 Réflexion .....	8
5.3 Répartition des tâches .....	9
5.4 Opérations sur le générateur .....	9 - 15
5.5 Adaptation à l'interface graphique .....	16 - 19
<b>6. Perspective d'améliorations .....</b>	<b>20</b>
<b>7. Conclusion .....</b>	<b>21</b>

---

## **2. Introduction**

### **2.1 Cadre de travail**

Ce projet nous est proposé en fin d'année de Licence Professionnelle orientée métiers de l'informatique (logiciels libres) à l'université d'Angers. Il consiste à guider les étudiants vers des projets de groupe tout en restant en totale autonomie. Sans créneau dans l'emploi du temps consacré à ces projets, c'est aux étudiants de gérer leur temps afin de réaliser ce qui leur a été demandé sur une période d'environ un mois et demi.

La communication et l'entraide est un atout majeur au bon déroulement du projet. On peut considérer comme acquises, les compétences en développement web (de bas niveau) grâce au module « Développement Web », que nous avons étudié tout au long de cette année scolaire et qui nous permettra une mise en oeuvre au sein de ce projet.

Nous décrirons à travers ce rapport, de manière chronologique, notre façon d'avoir abordé ce projet et nos différents choix pour le mener à bien. Il tentera donc de répondre à la question suivante :

**- Quelles ont été les difficultés de ce projet et comment ont-elles été surmontées ?**

### **2.2 L'Équipe**

Amaury Daumy : étudiant à l'université d'Angers depuis 2016, le développement d'application métier est le domaine qu'il préfère. Le développement web n'est pas une technologie qu'il maîtrise bien.

Julien Henry : étudiant à l'université depuis 2015. Le développement Web est l'une des technologies qu'il maîtrise le mieux.

---

### 3. Remerciements

Nous profitons de ce rapport pour remercier Gilles Hunault en tant que créateur de ce projet mais également en tant que tuteur. Ces conseils nous ont très certainement aidé à arriver à un tel résultat aujourd'hui.

Nous tenons également à remercier les anciens étudiants de ce projet qui ont fourni une documentation relativement précise.

### 4. Présentation

#### 4.1 Description du projet

Le paragraphe présentant le projet est la description même apportée par le créateur de ce projet : Gilles Hunault.

Vous retrouvez ce même paragraphe à l'adresse suivante :

<http://forge.info.univ-angers.fr/~gh/Projets/Lp/proj2016.php>

Lorsqu'on écrit un programme informatique ou lorsqu'on conçoit un système d'informations avec des bases de données, on a souvent besoin de jeux de données de tests, parfois très structurés et avec de gros volumes de données.

Il serait très utile de pouvoir disposer d'un générateur avancé de telles données. Le but du projet est de concevoir les types de données et de structures de données liées à des générations automatiques de données pour tests et d'implémenter un tel générateur qui fonctionnera en ligne de commandes et via une interface Web. La configuration de la génération sera décrite dans un fichier XML. Cette génération peut se faire sous différents formats, selon leur utilisation finale. A terme, ce projet serait un outil très utile pour effectuer des tests mais aussi pour peupler une base afin de réfléchir à l'optimisation des requêtes.

---

## 4.2 Objectif du projet

Nous ne sommes pas les premiers à travailler sur ce projet. En effet, c'est sur l'année scolaire 2016-2017 que ce projet a commencé à voir le jour. Il a ensuite été repris par des étudiants en 2017-2018 puis enfin par nous même pour cette année 2018-2019. Il nous avait été remis, un générateur capable de générer des données universelles dans différents formats de sorties telles que le XML, le SQL et enfin le CSV. Ce sont les formats les plus standards du monde de l'informatique. Notre mission a été d'apporter à ce générateur la possibilité de lier des tables de données entre elles. Ce point étant relativement complexe, il fallait veiller à ne pas dégrader l'existant. Une fois cette nouvelle fonctionnalité mise en place, nous devions adapter l'interface graphique de sorte à ce que nous puissions nous servir de la nouvelle fonctionnalité. Nous devions également vérifier si l'existant était stable et rendre l'interface graphique plus facile à utiliser. Enfin, il nous a été proposé de mettre en place un système permettant de générer des données cohérentes. Par exemple, lorsque qu'une personne est générée, si son genre est masculin, il fallait lui attribuer un prénom masculin. Nous avons également tenté de généraliser cette approche afin de permettre de produire des données cohérentes en fonction des dictionnaires disponibles, sans se soucier des cas particuliers.

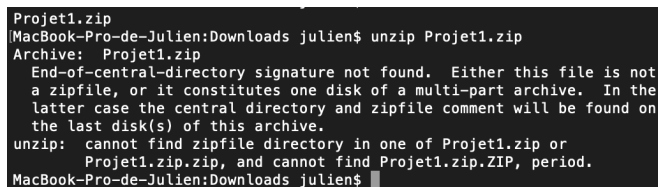
## 5. Notre travail

### 5.1 Analyse de l'existant

Nous avons, dans un premier temps, lu les différents documents remis à notre tuteur de stage par les anciens étudiants, aussi bien la documentation développeur que la documentation utilisateur. Celle-ci était relativement détaillée mais pas forcément structurée de manière cohérente. À cet effet, nous avons rédigé un manuel d'utilisation succinct pour celui qui souhaiterait utiliser cette solution sans avoir de connaissances particulières du projet.

---

Lorsque nous avons téléchargé l'archive, afin d'essayer sur nos propres ordinateurs, nous avons commencé à rencontrer des difficultés. Cependant, nous ne connaissions pas les raisons de celles-ci. Nous vous joignons ci-dessous un premier aperçu de l'archive compressée téléchargée depuis l'interface web sous un serveur apache.



```
Projet1.zip
MacBook-Pro-de-Julien:Downloads julien$ unzip Projet1.zip
Archive:  Projet1.zip
  End-of-central-directory signature not found. Either this file is not
  a zipfile, or it constitutes one disk of a multi-part archive. In the
  latter case the central directory and zipfile comment will be found on
  the last disk(s) of this archive.
unzip:  cannot find zipfile directory in one of Projet1.zip or
        Projet1.zip.zip, and cannot find Projet1.zip.ZIP, period.
MacBook-Pro-de-Julien:Downloads julien$
```

*Figure 1 : capture d'écran d'un ZIP généré par l'interface web.*

Il est également arrivé, parfois, de se retrouver avec une archive vide malgré un remplissage complet du formulaire de l'interface web. En revanche, ce problème n'a jamais été rencontré via le générateur en ligne de commande.

Lorsque le format de sortie choisi est le SQL, la table créée contient un ID auto-incrémenté. Si nous souhaitons une table SQL contenant exactement 2 colonnes (par exemple ID et Nom\_Ville), il suffit de ne créer qu'un seul champ dans le fichier de paramètres envoyé au générateur. Le fichier de sortie aura bien une extension SQL. Cependant, le script de la création de la table ne fonctionne pas, notre champ n'est pas pris en compte. Vous trouverez ci-dessous une capture d'écran illustrant ce cas.

```
DROP TABLE IF EXISTS villes;
CREATE TABLE villes (
  id mediumint(8) unsigned NOT NULL auto_increment,
  PRIMARY KEY (`id`)
) AUTO_INCREMENT=1;

INSERT INTO villes (Nom_Ville) VALUES ('FRANCARVILLE');
INSERT INTO villes (Nom_Ville) VALUES ('PRADINES');
INSERT INTO villes (Nom_Ville) VALUES ('SAI');
INSERT INTO villes (Nom_Ville) VALUES ('WIMMENAU');
INSERT INTO villes (Nom_Ville) VALUES ('PAIMPOL');
INSERT INTO villes (Nom_Ville) VALUES ('RAMOULU');
INSERT INTO villes (Nom_Ville) VALUES ('BORNAY');
INSERT INTO villes (Nom_Ville) VALUES ('PORTBAIL');
INSERT INTO villes (Nom_Ville) VALUES ('ST');
INSERT INTO villes (Nom_Ville) VALUES ('OYRE');
```

*Figure 2 : capture d'écran d'un fichier SQL généré par le générateur en ligne de commande.*

Comme vous pouvez le constater, notre fichier est généré au format SQL. Notre colonne contenant l'id est bien générée mais nous ne retrouvons pas la colonne Nom\_Ville. Nous nous apercevons qu'en insérant des valeurs dans cette table, que le script devient irrationnel. Nous ne pouvons pas insérer des données dans une colonne Nom\_Ville si elle n'existe pas.

Outre les différentes remarques citées ci-dessus, le générateur semble assez bien fonctionner.

---

## 5.2 Réflexion

Pendant une période d'environ une semaine, nous nous sommes concentrés sur la notion de tables liées. Nous n'avons pas directement écrit le code PHP permettant cet ajout. Dans un premier temps, nous avons pensé à la mise en place de celle-ci sans changer toute la structure déjà existante. Le second point important de cet ajout était la simplicité. Nous étions conscients que notre code serait repris par la suite par notre tuteur et/ou d'autres étudiants. Il fallait donc penser à un système clair et concis. De plus, nous avons pensé à quelques améliorations à implémenter dans le projet qui apporteraient des fonctionnalités supplémentaires :

Aujourd'hui, les formats les plus standards sont le SQL, le XML et le CSV. Cependant, le JSON commence à être un format très répandu, notamment grâce à sa légèreté. En effet, il est probablement plus facile de comprendre un fichier XML grâce à des éléments possédant des nom concrets et bien structurés, mais le JSON est beaucoup moins volumineux en terme d'écriture. Nous avons donc entrepris de proposer la génération dans ce format.

Un générateur de donnée est intéressant pour simuler des exercices concrets relatifs à des examens. En revanche, il perd en intérêt si nous ne travaillons pas avec des données cohérentes. Il est préférable, par exemple, d'avoir des villes avec leurs vrais codes postaux plutôt que des données générées indépendamment. Le générateur prenait déjà en compte ce cas de figure avec un attribut « générerdépendances » mais le défaut est que la génération de colonnes cohérentes se faisait au cas par cas et le générateur proposait seulement la correspondance nom/sexe et ville/code/postal. Nous avons donc voulu généraliser le code afin que celui-ci permette la génération de données cohérentes en général. L'idée était qu'il suffisait de modifier le dictionnaire pour permettre une correspondance exacte. Malheureusement cette fonctionnalité n'a pas pu être finalisée et suffisamment testée dû à un manque de temps. Toutefois nous pensons qu'il s'agirait d'une amélioration significative au projet.



---

## 5.3 Répartition des tâches

Julien, portant un intérêt majeur pour le développement web, sa mission dans ce projet, a été d'ajouter la fonctionnalité des tables liées dans le générateur puis de l'adaptation graphique et fonctionnelle de celle-ci.

La notion de tables liées était une grosse partie du développement, nous avons notamment besoin de connaître les table à lier ainsi que leurs colonnes. De plus, d'un point de vue interface web, il fallait que cette fonctionnalité soit triviale pour l'utilisateur.

Amaury s'est quant à lui orienté vers la notion de données cohérentes avec l'ajout de dictionnaires (fonctionnalité déjà existante) par des traitements de lignes complètes. C'est également Amaury qui a mis en place le nouveau format de sortie (JSON) au sein de ce projet. De plus, il a également réalisé une grande partie des tests pour l'interface web hébergée sur un serveur Apache et sur de nombreux environnements, et ce afin de corriger un maximum de bugs, ainsi que de tester les limites du générateur.

## 5.4 Opérations sur le générateur

Pour la notion de tables liées :

L'intérêt du générateur est de répondre à des problèmes concrets proposés dans des sujets d'examens. Nous nous sommes basés sur un exercice dont vous retrouverez la consigne complète à l'adresse ci-dessous :

[http://forge.info.univ-angers.fr/~gh/internet/sen2016\\_3.pdf](http://forge.info.univ-angers.fr/~gh/internet/sen2016_3.pdf)

En résumé, nous avons besoin de créer trois tables (villes, machines, trajets) où nous supposons qu'une ville est composée d'un id, d'un nom et d'un code postal. Une machine est composée d'un id et d'un type. Un trajet est composé d'un id, d'un train, d'une ville d'arrivée et d'une ville de départ. On comprend tout de suite qu'un trajet est composé des ID des trains et des villes respectives.

Jusqu'à maintenant, le remplissage des tables se faisait de manière aléatoire (ou logique selon des paramètres). Les types de données pour notre exemple sont des valeurs numériques et des chaînes de caractères issues de dictionnaires. Pour la création de la table trajet, il ne fallait pas que les données soient des données générées aléatoirement. Nous n'avions donc pas besoin de préciser à nouveau le type puisque nous l'avions déjà fait pour les trains (machines) et les villes. Pour récupérer la valeur que nous voulions, il suffisait d'aller la chercher dans ce qui était déjà sorti. C'est pour cela que nous avons opté pour la solution de créer un nouveau type de donnée. Ce type sera nommé « Reference ». Le générateur fonctionne avec un fichier de paramètres (au format XML). C'est dans celui-ci que nous précisons le type pour le champ voulu. Vous trouverez ci-dessous une capture d'écran d'un fichier de paramètres non fonctionnel illustrant l'utilisation du nouveau type.

```
<?xml version="1.0" encoding="UTF-8"?>
<Générateur>
  <Table>
    <Champs>
      <Donnee Type="Reference" NomColonne="ColonneParReference" TableReference="TableDeLaReference"
        ColonneReference="ColonneDeLaReference" />
    </Champs>
    <Parametre>
      <Sortie SQL="True"/>
      <Nbligne valeur="10"/>
      <NomTable nom="test_reference"/>
      <Seed valeur="10276666" />
    </Parametre>
  </Table>
</Générateur>
```

Figure 3 : capture d'écran d'un fichier de paramètre non fonctionnel.

L'exemple ci-dessus permet de créer un fichier test\_reference.sql contenant une table « test\_reference », possédant un id généré par défaut ainsi qu'une colonne « ColonnePar-Reference », remplie grâce à des données situées dans une table « TableDeLaReference » et dans la colonne « ColonneDeLaReference ».

Pour que notre fichier de paramètres soit syntaxiquement valide, nous avons ajouté au fichier Paramètre.xsd, notre nouveau type (Reference). Ce type possède les attributs TableReference (correspondant à la table que nous pointons) ainsi que ColonneReference (correspondant à une certaine colonne de cette même table).

C'est pour la partie développement en PHP, que les plus grosses modifications ont été réalisées. Dans un premier temps, nous nous sommes rendu compte que lorsqu'un fichier est produit, les données de ce fichier ne sont pas gardées en mémoire. Si nous souhaitons aller piocher dans celles-ci pour remplir une table contenant une colonne générée par référence, nous avons besoin de garder celles-ci en mémoire. C'est donc la première chose que nous avons mis en place. Lorsqu'un fichier est généré, nous plaçons ces données dans un tableau php avec pour en-tête le nom de la table. Nous décrirons ci-dessous le comportement du générateur lorsque qu'il génère un fichier. Reprenons l'exercice proposé. Nous allons analyser étape par étape comment les villes sont générées.

```
<Table> <!-- VILLES -->
  <Champs>
    <Donnee Type="Numerique" NomColonne="idVille" Null="0" Min="1" Max="500" ModeGeneration="
aleatoire"/>
    <Donnee Type="Dictionnaire" NomColonne="nomVille" NomDictionnaire="villes" ModeGeneration="unique
"/>
  </Champs>
  <Parametre>
    <Sortie SQL="True"/>
    <Nbligne valeur="10"/>
    <NomTable nom="villes"/>
    <Seed valeur="1027666" />
  </Parametre>
</Table>
```

Figure 4 : Extrait du fichier Paramètre pour la création de la table villes.

---

Voici les grandes étapes produites par la générateur :

- 1 - Préparation de la table nommée « villes » via un tableau PHP
- 2 - initialisation de tableaux qui pour chaque champ, se remplit via différentes possibilités (Numerique, Dictionnaire...).
- 3 - Écriture du fichier villes.sql

Pour le remplissage des villes, notre générateur reconnaît le type « Dictionnaire », il fait donc appel à une fonction « genererDico » située dans le fichier genererDico.php qui consiste à ouvrir un dictionnaire et récupérer des valeurs dans celui-ci.

Pour le type Reference, nous avons également une fonction nommée genererReference. Nous avons commenté le code de cette fonction afin que la modification de celui-ci ne soit pas compliqué.

```
<?php
function genererReference($donnees, $nbligne, $TableReference, $ColonneReference){ // remplissage des
valeurs pour les references tables liées

    $array_reference = array(); // On créer le tableau qui contiendra le tableau de référence voulu

    // On parcourt toutes les données, lorsqu'on trouve la table voulue, on la met dans notre variable (
système de pointage)
    for ($i=0; $i < count($donnees); $i++) {
        if($donnees[$i][0] == $TableReference){
            $array_reference = $donnees[$i];
            break;
        }
    }

    array_shift($array_reference); // On ne conserve pas le nom de la table

    for ($i=0; $i < count($array_reference); $i++) {
        array_shift($array_reference[$i]); // On ne s'interesse pas à si les colonnes vont reference à
une table
        if($array_reference[$i][0] == $ColonneReference){
            $array_reference = $array_reference[$i]; // On ne garde que la colonne que l'on veut
            break;
        }
    }

    // À ce stade, $array_reference ne contient que la colonne voulue de la table voulue

    array_shift($array_reference); // On ne garde pas le nom de la colonne

    $reference = array();
    for ($j = 0; $j < $nbligne; $j++) {
        $reference[$j] = $array_reference[(rand(0,count($array_reference) - 1))];
    }

    return $reference;
}

?>
```

Figure 5 : Capture d'écran de la fonction genererReference

---

Cette fonction consiste à récupérer dans les données gardées en mémoire, des valeurs aléatoires de la table et la colonne voulue.

Un second élément dont nous avons besoin, était de savoir si une valeur était générée de manière indépendante ou par référence. Pour cela nous avons ajouté en en-tête (en plus du nom de la table) un attribut à vrai ou faux. S'il est vrai, nous donnons la table de référence ainsi que la colonne de référence. Cette information est importante, notamment pour la création du fichier SQL. Si une donnée est générée par référence, c'est alors une clé étrangère, il faut donc préciser sur quelle table et quelle colonne, elle se réfère.

Pour récupérer cette valeur, nous avons ajouté à toutes les fonctions de générations de fichiers de sorties, le bout de code suivant :

```
$reference = array();
for ($i=0; $i < count($donnees); $i++) {
    array_push($reference, array_shift($donnees[$i]));
}
```

Figure 6 : extrait de code PHP récupérant les valeurs booléennes de référence.

Nous avons également modifié le fichier FoncEcrireSql permettant d'ajouter le système de clé étrangère. Voici le code rajouté :

```
//
if($reference[$i]["reference"] == true){
    $TableReference = $reference[$i]["TableReference"];
    $ColonneReference = $reference[$i]["ColonneReference"];
    $estReference = "references $TableReference($ColonneReference)";
}else{
    $estReference = "";
}
//

if ($temp == 'string') {
    fputs($fp, $donnees[$i][0] . " VARCHAR(255) " . $estReference .",\n");
} else {
    fputs($fp, $donnees[$i][0] . " NUMERIC(9) " . $estReference .",\n");
}
```

Figure 7 : extrait de code PHP ajouté au fichier FoncEcrireSql.php

Pour revenir à notre exercice, voici le fichier de paramètre :

```
<?xml version="1.0" encoding="UTF-8"?>
<Générateur>

  <Table> <!-- VILLES -->
    <Champs>
      <Donnee Type="Numerique" NomColonne="idVille" Null="0" Min="1" Max="500" ModeGeneration="
aleatoire"/>
      <Donnee Type="Dictionnaire" NomColonne="nomVille" NomDictionnaire="villes" ModeGeneration="unique
"/>
    </Champs>
    <Parametre>
      <Sortie SQL="True"/>
      <Nbligne valeur="10"/>
      <NomTable nom="villes"/>
      <Seed valeur="1027666" />
    </Parametre>
  </Table>

  <Table> <!-- TRAINS -->
    <Champs>
      <Donnee Type="Numerique" NomColonne="idTrain" Null="0" Min="1000" Max="10000" ModeGeneration="
aleatoire"/>
      <Donnee Type="Dictionnaire" NomColonne="typeTrain" NomDictionnaire="typestrains" ModeGeneration="
unique"/>
    </Champs>
    <Parametre>
      <Sortie SQL="True"/>
      <Nbligne valeur="10"/>
      <NomTable nom="machines"/>
      <Seed valeur="1027666" />
    </Parametre>
  </Table>

  <Table> <!-- PASSAGES -->
    <Champs>
      <Donnee Type="Numerique" NomColonne="idTrajet" Null="0" Min="0" Max="10" ModeGeneration="
aleatoire"/>
      <Donnee Type="Reference" NomColonne="numTrain" TableReference="machines" ColonneReference="
idTrain" />
      <Donnee Type="Reference" NomColonne="villeDepart" TableReference="villes" ColonneReference="
idVille" />
      <Donnee Type="Reference" NomColonne="villeArrivee" TableReference="villes" ColonneReference="
idVille" />
    </Champs>
    <Parametre>
      <Sortie SQL="True"/>
      <Nbligne valeur="10"/>
      <NomTable nom="trajets"/>
      <Seed valeur="1027666" />
    </Parametre>
  </Table>
</Générateur>
```

Figure 8 : Capture d'écran du fichier Paramètre.xml

et la table de sortie trajets.

```
1 DROP TABLE IF EXISTS trajets;
2 CREATE TABLE trajets (
3   id mediumint(8) unsigned NOT NULL auto_increment,
4   idTrajet NUMERIC(9) ,
5   numTrain NUMERIC(9) references machines(idTrain),
6   villeDepart NUMERIC(9) references villes(idVille),
7   villeArrivee NUMERIC(9) references villes(idVille),
8   PRIMARY KEY (`id`)
9 ) AUTO_INCREMENT=1;
10
11
12
13 INSERT INTO trajets (idTrajet,numTrain,villeDepart,villeArrivee) VALUES (6,4823,277,253);
14 INSERT INTO trajets (idTrajet,numTrain,villeDepart,villeArrivee) VALUES (10,5972,480,277);
15 INSERT INTO trajets (idTrajet,numTrain,villeDepart,villeArrivee) VALUES (4,5972,144,253);
16 INSERT INTO trajets (idTrajet,numTrain,villeDepart,villeArrivee) VALUES (10,9667,131,469);
17 INSERT INTO trajets (idTrajet,numTrain,villeDepart,villeArrivee) VALUES (8,9667,469,469);
18 INSERT INTO trajets (idTrajet,numTrain,villeDepart,villeArrivee) VALUES (0,5972,116,116);
19 INSERT INTO trajets (idTrajet,numTrain,villeDepart,villeArrivee) VALUES (8,9667,144,253);
20 INSERT INTO trajets (idTrajet,numTrain,villeDepart,villeArrivee) VALUES (4,3140,183,201);
21 INSERT INTO trajets (idTrajet,numTrain,villeDepart,villeArrivee) VALUES (7,6175,18,253);
22 INSERT INTO trajets (idTrajet,numTrain,villeDepart,villeArrivee) VALUES (3,4823,144,277);
23
```

Figure 9 : capture d'écran de la table trajets de l'exercice d'examen.

Notre mission « ajouter au générateur la possibilité de générer des tables liées » est réussie. Le système est opérationnel avec cette nouvelle fonctionnalité. Cependant, nous rencontrons différents points sensibles, notamment le fait de devoir créer les tables dans un ordre bien précis. Un second point sensible est de garder en mémoire, les tables déjà créées. Notre générateur est capable de produire des tables possédant un très grand nombre de colonnes et de lignes (environs 100 000). Si nous créons des milliers de tables, nous risquons de dépasser la mémoire de manière considérable qui provoquerait un dysfonctionnement du système.

Les autres point sensibles seront cités à la fin de ce rapport en proposition d'améliorations.

---

## 5.5 Adaptation à l'interface graphique

Nous avons changé quelques propriétés CSS de l'interface web car nous avions des problèmes d'affichage sur des écrans 13 pouces. Des champs du formulaire se superposaient. Il en était de même pour le bouton permettant d'afficher l'aide.

Nous avons ajouté une option à l'élément « select » de notre fichier PHP permettant d'avoir le rendu suivant :

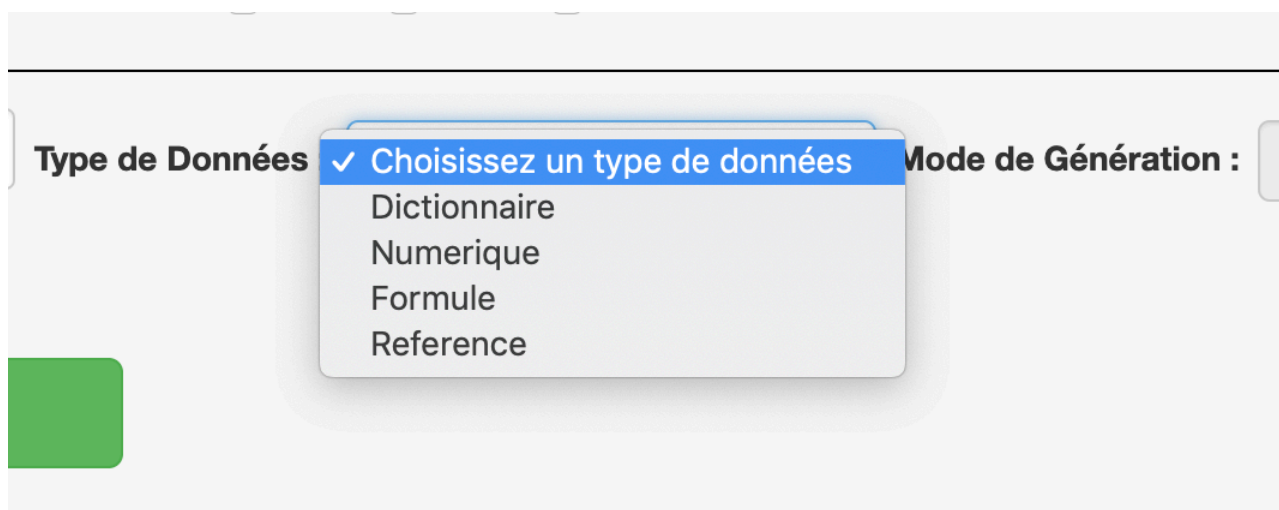


Figure 10 : Capture d'écran du menu déroulant contenant l'option « Reference ».

Grâce à notre méthode « onChange » déjà existante au projet, nous affichons deux champs libres : Table de Référence et Colonne de Reference.

A screenshot of the web form after selecting 'Reference' as the data type. It shows several input fields: 'Nom :', 'Null :', 'Type de Données : Reference' (in a dropdown), 'Mode de Génération : Aléatoire (uniforme)' (in a dropdown), 'Table de Reference :', and 'Colonne de Reference :'. The form has a light gray background.

Figure 11 : Capture d'écran des différents champs pour le type « Reference »



---

## Pourquoi avons-nous choisi des champs textes plutôt qu'une liste déroulante pour la table de référence et la colonne de référence ?

Une liste déroulante serait en parfait accord avec l'intérêt porté par les anciens étudiants, autrement dit « user-friendly ». Cependant, il était plus simple pour nos tests de mettre des champs libres. En effet, pour les listes déroulantes, cela demandait de nombreux changements. Il faut ajouter les options dès lors que nous créons une nouvelle table. L'ordre de création des tables doit être respecté pour éviter des conflits de champs non existants. De plus, il faudrait à chaque changement de l'option de la table référence, afficher les colonnes de cette table.

Nous avons opté pour une solution plus simple et tout aussi efficace. Lorsqu'un champs est mal rempli, celui-ci apparait en fond rouge et le bouton « générer » ne devient plus cliquable.

Lorsque nous ajoutons une nouvelle table, il fallait également penser à ajouter l'option dans l'élément select. Vous trouverez ci-dessous le code des fonctions permettant respectivement d'ajouter et supprimer les champs Table de Référence et Colonne de Reference.

```
function ajoutReference(NumTabl, Numligne){  
  
    var div = document.getElementById("tab" + NumTabl + "Reference" + Numligne);  
  
    var TableReferenceLabel = document.createElement('label');  
    TableReferenceLabel.setAttribute('id', "tab" + NumTabl + "TableReferenceLabel" + Numligne);  
    TableReferenceLabel.innerHTML = "&nbsp;Table de Reference&nbsp;";  
  
    var InputTableReference = document.createElement('input');  
    InputTableReference.setAttribute('class', "form-control");  
    InputTableReference.setAttribute('type', 'text');  
    InputTableReference.setAttribute('id', "tab" + NumTabl + "TableReference" + Numligne);  
    InputTableReference.setAttribute('name', "tab" + NumTabl + "TableReference" + Numligne);  
    InputTableReference.setAttribute('onchange', "ValidationTableReference("+NumTabl+", "+Numligne+")");  
  
    var ColonneReferenceLabel = document.createElement('label');  
    ColonneReferenceLabel.setAttribute('id', "tab" + NumTabl + "ColonneReferenceLabel" + Numligne);  
    ColonneReferenceLabel.innerHTML = "&nbsp;Colonne de Reference&nbsp;";  
  
    var InputColonneReference = document.createElement('input');  
    InputColonneReference.setAttribute('class', "form-control");  
    InputColonneReference.setAttribute('type', 'text');  
    InputColonneReference.setAttribute('id', "tab" + NumTabl + "ColonneReference" + Numligne);  
    InputColonneReference.setAttribute('name', "tab" + NumTabl + "ColonneReference" + Numligne);  
    InputColonneReference.setAttribute('onchange', "ValidationColonneReference("+NumTabl+", "+Numligne+")");  
  
    div.appendChild(TableReferenceLabel);  
    div.appendChild(InputTableReference);  
    div.appendChild(ColonneReferenceLabel);  
    div.appendChild(InputColonneReference);  
}
```

```
function DelReference(NumTabl,NumLigne){
    if(document.getElementById('tab'+NumTabl+'Reference'+NumLigne)){
        var containerReference = document.getElementById('tab'+NumTabl+'Reference'+NumLigne);
        while (containerReference.hasChildNodes()) {
            containerReference.removeChild(containerReference.lastChild);
        }
    }
}
```

Figure 12 : Capture d'écran des fonctions ajouter / supprimer les champs pour le type Reference

Vous trouverez également le code de vérification de ces champs.

```
function ValidationTableReference(NumTabl,NumLigne){
    var tableReference = document.getElementById("tab"+NumTabl+"TableReference"+NumLigne);
    var valeurTableReference = tableReference.value; // On recupere la valeur du input

    // Puis on va comparer cette valeur avec le nom de toutes les tables voir si ça correspond

    var tableFound = -1; // Par défaut on va considérer que la table nommée "xxxxxxx" existe pas

    for(var i = 0; i < TabIndex.length && tableFound == -1; i++){
        var nomTable = document.getElementById("NomTable"+i).value;
        if(valeurTableReference == nomTable && valeurTableReference != ""){
            tableFound = i;
        }
    }

    if(tableFound == -1){
        tableReference.removeAttribute("class");
        tableReference.setAttribute("class","form-control inputInvalid");
        tableReference.setAttribute("title", "Vous tentez de faire référence à une table qui n'existe pas");
        LockGenerer();
    }else{
        tableReference.removeAttribute("class");
        tableReference.removeAttribute("title");
        tableReference.setAttribute("class","form-control");
        tableReference.removeAttribute("title");
        UnLockGenerer();
    }

    return tableFound;
}
```

```

function ValidationColonneReference(NumTabl, NumLigne){
    var colonneReference = document.getElementById("tab"+NumTabl+"ColonneReference"+NumLigne);
    var valeurColonneReference = colonneReference.value; // On récupère la valeur du input
    var colonneFound = false;

    var tableReference = ValidationTableReference(NumTabl, NumLigne);

    if(tableReference > -1){
        for(var i = 0; i < TabIndex[tableReference] + 1 && !colonneFound; i++){
            var nomColonne = document.getElementById("tab"+tableReference+"label"+i).value;
            if(valeurColonneReference == nomColonne && valeurColonneReference != ""){
                colonneFound = true;
            }
        }
    }

    if(!colonneFound){
        colonneReference.removeAttribute("class");
        colonneReference.setAttribute("class", "form-control inputInvalid");
        colonneReference.setAttribute("title", "Vous tentez de faire référence à une colonne introuvable");
        LockGenerer();
    }else{
        colonneReference.removeAttribute("class");
        colonneReference.removeAttribute("title");
        colonneReference.setAttribute("class", "form-control");
        colonneReference.removeAttribute("title");
        UnLockGenerer();
    }
}

```

Figure 13 : Capture d'écran des vérifications de champs

Nous avons également corrigé un soucis de l'interface web. Lorsque nous ajoutons des tables, une variable VarNumTab s'incrémentait mais restait la même si nous supprimons une table. Nous avons donc retiré 1 à notre variable entière, lorsque nous supprimons une table.

---

## 6. Perspective d'améliorations

Le projet de génération automatisée de données universelles est un système très utile et très exploitable. Dans un premier temps, nous parlons de l'existant en citant les différentes améliorations que l'on pourrait apporter. Ensuite, nous citons des exemples possibles à intégrer.

Pour la notion de tables liées : Actuellement, il est possible de générer des tables liées via un fichier de paramètres parfaitement rempli.

Exemple : considérons un fichier de référence produisant une référence vers une table non existante.

Une table vide se génère. Dans le cas du SQL, le fichier n'est pas valide. En revanche, pour ce qui est de l'interface graphique (et c'est sur ce point que nous avons insisté), il n'est pas possible de générer les tables si le formulaire n'est pas correctement rempli.

Il serait également intéressant de passer les champs libres de saisie de la table référence et la colonne référence en liste déroulante pour faire gagner du temps à l'utilisateur et pour éviter une mauvaise saisie. Il ne sera pas nécessaire de supprimer les fonctions que nous avons réalisées. En effet, un utilisateur pourrait tout à fait rajouter manuellement une option erronée à la liste déroulante et donc produire un dysfonctionnement. Lorsque nous mettons un type Reference à un champ, notre fonction `genererReference` ne fait que piocher de manière aléatoire dans les données voulues. Il pourrait être intéressant d'ajouter un peu de logique à la récupération et non plus avoir des données aléatoires. Il pourrait également être intéressant de ne pas prendre en compte les doublons.

Pour le nouveau format de sortie, nous l'avons testé en ligne de commande. Le fichier généré est correct. Il ne manque donc plus qu'à ajouter ce nouveau format à l'interface web.

---

## 7. Conclusion

Un générateur de données universelles est un puissant programme. Il offre de grandes possibilités aussi bien d'un point de vue professionnel qu'éducatif. En effet, en tant qu'alternants, nous nous sommes rendu compte que nous avons souvent besoin de générer des bases de données (non nécessairement SQL) avec de fausses de données permettant de tester nos systèmes. D'un point de vue éducatif, ce projet nous a permis de mettre en oeuvre nos différentes compétences acquises tout au long de cette année scolaire. Ce projet n'étant pas un projet à court terme, nous avons forgé notre façon de penser afin que notre production en terme de programmation soit simple, claire et concise.

Nous avons eu besoin d'un temps relativement long pour prendre en main ce projet, d'une part à cause de notre temps passé en entreprise et du fait que ce projet soit déjà élaboré.

Nous n'avons pas toujours pris le chemin le plus court pour arriver à notre but mais notre tuteur a parfaitement su nous remettre sur le chemin. Nous ne percevons pas ses différentes remarques comme des reproches mais plutôt des conseils. Nous lui en sommes reconnaissants.